Aranda, L.A., Reviriego, P. y Maestro J. A. (2020). Toward a Fault-Tolerant Star Tracker for Small Satellite Applications. *IEEE Transactions on Aerospace and Electronic Systems*, 56 (5), pp. 3421 - 3431.

DOI: https://doi.org/10.1109/TAES.2020.2971289

Towards a Fault-tolerant Star Tracker for Small Satellite Applications

Luis Alberto Aranda, Pedro Reviriego, and Juan Antonio Maestro.

***Abstract*—Star trackers are autonomous, high-accuracy electronic systems used to determine the attitude of a spacecraft. In recent years, Commercial Off-The-Shelf (COTS)-based star trackers are growing in importance for low-cost and short-duration missions, but their fault tolerance against soft errors has not been studied in detail. In this paper, we propose a self-healing system protected with ad-hoc techniques that can be used as the first step to implement a fault tolerant COTS-based star tracker for smallsat applications.**

***Index Terms*—Error detection and correction, fault tolerance, soft error, SRAM-based FPGA, star tracker.**

I. Introduction

STAR trackers are widely used in satellites due to their high-accuracy [1]. This electro-optical device measures the position and intensity of the stars in the captured image, and then calculates the spacecraft attitude by comparing this information to a star catalogue stored in memory [2]. Over the last decades, major progress in terms of size, weight, and power consumption reduction has been achieved in order to enable the use of star trackers in small spacecraft such as picosats or nanosats [3]–[5].

Small satellites are currently used in earth observation, interplanetary missions, and on-orbit operations [6]. In addition, their low cost and small size have also supported the emergence of multisatellite missions such as those based on distributed sensing [7]. Several small satellites can be launched together instead of a bigger and more expensive satellite, so the specific mission aims can be achieved exploiting the flexibility and adaptability of the satellite constellation.

Despite the improvements made in small satellites and star trackers, there are still some issues pending related to fault tolerance against radiation strikes [8]. Small satellites are usually built of current commercial off-the-shelf (COTS) electronic components. This means that they can achieve high performance levels due to COTS technology miniaturization, but also their vulnerability to space radiation is higher [9]. For instance, SRAM-based field programmable gate arrays (FPGAs), which are commonly used as onboard data processing devices, provide high processing speed, low cost, and a practically unlimited number of reconfigurations [10]. However, these FPGAs are sensitive to single event upsets (SEUs), a type of soft error that changes the content of a flip-flop or a memory cell. Therefore, a small satellite which uses a COTS-based star tracker made of an SRAM-based FPGA can experience malfunctions due to space radiation that may lead to data corruption, or even the loss of communication with the satellite [11].

One option to reduce the susceptibility of COTS electronic components to radiation-induced soft errors is to use an expensive manufacturing process to harden the integrated circuit from radiation. However, this alternative does not fit with the low-cost principle of most small satellite missions, so an SRAM-based FPGA star tracker should be protected using a different approach to avoid incurring prohibitive costs. The common and less expensive alternative to the manufacturing approach is based on detecting and/or correcting errors in the circuit through some kind of redundancy, what is known as radiation hardening by design (RHBD).

Several RHBD techniques have been developed over the years to produce devices and systems of enough hardness to resist the space environment [12, 13]. Typically, when the complexity and heterogeneity of the system that is going to be implemented in the COTS component is high, classic protection schemes based on spatial redundancy such as dual or triple modular redundancy (i.e. DMR or TMR) are chosen to shorten development times. However, this usually implies higher FPGA resource usage and power consumption compared to ad-hoc protection techniques based on system knowledge [14].

In this paper, we present a low-cost fault tolerant image processing system implemented in an SRAM-based FPGA that can be used to create a COTS-based star tracker for small spacecraft applications. This system has been designed by applying a "divide-and-conquer" approach combined with ad-hoc protection techniques. The image processing system has been divided into smaller and less complex modules with homogeneous properties that have been protected using the mentioned techniques. These ad-hoc techniques have already been presented separately in our previous works [15]–[17] but are also briefly described in this paper for the sake of completeness. Our techniques exploit specific circuit characteristics to create an integrated error detection/correction feature using fewer FPGA resources than traditional redundancy-based techniques. Therefore, the main contribution of this paper is the evaluation of the reliability of these techniques from a system-level point of view when used to create a self-healing system that can autonomously reconfigure the FPGA to remove the detected errors.

The rest of the paper is organized as follows. Section II presents other protection schemes for star tracker systems. In Section III, the proposed fault tolerant image processing sys-

L. A. Aranda and J. A. Maestro are with ARIES Research Center, Universidad Antonio de Nebrija, Madrid 28040, Spain (e-mail: laranda@nebrija.es; jmaestro@nebrija.es).

P. Reviriego is with Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid, Leganés 28911, Spain (e-mail: previrie@nebrija.es

tem of a star tracker is explained. The evaluation of the system as well as the results obtained are described in Section IV. Finally, the conclusions and future work are summarized in Section V.

II. Radiation-induced Errors in Star Trackers

Low-cost fault tolerant imaging systems are essential in current and future small satellites. These optical systems can be used as a payload for a wide variety of applications such as Earth observation or debris detection, or as an onboard attitude determination and control system such as the star tracker. Star trackers provide accurate attitude information that can be used to facilitate pointing or proximity operations, which are common maneuvers in all satellite missions [18].

As stated in the Introduction, COTS electronic components are vulnerable to radiation-induced soft errors. These radiation effects can become even more significant in a COTS-based star tracker since it is an external spacecraft system directly exposed to the outer space. Consequently, some kind of protection mechanism has to be adopted in order to mitigate these undesirable radiation effects. In the BepiColombo spacecraft for example [19], two star trackers are simultaneously working, while a third one will be used as a back-up in case of discrepancy. Even so, star trackers can be temporarily blinded due to solar events, so auxiliary systems such as gyroscopes are still required [20]. The main problem with the gyroscopes is that they have a drift, so the longer they are used, the more the error will grow. This means that the star tracker can be temporarily disabled (if necessary) since the satellite has back-up attitude determination systems, but its normal operation has to be restored as soon as possible to avoid fatal consequences for the spacecraft as happened in the Hitomi case [11].

The simplest protection technique for the star tracker is the redundancy-based approach mentioned before. However, there are other more efficient alternatives based on characterizing each component independently to determine its vulnerable parts. Once the component is studied, a protection technique can be proposed to address the identified issues. Following this approach, the main components of a star tracker system illustrated in Fig. 1 have been protected over the years using different strategies.

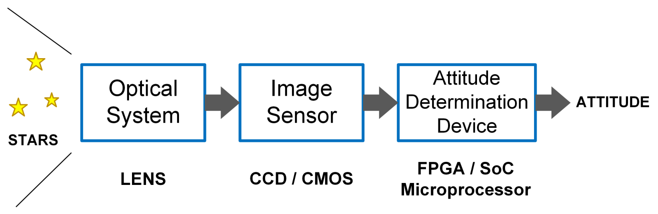

Fig. 1. Component-level scheme of a star tracker.

For example in the optical system, which consists of one or several lenses that focus the incident light from the stars onto the image sensor, the major concern is the total ionizing dose since the lenses are made of glass. Therefore, the glass can be doped with cerium oxide to reduce the radiation-generated impurities that darken the lens [21]. In the case of charge-coupled device (CCD) or complementary metal-oxide-semiconductor (CMOS) image sensors, it is widely known that space radiation can create a broad variety of effects such as hot or dead pixels [22], random telegraph signal noise [23], or dark current increases [24]. To the authors' best knowledge, there is no satisfactory solution to completely solve all these problems in image sensors. In fact, the general approach is to apply a posteriori hardware and/or software mitigation algorithms that remove the undesirable noise acquired by the sensor from the final image. These algorithms are based on image subtraction, image thresholding, or image filtering operations and are implemented in the attitude determination device [25].

The attitude determination device of a star tracker converts the signals read from the image sensor into attitude values. First, the captured image is streamed pixel by pixel, and then it is modified using several image processing algorithms to extract the attitude information from the image. These values are later transmitted to the actuators to correct the orientation of the spacecraft. The mentioned hardware/software mitigation algorithms are integrated within the normal star tracker data flow, so the attitude determination device plays two important roles in the system: it extracts the attitude information from the image captured by the image sensor and, simultaneously, ensures that this information is not affected by the radiation-induced defects in the image sensor enumerated before. This means that soft errors in this device can lead to malfunctions or image data corruption, so the final attitude values determined by the star tracker may not be correct.

In a star tracker, the attitude determination device can be a microprocessor, an FPGA, or a combination of both in a system on chip (SoC) [26]. A microprocessor can be used alone to perform the image processing and star identification algorithms in software, but better results in terms of performance can be achieved if the microprocessor is combined with an FPGA and the image processing operations are implemented in hardware [27]. The FPGA can speed up those operations that are parallelizable such as the image acquisition or the noise filtering procedures mentioned before, while the microprocessor performs complex tasks such as the centroid calculation or the star identification algorithms [28]. As explained in the Introduction, SRAM-based FPGAs are commonly used as onboard data processing devices in satellites due to their high densities, processing speed, and low cost. Therefore, this FPGA technology can also be used to perform the image processing part of a star tracker. The main problem of using SRAM-based FPGAs in space applications is that they are mostly made of SRAM cells, so both configuration memory and user memory errors may occur due to SEUs. The type of error induced depends on the component affected by the incident radiation. If the energetic particles affect the FPGA configuration memory elements, then the design functionality may change. These errors are sometimes called persistent since the design malfunctions persist indefinitely after the upset, but they can be removed by reloading the original bitstream or power cycling the device. On the other hand, user memory errors happen when the content of user-logic memory elements such as flip-flops or block RAMs (BRAMs) is affected by the energetic particles. Therefore, user memory errors corrupt the stored pixels processed by the design, while configuration memory errors modify the behavior of the design itself.

In order to deal with configuration memory and user memory errors in SRAM-based FPGAs, several protection techniques can be implemented. In [29] for example, an error-correcting code (ECC) with bit-interleaving was adopted to protect the star tracker registers, but there are other standard RHBD techniques such as TMR and voting, or cyclic redundancy check that can also be applied to star trackers [30]. However, these techniques have a cost since they are not optimized for each particular star tracker algorithm. Their use can increase the resource overhead, the delay, or the overall power consumption of the star tracker, so the added penalties have to be carefully considered to select the best protection for each case. In this paper, we propose an image processing system for a COTS-based star tracker that has been protected using ad-hoc techniques. These techniques have been presented in our previous works [15]–[17] and are based on knowledge of the system, so they are designed to obtain the right balance between the FPGA resources used and the final error detection/correction rate achieved. In this paper, these ad-hoc techniques are evaluated from the system-level point of view by creating the proposed self-healing system described below. The evaluation of the entire system is presented in Section IV.

III. Proposed Protected System

As stated before, the attitude determination device of a star tracker can be a microprocessor, an FPGA, or a SoC. In this paper, the proposed protected image processing system has been implemented in the SRAM-based FPGA part of a Xilinx Zynq-7000 SoC device. A SoC device has been chosen to enable the system to be further extended with future centroid calculation and star matching algorithms implemented in the microprocessor part. However, the presence of a microprocessor in the same device has been additionally exploited to create a robust fault tolerant system.

The main idea behind using a SoC device is to inform the microprocessor of the radiation-induced errors that occur in the configuration memory of the SRAM-based FPGA part to enable a self-healing behavior. In order to do that, the image processing system implemented in the FPGA has been protected using ad-hoc error detection techniques whose “error detected” signals have been connected to the microprocessor. This way, once a configuration memory corruption is detected in the FPGA by the protection techniques, a software interruption programmed in the microprocessor is automatically triggered to perform the reconfiguration of the FPGA and thus remove the detected error.

As previously stated, the SRAM-based FPGA part of the proposed system has been protected against radiation-induced SEUs using ad-hoc protection techniques. These techniques have been developed by dividing the image processing system into functional modules to study and protect them more easily. Following this “divide-and-conquer” approach, the overall system complexity is decreased to facilitate the evaluation of each module. The “divide-and-conquer” strategy applied consists of the next steps:

1. Divide the image processing system into smaller and less complex functional modules.
2. Study the structure and behavior of each functional module individually to find its main properties.
3. Develop an ad-hoc protection technique for each module exploiting the homogeneous properties found in the previous point.
4. Evaluate the ad-hoc technique in terms of error detection and FPGA resource usage.
5. Iterate the previous steps to refine the FPGA resource usage vs. fault tolerance trade-off of the solution found.

Using this “divide-and-conquer” approach, the proposed protected star tracker system illustrated in Fig. 2 has been implemented in the Zynq-7000 SoC device, which combines an SRAM-based FPGA and a microprocessor.

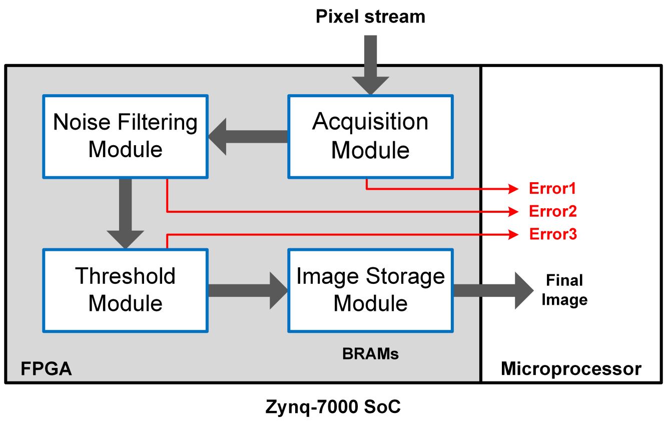

Fig. 2. Overview of the protected image processing system.

In this figure, the image processing system in the FPGA receives an 8-bit grayscale image captured by the image sensor in pixel-stream format and the following operations are performed:

- **Acquisition Module:** the acquisition module prepares the input pixels for the noise filtering process by buffering them using a first in, first out (FIFO) pipeline.
- **Noise Filtering Module:** the buffered pixels are filtered in groups to remove the noise captured by the sensor. In particular, a median filter has been chosen to remove impulsive noise.
- **Threshold Module:** after the noise filtering operation, the image is thresholded to separate the pixels that belong to stars from those that are part of the background. This module sets to zero those pixels that are not required for the centroid calculation or the star matching algorithms (i.e. the background pixels).
- **Image Storage Module:** the thresholded pixels are stored in the FPGA BRAMs to share the final processed image with the microprocessor.

All these FPGA modules have been protected against SEUs using ad-hoc techniques except for the threshold module, which has been protected using a standard technique (as will be explained later). As can be observed in Fig. 2, the error signals of the acquisition, noise filtering, and threshold modules have been connected to the microprocessor. These three module-generated error signals are used to trigger an FPGA reconfiguration using the microprocessor software interruption mentioned before. Therefore, the proposed system behavior can be defined by two states:

- **Operation in the absence of errors:** in an error-free environment, the error signals *Error1*, *Error2*, and *Error3* are equal to zero and the system behaves normally. The sky images captured by the star tracker are processed by the FPGA and sent to the microprocessor for subsequent star identification algorithms.
- **Operation in the presence of errors:** when a configuration memory error is detected by the FPGA modules (*Error1* or *Error2* or *Error3* = 1), the captured image is discarded and a reconfiguration of the FPGA is performed by the microprocessor to remove the persistent error. The reconfiguration clears and rewrites the content of the FPGA configuration memory returning the system to the correct behavior.

Apart from that, it is worth noting that the image storage module depicted in Fig. 2 does not have any "error detected" signal. This is because this module mainly consists of BRAMs, which are vulnerable to user memory errors that corrupt the stored image instead of configuration memory errors that modify the design structure. For this reason, and because the proposed protection technique for this module can detect and correct user memory errors on-the-fly (as will be explained later), there is no "error detected" signal connected to the microprocessor. Some examples for each of the previously listed modules along with their protection schemes are detailed in the following subsections. It should be mentioned that the algorithms within these modules could be replaced by other algorithms that perform similar tasks. However, other ad-hoc protection techniques would have to be developed to achieve the desired resource usage vs. fault tolerance trade-off.

A. Protection technique for the acquisition module

This protection technique has been individually evaluated in detail in our paper [15]. As mentioned before, each 8-bit grayscale image captured by the image sensor is sent in pixel-stream format. This means that the image pixels are received one at a time column by column and row by row every clock cycle. Point operations such as brightness adjustment or image thresholding, in which the output pixel is only a function of the corresponding input pixel value, can be easily integrated into this pixel stream flow. However, certain local operations such as median or rank filters, in which the output pixel is a function of the input pixel and its surrounding neighbors, need to process the image locally instead of pixel by pixel (as illustrated in Fig. 3). Hence, a pixel caching mechanism is required.

In the previous figure, gray-shaded pixels on the input image represent a 3x3 square window that feeds the local filter. The resulting filtered pixel replaces the center pixel of the window. In order to generate the output filtered image, the window must be moved along the entire image to process each input pixel. In software, this algorithm can be easily implemented using loops but, in hardware, the movement of the square window

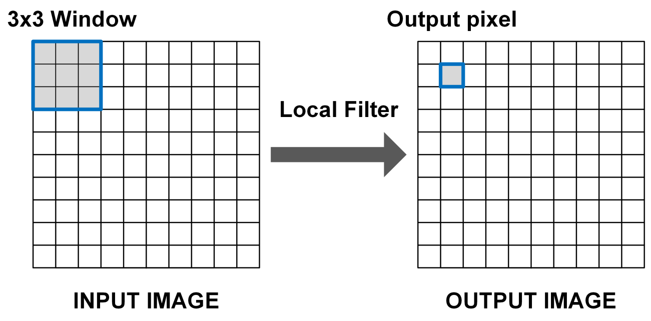

Fig. 3. Local filtering process.

should be done using a special structure made of FIFO buffers and registers [31]. This structure is illustrated in Fig. 4.

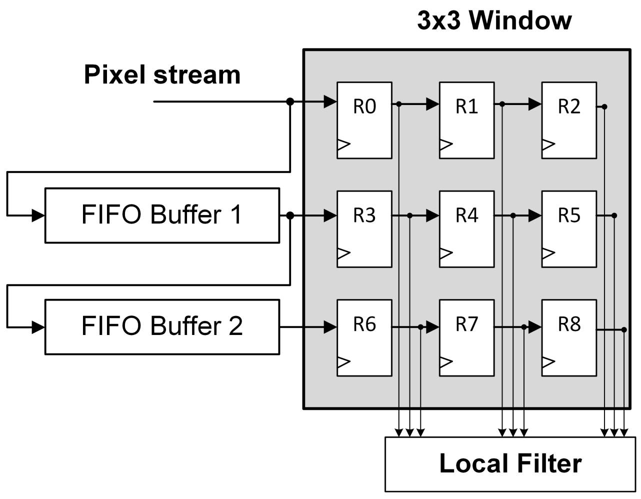

Fig. 4. Line buffer pipeline structure [31].

The structure shown in the previous figure consists of nine registers and two FIFO buffers that temporarily store two rows of the image and the first three pixels of the third row to "simulate" the 3x3 window movement along the image mentioned before. Once the entire pipeline is filled with pixels, the nine pixels of the window are outputted simultaneously every clock cycle to the local filter module. Then, the desired local filtering operation can be performed.

Studying the line buffer pipeline in detail, it can be observed that it is a long delay line with two parallel branches (from $R0$ to $R2$ and from $R3$ to $R5$). Therefore, if there is no malfunction, any pixel fed to the pipeline should be outputted through $R2$, $R5$ and $R8$ but in different time periods. Following this structural property of the design, a malfunction in the pipeline can be detected by temporarily storing the pixels outputted through $R2$ and $R5$ and comparing them against the pixel outputted through $R8$ at this precise moment. In order to create this ad-hoc decimated comparison, a couple of 8-bit detection registers, a counter, and a three-input comparator have to be added to the original structure. Fig. 5 illustrates the proposed protection technique in which the *Error1* signal is connected to the microprocessor as presented before in Fig. 2.

It should be noticed that the detection registers $DR1$ and

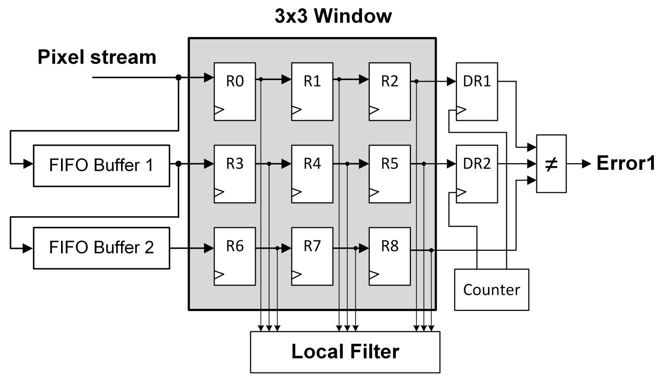

Fig. 5. Protected acquisition module [15].

$DR2$ depicted in Fig. 5 are different from the registers within the 3x3 window. The detection registers are also 8-bit registers but controlled by a counter module, which enables their reading/writing operations, instead of by the global clock signal of the system. The counter is required to perform the comparison of the content of $DR1$, $DR2$, and $R8$ at particular time periods. For example, a pixel that enters in the pipeline at time t is stored both in $R0$ and *FIFO Buffer 1*, then, at time $(t+3)$ the pixel is outputted through $R2$ and the counter enables the writing of $DR1$ to store it. The same pixel will be stored in $DR2$ at ($t+3+$*line buffer length*) and, finally, the pixel will also be outputted through $R8$ ($t+3+2\cdot$*line buffer length*) clock cycles later. The comparison of the content of $DR1$, $DR2$, and $R8$ has to be performed at this precise moment to compare the same pixel value. If the pixel value changes in some parts of the pipeline due to a configuration memory error, a comparison mismatch will occur and *Error1* signal will be triggered. This decimated comparison approach is based on the fact that configuration memory errors modify permanently the structure of the design so, once the pipeline is altered, most of the subsequent pixel values that pass through this point will be corrupted.

B. Protection technique for the noise filtering module

This protection technique has been individually evaluated in detail in our paper [16]. In image processing, the noise filtering operation is usually the next step to image acquisition. This order is followed to avoid propagating the undesirable noise captured by the image sensor to other modules. This input noise can be Gaussian, periodic, or impulsive depending on the source. Impulsive noise is harmful to star trackers since it can create new false stars in the image or modify the centroid/intensity of the stars already in the image. For this reason, we have included an impulsive noise filtering module in the image processing system. In particular, a median filter has been selected due to its robustness and its capabilities to remove the noise while preserving the edges of the stars in the image.

The median filtering operation consists in sorting the pixels outputted by the acquisition module to obtain the median value. Using the median, the pixel in the original image that corresponds to the center of the current 3x3 window position (see Fig. 3) is replaced to remove any impulsive noise in this pixel. In order to filter the entire image, the median filtering operation has to be performed for each window position along the image. In hardware, the median filter can be implemented using an exchange network scheme as the one shown in Fig. 6.

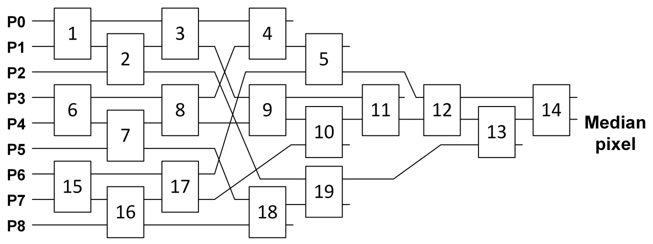

Fig. 6. Hardware implementation of a 3x3 median filter [32].

As mentioned before, this scheme receives the nine pixels from the acquisition module and determines their median value. Each box numbered from 1 to 19 is an identical exchange node that performs a two-input sorting using an 8-bit comparator and two 2:1 multiplexers. The two inputs are internally compared and the higher (H) and the lower (L) values are obtained as illustrated in Fig. 7. The lower output of the node number 14 is the median of the inputs.

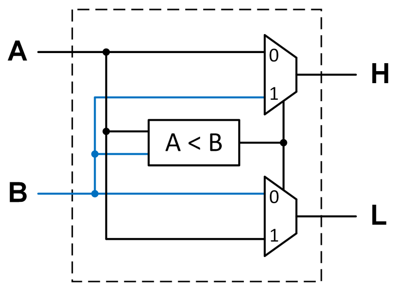

Fig. 7. Internal diagram of one exchange node [16].

Again, as was done for the previous module, the median filter structure has been studied to find a specific property that can be exploited to create a protection technique. Looking at Fig. 6, it can be observed that there are unused outputs in some exchange nodes. These outputs are, by definition, higher or lower than the median value, so they can be used to create a dynamic range. If the median is out of this range, then a malfunction has occurred. Consequently, this dynamic range can be used to activate an error signal (*Error2* from Fig. 2) when a configuration memory error is detected. Then, a reconfiguration of the FPGA can be performed to remove the error. Fig. 8 illustrates our protection technique presented in [16].

This technique is based on adding the gray-shaded blocks shown in Fig. 8 to the original median filter scheme. These blocks are used to create the mentioned range with the non-median outputs. The range is dynamically determined for each nine pixel group using identical exchange nodes as the one shown in Fig. 7. The upper value of the range ($H3$ low output) is calculated as the lower value of the four higher input values, while the lower value of the range ($L3$ high output) is obtained using the higher value of the four lower input pixel values.

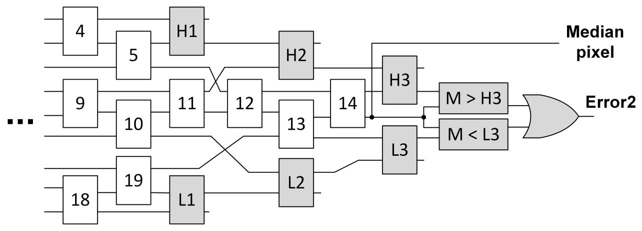

Fig. 8. Protected noise filtering module [16].

Once the range is calculated, two 8-bit comparators check if the median is within the range. If the median value is above $H3$ or below $L3$, then *Error2* signal is triggered.

C. Protection technique for the threshold module

After the noise filtering operation, the threshold module performs a background subtraction to facilitate the star centroid determination algorithm. Basically, the threshold module in a star tracker replaces those pixels that are below a fixed value (the threshold) by a black pixel, and leaves unmodified the rest of the pixels. In this way, the intensity of the star pixels is not altered and can be used later in the star matching process.

The threshold module can be implemented in hardware using combinational logic, so it can be directly connected in series to the pixel stream to process each pixel as it arrives from the previous filtering module. The FPGA implementation of the threshold module is depicted in Fig. 9.

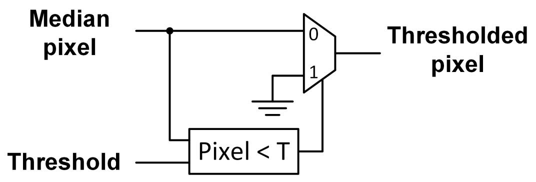

Fig. 9. Internal diagram of the threshold module.

The input pixel is compared to the user-defined constant threshold value. The output signal of this comparison is connected to the 2:1 multiplexer control signal in the figure. This means that, if the pixel value is lower than the threshold value, the ground input will be selected, but if the pixel value is above the threshold, then the input pixel is outputted normally. Looking at the previous figure, it can be observed that the threshold circuit consists of a couple of components so, in this case, a custom protection technique has not been developed for this module. In fact, a classic DMR scheme has been directly used to protect it as illustrated in Fig. 10.

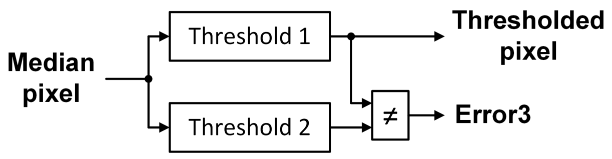

Fig. 10. Protected threshold module.

The approach presented in this figure has been followed since the increment in terms of FPGA resource usage will be negligible for this small design. Finally, it is worth noting that the *Error3* signal is connected to the microprocessor in Fig. 2.

D. Protection technique for the image storage module

This protection technique has been individually evaluated in detail in our paper [17]. After the previous image processing steps, the final image will be almost noise-free and entirely black, except for small star regions that are above the threshold (see Fig. 11). Now, the final image can be stored in an internal or external memory to share it with the microprocessor that will perform the centroid calculation and the star matching algorithms. In our case, we have chosen internal Xilinx BRAMs since they are embedded memories with low access latency and, moreover, the number of BRAMs in the FPGA part of the Zynq-7000 SoC is enough to store a 640x480 image, which is the standard VGA resolution. However, the main drawback of Xilinx BRAMs is that they are highly sensitive to radiation-induced user memory errors due to their high density in the newer 7-series technologies, so the stored pixels can be corrupted by SEUs.

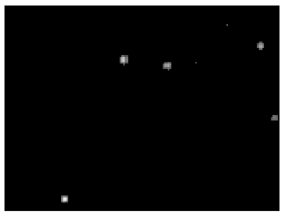

Fig. 11. Example of final preprocessed star tracker image.

Analyzing the properties of the final star images that will be stored in the BRAMs and according to [33], it can be concluded that a typical 640x480 image will usually contain less than 500 star pixels since the number of stars per image is usually between 4 to 6 and each star is spread over a small region of 9x9 pixels. This means that the percentage of pixels in the image that belong to stars will be less than 0.2%. Our protection technique exploits this particular feature of the stored data to detect and correct transient errors in the stored pixels. In particular, the correction of the corrupted pixels in the proposed scheme is achieved thanks to the use of a back-up memory which contains a copy of the star pixels. The proposed protection scheme for the image storage module is illustrated in Fig. 12.

In addition to the back-up memory mentioned before, an encoding/decoding operation has to be performed to be able to distinguish between star and background pixels after the effect of an SEU. For a better understanding of the scheme presented in Fig. 12, the encoding and decoding procedures are explained separately.

- **Encoding operation:** First, each median pixel from the filtering module is classified as star or background using the threshold module. If the pixel is below the threshold, it is considered background and set to zero, and in the encoding stage its parity is also set to zero. Conversely, if the pixel is above the threshold, it is considered part of

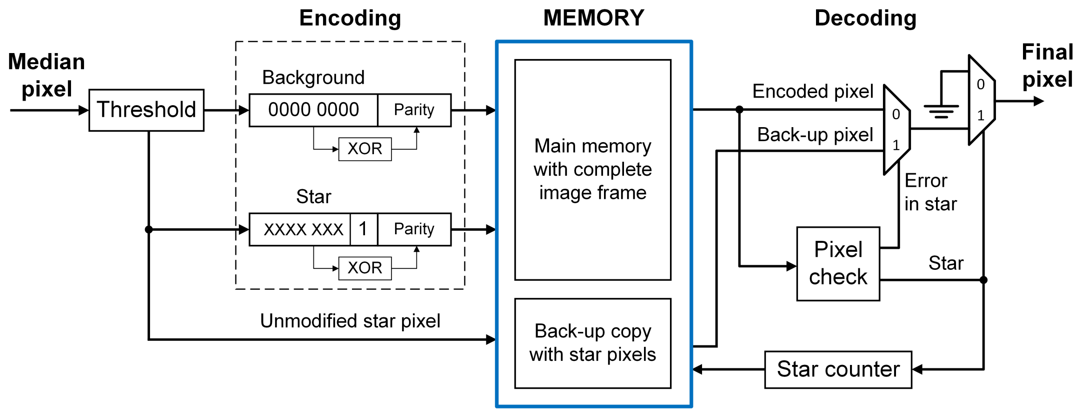

Fig. 12. Protected image storage module [17].

a star and, in the proposed technique, the pixel is duplicated. One copy of the star pixel is stored unmodified in the back-up memory and, at the same time, the other copy is encoded and stored in the main memory. The star encoding procedure replaces the Least Significant Bit (LSB) of the pixel by '1', leaving the other seven bits unaltered. Then, the parity of the star pixel is calculated using these seven bits to create a minimum Hamming distance of three between a star and a background pixel. This way, the capability to differentiate between a corrupted star pixel and a corrupted background pixel during the read operation is enabled. Finally, the encoded star pixel and its parity are stored in the main memory.

- **Decoding operation:** the decoding operation consists of two multiplexers, a star counter, and a "pixel check" block. The "pixel check" block is used to determine whether the read pixel is background or star. If the read pixel is background, the "Star" signal is zero and then the output pixel is set to zero as well, removing every possible SEU in the background pixel. Otherwise, the pixel is star, and then the "Star" signal is set to one. In this case, the parity of the seven MSBs of the pixel is checked. If the parity check is correct, the "Error in star" signal is set to zero and the pixel is outputted normally but, if an error is detected during the parity check, then this signal is set to one and an uncorrupted copy of the pixel is retrieved from the backup memory. In order to determine the current back-up memory address, the "Star" signal is also used to increment the address of the star counter block. Finally, the internal scheme of the "pixel check" block is depicted in Fig. 13 for the sake of completeness, but a detailed description and evaluation of the protection technique explained in this subsection can be found in our previous work [17].

After presenting each protected module, the image processing system of our SRAM-based FPGA star tracker has been evaluated in the next section. The behavior of the ad-hoc techniques has also been analyzed to validate them from the system-level point of view.

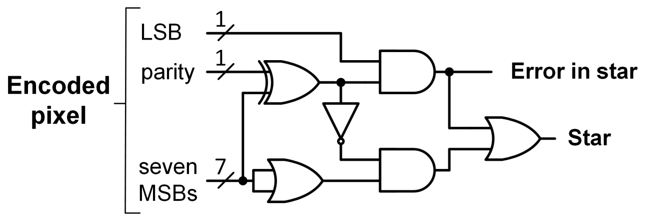

Fig. 13. Internal scheme of the "pixel check" block [17].

IV. System Evaluation

As mentioned at the beginning of the previous section, a "divide-and-conquer" strategy has been used to find modules with homogeneous properties in the image processing system of the star tracker. These modules have been studied and protected individually against configuration memory or user memory errors with the ad-hoc techniques explained before. The main purpose of this module-level division is to reduce the overall complexity of the whole star tracker system to apply ad-hoc techniques more easily. In this manner, the fault tolerance of the system can be improved by increasing the fault tolerance of its individual modules.

In this section, the behavior of the complete system presented in Fig. 2 has been analyzed to evaluate the correctness of the "divide-and-conquer" approach and the protection techniques from the system-level point of view. In order to do that, configuration memory errors have been exhaustively injected using the Xilinx Soft Error Mitigation (SEM) Intellectual Property (IP) controller [34]. User memory errors in BRAMs have not been injected since the image storage module, which is the only module that uses BRAMs in the system, has already been tested in our work [17], and the results show that the technique is able to detect and correct these errors in the stored pixels on-the-fly. Therefore, only configuration memory errors have been injected to test the complete system.

The results obtained for the proposed protected system have been put into perspective by comparing them against the results obtained by a system protected with a traditional redundancy-based technique. A redundancy-based system has been chosen as the closest neighbor because, as mentioned in the Introduction, these schemes are typically followed to shorten development times when the complexity and heterogeneity of the system that is going to be implemented in the COTS device is high. In particular, a system protected with DMR schemes for each module has been chosen. DMR has been chosen as protection technique instead of, for example, TMR, because DMR can be used to create a system with a similar error detection behavior to the proposed system. The total FPGA resource usage, the fault tolerance, and the reconfiguration rate of both ad-hoc and DMR-based systems are presented in the following subsections.

A. Resource usage

Both proposed and DMR-based systems have been implemented in the SRAM-based FPGA part of a Xilinx Zynq-7000 SoC together with an unprotected version of the star tracker image processing system to measure the FPGA resource overhead. The number of look-up tables (LUTs) and flip-flops (FFs) is summarized in Table I as well as the percentage of overhead added to the unprotected star tracker.

TABLE I
FPGA RESOURCE OVERHEAD COMPARISON

	Unprotected	DMR	Proposed
LUTs	456	957 (109.9%)	578 (26.8%)
FFs	127	260 (104.7%)	181 (42.5%)
BRAMs	75	76 (1.3%)	76 (1.3%)

It can be noticed that the DMR-based system requires more than 100% of LUTs and FFs, as expected. They are not exactly 100% because the system also requires comparators to check the outputs of the duplicated modules. Conversely, our proposed protected system adds a low overhead to the unprotected system. This is because the ad-hoc protection techniques have been designed to achieve a high error detection rate at the minimum resource usage cost. Finally, it can also be observed that the increase in BRAMs is low and equal in both cases. This is because the proposed protection technique for the image storage module has been applied to both systems since a DMR of this module does not fit in the selected Zynq-7000 SoC.

B. Fault tolerance

In this subsection, the behavior of the proposed protected system and the DMR-based system in terms of error detection capabilities has been evaluated with the SEM controller. The SEM controller is a Xilinx IP block that can be used to inject configuration memory errors in the design. Loading the SEM controller in the FPGA along with the system under test and using a serial port communication script, an automated fault injection campaign can be executed. Before starting the fault injection campaign, a golden simulation has to be performed to verify the proper operation of the system under test. For the experiments, 8-bit grayscale 640x480 star tracker images have been generated using a Matlab program created by E. Palombo from the European Space Research and Technology Centre (ESTEC) in the Netherlands. In the golden simulation, these input images are processed by the system in the absence of errors, obtaining the golden output images that are later used to assess the correctness of the final images obtained during the fault injection campaign. This campaign consists of a fault injection/correction loop, in which one configuration memory bit is flipped in each iteration. After comparing the new final image to the golden final image, the erroneous configuration memory bit is flipped again to correct the error and a new injection/correction loop iteration is performed until all the configuration bits are tested. Therefore, it can be considered an "exhaustive" fault injection campaign in which the single-error model of the system under test is checked.

Based on the effect that the bit flip produces in the image outputted by the system, a classification has been made as follows:

- **Corrupted image**: the current image outputted by the system is considered as "corrupted" when it is different from the golden image obtained in the absence of errors. The effects of the injected error can be observed and measured in the corrupted image. Two cases can be distinguished:
  1. Detected error: an error is considered as "detected" when (at least) one "error detected" signal of the system (*Error1*, *Error2*, or *Error3*) is high and the output of the design is different from the golden output. This means that a malfunction in the system has been detected by a protection technique. In this case, this will imply discarding the corrupted image and reconfiguring the FPGA to assure the reliability of the system.
  2. Undetected error: this is the opposite case. The injected bit flip produces a malfunction that modifies the outputted image but the protection technique does not detect the error (all the "error detected" signals are low).
- **Uncorrupted image**: in this case the current image outputted by the system matches the golden image. This means that the injected error has been masked and does not affect the final image. Two cases can be distinguished:
  3. False positive: at least one "error detected" signal is high, but the final image matches the golden image. False positives normally happen when the bit flip affects a protection technique itself, triggering the "error detected" signal. The behavior of the system in this case will be the same as in 1., discarding the image and reconfiguring the FPGA to recover the error detection capabilities.
  4. Normal operation: in this case the bit flip does not affect the proper operation of the system and the protection techniques does not detect any error. This case can happen when the injected error affects an SRAM cell that is unused by the design. The bit flip will have no impact in the system performance.

Using this classification, a reliability report can be generated to compare both systems between them. In order to perform the exhaustive fault injection campaigns, a fixed FPGA region has been selected to test each design individually. This region contains 255,517 bits, so 255,517 injections have been performed to cover the entire region. The following results summarized in Table II have been obtained after performing the exhaustive fault injection campaign procedure explained before.

TABLE II
ERROR DETECTION COMPARISON

	DMR	Proposed
Corrupted images	25,518	18,555
1. Detected errors	23,119 (90.6%)	15,743 (84.8%)
2. Undetected errors	2,399 (9.4%)	2,812 (15.2%)
Uncorrupted images	229,999	236,962
3. False positives	17,765 (7.7%)	1,735 (0.7%)
4. Normal operations	212,234 (92.3%)	235,227 (99.3%)
MSE (Undetected)	73	76

Analyzing the results in Table II, it can be observed that the percentage of detected errors in the DMR-based system is higher than the proposed protected system. However, it can also be noticed that this percentage is not 100% as should be expected. This phenomenon is due to configuration memory errors that affect the input/output routing connections of the modules by changing the input/output pixel values. These errors are not detected by the protection techniques since they occur before or after the module, so the “error detected” signal is not triggered. However, the comparison against the golden image mismatches, so the error is classified as undetected. These input/output routing errors have been included in the results because they are common in FPGAs as they are part of the implemented design. It can be observed that the percentage of these undetected errors for the proposed system is higher than the DMR-based system but, in absolute numbers, this value is close in both systems.

In order to evaluate the effect of the undetected errors in the final image, two approaches have been followed. A quantitative approach, in which the averaged mean square error (MSE) of the corrupted pixels in the images, which can be used to estimate the changes in the pixel values between the golden uncorrupted image and the corrupted images, is calculated. And a qualitative approach, in which some images were outputted for a visual inspection. About the quantitative approach, it can be observed in Table II that the MSE values are similar in both designs. This indicates that the pixel values in the images with undetected errors are close to their original uncorrupted values, so both protected systems behave in the same way. Consequently, the undetected corrupted image should not heavily impact the later star identification algorithms. Following the qualitative approach, both images with detected and undetected errors were analyzed to draw some conclusions. An example of these images is shown in Fig. 14.

Fig. 14 (a) shows a corrupted image that is undetected by the proposed protected system and Fig. 14 (b) a corrupted image that is detected by the protected system. First, it can be noticed

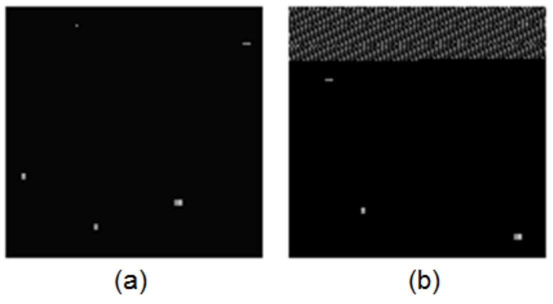

Fig. 14. Example of corrupted final images. (a) Corrupted image undetected by the system and (b) corrupted image detected by the system.

that the undetected image is quite similar to the one shown in Fig. 11. This is because, in this case, the error is slightly modifying the values of the pixels in the image, but not enough to observe a visual impact. This can be due to the limitation of the proposed protection technique for the noise filtering module, which cannot detect errors when the median value is within the range. About the corrupted image that is detected by the protection techniques (Fig. 14 (b)), it can be observed that the upper part of the image is damaged by only injecting one configuration memory error. This is because the image processing system is a long pipeline so, if a configuration memory error creates a malfunction in the structure, most of the pixels that pass through this damaged part will be corrupted. This corrupted image, which is detected by our design, would have significantly more impact in the centroid calculation than Fig. 14 (a). Therefore, from the qualitative and the quantitative points of view, it can be said that the fault tolerance of the system has been improved.

Finally, it can also be observed that fewer errors occur in the proposed protected system. This is because fewer FPGA resources were used during the design process. This is particularly noticeable in terms of false positives. In this case, our system has 10 times fewer false positive detections than the DMR-based system since it requires fewer FPGA resources to perform the error detection. Consequently, fewer reconfigurations will be performed in our protected system as will be shown in the next subsection.

C. *Reconfiguration rate*

As has been explained during this paper, the “error detected” signals (*Error1*, *Error2*, and *Error3*) have been connected to the microprocessor part of the SoC to trigger the complete reconfiguration of the FPGA and thus remove the detected configuration memory error. Analyzing the behavior of the system presented in the previous subsection, it can be deduced that the FPGA will not only be reconfigured when an error is detected by the protection technique, but also when a false positive detection occurs due to errors in the detection part of the design. Therefore, the total number of reconfigurations can be obtained for each protected system by adding the number of detected errors and false positives from Table II. Then, the number of reconfigurations for the DMR-based system is 40,884 and 17,478 for the proposed system.

By comparing these two values, it can be determined that the DMR-based system is reconfigured 2.3 times more than the proposed protected system. This higher number of reconfigurations means that the DMR-based system will be available less time. This is because the ad-hoc protection techniques have been designed having the resource overhead minimization in mind, which directly impacts in the reconfiguration rate.

In conclusion, after the evaluation performed in this section, it can be said that the "divide-and-conquer" strategy followed to analyze and protect the image processing system has provided adequate results. The use of ad-hoc protection techniques has implied a slightly higher number of undetected errors. However, the corrupted images due to these errors have a low MSE value, so their corrupted pixels are close to their uncorrupted values. Moreover, fewer FPGA resources were used during the design of the ad-hoc protection techniques, so fewer errors may occur. At the same time, this small resource overhead has contributed to reduce the number of false positive detection and thus the reconfiguration rate of the system.

V. Conclusions and Future Work

The use of COTS components in space applications implies a reduction in the overall cost of the satellite. However, it also results in a reduction of the system reliability against radiation-induced errors. Typically, when the complexity of the system is high, traditional protection approaches based on modular redundancy are usually implemented to shorten development times. In order to facilitate the development of custom techniques, a "divide-and-conquer" strategy has been followed in this paper to reduce the complexity of the entire system. Using this approach and the knowledge of the system, several ad-hoc techniques have been developed to protect the image processing system of a COTS-based star tracker. In this paper, the evaluation of these ad-hoc techniques from the system-level point of view is presented.

Experimental results have demonstrated a satisfactory behavior of the system in terms of FPGA resource usage, error detection rate, and reconfiguration rate since the individual techniques were developed to obtain a good balance between the FPGA resource usage and the final fault tolerance achieved. Based on the results, it can be said that the "divide-and-conquer" strategy combined with ad-hoc techniques can provide us with good results by reducing the complexity of the system and thus the development time. Following this approach, a self-healing system that automatically reconfigures the FPGA when an error is detected has been proposed. This system can be used as the first step to implement a low-cost fault tolerant star tracker based on COTS components for small spacecraft applications.

As a natural extension of our system, we will implement the centroid calculation and star matching algorithms in the microprocessor part in future works. In addition, some software protection techniques will be studied to protect these algorithms.

Acknowledgement

The authors would like to thank E. Palombo from ESA/ESTEC in Noordwijk, the Netherlands, for providing the star tracker images used in this paper.

References

[1] G. J. Zhang, *Star Identification. Methods, Techniques and Algorithms*, National Defense Industry Press, Beijing, 2017.

[2] J. Roshanian, S. Yazdani, and M. Ebrahimi, "Star identification based on euclidean distance transform, voronoi tessellation, and k-nearest neighbor classification," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 52, no. 6, pp. 2940-2949, 2016.

[3] R. Zenick and T. J. McGuire, "Lightweight, Low-Power Coarse Star Tracker," in *17th Ann. AIAA/USU Conf. on Small Satellites*, 2003.

[4] A. Joachim, "Star Imager for Nanosatellite Applications," M.S. thesis, Dept. Electron. Eng., York Univ., Toronto, ON, 2017.

[5] S. A. Rawashdeh and J. E. Lumpp, "Image-based attitude propagation for small satellites using RANSAC," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 50, no. 3, pp. 1864-1875, 2014.

[6] A. Nanjangud, P. C. Blacker, S. Bandyopadhyay, and Y. Gao, "Robotics and AI-Enabled On-Orbit Operations With Future Generation of Small Satellites," *Proc. IEEE*, vol. 106, no. 3, pp. 429-439, 2018.

[7] S. Bandyopadhyay, R. Foust, G. P. Subramanian, S. J. Chung, and F. Y. Hadaegh, "Review of formation flying and constellation missions using nanosatellites," *Journal of Spacecraft and Rockets*, vol. 53, no. 3, pp. 567-578, 2016.

[8] A. Poghosyan and A. Golkar, "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions," *Progress in Aerospace Sciences*, vol. 88, pp. 59-83, 2017.

[9] D. Bekker et al., "The COVE Payload - A Reconfigurable FPGA-Based Processor for CubeSats," in *AIAA/USU Conf. on Small Satellites*, 2011.

[10] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Availability analysis for satellite data processing systems based on SRAM FPGAs," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 52, no. 3, pp. 977-989, 2016.

[11] A. Witze, "Software error doomed Japanese Hitomi spacecraft," *Nature*, 533, 2016.

[12] R. Ladbury, "Radiation hardening at the system level," in *IEEE Nuclear and Space Radiation Effects Conference Short Course Notebook*, 2007.

[13] J. W. Gambles, G. K. Maki, and S. R. Whitaker, "Radiation hardening by design," *Int. Journal of Electronics*, vol. 95, no. 1, pp. 11-26, 2008.

[14] S. Liu, G. Sorrenti, P. Reviriego, F. Casini, J. A. Maestro, and M. Alderighi, "Increasing Reliability of FPGA-based Adaptive Equalizers in the Presence of Single Event Upsets," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 3, pp. 1072-1077, 2011.

[15] L. A. Aranda, P. Reviriego, and J. A. Maestro, "Protecting Image Processing Pipelines Against Configuration Memory Errors in SRAM-based FPGAs," *Electronics*, vol. 7, no. 11, p. 322, 2018.

[16] L. A. Aranda, P. Reviriego, and J. A. Maestro, "Error Detection Technique for a Median Filter," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 8, pp. 2219-2226, 2017.

[17] L. A. Aranda, P. Reviriego, R. G. Toral, and J. A. Maestro, "Protection Scheme for Star Tracker Images," *IEEE Trans. Aerosp. Electron. Syst.*, doi: 10.1109/TAES.2018.2849919.

[18] M. Aung et al., "An Overview of Formation Flying Technology Development for the Terrestrial Planet Finder Mission," in *Proc. of the IEEE Aerospace Conf.*, Piscataway, NJ, 2004, pp. 2667–2679.

[19] R. J. Wilson and M. Schelkle, "The BepiColombo Spacecraft, Its Mission to Mercury and Its Thermal Verification," in *Lunar and Planetary Science Conf.*, The Woodlands, TX, 2015, p. 1058.

[20] S. Janson and R. Welle, "The NASA Optical Communication and Sensor Demonstration Program," in *AIAA Small Satellite Conf.*, 2013.

[21] R. H. White and G. R. Wirtenson, "Radiation Induced Darkening of the Optical Elements in the Startracker Camera," *Lawrence Livermore National Laboratory Report*, UCRL-ID-113713, University of California, Livermore, CA, 1993.

[22] B. R. Hancock et al., "CMOS active pixel sensor specific performance effects on star tracker/imager position accuracy," in *Int. Society for Optics and Photonics Symp. on Integrated Optics*, 2001, pp. 43-53.

[23] P. Martin-Gonthier, V. Goiffon, and P. Magnan, "In-pixel Source Follower Transistor RTS Noise Behavior under Ionizing Radiation in CMOS Image Sensors," *IEEE Trans. Electron. Devices*, vol. 59, no. 6, pp. 1686-1692, 2012.

[24] V. Goiffon et al., "Identification of Radiation Induced Dark Current Sources in Pinned Photodiode CMOS Image Sensors," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 4, pp. 918-926, 2012.

[25] B. Jahne, "Applications and Tools," in *Digital Image Processing*, 6th ed., Berlin, Germany: Springer, 1991, ch. 1, sec. 3, pp. 14-16.

[26] E. Jalabert et al., "Optimization of Star Research Algorithm for Esmo Star Tracker," in *8th International ESA Conference on Guidance, Navigation and Control Systems*, Karlovy Vary, Czech Republic, 2011.

[27] I. Bahri, L. Idkhajine, E. Monmasson, and M. E. A. Benkhelifa, "Hardware/software Codesign Guidelines for System on Chip FPGA-based Sensorless AC Drive Applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2165-2176, 2013.

[28] X. Iturbe et al., "Towards a Generic and Adaptive System-onchip Controller for Space Exploration Instrumentation," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015, pp. 1-8.

[29] X. Wei et al., "Development of a radiation-hardened SRAM with EDAC algorithm for fast readout CMOS pixel sensors for charged particle tracking," *Journal of Instrumentation*, vol. 9, no. 8, pp. 1-14, 2014.

[30] Y. Bentoutou, "A Real Time EDAC System for Applications Onboard Earth Observation Small Satellites," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 48, no. 1, pp. 648-657, 2012.

[31] D. G. Bailey, "Local Filters," in *Design for Embedded Image Processing on FPGAs*, 1st ed., Asia: John Wiley & Sons, 2011, ch. 8, sec. 1, pp. 233-239.

[32] J. L. Smith, "Implementing Median Filters in XC4000E FPGAs," *Xcell*, vol. 23, no. 4, p. 16, 1996.

[33] C. C. Liebe, "Accuracy Performance of Star Trackers - A Tutorial," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 38, no. 2, pp. 587-599, 2002.

[34] Xilinx, *Soft Error Mitigation Controller*, LogiCORE IP Product Guide, v4.1, 2015.