Scene-based non-uniformity correction: from algorithm to implementation on a smart camera

Tomasz Tockzek, Faouzi Hamdi, Barthélémy Heyrman, Julien Dubois, Johel Miteran, Dominique Ginhac

Scene-based non-uniformity correction: from algorithm to implementation on a smart camera

T. Toczek[a], F. Hamdi[a], B. Heyrman[a], J. Dubois[a], J. Miteran[a], D. Ginhac[a,*]

[a]*LE2I UMR 6306, Univ Burgundy, Dijon, France*

Abstract

Raw output data from image sensors tends to exhibit a form of bias due to slight on-die variations between photodetectors, as well as between amplifiers. The resulting bias, called fixed pattern noise (FPN), is often corrected by subtracting its value, estimated through calibration, from the sensor's raw signal. This paper introduces an online scene-based technique for an improved fixed-pattern noise compensation which does not rely on calibration, and hence is more robust to the dynamic changes in the FPN which may occur slowly over time. This article first gives a quick summary of existing FPN correction methods and explains how our approach relates to them. Three different pipeline architectures for realtime implementation on a FPGA-based smart camera are then discussed. For each of them, FPGA implementations details, performance and hardware costs are provided. Experimental results on a set of seven different scenes are also depicted showing that the proposed correction chain induces little additional resource use while guarantying high quality images on a wide variety of scenes.

Keywords: Fixed spatial noise, Non-uniformity correction, FPGA-based Smart Camera, real-time implementation

1. Introduction

CMOS image sensors are built around a grid of active pixels, each pixel containing a photodetector and an associated amplifier. The rest of the area on a CMOS sensor is comprised of readout circuitry, column amplifiers, analog to digital converters. Each element of the image sensor should ideally be identical to any other from the grid. Whenever an element differs slightly from the norm, it is likely to produce an incorrect response signal. As a result, the image sensors suffer from noise sources giving artifacts of the output data. Since the design of the first CMOS sensor, numerous works have been conducted to deeply characterize noise sources [1, 2, 3, 4] because they limit image sensor performance, especially under low illumination and/or long exposure conditions [5]. The noise sources can be divided into two distinct families *Temporal Noise* and *Fixed Pattern Noise* [6]. Temporal noise is independent across pixels and varies from frame to frame. Sources of temporal noise include photodetector shot noise, pixel reset circuit noise, readout noise for examples. On the other hand, image sensors also suffer from Fixed Pattern Noise (FPN). FPN is the pixel-to-pixel output variation under uniform illumination due to device and interconnect mismatches across the array [7]. It includes offset FPN and gain FPN. First of all, offset FPN can be defined as a dark signal non-uniformity, which is the offset from the average dark current across the imaging array in the absence of light [8]. It is independent of pixel signal and fixed from frame to frame for a given sensor. Offset FPN can be pixel and column noise [6, 9]. Pixel FPN is due to variations in the photodetector and transistors geometry whereas column FPN is mainly caused by mismatches in the integrating amplifiers and the analog-to-digital converters common to all the photodetectors of a given column [10]. Since such defects affect all the pixels of a column, column FPN appears as stripes in the image leading to significant image quality degradation. Secondly, gain FPN, which is not cancelled by CDS, is defined as the spatial variation in pixel output values under uniform illumination due to pixel-to-pixel mismatches [8]. Gain FPN is difficult to correct and for years, it was one of the major CMOS imager's disadvantages [11] because it increases with illumination and exponentially increases with temperature.

*Corresponding author
Email address: dginhac@le2i.cnrs.fr (D. Ginhac)

For achieving high-quality CMOS imagers, it is essential to develop good FPN compensation based on accurate models of sensors. While in the general case the FPN may be the result of any alteration of the photodetector transfer function, it can be modeled in a quite simple way. The pioneering model described in [6] is based on a detailed characterization of the silicon structure of the pixel. It focuses on offset noise and presents a detailed method for estimating the pixel and column FPN components. The same research team reports also a complementary study on gain FPN [12]. Most of the recent correction methods [10, 13] consider the FPN to be characterized at each photodetector by either a gain and an offset or just an offset because it is the dominant source of FPN in most pixels [14]. Indeed, this offset (i.e. the photodetector dark current) is the most easily seen manifestation of the FPN, considerably affecting the sensor output under low luminosity and/or long exposure conditions. Gain variations, or more complex nonlinear behaviors of the photodetector, are seldom seen in practice. Lots of works have been done to integrate electronic circuitry in the sensors to significantly reduce offset FPN. The most commonly used technique is Correlated Double Sampling (CDS) that can be found in many analogue circuits. The pixel signal is sampled twice, once immediately after reset and the second time after integrating the signal. These two values are then read differentially so that offset FPN is eliminated [15]. Various other techniques have been proposed to reduce FPN by either employing pinned photodiodes [16, 17] or modifying the pixel circuit itself [18, 19], or optimizing the column readout circuit [20], or subtracting the response of each pixel from its response to a uniform input current generated either optically [21] or electronically [22, 23]. However, despite the integration of numerous FPN-reduction circuits, CMOS image sensors still have FPN noise problems because of mismatched pixels and associating readout circuits, especially in extreme light conditions. As a result of the complex nature of sensor non-uniformities, FPN correction, also called non-uniformity correction (NUC), by means of efficient post-processing algorithms is a necessary and unavoidable task to be performed in order to achieve higher-quality images, or image sequences.

So, in this paper, we will concentrate on FPN reduction algorithms and hardware implementation of these algorithms on FPGA-based smart camera architectures. Since FPN varies slowly over time, depending mainly on the operating conditions (e.g. temperature) of the image sensor, this poses additional challenges to achieve a high quality image. Based on a state-of-the-art study of existing FPN correction methods, we present a new algorithm and different real-time hardware implementations for improved FPN compensation. This algorithm does not require any sensor calibration phase and is more robust to any dynamic changes in the FPN which may occur slowly over time. Our algorithm is naturally candidate for a real-time implementation on any FPGA-based smart camera architecture such those described in [24, 25, 26, 27].

The rest of the paper is organized as follows. Section 2 gives a quick summary of existing FPN correction methods, and explains how our approach relates to them. Section 3 deals with our improved algorithm based on the approach developed by Harris and Chiang [28]. Three different hardware implementations on a FPGA-based platform are then discussed in Section 4. Finally, implementation details, performance, and hardware costs are described in the last section of this paper. Experimental results on a set of seven different scenes are also depicted. Comparisons in terms of image quality between our implementations and original Harris and Chiang's algorithm are also provided.

2. FPN correction algorithms

In order to significantly decrease the FPN, non-uniformity correction must be applied to images acquired by the sensor. Depending on the accuracy requirements of the candidate applications, several post-processing methods can be applied to correct non-uniformities. These techniques include basic methods such as calibration-based methods and more complex techniques such as scene-based methods [29].

2.1. Calibration-based methods

As already said, FPN is mainly due to inhomogeneity in the manufacturing of the sensor. It is also known that the resulting non-uniformity does not widely change during the service life of the sensor. So, the most common way to deal with FPN is through a calibration step at the factory in order to precisely evaluate a reference-based correction. The most basic technique is to measure the sensor output when no light reaches the photodetectors to obtain a good approximation of it [30]. Then, the resulting "dark frame" can be stored in a ROM and subtracted at each frame from the actual sensor output. While this method is easy and inexpensive, this one-time calibration has the drawback to ignore FPN variations induced by the sensor operating conditions. It is also known that FPN is not

totally stationary but instead drifts slowly in time, making a one-time calibration ineffective. Of course, calibration can be done frequently to take into account the FPN variations but this unfortunately requires halting normal imaging operation during each calibration process. One way to compensate efficiently the FPN drift is to perform more than one calibration (generally two), each of them at a different temperature. Such a technique is mainly used for infrared focal-plane sensors [31] but have been also applied to more conventional image sensors [32]. The corrected image is then the one whose calibration temperature is the closest to the operating temperature, or possibly an interpolation of images whose calibration temperatures are around the operating temperature. In either case, at least twice as much ROM is required to store the calibration images, and a means to estimate the sensor operating temperature is required.

A more cost-effective solution in terms of memory is described in [33]. This better strategy for FPN estimation consists in analyzing supplementary data provided by the sensor. At the top of the sensor, a series of black and dark extra lines being shielded from light is positioned. Specifically, black lines have zero integration time whereas dark lines are shielded from incident light while having the same exposure as the image. Consequently, the data from these lines can be used to estimate a column FPN signature for the whole image. This signature is continuously averaged and updated according to the operating conditions by using the dynamic information provided by the black and dark lines and subtracted from the image data. This method only provides FPN estimation per column and is not able to compensate the variation between pixels. Additionally, despite interesting results described in [34], such a method is unfortunately not available for designers or end-users of smart cameras because it requires both the VLSI design of extra-lines of pixels in the core of the sensor and the possibility to access and read data from these extra-lines.

2.2. Dynamic FPN estimation by scene-based methods

This kind of techniques is called "scene-based" non-uniformity correction methods because they do not require specific scenes or initial conditions for the purpose of calibration. FPN is dynamically estimated and updated from the real-time analysis of the scenes captured by the sensor. Therefore, scene-based techniques are able to permanently evaluate FPN by exploiting motion-related features in image sequences. They bring the advantage to fit the drifting operating conditions without the need for any physical measurement, by adaptively using the information provided by the sensor. They can traditionally yield better results than calibration-based techniques, especially when the operating conditions of the sensor cannot be correctly approximated by a simple combination of the calibration measurements. It is obvious that such techniques increase system reliability but have higher computational complexity [35] requiring significant hardware resources to be processed at the sensor frame rate.

Scene-based non-uniformity correction (NUC) methods include two distinct categories of post-processing: (1) algebraic techniques, and (2) statistical techniques. As defined in [36], the algebraic techniques make use of global motion between the frames in the video sequence and attempt to compensate for the non-uniformity by means of algebraic methods without making statistical assumptions about the FPN [37, 38, 39]. Such approaches aim at analyzing the values of nearby pixels in order to separate the scene from the noise. Ratliff et al., in particular, have proposed an algebraical method [40] based on motion vector estimation. Such methods offer good robustness but are computationally heavy and not really suitable for real-time on-line correction.

On the other hand, statistical techniques model the FPN as a random spatial noise and estimate the statistics of the noise to remove it [41, 42, 43, 44]. Compared with registration-based methods, statistical approaches have been more widely studied because of their relatively lower computational complexity, smaller storage demands, and better realtime performance [44]. However, they rely on hypotheses regarding the sensor output signal (and therefore the scene) in order to separate the FPN from the actual signal. It is important for those hypotheses to be reasonable enough to be satisfied by most of the scenes acquired by the sensor. Nevertheless, when these assumptions are violated, performance can become poor and ghosting artifacts are easily produced. As an example, the most well-known statistical method developed by Harris and Chiang [28] supposes the average value and standard deviation of the photodetector output is the same among all the photodetectors of the sensor.

3. Improved Constant Statistics Method

We propose to extend the approach developed by Harris and Chiang [28] relying on the global constant statistic assumption which states that the statistics of the observed scene become constant over time. Our objective is to improve the method making it more flexible regarding the input scene, and then more efficient in any unknown visual scene.

The original paper models the non-linear behavior of each photodetector as an affine function. Each photodetector k is associated with a pair of scalars a_k and b_k such that the actual luminance $\tilde{I}_k$ received by the photodetector can be obtained from the measured one I_k through the relation:

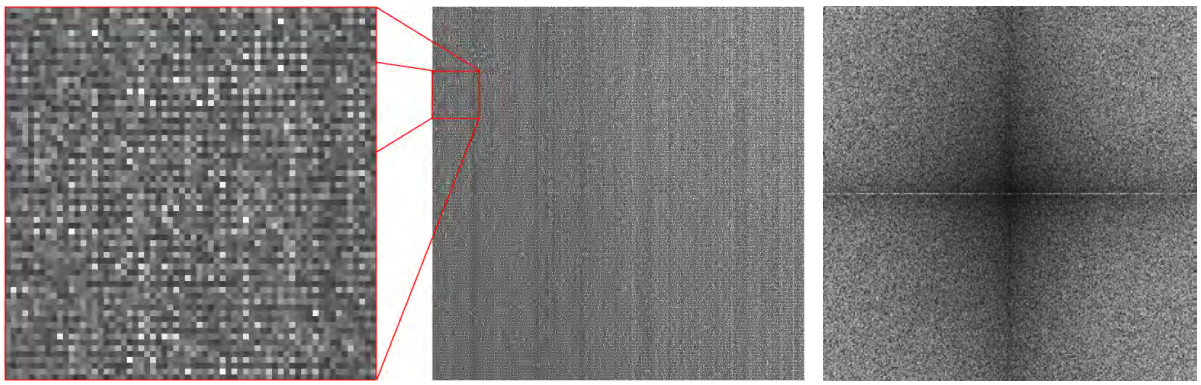
$$I_k = a_k \tilde{I}_k + b_k \tag{1}$$

Harris and Chiang assume that the measured luminance *over time* has the same mean and the same variance among all the photodetectors of the sensor. Under these conditions, it can be shown that for each photodetector k, assuming m_k is its average measured luminance and σ_k the measured luminance variance, the following holds:

$$\tilde{i}_k = \frac{I_k - m_k}{\sigma_k} \quad \text{and} \quad \tilde{I}_k = A\tilde{i}_k + B \tag{2}$$

There are several easy ways to iteratively estimate m_k and σ_k — for instance, an exponential window can be used —, so $\tilde{i}_k$ can be actually and cheaply computed. The $\tilde{i}_k$ signal mean is zero and its variance is unitary, but it is otherwise proportional to the actual luminance value plus an offset. $\tilde{I}_k$ can be obtained by knowing the factors A and B, which can be plausibly estimated by using the least squares method so as to minimize the error between the measured and corrected pixel values. Also, whenever the exact values of A and B do not matter, simply normalizing ($\tilde{i}_k$) over the dynamic range is enough (this is for example the case in IR imaging). This method has the advantage of being very simple, but is has several drawbacks in practice. The main problem is that the hypotheses regarding the mean and variance of the input signal are not satisfied in a lot of cases. Harris and Chiang recommend to move the camera or otherwise move the objects of the scene during calibration, but this implies the approach cannot be used to continuously calibrate the sensor. Whats more, even when the camera is moving and a dynamic scene is being shot, the mean and variance hypotheses may not hold. For instance, as the authors of the original paper admit, most of the time the output image is brighter at the top than at the bottom due to the fact that most light sources, both natural and artificial, illuminate the scene from above. We have observed that making accurate hypotheses regarding the FPN is much easier than regarding the scene. So, we propose to adapt this approach so as to make it work whenever the following is true for each photodetector k:

1. FPN has only an offset component (no gain component);
2. the mean value of the FPN offset is locally zero;
3. the high spatial frequency signal component at the corresponding pixel has an average value of zero over time.



(a) Dark current image, obtained by averaging a sequence of frames so as to filter out dynamic noise. Normalized.

(b) Corresponding Discrete Fourier Transform. Absolute magnitude is shown.

Figure 1: Dark current FPN of an OmniVision OV9715 sensor and corresponding DFT.

Concerning the validity of the first hypothesis, we can notice that FPN is almost only seen on low luminosity and/or long exposure shots. If a significant gain component exists, its influence should be noticeable regardless of luminosity

or exposure. Since it is not the case for almost any visible light sensor we studied, it means that either no FPN gain component exist at the photodetector level, or that current compensation mechanisms for this component work just fine, unlike those aiming to correct the offset component. The second hypothesis is more or less justified by assuming that manufacturing defects occur randomly. It can be partially confirmed by observing the output of the OmniVision OV9715 sensor embedded in the smart camera platform used in this study. Image capturing was performed while preventing any light from reaching the photodetectors as shown on Fig. 1a. A zoom on a 64×64-pixel sub-window illustrates the random distribution of noise. The absolute magnitude of its Discrete Fourier Transform is depicted on Fig. 1b and reveals that the column FPN contribution appears clearly as an horizontal line at $y = 0$. It can be seen that the high frequencies indeed dominate the image, but some low frequencies remain, however, and it appears that column FPN is the main source of them. The last constraint is the only one concerning the scene. Since the FPN is mostly a high frequency component, it is enough to average over time the high spatial frequencies at each pixel to determine the noise value, should this constraint be satisfied. In practice, it is often naturally satisfied, for instance whenever the camera is panning, or whenever an object traverses the field of view. While it is still possible to find pathological cases (for instance, a motionless camera shooting a completely static scene — in which case it seems impossible to remove the noise automatically and reliably short of using strong artificial intelligence anyway), it is much weaker and more naturally occurring than the constant mean/constant variance hypothesis used previously.

The general principle of our method is to remove the low spatial frequencies from the current frame, to apply the Harris and Chiang method on them, and to add them back to obtain the final image. When applying the Harris and Chiang method, we will assume that $a_k = 1$ for each k. Benefits of our method are various: 1) range of scenes correctly corrected is much wider and, 2) the post-processing dynamic range adjustment step is no longer necessary. Indeed, for each photodetector k, let us separate the high frequency component I_k^H from low frequency one I_k^L. Using the previous notations, we will have:

$$I_k^H + I_k^L = \tilde{I}_k + b_k \tag{3}$$

Assuming $a_k = 1$ (constraint 1), using the same reasoning as in the original article but on I_k^H instead of I_k, we get:

$$\tilde{i}_k^H = I_k^H - m_k^H \tag{4}$$

Where m_k^H is the average over time of the high spatial frequency component of the values of the photodetector k. The I_k^L component can be considered as noise free according to the constraint 2. Finally, let us notice that while $(\tilde{i}_k^H)$ is zero-meaned temporally just as in the original method, so is $(\tilde{I}_k^H)$, the high frequency component of the actual luminance the detector would measure were it perfect. This is a consequence of the constraint 3. So, we get:

$$\tilde{I}_k = \tilde{i}_k^H + I_k^L \tag{5}$$

This can be computed as we go, confirming that no further post-processing is needed. Low-pass filtering aside, we will just need to update the (m_k^H) estimate at each frame. The cheapest method to do this is to use an exponential window [45]. We will therefore take a scalar α, such as, if $\hat{m}_k^H(n)$ is the m_k^H estimate at the n-th frame:

$$\hat{m}_k^H(n) = \alpha I_k^H + (1 - \alpha)\hat{m}_k^H(n - 1) \tag{6}$$

Alternative methods include using a sliding rectangular window, which requires to store the history of the last I_k^H values, or accounting for all the samples, which would require a division. Those methods do not yield significantly better results than the exponential window based one.

4. Generic architecture for hardware implementation in a FPGA-based smart camera

Any implementation requires to perform two main tasks: first, extracting the low frequencies of the current frame through filtering, and then correcting the individual pixel values. The first part is a well understood problem, and the second is embarrassingly parallel. What is more, both parts rely more on arithmetical performance than on bit mangling or fine grained flow control. Finally, for low values of α, using floating point arithmetics may seem to be a good idea. As a result, processor-based machines with SIMD capabilities (such as PC equipped with programmable GPUs) are quite good candidates for real-time implementations but are not easily embeddable on a smart camera. The choice of an FPGA as the implementation target should be privileged whenever the consumption is a significant problem, the FPN correction logic is to be shipped with the sensor, and/or the latency of the sensor data induced by the correction must be kept minimal.

4.1. Correction pipeline architecture

According to the principles exposed so far, the correction chain can be implemented by a simple pipeline consisting of a low-pass filter followed by a correction block working at the pixel level. The corresponding architecture is shown on Fig. 2. The high-frequency data is obtained by simply subtracting the low frequencies from the raw signal. Those low frequencies are later added back to the corrected high-frequency data. The low-pass filter and the correction block have an average throughput of one datum per cycle. The correction block has a constant latency of several cycles. Consequently, using an appropriate delay is required to synchronize the low frequency component with the high frequency one before adding them back together. Depending on its exact implementation and parameters, the low-pass filter has a latency ranging from a few cycles to a few thousands cycles (see Section 4.3 for details). In any case, its latency is much shorter than a frame duration. Since the pixel correction block needs to store the temporal average value estimate m_k^H for each pixel k, it needs an access to the external memory if the input resolution is too high. Assuming a resolution of 1280×720 and using 16 bits per estimator, 1.84 MB of memory are required, i.e. more than most FPGAs can provide internally. It is possible to prefetch the estimator values and store them back once updated in a transparent fashion, using a small on-chip scratch pad, without affecting the overall correction block latency or throughput.

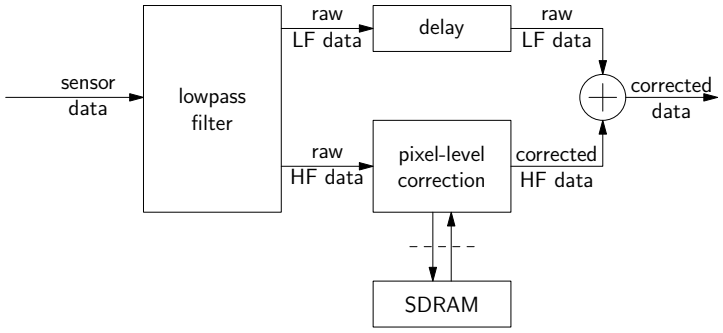


Figure 2: Correction pipeline, variant A (pixel level correction only)

While this first version gives satisfying results, it is interesting to add a block to specifically tackle column FPN correction. The column-level correction would be performed similarly as the pixel-level correction, except that at each frame, the average estimator m'^H_j for the column j would be updated by taking into account the values of every pixel of that column. Also, the corresponding m'^H_j will be subtracted from every pixel of a given column j. Let us call α' the equivalent of the α coefficient for column average estimator computation. Since they are updated more frequently than pixel average estimators, column average estimators converge more quickly. Convergence time issues are discussed in the Harris and Chiang's original article but also in [45] in the particular case of exponential windows. Moreover, column estimators are more robust than their pixel counterparts, and column correction is less likely to produce artifacts whenever any of our three working conditions is not met. In particular, they tend to give acceptable results even for highly static scenes.

Fig. 3 shows the most natural way to integrate column-level correction the previous pipeline architecture. One should note that no SDRAM access is required this time. Indeed, for a resolution of 1280×720 and using 20 bits per estimator (see Section 5.1 for justification), all of the (m'^H_j) take about 3.2 KB. In Section 3, we have seen that the column noise is actually the noise component with the most low frequencies (cf. Fig. 1b), and therefore the one most likely not to satisfy our second working constraint. Given this, we propose another variant (shown on Fig. 4) of the previous architecture. In this variant, the column correction occurs before the low-pass filter, directly on the sensor data (and not on their high frequency component). This is akin to using the original approach from Harris and Chiang on whole columns instead of pixels, before using our improvements for noise correction at the pixel level. However, this new approach leads to a latency drawback that has to be resolved. Indeed, since the output of the column corrector is by principle zero-meaned over time, the only way to obtain the correct output intensity is to compute the global spatial average pixel value of the frame before the column correction step, and to add it back behind the column correction block (for instance, directly after it). In order to compute this average, the values of all the pixels of the

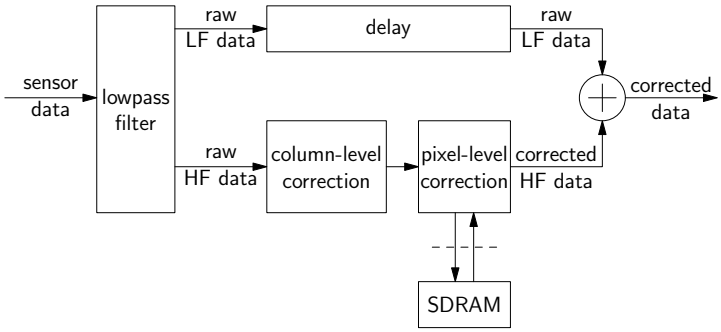


Figure 3: Correction pipeline, variant B (column and pixel level correction)

frame must be known. This implies a latency greater than a whole frame duration. This aspect can be heuristically mitigated by using the global average of the previous frame instead of the current one in the computations. So, there is no extra latency induced, but the corrected output will appear less precise whenever the scene luminosity varies too quickly between two successive frames.

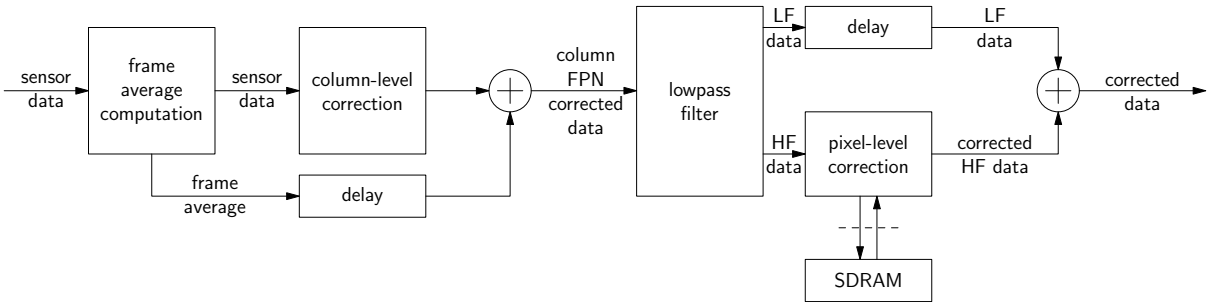


Figure 4: Correction pipeline, variant C (column level correction before filtering, standard pixel level correction)

4.2. Correction block

So far, we have not discussed the way the video data bus is organized. For the rest of this paper, we will assume it is composed of the video data itself (typically an 8-to-10 bit signal), and three one bit control signals: the active video signal indicating the video data is valid, and horizontal and vertical synchronization signals marking the beginning of a scanline or of a frame, respectively.

The overall principle of the column correction block itself is shown on Fig. 5. The main issue when implementing the correction blocks is the storage of the temporal averages for each pixel or each column. For column correction, when aiming for a throughput of one datum per cycle, it must be possible to read a coefficient and write another one during the same cycle. This can be done without using two port on-chip RAM by relying on two banks: one for even-numbered columns and one for odd-numbered columns. Assuming that the correction block is fed a valid pixel value every cycle, the column whose m'^{H}_{j} are read and written will have its index j alternating between even and odd. Since there is a FIFO before the writing port of the temporal average memory interface, it is possible to perform those read and write operations simultaneously by alternating between the two banks, no matter whether the number of stages of the correction block pipeline is even or odd (Fig. 5 shows a four stage pipeline, but it can be adapted easily without loss of generality). The FIFO will contain exactly one or zero element. This works as well when there are bubbles in the pipeline. The policy is simply not to write (by leaving the value to be written in the FIFO until the next cycle) whenever a bank conflict arises. For column correction, a FIFO size of exactly one element is enough to avoid overruns.

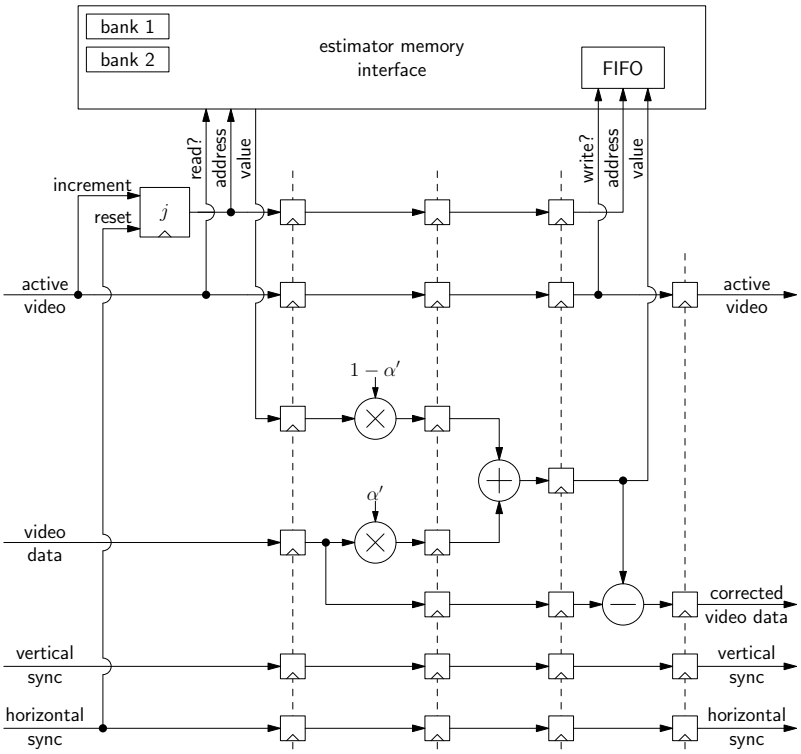


Figure 5: Column correction block principle schematic

The pixel correction block (Fig. 6) obeys essentially to the same principles, but in this case, the memory interface serves as a cache controller. Since the (m_k^H) are read and written in ascending order and that none of them is ever skipped, it is easy to prefetch them from the SDRAM, as well as store them back, in a timely manner, as long as one knows the worst case SDRAM access latency. However, to keep the throughput at one value per cycle without using double port on-chip memories, one will have to use four or more memory banks instead of two. The banks are associated by pairs of even and odd-numbered pixels (just like for columns), and at least two such bank pairs have to be used: one for the read and write operations from the correction logic, and another for storing back previously written values in the SDRAM as well as prefetching the ones to be read next. As previously, the FIFO located just before the writing port of the memory interface should have a size of at least one element for everything to work. It can nevertheless be larger, for instance when the correction block is used within a system-on-chip which contains other IPs using the SDRAM, or sharing the bus which leads to the SDRAM controller. In such systems, it may be difficult to estimate the worst-case SDRAM access latency. In such cases, it is possible to assume reasonable access times to external memory, and rely on the FIFO as a temporary store whenever the assumed deadline is not met.

4.3. Low-pass filtering

The low and high frequency components of frame data are obtained through low-pass filtering. The high frequencies are simply the difference between the original image and the low-pass filter output. There are several ways to filter an image so as to cut its high frequencies. Since we want to do noise correction on the high frequencies, we ideally have to to cut all the noise frequencies while cutting as little other frequencies as possible. When in doubt, it is definitely better to cut too much than too little. Indeed, the low frequencies are added back as is to the corrected signal, so any noise they carry will not be corrected. On the contrary, assuming the filter cuts everything but the fundamental, the result will be similar to that of a simplified (no noise gain correction) implementation of the Harris and Chiang method. Despite some severe limitations (in particular in presence of high amplitude low frequency components; see for instance [40]), it is supposed to remove the FPN as expected. Since it is difficult to predict ahead of time the

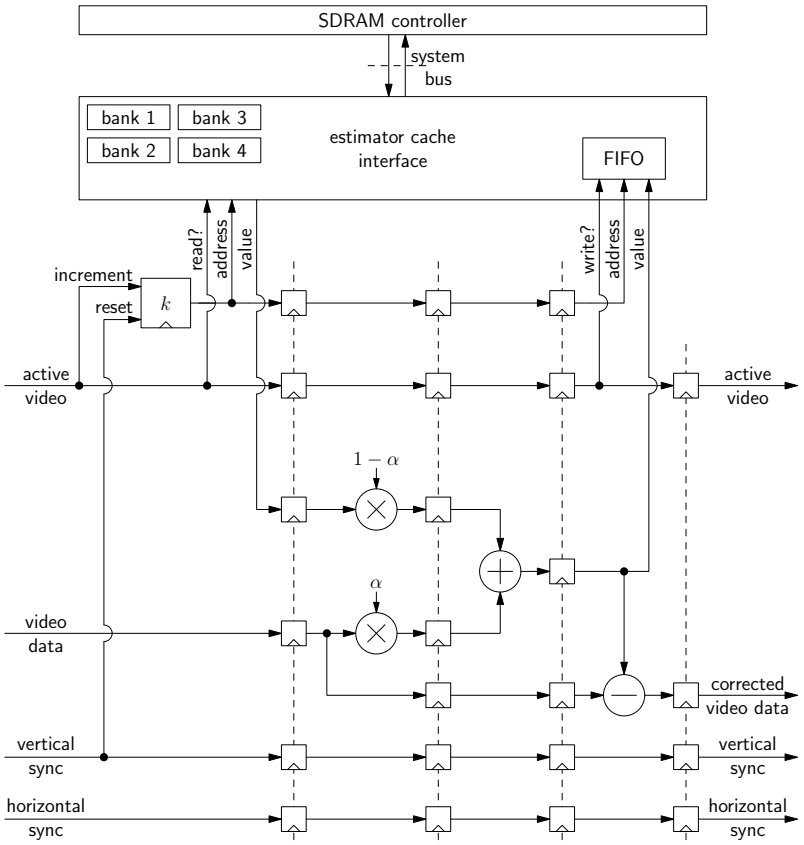


Figure 6: Pixel correction block principle schematic

amount of FPN a sensor will generate, the prudent course of action is to use a large filtering window. The exact kernel used matters little, as long as the filter is a low-pass. It should therefore be selected for its implementation simplicity.

An obvious solution is to use a rectangle window with a width and height that are powers of two, and obtain the result by summing all the pixel values of the window before right shifting them so as to obtain the spatial average. This has the advantage to be implementable without any multiplier or divider. Let N and M be integers such as the window size is 2^N lines by 2^M columns, and let w and h be powers of two greater or equal to the frame width and height, respectively. The most complicated part is the on-chip RAM organization. We will need three memories:

- the input memory I, storing as is the N last pixel lines, such that the pixel of coordinates (i, j) is at the address $j + (i \mod 2^N).w$; size: $w \times N$ elements
- the column memory C, storing for each column the average of the values of the corresponding pixels of the input memory; i.e., the value at the address j is $\frac{1}{2^N} \sum_{i=0}^{2^N-1} I(j + iw)$; size: w elements
- the result memory R, storing the final filtering result, that is $\frac{1}{2^M} \sum_{j=0}^{2^M-1} C(j)$ for a given pixel (i, j); however, the result is the average of the values of the pixels in the area $([(i-2^N+1)..i], [(j-2^M+1)..j])$, which is not centered on (i, j); the addressing scheme is identical to that of the I memory; size: $w \times N$ elements

For any pixel of coordinates (i, j) and of value p, the memories can be updated from their previous values using very few arithmetical operations:

- $I(j + iw) \leftarrow p$

- $C(j) \leftarrow c$ where $c = C(j) + \frac{p - I(j+iw)}{2^N}$
- $R(j + iw) \leftarrow R((j-1) + iw) + \frac{c - C(j-M)}{2^M}$

To avoid adding another port to the result memory due to the read of $R((j-1)+iw)$ in the last expression (and to avoid the $j-1$ subtraction), a register holding the last result computed may be used. It can be seen that the arithmetical complexity of those computations is of two additions and three subtractions (the $j - M$ subtraction must not be forgotten in the count). Four more registers are required to store the coordinates (i, j) of the last pixel read from the sensor, and the coordinates (i', j') of the last pixel output. A pixel is output whenever $i' \leq i - 2^{N-1}$ and $j' < j - 2^{M-1}$. Assuming $l_I = j' + (i' \mod 2^N)$ and $l_R = j' + 2^{M-1} + ((i' + 2^{N-1}) \mod 2^N).w$, the output values are $R(l_R)$ for the low frequency component, and $I(l_I) - R(l_R)$ for the high frequency component. This is because $R(l_R)$ is the average of the pixels of a zone approximately centered at the pixel of value $I(l_I)$. Incrementing i and j requires one adder, and i' and j' another one, while computing l_R requires two additions. Finally, subtracting $I(l_I)$ from $R(l_R)$ requires a subtraction. So, there are 10 adder/subtractors and two comparisons required, no matter the values chosen for N and M. It is quite clear most of the resources used will reside in the registers and the on-chip memory instead of arithmetic operators. The number of registers depends on the number of pipeline stages, which is tightly linked to the sensor operating clock frequency. It is also worth evaluating the number of ports required for each memory. It is possible to minimize them by using once again some ingenious bank organization:

- I requires three ports: for each valid pixel (i, j), $I(j + iw)$ must be read and written back, and $I(l_I)$ must be read. Since reading and writing $I(j + iw)$ can't be done anyway on the same pipeline stage on most FPGAs (it is generally not possible to read and write at the same address with predictable results), the same trick as in the correction blocks can be used: two banks of on-chip ram with selection based on the parity of j, and a 1-element FIFO just before the writing port gives good results. Each bank must be double port, though.
- C needs three ports but can be implemented with four banks of single port memory; the read addresses are j and $j - M$, and the written address is j. Let us use the j parity trick for the read and write at the address j, and use another pair of banks when the M^{th} bit, counting from the least significant to the most significant, changes. For instance, if $M = 2$ (i.e., when the window width is 4), the banks used for consecutive j would be 1, 2, 1, 2, 3, 4, 3, 4, 1, 2, 1, 2, etc. This works if the reads at j and at $j - M$ are located at the same pipeline stage.
- R needs two ports (if a register is used to store the last written value). By adding the extra constraint that an output is written only if j' and j have different parties, two banks of single port memory suffice.

Everything said so far does not take into account image borders. It is possible to handle them while still keeping a throughput of one value per cycle, but at the cost of extra combinatorial complexity. Alternatively, it is possible to "cut" the borders by removing the N first and last lines and the M first and last columns in order to keep the hardware simple. It is often desirable to take M such as $M > N$. This way, column noise is filtered more efficiently. An interesting alternative version of the low-pass filter can handle the extreme case where M is the image width and N is one. While this does not give the best results (see Section 5.1), it has the advantage of using very few hardware resources, in particular no on chip memory at all, and being fit to eliminate the column component of the FPN very efficiently. Also, by taking the approximation that the average value of the previous image line is almost equal to the one of the current, it is possible to implement such a filter with a latency of just a few cycles (depending on the number of pipeline stages), instead of the latency equivalent to the duration of 2^{N-1} lines for the previous implementation (which is quite low, especially when $N < M$, but still much more than a few cycles). This new version is pretty straightforward, with just one register summing all the pixel values of the line. At every horizontal synchronization, this value is divided by the number of pixels of the line (through fixed point multiplication), and stored in a separate register. The value of this new register is used as the low frequency output, and subtracted from the original pixel value in order to obtain the high frequency output.

5. Results

5.1. Correction quality

We have tested the proposed method by using video sequences tainted with artificial noise. This way, an image comparison metric can be used to estimate how much the corrected version differs from the original sequence. We

have chosen to use the UQI [46] as such a metric. It roughly corresponds to the human perception of distance between images. The value of the UQI between two images is in the $[-1, 1]$ range, and is 1 for identical images, 0 for uncorrelated images, and -1 for completely anticorrelated images. As recommended in [46], we used a sliding window of size 8×8 pixels for UQI computation. Generally, a high UQI corresponds to a low MSE and therefore a high PSNR, but it tends to offer a higher degree of correlation with human assessment of similarity between images than those two metrics, and this is why it has been preferred to them in this article.

The artificial offset noise used is simply a uniform pixel noise within ±12.5% of the dynamic range, plus an uniform column noise in this range as well. The video sequence used in the tests that follow consists of a 320×240 recording with 1110 frames. The scene is a room with no mobile objects in it, but the panning of the camera across the room provides enough information for noise elimination (see Fig. 7). In order to work correctly, statistics based methods need some time for the estimator values to converge. Therefore, in the rest of this section, the evaluations of efficiency will be given by computing the average UQI on the 100 last frames only. For reference, the average UQI of the 100 last frames is 0.081 when no correction is performed (the comparison is done between the original frames and their artificially noisy versions); evaluating the mean square error on all the pixels of those 100 last frames yields a PSNR of 20.28 dB. In the rest of this section, we will refer to the average UQI between a corrected version and the original sequence on the 100 last frames as the *correction quality*.

We have assumed that there is no loss of precision in the fixed point arithmetic operations of the correction and low-pass blocks. This may or may not be true depending on the actual implementation. Of course, some precision is lost when storing the estimators in memory. Let us assume the analog-to-digital converters of the sensor produce an unsigned fixed point value between 0 and almost 1, the point being located just before the most significant bit. Similarly, each average estimator is a signed fixed point number in the range $[-1, 1[$, and the coefficients α and α' are unsigned fixed point numbers in $[0, 1[$. Taking $\alpha = 2^{-n}$ for some natural n allows to trade the two multipliers of the pixel-level correction block for a subtractor. The same can be achieved for the column-level correction block by taking $\alpha' = 2^{-n'}$. We will therefore assume α and α' to have this form.

α'	α						
	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}
2^{-4}	0.739	0.779	0.792	0.799	0.801	0.702	0.395
2^{-5}	0.746	0.787	0.801	0.807	0.810	0.710	0.400
2^{-6}	0.751	0.791	0.806	0.813	0.815	0.715	0.402
2^{-7}	0.757	0.798	0.813	0.819	0.822	0.721	0.404
2^{-8}	0.766	0.808	0.822	0.829	0.832	0.730	0.410
2^{-9}	0.774	0.816	0.831	0.838	0.841	0.738	0.414
2^{-10}	0.779	0.821	0.836	0.843	0.846	0.742	0.418
2^{-11}	0.781	0.823	0.838	0.845	0.848	0.745	0.419
2^{-12}	0.782	0.824	0.839	0.846	0.849	0.745	0.420
2^{-13}	0.783	0.825	0.839	0.846	0.849	0.745	0.420
2^{-14}	0.739	0.787	0.806	0.819	0.832	0.738	0.418

Table 1: Correction quality depending on the pixel and column correctors exponential window coefficients (Pipeline B, 16×16 low-pass filter window, floating point estimators)

The Table 1 gives correction qualities depending on the values of α and α'. The estimators were stored as double precision floating point numbers for this measurement, so it is safe to assume that their precision was not an issue. The correction was performed according to the organization of the pipeline B (Fig. 3), with a 16×16 low-pass filter. One might have thought that the value of α' should be approximately equal to $\frac{\alpha}{\text{frame height}}$ (so as the column and pixel estimators to converge at the same rate per frame), but in fact it is better for the column corrector to converge as early as possible. This helps the pixel mean estimators to converge into correct values (that is, values accounting for the photodetector induced noise only, excluding any noise caused by the column amplifier). We will assume $\alpha = 2^{-8}$ and $\alpha' = 2^{-12}$ for the subsequent tests. Thus, the column estimators will converge $\frac{240}{16} = 15$ times more quickly than the pixel estimators. Once acceptable values for α and α' chosen, it is possible to choose the estimator data-type. Since

the required estimator precision depends on the value of α or α' (as seen on equation 6), it is advisable to store the column estimators with $\log_2 \frac{\alpha'}{\alpha}$ more bits than the pixel estimators. This leads to column estimators 4 bits larger than pixel estimators for the same computational precision, assuming the previously chosen values for α and α'. We have obtained the Table 2 by testing several candidate data-types. As we hoped, fixed point arithmetics are as suitable as floating point arithmetics, confirming that FPGAs are good candidates for the implementation of FPN correctors. In the rest of this section, we will assume the use of 16-bit fixed-point pixel estimators and 20-bit fixed-point column estimators.

estimator datatype and size (pixel / column)	correction quality
fixed point, 12-bit / 16-bit	0.527
fixed point, 14-bit / 18-bit	0.832
fixed point, 16-bit / 20-bit	0.849
fixed point, 18-bit / 22-bit	0.849
floating point, 64-bit / 64-bit	0.849

Table 2: Correction quality depending on the estimator data-type and precision (Pipeline B, 16×16 low-pass filter window, $\alpha = 2^{-8}$, $\alpha' = 2^{-12}$)

The Table 3 shows the impact of the low-pass filtering method and window size on the correction quality. One can see that the one-line filter (noted as $\infty \times 1$) gives quite honorable results compared to full-fledged rectangular window filtering, while being much simpler and not requiring any on-chip RAM at all. As expected, the horizontal filtering window size affects the correction quality more than the vertical window size, due to the column noise. The best value found is 64×32 on our test scene.

2^M	2^N						
	2	4	8	16	32	64	∞
1	0.143	0.251	0.415	0.600	0.737	0.805	0.905
2	0.184	0.323	0.518	0.713	0.832	0.879	
4	0.219	0.383	0.600	0.791	0.891	0.925	
8	0.242	0.423	0.648	0.833	0.922	0.950	
16	0.253	0.441	0.670	0.849	0.932	0.957	
32	0.256	0.448	0.676	0.853	0.934	0.958	
64	0.257	0.449	0.677	0.853	0.933	0.958	

Table 3: Correction quality depending on the low-pass filter parameters (Pipeline B, 16-bit/20-bit fixed-point estimators, $\alpha = 2^{-8}$, $\alpha' = 2^{-12}$)

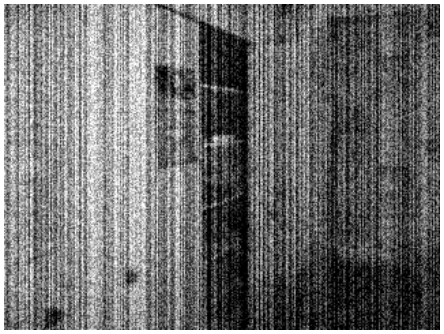
Finally, the Table 4 compares the three pipelines architectures proposed earlier, parametrized with the optimum parameters found until now (16-bit/20-bit fixed-point estimators, $\alpha = 2^{-8}$, $\alpha' = 2^{-12}$, 64×32 low-pass filter).

architecture	correction quality (UQI / PSNR)
pipeline A	0.961 / 36.36 dB
pipeline B	0.958 / 36.18 dB
pipeline C	0.890 / 28.78 dB
simplified Harris and Chiang method	0.845 / 24.78 dB

Table 4: Correction quality depending on pipeline architecture (16-bit/20-bit fixed-point estimators, $\alpha = 2^{-8}$, $\alpha' = 2^{-12}$, 64×32 low-pass filter; the simplified version of Harris' and Chiang's corrector uses double precision floating point arithmetics)

It also includes for comparison purposes a simplified (no gain correction) version of the Harris and Chiang corrector. On our test sequence just like on the series of 6 other real world sequences, it results in incorrect low frequency distribution due to unsatisfied working hypotheses, and yields worse results that any of the three other pipelines. The Fig. 7 shows a frame near the end of the sequence for each of the correction methods, once the estimators have converged. The pipeline C may produce "ghost columns" just like the original Harris and Chiang method may produce "ghost images". It is the case with our test scene (Fig. 7). It may give better results than the pipelines A or B when there is a significant low frequency component in the column FPN. It is not the case with our noise model, and is not generally the case with usual visible light domain sensors.

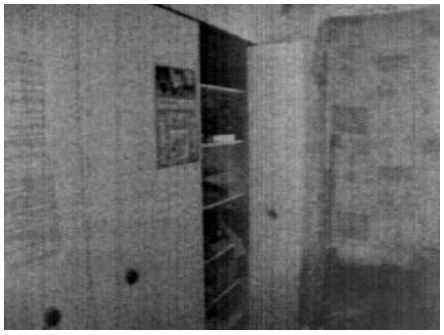
(a) Original (without artificial noise)



(b) Uncorrected

(c) Corrected using pipeline A

(d) Corrected using pipeline B



(e) Corrected using pipeline C

(f) Corrected using the Harris and Chiang method (with unitary gain)

Figure 7: 1010^{th} frame of our test scene (same correction parameters as in Table 4)

The pipeline B, which is a little more complex than the A, provides roughly the same correction quality. It converges a bit faster because of the column estimators as seen on Fig. 8. It also implies it may diverge more quickly

whenever the working hypotheses of our algorithm are not met. It might be possible to work around this limitation, though: an arbiter could disable the column estimator updates when it detects insufficient variation for a given column between two consecutive frames. While the same principle could be applied to the pixel-level corrector, it seems more practical to implement it on whole columns (dynamic noise is a major problem at the pixel level). This could allow to avoid the fluctuations in the MSE when using the pipeline B and which can be seen on Fig. 8.

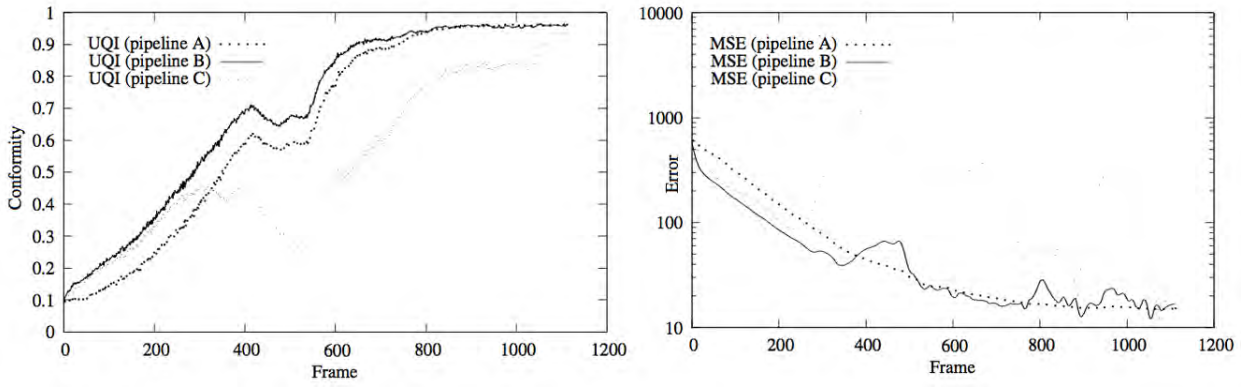


Figure 8: UQI and mean square error depending on the frame number; same sequence and parameters as in Table 4.

We have tested the three pipelines over a set of six other shorter video sequences in order to test the convergence of the proposed method. (Table 5).

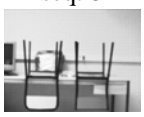
architecture	correction quality (UQI / PSNR)						
	seq. 1	seq. 2	seq. 3	seq. 4	seq. 5	seq. 6	ref. seq.
							
pipeline A	0.895 / 35.50 dB	0.254 / 20.11 dB	0.941 / 22.87 dB	0.762 / 33.01 dB	0.914 / 36.78 dB	0.833 / 22.40 dB	0.912 / 34.42 dB
pipeline B	0.885 / 34.25 dB	0.250 / 20.10 dB	0.940 / 22.23 dB	0.700 / 31.04 dB	0.907 / 36.72 dB	0.827 / 22.22 dB	0.888 / 33.54 dB
pipeline C	0.840 / 23.82 dB	0.220 / 18.23 dB	0.875 / 19.63 dB	0.683 / 31.92 dB	0.917 / 37.73 dB	0.823 / 21.84 dB	0.876 / 30.38 dB
simplified Harris and Chiang method	0.881 / 27.18 dB	0.203 / 14.59 dB	0.510 / 12.88 dB	0.417 / 23.76 dB	0.901 / 34.29 dB	0.795 / 16.47 dB	0.850 / 25.89 dB

Table 5: Correction quality for six shorter scenes (125 frames only, correction quality evaluated on the last 12 frames). Same parameters as in table 4, save for $\alpha = 2^{-5}$ and $\alpha' = 2^{-9}$. For comparison purposes, the previously used test sequence shortened to 125 frames appears in the column "ref. seq".

Each sequence only contains 125 frames, which is about eight times shorter than the previous test sequence. Sequences 1 and 2 use a still camera while some subjects are moving over part of its field of view. Sequence 1 has an uniform background, while sequence 2 has a very detailed still background (hence violating our third hypothesis). Sequences 3 to 6 involve a moving camera, and vary by the level of luminosity and of contrast. Sequences 3 and 4 are rather bright, while sequences 5 and 6 are almost pitch dark. High contrast elements are found on the sequences 4 and 6. The reference sequence is the one used in the previous tables, but restricted to its 125 first frames, for comparison purposes. Once again, the pipeline A gets the best results. As expected, the sequence 2 is the least effectively corrected but the proposed method still outperforms the Harris' and Chiang's on that sequence. According to PSNR measurements, the sequences 3 and 6 are also poorly corrected, while they are satisfying using a visual criterion, such

as the UQI. On the contrary, the corrected version of the sequence 4 shows visual shortcomings while having descent PSNRs for the three tested pipelines. Globally, the proposed correction method does not seem to specifically favour bright sequences over dark ones, and works for different levels of contrast.

5.2. Implementation on a smart camera

We have worked on implementing the proposed corrector using the Industrial Video Processing Kit commercialized by Avnet (part reference: AES-S6IVK-LX150T-G). It is composed of the general-purpose Avnet Spartan-6 LX150T development board (based on the Xilinx XC6SLX150T-3FGG676 FPGA) and of two FMC daughter cards with IO ports dedicated to video processing. It is shipped with an omnivision OV9715 image sensor [47], which can be connected to one of the daughter FMCs, and is capable of video capture at resolutions up to 1280×800 at 30 frames per second. The Fig. 9 is a photograph of the whole setup.

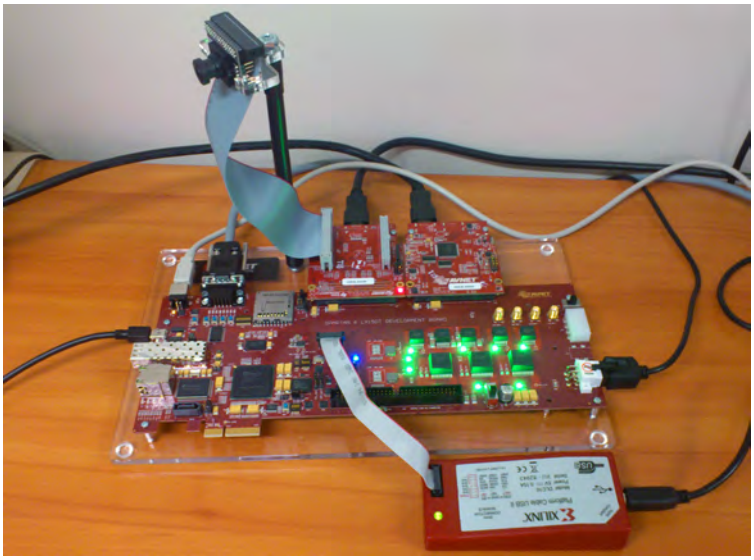


Figure 9: The Avnet Industrial Video Processing Kit

We have implemented the different parts of the proposed correction pipelines and assembled them into a pcore for use with the Xilinx EDK. Proceeding like this eases its reuse can allows its parametrization through memory-mapped registers accessible from software via the PLB bus. The resulting pcore can be instantiated inside virtually any existing XPS design just between the camera input and the first actual video processing IP.

The implementation itself has been done in VHDL and in Haskell. VHDL was used for the control-dominated components, for those dealing with more than one clock domain (typically, at the interface with the memory controller), and whenever the critical path length was not a limiting factor. We have used an in-house Haskell based hardware description embedded language to generate pipelined netlists from combinatorial datapaths. Most of the low pass and corrector blocks, which are arithmetics-dominated, could be implemented quite straightforwardly this way. One of the benefits of our automatized pipelining process is that arithmetical operators (such as adders, subtractors or multipliers) are not necessarily bound to a single pipeline stage, as it is the case when using a VHDL library such as IEEE `numeric_std`. On the contrary, the pipeline stages were generated in such a way that the critical paths of the different stages have approximately the same length. As a result, in our implementation the barriers between pipeline stages do not match those shown on the schematics of Fig. 5 and Fig. 6. However, as discussed later in this section, the illustrated total pipeline stage count is sensible.

A minimalist design was used for our tests as illustrated on Fig. 10. It is a system with two DMAs: one used to write at 30 frames per second the output of the sensor to the external DDR3 memory, and the other to read it and output it through a DVI port at 60 frames per second. The sensor output is treated with our corrector before being transmitted to the DMA controller. Finally, a Xilinx Microblaze softcore is used to handle the initialization of all the peripherals,

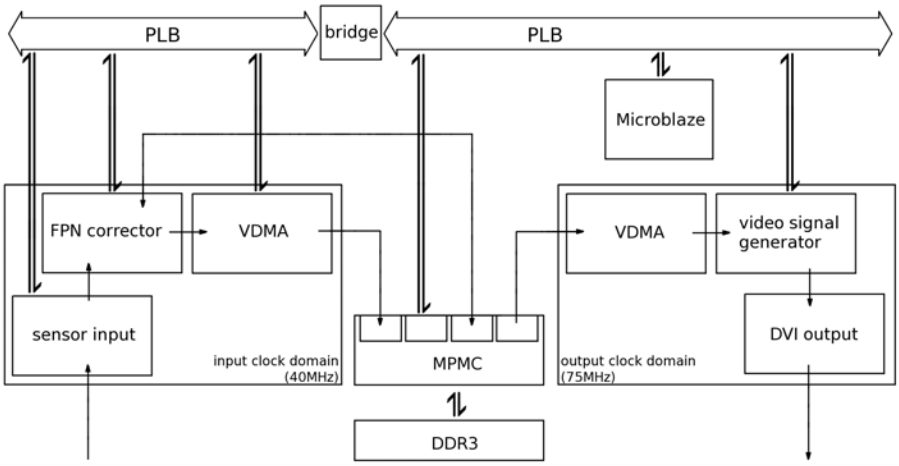


Figure 10: Minimalist corrector pcore test design

and can be used during operation to alter parameters such as sensor exposure and gain, resolution, and so on. During our tests, those parameters have been fixed and any automatic sensor behaviour disabled. Nevertheless, our correction pipeline should be able to operate with automatic exposure and gain adjustments, as long as their variations are slow. It should also be possible to feed in real time the sensor gain value to the corrector for better results, especially with quickly varying gain. Those aspects are currently being investigated and will not be discussed further in this paper.

We have evaluated the hardware complexity of our pipelines in the case where they use a NPI (native port interface) point-to-point link to the Xilinx MPMC external memory controller and the one-line version of the lowpass filter. The NPI interface is the simplest of the interfaces supported by the MPMC, but requires synchronicity with the MPMC clock domain (which differs from the video input clock domain in our system). Other possible implementation choices were the use of the VFBC (video frame buffer controller) point-to-point link, or simply to access the memory controller through the PLB. However, using the NPI interface as well as the one-line filter, much simplier than a conventional rectangle window low pass filter, allows us to get an upper bound of the frequency achievable and a lower bound of the slice usage.

# pipeline stages		Frequency	Pipeline A		Pipeline B		Pipeline C	
Lowpass	Corrector		# registers	# LUTs	# registers	# LUTs	# registers	# LUTs
1	1	80.652 Mhz	496	176	580	289	742	311
2	2	99.502 Mhz	608	277	748	280	910	302
4	3	140.076 Mhz	773	497	994	762	1156	784
5	4	149.410 Mhz	844	490	1099	745	1261	767
6	4	169.319 Mhz	895	475	1172	596	1334	618
7	5	129.232 Mhz	946	513	1258	786	1420	808
8	6	158.278 Mhz	1007	503	1348	854	1510	876
11	8	148.170 Mhz	1128	517	1546	671	1708	693
22	16	134.391 Mhz	1347	719	1878	1124	2040	1146

Table 6: FPGA implementation performance and ressource usage, depending on pipeline depths.

The table 6 sums up the performance of the correction pcore and its resource usage measured on the above-described test system, depending on the parameters passed to our pipelining tool. As expected, the higher the pipeline stage count, the higher the register count and the higher the slice usage, as more of them are used either as registers without accompanying logic, or simply as route-thrus. The highest reachable frequency is around 169 MHz, when

using 6 pipeline stages for the low pass filter and 4 stages per corrector (the pixel and column correctors share the same architecture, so their pipelines have the same length). The maximum frequency goes down when using deeper pipelines, most likely due to routing issues as the number of slices increases. It is interesting to note that in our test system, we could have afforded to use single stage correctors, as the sensor input clockrate is 40 MHz. This is not the case for systems using better cameras: with a sensor capable of 1080p capture at 60Hz, a working frequency of 125 MHz or higher is necessary, requiring moderately deep correction pipeline use. When comparing the resources used those available in the FPGA, the slice register usage ranges from 0.27% to 1.11%, and the slice LUT usage from 0.20% to 1.25%.

6. Conclusion

We have proposed an improved algorithm for FPN correction in visible light sensors and three associated architectural pipelines that can be easily implemented in any FPGA-based smart camera. Its hardware complexity is marginally higher than that of a simple reference image while it significantly achieves improved quality of resulting images. Moreover, it is more robust to variations in the sensor environment or operating mode. Contrarily to the original approach we improve on, it is capable of handling a wide variety of scenes correctly by separating the signal from the noise in a more reliable fashion. This is achieved by taking into account information from the spatial neighborhood of the pixels being corrected.

Finally, our correction pipeline adds little latency in most cases and hence is suited for incorporation in applications where this is an issue. A first notable future work perspective is the automatic detection of static scenes. Indeed, a mechanism to temporarily prevent the update of estimators when two consecutive frames are too similar would allow to preserve their correctly computed values when the convergence hypotheses are not met, further increasing correction robustness. A second perspective is the implementation of this method in other FPGA smart cameras developed in our research group. We aim at implementing such a correction method to the HDR-ARtist platform [48] dedicated to high dynamic range imaging in order to achieve the best quality possible.

Acknowledgment

The authors would like to thank the DGCIS (French Ministry for Industry) for financial support within the framework of the project HiDRaLoN.

References

[1] Tian, H.. Noise analysis in cmos image sensors. Ph.D. thesis; 2000.
[2] Tian, H., Fowler, B., Gamal, A.. Analysis of temporal noise in cmos photodiode active pixel sensor. IEEE Journal of Solid-State Circuits 2001;36(1):92 –101.
[3] Nakamura, J.. Image Sensors and Signal Processing for Digital Still Cameras. Boca Raton, FL, USA: CRC Press, Inc.; 2005.
[4] Morfu, S., Marquie, P., Nofiele, B., Ginhac, D.. Chapter 3: Nonlinear systems for image processing. In: Hawkes, P.W., editor. Advances in Imaging and Electron Physics; vol. 152. Elsevier; 2008, p. 79 – 151.
[5] HPComponentsGroup, . Noise sources in CMOS image sensors. Tech. Rep.; Hewlett-Packard Company; 1998.
[6] Gamal, A.E., Fowler, B., Min, H., Liu, X.. Modeling and estimation of fpn components in cmos image sensors. In: Proc. SPIE, Solid State Sensor Arrays: Development and Applications II; vol. 3301. 1998, p. 168–177.
[7] El Gamal, A., Eltoukhy, H.. Cmos image sensors. IEEE Circuits and Devices Magazine 2005;21(3):6 – 20.
[8] Konnik, M.V., Welsh, J.S.. On numerical simulation of high-speed ccd/cmos-based wavefront sensors in adaptive optics. vol. 8149. SPIE; 2011,.
[9] Kelly, S.C., Guidash, R.M., Pillman, B.H.. Fixed pattern noise removal in cmos imagers across various operational conditions. 2008.
[10] Schöberl, M., Senel, C., Fößel, S., Bloss, H., Kaup, A.. Non-linear dark current fixed pattern noise compensation for variable frame rate moving picture cameras. In: Proceedings of European Signal Processing Conference (EUSIPCO). Glasgow, Scotland; 2009,.
[11] Bigas, M., Cabruja, E., Forest, J., Salvi, J.. Review of cmos image sensors. Microelectronics Journal 2006;37(5):433 – 451.
[12] Fowler, B., Gamal, A.E., Yang, D., Tian, H.. A method for estimating quantum efficiency for cmos image sensors. In: Proc. SPIE, Solid State Sensor Arrays: Development and Applications II; vol. 3301. 1998, p. 178–185.
[13] Schoandberl, M., Foandssel, S., Kaup, A.. Fixed pattern noise column drift compensation (cdc) for digital moving picture cameras. In: Image Processing (ICIP), 2010 17th IEEE International Conference on. 2010, p. 573 –576.
[14] Otim, S., Choubey, B., Joseph, D., Collins, S.. Characterization and simple fixed pattern noise correction in wide dynamic range logarithmic imagers. IEEE Transactions onInstrumentation and Measurement 2007;56(5):1910 –1916.

[15] Nixon, R., Kemeny, S., Pain, B., Staller, C., Fossum, E.. 256x256 cmos active pixel sensor camera-on-a-chip. IEEE Journal of Solid-State Circuits 1996;31(12):2046 –2050.
[16] Guidash, R., Lee, T.H., Lee, P., Sackett, D., Drowley, C., Swenson, M., et al. A 0.6mu cmos pinned photodiode color imager technology. In: IEDM '97. Electron Devices Meeting, 1997. Technical Digest. 1997, p. 927 –929.
[17] Yonemoto, K., Sumi, H.. A cmos image sensor with a simple fixed-pattern-noise-reduction technology and a hole accumulation diode. IEEE Journal of Solid-State Circuits 2000;35(12):2038 –2043.
[18] Kavadias, S., Dierickx, B., Scheffer, D., Alaerts, A., Uwaerts, D., Bogaerts, J.. A logarithmic response cmos image sensor with on-chip calibration. IEEE Journal of Solid-State Circuits 2000;35(8):1146 –1152.
[19] Loose, M., Meier, K., Schemmel, J.. A self-calibrating single-chip cmos camera with logarithmic response. IEEE Journal of Solid-State Circuits 2001;36(4):586 –596.
[20] Degerli, Y., Lavernhe, F., Magnan, P., Farre, P.. Column readout circuit with global charge amplifier for cmos aps imagers. Electronics Letters 2000;36(17):1457 –1459.
[21] Dierickx, B., Dierickx, B., Scheffer, D., Scheffer, D., Meynants, G., Meynants, G., et al. Random addressable active pixel image sensors. In: Proceedings of the SPIE; vol. 2950. 1996, p. 2–7.
[22] S. Kavadias, B.D., Scheffer, D.. On-chip offset calibrated logarithmic response image sensor. In: Proc. IEEE Workshop Charge-Coupled Devices and Advanced Image Sensors. 1999, p. 68–71.
[23] Lai, L.W., Lai, C.H., King, Y.C.. A novel logarithmic response cmos image sensor with high output voltage swing and in-pixel fixed-pattern noise reduction. IEEE Sensors Journal 2004;4(1):122 – 126.
[24] Mosqueron, R., Dubois, J., Paindavoine, M.. High-speed smart camera with high resolution. EURASIP Journal on Embedded Systems 2007;(1):024163.
[25] Mosqueron, R., Dubois, J., Mattavelli, M., Mauvilet, D.. Smart camera based on embedded hw/sw coprocessor. EURASIP Journal on Embedded Systems 2008;(1):597872.
[26] Real, F.D., Berry, F.. Smart cameras: Technologies and applications. In: Belbachir, A.N., editor. Smart Cameras. Springer US. ISBN 978-1-4419-0953-4; 2010, p. 35–50.
[27] Lapray, P.J., Heyrman, B., Rosse, M., Ginhac, D.. Smart camera design for realtime high dynamic range imaging. In: Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2011). 2011, p. 1 –7.
[28] Harris, J., Chiang, Y.M.. Nonuniformity correction using the constant-statistics constraint: Analog and digital implementations. In: Proceedings of SPIE; vol. 3061. 1997, p. 895–905.
[29] Ratliff, B., Hayat, M., Tyo, J.. Generalized algebraic scene-based nonuniformity correction algorithm. J Opt Soc Am A 2005;22(2):239–249.
[30] Richard M. Malueg, G.C.. Detector array fixed-pattern noise compensation. 1976.
[31] Friedenberg, A., Goldblatt, I.. Nonuniformity two-point linear correction errors in infrared focal plane arrays. Optical Engineering 1998;37(4):1251–1253.
[32] Newman, J.W.. Method and apparatus for image signal compensation of dark current, focal plane temperature, and electronics temperature. 2007.
[33] Bosco, A., Findlater, K., Battiato, S., Castorina, A.. A temporal noise reduction filter based on image sensor full-frame data. In: IEEE International Conference on Consumer Electronics, ICCE 2003. 2003, p. 402 – 403.
[34] Bosco, A., Findlater, K., Battiato, S., Castorina, A.. A noise reduction filter for full-frame data imaging devices. IEEE Transactions on Consumer Electronics 2003;49(3):676 – 682.
[35] Huang, T., Xiao-Lin, L., Yang, S.. Fixed pattern noise suppression algorithm based on background modeling. Procedia Engineering 2012;29(0):884 – 888.
[36] Sakoglu, U., Hardie, R., Hayat, M., Ratliff, B., Tyo, J.. Nan algebraic restoration method for estimating fixed pattern noise in infrared imagery from a video sequence. In: 9th Annual Meeting of the SPIE: Applications of Digital Image Processing XXVII, Denver, CO, SPIE Proc; vol. 5558. 2004, p. 69–79.
[37] Hardie, R.C., Hayat, M.M., Armstrong, E., Yasuda, B.. Scene-based nonuniformity correction with video sequences and registration. Appl Opt 2000;39(8):1241–1250.
[38] Torle, P., Andersson, I.A., Haglund, L.. Scene-based correction of image sensor deficiencies. vol. 5074. SPIE; 2003, p. 249–260.
[39] Zuo, C., Chen, Q., Gu, G., Sui, X.. Scene-based nonuniformity correction algorithm based on interframe registration. J Opt Soc Am A 2011;28(6):1164–1176.
[40] Ratliff, B.M., Hayat, M.M., Hardie, R.C.. An algebraic algorithm for nonuniformity correction in focal-plane arrays. J Opt Soc Am A 2002;19(9):1737–1747.
[41] Harris, J.. Continuous-time calibration of vlsi sensors for gain and offset variations. In: Proceedings of the SPIE: Smart Focal Plane Arrays and Focal Plane Array Testing; vol. 2474. 1995, p. 23–33.
[42] Vera, E., Reeves, R., Torres, S.. Adaptive bias compensation for non-uniformity correction on infrared focal plane array detectors. In: HIS. 2002, p. 725–734.
[43] Zhang, C., Zhao, W.. Scene-based nonuniformity correction using local constant statistics. J Opt Soc Am A 2008;25(6):1444–1453.
[44] Zuo, C., Chen, Q., Gu, G., Sui, X., Qian, W.. Scene-based nonuniformity correction method using multiscale constant statistics. Optical Engineering 2011;50(8):087006 (pages 11).
[45] Kumar, A., Sarkar, S., Agarwal, R.. Fixed pattern noise correction and implementation for infrared focal plane array based staring system using scene statistics. International Journal of Industrial and Systems Engineering 2007;1.
[46] Wang, Z., Bovik, A.C.. A universal image quality index. IEEE Signal Processing Letters 2002;9(3):81–84.
[47] Ov9715 1-megapixel product brief. 2011. OmniVision Technologies, Inc.
[48] Lapray, P.J., Heyrman, B., Rosse, M., Ginhac, D.. Hdr-artist: High dynamic range advanced real-time imaging system. In: IEEE International Symposium on Circuits and Systems (ISCAS 2012). 2012, p. 1 –6.