An FPGA-oriented HW/SW Codesign of Lucy-Richardson Deconvolution Algorithm for Hyperspectral Images

Karine Avagian*, Milica Orlandić*, Tor Arne Johansen †
* Department of Electronic Systems,
† Centre for Autonomous Marine Operations and Systems (NTNU-AMOS),
† Department of Engineering Cybernetics
Norwegian University of Science and Technology
karineav@stud.ntnu.com, milica.orlandic@ntnu.no, tor.arne.johansen@ntnu.no

***Abstract*—Hyperspectral images acquired during remote sensing are usually corrupted by optical blur and additive noise. This article considers a method to reduce the spatial blur in hyperspectral images by applying the Lucy-Richardson deconvolution algorithm. The algorithm is based on convolution with a point-spread function, which is assumed to be known. The complete algorithm is implemented as a HW/Sw Co-Design between the central processing unit and the programmable logic. The convolution, as a time consuming operation, is accelerated in an FPGA. When compared to the software implementation, a speed-up by a factor of 2.7 is achieved when using the accelerated HW/SW Co-Design implementation. Hyperspectral images synthetically distorted by a known Gaussian blur and an additive noise are used for testing. The images are then restored by applying the Lucy-Richardson deconvolution algorithm on each spectral band.**

***Keywords* —** hyperspectral images, deconvolution, Lucy-Richardson Algorithm, FPGA

I. Introduction

A hyperspectral data cube is a composition of multiple images, also called bands, depicting the same scene at different wavelengths λ. The hyperspectral cube is a three-dimensional array, where a sample is defined by its position (x, y, λ). A pixel with fixed spatial (x, y) coordinates contains a set of N_λ components in the spectral domain, where each element describes the intensity of the reflected radiation. The reflected radiation plotted as a function of the wavelength is called the spectral reflectance curve and it shows the spectral signature of the objects surface.

Hyperspectral imaging (HSI) has been used in remote sensing Earth observation in recent decades. In HSI remote sensing applications, the observed data are degraded by various sources such as atmospheric effects and instrumental noise [1]. In addition, an image can be distorted by optical or motion blur. Optical blur is caused by the camera's optics and is modeled with the instrument's response function, whereas motion blur is caused by either the motion of the camera or the motion of the observed object. In this article, the optical blur and the additive Gaussian noise are assumed to be the only noise sources present in the acquired images. The image restoration can be performed on all spectral bands at the same time or in band-by-band manner assuming that the distortion in the bands in the hyperspectral image are independent of each other [2]. The latter assumption holds in the case there are no cross-channel correlations. If the blur, described by the convolution, is also assumed to be spatially invariant across the band, the image restoration can be modelled as the inverse process of the convolution, i.e. deconvolution.

A number of algorithms for image restoration, such as Tikonov-Miller [3], Wiener filter [4] and Lucy-Richardson deconvolution [5], [6] have been widely used. In this paper, the iterative Lucy-Richardson (LR) deconvolution algorithm is chosen to be implemented based on experimental results [7] in comparison to the other well-known deconvolution algorithms. The LR algorithm belongs to the non-blind deconvolution class of algorithms requiring prior knowledge about the blur in order to successfully restore the degraded image. The algorithm is, however, characterized by the high computational complexity equal to $\mathcal{O}(MNmn)$ in spatial domain, where $M \times N$ is image size and $m \times n$ is the size of convolution kernel. By the assumption that the kernel is separable, the computational complexity changes to $\mathcal{O}(MN(m+n))$. In addition to the software optimization, the paper presents the acceleration of the LR algorithm done by partitioning its functions between FPGA and general-purpose processors, where the time-consuming operation, convolution, is performed by a dedicated accelerator on programmable logic.

This paper is divided in the following sections: in Section II, the LR algorithm is described and recent state-of-the-art implementations of the algorithm are presented. In Section III, the proposed architecture is presented. In Section IV, the visual results from the software simulations and a comparison of timing results between the proposed hardware-software codesign implementation and a software-only implementation are presented. Finally, in Section V, the conclusions are drawn.

II. Background

Based on the assumption that the noise is represented by an independent and identically distributed Gaussian function and the spectral bands are independent, the image restoration can be performed on each band separately [8]. In the LR deconvolution algorithm, it is assumed that image $g(x, y)$ in

each band λ of a hyperspectral three-dimensional data cube is blurred by the two-dimensional point-spread function (PSF), $h(x, y)$. The LR algorithm is then defined by the recursion:

$$\hat{f}(x,y)^{n+1} = \hat{f}(x,y)^n \cdot \left[\frac{g(x,y)}{\hat{f}(x,y)^n * h(x,y)} * h(-x,-y) \right] \quad (1)$$

where $h(-x,-y)$ is flipped PSF, $\hat{f}(x,y)$ is the estimated image, $g(x,y)$ is the observed/degraded image, n is the index of iteration and the operator $*$ represents convolution. The multiplication and division are element-wise operations. The convolution operation is a neighborhood operation, where an input image region of the same size as the PSF is required. In addition, a 2D Gaussian PSF $h(x,y)$ is isotropic i.e. $h(x,y) = h(-x,-y)$. An initial estimated image $f(x,y)^0$ is usually set to be equal to the average intensity of the input image $g(x,y)$ or to the input image $g(x,y)$.

A. State-of-the-art

The LR deconvolution algorithm is usually used in the spatial domain [9], [10]. For shift-invariant PSF, the algorithm can be also employed in the frequency domain [11] based on the fact that the convolution operation in the space domain is transformed into the matrix multiplication in the frequency domain. In the work presented in [11], the frequency transform is done by performing the Fast Fourier Transform (FFT) and the Inverse Fast Fourier Transform (IFFT). The implemented method uses an embedded DSP solution for FFT and IFFT computations and a processing system for controlling the rest of the algorithm. The system is implemented on the Xilinx Virtex-4 with the maximum operating frequency of 100 MHz, whereas the operating frequency of DSPs is 600 MHz. The implementation is tested on the images of size 64 $\times$ 64 with a 13 $\times$ 13 kernel. Recent works [9], [10] implement the LR deconvolution algorithm fully in hardware. In [9], it is assumed that the blur is caused by the motion of the imaging system and that there is no additive noise. The architecture consists of a convolution module, a division module and a multiplication module. The testing is performed for image of size 800 $\times$ 525 and 9 $\times$ 9 kernel. For the given parameters, the maximum operating frequency is 61 MHz. The memory requirement is a limiting factor which does not support large images. In particular, for the space-variant point-spread function, it is required not only to store the image, but also the coefficients of the kernels. An implementation in [10] uses the space-variant PSF to model the lens distortions. The PSF is described as the 2-D asymmetric Gaussian function, and a different PSF is associated to each pixel. The images of size 640 $\times$ 480 are used for testing and the dimensions of the PSF kernels are smaller than 10.

B. Measure of quality

The performance of the LR algorithm is evaluated using the Peak Signal-to-Noise Ratio, PSNR, and Structural Similarity Index, SSIM. The PSNR metric is a standard quality metric for a comparison of the image manipulation methods and is computed across spectral bands as follows:

$$PSNR(\hat{f}, f, \lambda) = 10log_{10} \frac{peakval^2}{MSE(\hat{f}(\lambda), f(\lambda))} \quad (2)$$

where *peakval* is the largest value for a defined data width and MSE is mean square error between the restored image $\hat{f}(\lambda)$ and original blur- and noise-free image $f(\lambda)$.

The measure of restoration quality SSIM is given as:

$$SSIM(\hat{f}, f, \lambda) = [l(\hat{f}, f, \lambda)^\alpha \cdot [c(\hat{f}, f, \lambda)]^\beta \cdot [s(\hat{f}, f, \lambda)]^\gamma \quad (3)$$

where $l(\hat{f}, f, \lambda)$ compares the luminance of the observed objects in the reference image $f(\lambda)$ to the luminance of the same object in the estimated image $\hat{f}(\lambda)$, $c(\hat{f}, f, \lambda)$ is the contrast comparison and $s(\hat{f}, f, \lambda)$ is the comparison in the structural variations [12]. The parameters α, β and γ define the relative weight of each component. In order to assess the image quality, the SSIM index is applied locally, on the small image regions, rather than on the whole image. The overall image quality is then computed by the mean SSIM, *MSSIM*, as follows:

$$MSSIM(\hat{\mathbf{F}}(\lambda), \mathbf{F}(\lambda)) = \frac{1}{M} \sum_{i=1}^{M} SSIM(\hat{f}, f, \lambda) \quad (4)$$

where $\hat{\mathbf{F}}(\lambda)$ and $\mathbf{F}(\lambda)$ are the restorted and the reference images, respectively, $\hat{f}_i(\lambda)$ and $f_i(\lambda)$ are the image elements at the i-th local window of $\hat{\mathbf{F}}(\lambda)$ and $\mathbf{F}(\lambda)$ respectively and M is the number of local windows [12].

III. Architecture

A HW/SW codesign architecture is implemented and tested on the ZedBoard development board with Zynq-7020 SoC. The complete system is presented in Fig. 1 with processing system, memory and accelerator with optimized datapaths in programmable logic. The general program flow with portions of the algorithm partitioned between the host processing system and programmable logic is presented in Fig. 2. The processing system loads and exports images from an SD-card, type-casts data to/from fixed-point or floating point representations, performs element-wise multiplication and division operations, whereas the convolution is accelerated in programmable logic. The initialization and communication of the accelerator is controlled by the processing system. The implemented application-specific accelerator performs spatial convolution between an input image and a Gaussian kernel.

The implemented Convolution Accelerator consists of a convolution module and controllers as shown in a detailed block diagram in Fig. 3. The read and write operations are controlled by two separate controller modules. In addition, a counter module counts until the first valid output pixel is produced. The convolution module proposed by [13] is used for convolution computation of an image of size $M \times N$ with a kernel of size $k \times k$. In order to accelerate the convolution process, separability of the Gaussian kernel is exploited. The separability property provides a possibility to perform two 1-D convolutions with $1 \times k$ and $k \times 1$ kernels instead of 2-D convolution with kernel $k \times k$.

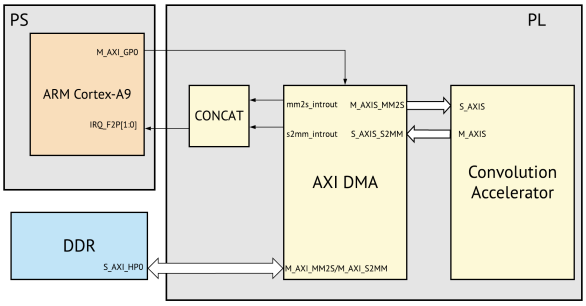


Fig. 1: Block design for LR deconvolution.

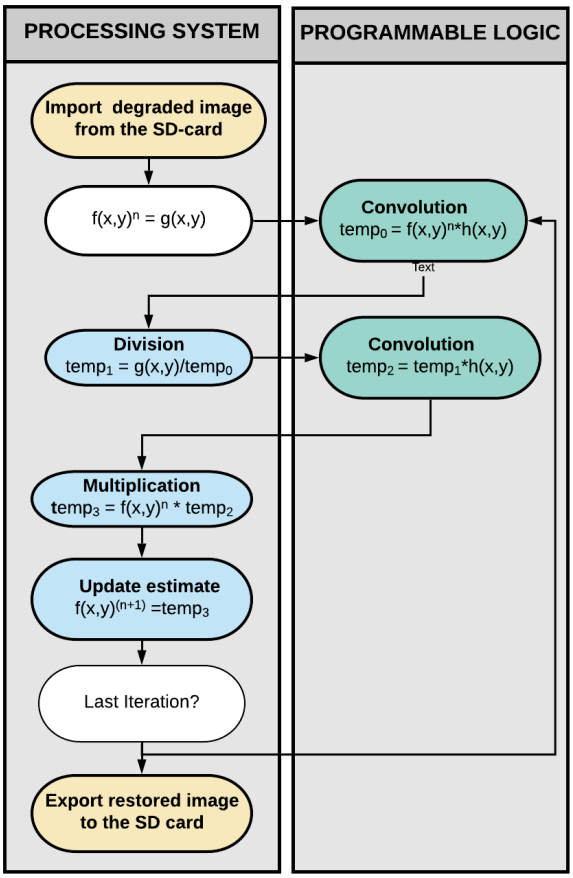


Fig. 2: A general program flow for the hardware/software implementation of the LR-deconvolution.

Convolution is performed by moving the kernel across the input image. At each position, a sample spanned by the neighborhood of the same size as the kernel, is processed. As presented in Fig. 4, the samples are used several times during the processing. For example, the sample I_{14} is used 7 times as a neighbor in the convolution process. The redundant reloading results in longer processing time due to the communication latency with the external memory. To avoid unnecessary loading of the same sample into the convolution module, line buffers are used locally in the accelerator. In Fig. 5, the hardware architecture of a separable 2-D convolution module is presented. The input sample is initially directly used in

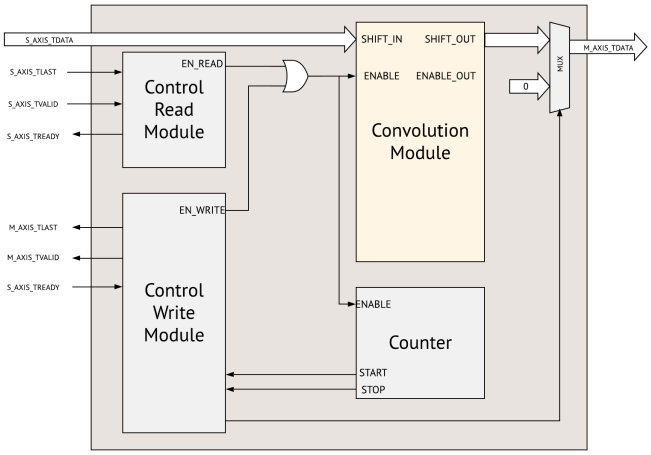


Fig. 3: Block diagram for the convolution module.

the convolution computation, and it is sent to the first line buffer in the next clock cycle. The number of required line buffers is $(k-1)$, whereas the size of each buffer is equal to the number of samples within a line in the x direction of the band, N_x. Computation of the 1-D convolution with the kernel, $H(y)$, is performed by loading the last elements from each line buffer into the computational module. The results of the multiplications are summed and sent to the second 1-D convolution module where the multiplication and summation are performed on the results of the 1-D convolution with the kernel $H(x)$.

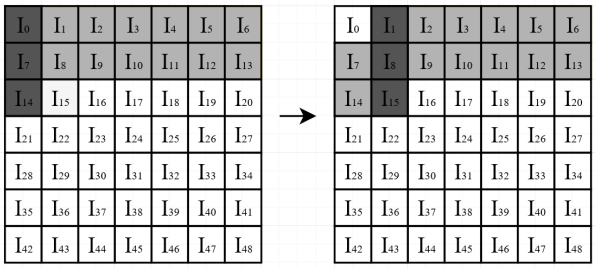


Fig. 4: An example of data requirements for 1-D convolution with a 3×1 kernel.

IV. Results

The proposed HW/SW codesign implementation partitions the LR deconvolution algorithm between processing system and programmable logic. The computationally intensive convolution operation is accelerated by the convolution module implemented in the programmable logic. The module is described by the VHDL language, and the Vivado tool is used for synthesis, implementation, system integration, testing and verification on a Zedboard development board [14]. For dataflow testing and verification, a 9×1 Gaussian kernel with $\sigma = 2.3$ is used as the PSF function, where the kernel coefficients are given as:

$$H = \frac{1}{128} \cdot [5 \quad 10 \quad 16 \quad 21 \quad 23 \quad 21 \quad 16 \quad 10 \quad 5]. \quad (5)$$

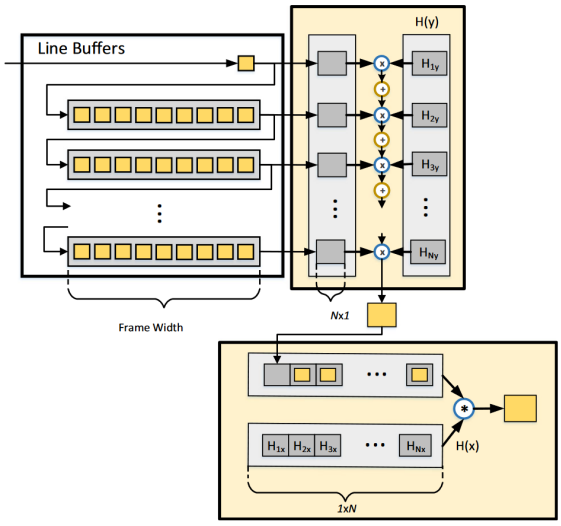


Fig. 5: The separable 2-D convolution [13].

The testing and verification of the complete implementation are performed by the use of *Pavia* hyperspectral data set acquired by the ROSIS sensor [15] and characterized by 103 spectral bands and spatial resolution 610×340 pixels. A gray-scale image of the 70-th band is shown in Fig. 6.

Fig. 6: *Pavia* data set, 70th band

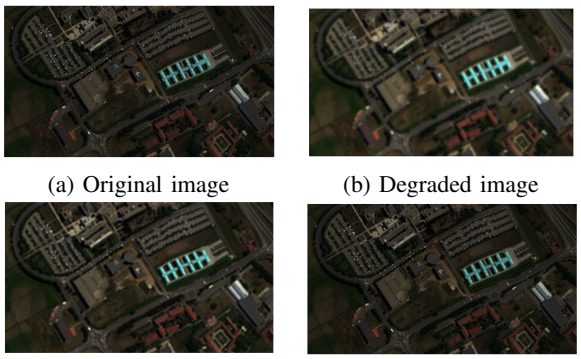
A. LR deconvolution on noise-free data

The results obtained after LR deconvolution on *Pavia* data set is shown in Fig. 7. The original image in Fig. 7(a) is the composite of three spectral bands (50, 30 and 10). The blurred image using the Gaussian kernel with σ = 2.3 and size $k = 9 \times 9$ is given in Fig. 7(b). The results after running LR-deconvolution for 10 and 1000 iterations are shown in Fig. 7(c) and Fig. 7(d), respectively. The quantitative measurements are shown in Table I, where the distorted and restored images are compared to the undistored image.

TABLE I: LR deconvolution on noise-free *Pavia* data set

	PSNR (dB)	MSSIM
Blurred	46.0666	0.9763
Deblurred 10	47.8156	0.9846
Deblurred 1000	50.5024	0.9918

It is observed that the estimated images converge to the original image both visually by inspecting the restored images in Fig. 7 and quantitatively by the use of quality metrics in Table I, where both PSNR and MSSIM values increase with the higher number of iterations.



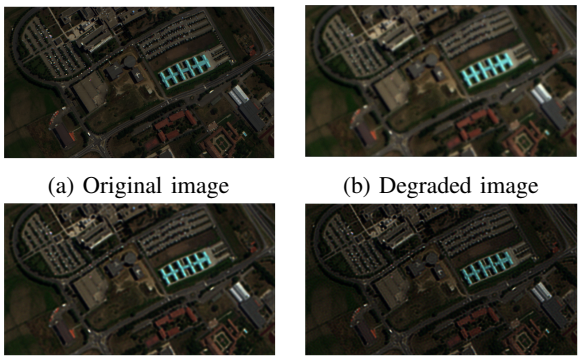
(a) Original image (b) Degraded image

(c) 10 LR iterations (d) 1000 LR iterations

Fig. 7: *Pavia* hyperspectral cube before and after LR deconvolution

B. LR deconvolution on noisy data

In this scenario, the *Pavia* data set is degraded by both Gaussian blur and additive Gaussian noise with two different mean value and variance sets, $[0, 0.0001]$ and $[0, 0.01]$. The LR deconvolution algorithm is tested for 10 and for 100 iterations. The degraded and restored images are given in Fig. 8 for the case of additive noise with $\sigma_1^2 = 0.0001$ and in Fig. 9 for the case of additive noise with $\sigma_2^2 = 0.01$.

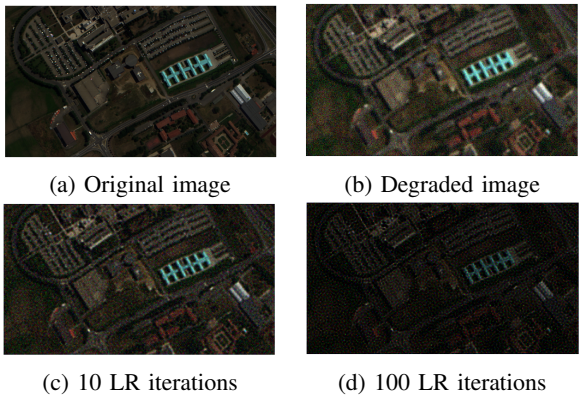


(a) Original image (b) Degraded image

(c) 10 LR iterations (d) 100 LR iterations

Fig. 8: Results obtained by running LR deconvolution algorithm on the *Pavia* data set degraded by the Gaussian blur with an additive Gaussian noise with $\sigma^2 = 0.0001$.

The quantitative measurements are shown in Table II. The quality of the restored images in the presence of noise depends on the amount of noise. If the noise level in the degraded images is relatively low, the LR deconvolution succeeds to restore images. For σ_1^2, both PSNR and MSSIM result in accepted quality metric values. For high noise levels, the LR deconvolution tends to amplify the noise. In Table II for σ_2^2, it is observed that both the PSNR and the MSSIM values decrease after LR deconvolution is employed. Thus, in the presence of large amount of noise, it is required to remove the noise before performing LR deconvolution.



(a) Original image (b) Degraded image

(c) 10 LR iterations (d) 100 LR iterations

Fig. 9: Results obtained by running LR deconvolution algorithm on the *Pavia* data set degraded by the Gaussian blur with an additive Gaussian noise with $\sigma^2 = 0.01$.

TABLE II: Performance metrics of LR deconvolution on noisy *Pavia* hyperspectral cube

	PSNR (dB)	M-SSIM
Noisy σ_1^2 = 0.0001	45.9745	0.9757
Deblurred 10	47.4241	0.9831
Deblurred 100	47.5869	0.9837
Noisy σ_2^2 = 0.01	41.3364	0.9266
Deblurred 10	41.0394	0.9232
Deblurred 100	36.6892	0.8204

C. Performance Analysis

The LR deconvolution algorithm is implemented both as a software-only solution running on the ARM processing system on Zynq SoC and as a HW/SW codesign implementation accelerating computationally intensive portions of LR deconvolution algorithm by placing them on FPGA.

One iteration on a gray-scale image of size 610×340 coded with 16-bit values takes 108 ms in a software-only solution. While the processing of the same image in the proposed method takes 40 ms. A speed-up by a factor of 2.7 is achieved using the HW/SW codesign implementation compared to software-only implementation. The maximum achieved operating frequency is 114 MHz. The post-implementation utilization summary is shown in Table III. In order to further optimize the performance of the implementation, both ARM cores of the Zynq processing system can be used to perform deconvolution of two different bands in parallel.

TABLE III: The post-implementation resource utilization for the input image of size 640×310 with address width 16-bits.

Resource	Utilization	Available	Utilization %
LUT	4930	53200	9.27
LUTRAM	598	17400	3.44
FF	6004	106400	5.64
BRAM	6	140	4.29

V. Conclusions

The paper proposes an efficient HW/SW codesing implementation of a fast LR deconvolution algorithm. The choice of the algorithm is based on low complexity, high performance and regular structure which allows to be accelerated on FPGA. Both software-only and HW/SW codesign implementations have been proposed and compared, where a speed-up by a factor 2.7 is achieved by using the HW/SW implementation. Further work can include an improved partitioning of sequential operations on the processing system and/or further mapping of computationally intensive operations to programmable logic.

Acknowledgement

This work was supported by the Research Council of Norway (RCN) through MASSIVE project, grant number 270959, and AMOS project, grant number 223254.

References

[1] Behnood Rasti, Paul Scheunders, Pedram Ghamisi, Giorgio Licciardi, and Jocelyn Chanussot. Noise reduction in hyperspectral imagery: Overview and application. *Remote Sensing*, 10(3):482, 2018.

[2] Haiyan Fan, Chang Li, Yulan Guo, Gangyao Kuang, and Jiayi Ma. Spatial-spectral total variation regularized low-rank tensor decomposition for hyperspectral image denoising. *IEEE Transactions on Geoscience and Remote Sensing*, PP, 05 2018.

[3] G. M. P. van Kempen, H. T. M. van der Voort, J. G. J. Bauman, and K. C. Strasters. Comparing maximum likelihood estimation and constrained Tikhonov-Miller restoration. *IEEE Engineering in Medicine and Biology Magazine*, 15(1):76–83, Jan 1996.

[4] R T Bates and M. J. McDonnell. *Image Restoration and Reconstruction*. Oxford University Press, Inc., New York, NY, USA, 1986.

[5] L. B. Lucy. An iterative technique for the rectification of observed distributions. , 79:745, jun 1974.

[6] William Hadley Richardson. Bayesian-Based Iterative Method of Image Restoration. *J. Opt. Soc. Am.*, 62(1):55–59, Jan 1972.

[7] Timo Bretschneider. On the deconvolution of satellite imagery. volume 4, pages 2450 – 2452 vol.4, 07 2002.

[8] Stanley J. Reeves. Chapter 6. image restoration: Fundamentals of image restoration. *Academic Press Library in Signal Processing*, 4, 12 2014.

[9] O. Anacona-Mosquera, J. Arias-García, D. M. Muñoz, and C. H. Llanos. Efficient hardware implementation of the Richardson-Lucy Algorithm for restoring motion-blurred image on reconfigurable digital system. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, Aug 2016.

[10] S. Carrato, G. Ramponi, S. Marsi, M. Jerian, and L. Tenze. FPGA implementation of the Lucy-Richardson algorithm for fast space-variant image deconvolution. In *2015 9th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 137–142, Sept 2015.

[11] Ze Wang, Kaijian Weng, Zhao Cheng, Luxin Yan, and Jing Guan. A co-design method for parallel image processing accelerator based on DSP and FPGA. In *MIPPR 2011: Parallel Processing of Images and Optimization and Medical Imaging Processing*, volume 8005, page 800506. International Society for Optics and Photonics, 2011.

[12] Gabriel Prieto Renieblas, Agustín Turrero Nogués, Alberto Muñoz González, Nieves Gómez-León, and Eduardo Guibelalde Del Castillo. Structural similarity index family for image quality assessment in radiological images. *Journal of medical imaging*, 4 3:035501, 2017.

[13] Milica Orlandić and Kjetil Svarstad. An adaptive high-throughput edge detection filtering system using dynamic partial reconfiguration. *Journal of Real-Time Image Processing*, Feb 2018.

[14] Avnet. ZedBoard, Hardware Users Guide. http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf, 2014. [Online; accessed 16-December-2018].

[15] (GIC) Grupo de Inteligencia Computacional. Hyperspectral Remote Sensinf Scenes. http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes, 2014. [Online; accessed 12-December-2018].