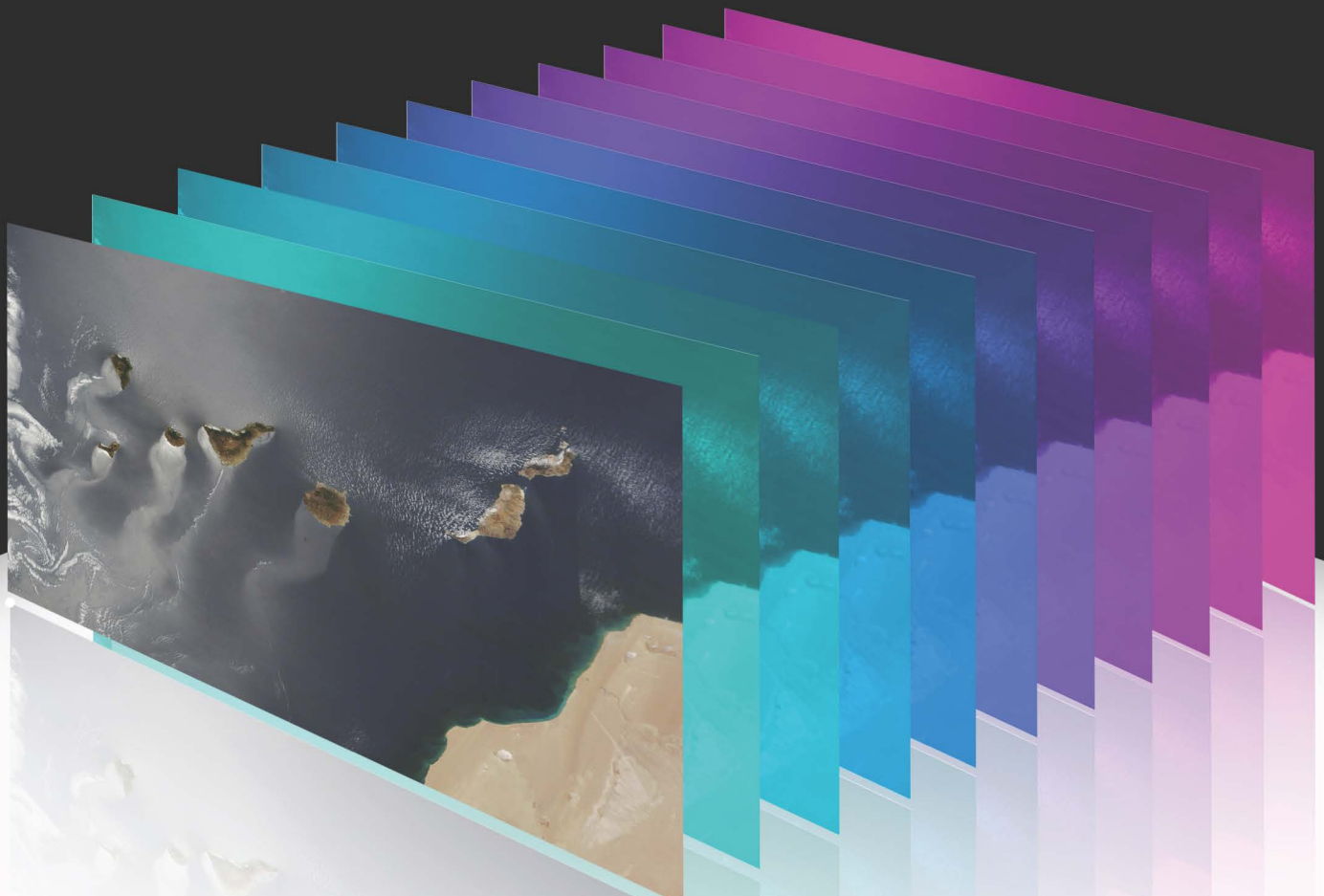
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

Towards the efficient processing of hyperspectral images: new hardware-friendly algorithms and OpenCL-based implementations

Doctorado en Tecnologías de la Telecomunicación



Tesis doctoral
Raúl Guerra Hernández
Las Palmas de Gran Canaria, Mayo 2017

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

División de Diseño de Sistemas Integrados

Tesis Doctoral

Towards the efficient processing of hyperspectral images: new hardware-friendly algorithms and OpenCL-based implementations

Raúl Celestino Guerra Hernández

Abstract

Hyperspectral imaging systems are considered nowadays one of the most powerful tools for acquiring surface information in many different fields. These systems are able to provide images in which single pixels have information from across the electromagnetic spectrum of the scene under observation. This information enables identification of the materials that make up a scanned object, providing a much better understanding of the scene under analysis.

In order to take full advantage of the hyperspectral imaging technology and make it useful for applications under real-time constraints, it is important to rapidly process the huge amount of data provided by the hyperspectral sensors. For such purpose, parallel computing platforms, such as GPUs or FPGAs, are typically used. Nevertheless, the mathematical complexity of the existing algorithms for processing hyperspectral images, as well as their so many data dependences, do not allow to take full advantage of the computing capability of the parallel devices, preventing the use of hyperspectral imaging technology in applications under real-time constraints. Accordingly, the development of new hardware-friendly algorithms which can be efficiently implemented into parallel devices for real-time applications is a at the same time a necessity and a challenge.

This Thesis proposes new hardware-friendly algorithms for processing hyperspectral images, as well as technological solutions for implementing them into parallel computing devices for applications under real-time constraints. In particular, three different hyperspectral domains, namely *linear hyperspectral unmixing*, *multispectral and hyperspectral image fusion* and *hyperspectral and ultraspectral image compression*, have been targeted, since providing solutions for these three domains will have positive repercussions in hyperspectral imaging analysis in general, and particularly, in applications with tight latency constraints.

First, one algorithm, named *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), has been developed. This algorithm is able to perform the entire unmixing process and does not require a previous spectral reduction of the data set. The FUN algorithm is based on the concept of orthogonal projections, involving very simple operations which can be independently performed for each pixel of the image. All these facts together allow to obtain a more efficient hardware implementation of the entire unmixing process in a parallel device.

One algorithm, named *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI), has been also developed. This algorithm is capable of independently fusing each pixel of the hyperspectral image. The CoEf-MHI algorithm is also based on the concept of orthogonal projections and involves very simple operations. Due to these reasons, this algorithm represents a very suitable option for being implemented in parallel devices for fusion applications under real-time constraints.

Furthermore, one algorithm, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), has also been developed. This algorithm provides an efficient solution for performing the spectral decorrelation and reduction of hyperspectral and ultraspectral images for their compression. The HyperLCA transform is based on orthogonal projections too, involving very simple operations. It is specially designed for independently processing small blocks of pixels of the image, being able to preserve the anomalous pixels through the compression-decompression process. All these characteristics make the HyperLCA algorithm a very suitable option for compressing hyperspectral images when the time and hardware resources are limited.

Finally, the FUN algorithm has been implemented in different parallel hardware devices, namely GPUs and FPGAs, and using different implementation languages, namely NVIDIA CUDA and OpenCL. Since the FUN algorithm, the CoEf-MHI algorithm and the HyperLCA algorithm are all three based on orthogonal projections and make use of the modified Gram-Schmidt orthogonalization as their main operation core, it is assumed that the results obtained for the implementations of the FUN algorithm, in terms of efficiency, could be transposed to the rest of the developed algorithms within this Thesis work. The accomplished implementations prove that real-time or near real-time results can be obtained by using algorithms thought from their conception to be implemented into parallel hardware. As a further research goal, the utility of OpenCL for developing hardware implementations for different parallel hardware devices, such as GPUs and FPGAs, using one single language with efficient results, have also been proved.

The benefits of the proposed algorithms and implementations have been addressed and compared in terms of accuracy and computational performance with the most relevant up to date works, confirming the contribution made by the work to the real-time hyperspectral imaging analysis research field.

Resumen

Actualmente los sistemas hiperespectrales son una de las herramientas más potentes para la adquisición de información superficial, siendo esta de gran utilidad en muchas áreas de aplicación. Estos sistemas proporcionan imágenes en las que cada pixel posee información espectral de la escena observada. Esta información permite identificar los materiales que componen la superficie de los objetos escaneados, permitiendo un mayor entendimiento de la escena analizada.

Para poder sacarle partido a la tecnología hiperespectral y poder emplearla en aplicaciones que deban funcionar en tiempo real es importante que la gran cantidad de datos capturados con estos sensores sea procesada rápidamente. Para ello se emplean dispositivos hardware de cómputo paralelo como GPUs y FPGAs. Sin embargo, la complejidad matemática de los algoritmos existentes para el análisis hiperespectral, así como las dependencias de datos inherentes a dichos algoritmos, impiden el aprovechamiento eficiente de los recursos hardware, impidiendo así mismo su ejecución en tiempo real. Por este motivo, el desarrollo de algoritmos altamente paralelizables, pensados desde su concepción para ser implementados eficientemente en plataformas de cómputo paralelo, es tanto una necesidad como un desafío.

En esta Tesis Doctoral se proponen nuevos algoritmos altamente paralelizables así como soluciones para implementarlos eficientemente en dispositivos hardware de computo paralelo. Se ha trabajado concretamente en tres procesos del análisis hiperespectral, *desmezclado espectral*, *fusión de imágenes multiespectrales e hiperespectrales* y *compresión de imágenes hiperespectrales y ultraespectrales*, ya que aportar soluciones para estos tres procesos tendrá repercusiones positivas en el análisis hiperespectral en general, y en las aplicaciones que deban funcionar en tiempo real en particular.

En primer lugar se ha desarrollado un algoritmo llamado *Fast algorithm for linearly UNmixing hyperspectral images* (FUN). Este algoritmo lleva a cabo el proceso de desmezclado espectral completo y no requiere una reducción espectral previa. El algoritmo FUN está basado en el concepto de proyecciones ortogonales y emplea operaciones matemáticas sencillas que pueden ser aplicadas de forma independiente para cada pixel de la imagen. Estas características facilitan la obtención de una implementación eficiente del proceso completo de desmezclado espectral en un dispositivo hardware de cómputo paralelo.

Tambíen se ha desarrollado un algoritmo llamado *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI). Este algoritmo permite fusionar imágenes multiespectrales e hiperespectrales con un alto grado de paralelización. El algoritmo CoEf-MHI también está basado en el concepto de proyecciones ortogonales y emplea operaciones matemáticas sencillas, lo que lo convierte en una opción muy buena para aplicaciones que deban funcionar en tiempo real.

Se ha desarrollado también un algoritmo llamado *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA) capaz de llevar a cabo de forma eficiente la decorrelación y reducción espectral de imágenes hiperespectrales y ultraespectrales para su compresión. Este algoritmo también está basado en proyecciones ortogonales y emplea operaciones sencillas. El algoritmo HyperLCA está especialmente diseñado para trabajar de forma independiente con pequeños bloques de la imagen. Todos estos factores lo convierten en una muy buena opción para aplicaciones de compresión donde el tiempo y los recursos hardware disponibles sean limitados.

Finalmente se ha implementado el algoritmo FUN en distintos dispositivos hardware de cómputo paralelo, GPUs y FPGAs, empleando diversos lenguajes, NVIDIA CUDA y OpenCL. Dado que los algoritmos FUN, CoEf-MHI e HyperLCA están todos basados en proyecciones ortogonales y emplean operaciones similares, puede asumirse que los resultados obtenidos en las distintas implementaciones del algoritmo FUN, en términos de eficiencia, podrían ser extrapolables al resto de algoritmos desarrolladas en esta Tesis Doctoral. Las implementaciones desarrolladas verifican que la obtención de resultados en tiempo real es viable si se emplean algoritmos pensados desde su concepción para ser implementados en dispositivos de cómputo paralelo. De forma adicional, en esta Tesis Doctoral se ha verificado la utilidad de OpenCL para el desarrollo de implementaciones eficientes para GPUs y FPGAs.

Los algoritmos e implementaciones desarrollados en esta Tesis Doctoral han sido evaluados y comparados con los trabajos más relevantes hasta el momento en términos de precisión y eficiencia, confirmando la utilidad de las contribuciones realizadas para el análisis de imágenes hiperespectrales en tiempo real.

Acknowledgements

I would like to thank my family and friends who have offered me unconditional aid and support during this long journey. I would like to thank also my supervisors and fellow labmates, for all the stimulating discussions, the fun we have had, the coffees and their patience. I could not have imagined a better work environment for pursuing the PhD.

Contents

Abstract i

Resumen iii

Acknowledgements v

List of Figures xiii

List of Tables xvii

Abbreviations xxi

1 Introduction 1
- 1.1 Outline . . . 2
- 1.2 Preliminary concepts . . . 4
  - 1.2.1 Characterization and resolution of the spectral images . . . 5
  - 1.2.2 Multispectral, hyperspectral and ultraspectral images . . . 6
  - 1.2.3 Targeted domains and motivations of this Thesis . . . 7
    - 1.2.3.1 Hyperspectral unmixing . . . 9
    - 1.2.3.2 Multispectral and hyperspectral images fusion . . . 10
    - 1.2.3.3 Hyperspectral and ultraspectral image compression . . . 11
    - 1.2.3.4 Acceleration - OpenCL implementations . . . 12
- 1.3 Research goals and contributions . . . 14
- 1.4 Organization of this document . . . 18
  - 1.4.1 Chapter 2: Linear hyperspectral unmixing . . . 18
  - 1.4.2 Chapter 3: Multispectral and hyperspectral image fusion . . . 19
  - 1.4.3 Chapter 4: Hyperspectral and ultraspectral image compression . . . 19
  - 1.4.4 Chapter 5: Hyperspectral imaging acceleration through the utilization of high-performance computing platforms . . . 19
  - 1.4.5 Chapter 6: Conclusions . . . 20

2 Hyperspectral unmixing **21**
2.1 Outline . . . 22
2.2 State-of-the-art in linear hyperspectral unmixing . . . 23
2.3 Computational complexity of the FUN algorithm . . . 27
2.4 Estimation of the number of endmembers and their extraction within the FUN algorithm . . . 28
2.4.1 Background: the Modified Gram-Schmidt Orthogonalisation . . . 29
2.4.2 Initialization of the algorithm. Exctraction of the first endmember . 29
2.4.3 Simultaneously estimation of the number of endmembers and their extraction within the FUN algorithm . . . 31
2.4.4 Stopping condition . . . 32
2.5 Abundance estimation within the FUN algorithm . . . 35
2.6 Experimental results . . . 39
2.6.1 Experiments on synthetic hyperspectral data . . . 39
2.6.1.1 Effect of noise filtering in the performance of the FUN algorithm . . . 40
2.6.1.2 Accuracy of the first endmember extracted with the different initialization methods . . . 42
2.6.1.3 Performance of the FUN algorithm for estimation the number of endmembers . . . 43
2.6.1.4 Performance of the FUN algorithm for the endmembers extraction knowing the number of endmembers in advance 44
2.6.1.5 Performance of the FUN algorithm for hyperspectral unmixing . . . 47
2.6.1.6 Performance of the FUN algorithm for abundance estimation 48
2.6.2 Experiments on real hyperspectral data . . . 50
2.7 Conclusions . . . 54

3 Multispectral and hyperspectral image fusion **57**
3.1 Outline . . . 58
3.2 State-of-the-art in multispectral and hyperspectral images fusion . . . 59
3.3 Proposed fusion strategy: the CoEf-MHI algorithm . . . 61
3.3.1 Process applied to the entire image within the CoEf-MHI algorithm 63
3.3.2 Process applied to each individual pixel within the CoEf-MHI algorithm . . . 64
3.3.2.1 Selection of the projection vectors . . . 64
3.3.2.2 Coefficient calculation . . . 66
3.3.2.3 Hyperspectral pixel reconstruction . . . 67
3.3.3 Computational advantages of the proposed CoEf-MHI algorithm . . 67
3.4 Data and evaluation metrics . . . 68
3.4.1 Evaluation method: Wald's protocol . . . 68
3.4.2 Reference hyperspectral data . . . 70
3.4.3 Reference algorithms . . . 72
3.4.4 Metrics used to evaluate the quality of the fused images . . . 72

3.4.5 Metrics used to evaluate the computational complexity and the computational efficiency of the algorithms . . . 75
3.5 Results . . . 76
3.6 Conclusions . . . 80

4 Hyperspectral and ultraspectral image compression 91
4.1 Outline . . . 92
4.2 State-of-the-art in transform-based hyperspectral and ultraspectral compression . . . 94
4.3 Hyperspectral image compression within the HyperLCA algorithm . . . 96
4.3.1 Description of the HyperLCA algorithm . . . 98
4.3.2 Possible stopping conditions within the HyperLCA algorithm . . . 100
4.3.2.1 RMSE based stopping condition. . . . 101
4.3.2.2 SNR based stopping condition. . . . 101
4.3.2.3 MaxSE based stopping condition. . . . 102
4.3.3 Effect of the block partition within the HyperLCA algorithm . . . 102
4.3.4 Block diagram of the compressor . . . 103
4.4 Compression results within the HyperLCA algorithm . . . 105
4.4.1 Hyperspectral and ultraspectral data used for the experiments . . . 106
4.4.2 Evaluation of the performance of the HyperLCA algorithm for spectrally decorrelating and reducing the hyperspectral image. . . . 107
4.4.3 Evaluation of the entire compression-decompression process within the HyperLCA algorithm . . . 110
4.4.4 Ultraspectral image compression results within the HyperLCA algorithm . . . 114
4.5 Evaluation of the compression performance within the HyperLCA algorithm for hyperspectral imaging applications . . . 116
4.6 Computational complexity of the HyperLCA transform . . . 121
4.7 Conclusions . . . 125

5 Hyperspectral imaging acceleration through the utilization of high-performance computing platforms 127
5.1 Outline . . . 129
5.2 State-of-the-art in hyperspectral imaging acceleration through the utilization of high-performance computing platforms . . . 130
5.3 Optimizations applied to the FUN algorithm for a more efficient hardware implementation. pFUN algorithm. . . . 135
5.3.1 Modifications in the endmember extraction process . . . 135
5.3.2 Modifications in the abundance calculation process . . . 138
5.4 Hardware platforms . . . 139
5.4.1 FPGAs description . . . 140
5.4.2 GPUs description . . . 142
5.4.3 FPGAs and GPUs comparison . . . 144
5.5 Targeted devices . . . 145

5.6 Reference hyperspectral data . . . 145
5.7 NVIDIA CUDA GPU implementation of the pFUN algorithm . . . 148
5.7.1 Preliminary Issues . . . 148
5.7.2 CUDA implementation of the pFUN algorithm: extraction of the endmembers . . . 149
5.7.2.1 Data initialization . . . 151
5.7.2.2 Extraction of the first endmember . . . 153
5.7.2.3 Extraction of the rest of endmembers . . . 156
5.7.3 CUDA implementation of the pFUN algorithm: abundances calculation . . . 160
5.7.3.1 Data initialization . . . 160
5.7.3.2 Abundance calculation . . . 161
5.7.4 Experimental results . . . 163
5.7.4.1 Evaluation Metrics . . . 163
5.7.4.2 Performance of the pFUN Algorithm . . . 164
5.8 OpenCL Implementations of the pFUN algorithm . . . 168
5.8.1 OpenCL programing paradigm . . . 168
5.8.2 GPUs implementation of the pFUN algorithm using OpenCL . . . 170
5.8.3 FPGA implementation of the pFUN algorithm using OpenCL . . . 171
5.8.3.1 FPGA endmembers extraction . . . 175
5.8.4 Experimental results . . . 178
5.8.4.1 Results using real hyperspectral images . . . 178
5.8.4.2 Results using synthetic hyperspectral images . . . 180
5.8.4.3 Real-time processing capabilities of the proposed implementations . . . 184
5.9 Conclusions . . . 186

6 Conclusions and further research **189**
6.1 Further research work . . . 195

A Sinopsis en español **197**
A.1 Introducción . . . 198
A.2 Objetivos y metodolgía de trabajo . . . 199
A.3 Motivaciones. . . . 200
A.3.1 Desmezclado espectral . . . 201
A.3.2 Fusión de imágenes hiperespectrales y multiespectrales . . . 203
A.3.3 Compresión de imágenes hiperespectrales y ultraespectrales . . . 204
A.3.4 Aceleración hardware. OpenCL . . . 205
A.4 Contribuciones y conclusiones alcanzadas . . . 206

B Publications **211**
B.1 Journals . . . 212
B.2 International Conferences . . . 212

References **215**

List of Figures

1.1 Spectral data cube. 5
1.2 Comparisons between the spatial and spectral resolutions of hyperspectral and multispectral images. a) Hyperspectral data cube. b) Multispectral data cube. c) Hyperspectral signatures. d) Multispectral signatures. 7
1.3 Electromagnetic spectrum. 8
1.4 Main goals and contributions 17

2.1 Linear and nonlinear mixing models. 24
2.2 Theoretical representation of the values that would be obtained for the stop factor as the number of endmembers extracted by the FUN algorithm increase, from 1 to $p + 1$, being p the real number of endmembers present in the hyperspectral image. 34
2.3 Example of Gram-Schmidt orthogonalization of the endmember e_2 with respect to the endmember e_1; q_2 represents the amount of information in e_2 which is not contained in e_1; u_1 and u_2 are a unitary vectors with the directons of q_1 q_2, respectively. 36
2.4 Example of projection of a pixel in the direction of two endmembers. Vectors q^* and u^* refer to the vectors obtained with the FUN algorithm by orthogonalazing the endmembers using the modified Gram-Schmidt method. Using these vectors, the abundance of each endmember in the pixel can be obtained as shown in a_1, which represents the abundance of the endmember e_1 in this pixel. 36
2.5 Mean values of the stop factor depending on the number of endmembers extracted and the SNR of the hyperspectral image, filtering the noise of the image. Images with 5 endmembers. 41
2.6 Mean values of the stop factor depending on the number of endmembers extracted and he SNR of the hyperspectral image, without filtering the noise of the image. Images with 5 endmembers. 41
2.7 Number of endmembers estimation. Percentage of correct estimations and mean error. a) Image type 1. b) Image type 2. c) Image type 3. 44
2.8 Mean spectral angle obtained with the FUN and the VCA algorithms using a set of different synthetic hyperspectral images, varying the signal to noise ratio, SNR, the number of endmembers in the scene, p, and the abundances distribution type. 46

2.9 a) Number of endmembers estimation. Percentage of correct estimations and mean error obtained using the FUN and the HySIME algorithms. b) Endmembers extraction. Mean spectral angle obtained using the FUN and the VCA algorithms. 49
2.10 *Root Mean Square Error* (RMSE) obtained with the FUN, the FCFUN and the FCLSU algorithms in the simulations performed with a set of synthetic hyperspectral images with different characteristics of noise and with different number of endmembers. 51
2.11 AVIRIS Cuprite image. 51
2.12 a) False colour representation of the AVIRIS Cuprite image used. b) RSE obtained with the FUN algorithm. c) RSE obtained with the FC-FUN algorithm. d) RSE obtained with the FCLSU algorithm. 54
2.13 Comparison of the five spectral signatures in the Cuprite scene (in blue) with respect to the endmembers extracted by the FUN (in green) and the VCA (in red) algorithms. 55

3.1 Images involved in the fusion process using the CoEf-MHI algorithm 62
3.2 Flow diagram of the experimental methodology, derived from Wald's protocol. 69
3.3 RGB representation of the different hyperspectral images used for the simulations. 71
3.4 Mean SAM (dashed bars) and mean PSNR (filled bars) values for different images. Lower SAM values and higher PSNR values indicate better fusion results. 77
3.5 Mean FSAM (dashed bars) and mean FPSNR (filled bars) values for different images. Higher FSAM and FPSNR values indicate a higher efficiency in the fusion process. 78
3.6 Mean MSAM (dashed bars) and mean MPSNR (filled bars) values for different images. Higher MSAM and MPSNR values indicate a higher efficiency in the fusion process. 78
3.7 RGB representation of the different hyperspectral images obtained in the fusion process. RHS image, LHS image, and the fused HHS images, obtained with the BN algorithm and the CoEf-MHI algorithm respectively. The LHS images have been obtained according to the two first test cases shown in Table 3.1 79

4.1 Remote sensing scanners. a) WhiskBroom b) Pushbroom 104
4.2 Block diagram of the compressor. 105
4.3 False color representation of the spectral images described in Table 4.1. . . 108
4.4 Spectral reduction performance with the PCA and HyperLCA transforms. a), b) Average SNR and MaxSE values. c), d) Best SNR and MaxSE values. e), f) Worst SNR and MaxSE values. 109
4.5 Average compression performance with the PCA and HyperLCA transforms for radiance and uncalibrated AVIRIS Yellowstone hyperspectral images. 110

4.6 Hyperspectral compression performance with the PCA+CCSDS122 and HyperLCA+CCSDS121 compressors. a), b) Average SNR and MaxSE values. c), d) Best SNR and MaxSE values. e), f) Worst SNR and MaxSE values. . . . 112
4.7 Improvement in the compression performance of the HyperLCA+CCSDS121 approach when compressing both the spectrally reduced hyperspectral image and the projection vectors within the CCSDS121 coder. . . . 113
4.8 Spectral reduction of ultraspectral images using the HyperLCA and the PCA transforms. . . . 115
4.9 Effect of the compression-decompression process within the HyperLCA+CCSDS121 and PCA+CCSDS122 compressors on anomaly detection using the RX detector and the Rochester Institute image. a) False color image of the data set. b) Ground truth. c) Result provided by the RX detector using the real data. d-m) Results provided by the RX detector using the compressed-decompressed data. . . . 118
4.10 Effect of the compression-decompression process within the HyperLCA+CCSDS121 and PCA+CCSDS122 compressors on anomaly detection using the RX detector and the World Trade Center image. a) False color image of the data set. b) Ground truth. c) Result provided by the RX detector using the real data. d-m) Results provided by the RX detector using the compressed-decompressed data. . . . 119

5.1 FPGA resources and interconnects. . . . 141
5.2 FPGA parallelism description. Pipeline . . . 141
5.3 Division of the pipeline structure in compute units. . . . 141
5.4 Thread level parallelism achieved in the GPUs. . . . 144
5.5 RGB representation of the different real hyperspectral images used for the simulations. . . . 146
5.6 Several types of launching kernels. . . . 150
5.7 CUDA implementation of the extraction of endmembers of the pFUN Algorithm. . . . 152
5.8 cuBLAS implementation of the orthogonalization in the first endmember extraction. . . . 155
5.9 Representation of the kernel to calculate $d_X \circ d_X$. . . . 156
5.10 Representation of the CUDA implementation of lines 7 and 10 of the algorithm in Figure 5.7. . . . 157
5.11 cuBLAS implementation for the orthogonalization to extract all the endmembers except the first one. . . . 158
5.12 CUDA implementation of the abundance calculation of the pFUN Algorithm 161
5.13 pFUN execution on a GeForce GTX 680 with different parameters for different synthetic images. . . . 165
5.14 Diagram of the OpenCL programing paradigm . . . 169
5.15 Structure of the pFUN algorithm implemented in the Bitware Stratix V FPGA Altera board using OpenCL. . . . 173

5.16 Endmembers extraction process implemented in the Bitware Stratix V FPGA Altera board. 177
5.17 Comparison of the time consumed for unmixing different real hyperspectral images, using the Bitware Stratix V FPGA Altera board and the GeForce GTX 950 and Tesla K40c NVIDIA GPUs, programmed using OpenCL and NVIDIA CUDA. 179
5.18 Comparison of the time consumed for unmixing different synthetic images, using the Bitware Stratix V FPGA Altera board and the GeForce GTX 950 and Tesla K40c NVIDIA GPUs, programmed using OpenCL and NVIDIA CUDA. 181
5.19 Percentage of time consumed in different parts of the process in the different devices and OpenCL implementations of the pFUN algorithm 182
5.20 Comparison of the time required for processing real hyperspectral images using the different implementations of the pFUN algorithm and the time that would be required by the NG-AVIRIS sensor for collecting images with the same amount of data. 185
5.21 Comparison of the time required for processing different synthetic hyperspectral images, with 20 endmembers each, using the pFUN algorithm, and the time that would be required by the NG-AVIRIS sensor for collecting images with the same amount of data. 186

A.1 Principales objetivos y contribuciones 208

List of Tables

2.1 Mean spectral angle obtained for the first endmember extracted using each of the three initialization methods proposed . . . 43
2.2 Mean spectral angle obtained with the FUN and the VCA algorithms . . . 45
2.3 Average results obtained for hyperspectral unmixing . . . 48
2.4 RMSE values obtained with the FUN, the FC-FUN and the FCLSU algorithms . . . 50
2.5 Comparison of the results obtained with the Cuprite dataset . . . 52
2.6 Mean results obtained with the Cuprite dataset . . . 53
2.7 Comparison of the spectral signatures obtained with the FUN algorithm and the VCA algorithm . . . 53
2.8 RMSE obtained with the FUN algorithm, with the FCFUN algorithm and with the FCLSU algorithm . . . 53

3.1 Spatial degradation methods for each RHS image . . . 72
3.2 Salinas. Test case 1 . . . 82
3.3 Salinas. Test case 2 . . . 82
3.4 Salinas. Test case 3 . . . 82
3.5 Salinas. Test case 4 . . . 82
3.6 Salinas. Test case 5 . . . 83
3.7 Salinas. Test case 6 . . . 83
3.8 Indian Pines. Test case 1 . . . 83
3.9 Indian Pines. Test case 2 . . . 83
3.10 Indian Pines. Test case 3 . . . 84
3.11 Indian Pines. Test case 4 . . . 84
3.12 Indian Pines. Test case 5 . . . 84
3.13 Indian Pines. Test case 6 . . . 84
3.14 Cuprite. Test case 1 . . . 85
3.15 Cuprite. Test case 2 . . . 85
3.16 Cuprite. Test case 3 . . . 85
3.17 Cuprite. Test case 4 . . . 85
3.18 Cuprite. Test case 5 . . . 86
3.19 Cuprite. Test case 6 . . . 86
3.20 Washington DC. Test case 1 . . . 86
3.21 Washington DC. Test case 2 . . . 86
3.22 Washington DC. Test case 3 . . . 87

3.23 Washington DC. Test case 4 . . . 87
3.24 Washington DC. Test case 5 . . . 87
3.25 Washington DC. Test case 6 . . . 87
3.26 Paris. Test case 1 . . . 88
3.27 Paris. Test case 2 . . . 88
3.28 Paris. Test case 3 . . . 88
3.29 Paris. Test case 4 . . . 88
3.30 Paris. Test case 5 . . . 89
3.31 Paris. Test case 6 . . . 89

4.1 Characteristics of the images used for the experiments . . . 107
4.2 Evaluation of the SNR between the real pixels labeled as anomalous pixels in the ground truth and their compressed-decompressed version using different approaches. . . . 119
4.3 Evaluation of the SNR between the anomalies matrix obtained with the RX detector using the real images and their compressed-decompressed versions provided by the different approaches. . . . 120
4.4 Number of flops required for achieving the different steps of the PCA and HyperLCA transforms for spectrally reducing a hyperspectral image. . . . 121
4.5 Number of FLOPs (in logarithmic scale) performed by the PCA and HyperLCA transforms for spectrally decorrelating and reducing hyperspectral images, according to the compression ratio achieved and the size of the image. 123
4.6 Number of FLOPs (in logarithmic scale) performed by the HyperLCA transforms for spectrally decorrelating and reducing hyperspectral images according to the block size used. . . . 123
4.7 Approximated number of FLOPs required for performing the spectral decorrelation of a hyperspectral image using the POT and the Wavelet CDF 9/7 transforms. . . . 124
4.8 Number of FLOPs (in logarithmic scale) performed by the POT and Wavelet CDF 9/7 transforms for spectrally decorrelatinghyperspectral images, according to the size of the image. . . . 125

5.1 FPGA implementations of unmixing algorithms . . . 132
5.2 GPU implementations of unmixing algorithms . . . 132
5.3 GPUs features . . . 145
5.4 FPGA features . . . 145
5.5 Real hyperspectral images used for the experiments . . . 147
5.6 Speedup for the parallel pFUN implementation in relation with two unmixing chains . . . 166
5.7 Speedup for the parallel pFUN implementation in relation with a new unmixing chain . . . 167
5.8 Evaluation metrics for the unmixing chains and the parallel pFUN implementation . . . 168
5.9 Specifications of the AVIRIS and NG-AVIRIS sensors . . . 184
5.10 Time required for collecting the different real hyperspectral images used . . . 185

5.11 Time required for collecting the different synthetic hyperspectral images used185

Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
ACOEE	Ant Colony Optimization algorithm for Endmember Extraction
AIRS	Atmospheric Infrared Sounder
ALI	Advanced Land Imager
ALM	Adaptive Logic Module
ALUT	Adaptive Look-Up Table
AMD	Advanced Micro Devices
AMEE	Automatic Morphological Endmember Extraction
ANC	Abundance Non-negativity Constraint
API	Application Programming Interface
ASC	Abundance Sum-to-one Constraint
AVIRIS	Airbone Visible/Infrared Imaging Spectrometer
BLAS	Basic Linear Algebra Subprograms
BN	Bayesian Naive
bpppb	Bits Per Pixel Per Band

BS	Bayesian Sparse
CC	Cross Correlation
CCSDS	Consultative Committee for Space Data Systems
CNMF	Coupled Nonnegative Matrix Factorization
CoEf-MHI	Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images
CPU	Central Processing Unit
CS	Component Substitution
CSNMF	Constrained Sparse Nonnegative Matrix Factorization
CUDA	Computer Unified Device Architecture
DDR	Double Data Rate
DSP	Digital Signal Processing
DWT	Discrete Wavelet Transform
ERGAS	Erreur Relative Globale Adimensionnelle de Synthese
ESA	European Space Agency
FCLSU	Fully Constrained Least Squares linear spectral Unmixing
FLOP	Floating-Point Operation
FPGA	Field-Programmable Gate Array
FUN	Fast algorithm for linearly UNmixing hyperspectral images
GEMM	General Matrix-Matrix multiplication
GGSA	Generalized Gram-Schmidt Adaptive
GLMMSE	Generalized Local Minimum Mean Square Error

GPU	Graphics Processing Unit
GSA	Gram-Schmidt Adaptive
HDL	Hardware Description Language
HHS	High resolution HyperSpectral
HLS	High-Level Synthesis
HMS	High resolution MultiSpectral
HYDICE	Hyperspectral Digital Imagery Collection Experiment
HyperLCA	Lossy Compression Algorithm for Hyperspectral image systems
HySIME	Hyperspectral Subspace Identification by Minimum Error
HySure	Hyperspectral Superresolution
IASI	Infrared Atmospheric Sounding Interferometer
ICA	Independent Component Analysis
ICE	Iterative-Constrained Endmembers
Ice	Internet Communications Engine
IEA	Iterative Error Analysis
ISRA	Image Space Reconstruction Algorithm
JPEG	Joint Photographic Experts Group
KLT	Kahrunen-Loève Transform
KPPI	Kernel version of the Pixel Purity Index
LHS	Low resolution HyperSpectral
LMMSE	Local Minimum Mean Square Error
LSU	Least Squares linear spectral Unmixing

LWIR Long Wavelength InfraRed

MaxSDE Maximum Scaled absolute Difference Error

MaxSE Maximum Single Error

MEM total allocated MEMory

MNF Maximum Noise Fraction

MPPA Massively Parallel Processor Array

MRA Multiresolution Analysis

MSE Mean Square Error

MVC-NMF Nonnegative Matrix Factorization based on Minimum Volume Constraint

MVCA Modified Vertex Component Analysis

MVSA Minimum Volume Simplex Analysis

MWIR Medium Wavelength InfraRed

NASA National Aeronautics and Space Administration

NG-AVIRIS Next Generation Airbone Visible/Infrared Imaging Spectrometer

NIR Near InfraRed

NMF Nonnegative Matrix Factorization

NRMSE Normalized Root Mean Square Error

OP Objective Pixels

OpenCL Open Computing Language

OSP Orthogonal Subspace Projection

OSP-GS Orthogonal Subspace Projection with Gram-Schmidt orthogonalization

PCA	Principal Component Analysis
PCI	Peripheral Component Interconnect
PCIe	PCI express
PCLSU	Partially Constrained Least Squares linear spectral Unmixing
PE	Processing Element
pFUN	parallel Fast algorithm for linearly UNmixing hyperspectral images
PLL	Phase-Locked Loop
POT	Pairwise Orthogonal Transform
PPI	Pixel Purity Index
PSNR	Peak Signal-to-Noise Ratio
RGB	Red, Green, Blue
RHS	Reference HyperSpectral
RIBSR	Ratio Image-Based Spectral Resampling
RMSE	Root Mean Square Error
RPC	Remote Procedure Call
RSE	Root Square Error
RTL	Register-Transfer Level
RT-MVSA	Real-Time Maximum Simplex Volume Algorithm
RX	Reed-Xiaoli
SA	Spectral Angle
SAM	Spectral Angle Mapper
SISAL	Simplex Identification via Variable Splitting and Augmented Lagrangian

SNR Signal-to-Noise Ratio

SP Surrounding Pixels

SPP Spatial Preprocessing

SSPP Spatial–Spectral Preprocessing

SVD Singular Value Decomposition

SUMSAL Spectral UnMixing by Splitting and Augmented Lagrangian

LWIR Long Wavelength Infrared

SWIR Short Wavelength Infrared

TIR Thermal InfraRed

UCLSU Unconstrained Least Squares linear spectral Unmixing

UHS Unmixed HyperSpectral

ULHS Upsampled HyperSpectral

USGS U.S. Geological Survey

USMS Simulated MultiSpectral

VCA Vertex Component Analysis

VD Virtual Dimensionality

Verilog HDL Verilog Hardware Description Language

VHDL VHSIC Hardware Description Language

VNIR Visible and Near InfraRed

VSWIR Visible Shortwave InfraRed

WASP Wildfire Airborne Sensor Program

Chapter 1

Introduction

The main objective of this Chapter is to present the motivation for this Thesis, whose major contributions are in the field of algorithms development for hyperspectral image analysis under real-time constraints. The basics about hyperspectral imaging are introduced by briefly explaining the notion of hyperspectral images, how they are collected and processed, and their applications. Furthermore, the necessity for developing new algorithms, thought from their conception to be implemented in parallel computing devices, for achieving real-time performance is highlighted. Finally, the research goals and contributions of this Thesis as well as the organization of this document are presented.

1.1 Outline

Hyperspectral imaging systems are considered nowadays one of the most powerful tools for acquiring surface information in many different fields [1–4]. These systems are able to provide images in which single pixels have information from across the electromagnetic spectrum of the scene under observation.

Whereas the human eye sees color of visible light in mostly three bands (red, green, and blue), hyperspectral imaging divides the spectrum into many more contiguous bands, which may be even out of the visible part of the electromagnetic spectrum. The main advantage of this technique is that certain objects leave unique 'fingerprints' in the electromagnetic spectrum, known as spectral signatures. These 'fingerprints' enable identification of the materials that make up a scanned object, providing a much better understanding of the scene under analysis.

Remote sensing is the field in which hyperspectral imaging techniques have traditionally acquired more relevance. In this field, hyperspectral sensors are frequently launched onboard satellites or airplanes, for collecting spectral information of the Earth's surface or the different layers of the atmosphere. Some examples of these sensors are the *Airbone Visible/Infrared Imaging Spectrometer* (AVIRIS) [5], the *Atmospheric Infrared Sounder* (AIRS) [6] and the *Infrared Atmospheric Sounding Interferometer* (IASI) [7] sensors. The AVIRIS sensor collects data that can be used for characterization of the Earth's surface and atmosphere from geometrically coherent spectroradiometric measurements. These data can be applied to studies in the fields of oceanography, environmental science, snow hydrology, geology, volcanology, soil and land management, atmospheric and aerosol studies, agriculture, and limnology. Applications under development include the assessment and monitoring of environmental hazards such as toxic waste, oil spills, and land/air/water pollution. The information collected by the AIRS sensor can be used for creating three-dimensional maps of the temperature of the atmosphere and of the earth surface, as well as for registering data related with the amount of clouds and water in the atmosphere. The IASI sensor is specially designed for weather forecasting. The information provided by this sensor can be use for accurately measuring the temperature of the troposphere and low stratosphere, as well as for accurately measuring the humidity in the troposphere. This sensor provides information about the emissivity of the earth surface and the temperature of the see surface when there are not clouds. The spectral range of the IASI sensor has been chosen so that it can identify the carbon dioxide, ozone,

water vapour and methane present in the atmosphere, according to the electromagnetic radiation absorbed by these components.

On the other hand, many laboratory and industrial applications for hyperspectral imaging have been gaining importance during the last years. Those applications are related with food processing, health care, mineralogy and surveillance, among others [8–10]. One of the most interesting applications is the identification of tumor tissues in different parts of the human body, since the hyperspectral images have been proven to provide information which allows to distinguish between the tumor and healthy tissue without the necessity of using more invasive techniques [11–15].

Many different algorithms able to produce accurate results have been developed for providing solutions to all those applications which make use of hyperspectral images as a source of information, which have increased the interest in this field. The algorithms applied for these applications require to process the huge amount of data provided by the hyperspectral sensors, which typically results in a high computational burden [16–21]. Hyperspectral imaging applications under real-time constraints require to rapidly execute those algorithms. The state-of-the-art solution consists on implementing the hyperspectral analysis algorithms into devices that are suitable for parallel computing, such as *Graphic Processing Units* (GPUs) and *Field Programmable Gate Arrays* (FPGAs). In fact, many works have been uncovered during the last years for implementing different algorithms for processing hyperspectral images into parallel hardware, with the purpose of speeding up the hyperspectral analysis process [22–28].

However, the algorithms for processing hyperspectral images have been typically developed with the purpose of producing the best hyperspectral analysis results, with independence of their mathematical complexity, and without having in consideration how these algorithms could be implemented into parallel computing platforms for providing results in real time. In example, many hyperspectral imaging algorithms use matrix operations which are very complex for large data sets, such as factorizations, the inverse of a matrix or the eigenvalues and eigenvectors calculation. Hence, when these algorithms are to be implemented into parallel devices for providing faster results, there is little room for speeding them up, due to their mathematical complexity as well as their so many data dependencies. As a result, many efforts have also been done during the last years for trying to modify or better adapt some hyperspectral image processing algorithms for being implemented into parallel computing devices in a easier and more efficient way [21, 29–31]. However, these approaches for speeding up the hyperspectral analysis are always

limited by the inherent complexity of the algorithms and typically result in decreasing their accuracy.

This is why the development of more efficient hardware-friendly algorithms for processing hyperspectral images, which can be efficiently implemented in parallel hardware devices for speeding up the process is an important and exciting challenge. These algorithms need to be thought from their conception for being implemented into parallel computing platforms. In this Thesis a new approach for hyperspectral imaging analysis applications under real-time constraints has been developed. This includes developing new parallel hardware-friendly hyperspectral imaging algorithms for applications under real-time constraints, as well as finding an efficient way of implementing the developed algorithms across different platforms.

In particular, a set of parallel hardware-friendly algorithms for processing hyperspectral images have been developed in this Thesis. Each of the developed algorithms target a different hyperspectral processing domain, namely (i) hyperspectral unmixing; (ii) hyperspectral and multispectral images fusion; and (iii) hyperspectral and ultraspectral image compression. As a further step, the developed algorithms have been implemented in different parallel hardware devices, *Graphics Processing Units* (GPUs) and Field Programmable Gate Arrays (FPGAs), using different frameworks, such as NVIDIA *Compute Unified Device Architecture* (CUDA) and *Open Computing Language* (OpenCL). The results obtained in these implementations verify the goodness of the developed algorithms for being implemented in parallel devices for providing real-time results, verifying as well the suitability of the proposed methodology for developing and implementing algorithms for hyperspectral imaging analysis applications under real-time constraints. The results obtained also provide very interesting comparisons about the different hardware description languages used as well as the targeted hardware devices (GPUs and FPGAs), according to the characteristic of the different applications.

1.2 Preliminary concepts

The following sections introduce the reader to the main topics that will be repeatedly mentioned throughout this document, including an explanation of the different types of resolution which are used to characterized spectral images, a definition of multispectral, hyperspectral and ultraspectral images, and a brief explanation of the hyperspectral unmixing process, the multispectral and hyperspectral image fusion and the remote sensed

hyperspectral and ultraspectral images compression, which are the objective processes of this Thesis.

1.2.1 Characterization and resolution of the spectral images

Spectral imaging sensors are designed to measure the reflectance of areas on the surface of the sensed scenes. Reflectance is defined as the percentage of the light hitting a material that is then reflected by that material, and can be represented across a range of wavelengths, showing a pattern or spectrum, which can uniquely identify certain materials. In spectral imaging, a sensor acquires a spectral vector in different wavelengths of every pixel in a given scene. This results in a three-dimensional spectral image, which is typically depicted as a data cube with the spatial information represented in the X-Y plane, and spectral information represented in the Z-direction as shown in Figure 1.1, where N_r, N_c and N_b are the number of rows, columns and bands, respectively.

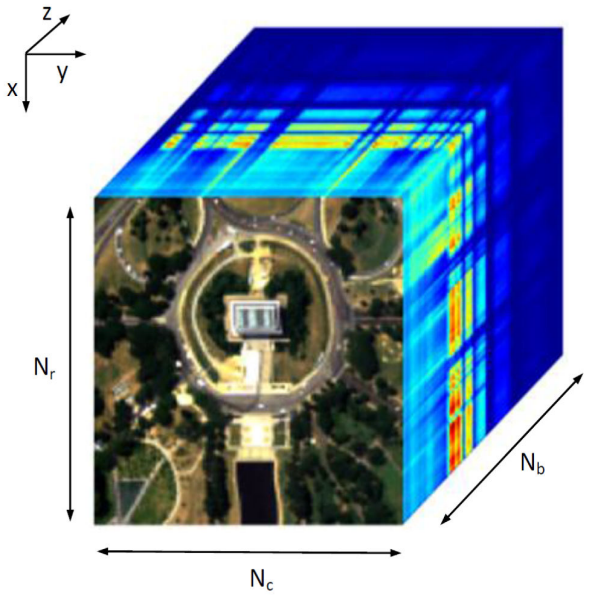


FIGURE 1.1: Spectral data cube.

Spectral images collect information in the spatial and spectral domain, and so, different resolution measurements are needed to characterize the acquired images. The spatial resolution of the spectral images refers to the size of each pixel in the X-Y plane, while the spectral resolution refers to the thickness of wavelength range of the spectral bands. A higher spatial and spectral resolution indicates a smaller size of the pixel in the X-Y plane and a smaller wavelength width for each spectral band, respectively. Since all this

information is collected by a digital sensor, one extra resolution measurement is needed, named radiometric resolution. The radiometric resolution of the sensor corresponds with its sensitivity to the magnitude of the electromagnetic energy. The radiometric resolution of an imaging system describes its ability to discriminate very slight differences in energy and determines the number of bits required for storing the energy sensed in each spectral band of each pixel. The higher radiometric resolution of the sensor, the more sensitive it is to detecting small differences in the reflected energy and the more bits are required.

1.2.2 Multispectral, hyperspectral and ultraspectral images

While the radiometric resolution of the spectral imaging sensors is typically set to 8 to 16 bits per pixel and per band, the spatial and spectral resolution, as well as the number of spectral bands, widely vary depending on the sensor and the target application. Spectral images are typically classified according to their spectral resolution and number of spectral bands as multispectral, hyperspectral and ultraspectral images.

There is no agreed criterion on how to make the difference between hyperspectral and multispectral images. The literature includes definitions in terms of the number of bands, their narrowness or if the bands are contiguous or there are gaps between them. In this work we define hyperspectral imaging systems as those which collect at least 100 spectral bands of less than 20 nm width. Multispectral sensors, on the other hand, can be defined as those collecting less than 20 non contiguous bands [32, 33]. Hyperspectral imagers produce a detailed, seemingly continuous spectrum that can provide much more information than a multispectral pixel spectrum. Hence, hyperpsectral systems have a very wide capability of spectral discrimination, while multispectral systems provide bands that detect information in specific combinations of desirable regions of the spectrum. Although the spatial resolution is not considered for making a distinction between hyperspectral and multispectral images, the increment in the number of bands of the spectral images is typically linked to a reduction of their spatial resolution, and hence, multispectral systems typically provide higher spatial resolutions than hyperspectral ones. Figure 1.2 graphically shows a comparison of the aforementioned details about hyperspectral and multispectral images.

Finally, for certain applications such as weather forecasting the sensors are design to collect spectral images with a really high spectral resolution and number of bands, but

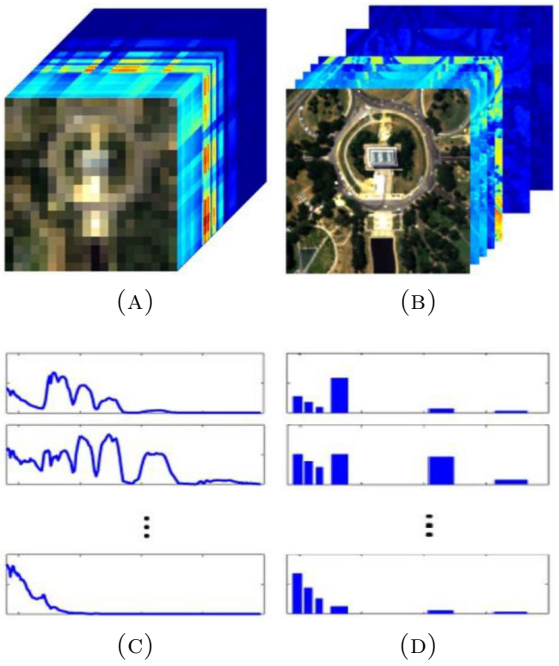


FIGURE 1.2: Comparisons between the spatial and spectral resolutions of hyperspectral and multispectral images. a) Hyperspectral data cube. b) Multispectral data cube. c) Hyperspectral signatures. d) Multispectral signatures.

with a lower spatial resolution. These systems, which typically collect thousands of bands, are classified as ultraspectral imaging systems.

Most spectral imaging systems work in a wavelength range from the visible to the infrared, the latter commonly divided in bands called *Near InfraRed* (NIR), λ: 0.7 - 1.1 μm; *Short Wavelength InfraRed* (SWIR), λ: 1.1 - 3.0 μm; *Medium Wavelength InfraRed* (MWIR), λ: 3.0 - 5.5 μm; and *Long Wavelength InfraRed* (LWIR), λ: 7.7 - 14 μm. Figure 1.3 shows the electromagnetic spectrum.

1.2.3 Targeted domains and motivations of this Thesis

Hyperspectral imaging have many different applications, and each single application requires to perform particular and different processes. For example, in applications for distinguishing tumors from the healthy tissue in different parts of the human body, a hyperspectral classification algorithm is probably required. However, the classification algorithm may be a different one for different parts of the human body, it may use different training sets, images with different spectral and spatial resolutions and collected in different spectral bands. Moreover, the images may be pre-processed according to

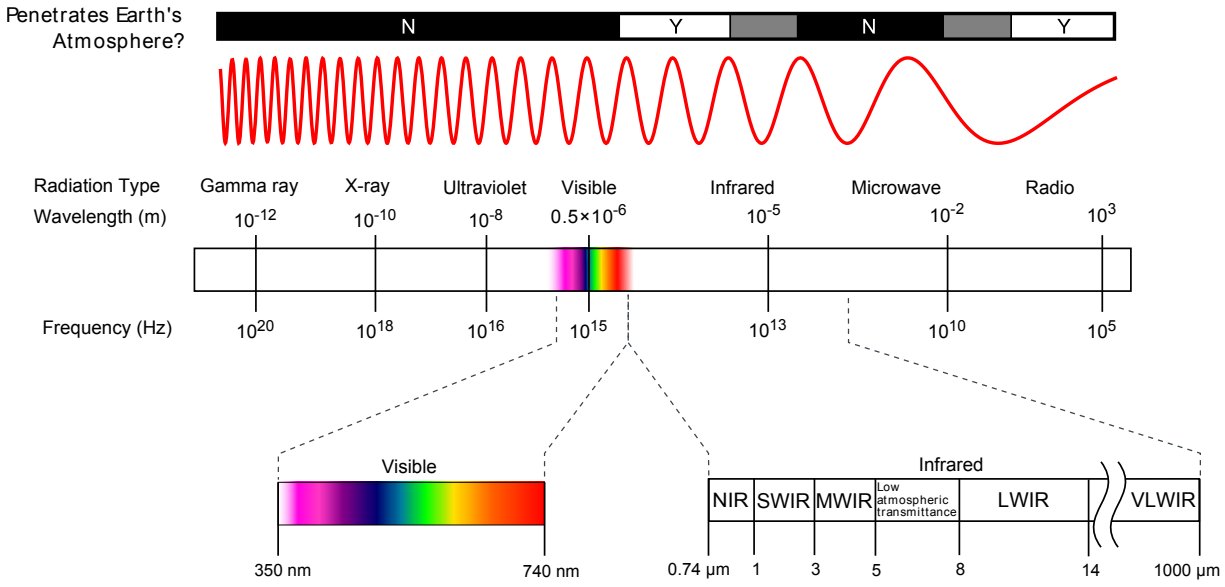


FIGURE 1.3: Electromagnetic spectrum.

the necessities of each particular applications, using different algorithms for filtering the noise, correcting the brightness, performing an spectral or spatial reduction of the dataset, selecting the most relevant bands for the applications, etc.

In this Thesis three processes which are widely used in hyperspectral imaging analysis [34] have been targeted, since simplifying and speeding up these processes will have repercussions in hyperspectral image processing in general, and more specifically, in those applications under real-time or near real-time constrains. The targeted processes are hyperspectral linear unmixing, hyperspectral and multispectral image fusion, and hyperspectral and ultraspectral image compression.

Furthermore, when the algorithms used for processing the hyperspectral data are parallelizable, a high speed up in the process can be achieved by implementing them into parallel hardware devices, such as *Graphic Processing Units* (GPUs) or *Field Programmable Gate Arrays* (FPGAs). The implementation of the developed solutions for the targeted hyperspectral domains into different hardware devices and architectures has also been studied in this Thesis, since developing hardware-friendly algorithms and finding an efficient way of implementing them into parallel computing devices for rapidly processing hyperspectral images will have positive repercussions on hyperspectral imaging applications under real-time or near real-time constraints.

1.2.3.1 Hyperspectral unmixing

Hyperspectral sensors collect the reflectance of the materials in each pixel of the sensed scene, representing them across a range of wavelengths, showing a pattern or spectrum which is typically referred as the spectral signature of the pixel. This spectral signature can be used for uniquely identifying certain materials. However, it is common that different materials coexist at a subpixel level within the image. This results in pixels which spectral signature does not directly correspond with any of the spectral signatures of the materials which conform that particular pixel, but with a mix of them. The hyperspectral unmixing process tries to identify the different materials, also called endmembers, present in the image by identifying their pure signatures. The unmixing process evaluates also the amount of each endmember in each pixel of the image [16–18].

Linear hyperspectral unmixing is based on the idea that each capured pixel in the hyperspectral image can be represented as a linear combination of a set of spectrally pure constituent spectra or endmembers, weighted by an abundance factor that establishes the proportion of each endmember in the pixel under inspection. This linear mixture model assumes that secondary reflections and scattering effects can be neglected from the data collection procedure, and hence, the measured spectra can be expressed as a linear combination of the spectral signatures of materials present in the mixed pixel. In those cases in which the impact of the secondary reflections or the scattering effects is relevant, it is necessary to resort to more complex non-linear models, which demand a priori information about the geometry and physical properties of the observed objects, which results in a further increase of the computational complexity of the unmixing process [35].

The linear hyperspectral unmixing process could be a preliminary step for many other hyperspectral processes, such as resolution enhancement, image fusion, classification, anomaly detection, etc. since it allows a better understanding of the scene under analysis. Hyperspectral unmixing has a higher impact in remote sensing since many remote hyperspectral imaging platforms have low spatial resolution ($> 5m$ per pixel) causing each pixel to be a mixture of several materials.

Although many different contributions have been uncovered during the last two decades, the majority of them are based on dividing the whole process of linearly unmixing a given hyperspectral image into three sequential steps: (i) estimation of the number of endmembers that are present in the hyperspectral image under consideration; (ii) extraction of these endmembers from the hyperspectral dataset ; and (iii) calculation of the abundances

associated with the endmembers induced in the previous step per each mixed pixel of the image. Although this *de facto* processing chain has proven to be accurate enough for unmixing most of the images collected by hyperspectral remote sensors, it is also true that it is not exempt of drawbacks, such as the fact that all the possible combinations of algorithms in order to fully unmix a hyperspectral image according to the aforementioned processing chain demand a formidable computational effort, which tends to be higher as better is the performance of the designed unmixing chain. This troublesome issue unfortunately prevents the use of hyperspectral imaging technology in applications under real-time constraints, in which hyperspectral images have to be analysed in a short portion of time. Hence, there is a clear need to face the challenge of fully exploiting the unquestionable benefits of the hyperspectral imaging technology for these applications, but concurrently overcoming the limitations imposed by the computationally complex nature of the processes involved.

1.2.3.2 Multispectral and hyperspectral images fusion

Remote spectral sensors collect data by measuring the energy reflected from the Earth's surface. Thanks to this process they are able to provide final users with images characterized by a different spatial resolution, spectral resolution and *Signal-to-Noise Ratio* (SNR). According to their spectral resolution, these images can be classified as multispectral, hyperspectral or even ultraspectral images. The spectral resolution of these different types of data varies from some spectral bands for the multispectral images, typically close to the visible spectral range, to hundreds or thousands of bands for the hyperspectral and ultraspectral images, which normally cover the visible and infrared ranges. However, the spatial resolution of multispectral remote sensing systems is typically much higher than the spatial resolution of the hyperspectral and ultraspectral remote sensing systems.

Hence, the main advantage of hyperspectral images with respect to multispectral ones is the more accurate spectral information they provide, which clearly benefits many applications such as unmixing, classification and/or target detection [19, 34]. Nevertheless, a potential improvement of the spatial resolution of hyperspectral images certainly benefits the results achieved for these applications. With this objective in mind, and due to the existence of remote sensing systems with both hyperspectral and multispectral sensors, the multispectral and hyperspectral image fusion process has become a popular task in the last decade. This process uses fusion algorithms to obtain remotely sensed images

with enriched spatial and spectral resolutions by wisely combining the data acquired for the same scene by multispectral and hyperspectral sensors.

The different fusion algorithms can be classified into *Component Substitution* methods (CS), *Multiresolution Analysis* (MRA), and more recent and complex methods based on Bayesian and matrix factorization approaches. CS-based fusion techniques typically provide a high fidelity in rendering the spatial details in the final image, a relative fast and easy implementation and a robustness to misregistration errors and aliasing. However, CS-based methods tend to generate a significant spectral distortion caused by the spectral mismatch between the hyperspectral and the multispectral image. MRA-based fusion methods typically provide spectral consistency and robustness to aliasing under proper conditions. However, the computational burden of the MRA-based methods is usually larger and their implementation is more complicated due to the design of spatial filters. Finally, Bayesian based methods and matrix factorization approaches for fusing multispectral and hyperspectral images provide more accurate results in exchange of an increase of their complexity and computational burden. In this scenario, the development of a more efficient fusion algorithm, understanding the efficiency of the algorithm as the relation between the quality of its fusion results and its computational complexity, would represent an important advantage for hyperspectral imaging applications under real-time constraints.

1.2.3.3 Hyperspectral and ultraspectral image compression

Similarly to RGB image compression, hyperspectral and ultraspectral image compression consists on reducing the amount of data to be send or stored, obtaining a compressed image, in such a way that the entire hyperspectral or ultraspectral image can be recovered from the compressed one, with or without losing information, according to the compressing method used, which can be lossy or lossless. However, while in RGB images the information is just spatially compressed (2D compression), in hyperspectral and ultraspectral images the information can be also spectrally reduced (3D compression), what makes possible to get higher compression ratios.

The hyperspectral and ultraspectral image compression process becomes specially important in remote sensing applications in which the data is acquired on a satellite, where the amount of storage and downlink bandwidth are limited. In this scenario, being able to rapidly compress the sensed images, obtaining high compression ratios without losing too

much information represents an important advantage. However this is a very challenging task, since the algorithms used have to meet specific requirements, as a consequence of the limited on-board computational power available on a satellite and the special operating conditions of the hardware in the space environment [36].

The increment in the data-rate of the new generations of hyperspectral and ultraspectral sensors is making more critical the necessity of obtaining higher compression ratios, making it necessary to use near-lossless and/or lossy compression techniques. Lossy compression approaches for hyperspectral images are typically evaluated in terms of the rate-distortion achieved. However, it has been proven that low average distortions in the compression-decompression process do not necessarily ensure good results when used in specific hyperspectral applications [37]. In general, lossy compression can produce a low-pass filter, which may produce a reduction of the noise present in the image, positively affecting the results of some hyperspectral applications [38]; but it can also destroy the most atypical elements of the image, which are crucial for several applications, such as anomaly detection, classification, unmixing, or target detection [39–42]. In this scenario, the development of new lossy compression techniques for hyperspectral and ultraspectral data, specifically developed for not negatively affecting the subsequent imaging applications results, which can be efficiently executed on-board satellites, would provide important benefits for remote sensing applications under real-time constraints.

1.2.3.4 Acceleration - OpenCL implementations

Hyperspectral imaging applications under real-time constraints require to rapidly process a huge amount of data. When the algorithms used for processing that information are parallelizable, parallel hardware devices, such as *Graphic Processing Units* (GPUs) or *Field Programmable Gate Arrays* (FPGAs), can be used for speeding up the process. There are different possible approaches for developing the implementations of the targeted algorithms into the desired parallel device.

First of all, the developing flow of parallel hardware applications widely varies according to the destination hardware device. On one side, the targeted algorithm can be directly programmed in some parallel devices, such as GPUs [43], using specific programming platforms and languages suitable for the considered device, such as NVIDIA CUDA for NVIDIA GPUs [44], or using a more generic computing language such as *Open Computing Language* (OpenCL).

On the other hand, when the targeted algorithm is to be implemented on a FPGA, *Hardware Description Languages* (HDLs) and many different development tools can be used [45]. The HDLs may differ in their level of abstraction. The commonly used levels of abstraction are gate level, *register-transfer level* (RTL), and algorithmic level. While manually going through each of these levels may provide accurate efficient hardware designs, the more automatic processes have gained traction during the last years [46]. This is mainly due to the increasingly complexity of the FPGAs that are able to cope with the computation of the complex algorithms required in nowadays digital signal processing applications. Moreover, the systems should be efficiently implemented with the time to market constraint [47]. *High-level synthesis* (HLS) is an automated design process that interprets an algorithmic description and creates the *register-transfer level* (RTL) implementation of its desire behavior. Synthesis begins with a high-level specification of the problem, where behavior is generally decoupled. The code is analyzed, architecturally constrained, and scheduled to create a RTL *hardware description language* (HDL), which is then in turn commonly synthesized to the gate level by the use of a logic synthesis tool [48]. The goal of HLS is to facilitate hardware designers to efficiently define and model the system, by giving them better control over the architecture optimization, and by allowing the designer to describe the design at a higher level of abstraction while the tool does the RTL implementation. In example, *Vivado High-Level Synthesis* (Vivado HLS) is HLS tool which accelerates IP creation by enabling C, C++ and System C specifications to be directly targeted into Xilinx All Programmable devices without the need to manually create RTL [49, 50].

On the other side, the aforementioned approaches also differ in what hardware devices may be considered with the developing flow used. There are developing flows which are focused in just one specific vendor or architecture, such as NVIDIA CUDA allows to program just NVIDIA GPUs, and VIVADO HLS targets Xilinx FPGAs. This kind of design flows are specifically developed for particular devices and architectures, and use to be very well optimized for obtaining efficient applications. However, when different vendors and architectures are to be used, acquiring expertise in different methodologies is needed. As an alternative, there are also some design flows which aim to provide high level solutions for many different vendors and architectures. Even if the generality of these flows may negatively affect the efficiency of the developing tools and the hardware implementations, they offer portable languages which can be used for developing applications for different

devices. In this scenario, *Open Computing Language* (OpenCL) represents a very interesting framework for writing programs that can be executed across heterogeneous platforms [51].

The main objective of this Thesis work involves providing a new approach for hyperspectral imaging analysis applications under real-time constraints. This objective includes developing a new set of parallel hardware-friendly algorithms for hyperspectral imaging analysis that can be efficiently implemented across different parallel devices for applications under real-time constraints, as well as finding an efficient way of implementing the developed algorithms across different platforms. According to this, OpenCL represents a very suitable option as hardware description language for this Thesis, since it allows to target different hardware devices with a high level of abstraction, allowing to invest the biggest effort in the algorithm developing stage.

OpenCL views a computing system as consisting of a number of *compute devices*, which might be *Central Processing Units* (CPUs) or 'accelerators' such as GPUs or FPGAs, attached to a *host* processor (a CPU). OpenCL defines a C-like language for writing programs. Functions executed on the OpenCL devices are called 'kernels'. A single compute device typically consists of several *compute units*, which in turn comprise multiple *processing elements* (PEs). A single kernel execution can run on all or many of the PEs in parallel. However, how a compute device is subdivided into compute units and PEs is up to the vendor. The compute unit can be thought of as a 'core' but the notion of core is hard to define across all the types of devices supported by OpenCL. In addition to its C-like programming language, OpenCL defines an *application programming interface* (API) that allows programs running on the host to launch kernels on the compute devices and manage device memory, which is (at least conceptually) separated from host memory [52].

1.3 Research goals and contributions

As it has been explained, hyperspectral imaging is a powerful tool for many different applications due to the so useful information that it provides. However, the processes required for analysing this information according to the final user applications are computationally demanding. This problem has been traditionally tackled by implementing the algorithms required for processing the hyperspectral images into high performance

parallel computing devices, such as GPUs and FPGAs, for speeding up the hyperspectral imaging analysis, specially for applications under real-time constraints. However, the mathematical complexity of the state-of-the-art algorithms, as well as their so many data dependencies, leave little room for their acceleration through the use of parallel computing platforms. Many state-of-the-art works have tried to simplify the existing algorithms with the purpose of reducing their computational burden and data dependencies, making them more suitable for being efficiently implemented into parallel devices. However these kind of approaches for speeding up the hyperspectral analysis are always limited by the inherent complexity of the algorithms and typically result in decreasing their accuracy.

The main objective of this Thesis is to provide the research community with new solutions for hyperspectral imaging analysis applications under real-time constraints. This objective includes developing a new set of parallel hardware-friendly algorithms for hyperspectral imaging analysis that can be efficiently implemented across different parallel devices for applications under real-time constraints, as well as finding an efficient way of implementing the developed algorithms across different platforms. This objective is divided in four main goals which are detailed next.

- Developing a new hardware-friendly algorithm for linearly unmixing hyperspectral images. The developed algorithm must be able to perform the entire unmixing process in such a way that it can be efficiently implemented into parallel computing devices for applications under real-time or near real-time constraints.
- Developing a new hardware-friendly algorithm for fusing multispectral and hyperspectral images. The developed algorithm must be able to perform the fusion process avoiding complex matrix operations and data dependences so it can be efficiently implemented into parallel computing devices for applications under real-time or near real-time constraints.
- Developing a new hardware-friendly transform-based algorithm for lossy compressing hyperspectral images. The developed algorithm must be able to perform the spectral decorrelation and reduction stage of the compression process. Furthermore, a low complexity and a high degree of parallelism and scalability is desired in the algorithm, so it can be suitable for on-board applications.
- Verifying the suitability of the developed algorithms for real-time applications by implementing them into different parallel devices, namely GPUs and FPGAs, using

OpenCL. Furthermore, it is desired to evaluate the efficiency of OpenCL for developing hyperspectral imaging applications under real-time constraints, in relation with more specific developing platforms such as NVIDIA CUDA for NVICIDA GPUs. It is also desire to evaluate the efficiency of the different parallel devices according to the characteristics of the targeted applications and images.

The main contributions of this Thesis for accomplishing the established objective are detailed next.

- One algorithm, named *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), has been developed. This algorithm is able to perform the entire unmixing process and does not require a previous spectral reduction of the data set. The FUN algorithm is based on the concept of orthogonal projections, involving very simple operations which can be independently performed for each pixel of the image. All these facts together allow to obtain a more efficient hardware implementation of the entire unmixing process in a parallel device.

- One algorithm, named *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI), has been developed. This algorithm is capable of independently fusing each pixel of the hyperspectral image. The CoEf-MHI algorithm is also based on the concept of orthogonal projections and involves very simple operations. Due to this reasons, this algorithm represents a very suitable option for being implemented in parallel devices for fusion applications under real-time or near real-time constraint.

- One algorithm, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), has been developed. This algorithm provides an efficient solution for performing the spectral decorrelation and reduction of hyperspectral and ultraspeectral images for their compression. The HyperLCA transform is based on orthogonal projections too, involving very simple operations. It is specially design for independently processing small blocks of pixels of the image, being able to preserve the anomalous pixels through the compression-decompression process. All this facts make the HyperLCA algorithm a very suitable option for compressing hyperspectral images when the time and hardware resources are limited.

- The FUN algorithm, has been implemented in different parallel hardware devices, namely GPUs and FPGAs, and using different implementation languages, namely

NVIDIA CUDA and OpenCL. Since the FUN algorithm, the CoEf-MHI algorithm and the HyperLCA algorithm are all three based on orthogonal projections and make use of the modified Gram-Schmidt orthogonalization as their main operation core, it is assumed that the results obtained for the implementations of the FUN algorithm, in terms of efficiency, could be transposed to the rest of the developed algorithms within this Thesis work. The accomplished implementations prove that real-time or near real-time results can be obtained by using algorithms thought from their conception to be implemented into parallel hardware. As a further research goal, the utility of OpenCL for developing hardware implementations for different parallel hardware devices, such as GPUs and FPGAs, using one single hardware description language with efficient results, have also been proved.

Figure 1.4 graphically summarizes the main goals and contributions of this Thesis work.

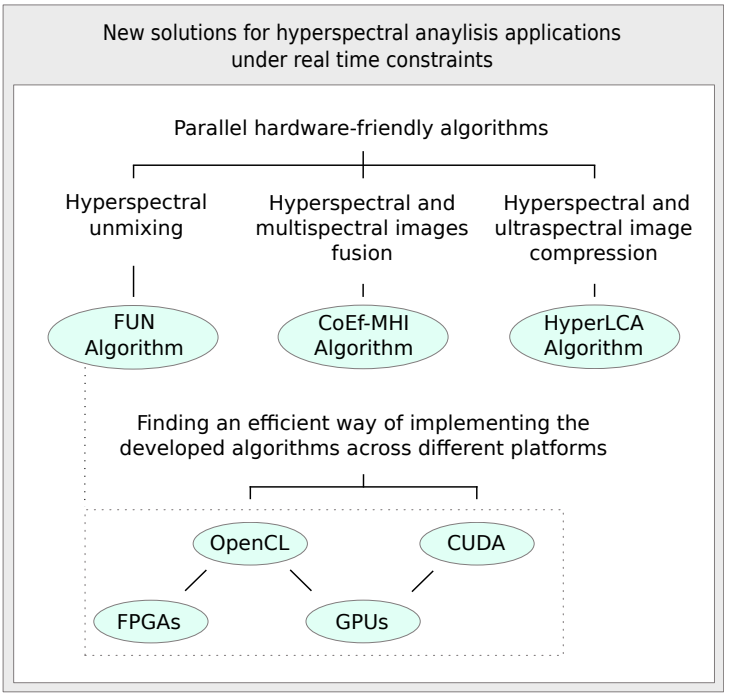


FIGURE 1.4: Main goals and contributions

Further contributions achieved as part of the progress done during this work have been done. These contributions are detailed next.

- The developed algorithms have been evaluated and compared against the state of the art algorithms for the same tasks, with the purpose of proving not only their efficiency, in terms of speed and computational burden, but also the accuracy of

their results. These comparisons result in a collection of different evaluation metrics and state-of-the art algorithms.

- The FUN algorithm has been implemented using different digital hardware design methodologies for the same hardware devices. This allows to compare the different methodologies and extract their benefits and weaknesses.
- The FUN algorithm has also been implemented into different kind of parallel hardware devices, using the same digital hardware design methodology, OpenCL. This allows not only to verify the suitability of the algorithm for applications under real-time or near real-time constrain, but also to make interesting comparisons between the devices, their benefits and weaknesses according to the characteristics of the different applications, and the versatility of OpenCL as a high level description language for heterogeneous hardware devices.

The results that will be shown in the following chapters of this Thesis demonstrate the goodness of all these contributions for actual and future hyperspectral imaging applications under real-time or near real-time constrains.

1.4 Organization of this document

The present document is structured in six chapters, including this introductory one, which is dedicated to present the main motivations and goals of this Thesis work. The rest of the chapters are briefly described next.

1.4.1 Chapter 2: Linear hyperspectral unmixing

This chapter presents the algorithm developed in this Thesis work for performing the linear hyperspectral unmixing process. The aforementioned algorithm, named *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), is totally described in this chapter. This algorithm is compared with the algorithms that conform the state-of-the art for the linear unmixing process using both synthetic and real hyperspectral images.

1.4.2 Chapter 3: Multispectral and hyperspectral image fusion

This chapter presents the algorithm developed in this Thesis work for performing the fusion of multispectral and hyperspectral images. The aforementioned algorithm, named *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI), is totally described in this chapter. This algorithm is compared with the algorithms that conform the state of the art for the fusion process using different images. Different evaluations metrics are used for comparing not only the accuracy of the fusion results, but also the efficiency of the algorithms, measuring the total number of floating point operations and the number of accesses to the memory.

1.4.3 Chapter 4: Hyperspectral and ultraspectral image compression

This chapter presents the algorithm developed in this Thesis for spectrally decorrelating and reducing hyperspectral and ultraspectral images. This algorithm, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), is totally described in this Chapter. The HyperLCA algorithm is evaluated using different images and also compared with the *Principal Component Analysis* (PCA) transform which provides the best state-of-the-art results for this compression stage. This algorithm is also used in conjunction with the CCSDS121 coder for evaluating the entire compression-decompression performance achievable using the proposed methodology. Finally, the effect of the compression within the HyperLCA on hyperspectral anomaly detection applications is also evaluated.

1.4.4 Chapter 5: Hyperspectral imaging acceleration through the utilization of high-performance computing platforms

This chapter presents different implementations of the FUN algorithm, described in Chapter 2. This chapter proves the suitability of the FUN algorithm for being implemented in parallel hardware devices. The FUN algorithm is implemented in different technologies, namely GPUs and FPGAs, and using different digital design methodologies. Interesting comparisons between the different hardware devices and technologies as well as between the different hardware description languages result from this Thesis work. Finally, since

all the algorithms developed in this Thesis work use similar operations, it is assumed that the results and conclusions obtained for the implementations of the FUN algorithm could be transposed to the rest of the developed algorithms within this Thesis work.

1.4.5 Chapter 6: Conclusions

Finally, the collection of contributions of this Thesis are summarized and further research works are proposed.

Chapter 2

Hyperspectral unmixing

This Chapter presents the algorithm developed in this Thesis work for performing the linear hyperspectral unmixing process. The aforementioned algorithm, named *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), is totally described in this chapter. This algorithm is compared with the algorithms that conform the state-of-the art for the linear unmixing process using both synthetic and real hyperspectral images. The results obtained in this Chapter demonstrate that the FUN algorithm is able to provide accurate results at a much lower computational burden than the state-of-the-art unmixing chains.

2.1 Outline

Linear spectral unmixing is nowadays an essential tool to analyse remotely sensed hyperspectral images. It is based on the idea that each captured pixel in a hyperspectral image, $M = [r_1, r_2, ..., r_{N_p}]$, which is composed by N_p pixels and N_b spectral bands, can be represented as a linear combination of a set of p spectrally pure constituent spectra or endmembers, e, weighted by an abundance factor, a, that establishes the proportion of each endmember in the pixel under inspection. This is shown in Equation 2.1, where e_j represents the j-th endmember signature, p is the number of endmembers in the image and $a_{i,j}$ is the abundance of endmember e_j in the pixel r_i. The noise present in the pixel r_i is contained in the vector n_i.

$$r_i = \sum_{j=1}^{p} e_j \cdot a_{i,j} + n_i \tag{2.1}$$

The unmixing of a hyperspectral image requires both the estimation of the number of endmembers, p, and the extraction of the endmembers, which conform a matrix $E = [e_1, e_2, ..., e_p]$, as well as the calculation of the abundance of each endmemer in each pixel of the image, conforming the matrix $A = [a_{r_1}, a_{r_2}, ..., a_{r_{N_p}}]$.

Although many different contributions have been uncovered during the last two decades, the majority of them are based on dividing the whole process of linearly unmixing a given hyperspectral image into three sequential steps: (i) estimation of the number of endmembers that are present in the hyperspectral image under consideration; (ii) extraction of these endmembers from the hyperspectral dataset ; and (iii) calculation of the abundances associated with the endmembers induced in the previous step per each mixed pixel of the image. Although this *de facto* processing chain has proven to be accurate enough for unmixing most of the images collected by hyperspectral remote sensors, it is also true that it is not exempt of drawbacks, such as the fact that all the possible combinations of algorithms in order to fully unmix a hyperspectral image according to the aforementioned processing chain demand a formidable computational effort, which tends to be higher as better is the performance of the designed unmixing chain. This troublesome issue unfortunately prevents the use of hyperspectral imaging technology in applications under real-time constraints, in which hyperspectral images have to be analysed in a short portion of time. Hence, there is a clear need to face the challenge of fully exploiting the unquestionable benefits of the hyperspectral imaging technology for these applications,

but concurrently overcoming the limitations imposed by the computationally complex nature of the processes involved.

For this purpose a new algorithm have been developed in this Thesis, named *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), which is capable of performing the entire unmixing process in an efficient way. This algorithm can be categorized as a *pure pixel geometrical* algorithm for linearly unmixing hyperspectral images, according to the definitions shown in Section 2.2, which briefly summarizes the state-of-the-art for linear hyperspectral unmixing.

2.2 State-of-the-art in linear hyperspectral unmixing

Hyperspectral unmixing refers to any process that separates the pixel spectra from a hyperspectral image into a collection of constituent spectra, or spectral signatures, called *endmembers* and a set of fractional *abundances*, one set per pixel. The endmembers are generally assumed to represent the pure materials present in the image and the abundances at each pixel to represent the percentage of each endmember that is present in the pixel [16–18].

The spectral sensors used to collect hyperspectral data can be calibrated to measure the reflectance of the incident radiation in the scene under analysis, instead of the absorbed radiation. This is the most commonly used set up. However, the reflectance is usually not a linear function due to secondary reflections and scattering effects. These nonlinear effects are not considered in the linear mixing moxel, which assumes that the secondary reflections and scattering effects can be neglected from the data collection procedure, and hence, the measured spectra can be expressed as a linear combination of the spectral signatures of the material present in the mixed pixel. Despite its simplicity, the linear mixing model is an acceptable approximation of the light scattering mechanisms in many real scenarios, and the basis of a plethora of unmixing models and algorithms. In those cases in which the impact of the secondary reflections or the scattering effects is relevant, it is necessary to resort to more complex nonlinear models [35], which demand *a priori* information about the geometry and physical properties of the observe objects, which results in a further increase of the computational complexity of the unmixing process. Figure 2.1 shows a graphical representation of the described linear and nonlinear mixing models.

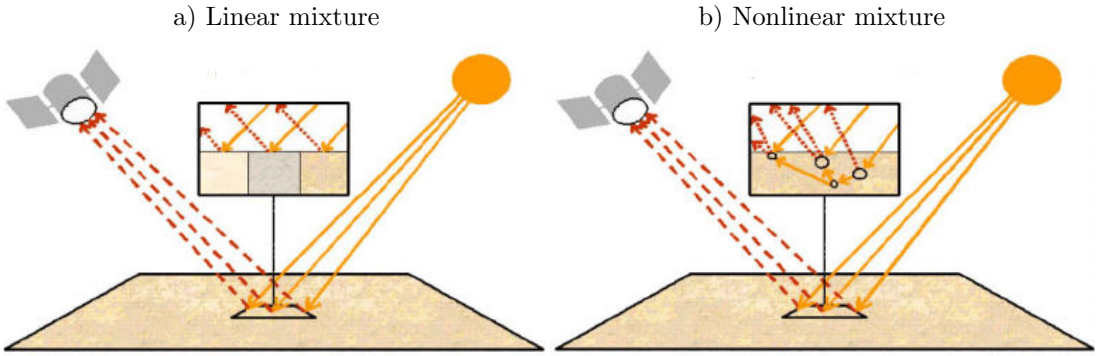


FIGURE 2.1: Linear and nonlinear mixing models.

For linearly unmixing a hyperspectral image, a set of steps are typically required, which conforms the unmixing chain. The most relevant steps, which are always required, involve:

- *Estimation of the number of endmembers* that are present in the hyperspectral image under consideration. The number of endmembers present in a given scene is usually much smaller than the number of bands. Hence, assuming that the linear mixing model is a good approximation, the hyperspectral vectors exist in a *p-dimensional* linear subspace, being p the number of endmembers present in the image. Therefore, the number of endmembers can be estimated as the intrinsic dimensionality of the data set. The *Virtual Dimensionality* (VD) [53] and the *Hyperspectral Subspace Identification by Minimum Error* (HySIME) [54] algorithms are typically used for this step.

- *Extraction of the endmembers* that are present in the data set. This step consists on identifying the pure spectral signatures present in the scene under inspection. Different approaches have been used for such purpose:

  - *Geometrical approaches* try to find the endmmbers as the spectral signatures which allows to build the simplex with the minimum volume that contains all the pixels of the image. Within the geometrical algorithms, there is a subset that can be categorized as *pure pixel* based unmixing algorithms. This subset selects the endmembers as real pixels of the image under analysis, assuming the presence in the data of at least one pure pixel per endmember. These algorithms find the set of most pure pixels in the data. The N-Finder algorithm [55], the *Vertex Component Analysis* (VCA) algorithm [56], the *Orthogonal Subspace Projection* (OSP) algorithm [57], the *Pixel Purity Index* (PPI) algorithm [58, 59] and the *Iterative Error Analysis* (IEA) algorithm [60] are some

of the most relevant *pure pixel* geometrical algorithms. On the contrary, the geometrical algorithms which are not categorized in the *pure pixel* subset select as endmembers 'virtual' vectors, which do not necessary match any pixel of the image under analysis, but that represent the vertexes of the simplex. The *Minimum Volume Simplex Analysis* (MVSA) [61] and the *Simplex Identification via Variable Splitting and Augmented Lagrangian* (SISAL) [62] algorithms are good examples of them.

- *Statistical approaches* use Bayesian methods for modeling statistical variability and to impose priors that can constrain solutions to physically meaningful ranges and regularize solutions to the linear unmixing process. The statistical unmixing algorithms are a powerful alternative when the spectral signatures are highly mixed, situation in which geometrical unmixing algorithms yield poor results. However, the statistical unmixing algorithms usually require a higher computational complexity compared with the geometrical based approach [63–67].
- *Sparse regression based approaches* assume that the observed image signatures can be expressed in the form of linear combinations of a number of pure spectral signatures known in advance. Hence, the unmixing process consist on finding the optimal subset of signatures in a (potentially very large) spectral library that can best model each mixed pixel in the scene [68–70].

- *Calculation of the abundances* associated with the endmembers induced in the previous step per each mixed pixel of the image. Once that the pure spectral signatures in the data set have been extracted, it is needed to measure the fraction of each pure material in each particular pixel. The calculation of the abundances are, very often, under two physical constraints, namely the *Abundance Non-negativity Constraint* (ANC), which forces the condition that $a_i > 0$ for all $1 < i < p$; and the *Abundance Sum-to-one Constraint* (ASC), which establishes that $\sum_{i=1}^{p} a_i = 1$. The *Fully Constrained Least Squares linear spectral Unmixing* (FCLSU) algorithm [71] is a popular algorithm for calculating the abundances under these two constraints. Despite the endmembers extraction and the abundances calculation processes are commonly implemented in separated steps when using the *pure pixel geometrical* approach, they are simultaneously performed in statistical and sparse regression unmixing algorithms.

Despite these three main steps, there are some extra steps which are usually carry on for improving the overall performance in particular unmixing applications:

- *Preprocessing.* When a data set is collected by a hyperspectral sensor, different processes are typically applied before the hyperspectral analysis, which depends on the sensor characteristics and the targeted application. In example, atmospheric corrections are normally applied to remote sensed hyperspectral data for compensating the attenuations and the scattering effect produced by the atmosphere at some particular wavelengths. Some extra processes may be also applied in some particular applications for reducing the noise of the hyperspectral data, for identifying and removing artifacts produced by the compression-decompression process, etc. [72].

- *Data reduction.* As aforementioned, the dimensionality of the space spanned by the pixels of a particular hyperspectral image is generally much lower than its number of spectral bands. This means that bands of hyperspectral images usually provide redundant information which can be reduced by identifying an appropriate subspace. This allows to carry on a spectral reduction of the data, typically improving the unmixing algorithms performance and complexity. The *Principal Component Analysis* (PCA) [73], the *Singular Value Decomposition* (SVD) [74] and the *Maximum Noise Fraction* (MNF) [75] algorithms are some of the most popular algorithm for this task. These three algorithms are based on projections techniques seeking for the best subspace for representing the collected hyperspectral data. Nevertheless, there exist different approaches which are not based on projections. In example, some techniques exploit the high correlation existing between adjacent bands of a hyperspectral image to select just the more representative bands [76, 77].

Although this processing chain has proven to be accurate enough for unmixing most of the images collected by hyperspectral remote sensors, it demands a formidable computational effort which unfortunately prevents the use of hyperspectral imaging technology in applications under real-time constraints. This computational effort is mainly due to the necessity of combining several algorithms in order to fully unmix a hyperspectral image according to the aforementioned processing steps. The computational effort tends to be higher as better is the performance of the designed unmixing chain.

According to the different approaches described, the FUN algorithm can be categorized as a *pure pixel geometrical* algorithm for linearly unmixing hyperspectral images. The

algorithms categorized as *pure pixel geometrical* typically try to select as endmembers the most different pixels within the data set. The orthogonality between the pixels as well as their magnitude are some of the characteristics used for identify which pixels should be selected as endmembers. In example, the PPI, VCA and the OSP algorithms sequentially extracts the endmembers as those pixels with the largest orthogonal projection with respect to the already extracted endmembers, using different methodologies for finding out the orthogonal components of the pixels of the data set with respect to the extracted endmembers. On the contrary, many unmixing algorithms evaluate all the pixels candidate to be endmembers at once, by evaluating the volume which can be spanned by the pixels selected as endmembers, such as the N-Finder algorithm, or by performing a constrained factorization of the hyperspectral image and evaluating the error obtained, such as the IEA algorithm or the *Nonnegative Matrix Factorization* (NMF) algorithm [78]. These methodologies are typically more complex and computational demanding than those based on orthogonal projections.

2.3 Computational complexity of the FUN algorithm

Generally, in order to complete the entire unmixing process, different algorithms have to be used to estimate the number of endmembers, extract them and calculate the abundances [16–18]. The main advantage of the FUN algorithm is that a single core algorithm (Gram-Schmidt) can be employed to simultaneously obtain the number of endmembers and extract them from the image under analysis. Additionally the same algorithm can be utilized to calculate the abundances. Besides the obvious gains that this fact brings in terms of faster computational times, the FUN algorithm has the advantage that it uses simple mathematical operations which can be easily parallelized, avoiding complex matrix operations like for instance calculating the inverse or computing the eigenvalues and eigenvectors, operations which are present in most of the unmixing chains. This represents an important advantage, because it eases the hardware implementation of the algorithm [25–28].

The parallelization suitability of the FUN algorithm when compared with other algorithms can be seen in the fact that the FUN algorithm is capable of estimating the number of endmembers and extracting them in a single process, performing exactly the same operations on each pixel of the image in each iteration, without needing any information from neighbouring pixels, thus avoiding creating data dependencies.

Another advantage of the FUN algorithm appears when we want to employ it to calculate the abundances, because it only needs the endmember matrix as an input, which is then orthogonalized. After the orthogonalization, the obtained matrix is multiplied by the hyperspectral image pixels directly obtaining the abundances of each pixel. This way, the FUN algorithm uses considerably less data during the process than other algorithms, which need not only the endmember matrix, but also the complete hyperspectral image.

2.4 Estimation of the number of endmembers and their extraction within the FUN algorithm

A new Fast algorithm for linearly UNmixing hyperspectral images, named FUN, has been developed in this Thesis with the aim of providing a solution to the problems mentioned before, achieving a high accuracy in the complete unmixing process. The FUN algorithm is based on the concept of orthogonal projections and allows performing the estimation of the number of endmembers and their extraction simultaneously, using simple operations, which can be also easily parallelized. Moreover, the FUN algorithm is able to calculate the abundances based on orthogonal projections too. The operations performed by the FUN algorithm to calculate the abundances are very similar to the operations performed to extract the endmembers, which makes it easier to achieve a hardware implementation of the entire unmixing process.

The FUN algorithm selects as first endmember the pixel of the hyperspectral image with the largest orthogonal projection to the centroid pixel. Afterwards, it sequentially performs the orthogonal projections of the hyperspectral image, in such a way that one endmember is found in each iteration.

After each projection, the algorithm evaluates if it is necessary to extract another endmember. For that purpose, the FUN algorithm estimates the percentage of information that can not be represented with the endmembers already selected using a stop factor, s. If the value obtained is smaller that an input parameter, α, the algorithm finishes, obtaining thus the total number of endmembers present in the image plus the extracted endmembers. This is explained with more details in Section 2.4.4.

The orthogonal projections of the hyperspectral image can be obtained using different methods. The FUN algorithm employs a modified version of the Gram-Schmidt method,

which features low computational complexity and allows the reuse of previously computed information, speeding up the overall process, and without any complex matrix calculation.

2.4.1 Background: the Modified Gram-Schmidt Orthogonalisation

The modified Gram-Schmidt method calculates the orthogonal projection of a vector e_i to a set of vectors $E = [e_1, e_2, ..., e_k]$ with $k < i$ by subtracting the portion of the vector e_i contained in the directions spanned by the vectors $E = [e_1, e_2, ..., e_k]$ with $k < i$.

Consequently, the modified Gram-Schmidt method allows to orthogonalize a set of independent vectors $E = [e_1, e_2, ..., e_n]$. The result is a set of orthogonal vectors $Q = [q_1, q_2, ..., q_n]$, and their normalized vectors $U = [u_1, u_2, ..., u_n]$. The Gram–Schmidt process works as shown in Algorithm 2.1.

Algorithm 2.1: Gram Schmidt Orthogonalization

Inputs: $E = [e_1, e_2, ..., e_n]$

for $i = 1$ **to** n **do**
  $q_i = e_i$;
  for $j = 1$ **to** $i - 1$ **do**
    $q_i = q_i - (e_i \circ u_j) \cdot u_j$;
  end
  $u_i = q_i / \|q_i\|$;
end

Outputs:
$Q = [q_1, q_2, ..., q_n]$ {Orthogonalized vector};
$U = [u_1, u_2, ..., u_n]$ {Orthonormalized vector};

In the pseudocode, "$\cdot$" represents the product of a vector and a scalar value, and the "$\circ$" operator is used for the scalar product between two vectors.

2.4.2 Initialization of the algorithm. Exctraction of the first endmember

The FUN algorithm has to be initialized with an endmember. Therefore, it is necessary to find a pure pixel in the image. In order to do that, different criteria can be applied. The FUN algorithm extracts each endmember based on the previously extracted endmembers, and hence, the accuracy of the first endmember extracted will affect the accuracy of the

entire unmixing process. Due to this reason it is crucial to ensure an accurate initialization criteria for the FUN algorithm. Three different initialization methods have been analyzed in this Thesis.

- **Initialization based on the maximum variance.**

  When observing a hyperspectral scene, one can see that in some bands, the reflectance of the different pixels has a higher variance than in others. Identifying the band in which the pixels present the highest variance is useful to find a pure pixel. Considering the linear mixing model, we can see that the mixed pixels are a weighted average of the reflectance of the different endmembers and their corresponding abundances; thus the reflectance of a mixed pixel is always a value which is between the maximum and minimum reflectance of the endmembers present in it.

  Taking this into account, it can be observed that in the band in which the differences between the pixels present in the scene is larger, the mixed pixels show intermediate values, while the more extreme values (maximum and minimum) correspond to pure pixels, and thus, can be considered as endmembers. This implies that in the band with the highest pixel variance, at least two pure pixels will always be identified, which correspond to those with the highest and lowest values. This is always fulfilled regardless of the number of endmembers and the number of mixed pixels.

  Considering the aforementioned facts, the FUN algorithm can be initialized by finding first the band where the variance among the pixels is the highest, and picking then the brightest pixel as first endmember. We note that the pixel with the lowest value could also be selected, but it is not advisable, since it might be comparatively more affected by the noise.

- **Initialization based on the centroid pixel**

  As many geometrical unmixing algorithms, the FUN algorithm tries to select as endmembers the most different pixels or the most extreme ones within the captured dataset. In order to select the first endmember according to this criterion, the average pixel, also named centroid, can be used, as it is done in the *Iterative Error Analysis* (IEA) algorithm [60].

  Considering how the FUN algorithm works, it is straightforward to use the centroid pixel to initialize the unmixing process. First, the average pixel is computed. Once this pixel is obtained, the orthogonal projection of all the pixels of the hyperspectral image with respect to this pixel is computed using the modified Gram-Schmidt

method. This orthogonal projection shows how much information of each pixel is not contained in the centroid pixel, showing how different each pixel is from the average. Finally, the FUN algorithm selects the pixel with the largest orthogonal projection as the first endmember.

- **Initialization based on the brightest pixel**

  Some geometrical unmixing algorithms selects the first endmember relying only on the bright of the sensed pixels, selecting the brightest one as endember. This is the case of the OSP algorithm [57].

The described initialization methodologies have been evaluated and compared in Section 2.6.1.2 using several synthetic hyperspectral images. According to the results obtained in the experiments, it is concluded that the initialization criteria based on the centroid pixel provides the most accurate results. Due to this reason, this is the initialization criteria used in the FUN algorithm in rest of this Thesis.

2.4.3 Simultaneously estimation of the number of endmembers and their extraction within the FUN algorithm

Once the first endmember has been selected, the FUN algorithm sequentially extracts new endmembers, by selecting the pixels with the largest orthogonal projections to the endmembers already extracted. Initially, the selected pixels are considered just candidates to be endmembers. Before these pixels are added to the list of extracted endmembers, the FUN algorithm estimates if it has really found a new endmember, or instead, if all the endmembers had already been found in the previous iteration and hence the extracted pixel is not really an endmember. This way, the FUN algorithm manages to extract the endmembers and to estimate the number of endmembers in a single process.

In order to decide if a pixel is really an endmember, when a new endmember candidate is found, the FUN algorithm calculates the percentage of information that would be lost if that endmember was not added, by calculating a stop factor, and comparing it with a parameter named α, which is given as an input to the algorithm and is defined by the user.

If the percentage of information lost is lower than α, the pixel is not considered an endmember, and the algorithm stops, since all the endmembers present in the image have already been found and extracted.

The pseudocode that describes the FUN algorithm is presented in Algorithm 2.2, where we consider that the noise in the input hyperspectral image (M) has been previously filtered. First of all, in line 5, the first endmember is selected according to one of the initialization criteria described in 2.4.2. In line 15, the orthogonal projection of each pixel to the endmembers already selected is calculated and saved in the matrix X. The stop factor, s, is calculated in line 16 for each pixel in the matrix X, obtaining a vector with the percentage of information of each pixel that cannot be represented with the endmembers already extracted. In line 18, the maximum value of the vector s, $s_{j_{\max}}$, is compared with the input parameter α. The algorithm finishes if $s_{j_{\max}} \leq \alpha$, otherwise, the pixel $r_{j_{\max}}$ is considered an endmember and the algorithm continues.

2.4.4 Stopping condition

According to the linear mixing model, all the pixels in a hyperspectral image are linear combinations of the pure spectral signatures (endmembers) present in it.

Considering p as the number of endmembers, a hyperspectral image corresponds with a subspace of p dimensions, conformed by p independent vectors, which are the endmembers. Hence, the maximum possible number of orthogonal vectors present is likewise p and as a consequence, the orthogonal projection of any other possible endmember candidate beyond the p present in the image will be null. The FUN algorithm takes advantage of this fact to determine when it has finished extracting the p endmebers.

This slightly changes when there is a certain amount of noise in the hyperspectral image. We assume that the noise present in the image has an aleatory nature and is not totally contained in the space spanned by the endmembers. Hence, the orthogonal projection of a new endmember candidate after the p endmembers have been already extracted will not be null, but instead very small, and will correspond to the noise present in the pixel which is not contained in the space spanned by the p extracted endmembers. Due to this reason, a higher *Signal-to-Noise Ratio* (SNR) will guarantee smaller values of the projection of a new endmember candidate after the p endmembers have been already extracted.

Algorithm 2.2: Process to extract the endmembers

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, $\alpha = 1$

$E = []$; {Endmembers matrix }
$Q = []$; {Endmembers' Gram Schmidt orthogonalization}
$U = []$; {Endmembers' Gram Schmidt orthonormalization}
$X = [x_1, x_2, ..., x_{N_p}] = M$ {Auxiliary copy of the hyperspectral image}
$e_1 = x_j$ where x_j is the pixel of the hyperspectral image selected as first endmember according to the initialization criteria; {Select the first endmember}
$q_1 = e_1$;
$u_1 = q_1/\|q_1\|$;
$E = [e_1]$;
$Q = [q_1]$;
$U = [u_1]$;
$p = 1$; {Number of endmembers found}
exit = 0;
while *exit* $= 0$ **do**
    for $j = 1$ **to** N_p **do**
        $x_j = x_j - (x_j \circ u_p) \cdot u_p$;
        $s_j = 100 \cdot \|x_j\|/\|r_j\|$;
    end
    if $\max(s) \leq \alpha$ **then**
        exit = 1;
    else
        $j_{\max} = \arg\max(s_j)$;
        $p = p + 1$;
        $q_p = x_{j_{\max}}$;
        $u_p = x_{j_{\max}}/\|x_{j_{\max}}\|$;
        $e_p = r_{j_{\max}}$;
    end
end

Outputs: p, {number of endmembers}
$E = [e_1, e_2, ..., e_p]$, {Endmembers}
$Q = [q_1, q_2, ..., q_p]$, {Orthogonalized endmembers}
$U = [u_1, u_2, ..., u_p]$, {Orthonormalized endmembers}

In order to create a stopping condition attending to the aforementioned facts, a stop factor (s) is created and defined as the percentage of a pixel that cannot be represented as a linear combination of the endmembers already extracted according to the following equation:

$$s = 100 \cdot \frac{\|x_i\|}{\|r_i\|} \tag{2.2}$$

where $\|r_i\|$ represents the norm of the i-th pixel and $\|x_i\|$ is the norm of the gram-schmidt orthogonalization of r_i.

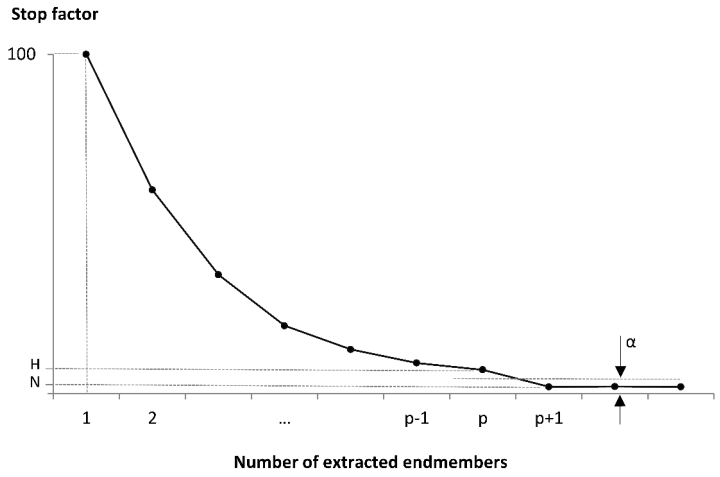


FIGURE 2.2: Theoretical representation of the values that would be obtained for the stop factor as the number of endmembers extracted by the FUN algorithm increase, from 1 to $p+1$, being p the real number of endmembers present in the hyperspectral image.

Figure 2.2 shows a theoretical example of how the value of the stop factor changes as the endmembers are extracted. As it can be observed, the stop factor for the first endmember is always 100 and it decreases smoothly as the number of endmember extracted increases, until the p endmembers present in the image are extracted. When trying to extract the $(p+1)$-th endmember, the stop factor gets significantly smaller. If we consider H the value of the stop factor of the p-th endmember and N the value of the stop factor of the $(p+1)$-th endmember candidate, the H/N ratio will increase with the SNR, as previously explained.

According to this, a parameter α can be defined for stopping the endmember extractoin process, which will represent the percentage of information of the next pixel susceptible of being selected as endmember that will be considered as noise, as shown in Figure 2.2. Selecting an appropriate α value will be easier for a high H/N ratio. Due to this reason, it is recommendable to perform a noise filtering process before the utilization of the FUN algorithm for unmixing hyperspectral images.

Several experiments have been uncovered in this Thesis which verify these reasonings. These experiments are deeply described in Section 2.6.1.1. Considering the exposed facts as well as the results obtained in the experiments shown in Section 2.6.1.1, α has been set to 1 in the rest of the experiments done in this Thesis, and the noise in the hyperspectral image has always been filtered before applying any algorithm for extracting the endmember.

2.5 Abundance estimation within the FUN algorithm

Considering the linear unmixing model shown in Equation 2.1, once the p endmembers $E = [e_1, e_2, ..., e_p]$ have been found, it is necessary to estimate the abundances $(a_{i,j})$ of each endmember e_j in a pixel r_i, in order to complete the unmixing process.

In this section we explain how the FUN algorithm estimates the abundances. The main advantage of the FUN algorithm when used to calculate the abundances is that it works only with the extracted endmembers $E = [e_1, e_2, ..., e_p]$, which are utilized to obtained a matrix Q^{**} that allows to directly estimate the abundances of each pixel r_i by the operation $a_{i,:} = (Q^{**})^T \cdot r_i$. Therefore, the abundances matrix can be estimated as $A = (Q^{**})^T \cdot M$.

The $*$ and $**$ symbols are used to show that matrices U^*, Q^* and Q^{**} are obtained as variations of the matrices U and Q, as shown below, but still preserve their properties of being formed by orthonormal and orthogonal vectors respectively.

In order to obtain the matrix Q^{**} which allows to easily estimate the abundances, the Gram-Schmidt orthogonalization has to be applied p times. This way, the FUN algorithm is able to extract the exact information that makes each endmember different. In particular, when the Gram-Schmidt method is applied in order to orthogonalize a set of endmembers $E = [e_1, e_2, ..., e_p]$, the orthogonalization is performed by subtracting to each endmember the information that is contained in the endmembers already orthogonalized. Figure 2.3 shows an example in which two vectors e_1 and e_2 are orthogonalized. It can be seen that the vectors resulting from the Gram-Schmidt process, q_2 and u_2, contain the information present in the endmember e_2 which is not part of endmember e_1. However, vectors q_1 and u_1 contain information which is part of e_2.

When the Gram-Schmidt method is applied to a set of p endmembers, the resulting vectors q_p and u_p will contain the information of the endmember e_p that is not contained in the rest of the endmembers $E = [e_1, e_2, ..., e_{p-1}]$. This information is what makes the endmember e_p different from the rest. However, the rest of vectors in $Q = [q_1, q_2, ..., q_{p-1}]$ and $U = [u_1, u_2, ..., u_{p-1}]$ will contain information of more than one endmember. If the Gram-Schmidt method is applied p times to the set of endmembers, and the last endmember (e_p) is a different one each time, the corresponding q_p and u_p can be saved to the matrices $Q^* = [q_p^{*(1)}, q_p^{*(2)}, ..., q_p^{*(p)}]$ and $U^* = [u_p^{*(1)}, u_p^{*(2)}, ..., u_p^{*(p)}]$, in such a way that these matrices

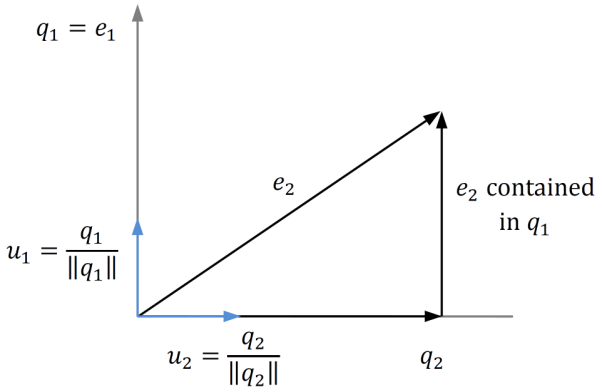


FIGURE 2.3: Example of Gram-Schmidt orthogonalization of the endmember e_2 with respect to the endmember e_1; q_2 represents the amount of information in e_2 which is not contained in e_1; u_1 and u_2 are a unitary vectors with the directons of q_1 q_2, respectively.

will contain the information that is exclusive to a specific endmember, i.e. the information in each endmemer e_i which is not present in the rest of endmembers.

The resulting orthogonalized and orthonormalized set of vectors, Q^* and U^*, both comprising the information exclusive to each endmember, can be used to project the pixels of the image in their directions, as shown in Figure 2.4.

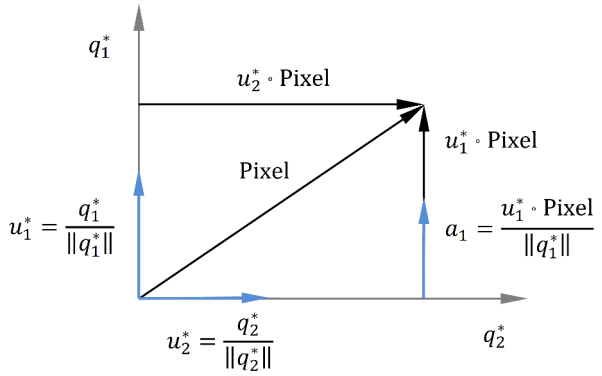


FIGURE 2.4: Example of projection of a pixel in the direction of two endmembers. Vectors q^* and u^* refer to the vectors obtained with the FUN algorithm by orthogonalazing the endmembers using the modified Gram-Schmidt method. Using these vectors, the abundance of each endmember in the pixel can be obtained as shown in a_1, which represents the abundance of the endmember e_1 in this pixel.

As shown in the example, by projecting the pixels of the hyperspectral image in the set of vectors Q^* and U^*, it is possible to obtain the portion of each pixel that can be represented in each of the directions $U^* = [u_p^{*(1)}, u_p^{*(2)}, ..., u_p^{*(p)}]$. This portion can be formally described as:

$$\text{Portion}_i = u_p^{*(i)} \circ r_i \tag{2.3}$$

where r_i is a pixel and $\circ$ is the scalar product, hence Portion_i is a scalar value. Finally, the reconstructed pixels (r_i) can be obtained as the sum of these portions multiplied by their corresponding directions as:

$$r_i = \sum_{i=1}^{p} \text{Portion}_i \cdot u_p^{*(i)} \tag{2.4}$$

Taking into account that $u_p^{*(i)} = \frac{q_p^{*(i)}}{\|q_p^{*(i)}\|}$, the expression can be re-written as:

$$r_i = \sum_{i=1}^{p} \text{Portion}_i \cdot \frac{q_p^{*(i)}}{\|q_p^{*(i)}\|} \tag{2.5}$$

We can assume that $q_p^{*(i)} \sim e_i$, since $q_p^{*(i)}$ comprises the information which makes e_i different from the rest of endmembers. Hence, the expression in Equation 2.5 can be re-written as:

$$r_i = \sum_{i=1}^{p} \text{Portion}_i \cdot \frac{q_p^{*(i)}}{\|q_p^{*(i)}\|} \sim \sum_{i=1}^{p} \text{Portion}_i \cdot \frac{e_i}{\|q_p^{*(i)}\|} \tag{2.6}$$

And finally the abundances of a pixel can be calculated as:

$$A_i = \frac{\text{Portion}_i}{\|q_p^{*(i)}\|} = \frac{u_p^{*(i)} \circ \text{Pixel}_i}{\|q_p^{*(i)}\|} \tag{2.7}$$

The vectors of the matrix $U^* = [u_p^{*(1)}, u_p^{*(2)}, ..., u_p^{*(p)}]$ can be divided once by the norms of the corresponding vector $Q^* = [q_p^{*(1)}, q_p^{*(2)}, ..., q_p^{*(p)}]$, obtaining the matrix $Q^{**} = [q_p^{**(1)}, q_p^{**(2)}, ..., q_p^{**(p)}]$. The resulting vectors in Q^{**} are then multiplied by each pixel of the hyperspectral image, obtaining thus their abundances, instead of dividing each of the pixels of the matrix by the norms of the corresponding vectors $Q^* = [q_p^{*(1)}, q_p^{*(2)}, ..., q_p^{*(p)}]$. This way, the abundances of the hyperspectral image can be directly obtained as:

$$A = (Q^{**})^T \cdot M \tag{2.8}$$

The pseudocode in Algorithm 2.3 shows the process that has been described to calculate the abundances from the obtained endmembers by using the Gram-Schmidt orthogonalization. In lines 3 to 11 the Gram-Schmidt orthogonalization is performed p times, each finishing with a different endmember e_p.

Algorithm 2.3: Process to obtain the abundances

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, p, $E = [e_1, e_2, ..., e_p]$

1 $E = [E, E]$;
2 Norms $= [], U = [], Q = [], U^* = [], Q^{**} = []$;
3 **for** $k = 2$ **to** $p + 1$ **do**
4 $\quad U_1 = E_k / \|E_k\|$;
5 $\quad$ **for** $j = 2$ **to** p **do**
6 $\quad\quad x = E_{k+j-1}$;
7 $\quad\quad$ **for** $i = 1$ **to** $j - 1$ **do**
8 $\quad\quad\quad q_j = x - (x \circ q_j) \cdot q_j$;
9 $\quad\quad$ **end**
10 $\quad\quad U_j = q_j / \|q_j\|$;
11 $\quad$ **end**
12 $\quad$ Norms $= [\text{Norms}, \|q_p\|]$;
13 $\quad U^* = [U^*, u_p]$;;
14 **end**
15 **for** $i = 1$ **to** p **do**
16 $\quad Q_i^{**} = U_i^* / \text{Norms}_i$;
17 **end**
18 $A = (Q_i^{**})^T \cdot \text{IMG}$; {Abundances}

Outputs: Abundances

As it can be observed, the FUN algorithm does not impose the nonnegative and the sum-to-one constraints, and therefore it can happen that the resulting abundances do not meet the mentioned constraints. However, we have experimentally observed that the abundances estimated by the FUN algorithm are in fact very close to meet these constraints. Although some negative values appear, they are really close to zero, and the sum of the abundances is also very close to one. Taking this into account, it is possible to apply the constraints to the abundances estimated by the FUN algorithm, by setting all negative abundances to zero and then dividing all the abundance vectors by the sum of their terms. All these facts are better illustrated and demonstrated in the experimental results of Section 2.6.

2.6 Experimental results

This section discloses the most significant results obtained by the FUN algorithm when it is utilized to perform the linear unmixing process. A set of artificially generated hyperspectral data as well as the real Cuprite hyperspectral image collected by the *Airborne Visible/Infrared Imaging Spectrometer* (AVIRIS) [5] have been used as inputs to the algorithm. The performance of the FUN algorithm is evaluated by comparing the results with those obtained, for the same input hyperspectral images, with some of the most relevant algorithms of the state-of-the-art, namely the *Hyperspectral Subspace Identification by Minimum Error* (HySIME) algorithm [54] for the estimation of the number of endmembers; the *Vertex Component Analysis* (VCA) algorithm [56] for the endmember extraction; and the FCLSU algorithm [71] for the calculation of the abundances. These algorithms have proven to yield very accurate results for the corresponding three stages of the unmixing process, as it can be concluded by the fact that they are considered a reference in many scientific works in the field.

2.6.1 Experiments on synthetic hyperspectral data

Artificial hyperspectral images represent an excellent test bench in order to study the goodness of our proposal, because both their signature endmembers, and their abundances are known in advance. In particular, the hyperspectral images used in this work were generated using three different Matlab functions, and consequently three different types of synthetic hyperspectral images were obtained. These three Matlab functions allow to create hyperspectral images of a spatial size defined by the user, from p spectral signatures randomly selected from the *U.S. Geological Survery* (USGS) digital spectral library [79] that are then mixed according to abundance fractions, which are generated using different probabilistic distributions. In particular, Type 1 hyperspectral images are generated using the Gauss spheric distribution; Type 2 images are generated with the Legendre polynomial distribution, and finally Type 3 images are generated with a properly tuned Dirichlet distribution. In addition, a certain amount of Gaussian noise is added so that the generated images have a user-defined *Signal-to-Noise Ratio* (SNR).

Different sets of synthetic hyperspectral images were randomly generated for each of the experiments described in Sections 2.6.1.1 to 2.6.1.6, varying their size, number of endmembers, abundances distribution and SNR. The spectral signatures used to create each of the images were randomly selected from the aforementioned USGS digital spectral library.

2.6.1.1 Effect of noise filtering in the performance of the FUN algorithm

In the linear mixing model, each pixel of the image can be represented as the linear combination of the endmembers present in it, weighted by the corresponding abundances, plus a certain amount of noise. The FUN algorithm performs the unmixing process according to this linear mixing model assuming that the amount of noise in the image is negligible, and hence, the endmembers can be directly extracted from the pixels in the hyperspectral image. The presence of a high level of noise in the scene would make the FUN algorithm to obtain less accurate results; and as it was explained in Section 2.4.4, filtering the noise has also a important impact in the stopping criterion.

In order to filter the noise, we utilize in this study the first stage of the HySIME algorithm, in which the noise is estimated and subtracted from the hyperspectral image. However, other noise filtering options are possible [80, 81].

The benefits of the noise filtering and its influence in the stopping condition of the FUN algorithm, are evaluated by performing simulations in which the endmembers are extracted with the FUN algorithm from hyperspectral images filtering and without previously filtering the noise. In these simulations, the number of endmembers to be extracted has been fixed to a number higher than the actual number of endmembers, and the stop factor, defined in Section 2.4.4, is saved for all iterations of the FUN algorithm (see line 18 of Algorithm 2.2).

Figures 2.5 and 2.6 respectively show the different values taken by the stopping condition depending on the number of endmembers extracted and the SNR of the image when the noise is filtered and when it is not.

As it can be seen in the results, when the noise of the hyperspectral image is previously filtered, the value taken by the stop factor tends to be null if we try to extract more endmembers than the real number of endmembers present in the image (p). However, when we are extracting a number of endmembers smaller than p, the stop factor is significantly

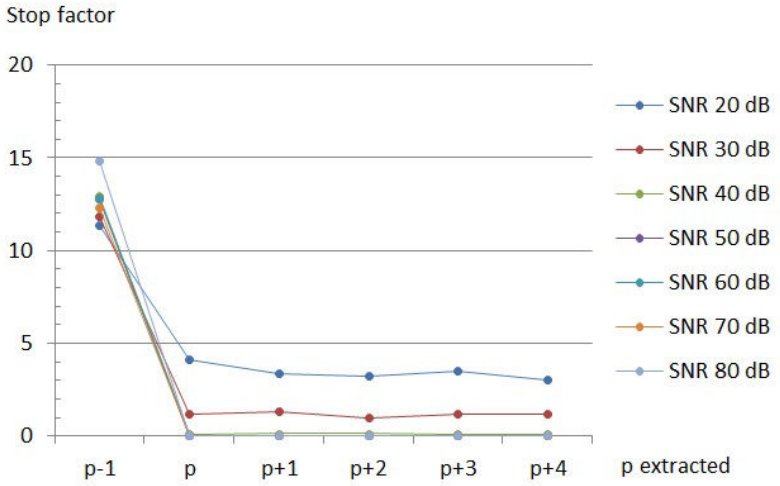


FIGURE 2.5: Mean values of the stop factor depending on the number of endmembers extracted and the SNR of the hyperspectral image, filtering the noise of the image. Images with 5 endmembers.

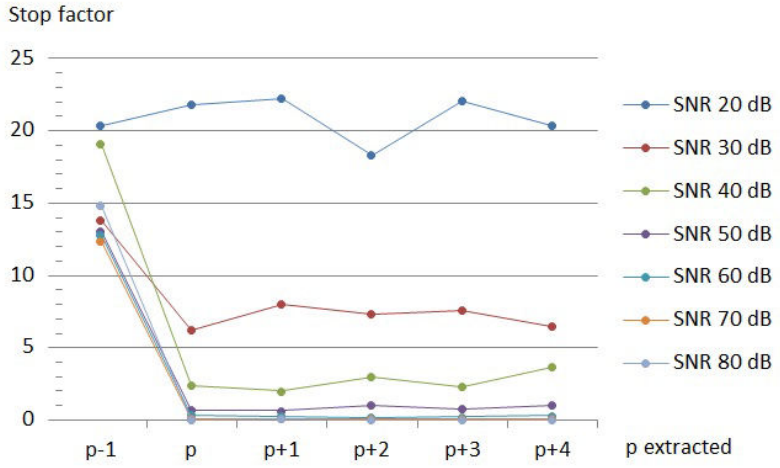


FIGURE 2.6: Mean values of the stop factor depending on the number of endmembers extracted and he SNR of the hyperspectral image, without filtering the noise of the image. Images with 5 endmembers.

higher. This makes it possible to define parameter α, which can be used to determine when to stop the FUN algorithm, as it was described in Section 2.4.4.

We observe a similar trend for the stop factor in those cases in which the noise is not previously filtered (Figure 2.6). However, in these cases the values taken by the stop factor once all the endmembers have been extracted is not so close to zero, what makes it more difficult to determine the right value of α.

Considering the exposed facts, in the subsequent simulations exposed in this Section, α has been set always to 1, and the noise in the hyperspectral image was always filtered before applying any algorithm for endmember extraction.

2.6.1.2 Accuracy of the first endmember extracted with the different initialization methods

In order to evaluate the quality of the different initialization methods proposed in Section 2.4.2, a set of simulations have been performed using synthetic hyperspectral images.

These images have been created with 5, 10 and 15 endmembers with 224 spectral bands each, all taken from the USGS library; and different SNR values, which have been set to 20, 30, 40, 50 and 60 dB, and a dimension of 90,000 pixels. Moreover, the type of image has been also varied, from Type 1 to Type 3, according to the different probabilistic distributions utilized to create the abundance maps, as it was explained in Section 2.6.1. All this makes a total of 45 groups of 10 different hyperspectral images each.

Ten simulations were performed for each group of synthetic hyperspectral images, selecting one pixel as first endmember with each of the three methods proposed in Section 2.4.2. The accuracy of the pixel selected as endmember is evaluated by calculating the spectral angle, as described in (2.9).

$$\mathrm{SA} = \arccos \frac{\mathrm{EE} \circ \mathrm{RE}}{\|\mathrm{EE}\| \cdot \|\mathrm{RE}\|} \tag{2.9}$$

where *EE* refers to the extracted endmembers, *RE* refers to the real endmembers present in the image, "$\circ$" represents the scalar product between the two endmembers vectors, and "$\cdot$" represents the product between two scalar values. The norm of the endmember vectors is represented as "$\|\|$".

The mean results obtained for the spectral angle for each of the initialization methods described in Section 2.4.2 are shown in Table 2.1. Column headed as *I1* refers to the spectral angle obtained with the first endmember extracted by selecting the pixel with the highest value in the band with the maximum variance. Column headed as *I2* refers to the spectral angle obtained with the first endmember extracted using the centroid pixel. Finally, column headed as *I3* refers to the spectral angle obtained with the first endmember extracted as the brightest pixel of the hyperspectral image.

TABLE 2.1: Mean spectral angle obtained for the first endmember extracted using each of the three initialization methods proposed

I1	I2	I3
0.815	0.759	0.851

According to the results obtained, it is concluded that the most accurate endmember is obtained computing the centroid, and hence, this method has been the one used in the rest of this Thesis work.

2.6.1.3 Performance of the FUN algorithm for estimation the number of endmembers

In order to evaluate the performance of the FUN algorithm when estimating the number of endmembers in the image, the results obtained by the FUN algorithm are compared with those of the HySIME algorithm targeting the same hyperspectral images. This bench of synthetic images is very similar to the one described in the Section 2.6.1.2.

Attending to the different configurable parameters, ten synthetic images were created for each *Type*, *SNR* and p combination. The mean results obtained in the simulations performed, for each group of hyperspectral images, are computed and graphically displayed in Figure 2.7, where the axis labeled as *Correct estimations* refers to the percentage of cases in which the number of endmembers estimated is exactly the number of endmembers present in the image (represented with circles), and the axis named *Mean error* shows the mean values of the differences between the number of endmembers obtained by the evaluated algorithms and the real number of endmembers present in the images (represented with bars).

It can be observed that the percentage of correct estimations is similar for both algorithms, but the mean errors obtained using the FUN algorithm are, in the vast majority of the cases, considerable smaller than the mean errors obtained with the HySIME algorithm. According to these facts, it can be concluded that the FUN algorithm provides better results than HySIME when estimating the number of endmembers of a hyperspectral image.

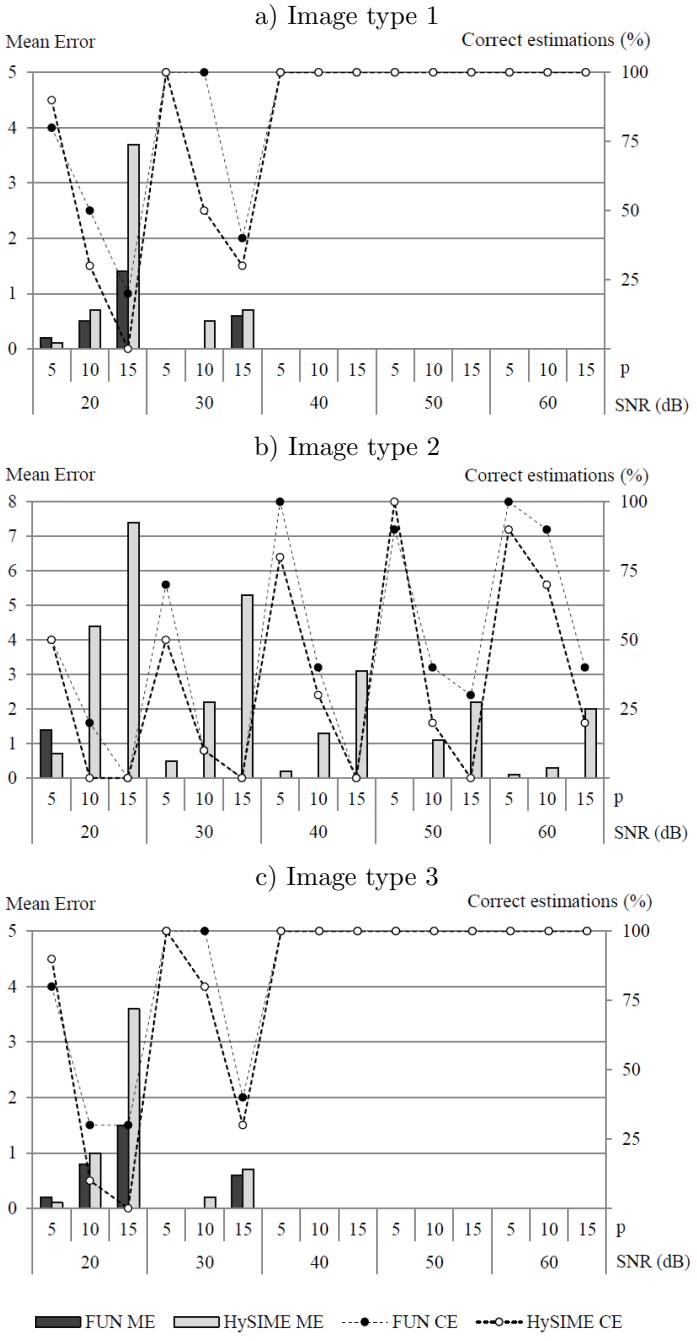


FIGURE 2.7: Number of endmembers estimation. Percentage of correct estimations and mean error. a) Image type 1. b) Image type 2. c) Image type 3.

2.6.1.4 Performance of the FUN algorithm for the endmembers extraction knowing the number of endmembers in advance

Using a similar bench of synthetic images to the one described in the Section 2.6.1.2, we performed additional simulations in which we evaluated the quality of the endmembers

extracted by the FUN algorithm. For this purpose, we explicitly informed the FUN algorithm of the number of endmembers present in the image, we extracted them and then we compared them with those obtained when the endmembers are extracted with the *Vertex Component Analysis* (VCA), by calculating the spectral angle, as described in (2.9).

Attending to the different configurable parameters, ten synthetics images were created for each *Type*, and p combination. The mean results obtained in the simulations performed using these images are displayed in Table 2.2.

TABLE 2.2: Mean spectral angle obtained with the FUN and the VCA algorithms

Image's characteristics			Mean spectral angle (°)	
SNR (dB)	p	Type	FUN	VCA
20	5	1	1.339	1.070
		2	3.258	2.329
		3	2.001	1.592
	10	1	2.167	1.954
		2	5.307	4.820
		3	1.997	1.927
	15	1	3.473	2.903
		2	7.748	7.083
		3	2.583	3.197
40	5	1	0.113	0.091
		2	1.702	1.728
		3	0.119	0.288
	10	1	0.273	0.208
		2	1.435	1.897
		3	0.331	0.314
	15	1	0.410	2.012
		2	3.286	7.090
		3	0.292	1.221
60	5	1	0.011	0.010
		2	0.701	0.697
		3	0.011	0.010
	10	1	0.023	0.043
		2	0.929	1.625
		3	0.030	0.019
	15	1	0.047	0.021
		2	1.059	9.325
		3	0.044	0.048

Figure 2.8 graphically shows the spectral angle obtained between the endmembers extracted by the FUN and the VCA algorithms and the real endmembers present in the hyperspectral image. These results are shown in more detail in Table 2.2. It can be seen that, in the images with a high amount of noise (SNR = 20) the results obtained using the

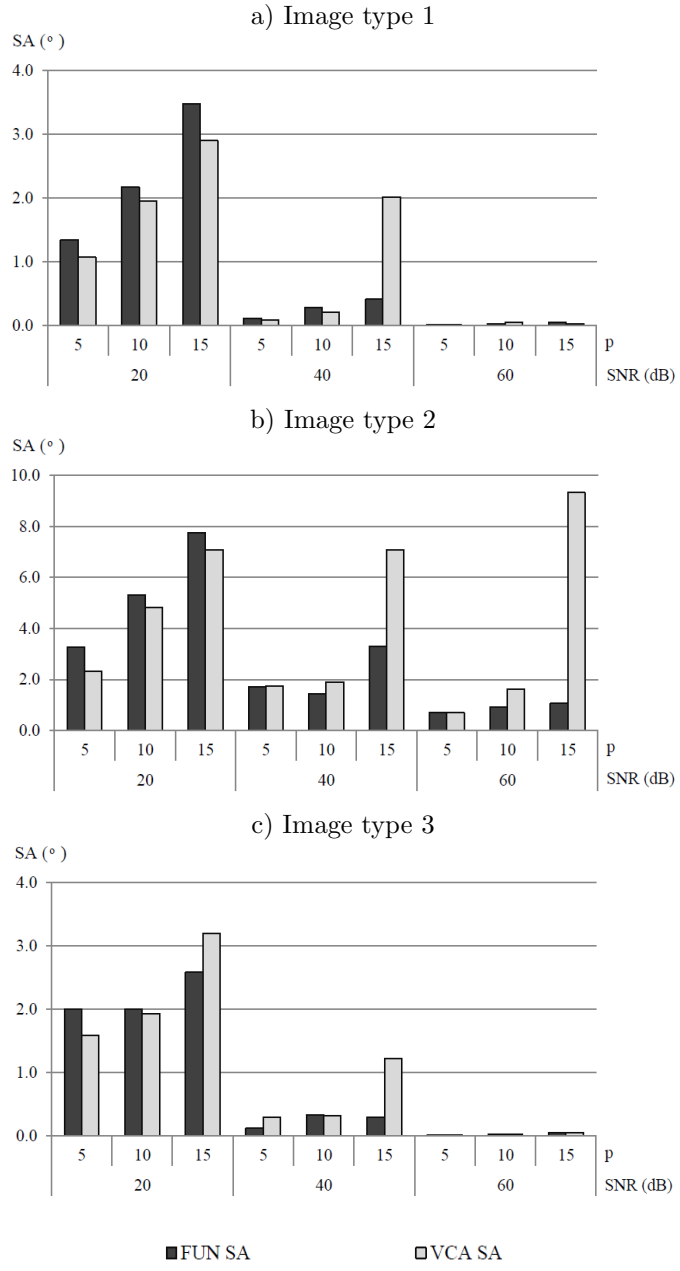


FIGURE 2.8: Mean spectral angle obtained with the FUN and the VCA algorithms using a set of different synthetic hyperspectral images, varying the signal to noise ratio, SNR, the number of endmembers in the scene, p, and the abundances distribution type.

FUN algorithm are very close to the ones obtained by the VCA algorithm. However, the images with less noise the results obtained by the FUN algorithm are much better than the results obtained with the VCA algorithm, improving as the number of endmembers increase. We can conclude that the FUN algorithm is as accurate as the VCA algorithm in terms of quality of the extracted endmembers, showing similar spectral angle values,

which tend to be significantly better for the FUN algorithm as the number of endmembers increase.

2.6.1.5 Performance of the FUN algorithm for hyperspectral unmixing

Finally, we evaluate the performance of the FUN algorithm for a typical use, i.e. when it is utilized for estimating the number of endmembers and extracting them simultaneously. The automatic stopping condition parameter α for the FUN algorithm has been set to $\alpha = 1$. The HySIME algorithm is utilized as a reference to evaluate the estimated number of endmembers, and the VCA algorithm is used as reference to evaluate the quality of the extracted endmembers.

A new image bench, with the same configuration of the data sets used in the previous experiments, has been generated and used as input of the algorithms. The generated images have been grouped in sets of ten images, according to their characteristics, and the mean results obtained for the images in each group are shown in Table 2.3. *FUN* columns contain the results obtained with the FUN algorithm, while the columns named *HS* contain the results obtained with the HySIME algorithm and columns headed as *VCA* contain the results obtained with the VCA algorithm. The spectral angle values were only calculated when the number of endmembers estimated by the HySIME algorithm and by the FUN algorithm was the same. The number of simulations in which the HySIME and the FUN algorithm estimated the same number of endmember is shown in the column named *Number of Matching Cases, NMC* (%).

Figure 2.9 graphically shows the percentage of correct number of endmembers estimation and the mean error in these estimations, respectively, obtained by the FUN and the HySIME algorithms. As it is seen, the results obtained by the FUN algorithm in terms of the mean error obtained when estimating the number of endmembers are generally better than the results obtained by the HySIME algorithm. Figure 2.9 also shows the mean spectral angle obtained by the FUN algorithm and the VCA algorithm. As it can be seen, the results obtained by the FUN algorithm are very similar, or much better in some cases, that the results obtained by the VCA algorithm.

These results demonstrate the good performance of the FUN algorithm when compared with the HySIME and the VCA algorithms, with the unquestionable gain that the FUN algorithm is able to perform two tasks in the same process, while to achieve the same with

TABLE 2.3: Average results obtained for hyperspectral unmixing

Image's characteristics		Estimation of the number of endmembers				Mean spectral angle (°)		
		Correct stimations (%)		Mean error				
SNR (dB)	p	FUN	HS	FUN	HS	FUN	VCA	NMC (%)
20	5	80	60	0.2	0.6	1.232	1.150	50
	10	40	0	0.9	1.8	1.656	3.985	30
	15	10	0	2.5	5.6	-	-	0
30	5	100	100	0	0	0.379	0.598	100
	10	80	40	0.3	0.9	0.575	0.834	40
	15	40	40	1.8	2.5	0.851	3.887	60
40	5	90	100	0.1	0	0.150	0.120	90
	10	70	80	0.3	0.2	0.470	0.865	90
	15	90	90	0.3	0.4	0.365	2.343	90
50	5	100	100	0	0	0.076	0.044	100
	10	90	90	0.1	0.2	0.162	0.127	90
	15	70	60	0.7	0.8	0.225	4.896	80
60	5	100	100	0	0	0.026	0.021	100
	10	70	80	0.5	0.2	0.029	0.074	70
	15	70	70	0.8	0.9	0.063	6.679	90

HySIME and VCA it is necessary to execute the two algorithms sequentially. This represents an advantage for the FUN algorithm, which is able to speed up the hyperspectral unmixing process, and will yield also hardware implementations of a reduced complexity.

2.6.1.6 Performance of the FUN algorithm for abundance estimation

Once we have evaluated the performance of the FUN algorithm for the simultaneous estimation of the number of endmember and their extraction, we have performed additional simulations in order to assess its usefulness for the estimation of the abundances of the extracted endmembers.

For this purpose, we generate a new set of synthetic hyperspectral images, with the same combinations of configuration parameters used in the previous experiments.

Considering that the number of endmembers is known in advance, the extraction of the endmembers is performed with the FUN algorithm, which is then utilized to estimate the abundances. The results are compared with those obtained with the *Fully Constrained Least Squares* (FCLSU) Linear Spectral Unmixing algorithm for the same images and endmembers.

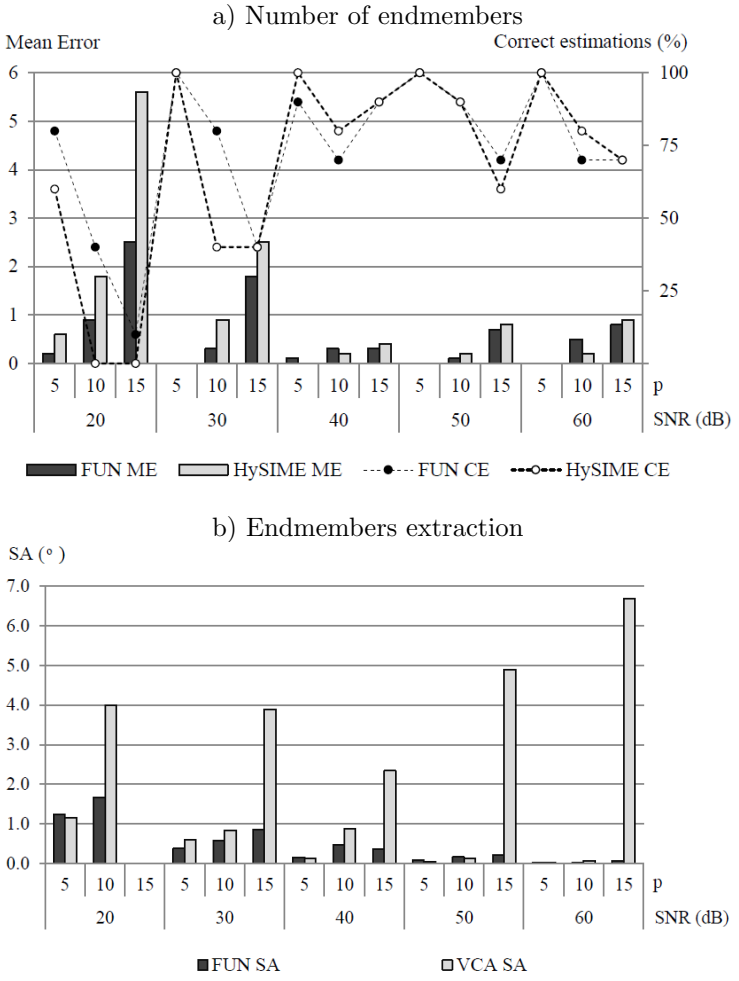


FIGURE 2.9: a) Number of endmembers estimation. Percentage of correct estimations and mean error obtained using the FUN and the HySIME algorithms. b) Endmembers extraction. Mean spectral angle obtained using the FUN and the VCA algorithms.

In order to estimate the quality of the results the *Root Mean Square Error* (RMSE) has been calculated in both cases, as follows:

$$\text{RMSE} = \frac{1}{N_p \cdot N_b} \cdot \sqrt{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (M_{i,j} - M'_{i,j})^2} \tag{2.10}$$

where M refers to the real hyperspectral image and M' refers to the reconstructed one, using the extracted endmembers and abundances, and N_p and N_b refers to the number of pixels and the number of bands in the hyperspectral image, respectively.

The spectral angle of the endmembers obtained by the FUN algorithm, which are used to calculate the abundances, has been also computed. The data have been grouped in sets of ten images, according to their characteristics, and the mean results are shown in Table

2.4, where RMSE is displayed for the FUN and FCLSU algorithms. Columns headed as *FCFUN* show the results obtained when the abundances are calculated with the FUN algorithm and subsequently the nonnegative and the sum-to-one constraints are applied, as explained in Section 2.5.

TABLE 2.4: RMSE values obtained with the FUN, the FC-FUN and the FCLSU algorithms

Image's characteristics		Mean spectral angle (°)	RMSE		
SNR (dB)	p		FUN	FCFUN	FCLSU
20	5	1.989	0.0354	0.0449	0.0363
	10	3.374	0.0330	0.0804	0.0404
	15	3.521	0.0343	0.0944	0.0409
30	5	0.629	0.0110	0.0195	0.0111
	10	1.898	0.0110	0.0282	0.0112
	15	3.320	0.0109	0.0584	0.0126
40	5	0.215	0.0033	0.0071	0.0033
	10	1.539	0.0037	0.0127	0.0037
	15	1.235	0.0035	0.0232	0.0035
50	5	0.046	0.0010	0.0017	0.0011
	10	1.208	0.0011	0.0100	0.0012
	15	1.030	0.0011	0.0178	0.0012
60	5	0.290	0.0004	0.0016	0.0005
	10	0.320	0.0003	0.0046	0.0004
	15	0.438	0.0003	0.0082	0.0004

We can observe that the FUN algorithm is able to estimate the abundances with a similar accuracy to the one provided by the reference FCLSU algorithm. These facts are graphically displayed in Figure 2.10, where it can be also seen that there is little or no difference in the accuracy of the results of the FUN algorithm with and without applying the nonnegative and the sum-to-one constraints. This means that the constraints are mostly satisfied inherently.

2.6.2 Experiments on real hyperspectral data

In order to test the proposed algorithm in a more realistic scenario, the AVIRIS Cuprite image has also been used in this work. This scene is well understood from the mineralogical point of view and consists of 224 spectral bands between 0.4 and 2.5 μm. Prior to the analysis, several bands have been removed due to water absorption and low SNR resulting in a total of 188 spectral bands. Particularly, we have used in our experiments a portion of the full Cuprite image, with spatial size of 350x350 pixels. Figure 2.11 shows a representation of this image.

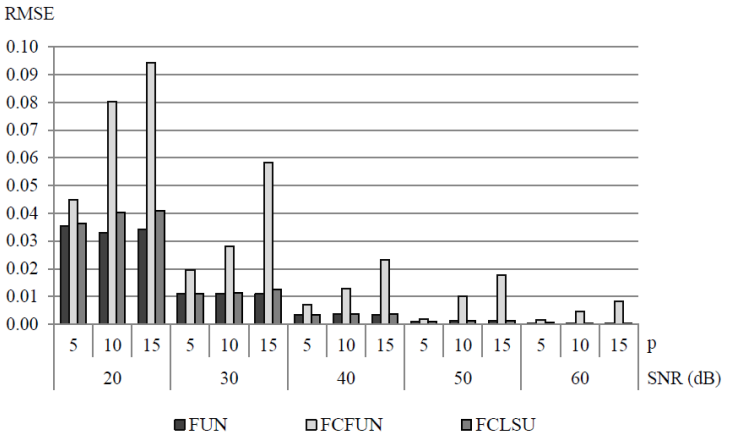


FIGURE 2.10: *Root Mean Square Error* (RMSE) obtained with the FUN, the FCFUN and the FCLSU algorithms in the simulations performed with a set of synthetic hyperspectral images with different characteristics of noise and with different number of endmembers.

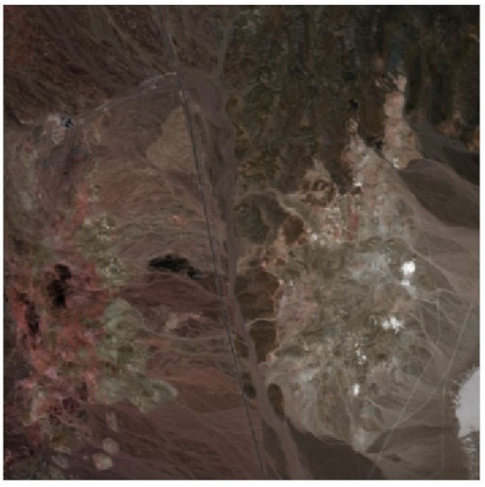


FIGURE 2.11: AVIRIS Cuprite image.

Ground truth information of the spectral signatures present in these images is available in the USGS library. The spectral signature of alunite, buddingtonite, calcite, kaolinite and muscovite, which are present in the Cuprite image, are compared with those obtained with the FUN algorithm in order to evaluate its endmember extraction accuracy.

The FUN algorithm, with $\alpha = 1$ as input parameter, is utilized to perform the entire unmixing process and the abundance estimation. The obtained results are compared with the HySIME, the VCA and the FCLSU, each for the particular stage of the unmixing process they were designed for.

Moreover, the execution time of each of the algorithms has been computed as an indicator of the computational complexity. Ten simulations have been performed for this portion

of the real AVIRIS Cuprite image. The results are shown in Table 2.5, in which we also show the number of endmembers estimated, and the spectral angle for the FUN and the reference algorithms. Specifically, the spectral angle value is the sum of the spectral angles obtained by comparing the endmembers extracted by the FUN algorithm and the VCA algorithm with the five available spectral signatures known in the AVIRIS Cuprite scene, which are alunite, buddingtonite, calcite, kaolinite and muscovite.

Table 2.5 also shows the time taken by the FUN algorithm in the different stages of the unmixing process, namely: the noise filtering (column *NF*), the estimation of the number of endmember and simultaneous extraction (*P+E*) and the abundance estimation (*A*). The column headed *Total* shows the time consumed by the algorithms to complete the unmixing process. Finally, column *RA* shows the total time taken by the reference algorithms to complete the unmixing process, as the combination of the HySIME algorithm, the VCA algorithm and the FCLSU algorithm.

Table 2.5: Comparison of the results obtained with the Cuprite dataset

P estimated		MSA		RMSE			Time consumed by the algorithms (s)							
							FUN			Reference algorithms			Total	
FUN	HS	FUN	VCA	FUN	FCFUN	FCLSU	N.F.	P+E	A	HS	VCA	FCLSU	FUN	RA
19	16	6.86	7.67	0.0018	0.0769	0.0067	3.8	2.7	0.1	4.4	3.4	199.6	6.5	207.5
19	16	6.86	7.43	0.0018	0.0769	0.0039	3.7	2.6	0.1	4.4	3.2	211.2	6.4	218.8
19	16	6.86	8.10	0.0018	0.0769	0.0073	3.7	2.6	0.1	4.5	3.1	199.2	6.5	206.8
19	16	6.86	7.63	0.0018	0.0769	0.0054	3.9	2.7	0.1	4.7	3.4	202.7	6.7	210.8
19	16	6.86	8.17	0.0018	0.0769	0.0058	3.7	2.7	0.1	4.7	3.3	188.6	6.5	196.6
19	16	6.86	7.72	0.0018	0.0769	0.0041	3.9	2.7	0.1	4.7	3.4	208.0	6.7	216.1
19	16	6.86	8.71	0.0018	0.0769	0.0073	3.9	2.8	0.1	4.6	3.3	167.0	6.8	174.9
19	16	6.86	8.00	0.0018	0.0769	0.0060	3.9	2.7	0.1	4.7	3.3	191.9	6.7	199.9
19	16	6.86	8.09	0.0018	0.0769	0.0049	3.9	2.7	0.1	4.5	3.3	203.2	6.7	211.0
19	16	6.86	8.59	0.0018	0.0769	0.0068	3.8	2.7	0.1	4.6	3.4	199.9	6.6	207.9

Table 2.6 shows the mean results obtained for the simulations displayed in Table 2.5, demonstrating the good performance of the FUN algorithm, since the spectral angle and the RMSE are smaller for the FUN algorithm than they are for the reference algorithms. Furthermore, the FUN algorithm has the additional advantage that performs the entire unmixing process more than 31 times faster than the reference algorithms.

Finally, one last simulation has been performed in order to graphically show the results obtained by both, the proposed algorithm and the reference algorithms. Table 2.7 shows the spectral angle scores obtained with the FUN algorithm and with the VCA algorithm

TABLE 2.6: Mean results obtained with the Cuprite dataset

P estimated		MSA		RMSE			Time consumed by the algorithms (s)							
							FUN			Reference algorithms			Total	
FUN	HS	FUN	VCA	FUN	FCFUN	FCLSU	N.F.	P+E	A	HS	VCA	FCLSU	FUN	RA
19	16	6.86	8.01	0.0018	0.0769	0.0058	3.8	2.7	0.1	4.6	3.3	197.1	6.6	205.0

when comparing the respective extracted endmembers with respect to the real pure spectral signatures present in the scene, while Figure 2.13 compares graphically the spectral signatures. Moreover, Figure 2.12 shows the *Root Square Error* (RSE) obtained for each pixel in the hyperspectral image as shown in Figure 2.11, using the proposed algorithm and the FCLSU algorithm. The square error of each pixel has been calculated comparing the real Cuprite image with the reconstructed Cuprite one. The reconstruction of the image is done using the extracted endmembers and their calculated abundances with both methods, according to the linear mixing model. These square errors correspond with the RMSE shown in Table 2.8.

$$\mathrm{RSE}_j = \frac{1}{N_b} \cdot \sqrt{\sum_{i=1}^{N_b} (M_{i,j} - M'_{i,j})^2} \tag{2.11}$$

TABLE 2.7: Comparison of the spectral signatures obtained with the FUN algorithm and the VCA algorithm

Algorithms	Spectral angle (°)				
	e_1	e_2	e_3	e_4	e_5
FUN	4.83	4.26	9.62	10.98	4.64
VCA	11.95	5.85	6.45	14.90	5.57

Reference spectral signatures	
Alunite	e_1
Buddingtonite	e_2
Calcite	e_3
Kaolinite	e_4
Muscovite	e_5

TABLE 2.8: RMSE obtained with the FUN algorithm, with the FCFUN algorithm and with the FCLSU algorithm

RMSE		
FUN	FCFUN	FCLSU
0.0018	0.0769	0.0058

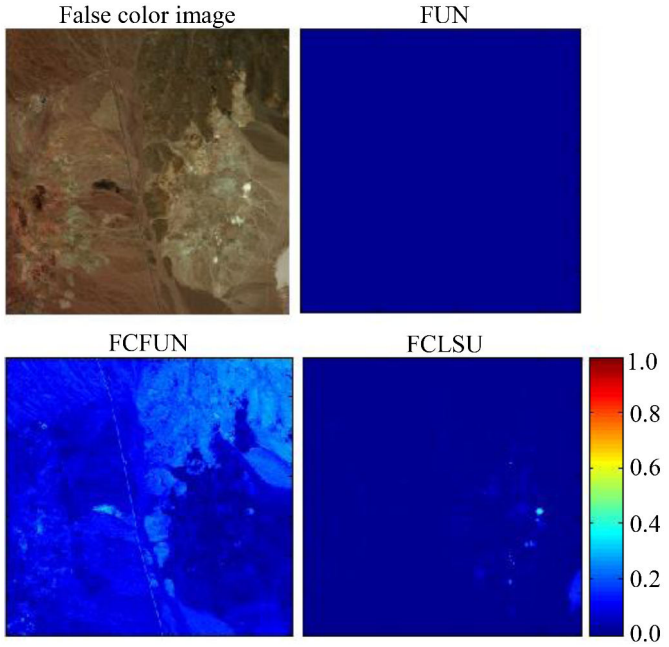


FIGURE 2.12: a) False colour representation of the AVIRIS Cuprite image used. b) RSE obtained with the FUN algorithm. c) RSE obtained with the FC-FUN algorithm. d) RSE obtained with the FCLSU algorithm.

2.7 Conclusions

A new algorithm, named FUN, for performing the unmixing process has been totally described in this Chapter. This algorithm is based on orthogonal projections, using the modified Gram-Schmidt method to perform them. This methodology allows the reutilization of information and a high level of parallelization. Moreover, the algorithm does not need to perform complex matrix operations, such as the inverse of a matrix or the extraction of eigenvalues and eigenvectors, which makes easier its ulterior hardware implementation.

The FUN algorithm performs simultaneously the number of endmembers estimation and their extraction. Once that the endmembers have been extracted, this algorithm is also capable of calculating the abundances by doing just some additional operations. In order to verify the well behaviour of the FUN algorithm for the different parts of the unmixing process, and for different scenarios, synthetic hyperspectral images have been generated. These images have been generated varying their abundances distribution, number of endmembers, endmembers, and signal to noise ratio. Moreover, in order to compare the results obtained with the FUN algorithm with some of the most relevant algorithms in

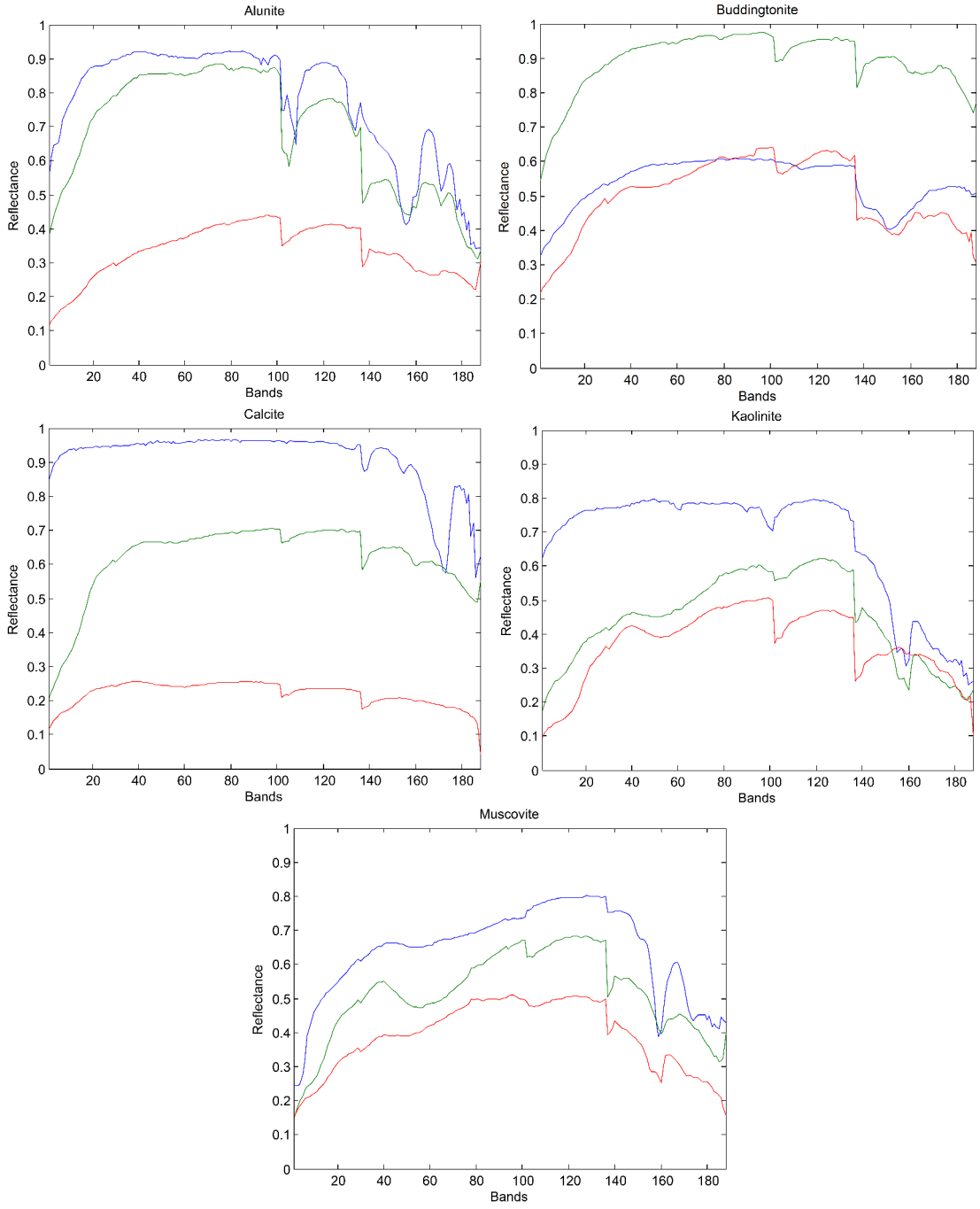


FIGURE 2.13: Comparison of the five spectral signatures in the Cuprite scene (in blue) with respect to the endmembers extracted by the FUN (in green) and the VCA (in red) algorithms.

the state-of-the-art for the same tasks, the entire unmixing process has also been performed using the unmixing chain formed by the HySIME algorithm, the VCA algorithm and the FCLSU algorithm for the same images.

The experimental results demonstrate that the accuracy obtained with the FUN algorithm when estimating the number of endmembers and extracting them is similar or better than the accuracy of the HySIME algorithm and the VCA algorithm. Moreover, the accuracy of the abundances obtained with the FUN algorithm is also similar to the accuracy of the abundances calculated with the FCLSU, while the FUN algorithm calculates them much faster.

Furthermore, in order to evaluate the FUN algorithm in a more realistic scenario, the AVIRIS Cuprite image has also been used. In this case, the proposed FUN algorithm has been able to reduce in a factor of more than 31 times the time required for processing it, while providing a better unmixing performance than traditional methods.

Hence, it is concluded that the FUN algorithm represents a more advantageous solution than the algorithms of the state-of-the-art, since this new algorithm allows a much simpler and faster hardware implementation, with simpler matrix operations which can be easily parallelized, providing accurate results. This contribution significantly facilitates the implementation of the unmixing process for applications which require real-time performance.

Chapter 3

Multispectral and hyperspectral image fusion

This Chapter presents the algorithm developed in this Thesis for fusing multispectral and hyperspectral images. This algorithm, named *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI) is compared with the algorithms that conform the state-of-the art for the fusion process using different methods and different real hyperspectral images. The results obtained in this work using different images demonstrate that the CoEf-MHI algorithm surpasses the state-of-the-art approaches in terms of efficiency, being this efficiency understood as the ratio between the quality of the fused image and the computational effort required to obtain such image.

3.1 Outline

Remote sensors collect data by measuring the energy reflected from the Earth's surface. Thanks to this process they are able to provide final users with images characterized by a different spatial resolution, spectral resolution and *Signal-to-Noise Ratio* (SNR). According to their spectral resolution, these images can be classified as panchromatic, multispectral, hyperspectral or even ultraspectral images. The spectral resolution of these different types of data varies from one spectral band for the panchromatic images, typically close to the visible spectral range of 0.4 to 0.8 μm; to hundreds or thousands of bands for the hyperspectral and ultraspectral images, which normally cover the visible and infrared ranges. However, the spatial resolution of panchromatic and multispectral remote sensing systems is typically much higher than the spatial resolution of the hyperspectral and ultraspectral remote sensing systems.

Hence, the main advantage of hyperspectral images with respect to multispectral ones is the more accurate spectral information they provide, which clearly benefits many applications such as unmixing, classification and/or target detection [16] [82]. Nevertheless, a potential improvement of the spatial resolution of hyperspectral images certainly benefits the results achieved for these applications. With this objective in mind, and due to the existence of remote sensing systems with both hyperspectral and multispectral sensors, many algorithms have been proposed in the last decade to perform the fusion of *Low spatial resolution HyperSpectral* (LHS) images and *High spatial resolution MultiSpectral* (HMS) images, in order to obtain a *High resolution HyperSpectral* (HHS).

These different fusion algorithms can be classified into *Component Substitution* methods (CS), *Multiresolution Analysis* (MRA), and more recent and complex methods based on Bayesian and matrix factorization approaches. CS-based fusion techniques typically provide a high fidelity in rendering the spatial details in the final image, a fast and easy implementation and a robustness to misregistration errors and aliasing. However, CS-based methods tend to generate a significant spectral distortion caused by the spectral mismatch between the hyperspectral and the multispectral image. MRA-based fusion methods typically provide spectral consistency and robustness to aliasing under proper conditions. However, the computational burden of the MRA-based methods is usually larger and their implementation is more complicated due to the design of spatial filters.

Finally, Bayesian based methods and matrix factorization approaches for fusing multispectral and hyperspectral images provide more accurate results in exchange of an increase of their complexity and computational burden.

In this Thesis, a new algorithm for data fusion named *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI) is proposed. This algorithm introduces the spatial details of the HMS image into the LHS image in order to obtain a HHS image. In order to do this, the CoEf-MHI algorithm independently approximates each pixel of the HHS image involving only a few pixels and very simple mathematical operations, which allows to dramatically accelerate the entire process. These characteristics, together with the performance achieved by the CoEf-MHI algorithm in terms of well accepted evaluation metrics for fusion algorithms, distinguish our proposal as the most efficient algorithm within the state-of-the-art, being the efficiency measured as the ratio between the quality of the fused image and the number of operations required to obtain such image.

3.2 State-of-the-art in multispectral and hyperspectral images fusion

Many relevant algorithms have been proposed for fusing hyperspectral images. A particular subset of these algorithms can be categorized as hyperspectral pansharpening algorithms, which are focused on the fusion of a panchromatic and a hyperspectral image. An excellent revision of these algorithms have been published in [83]. However, in this work we have focused our attention in those state-of-the-art proposals that have been originally conceived for the fusion of a multispectral and a hyperspectral image. Accordingly, we have selected as the most significant fusion proposals up to now the *Generalized Gram-Schmidt Adaptive* (GGSA) algorithm [84], the *Generalized Local Minimum Square Error* (GLMMSE) algorithm [84], the *Coupled Nonnegative Matrix Factorization* (CNMF) algorithm [85], the *Hyperspectral Superresolution* (HySure) algorithm [86] and the Bayesian Naive (BN) and Bayesian Sparse (BS) algorithms [87]. Nevertheless, as far as panchromatic images can be seen as particular bands of multispectral images, it is also worth to mention that algorithms [85], [86] and [87] have been also pointed out in [83] as the best options in terms of the quality of the fused image for hyperspectral pansharpening. Due

to this reason, these algorithms have been selected by us, together with the GGSA and the GLMMSE algorithms, for comparing their performance against our proposal.

The GGSA and the GLMMSE algorithms are generalized versions of the *Gram-Schmidt Adaptive* (GSA) [88] and the *Local Minimum Square Error* (LMMSE) [89] pansharpening algorithms respectively, which perform the fusion between a high spatial resolution panchromatic image and a multispectral image with a lower spatial resolution. Both the GGSA, which can be classified as a CS method, and the GLMMSE algorithms, which can be classified as a MRA method, first divide the spectrum covered by the hyperspectral image into several regions, according with the regions covered and uncovered by the multispectral image. Then, they artificially generate a multispectral band per uncovered region, applying *Ratio Image-Based Spectral Resampling* (RIBSR) [90]. After that, each group of hyperspectral bands are associated to each multispectral band, and the fusion is independently performed for each group, using the GSA and the LMMSE algorithm, respectively. The quality of the results obtained by these algorithms mainly depends on the behaviour of the pansharpening algorithms used, GSA and LMMSE, and so does the computational burden.

The CNMF algorithm is a factorization aproach of the fusion process. This method, as other data fusion algorithms, is based on unsupervised unmixing. In this aproach, the LHS image and the HMS image are alternatively unmixed by the *Nonnegative Matrix Factorization* (NMF) algorithm [78] [91]. By combining the hyperspectral endmember matrix and the high spatial resolution abundances matrix obtained from the HMS image, a *high spatial resolution hyperspectral* (HHS) image is generated. The NMF algorithm decomposes a set of highly dimensional data into two nonnegative matrices and finds "basis vectors" by which the original data are approximated through linear combination. Using this algorithm to unmix hyperspectral data, these two nonnegative matrices are the endmembers and the abundances matrices. It is a suitable unmixing algorithm for hyperspectral data sets, because it guarantees the nonnegativity of the results and needs no assumption for the presence of pure pixels. Generally, the NMF algorithm performs, iteratively, techniques derived on the basis of the traditional gradient descent method to reduce the error of the factorization of the original data into the endmembers and abundances matrices. In particular, the CNMF algorithm used as reference for data fusion in this work, first performs an unsupervised unmixing chain using the *Virtual Dimensionality* (VD) [53] algorithm, the *Vertex Component Analysis* (VCA) [56] algorithm, and the *Fully Constrained Least Square linear spectral Unmixing* (FCLSU) [71] algorithm. Using

the results obtained with this unmixing chain, the CNMF algorithm uses the NMF algorithm to sequentially optimize the endmembers obtained from the LHS image and the abundances obtained from the HMS image in order to improve the fusion results. This process leads to a huge computational burden, due to the necessity of the execution of the required algorithms to perform the entire unmixing chain to initialize the NMF algorithm, and the iterative process of the NMF algorithm itself.

The HySure algorithm, the BN algorithm and the BS algorithm follow a Bayesian approach. The Bayesian inference framework allows a convenient formulation of the fusion of hyperspectral and multispectral images. This formulation uncovers an intuitive interpretation of the fusion process via the posterior distribution of the Bayesian fusion model that typically leads to better results than the more traditional CS and MRA methodologies, with the negative aspect that its complexity and computational burden also increase.

3.3 Proposed fusion strategy: the CoEf-MHI algorithm

In this work, a *Low resolution HyperSpectral* (LHS) image and a *High resolution MultiSpectral* (HMS) image are assumed to be available for fusion. It is also assumed that the two images are coregistered and that the spectral responses of the multispectral sensor and the hyperspectral sensor are known. Therefore, the problem handled in this Thesis is to estimate a *High spatial resolution HyperSpectral* (HHS) image, containing the spectral features of the LHS image and the spatial features of the HMS image. Moreover, as reported in [85], the observed HMS image and LHS image can be viewed as spectrally and spatially degraded versions of the HHS image.

The LHS image can be upsampled to the spatial resolution of the HMS image, using an interpolator, obtaining an *upsampled hyperspectral* (ULHS) image. According to the spectral responses of both sensors, a *simulated multispectral* (USMS) image can be obtained from the ULHS image with the same bands than the HMS image. If these two images are compared, the simulated USMS image, and the HMS image, should be ideally the same. However, this does not happen, due to the fact that part of the spatial information contained in the HMS image is not present in the ULHS image, from which the USMS image is obtained. Figure 3.1 graphically shows the different images (HMS, LHS, ULHS, USMS and HHS images) involved in the fusion process within the CoEf-MHI algorithm.

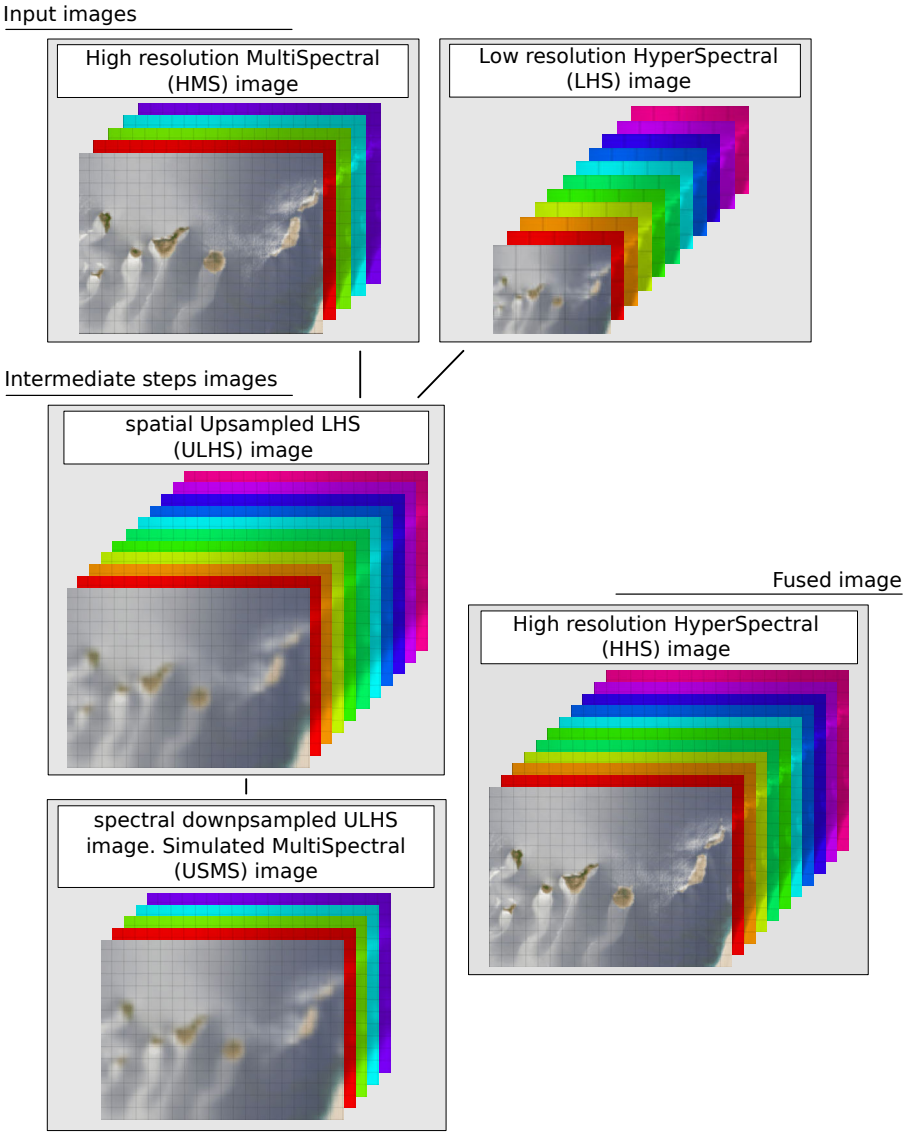


FIGURE 3.1: Images involved in the fusion process using the CoEf-MHI algorithm

The proposed method upsamples the LHS image to the spatial resolution of the HMS image, obtaining the ULHS image, and then incorporates the spatial details of the HMS image into the ULHS image, without introducing spectral distortions. In order to do this, the USMS image is obtained from the ULHS image, based on the spectral responses of both the hyperspectral and the multispectral sensor. After this, the proposed method initiates a process which is totally independent for each pixel of the image. The process independently performed for each pixel of the image is divided in the following three steps:

1. First of all, each pixel of the USMS image is compared with its corresponding pixel of the HMS image, extracting a vector which contains the differences between them.

2. Secondly, the vector which contains the differences between the two pixels is approximated as a linear combination of the pixels of the USMS image that surround the pixel under analysis. It is important to highlight that in order to approximate one single vector which represents the differences between a certain pixel of both the USMS image and the HMS image, a little number of vectors is needed. This vector is typically perfectly approximated using less than three pixels within the CoEf-MHI algorithm.

3. Finally, the corresponding pixel of the ULHS image is modified, by adding the result of the same linear combination performed before, but using the corresponding pixels of the ULHS image.

In order to identify which pixels best fits the linear combination that approximates the vector which contains the differences between a certain pixel of the USMS and HMS images, the modified Gram-Schmidt method is used, as it is done in the *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), which has been described in Chapter 2 of this Thesis. The modified Gram-Schmidt method used has been described in Section 2.4.1.

The entire process is deeply explained in subsections 3.3.1 and 3.3.2.

3.3.1 Process applied to the entire image within the CoEf-MHI algorithm

Before starting the process which can be independently applied to each pixel of the image, some operations must be performed. First of all, the parameter r which represents the ratio between the linear spatial resolution of the HMS and LHS images, is computed as:

$$r = int(\frac{\text{HMS linear spatial resolution}}{\text{LHS linear spatial resolution}}) \tag{3.1}$$

Once that the parameter r has been calculated, the LHS image is upsampled to the spatial resolution of the HMS image, obtaining the ULHS image, using a bilinear interpolation. Finally, the USMS image is obtained by spectrally degrading the ULHS image, based on the spectral responses of both the hyperspectral and the multispectral sensor.

3.3.2 Process applied to each individual pixel within the CoEf-MHI algorithm

Once that the ULHS and USMS images have been obtained, each pixel of the image is independently evaluated and modified. In order to obtain each pixel (i, j) of the objective HHS image, the corresponding (i, j) pixel of the ULHS, USMS and HMS images are selected, and named *Objective Pixels* (OP), obtaining the pixels OP_{ULHS}, OP_{USMS} and OP_{HMS}, respectively. Secondly, the pixels which surround the (i, j) pixel in the ULHS and USMS images are selected and named *Surrounding Pixels* (SP), obtaining the sets of pixels SP_{ULHS} and SP_{USMS}, respectively. Each of these sets of (x, y) pixels are selected according with the following restrictions:

$$\begin{aligned} (i-r) \leq x \leq (i+r) \\ (j-r) \leq y \leq (j+r) \\ (x, y) \neq (i, j) \end{aligned} \tag{3.2}$$

For each certain pixel (i, j) of the HHS image, once that its corresponding pixels OP_{ULHS}, OP_{USMS} and OP_{HMS}, and its corresponding sets of pixels SP_{ULHS} and SP_{USMS} have been defined, the proposed method first evaluates if pixel (i, j) under analysis needs to be modified in order to introduce the spatial details of the HMS image. If needed, the modification process explained in sections 3.3.2.1, 3.3.2.2 and 3.3.2.3 is applied.

3.3.2.1 Selection of the projection vectors

First, the algorithm compares the OP_{USMS} with the OP_{HMS} and extract how much information of OP_{HMS} is missed in OP_{USMS}. Secondly, the proposed algorithm selects the minimum number of pixels from SP_{USMS} which best represents the missing information. The pseudocode that describes this process is presented in Algorithm 3.1. First of all, in line 4, the information of OP_{HMS} which is missed in OP_{USMS} is calculated and saved in the vector Dif. In line 9, the information of each pixel from SP_{USMS} which is not contained in OP_{USMS} is calculated. In line 13, the pixel of SP_{USMS} which has more information not contained in OP_{USMS} is selected and saved in the matrix PM. Its corresponding pixel from SP_{ULHS} is saved in matrix PH in line 14. In line 16, the algorithm subtracts the

information from the vector Dif that can be represented with the selected pixel. This process is repeated until the vector Dif becomes null, or until the number of extracted vectors reach the number of bands nb of the HMS image minus one, as shown in line 7.

It may happen that OP_{USMS} contains all the information present in OP_{HMS}, and hence, no more vectors are needed. This typically occurs for very homogeneous zones of the image in which the bilinear interpolation perfectly obtains the spatial information missed in the LHS image with respect to the HMS image. For these cases, $p = 1$, and lines 8 to 18 of the pseudocode are not executed.

Algorithm 3.1: Selection of the vectors needed to reconstruct the hyperspectral pixel introducing the spatial details

Inputs:
OP_{ULHS} {Objective pixel from the ULHS image};
OP_{USMS} {Objective pixel from the USMS image};
OP_{HMS} {Objective pixel from the HMS image};
SP_{ULHS} {Surrounding pixels from the ULHS image};
SP_{USMS} {Surrounding pixels from the USMS image};

1 $pm_1 = OP_{USMS}$;
2 $ph_1 = OP_{ULHS}$;
3 $u_1 = OP_{USMS}/\|OP_{USMS}\|$;
4 $Dif = OP_{HMS} - (OP_{HMS} \circ u_1) \cdot u_1$;
5 $SP_{USMS2} = SP_{USMS}$;
6 i = 2;
7 **while** $((i < nb - 1)$ *and* $(\|Dif\| > 0)$ **do**
8 | **for** $j = 1$ **to** N_p **do**
9 | | $SP_{USMS2_j} = SP_{USMS2_j} - (SP_{USMS2_j} \circ u_{i-1}) \cdot u_{i-1}$;
10 | | $s_j = \|SP_{USMS2_j}\|$;
11 | **end**
12 | $j_{\max} = \arg\max(s_j)$;
13 | $pm_i = OP_{USMS_{j_{\max}}}$;
14 | $ph_i = OP_{ULHS_{j_{\max}}}$;
15 | $u_i = SP_{USMS2_{j_{\max}}}/\|SP_{USMS2_{j_{\max}}}\|$;
16 | $Dif = Dif - (Dif \circ u_i) \cdot u_i$;
17 | $i = i + 1$
18 **end**
19 $p = i - 1$;

Outputs:
$PM = [pm_1, pm_2, ..., pm_p]$, {Multispectral selected vectors}
$PH = [ph_1, ph_2, ..., ph_p]$, {Hyperspectral selected vectors}

3.3.2.2 Coefficient calculation

Once that the minimum number of pixels from SP_{USMS} which best represent the missing spatial details have been selected and saved in $PM = [pm_1, pm_2, ..., pm_p]$, and the corresponding pixels from OP_{ULHS} have been saved in $PH = [ph_1, ph_2, ..., ph_p]$, the CoEf-MHI algorithm calculates the coefficients which represent how much of each pixel must be used in the linear combination in order to reconstruct the objective pixel from the HHS image. The pseudocode that describes this process is presented in Algorithm 3.2.

In order to calculate these coefficients, the algorithms performs the Gram-Schmidt orthogonalization to the $PM = [pm_1, pm_2, ..., pm_p]$ selected pixels. This process is repeated p times, ending each time with a different pixel (lines 5 to 15 of the pseudocode). In line 17 the last pixel processed in each of the p iterations is saved in the matrix U^*. Finally, the coefficient corresponding to each pixel from $PM = [pm_1, pm_2, ..., pm_p]$ is calculated in line 21 and saved in the A vector.

Algorithm 3.2: Calculation of the coefficients assigned to the selected vectors to reconstruct the hyperspectral pixel introducing the spatial details

Inputs: $PM = [pm_1, pm_2, ..., pm_p]$, p, OP_{HMS}

```
1  if p > 1 then
2      PM = [PM, PM];
3      Q = [], U = [], U* = [];
4      for k = 2 to p + 1 do
5          q_1 = PM_k;
6          u_1 = PM_k / (PM_k ∘ PM_k);
7          for j = 2 to p do
8              x = PM_{k+j-1};
9              for i = 1 to j - 1 do
10                 x = x - (x ∘ q_i) · u_i;
11             end
12             q_j = x_j;
13             u_j = x_j / (x_j ∘ x_j);
14             Q = [Q, q_j];
15             U = [U, u_j];
16         end
17         U* = [U*, u_p];;
18         Q = [];
19         U = [];
20     end
21     A = U*^T · OP_HMS;
22 end
```

Outputs: A

3.3.2.3 Hyperspectral pixel reconstruction

Finally, once that the coefficients corresponding to each pixel of $PM = [pm_1, pm_2, ..., pm_p]$ have been calculated, these coefficients are directly applied to their corresponding pixels stored in $PH = [ph_1, ph_2, ..., ph_p]$ in order to perform the linear reconstruction of the pixel under analysis of the HHS image. This process is described in line 2 of the pseudocode described in Algorithm 3.3.

Algorithm 3.3: Reconstruction of the hyperspectral pixel introducing the spatial details

Inputs: $PH = [ph_1, ph_2, ..., ph_p]$, p, A

```
1 if p > 1 then
2 |  RP = PH · A;
3 else
4 |  RP = PH;
5 end
```

Outputs: RP {Reconstructed hyperspectral pixel}

3.3.3 Computational advantages of the proposed CoEf-MHI algorithm

There exist many algorithms for fusing multispectral and hyperspectral images. However, most of the existing algorithms require a formidable computational burden, what makes very difficult to achieve an efficient hardware implementation for real-time applications.

The proposed CoEf-MHI algorithm represents an important advantage with respect to the state-of-the-art algorithms. First of all, there are not complex matrix operations in the proposed method, making it easier to achieve a hardware implementation. Secondly, the fusion process is independently performed for each pixel of the objective HHS image, involving just a couple of multispectral and hyperspectral pixels in the process, as explained in Section 3.3.1. In fact, once that the process to be performed for a certain pixel is defined, it can be replicated as many times as desired and performed in parallel. This fact enables the achievement of a parallel and scalable harware implementation of the proposed algorithm. Finally, the computational burden of the CoEf-MHI algorithm for fusing hyperspectral and multispectral images is small when compared with the computational burden of the state-of-the art algorithms for this task, as it will be demonstrated in Section 3.5 in terms of the total amount of floating point operations performed as well as the total allocated memory required for performing the fusion of the images. This issue

distinguishes our proposal as the most suitable fusion algorithm for being implemented onto high-performance parallel computing systems, which are mandatory for real-time hyperspectral imaging applications.

3.4 Data and evaluation metrics

3.4.1 Evaluation method: Wald's protocol

A general paradigm for quality assessment of fused images that is commonly accepted is the Wald's protocol [92]. This paradigm is based on two properties that the fused data must have, as much as possible, namely consistency and synthesis properties. The first property requires the reversibility of the fusion process, in the sense that the original LHS should be obtained by properly degrading the fused image. The second property requires the fused image to be as similar as possible to the image of the same scene that would be obtained, by the same sensor, at the higher resolution. This condition entails that both the features of each single band and the mutual relations among bands have to be preserved.

Wald's protocol is used in this work for assessing the quality of the fusion methods, as depicted in Figure 3.2. According to this method, the observed LHS and HMS images are synthetically generated from a *Reference HyperSpectral* (RHS) image, and then, the fusion results are evaluated against that reference image. This process consists in three steps:

1. Given a hyperspectral image to be used as a reference, RHS image, a simulated observed *Low resolution HyperSpectral* (LHS) image is obtained. This process has been done in two different ways within this work. The first one consists in applying a Gaussian blurring to the RHS image and then downsampling the result by selecting one out of every r pixels, in both the horizontal and vertical directions, where r denotes the downsampling factor. The second way used to reduce the spatial resolution of the RHS image is by directly applying a bilinear decimation to this image. The amount of Gaussian blurring applied to the image considered in this work is controlled by the deviation of the normal distribution, which has been set to the same value for both directions. The combination of the different parameters and degradation methods leads to the generation of different LHS images for each

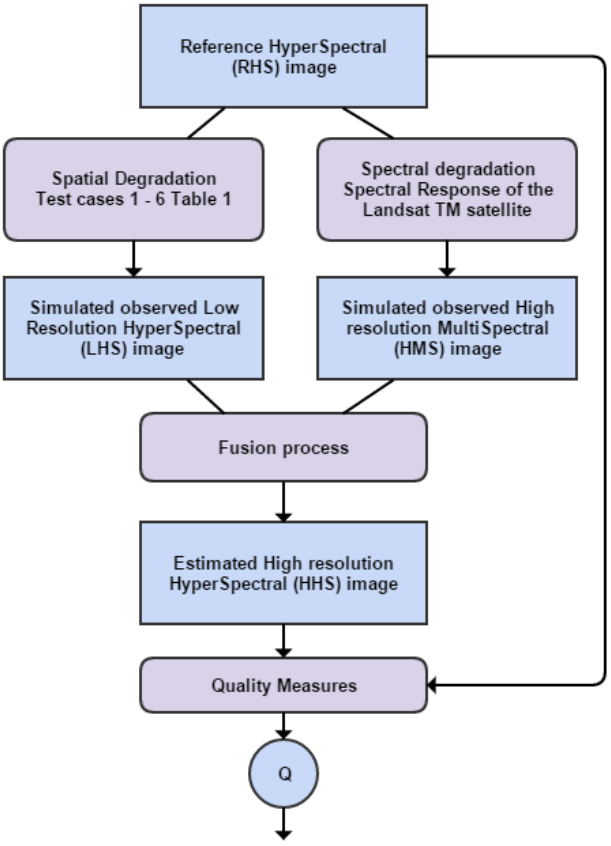


FIGURE 3.2: Flow diagram of the experimental methodology, derived from Wald's protocol.

RHS image, making it possible to have different test cases for each RHS image, as summarised in Table 3.1.

2. A simulated observed *High resolution MultiSpectral* (HMS) image is obtained by multiplying the RHS image by a suitably chosen spectral response vector. The vector used in this work emulates the spectral response of the Landsat TM satellite, which carries a multispectral sensor.

3. The fusion method to be evaluated is applied to the simulated observed LHS and HMS images, yielding the estimated *High resolution HyperSpectral* (HHS) image.

4. Finally, the estimated HHS image and the reference one are compared, to obtain quantitative quality measures.

3.4.2 Reference hyperspectral data

The proposed CoEf-MHI algorithm has been firstly evaluated on different HMS and LHS images obtained from real hyperspectral images. The first hyperspectral image has been taken over Washington DC by the *Hyperspectral Digital Imagery Collection Experiment* (HYDICE) [93] sensor in 1995, with 191 spectral bands in the 0.4 to 2.5 μm region. The second image was collected by the *Airbone Visible/Infrarred Imaging Spectrometer* (AVIRIS) [5] sensor in 1996 over Indian Pines in North-western Indiana. This scene consists on 220 spectral bands in the wavelength range of 0.4 to 2.5 μm. Bands [104-108], [150-163] and 220 have been removed due to water absorption, resulting in a total of 200 hyperspectral bands. The third image used was also taken by the AVIRIS sensor over the Salinas scene over Salinas Valley, California. This scene consists on 224 spectral bands in the wavelength range of 0.4 to 2.5 μm. The bands [108-112], [154-167] and 224 were also removed from the original image, resulting in a total of 204 bands. The last image used was taken over the region of Cuprite, Nevada, by the AVIRIS sensor too. Several bands have been removed due to water absorption and low SNR resulting in a total of 188 spectral bands.

This set of images represent a heterogeneous test bench, which allows to study the behaviour of the different algorithms under different situations. The Washington DC image, collected by the HYDICE sensor has a spatial resolution of 1 meter, being the highest spatial resolution of this set. The Cuprite and Salinas images collected by this sensor have around 4 meters of spatial resolution, while Indian Pines image has a spatial resolution of 20 meters, being the lowest spatial resolution of this testing set. Moreover, these images are collected over different scenes, having different spatial characteristics as it can be seen in Figure 5.5. Washington DC image represents an urban zone. This image contains buildings, roads, and many other forms with sharp edges which are lost in the spatial degradation and must be restored by the fusion process. Cuprite image is a scene captured over a mineral zone in which exist many different materials. These materials are mixed at subpixel level in many parts of the image, specially in the image which is spatially degraded, which represents a good test for the fusion process. Finally, Salinas and Indian Pines images are scenes captured over farming zones with much more homogeneous zones.

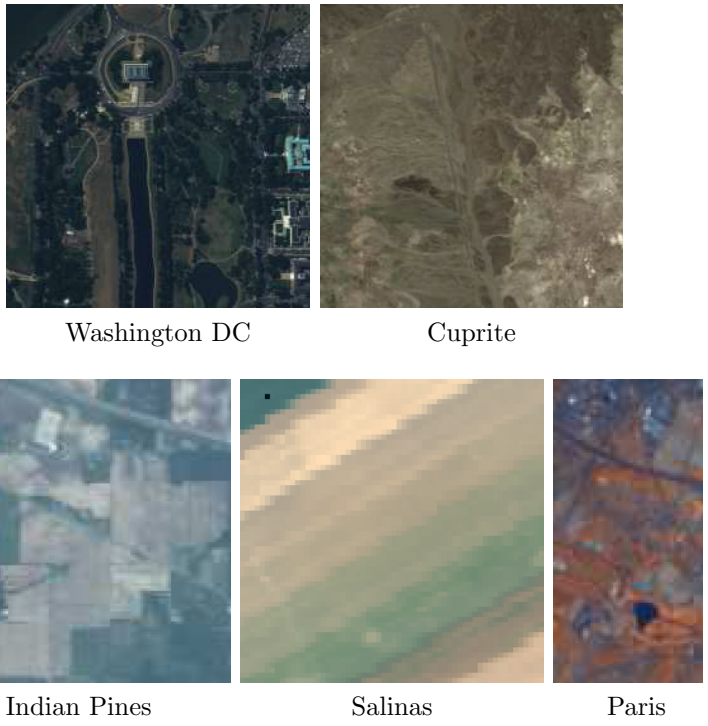


FIGURE 3.3: RGB representation of the different hyperspectral images used for the simulations.

In order to evaluate the goodness of the proposed CoEf-MHI algorithm with respect to the most significant state-of-the-art works, a portion of each of these real data sets has been used. In particular, a portion of 240x240 pixels has been selected from the Cuprite image and from the Washington DC image. From the Indian Pines, a portion of 120x120 pixels has been selected, and a portion of 60x60 pixels has been selected from the Salinas image. These images have been labelled as the *Reference HyperSpectral* (RHS) images for the simulations. The *Low resolution HyperSpectral* (LHS) image and the *High resolution MultiSpectral* (HMS) image have been obtained from the RHS images by downsampling these images in the spectral and spatial domains respectively, according to the Wald's protocol. Concretely, six LHS images are produced for each RHS image, obtaining the different test cases tabulated in Table 3.1. One HMS image is also produced for each case, with uniform spectral response functions corresponding to the Landsat TM bands 1-5 and 7, which cover the [0.45-0.52], [0.52-0.60], [0.63-0.69], [0.76-0.90], [1.15-0.175] and [2.08-2.35] μm. This process provides 6 LHS and 1 HMS images per each RHS image, resulting in a total of 24 test cases to evaluate and compare the goodness of our proposal.

Furthermore, in order to evaluate the proposed CoEf-MHI algorithm under a more realistic scenario, the Paris dataset has also been used. This dataset consist of a hyperspectral

and a multispectral image that were collected by two different sensors over the same zone. The hyperspectral image was collected by the Hyperion sensor, which provides 220 bands from 0.4 to 2.5 μm, with 30 meters of spatial resolution. The multispectral image was captured by the *Advanced Land Imager* (ALI) sensor, which provides 9 bands with 30 meters of spatial resolution, covering the [0.433 - 0.453], [0.45 - 0.515], [0.525 - 0.605], [0.63 - 0.69], [0.775 - 0.805], [0.845 - 0.89], [1.2 - 1.3], [1.55 - 1.75] and [2.08 - 2.35] μm regions of the electromagnetic spectrum. Both sensors were carried by the EO-1 satellite. Since both the hyperspectral and the multispectral image have the same spatial resolution, the hyperspectral image has been just spatially degraded, performing half of the Wald's protocol. Moreover, the images were coregistered before performing the fusion process and the portion of 120x60 pixels showed in Figure 5.5 were selected for the fusion procedure.

TABLE 3.1: Spatial degradation methods for each RHS image

Test case	Degradation method	Downsampling factor, r
1	Gaussian blurring. Deviation = 2	4
2	Gaussian blurring. Deviation = 2	6
3	Gaussian blurring. Deviation = 1	4
4	Gaussian blurring. Deviation = 1	6
5	Bilinear decimation	4
6	Bilinear decimation	6

3.4.3 Reference algorithms

The CoEf-MHI algorithm has been compared against the most relevant algorithms for hyperspectral and multispectral data fusion in terms of the quality of the fused images and/or in terms of the computational effort required for the fusion process. As it has been mentioned in Section 3.2 these algorithms are the *Generalized Gram-Schmidt Adaptive* (GGSA) algorithm, the *Generalized Local Minimum Square Error* (GLMMSE) algorithm, the *Coupled Nonnegative Matrix Factorization* (CNMF) algorithm, the *Hyperspectral Superresolution* (HySure) algorithm, the *Bayesian Naive* (BN) algorithm and the *Bayesian Sparse* (BS) algorithm.

3.4.4 Metrics used to evaluate the quality of the fused images

In order to compare the goodness of the different fusion methods, in terms of the spatial and spectral accuracy of their results, different quality measures have been used. In

this context, it is important to highlight that many different quality measures have been defined in the literature in order to determine the similarity between the fused *High resolution HyperSpectral* (HHS) image and the *Reference HyperSpectral* (RHS) image. These metrics are commonly classified into three categories, depending on whether they attempt to measure the spatial, spectral or global quality of the estimated image. The most widely accepted quality measures have been used within this work, namely the *cross correlation* (CC) [83], which is a spatial measure, the *spectral angle mapper* (SAM) [94], which is a spectral measure, and the *Peak Signal-to-Noise Ratio* (PSNR) [85] and *erreur relative globale adimensionnelle de synthese* (ERGAS) [95], which are global measures.

The *cross correlation* (CC), which characterizes the geometrical distortion, is defined as:

$$\mathrm{CC}(\mathrm{F},\mathrm{RHS}) = \frac{1}{N_b}\cdot\sum_{i=1}^{N_b}\mathrm{CCS}(\mathrm{F}_i,\mathrm{RHS}_i) \tag{3.3}$$

where N_b represents the number of bands of the RHS image, F represents the fused image, and *CCS* refers to the *Cross Correlation* index for a single-band image, which is defined as:

$$\mathrm{CCS}(\mathrm{F}_i,\mathrm{RHS}_i) = \frac{\sigma_{\mathrm{F}_i,\mathrm{RHS}_i}}{\sigma_{\mathrm{F}_i}\cdot\sigma_{\mathrm{RHS}_i}} \tag{3.4}$$

where $\sigma_{\mathrm{F}_i\mathrm{RHS}_i}$ refers to the covariance between the ith band of the fused image and the RHS image, and $\sigma^2_{\mathrm{F}_i}$ and $\sigma^2_{\mathrm{RHS}_i}$ refer to the variances of the ith band of the fused image and the RHS image, respectively. The ideal value for the *CC* index is 1.

The *Spectral Angle Mapper* (SAM), which is a spectral measure, is defined as:

$$\mathrm{SAM}(\mathrm{F},\mathrm{RHS}) = \frac{1}{N_p}\cdot\sum_{j=1}^{N_p}\mathrm{SA}(\mathrm{F}_j,\mathrm{RHS}_j) \tag{3.5}$$

where F_j and RHS_j refer to the jth pixel of the fused and RHS image, respectively. *SA* represents the spectral angle between two hyperspectral pixels, and is defined as:

$$\mathrm{SA}(\mathrm{F}_j,\mathrm{RHS}_j) = \arccos\frac{\mathrm{F}_j\circ\mathrm{RHS}_j}{\|\mathrm{F}_j\|\cdot\|\mathrm{RHS}_j\|} \tag{3.6}$$

where "$\circ$" represents the scalar product between two vectors, and "$\|\cdot\|$" represents the l_2 norm of these vectors. The ideal value for the SAM index is 0.

The *Peak Signal-to-Noise Ratio* (PSNR), which is a global measure, is defined as:

$$\mathrm{PSNR}(\mathrm{F},\mathrm{RHS}) = 10 \cdot \log_{10}(\frac{\mathrm{MAX}^2}{\mathrm{MSE}}) \tag{3.7}$$

where MAX refers to the maximum pixel value of the RHS image and MSE represents the *Mean Square Error*, of the fused image with respect to the RHS image, which is calculated as:

$$\mathrm{MSE} = \frac{1}{N_b \cdot N_p} \sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (\mathrm{F} - \mathrm{RHS})_{i,j}^2 \tag{3.8}$$

The quality of the fusion is higher as higher is the value of the PSNR index.

The *Erreur Relative Globale Adimensionnelle de Synthese* (ERGAS), which is a global measure, is defined as:

$$\mathrm{ERGAS}(\mathrm{F},\mathrm{RHS}) = \frac{100}{r} \cdot \sqrt{\frac{1}{N_b} \sum_{i=1}^{N_b} \frac{{\mathrm{RMSE}_i}^2}{{\mu_i}^2}} \tag{3.9}$$

where r is the ratio between the linear spatial resolution of the HMS and LHS images, as defined in (3.1), $RMSE_i$ is the *Root Mean Square Error* of the ith band of the fused image with respect to the ith band of the RHS image, defined in (3.10), and μ_i is the average value of the ith band of the RHS image.

$$\mathrm{RMSE}_i = \sqrt{\frac{1}{N_p} \sum_{j=1}^{N_p} (\mathrm{F} - \mathrm{RHS})_{i,j}^2} \tag{3.10}$$

The ideal value for the ERGAS index is 0.

3.4.5 Metrics used to evaluate the computational complexity and the computational efficiency of the algorithms

In order to evaluate the computational complexity of each of the different algorithms employed in the comparison, the total number of *floating point operations* (FLOPs) has been counted for such algorithms, as well as the *total allocated memory* (MEM) in Kb. Both the number of FLOPs and MEM have been measured using Matlab. The MEM has been measured using the own Matlab profiling tool, while the number of FLOPs has been measured using the tool provided in [96].

To ease the comparison of the results, the total number of FLOPs as well as the allocated memory have been represented in a logarithmic base (LFLOPs and LMEM), as shown in (3.11) and (3.12), since both of them tend to be very high.

$$\text{LFLOPs} = \log_{10}(\text{FLOPs}) \tag{3.11}$$

$$\text{LMEM} = \log_{10}(\text{MEM}) \tag{3.12}$$

Moreover, four more different metrics have been defined in this work, named FSAM, FPSNR, MSAM and MPSNR. These metrics relate the quality of the fused image to the total number of FLOPs (FSAM and FPSNR) and to the total allocated memory (MSAM and MPSNR) required to obtain them. Hence, these four metrics provide an efficient way to measure the efficiency of the fusion process. In particular, the FSAM and MSAM relates the *spectral angle mapper* (SAM) to the total number of FLOPs and to the total allocated memory, respectively, as shown in (3.13) and (3.14), while the FPSNR and the MPSNR relate the *peak Signal-to-Noise Ratio* (PSNR) to the total number of FLOPs and to the total allocated memory, respectively, as shown in (3.15) and (3.16).

$$\text{FSAM} = \frac{1}{\text{SAM} \cdot \text{LFLOPs}} \tag{3.13}$$

$$\text{MSAM} = \frac{1}{\text{SAM} \cdot \text{LMEM}} \tag{3.14}$$

$$\text{FPSNR} = \frac{\text{PSNR}}{\text{LFLOPs}} \tag{3.15}$$

$$\text{MPSNR} = \frac{\text{PSNR}}{\text{LMEM}} \tag{3.16}$$

Consequently, high values of FSAM, FPSNR, MSAM and MPSNR indicate a high efficiency of the fusion algorithm under evaluation.

3.5 Results

Tables 3.2 to 3.31 show the results obtained in the simulations performed for the Salinas, Cuprite, Indian Pines, Washington DC and Paris images, spatially degraded using the different degradation methods indicated in Table 3.1.

In order to better compare the quality of the fused images obtained by each of the algorithms under evaluation, Figure 3.4 graphically shows the mean values obtained for the SAM and PSNR metrics in the simulations performed using the different degradation methods. According to these results, it can be seen that the BN and BS algorithms are the ones from the reference algorithms which provide better results for the simulated images, while the GGSA and GLMSE algorithms provide the worst results for these images. On the other hand, for the Paris dataset, which represents a more realistic scenario, the reference algorithms which provide the best results are the HySure and the GGSA algorithms, while the BN and BS and CNMF algorithms provide the worst results. Finally, the results provided by the proposed CoEf-MHI algorithm are very close or even better than the best results obtained by the reference algorithms for all the situations, concluding that the CoEf-MHI algorithm provides very good results for fusing hyperspectral and multispectral images, in terms of the spectral and the spatial quality of the fusion. Moreover, Figure 3.7 displays the fused images obtained by the proposed CoEf-MHI and by the BN algorithms for the case of the Washington, Cuprite and Indian Pines datasets. We have selected these datasets and this last algorithm from the state-of-the-art set because for these images the BN is able to produce the best average results (according to Figure 3.4), surpassing the performance given by our CoEf-MHI algorithm. However, as it is seen in Figure 3.7, the differences in the fused images for all the cases considered are negligible, which demonstrates the suitability of our proposal.

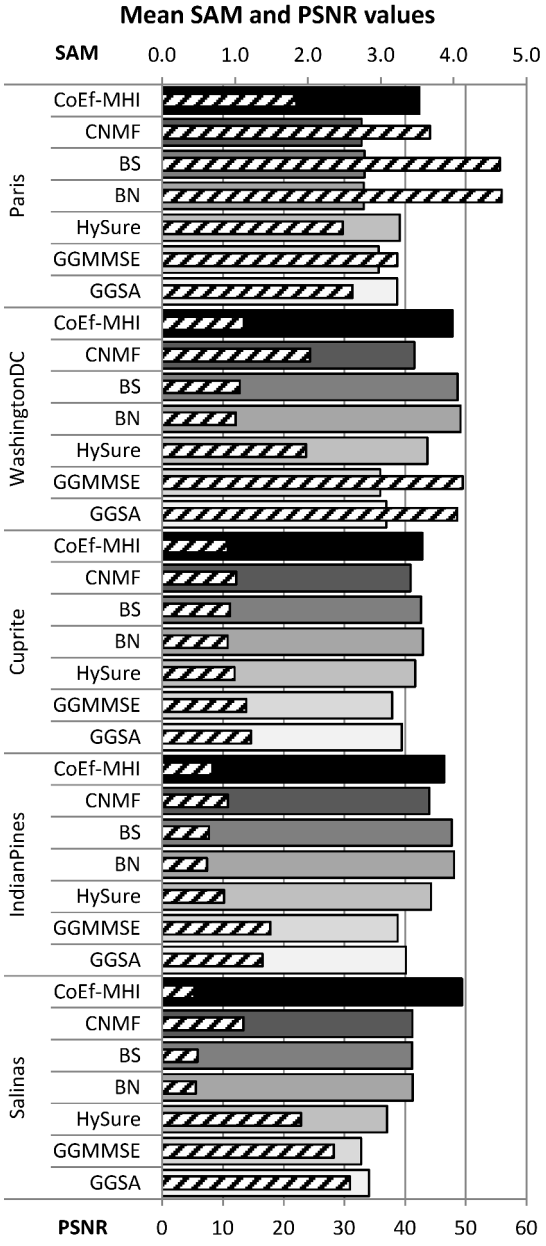


FIGURE 3.4: Mean SAM (dashed bars) and mean PSNR (filled bars) values for different images. Lower SAM values and higher PSNR values indicate better fusion results.

On the other hand, Tables 3.2 to 3.31 also reveal that the CoEf-MHI algorithm is the one which performs less floating point operations and has less memory allocation requirements, according to the evaluation methods described in Section 3.4.5. In particular, the CNMF algorithm is the one with the highest computational burden, followed by the BS, BN and HySure algorithms. The number of floating points operations performed by these algorithms surpass the total number of floating points operations performed by the CoEf-MHI algorithm in more than two orders of magnitude in all the simulations. Furthermore,

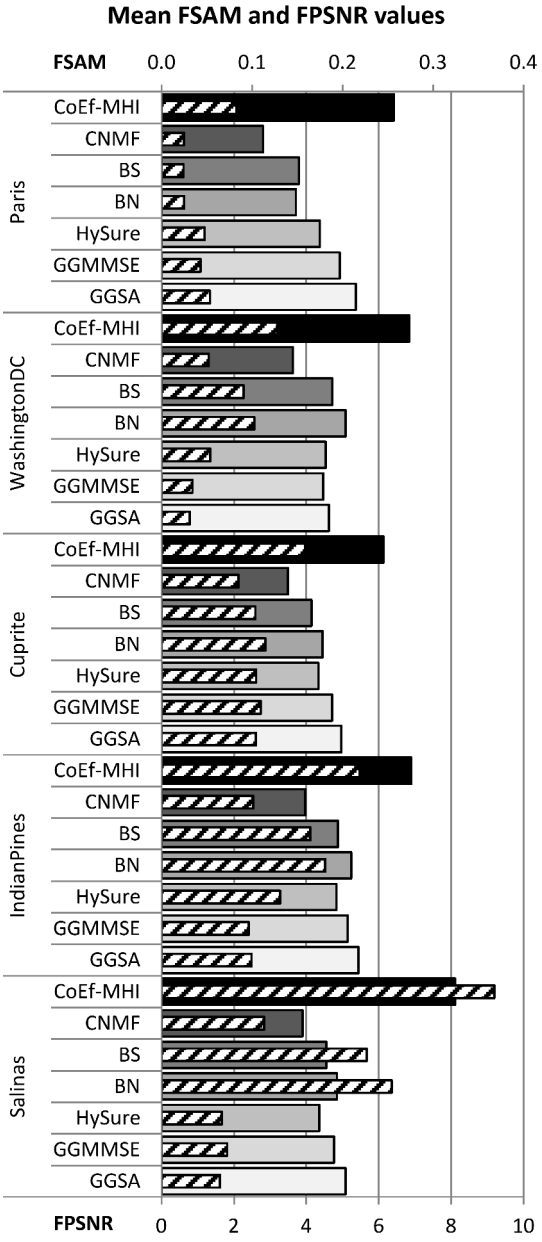


FIGURE 3.5: Mean FSAM (dashed bars) and mean FPSNR (filled bars) values for different images. Higher FSAM and FPSNR values indicate a higher efficiency in the fusion process.

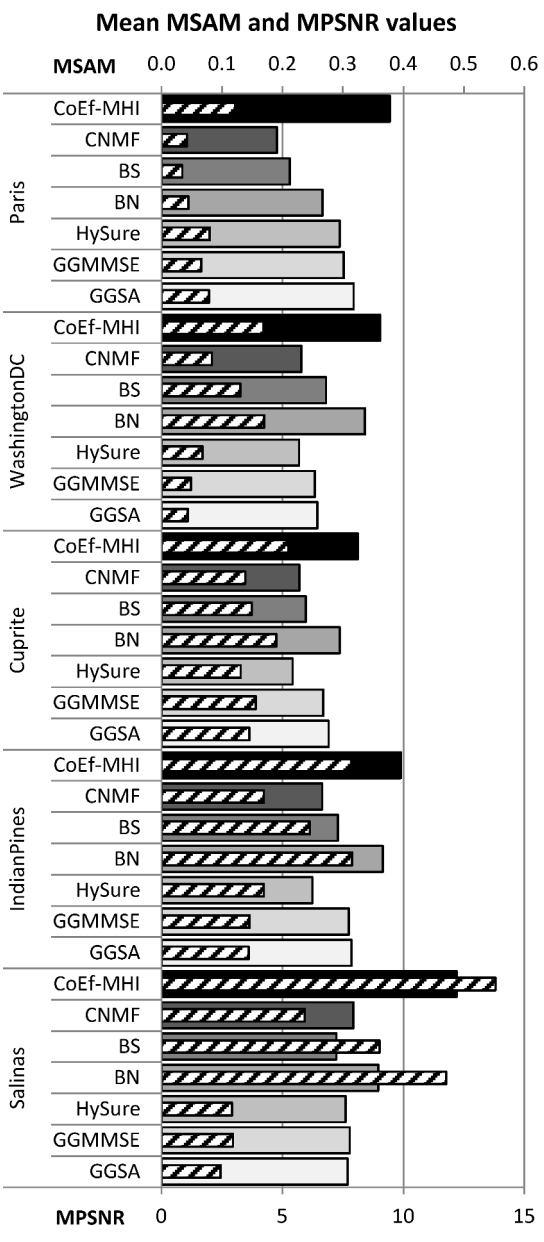


FIGURE 3.6: Mean MSAM (dashed bars) and mean MPSNR (filled bars) values for different images. Higher MSAM and MPSNR values indicate a higher efficiency in the fusion process.

the total memory allocated by the CNMF, BS and HySure algorithms also surpass the total memory allocated by the CoEf-MHI algorithm in at least one order of magnitude in all the simulations performed.

Finally, in order to easily evaluate the computational efficiency of our proposal, Figures 3.5 and 3.6 graphically show the mean values obtained for the FSAM and FPSNR, and MSAM

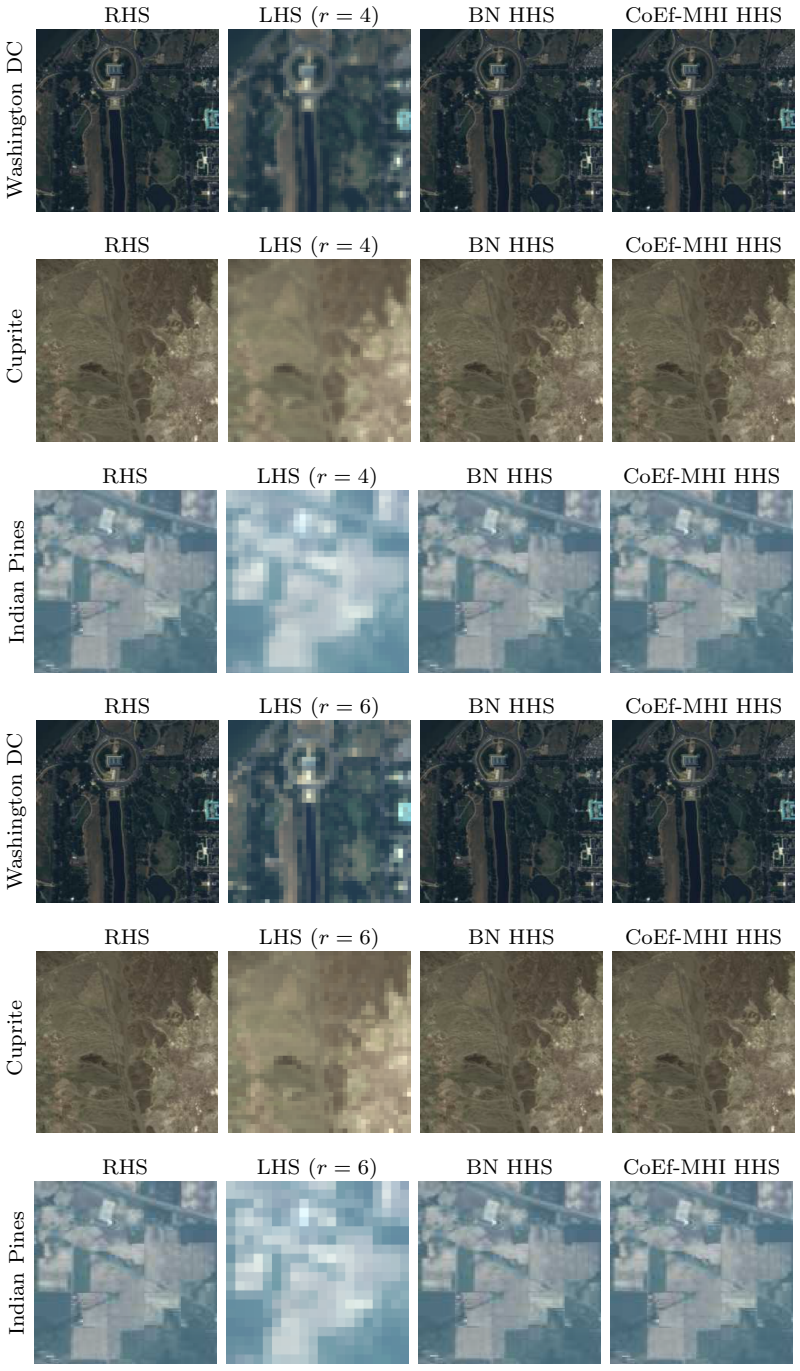


FIGURE 3.7: RGB representation of the different hyperspectral images obtained in the fusion process. RHS image, LHS image, and the fused HHS images, obtained with the BN algorithm and the CoEf-MHI algorithm respectively. The LHS images have been obtained according to the two first test cases shown in Table 3.1

.

and MPSNR metrics, respectively, in the simulations performed using the different images and degradation methods. As it can be clearly seen, the proposed CoEf-MHI algorithm is by far the most efficient option of all the considered in this work, with independence of the characteristics of the input hyperspectral and multispectral images.

All the simulations have been performed using MATLAB R2012b, in an Intel(R) Core(TM) i7-3537U @ 2.00 GHz 2.50 GHz.

3.6 Conclusions

In this Chapter a new algorithm, named CoEf-MHI, for the fusion of a *high-spectral low-spatial* (LHS) resolution hyperspectral image and a *high-spatial low-spectral resolution multispectral* (HMS) image, has been described.

This algorithm introduces the spatial details of the HMS image into the LHS image without introducing spectral distortions, in order to obtain a *high spatial and spectral resolution hyperspectral* (HHS) image. To do this, each pixel of HHS image is represented as a linear combination of some of its surrounding pixels. The CoEf-MHI algorithm has three important advantages with respect to the state-of-the art algorithms. First of all, each pixel of the HHS image is independently processed. Secondly, in order to perfectly represent a single pixel as a linear combination of its surrounding pixel, a low number of vectors is needed (typically no more than three pixels). Finally, the process independently performed for each pixel of the HHS image is based on the modified Gram-Schmidt method, which has a low computational burden and does not perform complex matrix operations. All these facts turn the CoEf-MHI algorithm into a very suitable option for applications under real-time constraints.

In order to evaluate the goodness of the proposed algorithm an extensive amount of simulations have been performed, comparing the results obtained using the CoEf-MHI algorithm with those provided by the most significant sate-of-the-art fusion algorithms up to this moment, i.e. the *Generalized Gram-Schmidt Adaptive* (GGSA) algorithm, the *Generalized Local Minimum Square Error* (GLMMSE) algorithm, the *Coupled Nonnegative Matrix Factorization* (CNMF) algorithm, the *Hyperspectral Superresolution* (HySure) algorithm, the *Bayesian Naive* (BN) algorithm and *Bayesian Sparse* (BS) algorithm. According to the results obtained, it can be concluded that the CoEf-MHI algorithm provides

very good results for fusing hyperspectral and multispectral images, in terms of the spectral and the spatial quality of the fusion and with independence of the spatial and spectral characteristics of the images to be fused. In particular, the fused images obtained by the CoEf-MHI algorithm show, in average, a better spatial and spectral quality than the one obtained by the GGSA, GLMMSE, HySure, and CNMF algorithms and a similar quality that the one achieved by the Bayesian Naive and the Bayesian Sparse algorithms under the same conditions. Moreover, the CoEf-MHI algorithm is able to complete the fusion process requiring a much lower computational complexity than all the aforementioned algorithms, which converts it in the most efficient fusion algorithm up to date.

Table 3.2: Salinas. Test case 1

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.841	2.549	34.40	7.328	6.703	0.059	5.132	4.335	0.090	7.935
GLMMSE	0.828	2.250	33.16	7.635	6.871	0.065	4.827	3.992	0.111	8.308
HySure	0.897	1.787	37.57	5.813	8.505	0.066	4.417	4.978	0.112	7.547
BN	0.913	0.459	41.53	6.222	8.522	0.256	4.874	4.594	0.475	9.042
BS	0.913	0.497	41.41	6.347	9.048	0.223	4.577	5.696	0.354	7.270
CNMF	0.881	1.095	41.66	7.645	10.23	0.089	4.072	4.213	0.217	9.889
CoEf-MHI	**0.979**	**0.411**	**52.12**	**2.985**	**6.089**	**0.376**	**8.563**	**3.856**	**0.588**	**13.52**

Table 3.3: Salinas. Test case 2

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.786	3.577	32.15	5.101	6.703	0.042	4.797	4.398	0.060	7.310
GLMMSE	0.767	3.349	30.97	5.351	6.871	0.043	4.507	4.266	0.070	7.260
HySure	0.773	2.927	34.80	6.608	8.491	0.040	4.099	4.824	0.071	7.214
BN	0.904	**0.472**	40.79	4.365	8.522	0.249	4.786	4.644	0.457	8.784
BS	0.904	0.492	40.74	4.398	9.048	0.225	4.502	5.760	0.353	7.073
CNMF	0.842	1.915	38.19	5.226	11.27	0.046	3.387	6.622	0.079	5.767
CoEf-MHI	**0.970**	0.474	**50.26**	**2.414**	**6.086**	**0.347**	**8.258**	**4.061**	**0.520**	**12.38**

Table 3.4: Salinas. Test case 3

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.838	2.528	34.20	7.451	6.703	0.059	5.102	4.437	0.089	7.708
GLMMSE	0.814	2.406	32.31	7.756	6.871	0.060	4.702	4.242	0.098	7.616
HySure	0.914	1.866	37.92	5.153	8.505	0.063	4.458	4.839	0.111	7.836
BN	0.951	0.447	45.15	4.501	8.522	0.263	5.298	4.646	0.482	9.718
BS	0.950	0.527	44.97	4.687	9.048	0.210	4.970	5.675	0.335	7.923
CNMF	0.876	1.000	39.44	7.841	10.06	0.099	3.921	4.589	0.218	8.595
CoEf-MHI	**0.966**	**0.434**	**49.45**	**4.029**	**6.086**	**0.378**	**8.124**	**4.118**	**0.559**	**12.01**

Table 3.5: Salinas. Test case 4

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.771	3.588	31.58	5.204	6.703	0.042	4.711	4.481	0.062	7.047
GLMMSE	0.740	3.515	30.03	5.470	6.871	0.041	4.371	4.276	0.067	7.024
HySure	0.866	2.012	37.27	4.939	8.491	0.059	4.390	4.848	0.103	7.688
BN	0.884	**0.440**	40.00	4.797	8.522	0.266	4.694	4.542	0.500	8.806
BS	0.884	0.445	39.98	4.798	9.048	0.248	4.419	5.669	0.396	7.053
CNMF	0.850	1.749	37.05	5.134	10.58	0.054	3.501	6.605	0.087	5.610
CoEf-MHI	**0.912**	0.487	**43.03**	**4.304**	**6.086**	**0.338**	**7.071**	**4.062**	**0.506**	**10.60**

Table 3.6: Salinas. Test case 5

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.873	1.414	36.55	7.164	6.703	0.105	5.453	4.438	0.159	8.237
GLMMSE	0.871	1.148	35.35	7.409	6.871	0.127	5.145	4.269	0.204	8.280
HySure	0.873	1.104	39.07	7.156	8.505	0.107	4.594	4.897	0.185	7.979
BN	0.892	0.464	40.33	6.947	8.522	0.253	4.732	4.652	0.463	8.669
BS	0.892	0.478	40.28	7.018	9.048	0.231	4.451	5.675	0.369	7.097
CNMF	0.926	0.454	46.08	6.508	10.24	0.215	4.498	4.218	0.522	10.92
CoEf-MHI	**0.975**	**0.408**	**51.56**	**3.339**	**6.086**	**0.403**	**8.471**	**4.116**	**0.596**	**12.53**

Table 3.7: Salinas. Test case 6

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.854	1.838	35.47	4.876	6.703	0.081	5.292	4.445	0.122	7.981
GLMMSE	0.848	1.476	34.59	5.050	6.871	0.099	5.034	4.244	0.160	8.150
HySure	0.847	1.770	35.51	4.905	8.491	0.067	4.182	4.821	0.117	7.366
BN	0.871	0.491	39.70	5.017	8.522	0.239	4.659	4.541	0.449	8.742
BS	0.870	0.494	39.70	5.023	9.048	0.224	4.387	5.667	0.357	7.004
CNMF	0.905	0.507	44.91	4.440	11.28	0.175	3.981	6.590	0.300	6.814
CoEf-MHI	**0.968**	**0.446**	**49.80**	**2.712**	**6.086**	**0.368**	**8.182**	**4.105**	**0.546**	**12.13**

Table 3.8: Indian Pines. Test case 1

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.984	1.230	39.34	0.780	7.957	0.102	4.944	5.121	0.159	7.681
GLMMSE	0.972	1.148	36.68	0.994	8.040	0.108	4.562	5.017	0.174	7.310
HySure	0.988	1.000	41.79	0.679	9.636	0.104	4.337	7.093	0.141	5.892
BN	**0.992**	0.882	43.18	0.582	9.658	0.117	4.471	5.268	0.215	8.196
BS	0.990	0.933	42.67	0.604	10.31	0.104	4.138	6.545	0.164	6.519
CNMF	0.955	**0.832**	**43.71**	**0.466**	11.38	0.106	3.848	6.762	0.178	6.464
CoEf-MHI	0.991	0.890	42.96	0.601	**6.993**	**0.161**	**6.143**	**4.705**	**0.239**	**9.130**

Table 3.9: Indian Pines. Test case 2

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.926	1.560	39.46	0.549	7.275	0.088	5.424	5.126	0.125	7.699
GLMMSE	0.922	1.930	36.68	0.683	7.464	0.069	4.915	5.011	0.103	7.320
HySure	0.964	0.950	42.83	0.338	9.074	0.116	4.720	7.087	0.149	6.043
BN	**0.977**	**0.565**	**49.06**	**0.163**	9.107	0.194	5.387	5.245	**0.337**	9.354
BS	0.976	0.584	48.65	0.172	9.689	0.177	5.021	6.539	0.262	7.439
CNMF	0.940	1.016	42.15	0.384	11.35	0.087	3.713	6.552	0.150	6.433
CoEf-MHI	0.949	0.678	46.59	0.266	**6.688**	**0.220**	**6.965**	**4.689**	0.314	**9.934**

Table 3.10: Indian Pines. Test case 3

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.940	1.247	41.25	0.663	7.275	0.110	5.671	5.123	0.156	8.052
GLMMSE	0.940	1.599	37.79	0.878	7.464	0.084	5.062	5.012	0.125	7.539
HySure	0.974	0.701	46.46	0.329	9.088	0.157	5.113	7.091	0.201	6.552
BN	**0.977**	**0.550**	**49.43**	**0.236**	9.107	0.200	5.428	5.285	**0.344**	9.354
BS	0.975	0.579	48.78	0.255	9.689	0.178	5.034	6.540	0.264	7.459
CNMF	0.950	0.861	44.32	0.440	11.15	0.104	3.976	6.614	0.176	6.701
CoEf-MHI	0.958	0.638	47.30	0.341	**6.688**	**0.234**	**7.072**	**4.706**	0.333	**10.05**

Table 3.11: Indian Pines. Test case 4

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.924	1.641	38.85	0.574	7.275	0.084	5.341	5.102	0.119	7.615
GLMMSE	0.907	2.107	35.54	0.768	7.464	0.064	4.761	5.001	0.095	7.106
HySure	0.967	0.838	45.04	0.272	9.074	0.131	4.963	7.087	0.168	6.355
BN	**0.976**	**0.576**	**48.89**	**0.167**	9.107	0.191	5.368	5.245	**0.331**	9.321
BS	0.975	0.586	48.64	0.172	9.689	0.176	5.020	6.542	0.261	7.435
CNMF	0.898	1.110	38.94	0.662	10.99	0.082	3.545	6.522	0.138	5.971
CoEf-MHI	0.947	0.711	45.96	0.284	**6.688**	**0.210**	**6.871**	**4.689**	0.300	**9.80**

Table 3.12: Indian Pines. Test case 5

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.947	1.174	41.73	0.598	7.275	0.117	5.736	5.099	0.167	8.183
GLMMSE	0.974	0.946	44.02	0.450	7.464	0.142	5.898	5.009	0.211	8.788
HySure	0.974	0.673	46.58	0.325	9.088	0.164	5.126	7.091	0.210	6.569
BN	0.978	**0.562**	**49.14**	**0.242**	9.107	0.195	5.396	5.252	0.339	9.356
BS	0.976	0.582	48.69	0.256	9.689	0.177	5.025	6.542	0.262	7.442
CNMF	**0.980**	0.790	47.60	0.857	10.97	0.115	4.339	6.669	0.190	7.137
CoEf-MHI	0.964	0.586	48.47	0.299	**6.688**	**0.255**	**7.246**	**4.706**	**0.362**	**10.30**

Table 3.13: Indian Pines. Test case 6

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.936	1.418	40.18	0.483	7.275	0.097	5.523	5.102	0.138	7.875
GLMMSE	0.965	1.198	41.97	0.382	7.464	0.112	5.623	5.004	0.167	8.388
HySure	0.967	0.955	42.96	0.326	9.074	0.115	4.734	7.087	0.148	6.061
BN	0.976	**0.583**	**48.71**	**0.170**	9.107	0.188	5.348	5.244	0.327	9.288
BS	0.975	0.591	48.54	0.174	9.689	0.175	5.009	6.538	0.259	7.424
CNMF	**0.979**	0.825	47.05	0.600	10.70	0.113	4.398	6.584	0.184	7.146
CoEf-MHI	0.956	0.640	47.47	0.232	**6.688**	**0.233**	**7.098**	**4.689**	**0.333**	**10.125**

TABLE 3.14: Cuprite. Test case 1

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.984	1.230	39.34	0.780	7.957	0.102	4.944	5.715	0.142	6.884
GLMMSE	0.972	1.148	36.68	0.994	8.040	0.108	4.562	5.666	0.154	6.473
HySure	0.989	0.992	41.92	0.670	9.636	0.105	4.350	7.693	0.131	5.449
BN	**0.992**	**0.882**	**43.18**	**0.582**	9.658	0.117	4.471	5.845	0.194	7.388
BS	0.990	0.933	42.66	0.604	10.31	0.104	4.138	7.160	0.150	5.958
CNMF	0.989	0.964	41.24	0.671	11.90	0.087	3.465	7.270	0.143	5.672
CoEf-MHI	0.991	0.890	42.96	0.601	**6.993**	**0.161**	**6.143**	**5.287**	**0.213**	**8.126**

TABLE 3.15: Cuprite. Test case 2

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.980	1.369	38.37	0.567	7.957	0.092	4.822	5.708	0.128	6.722
GLMMSE	0.963	1.290	35.55	0.747	8.040	0.096	4.419	5.649	0.137	6.289
HySure	0.989	1.029	41.29	0.455	9.622	0.101	4.291	7.690	0.126	5.369
BN	**0.991**	**0.897**	**43.02**	**0.393**	9.658	0.115	4.454	5.835	0.191	7.372
BS	0.990	0.932	42.66	0.402	10.31	0.104	4.138	7.158	0.150	5.960
CNMF	0.984	1.106	39.30	0.539	11.55	0.078	3.403	7.119	0.127	5.521
CoEf-MHI	**0.991**	0.917	42.68	0.413	**6.993**	**0.156**	**6.103**	**5.270**	**0.207**	**8.089**

TABLE 3.16: Cuprite. Test case 3

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.985	1.227	39.62	0.765	7.957	0.102	4.979	5.717	0.143	6.930
GLMMSE	0.968	1.189	36.14	1.051	8.040	0.105	4.495	5.669	0.148	6.375
HySure	0.989	0.980	41.85	0.673	9.636	0.106	4.343	7.692	0.133	5.440
BN	**0.992**	**0.886**	**43.12**	**0.592**	9.658	0.117	4.464	5.841	0.193	7.382
BS	0.990	0.930	42.71	0.604	10.31	0.104	4.143	7.159	0.150	5.967
CNMF	0.988	1.000	41.00	0.703	11.66	0.086	3.517	7.154	0.140	5.730
CoEf-MHI	0.991	0.912	42.68	0.622	**6.993**	**0.157**	**6.104**	**5.287**	**0.207**	**8.074**

TABLE 3.17: Cuprite. Test case 4

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.980	1.369	38.52	0.566	7.957	0.092	4.841	5.711	0.128	6.745
GLMMSE	0.950	1.407	33.99	0.880	8.040	0.088	4.228	5.650	0.126	6.017
HySure	0.988	1.028	41.42	0.448	9.622	0.101	4.305	7.690	0.127	5.386
BN	**0.991**	**0.919**	**42.78**	0.406	9.658	0.113	4.429	5.836	0.186	7.331
BS	0.990	0.933	42.68	**0.404**	10.31	0.104	4.140	7.157	0.150	5.963
CNMF	0.976	1.272	37.73	0.630	11.68	0.067	3.230	7.041	0.112	5.359
CoEf-MHI	0.990	0.959	42.24	0.435	**6.993**	**0.149**	**6.040**	**5.271**	**0.198**	**8.014**

Table 3.18: Cuprite. Test case 5

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.989	1.016	40.90	0.676	7.957	0.124	5.141	5.717	0.172	7.155
GLMMSE	0.991	0.915	42.79	0.584	8.040	0.136	5.322	5.668	0.193	7.549
HySure	0.991	0.902	42.76	0.598	9.636	0.115	4.438	7.692	0.144	5.559
BN	0.991	0.896	43.04	0.589	9.658	0.115	4.456	5.842	0.191	7.368
BS	0.990	0.933	42.68	0.604	10.31	0.104	4.140	7.159	0.150	5.962
CNMF	**0.992**	0.874	43.24	0.573	11.93	0.096	3.624	7.304	0.157	5.919
CoEf-MHI	**0.992**	**0.844**	**43.44**	**0.570**	**6.993**	**0.169**	**6.212**	**5.289**	**0.224**	**8.213**

Table 3.19: Cuprite. Test case 6

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.987	1.105	40.14	0.484	7.957	0.114	5.044	5.710	0.158	7.029
GLMMSE	0.989	0.995	42.02	0.418	8.040	0.125	5.227	5.649	0.178	7.439
HySure	0.988	1.038	40.82	0.462	9.622	0.100	4.243	7.690	0.125	5.309
BN	0.990	0.929	42.70	0.401	9.658	0.112	4.421	5.836	0.185	7.317
BS	0.990	0.945	42.55	0.406	10.31	0.103	4.127	7.157	0.148	5.945
CNMF	**0.991**	0.903	42.94	**0.392**	11.55	0.096	3.716	7.186	0.154	5.976
CoEf-MHI	**0.991**	**0.891**	**42.96**	0.400	**6.993**	**0.161**	**6.143**	**5.284**	**0.212**	**8.131**

Table 3.20: Washington DC. Test case 1

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.977	3.925	37.10	2.834	7.981	0.032	4.649	5.728	0.044	6.477
GLMMSE	0.962	4.292	34.71	3.814	8.047	0.029	4.313	5.681	0.041	6.109
HySure	0.992	1.654	44.85	1.471	9.650	0.063	4.647	7.693	0.079	5.830
BN	**0.996**	**0.969**	**49.43**	**1.079**	9.671	0.107	5.111	5.850	**0.176**	8.449
BS	0.995	1.061	48.61	1.097	10.31	0.091	4.715	7.160	0.132	6.790
CNMF	0.979	2.274	40.13	2.873	11.80	0.037	3.402	7.188	0.061	5.582
CoEf-MHI	0.990	1.110	47.97	1.967	**6.993**	**0.129**	**6.860**	**5.298**	0.170	**9.055**

Table 3.21: Washington DC. Test case 2

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.973	4.233	36.44	2.052	7.981	0.030	4.566	5.717	0.041	6.374
GLMMSE	0.949	5.101	33.58	2.886	8.047	0.024	4.174	5.655	0.035	5.938
HySure	0.988	2.278	41.97	1.256	9.636	0.046	4.356	7.690	0.057	5.458
BN	**0.995**	**1.014**	**48.96**	**0.727**	9.671	**0.102**	5.063	5.838	**0.169**	8.387
BS	**0.995**	1.068	48.54	0.733	10.31	0.091	4.708	7.157	0.131	6.782
CNMF	0.976	2.127	39.41	2.102	10.43	0.045	3.779	7.185	0.065	5.484
CoEf-MHI	0.988	1.230	47.02	1.502	**6.993**	0.116	**6.723**	**5.285**	0.154	**8.896**

Table 3.22: Washington DC. Test case 3

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.977	3.854	37.15	2.810	7.981	0.033	4.655	5.724	0.045	6.490
GLMMSE	0.955	4.586	34.09	4.081	8.047	0.027	4.236	5.675	0.038	6.007
HySure	0.992	1.730	44.54	1.459	9.650	0.060	4.616	7.693	0.075	5.791
BN	**0.996**	**0.939**	**50.05**	**1.052**	9.671	0.110	5.175	5.847	**0.182**	8.561
BS	0.995	1.049	48.98	1.074	10.31	0.092	4.751	7.158	0.133	6.843
CNMF	0.974	2.325	38.17	3.139	11.75	0.037	3.248	7.155	0.060	5.335
CoEf-MHI	0.992	1.037	48.51	1.805	**6.993**	**0.138**	**6.936**	**5.296**	**0.182**	**9.159**

Table 3.23: Washington DC. Test case 4

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.973	4.835	35.46	2.173	7.981	0.026	4.443	5.723	0.036	6.195
GLMMSE	0.927	6.037	31.69	3.516	8.047	0.021	3.938	5.661	0.029	5.597
HySure	0.992	1.949	44.36	0.993	9.636	0.053	4.604	7.690	0.067	5.768
BN	**0.995**	**1.036**	**48.96**	**0.719**	9.671	0.100	5.063	5.841	**0.165**	8.382
BS	**0.995**	1.065	48.84	**0.719**	10.31	0.091	4.737	7.157	0.131	6.824
CNMF	0.967	3.416	33.31	2.786	11.58	0.025	2.876	7.083	0.041	4.703
CoEf-MHI	0.988	1.215	46.88	1.497	**6.993**	**0.118**	**6.704**	**5.285**	0.156	**8.871**

Table 3.24: Washington DC. Test case 5

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.984	3.491	38.15	2.464	7.981	0.036	4.781	5.726	0.050	6.663
GLMMSE	0.991	2.165	41.45	1.786	8.047	0.057	5.151	5.680	0.081	7.298
HySure	0.993	1.644	45.35	1.382	9.650	0.063	4.699	7.693	0.079	5.895
BN	**0.995**	1.014	49.16	**1.082**	9.671	0.102	5.083	5.847	0.169	8.407
BS	**0.995**	1.067	48.68	1.093	10.31	0.091	4.722	7.159	0.131	6.800
CNMF	0.994	**0.947**	**49.71**	1.407	11.77	0.090	4.223	7.244	0.146	6.863
CoEf-MHI	0.992	0.991	48.94	1.760	**6.990**	**0.144**	**6.998**	**5.296**	**0.191**	**9.240**

Table 3.25: Washington DC. Test case 6

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.980	3.956	37.07	1.848	7.981	0.032	4.644	5.721	0.044	6.479
GLMMSE	0.988	2.576	40.05	1.358	8.047	0.048	4.977	5.660	0.069	7.076
HySure	0.987	2.606	40.98	1.298	9.636	0.040	4.253	7.690	0.050	5.329
BN	**0.995**	**1.091**	**48.37**	**0.739**	9.671	0.095	5.002	5.840	0.157	8.282
BS	**0.995**	1.101	48.31	**0.739**	10.31	0.088	4.686	7.158	0.127	6.750
CNMF	0.992	1.098	48.34	1.033	11.48	0.079	4.210	7.154	0.127	6.757
CoEf-MHI	0.989	1.170	47.64	1.397	**6.993**	**0.122**	**6.812**	**5.283**	**0.162**	**9.017**

Table 3.26: Paris. Test case 1

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.782	2.675	37.88	3.491	7.214	0.052	5.251	4.897	0.076	7.735
GLMMSE	0.846	2.796	36.64	**2.914**	7.249	0.049	5.054	4.752	0.075	7.711
HySure	0.850	2.029	41.41	2.981	8.954	0.055	4.625	5.796	0.085	7.146
BN	0.784	3.902	34.75	3.677	8.964	0.029	3.877	5.008	0.051	6.939
BS	0.746	4.727	33.08	4.287	8.783	0.024	3.767	6.281	0.034	5.267
CNMF	0.789	3.055	34.03	3.605	11.87	0.028	2.866	7.192	0.046	4.731
CoEf-MHI	**0.860**	**1.653**	**43.62**	3.200	**6.581**	**0.092**	**6.628**	**4.531**	**0.133**	**9.627**

Table 3.27: Paris. Test case 2

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.772	2.857	37.42	2.561	7.214	0.049	5.187	4.858	0.072	7.703
GLMMSE	0.817	2.953	36.64	2.052	7.249	0.047	5.054	4.730	0.072	7.746
HySure	**0.862**	1.794	42.83	**1.837**	8.939	0.062	4.792	5.217	0.107	8.211
BN	0.760	4.287	33.83	2.636	8.964	0.026	3.774	4.978	0.047	6.796
BS	0.744	4.746	33.05	2.869	8.780	0.024	3.763	6.281	0.034	5.263
CNMF	0.705	3.483	31.73	3.537	11.44	0.025	2.773	6.611	0.043	4.799
CoEf-MHI	0.851	**1.719**	**42.95**	2.270	**6.593**	**0.088**	**6.515**	**4.472**	**0.130**	**9.606**

Table 3.28: Paris. Test case 3

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.787	2.632	38.17	3.429	7.214	0.053	5.290	4.865	0.078	7.844
GLMMSE	0.837	2.893	36.54	**2.978**	7.249	0.048	5.040	4.731	0.073	7.724
HySure	0.834	2.523	38.52	3.307	8.954	0.044	4.303	5.221	0.076	7.379
BN	0.793	3.521	35.54	3.471	8.964	0.032	3.965	5.022	0.057	7.077
BS	0.752	4.569	33.43	4.195	8.783	0.025	3.806	6.285	0.035	5.318
CNMF	0.821	2.225	39.43	3.175	12.19	0.037	3.235	7.184	0.063	5.488
CoEf-MHI	**0.854**	**1.737**	**42.84**	3.228	**6.610**	**0.087**	**6.482**	**4.523**	**0.127**	**9.473**

Table 3.29: Paris. Test case 4

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.774	2.847	37.93	2.655	7.214	0.049	5.257	4.859	0.072	7.806
GLMMSE	0.783	3.413	35.00	2.354	7.249	0.040	4.829	4.731	0.062	7.398
HySure	**0.856**	2.008	40.93	**1.991**	8.939	0.056	4.579	5.212	0.096	7.854
BN	0.743	4.307	33.63	2.723	8.964	0.026	3.752	4.983	0.047	6.750
BS	0.749	4.609	33.34	2.825	8.780	0.025	3.796	6.279	0.035	5.309
CNMF	0.693	4.031	30.64	3.644	12.08	0.021	2.536	7.192	0.034	4.261
CoEf-MHI	0.846	**1.822**	**42.39**	2.410	**6.609**	**0.083**	**6.414**	**4.471**	**0.123**	**9.480**

TABLE 3.30: Paris. Test case 5

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.802	2.158	41.22	**3.258**	7.214	0.064	5.713	4.868	0.095	8.468
GLMMSE	0.844	3.422	35.15	3.400	7.249	0.040	4.849	4.727	0.062	7.436
HySure	0.815	3.276	35.52	3.797	8.954	0.034	3.967	5.206	0.059	6.823
BN	0.658	5.606	31.31	5.076	8.964	0.020	3.493	4.987	0.036	6.280
BS	0.745	4.506	33.63	4.216	8.783	0.025	3.829	6.281	0.035	5.354
CNMF	0.761	4.129	32.41	6.893	11.39	0.021	2.843	6.578	0.037	4.927
CoEf-MHI	**0.846**	**2.016**	**41.37**	3.966	**6.615**	**0.075**	**6.254**	**4.487**	**0.111**	**9.220**

TABLE 3.31: Paris. Test case 6

Algorithm	CC	SAM	PSNR	ERGAS	LFLOPs	FSAM	FPSNR	LMEM	MSAM	MPSNR
GGSA	0.781	2.518	39.56	**2.531**	7.214	0.055	5.484	4.860	0.082	8.141
GLMMSE	0.800	3.903	33.89	2.536	7.249	0.035	4.675	4.731	0.054	7.163
HySure	0.820	3.264	35.43	2.428	8.939	0.034	3.963	5.217	0.059	6.790
BN	0.603	6.326	30.23	3.712	8.964	0.018	3.372	4.987	0.032	6.061
BS	0.736	4.656	33.41	2.905	8.783	0.024	3.804	6.283	0.034	5.318
CNMF	0.647	5.146	28.94	3.914	11.39	0.017	2.541	6.591	0.029	4.391
CoEf-MHI	**0.844**	**2.111**	**41.10**	2.684	**6.615**	**0.072**	**6.213**	**4.475**	**0.106**	**9.185**

Chapter 4

Hyperspectral and ultraspectral image compression

This chapter presents the algorithm developed in this Thesis for spectrally decorrelating and compressing hyperspectral and ultraspectral images. This algorithm, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), is totally described in this Chapter. The HyperLCA algorithm is specifically design for preserving the most different pixels through the compression-decompression process, since these pixels are crucial for many hyperspectral imaging applications such as anomaly detection, classification or target detection. The HyperLCA algorithm is evaluated using different images and also compared with the *Principal Component Analysis* (PCA) transform which provides the best state-of-the-art results for this compression stage. This algorithm is also used in conjunction with the CCSDS121 coder for evaluating the entire compression-decompression performance achievable using the proposed methodology. Finally, the effect of the compression within the HyperLCA transform on hyperspectral anomaly detection applications is also evaluated.

4.1 Outline

Hyperspectral image compression, and more specifically on-board satellite compression, has become a very popular research topic, for both academics and the space industry. An efficient compression of hyperspectral images on-board satellites is mandatory, due to the huge amount of sensed data, in order to save bandwidth and storage space. Reducing the data volume in a harsh environment like space, where the computational power is limited, is also a challenge, which has been faced with two main approaches: proposing new algorithms, specially designed to efficiently perform the hyperspectral images compression; and presenting the technologies and strategies to execute the compression in the hardware available on the satellite, minimizing the complexity, and consequently the resources usage and power consumption.

The algorithms for compressing hyperspectral images, as any other state-of-the-art compression algorithm, take advantage of the redundancies in the image samples to reduce the data volume. Hyperspectral image compression algorithms may take into consideration the redundancies in the spatial and spectral domains for reducing the amount of data with or without loosing information. Lossless compression algorithms have been traditionally preferred, to preserve all the information present in the hyperspectral cube for scientific purposes, despite their limited compression ratio, which usually allows a data volume reduction from half up to one third of the size of the image, at most [97]. Nevertheless, the increment in the data-rate of the new-generation sensors is making more critical the necessity of obtaining higher compression ratios, making it necessary to use near-lossless and/or lossy compression techniques.

The general approach for compressing hyperspectral images consists of a spatial and/or spectral decorrelator, a quantization stage and an entropy coder, which explores the probability of the symbols to assign short codewords to the most probable symbols and long codewords to the less probable ones. The decorrelator can be transform-based or prediction-base. In the transform-based approaches, transforms like the *Discrete Wavelet Transform* (DWT) [98] or the *Kahrunen-Loève Transform* (KLT) [99, 100] are applied to decorrelate the data, while in the prediction-based approaches the samples are predicted from neighbouring (in the spectral or spatial directions) samples, and the predictions errors are encoded. While lossless compression is more efficiently performed by prediction-based methods, transform-based approaches are generally preferred for lossy compression.

Lossy compression approaches for hyperspectral images are typically evaluated in terms of the rate-distortion relation achieved. However, it has been proven that low average distortions in the compression-decompression process do not necessarily ensure good results when the decompressed images are used in specific hyperspectral applications [37]. In general, lossy compression behaves as a low-pass filter, which may produce a reduction of the noise present in the image, positively affecting the results when processing the decompressed hyperspectral images in some applications [38]; but at the same time, the low pass filter can also remove the most atypical elements of the image, which are crucial for several applications, such as anomaly detection, classification, unmixing, or target detection [39–42].

In this scenario, the transform-based lossy compression approaches, based on the KLT transform for decorrelating the spectral information, have been proven to yield the best results in terms of rate-distortion as well as in preserving the relevant information for the ulterior hyperspectral analysis [30, 37, 101, 101, 102]. In particular, the *Principal Component Analysis* (PCA) used for decorrelating and reducing the amount of spectral information, coupled with the JPEG2000 [103] for decorrelating the spatial information and performing the quantization stage and the entropy coding, stands out due to its lossy compression results, which have been demonstrated to be comparatively better than the results provided by other state-of-the-art approaches [37, 101, 102, 104]. Indeed, the PCA algorithm has been widely used as spectral decorrelator not only for compression, but also for other hyperspectral imaging applications such as classification or unmixing, increasing the accuracy of the obtained results [16, 82].

Despite their optimal decorrelation features, the KLT approaches, including the PCA algorithm, have important disadvantages that prevent their use in several situations. These disadvantages include an extremely high computational cost, intensive memory requirements, high implementation costs and a non-scalable nature, what make these approaches not suitable for applications under latency/power/memory constrained environments, such as compression on-board satellites. These limitations, as well as the promising compressions results achieved with the KLT approaches, have motivated the appearance of research works which aim to reduce the complexity of the transform by using divide-and-conquer strategies [105]. Nevertheless, the compression performance of these approaches is typically lower than the performance of the general KLT approach, or in particular than the performance of the PCA transform, while their computational complexity is still very high.

A new algorithm, named HyperLCA, for performing the spectral decorrelation and reduction of hyperspectral and ultraspectral images, is described in this Chapter. This algorithm is also based on orthogonal projections, but is not based on the KLT transform. Moreover, this algorithm is able to preserve most of the benefits provided by the KLT approaches, but with a reduced computational burden and implementation complexity, and with a high grade of parallelism and scalability, making it especially suitable for those compression applications in which the computational complexity or the real-time requirements make the PCA not applicable. As a further advantage, the proposed algorithm can independently compress different blocks of the image, decreasing the time and resources needed, what facilitates streaming applications as well as resilience to errors. This strategy would be highly inefficient in KLT-based approaches since KLT-based approaches require the extraction of the eigenvalues and eigenvectors of the covariance matrix, which number of rows and columns is equal to the number of bands of the hyperspectral image, with independence of the number of pixels involved.

The goodness of the proposed HyperLCA transform for spectrally decorrelating and reducing hyperspectral and ultraspectral images has been tested using different data sets, comparing its compression performance with the performance of the PCA transform, in terms of accuracy and computational burden. As a further step, the HyperLCA transform has been tested in conjunction with the CCSDS121 coder [106] for evaluating the overall performance of the entire compression process with the proposed methodology. Finally, the effects of applying the HyperLCA transform on hyperspectral images applications, such as anomaly detection, has been evaluated. The results obtained in the experiments demonstrate the suitability of the proposed transform for efficiently performing the spectral decorrelation and reduction of hyperspectral and ultraspectral images for applications under real-time constraints.

4.2 State-of-the-art in transform-based hyperspectral and ultraspectral compression

This Chapter proposes a new algorithm, named HyperLCA, for carrying on the spectral decorrelation and reduction of hyperspectral and ultraspectral images, by performing an spectral transform of the data set based on orthogonal projections. We have selected the PCA transform as the most significant proposal up to now, due to its superior performance

[30]. The spectral transforms for spectrally decorrelating and reducing the hyperspectral data are typically used in conjunction with additional compression stages, namely spatial decorrelation, and entropy coding. In this scenario, the PCA transform used in conjunction with the JPEG2000 compressor [103] for decorrelating the spatial information and performing the quantization and entropy coding stages, (PCA+JPEG2000) [102] provides excellent compression results.

The PCA transform is based on the KLT transform. The generic KLT transform is a representation of a stochastic process as an infinite linear combination of orthogonal functions. When this transform is applied to a discrete and finite process, for example, a particular hyperspectral data set, the problem takes a simpler form and standard algebra can be used to carry out the calculations. The process for spectrally decorrelating hyperspectral images using the KLT transform can be described as follows:

- A $N_b \times N_b$ covariance matrix which contains the variance and covariance among the different bands is calculated first. N_b represents the number of bands of the hyperspectral image.
- Secondly, the eigenvalues and eigenvectors of the covariance matrix are extracted. The set of all the extracted eigenvectors build up a $N_b \times N_b$ orthonormal matrix, able to span the entire space of the data set.
- Finally, the data set is projected onto the space spanned by the orthonormal set of extracted eigenvectors, obtaining the decorrelated data set. The real data set can be recovered from the decorrelated data set by projecting it onto the transpose of the version of the eigenvectors matrix.

Despite its great performance for compressing hyperspectral images, the KLT have some important disadvantages mainly due to its computational burden and lack of scalability. These disadvantages come from the necessity of performing the eigenvalues and eigenvectors calculation process.

The PCA approach performs the exact same operations than the KLT. However, it does not use all the eigenvalues and eigenvectors, but a set of a predefined number of eigenvectors (N_e), corresponding to the maximum positive values of the eigenvalues. Assuming that the hyperspectral data is highly correlated, and that the spectral vectors exist in a *n-dimensional* linear subspace ($n < N_b$), the data can be projected into the subspace projected by the N_e extracted eigenvectors without losing information if $N_e > n$. This way,

the hyperspectral information is not only spectrally decorrelated but also reduced. Moreover, high positives values of the extracted eigenvectors indicate that the corresponding eigenvalues are able to represent high amounts of information of the data set. Hence, if not all the eigenvectors associated with positive eigenvalues are used, but just the eigenvectors associated with the eigenvalues with the higher values, the image can be spectrally decorrelated and reduced to a smaller data volume without loosing too much information. The PCA provides better compression results than the KLT, but has the same disadvantages, in terms of computational complexity, due to the eigenvalues and eigenvectors calculation process, which complexity increases with the number of bands in the hyperspectral image.

The superior high coding performances of the KLT approaches has motivated several research works which aim to reduce its complexity using divide-and-conquer strategies [105]. These strategies try to independently apply the KLT transform to different portions of the data set with the objective of reducing its computational complexity while preserving its advantages. In this scenario, the *Pairwise Orthogonal Transform* (POT) approach provides the best relation between coding performance and complexity. This approach approximates the KLT transform as a composition of smaller KLT transforms for only two image components each as described in [31]. Nevertheless, all these divide-and-conquer strategies provide poorer rate-distortion results than the PCA transform at a still too high computational complexity.

4.3 Hyperspectral image compression within the HyperLCA algorithm

A new algorithm for performing lossy hyperspectral images compression, and in particular for performing the spectral decorrelation and the spectral reduction of hyperspectral data, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), has been developed in this Thesis with the purpose of providing a good compression performance at a reasonable computational burden. Similarly to the KLT transform-based approaches, and in particular to the PCA transform, the HyperLCA algorithm extracts a set of orthonormal vectors which are then used for projecting the data into a new subspace. This way, the information is decorrelated and also reduced according to the number of projections vectors extracted.

As already mentioned in Section 4.2, reducing the average losses does not always guarantee good compression results for hyperspectral imaging applications, particularly in applications in which some pixels may be very different from the rest of the pixels, such as anomaly detection, target detection or classification. For this reason, the HyperLCA algorithm is focused on selecting the projection vectors which best represent the most different pixels within the data set. The rest of the pixels are then represented as a linear combination of the extracted projection vectors. This process ensures that the anomalous pixels are preserved after the compression, making this transform more suitable for the mentioned hyperspectral applications.

The HyperLCA algorithm first centers the data set by substracting the centroid pixel to each pixel of the image. Secondly, the first projection vector is selected as the brightest pixel of the centered image. Afterwards, it sequentially performs orthogonal projections of the data set, in such a way that one projection vector is found in each iteration. The orthogonal projections of the data set are obtained using a modified version of the Gram-Schmidt method, which features low computational complexity and allows the reuse of previously computed information, speeding up the overall process, and without any complex matrix calculation. This modified Gram-Schmidt method has been described in Section 2.4.1.

A further advantage of applying this orthogonalization and projection vectors extraction method is that every time that a projection vector is calculated, the information of the hyperspectral data set which can be represented by the extracted vector is automatically subtracted from the image. Hence, the information remaining in the image is directly the information that would be lost in the spectral decorrelation and reduction process if no more projection vectors were calculated. This fact enables the possibility of easily providing a stopping condition according to different quality measures such as the *Signal-to-Noise Ratio* (SNR) or the *Maximum Single Error* (MaxSE). On the other hand, the spectral reduction achieved is directly proportional to the number of projection vectors extracted, which allows to easily define an stopping condition according to the compression ratio desired.

Finally, the amount of operations required for extracting the projection vectors is proportional to the number of pixels to be spectrally decorrelated and reduced. Therefore, the data set can be split in blocks of pixels, independently performing the spectral decorrelation and reduction of each of them, without increasing the overall computational effort. This fact allows a higher level of parallelization and a reduction of the amount of data

required in the memory for the compression. Moreover, dividing the data into blocks and independently processing each of them also increases the error-resilience of the process, which is always desirable, specially in the space environment. Furthermore, splitting the data set into blocks of pixels and independently processing them with the HyperLCA transform increases its overall compression performance, as demonstrated in the experiments shown in Section 4.4.

4.3.1 Description of the HyperLCA algorithm

The HyperLCA algorithm sequentially extracts projection vectors from the hyperspectral data set. The set of extracted vectors is then used for projecting the hyperspectral image, obtaining a spectral decorrelation and reduction of the data. The reduction achieved within this process directly depends on the number of projection vectors used. Obtaining more projections vectors provides better decompressed images but smaller compression ratios, understanding the compression ratio as the relation between the size of the real image and the compressed one (the bigger the better). Since the projection vectors are sequentially extracted, the sooner the algorithm stops, the higher the compression ratio will be. The proposed method takes advantage of this fact for allowing the user to determine a minimum desired compression ratio which is used for calculating the maximum number of projections vectors to be extracted. By doing so, the proposed method ensures the achievement of a compression ratio which will always be higher than the compression ratio specified by the user.

Moreover, each time that a projection vector is extracted, the information of the image that can be spanned by the extracted vector is subtracted from the image. Accordingly, the remaining information directly corresponds with the information that would be lost if no more projection vectors were calculated. This can be used for stopping the sequential extraction of projection vectors once that a specific accuracy level is achieved in the reduction step, according to any desired evaluation measurement, without reaching the maximum number of projection vectors to be extracted, as described in Section 4.3.2. This way, the proposed method guarantees that if an accurate enough compression of the hyperspectral image is obtained before extracting all the projection vectors according to the minimum compression ratio specified by the user, the algorithm will stop, providing a higher compression ratio at a lower computational burden.

Finally, both the spectrally decorrelated and reduced data set as well as the extracted projection vectors are set as outputs of the HyperLCA algorithm. This is due to the fact that the projection vectors used for spectrally decorrelating and reducing the hyperspectral image are also needed for recovering the original one.

The pseudocode that describes the HyperLCA algorithm for spectrally decorrelating and reducing the hyperspectral data set is presented in Algorithm 4.1, where '$\circ$' represents the inner product between two vectors. First of all, the maximum number of projection vectors to be extracted, $p_{\max}$, is calculated, in line 1, according to the minimum compression ratio desired in the spectral reduction step, mcr. Secondly, the hyperspectral data, M, is centered and stored in M_c, in line 2. Then, the projection vectors are sequentially extracted in lines 4 to 14. In this process, the bright of each pixel within the M_c image is first calculated in lines 5 to 7. Then, the pixel with the highest bright is selected as projection vector in line 8, and stored in U after its normalization in lines 9. After that, the information that can be spanned by the selected projection vector is subtracted from the M_c image in lines 10 to 12. Once that all the projection vectors have been calculated, the spectrally decorrelated and reduced data set, M_r is finally obtained, in line 15, as the projection of the real hyperspectral image, M, onto the subspace spanned by the set of projection vectors extracted, U.

Algorithm 4.1: Spectral decorrelation and reduction

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, mcr

1 $p_{\max} = int(\frac{N_p \cdot N_b}{mcr \cdot (N_p + N_b)})$; {Maximum number of projection vectors to extract}
2 $M_c = [x_1, x_2, ..., x_{N_p}]$ {Centralized version of M}
3 $U = [u_1, u_2, ..., u_{p_{\max}}]$; {Gram Schmidt orthonormalized vectors. Projection vectors.}
4 **for** $p = 1$ **to** $p_{\max}$ **do**
5 **for** $j = 1$ **to** N_p **do**
6 $b_j = x_j \circ x_j$;
7 **end**
8 $j_{\max} = \arg\max(b_j)$;
9 $u_p = x_{j_{\max}} / \|x_{j_{\max}}\|$;
10 **for** $j = 1$ **to** N_p **do**
11 $x_j = x_j - (x_j \circ u_p) \cdot u_p$;
12 **end**
13 {Additional stopping condition.}
14 **end**
15 $M_r = U^T \cdot M$; {Decorrelated and reduced hyperspectral image.}

Outputs: M_r, U

This methodology provides one important advantage for compressing and decompressing images for hyperspectral imaging applications. As described in lines 8 and 9 of Algorithm

4.1, each projection vector is selected as the vector which spans the pixel with the largest amount of remaining information. By doing this, it is guaranteed that the most different pixels within the data set are preserved through the spectrally decorrelation and reduction steps of the compression-decompression process. This fact makes this compression approach specially suitable for applications in which some pixels may be very different from the rest of the pixels, such as anomaly detection, target detection or classification.

4.3.2 Possible stopping conditions within the HyperLCA algorithm

As aforementioned, matrix M_c contains the information that would be lost if no more projection vectors were extracted. This information can be used for setting an stopping condition in the sequential projection vectors extraction process, based on the amount of losses desired within the compression-decompression process. For such purpose two different approaches can be followed:

- *Global error approaches* measure the global or average error in the decompressed image with respect to the original one. Despite these approaches provide a good idea of the accuracy of the compression-decompression process, they do not necessary yield an accurate reconstruction of the anomalous pixels.
- *Single error approaches* focus on measuring the largest single errors in the decompressed image. Despite these approaches do not necessary provide a very good idea of the overall compression-decompression performance, they are more suitable for verifying the accuracy of the reconstruction of the anomalous pixels.

Although many different metrics can be efficiently applied to the proposed HyperLCA Algorithm, we will focus on the RMSE, the SNR and the MaxSE metrics since these are widely used state-of-the-art metrics for evaluating the compression-decompression performance. While the RMSE and SNR are global evaluation metrics that are useful for verifying an average good compression-decompression performance, the MaxSE, despite its simplicity, is more suitable for ensuring that the anomalous pixels are preserved through the compression-decompression process. Any of the possible stopping conditions based on these metrics would be placed in line 13 of the Algorithm 4.1, and would prevent the extraction of more projection vectors if the losses achieved are small enough according to the quality metric used.

4.3.2.1 RMSE based stopping condition.

The *Root Mean Square Error* (RMSE) is a frequently used metric for measuring the average differences between the real hyperspectral image and the decompressed one. A low RMSE represents low average compression-decompression errors. This metric is defined as:

$$\text{RMSE} = \frac{1}{N_p \cdot N_b} \cdot \sqrt{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (M_{i,j} - M'_{i,j})^2} \tag{4.1}$$

where M and M' refer to the real hyperspectral image and to the decompressed one, and N_p and N_b refers to the number of pixels and the number of bands in the hyperspectral image, respectively. Since M_c contains the information that cannot be represented with the already extracted vectors, it could be directly calculated within the HyperLCA algorithm as:

$$\text{RMSE} = \frac{1}{N_p \cdot N_b} \cdot \sqrt{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (M_{c_{i,j}})^2} \tag{4.2}$$

4.3.2.2 SNR based stopping condition.

The *Signal-to-Noise Ratio* (SNR) generally compares the level of the desired signal to the level of background noise. It is defined as the ratio of signal power to the noise power, often expressed in decibels. In the compression scenario, the SNR metric measures the ratio between the real hyperspectral image power and the compression-decompression looses power, as shown in Equation 4.3, where M and M' refer to the real hyperspectral image and to the decompressed one, and N_p and N_b refers to the number of pixels and the number of bands in the hyperspectral image, respectively.

$$\text{SNR} = 10 \cdot \log_{10}\left(\frac{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (M_{i,j})^2}{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (M_{i,j} - M'_{i,j})^2}\right) \tag{4.3}$$

As aforementioned, $(M_{i,j} - M'_{i,j})$ directly corresponds with $M_{c_{i,j}}$. Hence, the SNR can be calculated within the proposed HyperLCA algorithm every time that a new projection vector is extracted as:

$$\mathrm{SNR} = 10 \cdot \log_{10}\left(\frac{\sum_{i=1}^{N_b}\sum_{j=1}^{N_p}(M_{i,j})^2}{\sum_{i=1}^{N_b}\sum_{j=1}^{N_p}(M_{c_{i,j}})^2}\right) \tag{4.4}$$

As a further optimization, $\sum_{i=1}^{N_b}\sum_{j=1}^{N_p}(M_{i,j})^2$ can be calculated just once at the beginning of the compression process, since it evaluates just the power of the real hyperspectral image, M.

4.3.2.3 MaxSE based stopping condition.

The *Maximum Single Error* (MaxSE) evaluates the maximum absolute difference between the real hyperspectral image and the compressed-decompressed one. The MaxSE that would be obtained in the compression-decompression process within the HyperLCA algorithm if no more projection vectors were extracted can be directly calculated, in each iteration, as the maximum absolute value of M_c.

4.3.3 Effect of the block partition within the HyperLCA algorithm

As it can be seen in Algorithm 4.1 the amount of operations to be performed directly depends on the number of pixels of the image to be compressed. Hence the hyperspectral image can be split into smaller sub-images or blocks of pixels, and the amount of operations required for processing all the individual blocks will be similar to the amount of operations required for processing the entire image. This fact provides several advantages:

- Speed up due to parallelization. The compression process can be parallelized into the compression of smaller images, speeding up the compression of the entire image. Moreover, for those sensors which do not collect all the image at once, such as the pushbroom or whiskbroom sensors, the compression process can be initiated before the entire image is collected.
- Reduction of the resources required. Since each block of pixels can be independently compressed, it is not necessary to keep in memory the entire image for its compression, but just a portion of it. This is specially useful in situations in which the sensor does not collect the image at once, but small portions of it, which could be directly compressed and stored before the entire image is captured.

- Error-resilience. Independently compressing each block ensures that if one error occurs during the compression of one block, this error will not be transmitted to another block. This is particularly useful for space on-board compression, since errors can take place due to the effect of the radiation.

- Better relation between compression performance and preservation of anomalous pixels. The HyperLCA algorithm focuses on extracting the projection vectors which best represents the most different pixels within the data set. If there are many pixels which are very different within the data set, more projections vectors are required for accurately compressing-decompressing the data. On the other hand, the variance among a small set of pixels is much smaller than the variance between a large number of them. Hence, the HyperLCA algorithm will be able to better compress individual blocks of pixels than the entire image. However, if the number of pixels per blocks is too small, the size of the projection vectors is comparable to the size of the reduced image, and the compression performance decreases. As a result, dividing the image to be compressed with the HyperLCA algorithm into blocks of pixels will provide better compression performance and lower compression-decompression losses, as long as the number of pixels per block is much larger than the size of the set of projection vectors needed.

4.3.4 Block diagram of the compressor

Hyperspectral sensors can be classified according to how they sense the hyperspectral data. Pushbroom and whiskbroom scanners capture the hyperspectral images line by line or pixel by pixel, respectively. There also exist snapshot sensors which capture the entire hyperspectral cube at once, and panchromatic sensors which are able to collect a hyperspectral image, band by band, by using a liquid crystal tunable filters. Pushbroom and whiskbroom scanners are the most widely used sensors in remote sensing applications, since they take advantage of the movement of the aircraft of satellite that carries them for capturing the data. Figure 4.1 graphically describes the sensing process with pushbroom and whiskbroom scanners.

Once that the data is collected by the hyperspectral sensor, it can be stored in *band-squential* (BSQ) or *band-interleaved* (BI) order. In BSQ order, all the samples in a particular band are stored in raster order before storing the following band. On the other

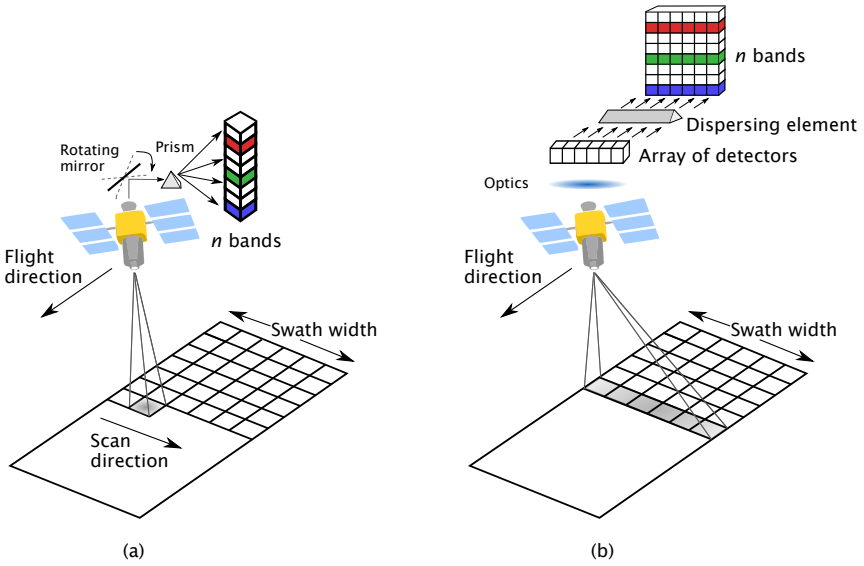


FIGURE 4.1: Remote sensing scanners. a) WhiskBroom b) Pushbroom

hand, in BI order, a particular sample is stored in all its spectral channels before the compression of the next sample in raster order.

The HyperLCA transform has been thought for performing the spectral decorrelation and reduction stage of the on-board compression process of hyperspectral images under real-time constraints. For such purpose it has been designed for working with small blocks of pixels of the data set which are independently compressed. It is not necessary that the pixels within a particular block are spatially aligned in a 2D image-like way. This fact makes the HyperLCA transform specially suitable for spectrally decorrelating and reducing hyperspectral data provided by pushbroom or whiskbroom sensors in BI order, since it can pipeline the sensing and compression processes, as shown in Figure 4.2.

On the other hand, after the spectral decorrelation and compression with the HyperLCA transform, further compression can be achieved by applying extra compression stages such as the spatial compression and the entropy coding. However, we consider that introducing a spatial compression stage before the entropy coding does not really entail an advantage for the HyperLCA transform. While the HyperLCA transform has the advantage of being able to independently spectrally compress small blocks of pixels of the image, an efficient spatial compression cannot be independently applied to small blocks of pixels of the image. Due to this reason, we consider that it is more efficient to apply just the entropy coding stage after the spectral decorrelation and reduction within the HyperLCA transform. Moreover, the fact of removing the spatial decorrelation and reduction stage

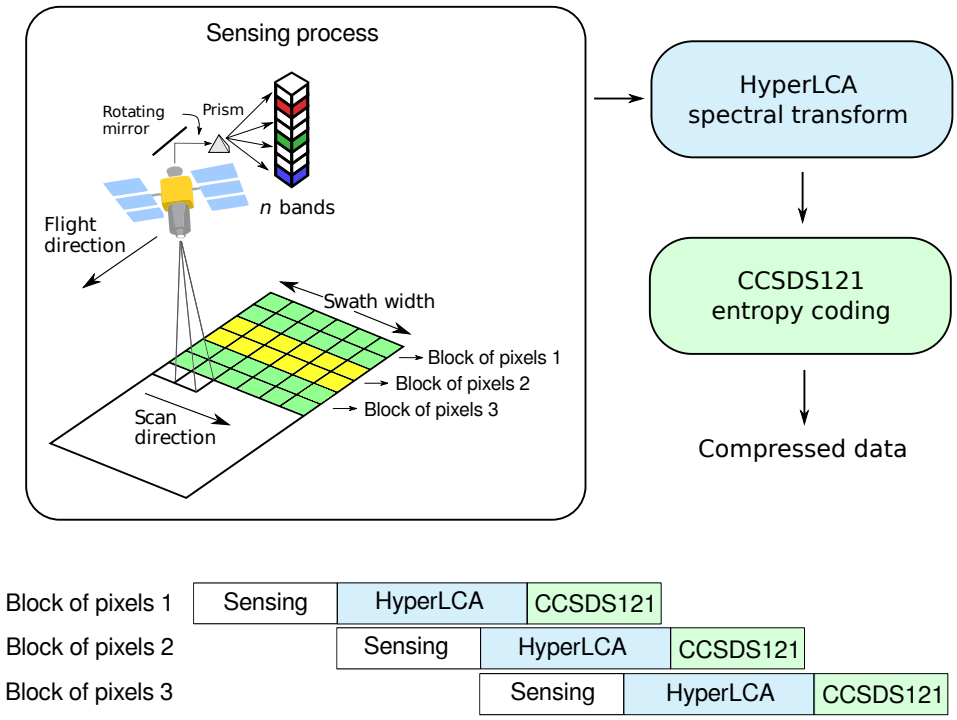


FIGURE 4.2: Block diagram of the compressor.

from the compression process also reduces its overall computational complexity, making it more viable for applications under real-time constraints, as shown in Figure 4.2.

4.4 Compression results within the HyperLCA algorithm

This Section discloses the most significant results obtained with the HyperLCA algorithm when it is utilized to perform the spectral decorrelation and reduction of hyperspectral images for their compression. Several images from the *Airborne Visible/Infrared Imaging Spectrometer* (AVIRIS) [107] and the Hyperion [108] sensors have been used in the experiments described herein, including images from various stages of the acquisition chain: uncalibrated, radiance, and reflectance images. The performance of the HyperLCA algorithm is evaluated by comparing the compression results with those obtained, for the same hyperspectral images, with the PCA transform, which provides the best optimal spectral decorrelation and reduction among the state-of-the-art algorithms for this task. Finally, the HyperLCA algorithm is tested for ultraspectral images compression, using data from the *Atmospheric Infrared Sounder* (AIRS) [6].

4.4.1 Hyperspectral and ultraspectral data used for the experiments

The HyperLCA transform proposed in this Thesis spectrally decorrelates and reduces the spectral data. After applying the proposed transform, the number of bands to be compressed in the subsequent compression stages is much smaller than the original number of bands of the hyperspectral image, according to the compression ratio achieved by the HyperLCA transform. This fact reduces the complexity of the compression stages applied after the HyperLCA transform, making it very suitable for images with a large number of bands (hyperspectral and ultraspectral images). On the other hand, it is not conceived for images with a small number of bands. Due to this reason, multispectral images have not been considered in the experiments.

In order to evaluate the performance of the proposed methodology for compressing and decompressing real hyperspectral and ultraspectral data, we have used a test bench that comprises images from different sensors. In particular, this test bench contains 8 images from the AVIRIS sensor, 3 images from the Hyperion sensor and 2 images from the AIRS sensor.

Hyperspectral data collected by the AVIRIS sensor cover the visible to near-infrared spectrum, from 0.4 to 2.5 μm, with 224 contiguous channels, approximately 0.01 μm wide each. The AVIRIS data used for the experiments include images from various stages of the acquisition chain: uncalibrated, radiance, and reflectance images. The size of all the AVIRIS images used has been fixed to 512x512 pixels. The Hyperion data have 242 spectral components which cover wavelengths from 0.357 to 2.576 μm. The images from the Hyperion sensor used in the experiments are uncalibrated images, and their size have been fixed to 512 lines of 256 pixels each. The AIRS sensor collects information in the visible and infrared parts of the spectrum, from 3.7 to 15.4 μm, with 1501 channels. The size of the images from the AIRS sensor used has been fixed to 128x64 pixels. All these images are represented in Figure 4.3 as false color RGB images. Table 4.1 summarizes the characteristics of the different images used in the experiments. Column *Code* shows the code used for referencing the images described in this table with their false color representation shown in Figure 4.3.

TABLE 4.1: Characteristics of the images used for the experiments

Image Name	**Sensor**	**Fixed Size**	**Spectral Coverage μm**	Stage	Code
Yellowstone (Sc 0)	AVIRIS	512x512x224	0.4 - 2.5	Radiance	AV1
Yellowstone (Sc 11)	AVIRIS	512x512x224	0.4 - 2.5	Radiance	AV2
Yellowstone (Sc 18)	AVIRIS	512x512x224	0.4 - 2.5	Radiance	AV3
Low Altitude	AVIRIS	512x512x224	0.4 - 2.5	Reflectance	AV4
Lunar Lake	AVIRIS	512x512x224	0.4 - 2.5	Reflectance	AV5
Moffet Field	AVIRIS	512x512x224	0.4 - 2.5	Reflectance	AV6
Hawaii	AVIRIS	512x512x224	0.4 - 2.5	Uncalibrated	AV7
Maine	AVIRIS	512x512x224	0.4 - 2.5	Uncalibrated	AV8
Yellowstone (Sc 0)	AVIRIS	512x512x224	0.4 - 2.5	Uncalibrated	AV9
Yellowstone (Sc 11)	AVIRIS	512x512x224	0.4 - 2.5	Uncalibrated	AV10
Yellowstone (Sc 18)	AVIRIS	512x512x224	0.4 - 2.5	Uncalibrated	AV11
Erta Ale	Hyperion	256x512x242	0.357 - 2.576	Uncalibrated	Hy12
Lake Monona	Hyperion	256x512x242	0.357 - 2.576	Uncalibrated	Hy2
Mt. St. Helens	Hyperion	256x512x242	0.357 - 2.576	Uncalibrated	Hy3
Granule 9	AIRS	128x64x1501	3.7 - 15.4	Radiance	AI1
Granule 16	AIRS	128x64x1501	3.7 - 15.4	Radiance	AI2

4.4.2 Evaluation of the performance of the HyperLCA algorithm for spectrally decorrelating and reducing the hyperspectral image.

The HyperLCA Algorithm, as well as the PCA transform, allows not only to spectrally decorrelate the information in the hyperspectral image, but also spectrally reducing it. This is achieved by projecting the hyperspectral image onto a smaller subspace, spanned by a set of projection vectors. The amount of data in the projected subspace is directly determined by the number of projection vectors extracted by the HyperLCA or PCA transforms, and so is the compression ratio achieved at this compression stage. The compression performance obtained in the spectral reduction of the data set by the HyperLCA algorithm is compared in this Section with the compression performance obtained with the PCA for this same stage. For such purpose the hyperspectral images collected by the AVIRIS and Hyperion sensors described in Section 4.4.1 have been used.

Figure 4.4 graphically shows the results obtained with the PCA and the HyperLCA transforms for spectrally reducing the data sets. The HyperLCA performance is tested using three different block sizes: 256, 512 and 1024 pixels per block. First of all, Figures 4.4a and 4.4b show the mean values of SNR and MaxSE obtained for all the data sets, for different compression ratios, respectively. Similarly, Figures 4.4c and 4.4d, and Figures 4.4e and 4.4f graphically show the best and worst spectral reduction performance achieved by the different transforms for any of the of the datasets, respectively.

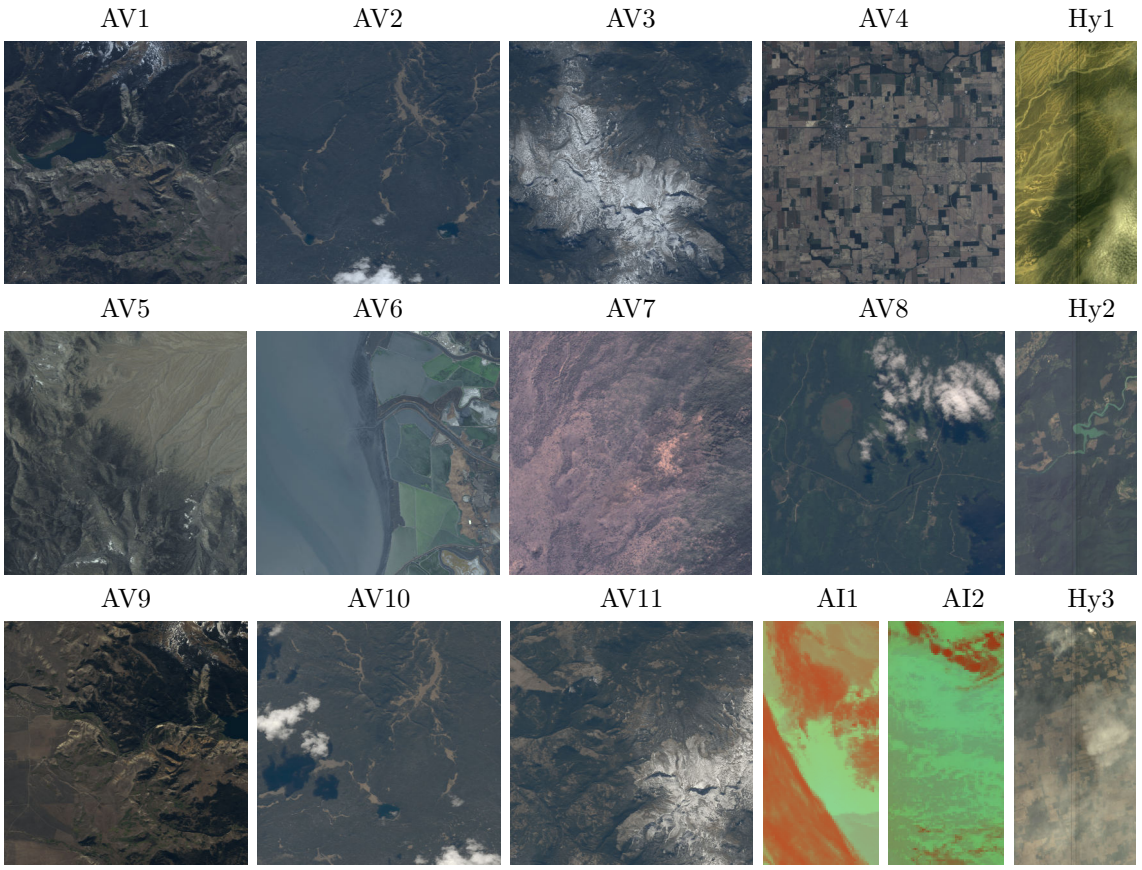


FIGURE 4.3: False color representation of the spectral images described in Table 4.1.

According to the results shown in Figure 4.4, it is demonstrated that the HyperLCA algorithm provides good results for spectrally reducing the data set using different block sizes. However, the best results have been obtained with the biggest block size used. This is due to the fact that when the image is split in smaller groups, a set of orthogonal vectors is obtained for each group, and hence, for obtaining the same compression ratio, each block has to be compressed using less projection vectors. If there is little variability between groups, it is possible to compress many of them (a bigger group) with a single set of projection vectors with the same accuracy, obtaining a higher compression ratio. On the other hand, when there is too much variability between pixels the use of smaller groups may lead to better results. This facts correspond with the effect of the block partition in the HyperLCA algorithm described in 4.3.3. It is also observed that the average SNR obtained by the PCA transform is slightly better than the average SNR obtained by the HyperLCA transform. However, the worst SNR results obtained with the HyperLCA transform are better for some compression ratios than the worst results provided by the PCA transform. Similarly, the relation between the best and average SNR results provided by the PCA transform is higher than the relation between the

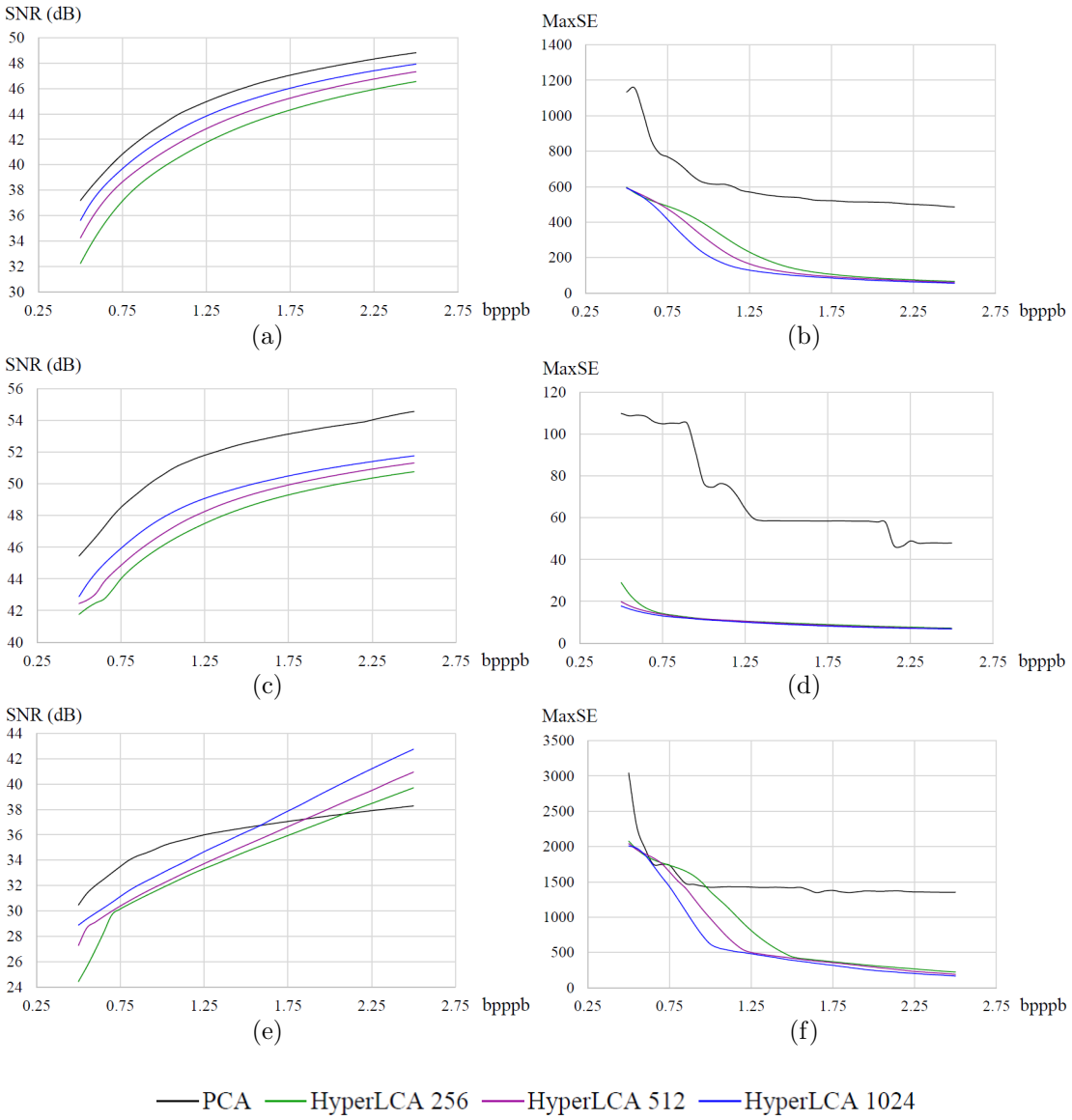


FIGURE 4.4: Spectral reduction performance with the PCA and HyperLCA transforms. a), b) Average SNR and MaxSE values. c), d) Best SNR and MaxSE values. e), f) Worst SNR and MaxSE values.

best and average results provided by the HyperLCA algorithm. This facts show that the HyperLCA algorithm provides more stable SNR results than the PCA transform for spectrally reducing hyperspectral images.

On the other hand, the performance of the HyperLCA transform, evaluated in terms of MaxSE, clearly surpasses the performance of the PCA transform in all the tested situations, as it is observed in Figures 4.4b, 4.4d and 4.4f. This is due to the fact that the HyperLCA is designed for representing the most different pixels within the data set, as

mentioned in 4.3.1, what makes this algorithm specially suitable for situations in which it is important to preserve the anomalous pixels through the compression-decompression process.

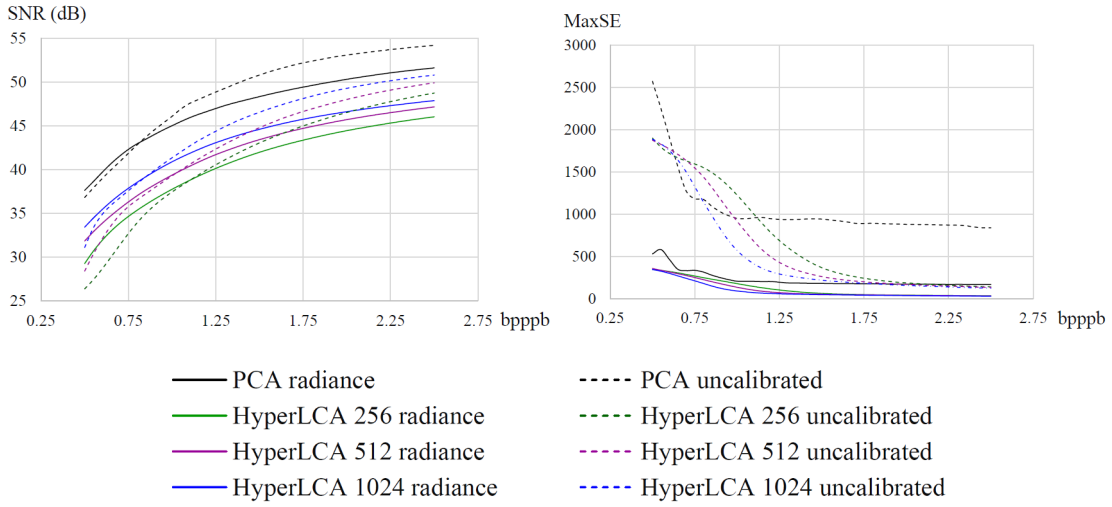


FIGURE 4.5: Average compression performance with the PCA and HyperLCA transforms for radiance and uncalibrated AVIRIS Yellowstone hyperspectral images.

As a further step, Figure 4.5 graphically shows the mean SNR and MaxSE obtained with the HyperLCA and the PCA transforms, just for the radiance AVIRIS Yellowstone images in comparison with their uncalibrated version. This results allows the evaluation of the effect of the calibration process in the compression performance. According to Figure 4.5 it is observed than the MaxSE values obtained in the radiance images with the HyperLCA and the PCA transforms are proportional but much lower that the MaxSE values obtained in their uncalibrated version. On the other hand, the SNR values tend to be higher in the uncalibrated images, using both transforms, specially for higher bpppb values.

4.4.3 Evaluation of the entire compression-decompression process within the HyperLCA algorithm

A spectral transform and reduction is usually the first stage of the hyperspectral coding process. Hence, to assess the compression gains within the HyperLCA algorithm, compared with the PCA transform, they have to be tested in conjunction with an image coder. The purpose of these experiments is to evaluate the HyperLCA spectral transform as part of the compression process, and hence we do not desire to introduce further compression losses, and thus, just lossless coders have been used. The reason behind this decision is that

we would like to preserve the anomalous pixels through the compression-decompression process so that hyperspectral applications such as anomaly detection, target detection or classification do not get negatively affected by the compression process, since the HyperLCA algorithm is particularly suitable for this purpose.

Both the PCA and the HyperLCA transforms have been used in this experiment in conjunction with the lossless coders which conform the standards provided by the *Consultative Committee for Space Data Systems* (CCSDS) [106]. In particular, the PCA transform has been used in this Section in conjunction with the CCSDS122 in its lossless configuration [106]. The CCSDS122 performs a 2D wavelet transform of each band of the spectrally decorrelated and reduced image, as well as the entropy coding of the data. On the other hand, the HyperLCA Algorithm has been used in conjunction with the CCSDS121 [106] which only performs the entropy coding stage. Although this way we are loosing the possibility of obtaining higher compression performance by reducing the spatial redundancies within the spectrally reduced hyperspectral image, we consider that this approach reports more benefits in terms of computational complexity and real-time performance, as explained in Section 4.3.4.

The compression performance using both the proposed HyperLCA+CCSDS121 and the PCA+CCSDS122 approaches is compared using the hyperspectral images collected by the AVIRIS and Hyperion sensors described in Section 4.4.1. Figure 4.6 graphically shows the results obtained in these experiments, in which, just the spectrally decorrelated and reduced hyperspectral images have been further compressed with the CCSDS121 and CCSDS122 compressors. The projection vectors have been represented without encoding, i.e. using 16 bpppb in both approaches, since the size of the projection vectors provided by the PCA is negligible in relation with the size of the spectrally reduced image. However, the HyperLCA produces several sets of projection vectors (one set per block), which may negatively affect the compression performance. As a further improvement, each of these sets of projection vectors could also be compressed using the same CCSDS121 coder, which is already being used for coding the spectrally reduced image. Figure 4.7 shows the improvement in the performance of the HyperLCA+CCSDS121 compressor when using the CCSDS121 coder for independently entropy coding both the spectrally reduced hyperspectral image and the set projection vectors used for reducing it. In this Figure, dashed lines show the performance of the HyperLCA+CCSDS121 approach when using the CCSDS121 coder for compressing just the spectrally reduced image, while solid lines show the performance of the HyperLCA+CCSDS121 approach when compressing both the reduced image and the projection vectors within the CCSDS121 coder.

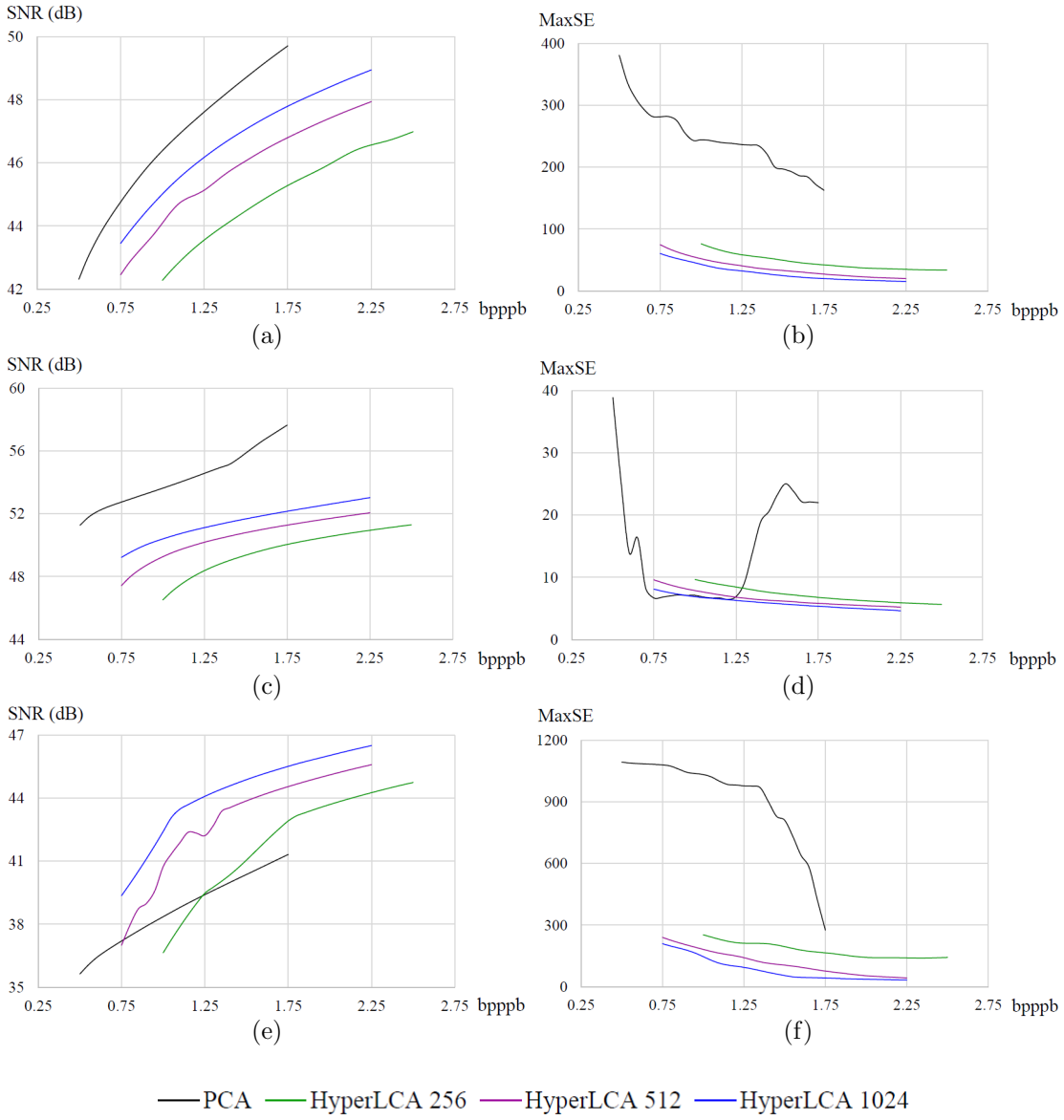


FIGURE 4.6: Hyperspectral compression performance with the PCA+CCSDS122 and HyperLCA+CCSDS121 compressors. a), b) Average SNR and MaxSE values. c), d) Best SNR and MaxSE values. e), f) Worst SNR and MaxSE values.

As expected, the results shown in Figure 4.6 show higher compression performance (higher compression ratios for the same SNR values) for both approaches, the PCA+CCSDS122 and HyperLCA+CCSDS121 compressors, than when using just the spectral reduction and transform. The PCA+CCSDS122 approach provides slightly higher average SNR results than the HyperLCA+CCSDS121 compressor, as shown in Figure 4.6a. The SNR values obtained for the best scenario using the PCA+CCSDS122 compressor are also slightly better than the best SNR results provided by the HyperLCA+CCSDS121 approach, as shown

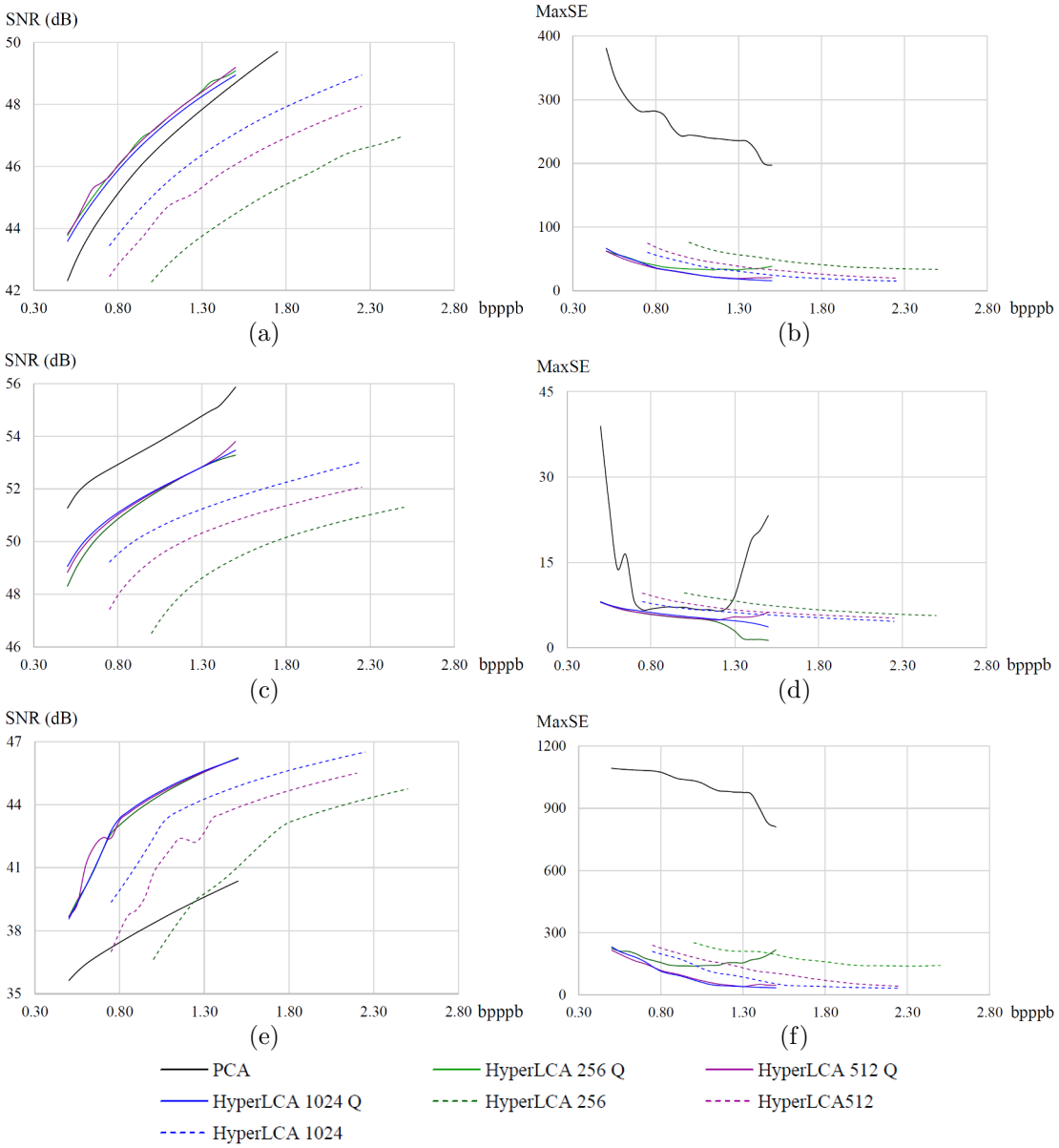


FIGURE 4.7: Improvement in the compression performance of the HyperLCA+CCSDS121 approach when compressing both the spectrally reduced hyperspectral image and the projection vectors within the CCSDS121 coder.

in Figure 4.6c. However, the worst SNR values obtained with the HyperLCA+CCSDS121 compressor, shown in Figure 4.6e, are higher than the worst SNR values obtained with the PCA+CCSDS122 approach. These facts demonstrate that the HyperLCA+CCSDS121 approach provides more stable results than the PCA+CCSDS122 compressor, being less affected by the variability of the data to be compressed. Moreover, the MaxSE results obtained when using the HyperLCA+CCSDS121 method clearly surpass the MaxSE results

provided by the PCA+CCSDS122 compressor in all the situations, as shown in Figures 4.6b, 4.6d and 4.6f.

In this scenario, it is important to remark that the PCA+CCSDS122 approach is performing a spectral decorrelation and reduction first, using the PCA algorithm, and a spatial compression and entropy coding within the CCSDS122 compressor, while the HyperLCA+CCSDS121 approach is just performing the spectral decorrelation and entropy coding stages using the HyperLCA and CCSDS121 algorithms, respectively. Moreover, the HyperLCA+CCSDS121 approach has been independently applied to the different blocks of pixels of the image. These facts verifies that the HyperLCA+CCSDS121 approach provides not only competitive compression results, but also at a much lower computational complexity. On the other hand, the results shown in Figure 4.7 show that the performance obtained with the proposed HyperLCA+CCSDS121 compressor increases when using the CCSDS121 coder for entropy coding not only the spectrally reduced image but also the projection vectors, surpassing the compression performance of the PCA+CCSDS122 approach. According to the results shown in this Figure, it is observed that the proposed HyperLCA+CCSDS121 approach provide better average results than the PCA+CCSDS122 approach, according the SNR and MaxSE metrics, shown in Figures 4.7a and 4.7b. Moreover, the best and worst compression performance obtained with both approaches, shown in Figures 4.7c and 4.7d, and 4.7e and 4.7f, respectively, demonstrate that the HyperLCA+CCSDS121 compressor provides more stable results than the PCA+CCSDS122 approach.

These results verify the suitability of the HyperLCA for efficiently performing the spectral decorrelation and reduction stage within the entire compression process, at a much lower computational requirements than the PCA transform.

4.4.4 Ultraspectral image compression results within the HyperLCA algorithm

On-board ultraspectral sensors provide huge amounts of information which are mandatory to compress for reducing the amount of data to store and transfer. While hyperspectral imaging systems typically collect hundred of bands, ultraspectral systems collect thousands of them. This fact increases the complexity and the computational burden required for performing a spectral reduction based on any KLT approach, such as the PCA transform. These approaches need to calculate a set of eigenvalues and eigenvectors from

the covariance matrix, which number of lines and rows are both equal to the number of bands of the image under inspection. Hence, the size of the covariance matrix drastically increases for ultraspectral images, and so does the complexity of the eigenvalues and eigenvectors calculation process.

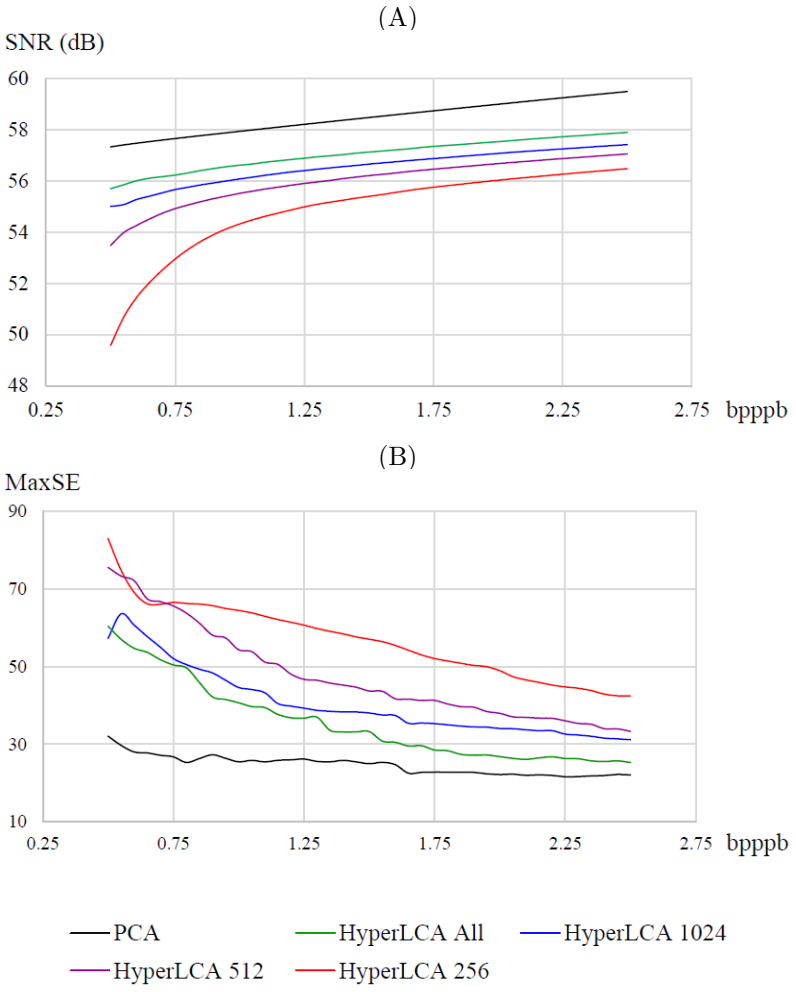


FIGURE 4.8: Spectral reduction of ultraspectral images using the HyperLCA and the PCA transforms.

The HyperLCA transform has been tested in this Section for providing a more computational efficient solution for spectrally decorrelating and reducing ultraspectral images. For such purpose, the two ultraspectral images collected by the AIRS sensor, described in Section 4.4.1 have been used. These two images have been spectrally decorrelated and reduced using both the PCA and the HyperLCA transforms for different compression ratios and block sizes. Figure 4.8 graphically shows the average results obtained in the experiments using these two images. Labels *HyperLCA 256*, *HyperLCA 512* and *HyperLCA 1024* refers to the different block sizes used in the compression-decompression

process within the HyperLCA transform. On the other hand, labels *HyperLCA All* and *PCA* refers to the results obtained using the HyperLCA and the PCA transforms for compressing the entire image at once.

Despite the PCA provides a slightly better compression performance in these experiments, the results shown in Figures 4.8a and 4.8b indicate that the HyperLCA transform still provides a very accurate compression at a much smaller computational complexity than the PCA transform, as demonstrated in Section 4.6. Figure 4.8a also shows that high SNR values can be obtained for very high compression ratios just by spectrally reducing the ultraspectral images within the HyperLCA transform. Furthermore, Figure 4.8b also shows that the maximum absolute errors obtained in these compressions are very small, with independence of the compression ratio achieved and the block sizes used. All these facts verify the suitability of the HyperLCA transform for spectrally decorrelating and reducing ultraspectral images.

4.5 Evaluation of the compression performance within the HyperLCA algorithm for hyperspectral imaging applications

Anomalies are defined as a small group of pixels whose spectral signature significantly differ from the rest of the pixels of the image. Anomaly detection applications exploit these facts for identifying items in the image that clearly differ from the background. However, preserving these pixels through the lossy compression-decompression process represents a challenging task, since any low-pass filter will probably destroy them. Moreover, due to the fact that these pixels represents a very small fraction of the entire image, having big compression-decompression losses for these pixels hardly affect the average compression-decompression losses, making it difficult to identify if there are anomalous pixels being lost during the compression.

Although the HyperLCA performs a lossy compression, it has been specially designed for preserving the anomalous pixels through the compression-decompression process, as described in Section 4.3.1. In this Section we evaluate how the anomaly detection algorithms are affected by the compression-decompression process within the HyperLCA algorithm. Two different hyperspectral images have been used for such purpose. Furthermore, the

effect of the HyperLCA compression approach in anomaly detection applications is compared with the effect of applying the PCA transform for compressing and decompressing the images before the anomaly detection process.

The first image was taken over the Rochester Institute of Technology by the *Wildfire Airborne Sensor Program* (WASP) Imaging System [109]. This system covers different parts of the visible, short, mid and long-wave infrared regions of the spectrum. A portion of the overall image, taken over a parking lot, with a size of 180x180 pixels and 120 bands, has been used in the study. Figure 4.9a is a false color representation of this dataset. The anomalies to be detected within this data set are fabric targets placed in the ground, as shown in Figure 4.9b.

The second image used was collected by the AVIRIS sensor over the World Trade Centre area in New York City on September 16, 2001. Figure 4.10a shows a false color representation of this image. The original data set contains 614x512 pixels, however, just a 200x200 portion is usually targeted in anomaly detection studies. Some thermal hot spots represent the anomalies within this portion of the image, as shown in Figure 4.10b.

In these experiments, these images are compressed and decompressed using the proposed HyperLCA+CCSDS121 process using different block sizes and compression ratios for the HyperLCA algorithm, as well as using the PCA+CCSDS122 process using the same compression ratios. After that, the decompressed images are processed using the *Reed-Xiaoli* (RX) detector [110, 111] for the anomalies detection. The results obtained in this process are graphically compared with the results obtained when using the same anomaly detection algorithms with the original images in Figures 4.9 and 4.10.

Figure 4.9 shows the results obtained for the Rochester Institute dataset. Fist of all, Figure 4.9a shows a false color representation of the real image used. Figure 4.9b shows the ground truth defined for this scene, which would represent the ideal results when using any anomaly detection algorithm. Figure 4.9c shows the result provided by the RX detector using the real dataset. Figures 4.9d to 4.9f and 4.9i to 4.9k show the results obtained with the RX detector using the compressed-decompressed image using the HyperLCA+CCSDS121 process, with different blocks sizes and compression ratios. Similarly, Figures 4.9g and 4.9l, and Figures 4.9h to 4.9m show the results obtained with the RX detector using the compressed-decompressed image provided by the HyperLCA+CCSDS121 and the PCA+CCSDS122 approaches, respectively, for compressing the entire image at once.

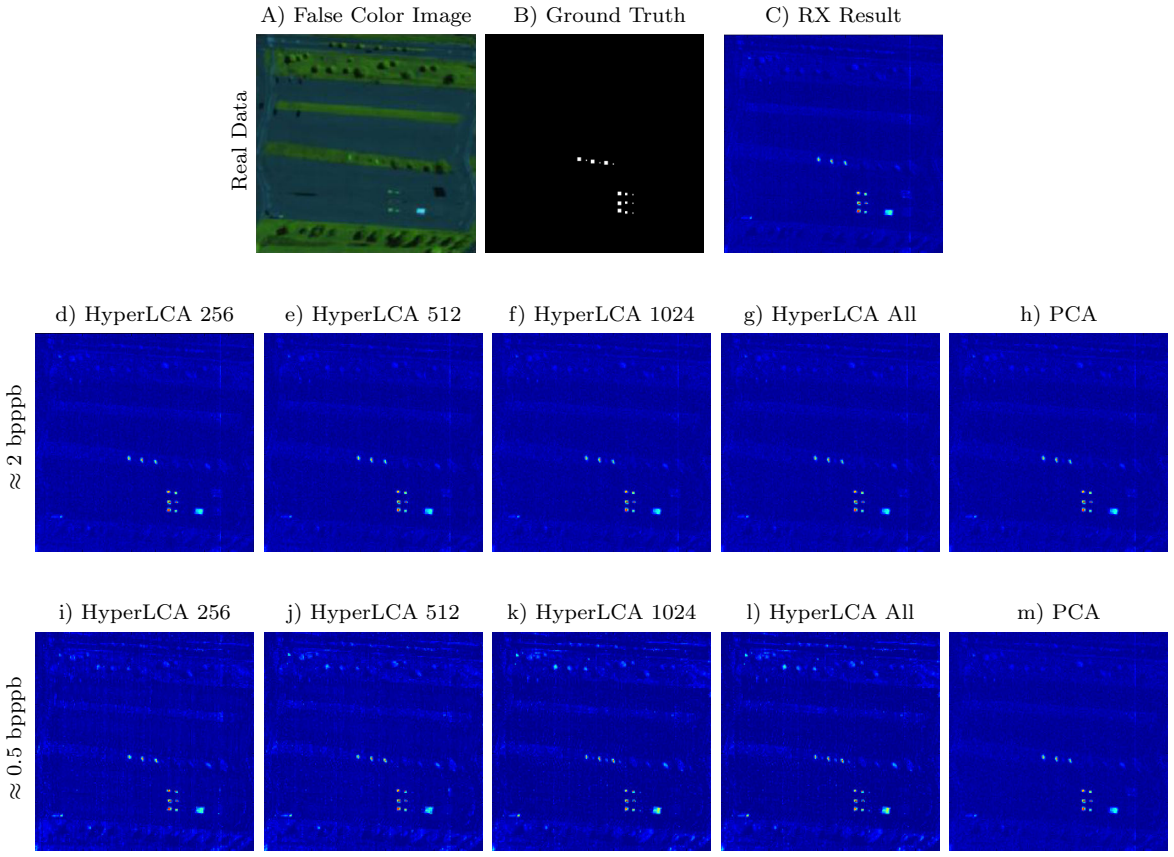


FIGURE 4.9: Effect of the compression-decompression process within the HyperLCA+CCSDS121 and PCA+CCSDS122 compressors on anomaly detection using the RX detector and the Rochester Institute image. a) False color image of the data set. b) Ground truth. c) Result provided by the RX detector using the real data. d-m) Results provided by the RX detector using the compressed-decompressed data.

.

Figure 4.10 shows the results obtained for the World Trade Center dataset. These results are organized in the same way than the results obtained with the Rochester Institute dataset. First, a false color representation of the image used, as well as the ground truth for the anomaly detection are shown in Figures 4.10a and 4.10b, respectively. Figure 4.10c shows the result provided by the RX detector using the real dataset. Finally, Figures 4.10d to 4.10m show the results obtained with the RX detector using the compressed-decompressed image provided by the different compression approaches with different compression ratios.

Figures 4.9 and 4.10 show that the anomalous pixels are preserved through the compression-decompression process with both the HyperLCA+CCSDS121 and PCA+CCSDS122 approaches. However, the quality of the reconstructed anomalous pixels is not really evaluated in these figures. For doing so, the pixels classified as anomalous

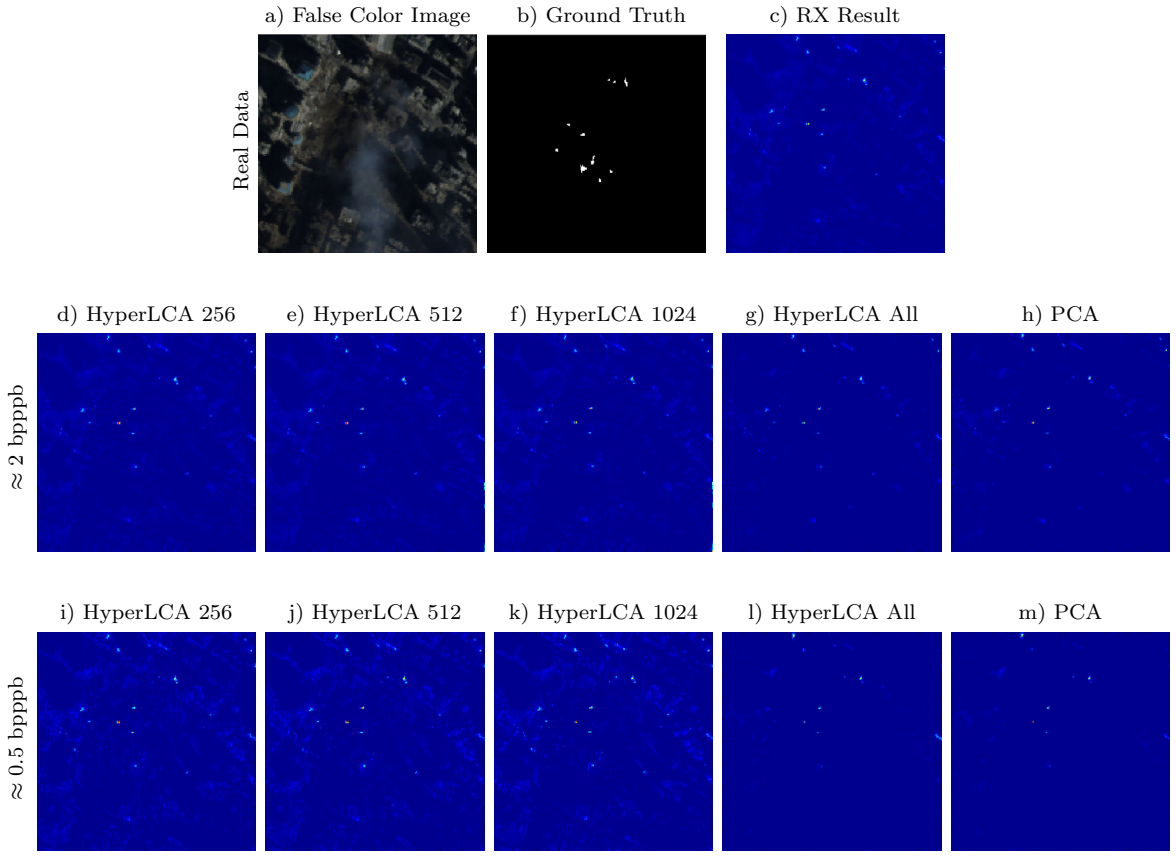


FIGURE 4.10: Effect of the compression-decompression process within the HyperLCA+CCSDS121 and PCA+CCSDS122 compressors on anomaly detection using the RX detector and the World Trade Center image. a) False color image of the data set. b) Ground truth. c) Result provided by the RX detector using the real data. d-m) Results provided by the RX detector using the compressed-decompressed data.

.

pixels in the ground truths of both scenarios, shown in Figures 4.9b and 4.10b, are specifically evaluated, by measuring the SNR between these pixels in the real images, shown in Figures 4.9a and 4.10a, and their corresponding compressed-decompressed versions using the different compression approaches. Table 4.2 show the results obtained in these experiments.

TABLE 4.2: Evaluation of the SNR between the real pixels labeled as anomalous pixels in the ground truth and their compressed-decompressed version using different approaches.

Image	bpppb	SNR between the real and the compressed-decompressed anomalous pixels				
		HyperLCA 256	HyperLCA 512	HyperLCA 1024	HyperL CAAll	PCA
AVE3	2	73.45	**75.10**	73.18	70.14	69.62
	0.5	58.93	**59.74**	59.11	57.26	53.64
WTC	2	78.11	**80.02**	79.80	75.07	73.30
	0.5	61.20	**63.32**	63.31	54.94	56.99

The RX detector provides an anomalies map, which values indicate the probability of each pixel for being an anomalous pixel. The RX detector measures this probability by comparing each pixel with the rest of the pixels of the image. Hence, obtaining a high value for a particular pixel in this anomalies matrix shows that the pixel differs from the rest of the pixels of the image. The anomalies matrices provided by the RX detector (Figures 4.9c and 4.10c) using the real images (Figures 4.9a and 4.10a) show much more pixels with relatively high scores than the number of pixels labeled as anomalous pixels in the ground truth corresponding to these images. Despite these pixels are not considered anomalous pixels in the ground truth, they clearly differ from the background pixels according to the RX detector and hence, should be the most challenging pixels for the compression decompression process. In order to evaluate the accurate preservation of these pixels through the compression-decompression process using both the HyperLCA+CCSDS121 and PCA+CCSDS122 compression approaches, as well as the effect of both compression-decompression processes in the results of the RX detector, the SNR between the anomalies matrix provided by the RX detector using both the real hyperspectral image and its compressed-decompressed version, using the different compression approaches, has been measured. Table 4.3 show the results obtained in these experiments.

Table 4.3: Evaluation of the SNR between the anomalies matrix obtained with the RX detector using the real images and their compressed-decompressed versions provided by the different approaches.

Image	bpppb	SNR between the anomalies matrix provided by the RX detector				
		HyperLCA 256	HyperLCA 512	HyperLCA 1024	HyperL CAAll	PCA
AVE3	2	**13.69**	13.68	13.62	4.84	4.83
	0.5	**8.37**	7.83	6.77	1.92	1.91
WTC	2	**15.05**	9.76	8.99	6.33	7.14
	0.5	**7.65**	7.57	6.56	2.79	2.72

Results shown in Tables 4.2 and 4.3 demonstrate that the anomalous and the most different pixels are better preserved through the compression-decompression process using the HyperLCA+CCSDS121 compressor for independently compressing blocks of 256, 512 or 1024 pixels than when using the PCA+CCSDS122 approach.

TABLE 4.4: Number of flops required for achieving the different steps of the PCA and HyperLCA transforms for spectrally reducing a hyperspectral image.

Process	PCA transform	HyperLCA transform
Image centralization	$2(N_p \cdot N_b) + N_b$	
Covariance matrix calculation	$2 \cdot {N_b}^2 \cdot {N_p}^2$	-
Projection vectors calculation	$4 \cdot {N_b}^3$	$6p \cdot N_p \cdot N_b + 4p \cdot N_b$
Image reduction (projection)	$p \cdot N_p(2N_b - 1)$	
Inverse Transform	$N_b \cdot N_p(2p - 1)$	

4.6 Computational complexity of the HyperLCA transform

The HyperLCA algorithm has been designed for dividing the data set into blocks of pixels and independently compressing each of then. This fact easies the parallel implementation of the algorithm making it more suitable for applications under real-time constraints. Besides the obvious gains that this fact brings in terms of faster computational times, the HyperLCA algorithm has the advantage that it uses simple mathematical operations which can be easily parallelized, avoiding complex matrix operations like for instance computing the eigenvalues and eigenvectors, operations which are present in all the KLT based compression approaches. Furthermore, the HyperLCA algorithm performs less FLOPs for achieving the same compression ratios, in the spectral decorrelation and reduction stage of the compression process, than the PCA transform. Table 4.4 contains the expressions of the amount of FLOPs performed by both the PCA and HyperLCA transforms, where N_p and N_b refers to the number of pixels and bands of the hyperspectral image, respectively; and p refers to the number of projection vectors to be extracted according to the reduction ratio desired.

The expressions shown in Table 4.4 for the HyperLCA transform assume that the entire image is decorrelated and reduced as a single block of N_p pixels. If the hyperspectral image is divided in independent blocks of $N_{p_{\text{block}}}$ pixels, the number of FLOPs needed for processing each block perfectly matches the expression of the table just by using $N_{p_{\text{block}}}$ instead of N_p as the number of pixels. However, for achieving the same reduction ratio, the number of projection vectors to be extracted per block, p, decreases as the image is divided into smaller blocks of pixels, as explained in Section 4.3.3. Due to this reason, the total number of FLOPS performed by the HyperLCA transform is smaller as the size of the blocks decreases, according to the expression shown in Equation 4.5.

$$FLOPs\ HyperLCA\ (N_b, N_p, p) < FLOPs\ HyperLCA\ (N_b, N_{p_{\text{block}}}, p_{\text{block}}) \tag{4.5}$$

The number of projection vectors used for achieving a particular reduction ratio when processing the entire image as a single block p is always larger than the number of projection vectors used for reducing smaller blocks of the image, p_{block}, as shown in Equation 4.6, where mcr refers to the minimum compression ratio achieved in the spectral reduction of the data set.

$$p = int(\frac{N_p \cdot N_b}{mcr \cdot (N_p + N_b)}) < p_{block} = int(\frac{N_{p_{\text{block}}} \cdot N_b}{mcr \cdot (N_{p_{\text{block}}} + N_b)}) \tag{4.6}$$

Tables 4.5 and 4.6 show the number of FLOPs needed for compressing the different images that have been used in the experiments, which are described in Table 4.1. The number of FLOPs is represented in logarithmic base in both tables as it is easier to appreciate the differences in orders of magnitude. Table 4.5 compares the number of FLOPs required for processing these hyperspectral images using the PCA and HyperLCA transforms. The number of FLOPs achieved by the HyperLCA shown in this table has been measured by processing each of the images as a single block of N_p pixels, which represents the worst scenario for the HyperLCA algorithm in terms of computational burden. On the other hand, Table 4.6 compares the number of FLOPs required by the HyperLCA for reducing the same hyperspectral images, when different block sizes are used. These measurements have been done for different reduction ratios, as shown in columns labeled as *bpppb* in both tables. Both tables also show the number FLOPs required for performing the inverse transform needed for recovering the original image from the spectrally reduced one. This process is exactly the same for both algorithms, and consists on multiplying the pixels from the reduced image by the projection vectors extracted by each of the methods. The number of FLOPs required by the inverse transform when the HyperLCA is applied to small blocks is smaller since the number of extracted projection vectors decreases.

According to the results shown in Table 4.5, it is observed that the number of FLOPs performed by the HyperLCA algorithm for spectrally reducing the data sets is much smaller than the number of FLOPs performed by the PCA transform (about 5 orders of magnitude in all the experiments performed). Moreover, according to the results shown in Table 4.6 it is also observed that the efficiency of the HyperLCA, in terms of the number of FLOPs, increases when the image is divided in blocks of pixels. Independently processing

TABLE 4.5: Number of FLOPs (in logarithmic scale) performed by the PCA and HyperLCA transforms for spectrally decorrelating and reducing hyperspectral images, according to the compression ratio achieved and the size of the image.

Image characteristics			bpppb	FLOPs		
Sensor	N_p	N_b		PCA	HyperLCA	Inverse Transform
AVIRIS	262144	224	2	15.84	10.12	9.51
			1	15.84	9.83	9.20
			0.5	15.84	9.53	8.88
Hyperion	131072	242	2	15.30	9.90	9.29
			1	15.30	9.62	8.99
			0.5	15.30	9.32	8.68
AIRS	8192	1501	2	14.48	10.19	9.59
			1	14.48	9.90	9.29
			0.5	14.48	9.60	8.99

TABLE 4.6: Number of FLOPs (in logarithmic scale) performed by the HyperLCA transforms for spectrally decorrelating and reducing hyperspectral images according to the block size used.

Image			bpppb	FLOPs per one single block			FLOPs for the entire image		
Sensor	N_p	N_b		256 pixels	512 pixels	1024 pixels	256 pixels	512 pixels	1024 pixels
AVIRIS	262144	224	2	6.85	7.27	7.63	9.86	9.98	10.04
			1	6.58	6.97	7.35	9.59	9.68	9.76
			0.5	6.29	6.68	7.06	9.30	9.39	9.47
Hyperion	131072	242	2	6.91	7.32	7.70	9.62	9.73	9.81
			1	6.61	7.05	7.42	9.32	9.46	9.53
			0.5	6.32	6.79	7.16	9.03	9.20	9.26
AIRS	8192	1501	2	7.94	8.47	8.98	9.44	9.68	9.88
			1	7.64	8.17	8.68	9.15	9.38	9.59
			0.5	7.35	7.88	8.40	8.85	9.08	9.30

these blocks may reduce the total number of FLOPs almost one order of magnitude, as shown in the experiments with blocks of 256 pixels.

On the other hand, in order to evaluate the computational complexity of the HyperLCA transform against some of the less computational demanding spectral transforms used in hyperspectral lossy compression, the POT [31] and the Wavelet CDF 9/7 [31] transforms have been used. Table 4.7 shows the approximated number of FLOPs required by these transforms for spectrally decorrelating a hyperspectral image according to the size of the image. On the other hand, table 4.8 shows the approximated number of FLOPs required by these transforms for spectrally decorrelating the previously used hyperspectral images. Comparing the results shown in Tables 4.8 and 4.5 it is observed that the number of FLOPs required by the HyperLCA is higher, but still comparable, with the number of FLOPs required by the POT and Wavelet CDF 9/7 transforms.

TABLE 4.7: Approximated number of FLOPs required for performing the spectral decorrelation of a hyperspectral image using the POT and the Wavelet CDF 9/7 transforms.

Process	POT transform	Wavelet CDF 9/7 transform
Transform	$12 \cdot N_p \cdot N_b$	$14 \cdot N_p \cdot N_b$
Inverse Transform	$7 \cdot N_p \cdot N_b$	$14 \cdot N_p \cdot N_b$

Nevertheless, as described in Section 4.3.4, the spectral decorrelation is just one of the stages of the entire compression process for hyperspectral images. Two more compression stages are typically applied after spectrally decorrelating the data, namely spatial decorrelation and compression, and entropy coding. At this point it is important to mention that, while the POT and the Wavelet CDF 9/7 transforms perform just the spectral decorrelation of the hyperspectral images, the HyperLCA and the PCA transforms also reduce amount of data. These means, that after decorrelating the hyperspectral image using the POT or the Wavelet CDF 9/7 transforms, the total amount of data to be coded is still the same. On the other hand, after applying the PCA or the HyperLCA transforms the total amount of data to be coded has been reduced, according to the compression ratio achieved in this compression stage. According to this fact, the number of FLOPs require in the subsequent stages of the compression process is probably less when using the HyperLCA or the PCA transforms than when using the POT or the Wavelet CDF 9/7 transforms.

Finally, as explained in Section 4.3.4, we propose the application of just the entropy coding stage after the HyperLCA transform, reducing even more the computational burden of the entire compression process by removing one of its major compression stages (the spatial decorrelation and compression). This methodology has been evaluated in Section 4.4.3, demonstrating the achievement of competitive results for hyperspectral imaging compression. Furthermore, the entropy codding stage applied in this work in conjunction with the HyperLCA transform, carried on by the CCSDS121 compressor, requires a very low computational burden.

All these facts demonstrate the efficiency of the proposed HyperLCA transform for spectrally decorrelating and reducing hyperspectral images, as well as the efficiency of the proposed HyperLCA+CCSDS121 hyperspectral images compressor in relation with other state-of-the art approaches, making them very suitable options for on-board applications or applications under real-time constraints.

TABLE 4.8: Number of FLOPs (in logarithmic scale) performed by the POT and Wavelet CDF 9/7 transforms for spectrally decorrelatinghyperspectral images, according to the size of the image.

Image characteristics			FLOPs			
Sensor	N_p	N_b	POT	Wavelet CDF 9/7	Inverse POT	Inverse Wavelet CDF 9/7
AVIRIS	262144	224	8.85	8.91	8.61	8.91
Hyperion	131072	242	8.58	8.65	8.35	8.65
AIRS	8192	1501	8.17	8.24	7.93	8.24

4.7 Conclusions

In this Chapter a new algorithm, named *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA), has been described. This algorithm performs the spectral decorrelation and reduction of the hyperspectral or ultraspectral data by projecting it onto a set of orthogonal vectors, similarly to other state-of-the-art approaches that employ the KLT transform. However, the extraction of the orthogonal set of vectors is performed in the HyperLCA by using a modified version of the Gram-Schmidt orthogonalization process. This provides several advantages. On one side, the proposed methodology has a reduced computational burden in relation with KLT approaches, and can be efficiently and independently applied to small blocks of the data set with accurate compression results, what makes it more suitable for applications under real-time constraints, also providing error resilience. On the other side, the HyperLCA transform is specially designed for preserving the anomalous pixels, or the most different pixels in the data set, through the compression-decompression process, what is crucial for many hyperspectral applications such as anomaly detection, tracking or classification.

An extensive amount of experiments have been performed in order to evaluate the goodness of the proposed algorithm. The results obtained using the HyperLCA transform for spectrally decorrelating and reducing the data set have been compared with those provided by the PCA transform, which is the best state-of-the-art approach for this task. As a further step, the HyperLCA transform has been used in conjunction with the CCSDS121 entropy coder for performing the entire compression-decompression process. The results obtained have been compared with the results provided by the PCA transform in conjunction with the CCSDS122 compressor. Furthermore, the HyperLCA algorithm has also been tested for spectrally decorrelating and reducing ultraspectral images. Finally, the effect of the compression-decompression process within the HyperLCA transform (in conjunction with the CCSDS121 coder) on anomaly detection application has also been tested. On the other hand, the computational burden of the HyperLCA algorithm has

also been evaluated and compared with the computational burden of the PCA transform, for different images, measuring the FLOPs required for spectrally reducing the data sets. Furthermore, the number of FLOPs required by the HyperLCA transform has also been compared with the number of FLOPs required by the POT and the Wavelet CDF 9/7 transforms for spectrally decorrelating the same hyperspectral images. According to the results obtained in the experiments, it is concluded that the HyperLCA transform efficiently provides high compression performance. It is specially suitable for on-board compression and applications under real-time constraints since it is able to preserve the quality of the compressed pixels for hyperspectral image applications such as anomaly detection or classification.

Chapter 5

Hyperspectral imaging acceleration through the utilization of high-performance computing platforms

A set of hardware-friendly algorithms for hyperspectral imaging applications has been described in the previous Chapters of this Thesis. This Chapter proves the suitability of those algorithms for being implemented in parallel computing platforms for applications under real-time constraints. Since all the algorithms developed in this Thesis use similar operations, it is assumed that the results and conclusions obtained for the implementations of the FUN algorithm described in this Chapter could be transposed to the rest of the developed algorithms within this Thesis.

This Chapter also evaluates the efficiency of two of the most widely used parallel computing platforms for hyperspectral imaging applications, namely *Graphic Processing Units* (GPUs) and *Field Programmable Gate Arrays* (FPGAs), according to the characteristics of the targeted images and applications, through the implementation of the FUN algorithm. *Open Computing Language* (OpenCL) has been used for such purpose, since it allows to develop implementations for different computing devices and architectures using one single programming interface. Nevertheless, NVIDIA *Computer Unified Device Architecture* (CUDA) has also been used for implementing the FUN algorithm onto the same

GPUs, for evaluating the efficiency of OpenCL against more device-specific programming interfaces such as CUDA for NVIDIA GPUs.

5.1 Outline

Hyperspectral imaging algorithms for applications under real-time constraints are typically implemented onto high-performance computing architectures in which the operations involved are executed in parallel devices, such as GPUs and/or FPGAs. However, the achievable speed up mainly depends on how parallelizable the algorithms are, as well as the complexity of the operations performed by the algorithms. This means that when there exist many data dependences in the algorithms which force sequential executions, the benefits of the high-performance parallel architectures can hardly be exploited. The algorithms developed in this Thesis have been thought from their conception for being efficiently implemented into parallel devices, and hence, the data and operations dependences, as well as complex matrix operations, have been avoided, what eases their parallel implementations for real-time hyperspectral imaging applications. In this Chapter, a modified version of the FUN algorithm, which has been named *parallel Fast algorithm for linearly UNmixing hyperspectral images* (pFUN), is implemented onto different computing platforms, proving its viability for applications under tight latency constraints. This modified version of the FUN algorithm provides exactly the same results than the original FUN algorithm but has some computational advantages with respect to the original FUN, specially for being implemented into parallel devices such as GPUs and FPGAs. Since all the algorithms developed in this Thesis use similar operations, it is assumed that the results and conclusions obtained for the implementations of the pFUN algorithm could be extrapolated to the rest of the developed algorithms within this Thesis.

This Chapter also presents a description of the main parallel computing platforms, FPGAs and GPUs, typically used for implementing parallel algorithms for hyperspectral imaging analysis, for applications under real-time constraints. The intrinsic parallelism which can be exploited in these devices varies according to the different architectures. This results in the necessity of developing the algorithm for the particular targeted device, using the programing platforms and/or hardware description languages available for such device. In this Chapter, we explore the efficiency of OpenCL for developing hyperspectral imaging applications for multiple devices with different architectures and vendors. For such purpose, different comparisons are done in this Chapter using the pFUN algorithm. First of all, the pFUN algorithm is implemented in two NVIDIA GPUs using NVIDIA CUDA, for performing the entire unmixing process. The results obtained are compared with the fastest implementation of the entire unmixing process in the state of the art, which allows to use this CUDA implementation as a reference for evaluating the efficiency

of the rest of the implementations done in this Chapter. Secondly, the pFUN algorithm is implemented in the same GPUs using OpenCL, following the same structure than in the aforementioned CUDA implementation. This allows to compare the efficiency of OpenCL for performing a NVIDIA GPU implementation against using its specific programing platform, CUDA. Finally, the pFUN algorithm is implemented in an Altera FPGA, using OpenCL.

5.2 State-of-the-art in hyperspectral imaging acceleration through the utilization of high-performance computing platforms

Hyperspectral imaging applications under real-time constraints require to rapidly process a huge amount of data. In these applications, the algorithms required for processing the data are typically implemented onto high-performance computing architectures in which the operations involved are executed in parallel devices, such as GPUs and/or FPGAs, for speeding up the process. However, the achievable speed up hardly depends on the characteristics of the algorithms to be implemented. This means that when there exist many dependences in the algorithms which force sequential executions, the benefits of the high-performance parallel architectures can hardly be exploited.

Unfortunately, the majority the algorithms for processing hyperspectral images so far published have been typically developed with the purpose of producing the best hyperspectral analysis results, with independence of their mathematical complexity, and without having in consideration how these algorithms could be implemented into parallel computing platforms for providing results in real time. Hence, when these algorithms are to be implemented into parallel devices for providing faster results, there is little room for speeding them up, due to their mathematical complexity as well as their so many data dependencies. Even so, many works have been uncovered during the last years for implementing different algorithms for processing hyperspectral images into parallel devices, with the purpose of speeding up the hyperspectral analysis process.

Due to the complexity of the existing hyperspectral image processing algorithms many efforts have also been done for trying to modify or better adapt some of these algorithms, most of them for unmixing or compressing hyperspectral images, for being implemented

in an easier and more efficient way [21, 29–31]. However, these approaches for speeding up the hyperspectral analysis are always limited by the inherent complexity of the algorithms and typically results in decreasing their accuracy.

According to the objectives of this Thesis, three different algorithms, thought from their conception for being efficiently implemented into parallel devices, have been developed, named FUN, CoEf-MHI and HyperLCA. These algorithms perform the hyperspectral linear unmixing process, the fusion of multispectral and hyperspectral images, and the hyperspectral and ultraspectral images compression, respectively. As a further step, this Chapter aims to prove the suitability of these algorithms for hyperspectral imaging applications under real-time constraints. For such purpose, a modified version of the FUN algorithm, which has been named *parallel Fast algorithm for linearly UNmixing hyperspectral images* (pFUN), is implemented onto different computing platforms, proving its viability for applications under tight latency constraints. This modified version of the FUN algorithm provides exactly the same results than the original FUN algorithm but has some computational advantages with respect to the original FUN, specially for being implemented into parallel devices such as GPUs and FPGAs. Since all the algorithms developed in this Thesis use similar operations, it is assumed that the results and conclusions obtained for the implementations of the pFUN algorithm could be transposed to the rest of the developed algorithms within this Thesis.

In order to evaluate the efficiency of the developed implementations of the pFUN algorithm against state-of-the-art implementations of the hyperspectral unmixing process, the most representative state-of-the-art implementations of unmixing algorithms have been compiled and deeply analyzed. They have been summarized in Tables 5.1 and 5.2, where Table 5.1 contains the implementations done into FPGA devices while Table 5.2 contains the implementations done into GPUs. Columns labeled as *Unmixing step* show which unmixing stages are uncovered in each implementation. According to the works described in these tables, there are many FPGA implementations of algorithms for different stages of the unmixing process that provide fast results, however, none of them accomplish the implementation of the entire unmixing process. On the other hand, works [25], [112], [113] and [114] implement the entire unmixing chain into GPUs, being [112] and [114] the works providing the fastest unmixing results (to the best of the author's knowledge) with independence of the size and content of the hyperspectral image to be unmixed and the internal characteristics of the targeted NVIDIA GPU.

Table 5.1: FPGA implementations of unmixing algorithms

Vendor	Methodology	Algorithm	Unmixing step			Publication
			Number of endmembers	Endmembers extraction	Abundances calculation	
Xilinx	Verilog HDL	MVCA		x		[115], [116]
	Matlab - Catapult C	N-Finder		x		[117]
	VHDL	VD	x			[118]
		HySIME	x			[119], [120]
		VCA		x		[121], [122], [123]
		MVCA		x		[124]
		PPI		x		[125], [126]
		N-Finder		x		[126], [127], [128]
		RT-MSVA		x		[129], [130]
		ISRA			x	[131], [132], [133]
		ICA		x	x	[134]

Table 5.2: GPU implementations of unmixing algorithms

Vendor	Methodology	Algorithm	Unmixing step				References
			Other	Number of endmembers	Endmembers extraction	Abundances calculation	
NVIDIA	OpenCL	SUMSAL			x	x	[135]
		VCA + SUMSAL			x	x	[136]
		MVCA			x		[22]
	CUDA	VD + OSP-GS + UCLS		x	x	x	[25]
		HySIME + PCA + N-Finder + LSU	x	x	x	x	[112]
		FCLSU / PCLSU / LSU				x	[137]
		VD / HySIME		x			[138]
		MVSA			x		[139]
		OSP + LSU			x	x	[140]
		VD / HySIME + N-Finder / OSP-GS		x	x		[141]
		PPI			x		[142]
		VCA			x		[143]
		FCLSU				x	[144]
		CSMNF			x	x	[145]
		HySIME + VCA + FCLSU		x	x	x	[113]
		AMEE + LSU			x	x	[146]
		IEA			x		[147]
		N-Finder			x		[148]
		SPP	x				[149]
		ICE			x	x	[150]
		VCA			x		[151]
		SSPP	x				[152]
		SPP	x				[153]
		VCA			x		[154]
		MVC-NMF			x		[155]
		OSP-GS + UCLS			x	x	[156]
		PPI / KPPI / AMEE			x		[157]
		ACOEE			x		[158]
		VD + SPP + VCA + LSU	x	x	x	x	[114]

Due to this reason, the efficiency of the developed implementations of the pFUN algorithm, which perform the entire unmixing process, has been evaluated against these two GPU-based unmixing chains. At this point it is important to highlight that hyperspectral imaging compression has also concentrated many of the efforts in order to accelerate the whole process by mean of the utilization of high performance parallel computing platforms, such as FPGAs or GPUs [159–176]. Due to the reason that the pFUN algorithm has been conceived for unmixing purposes, Table 5.1 and 5.2 are dedicated to the most relevant FPGA-based and GPU-based works for unmixing a given hyperspectral image.

The two chosen GPU-based unmixing chains use a set of well known algorithms for performing the entire linear unmixing process. The first unmixing chain, [112], implements the *Hyperspectral Signal Identification by Minimum Error* (HySIME) [54] algorithm for estimating the number of endmembers in the image under inspection, the *Principal Component Analysis* (PCA) [73] algorithm for spectrally reducing it, the N-Finder algorithm for extracting its endmembers and the *Linear Spectral Unmixing* (LSU) algorithm for calculating their abundances in the image. Similarly, the second unmixing chain selected, [114], implements the *Virtual Dimensionality* (VD) [53] algorithm for estimating the number of endmembers in the scene, the *Spatial Preprocessing* (SPP) [177] algorithm for spatially and spectrally preprocessing the image under analysis, and the *Vertex Component Analysis* (VCA) [56] and LSU algorithms for extracting its endmembers and calculating their abundances. Both chains have been implemented using CUDA and have been heavily optimized by their authors for being executed onto NVIDIA GPUS, allowing for a more efficient exploitation of hyperspectral data in applications requiring a fast response.

On the other hand, many decisions are to be taken when a set of algorithms for hyperspectral imaging applications are to be implemented into parallel computing platforms. First of all, different parallel devices, such as GPUs or FPGAs, from different vendors, such as NVIDIA, AMD, Altera or Xilinx, can be chosen. Secondly, according to the targeted computing platform chosen, different developing flows can be followed, which may vary in their level of abstraction as well as in their portability across different devices, architectures and vendors. Moreover, the algorithms may be adapted for exploiting the inherent parallelism of the chosen device in a more efficient way. Despite all the different implementations developed during the last years for hyperspectral imaging applications, little efforts have been made [178–180] in order to compare or evaluate which could be an efficient approach when facing the development of new hyperspectral imaging applications under real-time constraints. Moreover, due to the relevance of this topic, a more general approach becomes mandatory in order to extract a set of conclusions about the

best computing platforms for each hyperspectral imaging application under tight latency constraints.

In particular, we have found three main shortcomings within the publications cited in Tables 5.1 and 5.2. First of all, each of the publications shown in Tables 5.1 and 5.2 target one specific architecture, which may be a FPGA or a GPU, but give no justification of why it is a better solution using the targeted architecture for unmixing hyperspectral images under real-time constraints. Secondly, all the cited works use Xilinx FPGAs or NVIDIA GPUs, being VHDL and CUDA the most widely used methodologies for developing the implementations of the unmixing algorithms. Moreover, each implementation is done using just one single methodology. Due to this reason, there is no comparison between different possibles methodologies when facing the development of unmixing applications using a particular FPGA or GPU for applications under tight latency constraints. Finally, almost all the published implementations are tested using the same two hyperspectral images, collected by the well known AVIRIS sensor over Cuprite and World Trade Center. Hence, the behavior of the developed applications under different unmixing situations, such as images with different sizes and number of endmembers, is not really evaluated.

Due to these reasons, this Chapter also aims to provide a set of comparisons between different approaches for developing implementations of algorithms for hyperspectral imaging analysis under tight latency constraints, which allows to have a better idea of which approach could be more suitable according to the characteristics of the targeted application.

Different implementations of the pFUN algorithm have been carried out in this Chapter for this purpose using different parallel devices and architectures. OpenCL has been selected as developing platform for achieving the implementation of the pFUN algorithm in different devices, since it allows to develop implementations for different computing devices and architectures using one single programming interface. In particular, the pFUN algorithm has been implemented, using OpenCL, in a Geforce GTX 950 and a Tesla K40c NVIDIA GPUs and a Bitware Stratix V Altera FPGA. Furthermore, the pFUN algorithm has also been implemented into the same GPUs, but using its native programming language, CUDA, for evaluating the performance of OpenCL in relation with a more specific programming platform for the targeted GPUs.

5.3 Optimizations applied to the FUN algorithm for a more efficient hardware implementation. pFUN algorithm.

The FUN algorithm has been described in Chapter 2 in such a way that it directly applies the modified Gram-Schmidt orthogonalization for unmixing hyperspectral images. However, the description of the FUN algorithm can be slightly modified in order to obtain a mathematically equivalent version, which has been named *parallel FUN* (pFUN), that has some computational advantages, specially for being implemented into parallel devices such as GPUs and FPGAs. The pFUN algorithm provides exactly the same results than the FUN algorithm. The main purpose of the modifications applied to the FUN algorithm for obtaining the pFUN algorithm is to avoid the calculation of the square roots present in the FUN algorithm, which is mainly needed in the FUN algorithm for calculating the norm of the vectors. These modifications of the FUN algorithm provide some extra advantages, since some operations can be removed, specially in the abundances calculation process of the FUN algorithm, as it is described in Sections 5.3.1 and 5.3.2.

This Section is devoted to show the differences between the pFUN and the original FUN algorithm. The former is presented in Algorithms 5.2 and 5.4, whereas the latter is summarized in Algorithms 5.1 and 5.3 in order to highlight the modifications which have been introduced in the pFUN algorithm. These modifications reduce the number of operations to be performed while guaranteeing the same unmixing results.

5.3.1 Modifications in the endmember extraction process

The pseudocodes of the FUN and the pFUN algorithms for the endmember extraction stage are shown in Algorithms 5.1 and 5.2, respectively. The letter R has been used for referring to the lines of the pseudocode of the FUN algorithm, while letter P has been used for referring to the lines of the pFUN algorithm presented in this Section.

Algorithm 5.1: Process to extract the endmembers in the FUN algorithm

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, $\alpha = 1$

R.1 $E = []$; {Endmembers matrix }
R.2 $U = []$; {Endmembers' Gram Schmidt orthonormalized vectors}
R.3 $X = [x_1, x_2, ..., x_{N_p}] = M$ {Auxiliary copy of the hyperspectral image}
R.4 $e_1 = x_j$ where x_j is the pixel of the hyperspectral image selected as first endmember according to the initialization criteria; {Select the first endmember}
R.5 $u_1 = e_1/\|e_1\|$;
R.6 $E = [e_1]$;
R.7 $U = [u_1]$;
R.8 $p = 1$; {Number of endmembers found}
R.9 exit = 0;
R.10 **while** *exit* $= 0$ **do**
R.11 **for** $j = 1$ **to** N_p **do**
R.12 $x_j = x_j - (x_j \circ u_p) \cdot u_p$;
R.13 $s_j = \|x_j\|$;
R.14 **end**
R.15 $j_{\max} = \arg\max(s_j)$;
R.16 **if** $\max(s) \leq (\alpha/100) \cdot \|r_{j_{\max}}\|$ **then**
R.17 exit = 1;
R.18 **else**
R.19 $p = p + 1$;
R.20 $u_p = x_{j_{\max}}/\|x_{j_{\max}}\|$;
R.21 $e_p = r_{j_{\max}}$;
R.22 $U = [U, u_p]$;
R.23 **end**
R.24 **end**

Outputs: p, {number of endmembers}
$E = [e_1, e_2, ..., e_p]$, {Endmembers}

Algorithm 5.2: Process to extract the endmembers in the pFUN algorithm

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, $\alpha = 1$

P.1 $E = []$; {Endmembers matrix }
P.2 $Q = []$; {Endmembers' Gram Schmidt orthogonalization}
P.3 $U^* = []$; {Endmembers' Gram Schmidt orthogonalization divided by the bright of the orthogonalized vector}
P.4 $X = [x_1, x_2, ..., x_{N_p}] = M$ {Auxiliary copy of the hyperspectral image}
P.5 $e_1 = x_j$ where x_j is the pixel of the hyperspectral image selected as first endmember according to the initialization criteria; {Select the first endmember}
P.6 $q_1 = e_1$;
P.7 $u_1^* = e_1/(e_1 \circ e_1)$;
P.8 $E = [e_1]$;
P.9 $Q = [q_1]$;
P.10 $U^* = [u_1]$;
P.11 $p = 1$; {Number of endmembers found}
P.12 exit = 0;
P.13 **while** *exit* $= 0$ **do**
P.14 **for** $j = 1$ **to** N_p **do**
P.15 $x_j = x_j - (x_j \circ q_p) \cdot u_p^*$;
P.16 $s_j^2 = x_j \circ x_j$;
P.17 **end**
P.18 $j_{\max} = \arg\max(s_j^2)$;
P.19 **if** $\max(s^2) \cdot 100^2 \leq \alpha^2 \cdot (r_{j_{\max}} \circ r_{j_{\max}})$ **then**
P.20 exit = 1;
P.21 **else**
P.22 $p = p + 1$;
P.23 $q_p = x_{j_{\max}}$;
P.24 $u_p^* = x_{j_{\max}}/(x_j \circ x_j)$;
P.25 $e_p = r_{j_{\max}}$;
P.26 $Q = [Q, q_p]$;
P.27 $U^* = [U^*, u_p^*]$;
P.28 **end**
P.29 **end**

Outputs: p, {number of endmembers}
$E = [e_1, e_2, ..., e_p]$, {Endmembers}

The are three main differences between the pFUN and the FUN algorithms with respect to the endmembers extraction. The first one is that the pFUN algorithm uses two sets of projection vectors, labelled as Q and U^* in the Algorithm 5.2 (lines $P.2$ and $P.3$), instead of the projection vectors U in the Algorithm 5.1 (line $R.2$). While the U vectors are orthonormalized vectors, the Q vectors are just orthogonal vectors, and the U^* vectors correspond with the Q vectors divided by their own bright. By doing this, lines $R.12$ and $R.20$ are substituted by lines $P.15$, $P.23$ and $P.24$, avoiding the calculation of the norm

of the vectors, which involves resolving a square root, and simplifying the operations of the algorithm. This affects to the implementation of one of the main mathematical expressions of the algorithm, which has been modified as shown in Equation 5.1.

$$\begin{aligned}(x_j \circ u_p) \cdot u_p &= (x_j \circ \frac{q_p}{\|q_p\|}) \cdot \frac{q_p}{\|q_p\|} \\ &= (x_j \circ q_p) \cdot \frac{q_p}{q_p \circ q_p} \\ &= (x_j \circ q_p) \cdot {u_p}^*\end{aligned} \tag{5.1}$$

The second difference between the FUN and the pFUN algorithms is in lines $R.13$, $R.15$ and $R.16$, and $P.16$, $P.18$ and $P.19$ of their pseudocodes. The FUN algorithm selects each endmember as the pixel which more information that cannot be represented by the already extracted endmembers. This is done by comparing the norm of the information remaining in each pixel of the image after removing the information represented by the last endmember extracted (lines $R.13$ and $R.15$). In the proposed version, this measurement is done using the bright of the pixels instead of their norm, since it is proportional (lines $P.16$ and $P.18$). This allows to avoid the square root calculation. This modification has one extra benefit, since the stopping condition of the algorithm is also simplified (lines $R.16$ and $P.19$). Equation 5.2 shows the proposed modification of the stopping condition of the algorithm.

$$\begin{aligned}\max(s) \leq \frac{\alpha}{100} \cdot \|r_{j_{\max}}\| \Leftrightarrow \\ \max(s^2) \leq \frac{\alpha^2}{100^2} \cdot (r_{j_{\max}} \circ r_{j_{\max}})\end{aligned} \tag{5.2}$$

Finally, the stopping condition of the pFUN algorithm has been represented as shown in line $P.19$ to avoid division by 100^2. This modification is shown in Equation 5.3.

$$\begin{aligned}\max(s^2) \leq \frac{\alpha^2}{100^2} \cdot (r_{j_{\max}} \circ r_{j_{\max}}) \Leftrightarrow \\ \max(s^2) \cdot 100^2 \leq \alpha^2 \cdot (r_{j_{\max}} \circ r_{j_{\max}})\end{aligned} \tag{5.3}$$

5.3.2 Modifications in the abundance calculation process

The pseudocodes of the FUN and pFUN algorithms for the abundance calculation are shown in Algorithms 5.3 and 5.4, respectively. As before, the letter R has been used for referring to the lines of the FUN algorithm, while letter P has been used for referring to the lines of the pFUN algorithm described in this Section.

Algorithm 5.3: Process to obtain the abundances in the FUN algorithm

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, p, $E = [e_1, e_2, ..., e_p]$

R.1 $E = [E, E]$;
R.2 $Q = [], U = [], U^* = [], U^{**} = [], \text{NORM} = []$;
R.3 **for** $k = 2$ **to** $p + 1$ **do**
R.4 $\quad u_1 = \|e_k\|$;
R.5 $\quad$ **for** $j = 2$ **to** p **do**
R.6 $\quad\quad x = e_{k+j-1}$;
R.7 $\quad\quad$ **for** $i = 1$ **to** $j - 1$ **do**
R.8 $\quad\quad\quad x = x - (x \circ u_i) \cdot u_i$;
R.9 $\quad\quad$ **end**
R.10 $\quad\quad q_j = x$;
R.11 $\quad\quad u_j = q_j / \|q_j\|$;
R.12 $\quad\quad Q = [Q, q_j]$;
R.13 $\quad\quad U = [U, u_j]$;
R.14 $\quad$ **end**
R.15 $\quad U^* = [U^*, u_p]$;
R.16 $\quad \text{NORM} = [\text{NORM}, \|q_p\|]$;
R.17 $\quad Q = []$;
R.18 $\quad U = []$;
R.19 $\quad$ **for** $i = 1$ **to** p **do**
R.20 $\quad\quad u_i^{**} = u_i^* / \text{norm}_i]$;
R.21 $\quad\quad U^{**} = [U^{**}, u_i^*]$;
R.22 $\quad$ **end**
R.23 **end**
R.24 $A = (U^{**})^T \cdot \text{M}$; {Abundances}

Outputs: Abundances

Algorithm 5.4: Process to obtain the abundances in the pFUN algorithm

Inputs: $M = [r_1, r_2, ..., r_{N_p}]$, p, $E = [e_1, e_2, ..., e_p]$

P.1 $E = [E, E]$;
P.2 $Q = [], U^{**} = [], U^* = []$;
P.3 **for** $k = 2$ **to** $p + 1$ **do**
P.4 $\quad Q = e_k$;
P.5 $\quad u_1{}^{**} = e_k / (e_k \circ e_k)$;
P.6 $\quad$ **for** $j = 2$ **to** p **do**
P.7 $\quad\quad x = e_{k+j-1}$;
P.8 $\quad\quad$ **for** $i = 1$ **to** $j - 1$ **do**
P.9 $\quad\quad\quad x = x - (x \circ q_i) \cdot u_i^{**}$;
P.10 $\quad\quad$ **end**
P.11 $\quad\quad q_j = x$;
P.12 $\quad\quad u_j^{**} = x / (x \circ x)$;
P.13 $\quad\quad Q = [Q, q_j]$;
P.14 $\quad\quad U^{**} = [U^{**}, u_j^{**}]$;
P.15 $\quad$ **end**
P.16 $\quad U^* = [U^*, u_p^{**}]$;
P.17 $\quad Q = []$;
P.18 $\quad U^{**} = []$;
P.19 **end**
P.20 $A = (U^*)^T \cdot \text{M}$; {Abundances}

Outputs: Abundances

Similarly to the modifications proposed for the endmembers extraction part of the FUN algorithm, the modifications performed in the abundances calculation stage are done to avoid the calculation of the norms of orthogonal vectors, which involves an square root calculation. This implies the substitution of the orthonormalized set of vectors, U, in the Algorithm 5.3, by the Q and U^{**} vectors in the Algorithm 5.4. Equation 5.1 shows the mathematical correctness of this modification, which affects the lines $R.4$ to $R.18$ and $P.4$ to $P.14$, of the Algorithms 5.3 and 5.4, respectively. By doing this, one extra benefit is obtained. The corresponding ${u_p}^{**}$ vectors can be directly stored in the U^* vectors (line $P.16$) significantly reducing the number of operations to be performed, since the

operations shown in the lines $R.19$ to $R.22$ of the Algorithm 5.3 are no longer necessary in the Algorithm 5.4 as demonstrated in Equation 5.4.

$$\frac{u_p}{\text{norm}_p} = \frac{u_p}{\|q_p\|} = \frac{1}{\|q_p\|} \cdot \frac{q_p}{\|q_p\|} = \frac{q_p}{q_p \circ q_p} = {u_p}^{**} \tag{5.4}$$

5.4 Hardware platforms

This work focuses on GPUs and FPGAs among all the different devices that can be programmed using OpenCL. In particular, the pFUN algorithm has been independently implemented into one Bitware Stratix V Altera FPGA board and two NVIDIA GPUs, the GeForce GTX 950 and the Tesla K40c in this work. Due to the huge differences between the architecture of the targeted devices, the pFUN algorithm has been structured in two different ways for taking advantage of each particular architecture.

For developing an efficient OpenCL implementation for any of the considered computing platforms, it is important to understand how to take advantage of their inherent parallelism in the most efficient way, and for that, it is critical to understand how the device works. Hence, before going into details on how the pFUN algorithm has been adapted to exploit the parallelism in the targeted devices, a brief explanation of how the FPGAs and GPUs process data is advisable. This explanation is not focused in the most technical details of the architectures, which may vary between devices and vendors, but in the general concepts of each architecture, which need to be taken into account as a starting point for developing an efficient implementation. For this purpose, vectors A, B and C, as well as the Equations 5.5 and 5.6 will be used for introducing these concepts in a simple way. Equations 5.5 and 5.6 define the inner product between the vectors A and B, and the product between vector C and the scalar value ip, respectively. Sections 5.4.1 and 5.4.2 use these equations for describing some general tips of how to take advantage of the inherit parallelism in FPGAs and GPUs architectures, respectively.

$$A = (a_1, a_2, a_3) \quad B = (b_1, b_2, b_3) \quad C = (c_1, c_2, c_3)$$

$$ip = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3 \tag{5.5}$$

$$D = (ip \cdot c_1, ip \cdot c_2, ip \cdot c_3) \tag{5.6}$$

5.4.1 FPGAs description

Field Programmable Gate Arrays (FPGAs) can be understood as a group of programmable logic blocks and a hierarchy of reconfigurable interconnects that allow the blocks to be 'wired together', like many logic gates that can be inter-wired in different configurations. These logic blocks can be configured to perform complex combinational functions, or merely simple logic gates. Therefore, let's consider these logic blocks as the smallest compute unit of the FPGA. Additionally, FPGAs may contain other different blocks such as *Digital Signal Processing* (DSP) blocks or memory blocks, as illustrated in Figure 5.1. By combining some of these blocks and assuming that all the data are available for the operations, the inner product between A and B, shown in Equation 5.5, could be done in a few clock cycles. Similarly, if A and B have more components, the inner product between these two vectors could probably be still done the same number of clock cycles. However, more FPGA resources would be needed. On the other hand, for obtaining vector D, as shown in Equation 5.6, extra clock cycles would be needed, since it is necessary to first compute the value of the inner product ip and then use this value for scaling the vector C. The value of D could also be computed as $D = ((a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3) \cdot c_1, (a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3) \cdot c_2, (a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3) \cdot c_3)$. This way, D can be processed without processing the inner product between A and B first, and hence, it could be done without needing extra clock cycles. However, the number of required FPGA resources drastically increase, since the inner product between A and B is actually being replicated three times. When programming a FPGA, the total number of resources available is limited, and it is extremely important to reach an optimal trade off between the number of resources and number of clock cycles needed for the different operations that are to be implemented.

This explanation shows operation level parallelism. However, data level parallelism can also be inferred in the FPGAs. Lets suppose that there is an image with n pixels, and that three groups of operations GA, GB and GC must be performed to each pixel of the image. Lets also assume that each pixel can be independently processed. The FPGA could be programmed so that some resources are used to perform operations GA for a single pixel, some resources are used for performing operations GB, and some resources are used to perform operations GC. This way, a pipeline structure could be achieved as

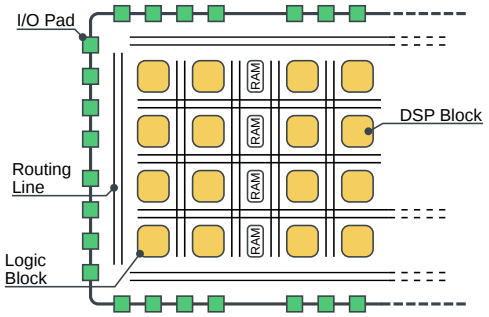


FIGURE 5.1: FPGA resources and interconnects.

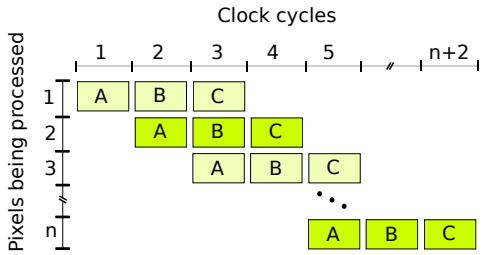


FIGURE 5.2: FPGA parallelism description. Pipeline

shown in Figure 5.2. Assuming that each group of operations GA, GB and GC can be performed in one single clock cycle for each pixel, the total number of cycles needed to process the entire image would be $n + 2$ cycles, since once that the process is finished for the first pixel, the process for one more pixel is finished every clock cycle, as shown in Figure 5.2. In OpenCL, all this set up could be understood as a compute unit. If there were enough available resources in the FPGA, this compute unit could be duplicated, and the entire process could be twice faster, as shown in Figure 5.3.

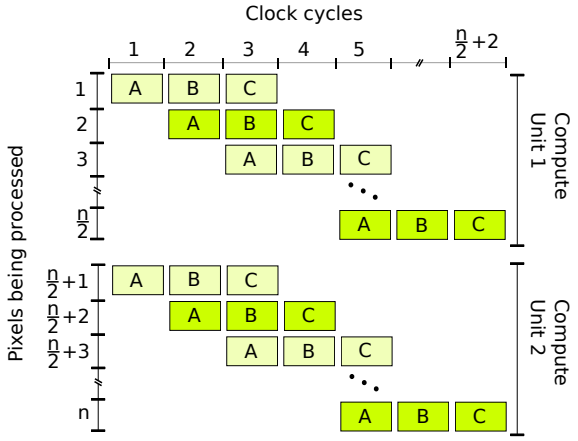


FIGURE 5.3: Division of the pipeline structure in compute units.

On the other hand, the efficiency of the FPGA implementation also depends on how the memory storage and accesses are organized. The FPGA device memory can be split in on

chip and off chip memory, referred as Local and Global memory in OpenCL, respectively. Reading and writing accesses to the local memory are faster, but this memory is limited. Hence, a good practice is to store in the local memory those things which are accessed more times. When managing big amounts of data, such as hyperspectral images, it is necessary to store them in the global memory. The total amount of data that can be read and written in a single clock cycle to the global memory is limited, and also is affected by the alignment of the accessed elements. Hence, a good practice for obtaining more efficient implementations is to reduce the number of accesses to the global memory. In the example shown in Figures 5.2 and 5.3, it would be more efficient reading each pixel from the global memory just once and performing all the required operations to that pixel (groups of operations GA, GB and GC), temporarily storing the pixel in the local memory if needed until all the operations involving that pixel finishes, instead of reading the pixel several times from the global memory.

Finally, it is important to have in mind that the whole structure of the code may affect the FPGA clock frequency. When performing very complex operations and/or using too many FPGA logic blocks, the interconnections between the logic blocks and/or memory become more complex, negatively affecting the FPGA clock frequency and reducing the overall performance.

5.4.2 GPUs description

Graphic Processing Units (GPUs) can be understood as an array of independent processors. Each of these processors corresponds with an independent thread of execution. These processors are organized in blocks (blocks of threads), in such a way that different levels of memory appear. These memory levels vary in capacity and bandwidth. The memory is typically divided in cache memory, shared memory, global memory and constant memory. The cache memory is local to each individual thread, and no thread can access to any other thread cache memory. Similarly, shared memory is the memory shared within the threads which conform a block, and the threads from one block can only access the shared memory of its own block. Finally, the global memory is accessible by all the GPU threads. Cache memory has a higher bandwidth than shared memory, but smaller capacity, and similarly, shared memory has higher bandwidth and smaller capacity than global memory. Constant memory is similar to the global memory in the sense that all the threads can access it. As the name may indicate, this memory is useful for storing data that will not change over the course of the execution. When more than one thread

accesses for reading the same constant memory location, it is treated as one single access, providing a speed up with respect to the same situation when using the global memory, by proportional reducing the required bandwidth, according to the number of threads accessing that particular address.

The parallelism which can be inferred in this architecture totally differs from the parallelism inferred in the FPGAs architecture. Each GPU thread can only execute one operation per cycle. However, many different threads can perform the exact same operation at the same time, with different data. Hence, the parallelism inferred in the GPU can be efficiently exploited when there is a situation in which a set of operations is to be independently performed to many different elements of a data set, since a different thread can be used to independently performed the required operations to the each element of the data set.

This can be illustrated using the examples shown in Section 5.4.1. In the example shown in Equation 5.6, three different threads could be used to compute, in one single step, vector D, as each of the threads would independently compute $ip \cdot c_1$, $ip \cdot c_2$ and $ip \cdot c_3$ respectively. Similarly, if vector D would have more components, the result could still be computed in one step by simply using more threads. However, unlike in the FPGA, the inner product between the vectors A and B, shown in Equation 5.5, would take more than one step in a GPU, because it is first necessary to compute the products of the different elements of the vectors, $a_1 \cdot b_1$, $a_2 \cdot b_2$ and $a_3 \cdot b_3$, which can take just one step if three threads are used, and then performing the sum of them, which would take two more steps, with independence of the number of threads used.

In the example illustrated in Section 5.4.1, in which three groups of operations GA, GB, and GC must be performed to each pixel of one image, the GPU could be programmed in such a way that one thread would be used for each single pixel. Each thread would serially perform the groups of operations GA, GB, and GC to its corresponding pixel, as shown in Figure 5.4. Assuming that these three groups of operations are totally pixel independent, that the data are always available, and that there are always threads enough in the GPU to independently process each pixel of the image, the total number of cycles needed to perform the groups of operations GA, GB, and GC to the entire image would be ideally the same that the number of cycles required to process one single pixel, as shown in Figure 5.4. In a more realistic scenario, the number of cycles required by the GPU to perform the groups of operations GA, GB, and GC will also depend on the number of

cycles required for reading and writing the data, which will depend on how the data is stored in the GPU memory and the number of required accesses.

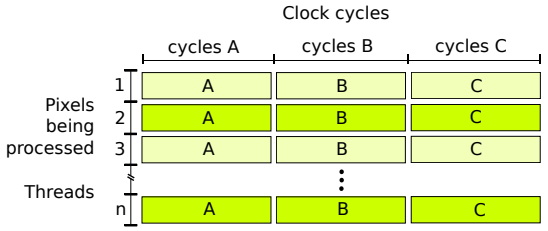


FIGURE 5.4: Thread level parallelism achieved in the GPUs.

5.4.3 FPGAs and GPUs comparison

According to the differences between the FPGAs and GPUs architectures, briefly described in Sections 5.4.1 and 5.4.2, two different parallel implementation models could be obtained, when trying to perform three groups of operations GA, GB, and GC to each pixel of an image, as shown in Figures 5.2 and 5.4.

Due to the fact that each thread of one GPU can only execute one operation at a time, each group of operations GA, GB, and GC for a single pixel will probably take more cycles in a GPU than in a FPGA.

However, comparing these two parallel implementation models, summarized in Figures 5.2 and 5.4, it can be easily understood that, if the number of pixels to be processed increases, the performance achieved in the GPU model may probably surpass the performance achieved in the FPGA model. This will also depend in the number of cycles required to perform the groups of operations GA, GB, and GC for a single pixel in the FPGA and in the GPU.

It is also important to mention that these models represent a very particular situation in which the processes to be performed to each individual pixel are totally pixel independent. In most of the algorithms for processing images this condition is rarely achieved, and the programmer needs to manage the dependences present in the algorithms for trying to exploit the benefits of each parallel computing platform in the most efficient way. Moreover, more restrictions may appear, such as limited device resources (memory, logic blocks in the FPGAs, blocks and threads in the GPUs, etc.) or limited bandwidth between the host and the devices. All these facts makes the implementation problem a complex one, which results partially depend on the programmer experience.

On the other hand, the described models, shown in 5.2 and 5.4, represent the general idea of how the algorithm to be implemented should be structured for taking advantage of the FPGAs and GPUs architectures, respectively, and are the models followed in the implementations of the pFUN algorithm, shown in this Thesis work.

5.5 Targeted devices

Four different devices have been used in this Chapter. Three of these devices are GPUs, the GeForce GTX 680, the GeForce GTX 950 and the Tesla K40c NVIDIA GPUs. The remaining device used in this Chapter is a Bitware Stratix V Altera FGPA. The GeForce GTX 680 and the Tesla K40c GPUs are based on the Kepler architecture released in 2012, while the GeForce GTX 950 is based on the newer and more efficient Maxwell architecture, released in 2014. The specific characteristics of the different GPUs used are summarized in Table 5.3. The Bitware Stratix V FPGA board is cerfitied as one of the Altera preferred boards for OpenCL. In particular, the Bitware board used carries a Stratix V GSD8 FPGA, which main features are shown in Table 5.4.

Table 5.3: GPUs features

	GeForce GTX 680	GeForce GTX 950	Tesla k40c
CUDA cores	1,536	768	2,880
Total amount of global memory (Mb)	2,048	2,048	11,520
GPU maximum clock rate (Mhz)	1,058	1188	745
Memory clock rate (MHz)	6,008	6,656	3,004

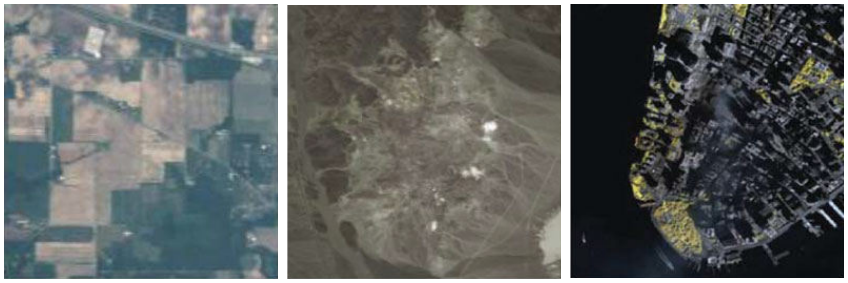
Table 5.4: FPGA features

SVGSD8	
Logic Elements (K)	695
ALMs	262,400
Registers (K)	1,050
14.1-Gbps transceivers	36 or 48
PCIe hard IP blocks	1, 2, or 4
Fractional PLLs	28
M20K Memory Blocks	2,567

5.6 Reference hyperspectral data

The implementations of the pFUN algorithm for different devices and architectures developed in this Thesis work have been evaluated and compared in this Chapter using different real and synthetic hyperspectral images.

Three different real images collected by the *Airbone Visible/Infrarred Imaging Spectrometer* (AVIRIS) [5] have been used. This sensor captures 224 spectral bands in the wavelength range of 0.4-2.5 μm. The first image used was collected in 1966 over Indian Pines in North-western Indiana. The selected scene is a total of 145 x 145 pixels with a spatial resolution of 20 m. Bands [1-4], [108-112], [154-167], and 224 have been removed due to water absorption, resulting in a total of 200 hyperspectral bands. The Indian Pines scene contains two-thirds agriculture, and one-third forest or other natural perennial vegetation. There are two major dual-lane highways, a rail line, as well as some low-density housing, other build structures, and smaller roads.



Indian Pines Cuprite World Trade Center

FIGURE 5.5: RGB representation of the different real hyperspectral images used for the simulations.

The second image used was taken over the region of *Cuprite*, Nevada, in the summer of 1997. The selected scene consists of a total of 250x191 pixels with a spatial resolution of 4 m. Several bands have been removed due to water absorption and low SNR resulting in a total of 188 spectral bands. The site is well understood mineralogically, and has several exposed minerals of interest including alunite, buddingtonite, calcite, kaolinite and muscovite.

The last hyperspectral image used was captured over the *World Trade Center* area in New York City on 16 September 2001, just 5 days after the terrorist attacks that collapsed the two main towers and other buildings in the WTC complex. The full data consists of a total amount of 614x512 pixels. Several bands have been removed due to water absorption and low SNR resulting in a total of 171 spectral bands.

This set of images represents a heterogeneous test bench, according to the different spatial resolution of the images as well as different content, which allows to compare the behaviour of the different implementations of the pFUN algorithm under different situations. Figure 5.5 shows a false color representation of the aforementioned data sets whereas Table 5.5 summarizes the features of the real hyperspectral images presented in this Section.

TABLE 5.5: Real hyperspectral images used for the experiments

	Pixel number	Bands
Indian Pines	21,025 (145 x 145)	200
Cuprite	47,750 (250 x 191)	188
World Trade Center	314,368 (614 x 512)	171

For performing a more controlled comparison of the performance of the different implementations, a set of synthetic hyperspectral images has been generated. These images represent an excellent test bench in order to study the efficiency of the different implementations because the number of pixels and endmembers can be fixed in advance. While the number of pixels of the image is directly proportional to the amount of data to be processed by the pFUN algorithm, the number of endmembers determines the number of iterations performed by this algorithm. These two characteristics determine the total computation to be done for unmixing these images using the pFUN algorithm.

The synthetic hyperspectral images used in this work were generated using one MATLAB function. This function allows creating hyperspectral images of a spatial size defined by the user, from p spectral signatures randomly selected from the *U.S. Geological Survey* (USGS) digital spectral library [79] that are then mixed according to abundance fractions, which are generated using a Gauss spheric distribution. In addition, a certain amount of Gaussian noise is added so that the generated images have a user-defined SNR. The SNR has been set to 80 dB for ensuring that the number of endmembers extracted by the pFUN algorithm will match the real number of endmembers present in the scene, making it easier to compare the time consumed by the different implementations during the unmixing process.

These images have been created using 10, 15 and 20 endmembers with 224 spectral bands each, all taken from the USGS library, and different spatial dimension, 100x100, 250x250, 400x400 and 500x500 pixels (10.000, 62.500, 160.000 and 250.000 pixels, respectively). This results in a total of 12 synthetic hyperspectral images.

5.7 NVIDIA CUDA GPU implementation of the pFUN algorithm

This section focuses on the parallel implementation of the pFUN algorithm onto NVIDIA GPUs through CUDA. For this purpose, important CUDA preliminary issues are presented before the CUDA implementation of the pFUN algorithm. After this, the CUDA implementation of the pFUN is presented divided into two parts: the first part focuses on the process of the endmembers extraction, whereas the second part is devoted to the calculation of the abundances.

5.7.1 Preliminary Issues

NVIDIA CUDA is a parallel computing platform and model that enables improvements in computing performance by harnessing the power of NVIDIA GPUs. The CUDA programming model differentiates the CPU and its memory as the host and the GPU and its memory as the device. Parallel portions of an application are executed in the device as kernels. GPU has several kinds of memory: global memory, shared memory, constant memory, and cache memory in some of the high-end models. Shared and global memory are faster than global memory. These memory types vary in capacity and bandwidth, and can be used in order to optimize applications.

The implementation of the pFUN algorithm described in Section 5.3 has been developed using CUDA onto NVIDIA propietary GPUs and the cuBLAS library. This library is an implementation of *Basic Linear Algebra Subprograms* (BLAS) on top of the NVIDIA CUDA runtime. cuBLAS provides functions whose execution take place on the device exploiting the maximum degree of parallelism. Only the cuBLAS functions which provide the best performance in terms of execution time have been exploited in this Thesis. This means that for those operations where the execution time could be improved, the cuBLAS functions calls have been replaced for custom kernels. Moreover, single-precision and double-precision have been checked. Since both provides good results for the pFUN implementation, single-precision floating point has been chosen due to performance issues.

The main limitation of GPUs is the need for intensive memory transfer, which is used for copying data from CPU memory to GPU global memory. This communication is carried out over the *Peripheral Component Interconnect* (PCI) express bus and it can

improve or make worse the overall application performance. Techniques to minimize the amount of transferred data between the CPU memory and GPU global memory have been used to obtain the best possible execution time. In this sense, coalesced memory transactions and shared memory with reduction building blocks have been used where possible. Furthermore, the use of pinned memory has also been exploited to avoid the pageable CPU (host) data allocations. Normally, the GPU cannot access data directly from pageable host memory, so when data transfer from pageable host memory to device memory are invoked, the CUDA driver must first allocate a temporary page-locked, or a pinned array on the host side, copy the host data to the pinned array, and then, transfer the data from the pinned array to device memory. The pinned memory is used as a staging area for transfers from host to device, and in this way, the cost of the transfer between pageable and pinned host arrays can be avoided by by directly allocating host arrays in pinned memory. However, the pinned memory should not be overallocated because it reduces the amount of physical memory available to the operating system and other programs and in consequence affects the overall system performance. In the pFUN implementation, only two variables use the pinned memory, the variable that controls the stop condition of the first part of the pFUN algorithm, and the variable that counts the number of endmembers.

5.7.2 CUDA implementation of the pFUN algorithm: extraction of the endmembers

This section focuses on the implementation of the Algorithm 5.2 using CUDA programming model and cuBLAS library. The correspondence of this algorithm and its CUDA implementation is depicted in Figure 5.7. The order of some lines of Algorithm 5.2 have been altered to simplify the explanation of this Figure. Moreover, Figure 5.7 shows the CUDA implementation divided into sections (represented as $\boxed{n}$). In this Figure, each section may be divided into some steps (represented as $\textcircled{n}$). The lines of the algorithm in Figure 5.7 have been labelled as $P.n$ (these lines are the same as the proposed pseudocode for the extraction of endmembers). In this notation n can be either the number of the line in the algorithm or the section or step depending on the shape where n is inside. Due to the fact that some parts of this algorithm are executed on the device and some others on the host, this difference has been highlighted in different colour in Figure 5.7. Moreover, from now on, the name of any variable allocated on the device will be preceded

by $d_$. According to the kernels of this section we can differentiate several types of them depending on the number of blocks and the threads per block used in its invocation:

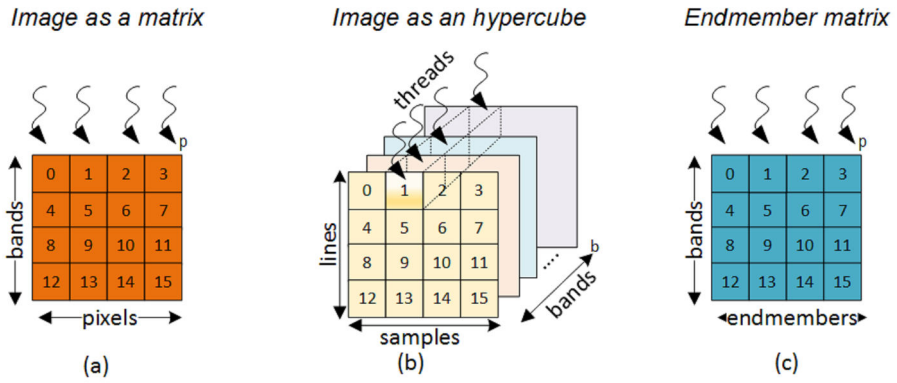


FIGURE 5.6: Several types of launching kernels.

(a) Kernels which are launched with so many blocks as the relation between the number of pixels and the GPU block size. This block size is a characteristic of the GPU architecture which sets up the number of threads per block to 1024. This invocation ensures that each thread works on a pixel (each pixel contains so many values as the number of bands of the hyperespectral image). The graphical representation of this kernel is reflected in Figure 5.6-(a). These kernels have been highlighted in green colour in Figure 5.7 and they are invocated as follows:

$$number\,blocks = \frac{number\,of\,pixels}{GPU\,block\,size} \tag{5.7}$$

$$kernel <<< number\,blocks, GPU\,block\,size >>> (...) \tag{5.8}$$

(b) Kernels which are invocated with an unique block with so many threads as the number of bands of the hyperspectral image. This invocation guarantees that evey thread works over a single band of a pixel. The graphical representation of this kernel is reflected in Figure 5.6-(b). These kernels have been highlighted in blue colour in Figure 5.7 and they are invocated as follows:

$$kernel <<< 1, number\,of\,bands >>> (..., ..., ...) \tag{5.9}$$

(c) Kernels which are launched with so many blocks as the relation between the number of endmembers and the GPU block size. This invocation ensures that each thread

works on an endmember (each endmember contains so many values as the number of bands of the hyperespectral image). The graphical representation of this kernel is reflected in Figure 5.6-(c). These kernels have been highlighted in red colour in Figure 5.7 and they are invocated as follows:

$$number\,blocks = \frac{number\,of\,endmembers}{GPU\,block\,size} \tag{5.10}$$

$$kernel <<< number\,blocks, GPU\,block\,size >>> (...) \tag{5.11}$$

(d) Kernels which are invocated with an unique block with so many threads as the block size of the GPU. This kernel is the second kernel used whenever the maximum value of an array needs to be calculated and it has been bolded and underlined in Figure 5.7. It is invocated as follows:

$$kernel <<< 1, GPU\,block\,size >>> (..., ..., ...) \tag{5.12}$$

The rest of this subsection is organized as follows. First, we focus on the data allocation and initialization on the device (sections 1 to 3 in Figure 5.7), named as *Data Initialization* later on. Then, the process of extraction of the first endmember of the hyperspectral image is presented (section 4 in Figure 5.7). Finally, the extraction of the rest of endmembers is addressed in Figure 5.7 from sections 5 to 8.

5.7.2.1 Data initialization

The process to extract the endmembers requires the transfer of the hyperspectral image from the CPU (host) memory to the global device memory. Because some operations are implemented by means of cuBLAS library which uses column-major storage, the hyperspectral image in column-major order has been transfered from the host (M) to the GPU memory (d_M) at the beginning of the process by means of the *cudaMemcpy* function specifying the *cudaMemcpyHostToDevice* value for its last parameter (see section 1 of Figure 5.7).

In order to reduce the data transfer from host to device, only the necessary data (d_E, d_q, d_u, d_X, $d_indexes$) will be allocated in the device memory (see section 2 of Figure

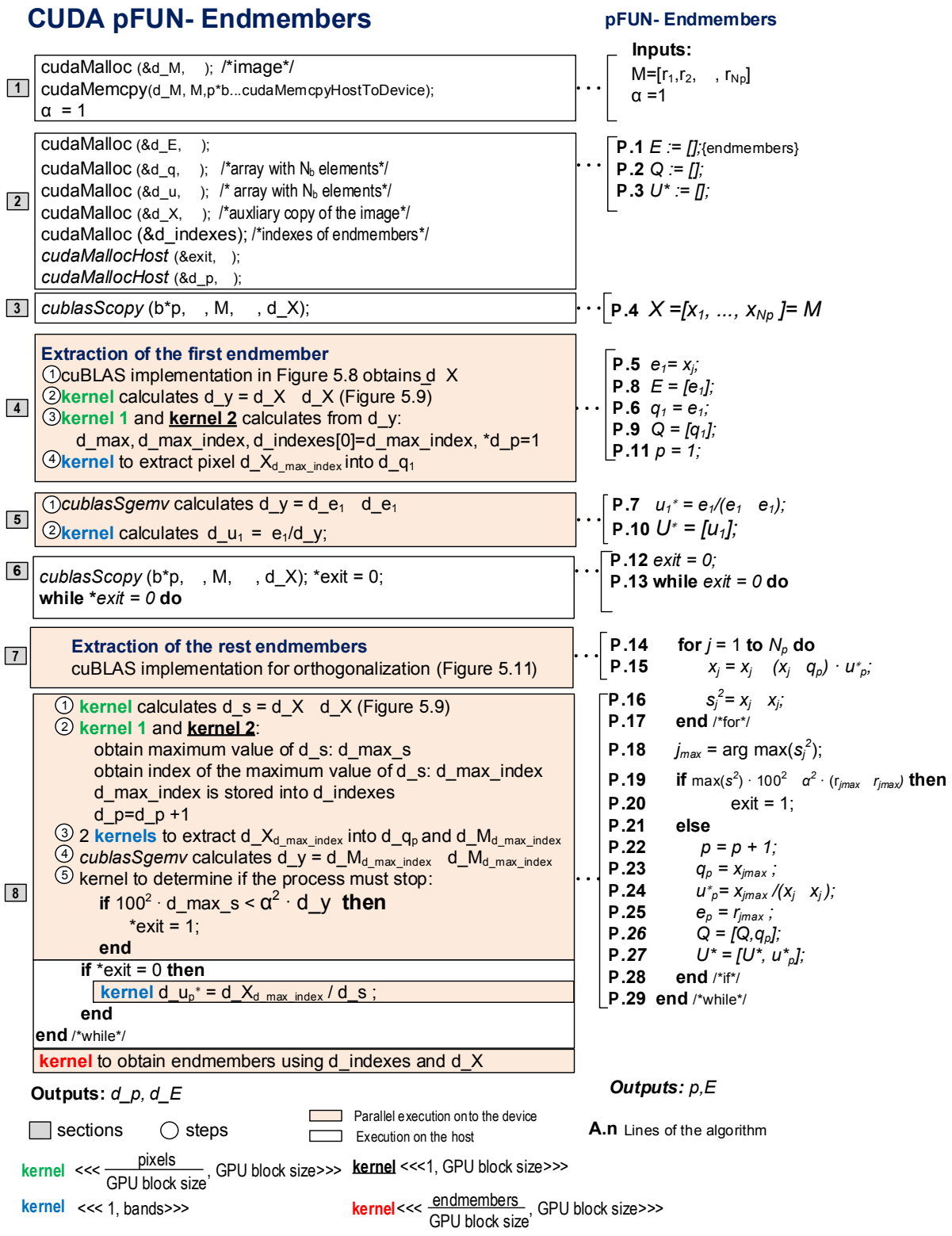


FIGURE 5.7: CUDA implementation of the extraction of endmembers of the pFUN Algorithm.

5.7). For this purpose, *cudaMalloc* function allocates on the device so many bytes of linear memory as the data size and returns a pointer to the allocated memory. The CUDA implementation of the pFUN algorithm adds an array which contains the indexes of the endmembers (*d_indexes*) for a better memory device space usage. Moreover, as compared with the pFUN algorithm which stores all orthogonalized and orthonormalized endmembers, matrices Q and U, respectively, the proposed CUDA implementation of pFUN only keeps the last computed orthogonalized and orthonormalized endmembers, arrays *d_q* and *d_u*, respectively. For this reason, we only allocate N_b (number of bands) elements for these arrays (see section 2 of Figure 5.7), instead of $N_b \times N_p$ elements which should have been allocated if we had used the approach of the algorithm.

The last two operations in the section 2 of Figure 5.7 correspond with allocation of pinned memory for both *exit* and *d_p* variables. The former will determine the stop of the endmember extraction process, and the latter will keep the number of endmembers. These variables will be explained later.

The auxiliary copy of the hyperspectral cube of the image, *d_M*, into *d_X* is carried out by means of the *cublasScopy* function (see section 3 of Figure 5.7). This copy is needed because the original image is modified during the orthogonalization process in each iteration of the algorithm.

5.7.2.2 Extraction of the first endmember

As of now, the calculation of the first endmember is differentiated from the calculation of the rest of the endmembers. Although they have some parts in common, they are slightly different. Section 4 of Figure 5.7 presents the extraction of the first endmember by means of the following steps:

The first step (step (1) in section 4 of Figure 5.7) corresponds to the the computation of the *modified Gram-Schmidt orthogonalization* to get the first endmember. The cuBLAS implementation of this step is explained in detail in Figure 5.8. In this Figure the orthogonalized computation starts from two inputs which have already been located on the device memory: the auxiliary copy of the hyperspectral image (*d_X*) and the average pixel, (*d_centroid*). After this, several calls to cuBLAS functions are performed, as Figure 5.8 shows: in △1 the call to the *cublasSgemv* function calculates $d_X \circ d_centroid$ on the device; in △2 the call to the *cublasSgemv* function is used to calculate $d_centroid \circ d_centroid$, and in △3 the call to the *cublasSgemm* function performs the complete orthogonalization

using the image (d_X), the centroid ($d_centroid$) and the results obtained in 1 and 2. According to the cuBLAS documentation [181] *cublasSgemv* performs the array-matrix multiplication

$$y = \alpha\, op(A)\, x + \beta\, y \tag{5.13}$$

where A is a matrix stored in column-major format, x and y are vectors, and α and β are scalars. The $op(...)$ denotes the use or not of the transpose operation. Furthermore, *cublasSgemm* performs the matrix-matrix multiplication

$$C = \alpha\, op(A)\, op(B) + \beta\, C \tag{5.14}$$

where C, A and B are matrices stored in column-major format, and α and β are scalars. Figure 5.8 also shows the specific operands of the calls to *cublasSgemv* and *cublasSgemm* to calculate the Gram-Schmidt orthogonalization during the first member extraction.

The second step (step (2) in section 4 of Figure 5.7) uses a kernel to calculate $d_X \circ d_X$. The parallel execution of this kernel is represented in Figure 5.9. This kernel works on d_X, which contains the image obtained in the previous step. Although this operation could have been calculated by means of cuBLAS, we obtained better execution times with our kernel due to the size of the image. In this kernel, each thread acts over a pixel (column) of the image and calculates $d_xj \circ d_xj$ where $j = 1 \ldots Np$ (number of pixels). That is, this kernel calculates the sum of the squares of the bands per each pixel, and the result is an array with Np elements.

In the third step (step (3) in section 4 of Figure 5.7) two consecutive kernels are used to calculate the maximum value, d_max, and its position, d_max_index, on the array obtained in the last step (named d_y in Figure 5.7). This index provides the position of the pixel (d_max_index) considered as an endmember in the original image and it is kept in the device memory ($d_indexes[0]$) until the end of the extraction of all endmembers. Moreover, this kernel also initializes the number of endmembers (d_p) to one.

Although cuBLAS provides a function to find the (smallest) index of the element of the maximum magnitude, it has been checked that the kernels developed in this work provide better results in terms of execution times. These kernels use an efficient reduction technique to find the maximum value. In accordance with this technique, each block

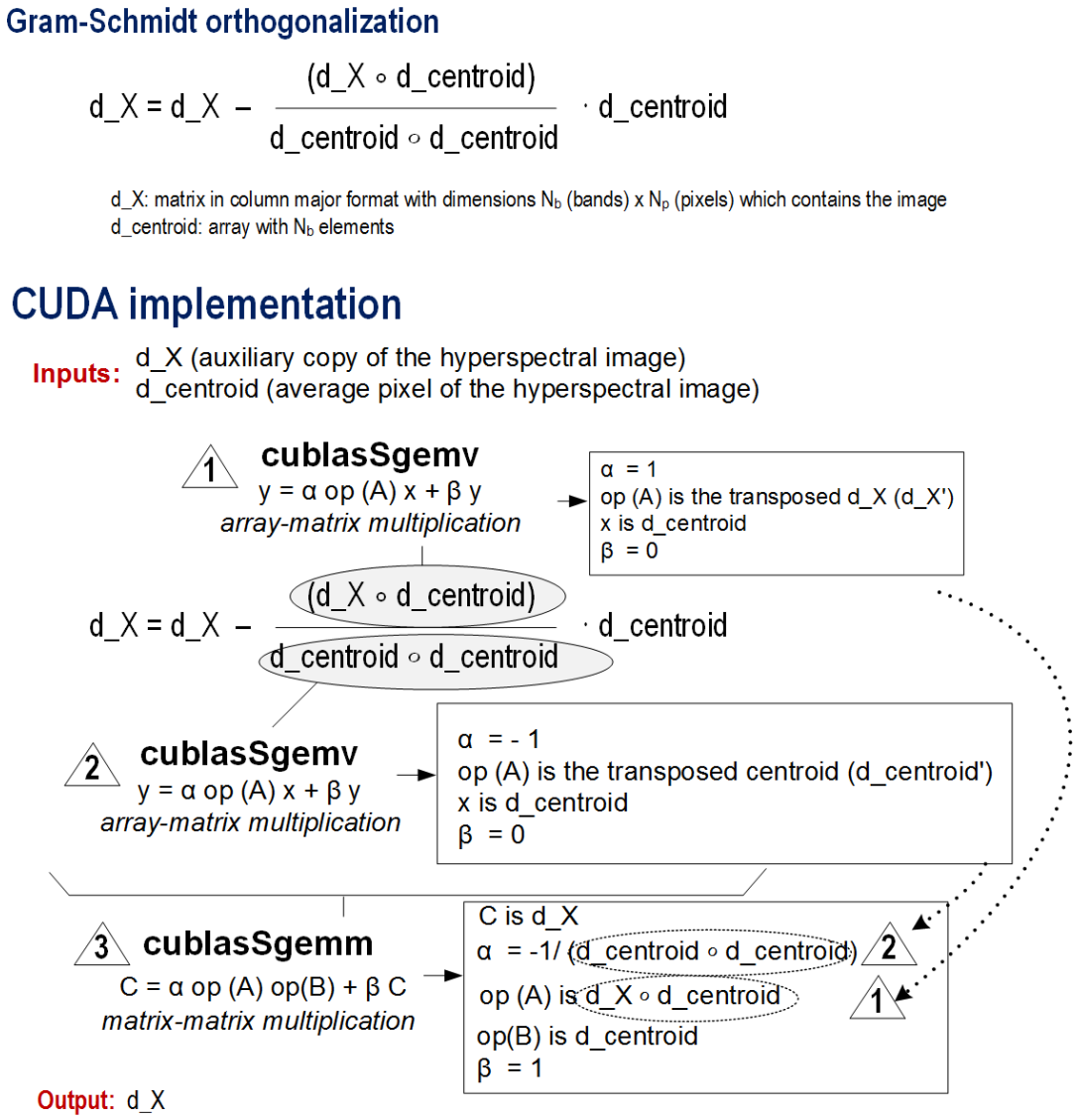


FIGURE 5.8: cuBLAS implementation of the orthogonalization in the first endmember extraction.

of threads works on a chunk of data and provides the maximum value of its chunk. Although this approach needs a second kernel to obtain the maximum value among the values calculated by the first kernel, it assures that all threads per warp take the same path and there is no thread divergence, hence improving the resource usage. This is achieved because any thread block calculates the maximum value over a copy of the original array in shared memory.

Finally, in the fourth step (step ④ in section 4 of Figure 5.7), a kernel is launched to extract the bands of the pixel located in the pixel (column) d_max_index of the image d_X. Each thread of this kernel extracts a single band value of the selected pixel. After

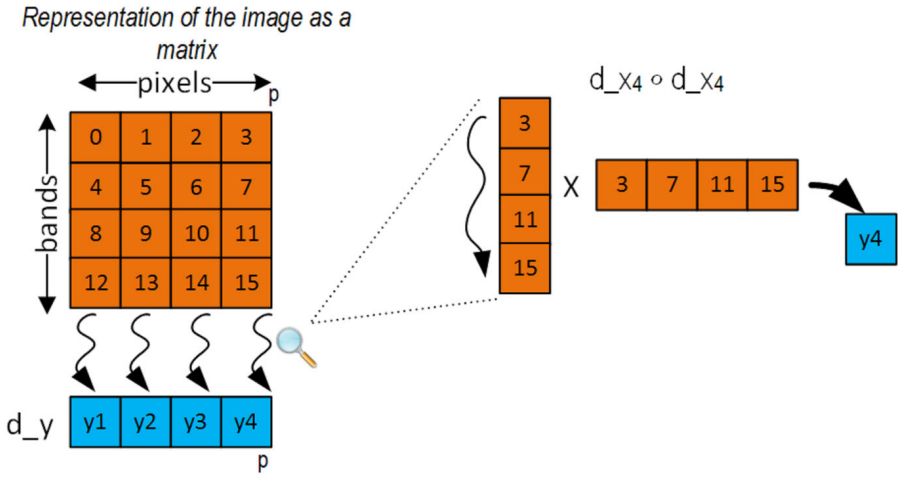


FIGURE 5.9: Representation of the kernel to calculate $d_X \circ d_X$.

this step, an array of N_b elements, d_q_1, which constitutes the first endmember of the hyperespectral image, is obtained.

As opposed to Algorithm 5.2, in the CUDA implementation the endmembers are not kept on the device during the process of endmember extraction. Instead of it, the indexes of the pixels considered as endmembers are kept in the memory device. This improves the memory device usage because these indexes (integers) use less memory than endmembers (each endmember is an array with N_b elements). The endmembers will be obtained from these indexes after the process of extraction. In the same way, only the last calculated orthogonalized and orthonormalized arrays, d_q and d_u, respectively, are kept in the memory device because only the last calculated arrays are used in the next iteration of the process. Furthermore, lines $P.5$, $P.8$, $P.6$ and $P.9$ of the algorithm presented in Figure 5.7 are reduced to keep d_q in the memory device. This is the reason why the *cudaMalloc* operation for d_q and d_u only allocates memory for a single pixel (array with bands elements) (see section 1 in Figure 5.7), and not for all the values of pixels of the image (matrix with $bands \times pixels$ elements).

5.7.2.3 Extraction of the rest of endmembers

The process of extraction of the rest of endmembers can be summarized as follows. First the modified Gram-Schmidt orthogonalization is computed to obtain d_X. After that, an array which contains the result of $d_X \circ d_X$ operation is obtained. The index of this array with the maximum value, provides the position of the endmember in the original

image. Finally, this endmember is extracted from the original image and stored into d_q_p, an array of N_b elements.

In order to extract the rest of endmembers, first the operations in sections 5 and 6 of Figure 5.7 are performed. Thus, lines $P.7$ and $P.10$ of the algorithm in Figure 5.7 are implemented by *cublasSgemv* function and an additional kernel. *cublasSgemv* calculates $d_e_1 \circ d_e_1$ in parallel on the device (see section 5, step (1) in Figure 5.7) and provides the result of this operation, (d_y), to a kernel which completes the rest of operation in the line $P.7$ of the algorithm (see section 5, step (2) in Figure 5.7). In this case, and in accordance with (5.13), $op(A)$ is the transposed array $(d_e_1^{'})$, x is the original array (d_e_1), and the scalar β is set to zero and α to 1. The kernel is launched with so many threads as the number of bands per pixel (one block with number of bands threads). This means that each thread acts in parallel over a single band value of the pixel selected as first endmember in order to complete the operation in line $P.7$. In other words, every thread is in charge of dividing the value in its band (d_e_1) by the result obtained by the cuBLAS function $(d_y = d_e_1 \circ d_e_1)$ (see Figure 5.10). Every thread also stores the result of this operation into d_u.

Moreover, section 6 includes a call to *cublasScopy* function in order to work with the original hyperspectral image because this image has been altered during the orthogonalization process to extract the first endmember.

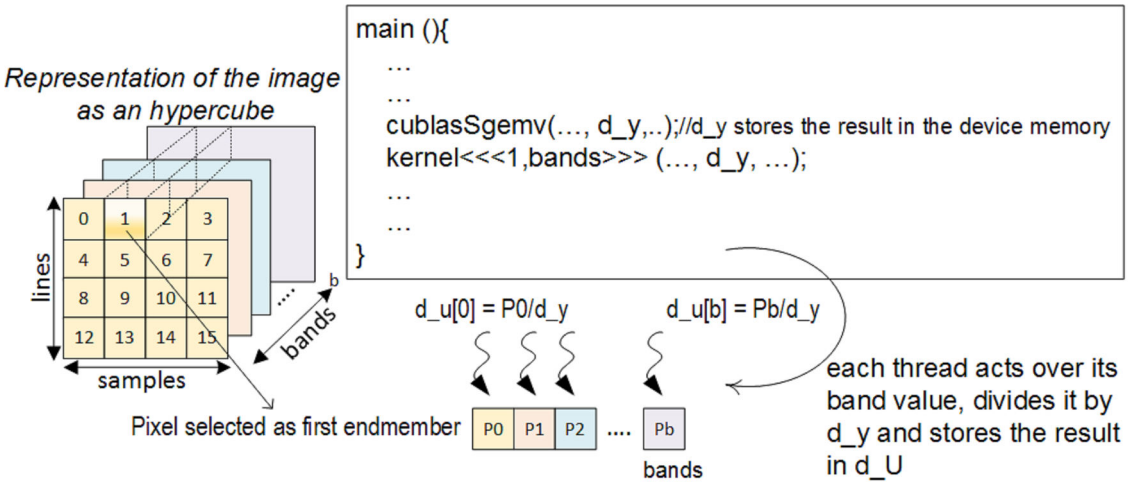


FIGURE 5.10: Representation of the CUDA implementation of lines 7 and 10 of the algorithm in Figure 5.7.

The *stop condition* in line $P.13$ of the algorithm in Figure 5.7 has been developed by using pinned memory for the *exit* variable (see section 6 of Figure 5.7). This is needed because this variable changes its value from zero to one on the device side and the stop condition

is evaluated on the host side. The use of pinned memory avoids transferring the value of the *exit* variable in any iteration of the loop. Moreover, the use of pinned memory allows us to access the value of the *exit* variable from both host and device side without any transfer from device to host. Pinned memory is also used for the number of endmembers variable (d_p) which is updated in the kernel which calculates the maximum value (see sections 4 and 8 of Figure 5.7) and also used on the host side.

The modified Gram Schmidt orthogonalization to extract the rest of endmembers (see section 7 of Figure 5.7) is developed by means of two consecutive calls to cuBLAS functions: *cublasSgemv* and *cublasSgemm* (see Figure 5.11). These functions ensure the parallel calculation of the Gram-Schmidt orthogonalization on the device. In the proposed CUDA implementation the loop used in the algorithm (line *P*.14) is not needed because cuBLAS provides functions to manage the whole image as a matrix with $N_b \times N_p$ elements (where N_b is the number of bands per pixel and N_p is the number of pixels). For the Gram Schmidt orthogonalization the *cublasSgemv* to calculate $-(d_X \circ d_q_p)$ is called first. This function call follows (5.13), where $op(A)$ is the transposed matrix of the whole image (d_X'), the array x is d_q_p (array with N_b elements which represents the orthogonal vector related to the endmember p), α is set to -1 and β to 0. Next, a call to the *cublasSgemm* function is done to complete the Gram Schmidt orthogonalization. Following (5.14) for *cublasSgemm*, C is the image (d_X), $op(A)$ is $d_u_p^*$, $op(B)$ is the result from the prior call to the *cublasSgemv* function, and α and β are both set to 1.

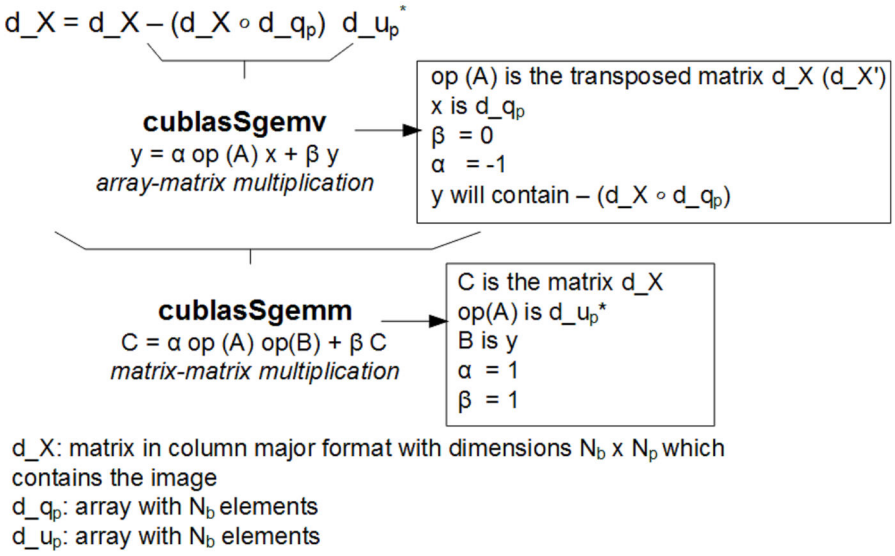


FIGURE 5.11: cuBLAS implementation for the orthogonalization to extract all the endmembers except the first one.

The next operations for the CUDA implementation of the algorithm (see section 8, steps (1) and (2) in Figure 5.7) are similar to steps (2) and (3) of the extraction of the first endmember (section 4). First, the kernel in Figure 5.9 calculates $d_X \circ d_X$ and stores its result into d_s. After this, two consecutive kernels are launched to calculate the maximum value, d_max_s, and its position, d_max_index, on the array obtained by the prior kernel, d_s. Moreover, since this last step means that a new endmember has been found, the number of endmembers, d_p, is also incremented on the device side, and d_max_index is stored into the array $d_indexes$, which keeps the position of the pixels considered endmembers in the original image.

In order to determine if the iterative process must stop or not, the next steps are followed:

1. Two consecutive calls to a kernel which extracts from the image the bands of the pixel located in the pixel d_max_index (column) (see section 8, step (3) in Figure 5.7): one call acts over the original image, d_M, and gets $d_M_{d_max_index}$, whereas the other acts over the auxiliar image, d_X, and obtains $d_X_{d_max_index}$. Each thread of this kernel extracts one band value of the selected pixel. Two arrays of N_b elements, $d_M_{d_max_index}$ and $d_X_{d_max_index}$ are obtained after this step. This last array is stored into d_q_p as a candidate to be an endmember depending on the stop condition of the algorithm.

2. In section 8, step (4) in Figure 5.7 *cublasSgemv* calculates $d_M_{d_max_index} \circ d_M_{d_max_index}$ and stores the result in an array located on the device, named d_y.

3. Since the results of the last operations are on the device, another kernel is used (see section 8, step (5) in Figure 5.7) to update the stop condition variable. This kernel is executed by an unique thread. Furthermore, the kernel updates the stop condition variable to one if the condition $100^2 \cdot d_max_s < \alpha^2 \cdot d_y$ is fulfilled (step (5) of section 8 in Figure 5.7). If the stop condition variable does not change, the array $d_u_p^*$ is updated with the $d_X_{d_max_index}/d_s$ by means of the kernel presented in Figure 5.10, and the process will continue with the extraction of endmembers. Otherwise, the process stops and a kernel extracts the endmembers from the indexes array in memory device.

5.7.3 CUDA implementation of the pFUN algorithm: abundances calculation

This section focuses on the implementation of the Algorithm 5.4 using CUDA programming model and cuBLAS library. The correspondence of this algorithm and its CUDA implementation is depicted in Figure 5.12. The same notation for sections ($\boxed{n}$), steps ($\textcircled{n}$) and lines of the algorithm that was used for the extraction of endmembers is followed in this Figure. Moreover, the parallel implementation on the device side has been carried out by either cuBLAS functions or custom kernels developed in this work. Two types of kernels can differentiated depending on the number of blocks and the threads per block used in the invocation:

(a) Kernels which are invocated with an unique block with so many threads as the number of bands of the hyperspectral image. These kernels have already been presented before in the process of endmember extraction and they have also been highlighted in blue colour in Figure 5.12.

(b) Kernels which are launched with so many blocks as the relation between the number of endmembers and the GPU block size. These kernels have also been presented before in the process of endmember extraction and they have been highlighted in red colour in Figure 5.12.

The rest of this section is organized as follows. First, the inputs and the data allocation and initialization on the device, named as *data initialization* later on (sections $\boxed{1}$ and $\boxed{2}$ in Figure 5.12) are described. Then, the process of calculation of the abundances for every endmember found in the hyperspectral image (section $\boxed{3}$ in Figure 5.12) is presented.

5.7.3.1 Data initialization

The process to obtain the abundances requires the original image, d_M, and some data obtained from the extraction of the endmembers (the endmembers, d_E, and the number of endmembers, d_p) (see section $\boxed{1}$ in Figure 5.12). All these data are already on the memory device. In section $\boxed{2}$ in Figure 5.12 the necessary data are allocated on the device ($d_Q, d_U^{**}, d_U^{*}, d_x$).

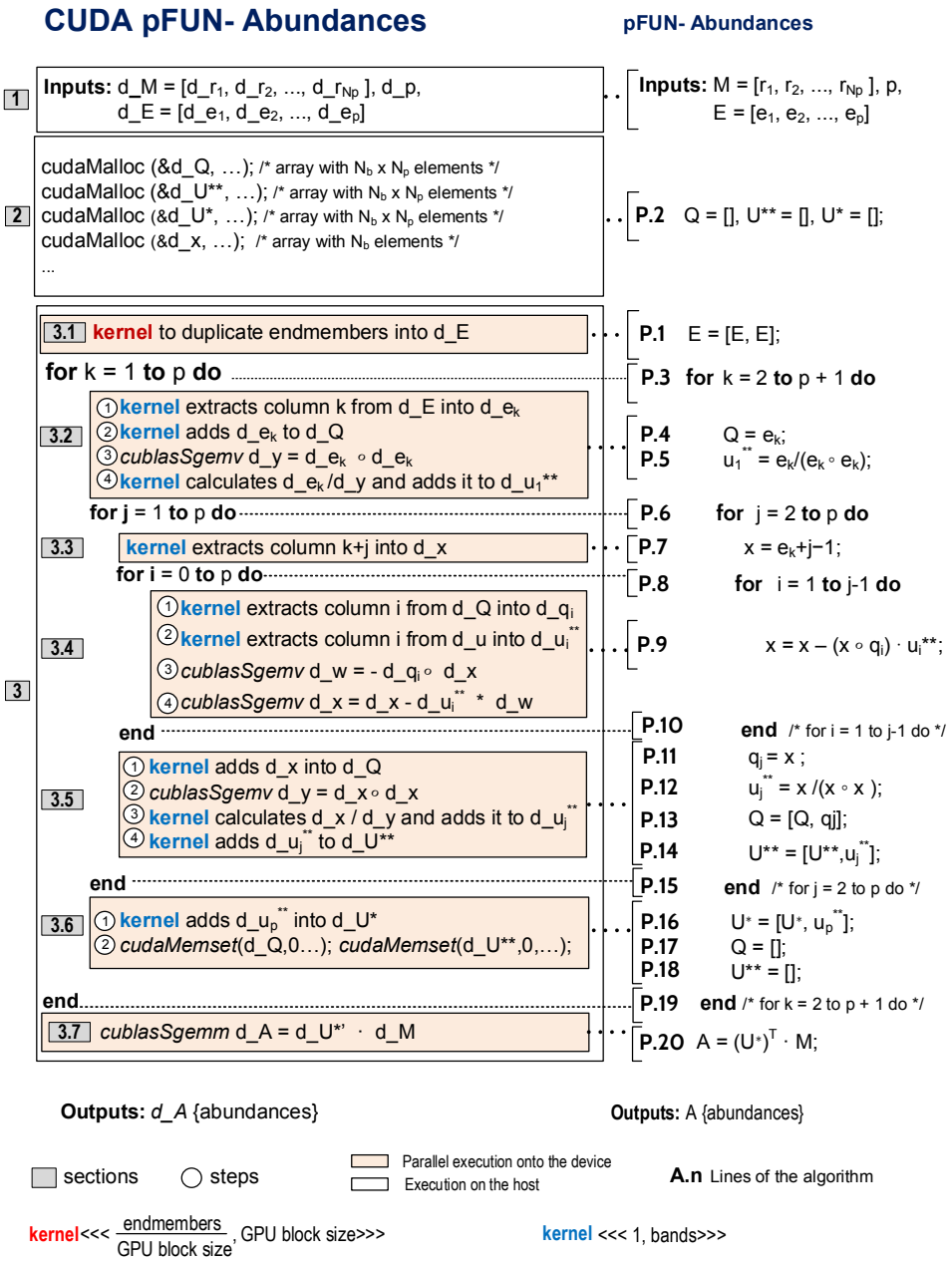


FIGURE 5.12: CUDA implementation of the abundance calculation of the pFUN Algorithm

5.7.3.2 Abundance calculation

The calculation of abundances is depicted in section 3 in Figure 5.12 by means of the following subsections:

3.1 In order to duplicate the endmembers a kernel is launched in subsection 3.1. Each thread which runs this kernel is in charge of duplicating a complete endmember (a set with so many elements as the number of bands).

3.2 Line *P*.4 of the algorithm has been developed in step (1) of subsection 3.2. This kernel extracts the endmember located in $k'th$ column of the endmember matrix and stores it into the d_e_k array. This kernel is launched with an unique block with so many threads as the number of bands, and each thread acts over a single element of the $k'th$ column. In step (2) of subsection 3.2 the d_e_k array is added as a column of the d_Q matrix. Furthermore, steps (3) and (4) of subsection 3.2 implements the line *P*.5 of the algorithm. These operations were already explained in section 5 for the extraction of endmembers in Figure 5.7. Moreover, in step (4) of subsection 3.2, the result of the last operation is added as column of the $d_u_1^{**}$ array.

3.3 In order to calculate the Gram-Schmidt orthogonalization in the next step, a kernel to extract the $k + j'th$ column of the endmember matrix is performed in subsection 3.3. This kernel also stores the extracted column into d_x.

3.4 The Gram-Schmidt orthogonalization is performed in steps (3) and (4) of subsection 3.4. This orthogonalization uses the d_x array, obtained in section 3.3, and both d_q_i and $d_u_i^{**}$, obtained in steps (1) and (2) of subsection 3.4, respectively.

3.5 Step (1) of subsection 3.5 adds the d_x array as a column of the d_Q matrix. This step is equivalent to both the lines *P*.11 and *P*.13 of the algorithm. Moreover, steps (2) and (3) implements the line *P*.12 of the algorithm by means of *cublasSgemv* and a kernel in a similar way as explained in subsection 3.2 (steps (3) and (4)). Finally, in step (4) of subsection 3.5 a kernel adds the $d_u_j^{**}$ array, obtained in the last step, to the d_U^{**} matrix. The invocation of this kernel guarantees that each thread acts over a single element of the $d_u_j^{**}$ array.

3.6 In a similar way to the last subsection, step (1) of subsection 3.6 adds the last array, $d_u_p^{**}$, obtained in the last loop, to the d_U^* matrix. Finally, step (2) empties both d_Q and d_U^* matrices in order to use them in the next iterations of the outer loop.

3.7 The last subsection performs *cublasSgemm* to obtain the abundances from the transposed d_U^* matrix and the original image, d_M. According to (5.14), $op(A)$ is the transposed matrix ($d_U^{*'}$), $op(B)$ is the original image (d_M), and the scalar α is set to one and β to 0.

5.7.4 Experimental results

The CUDA implementation of the pFUN algorithm is evaluated in this Section against the two fastest (to the best of the author's knowledge) GPU-based unmixing chains published up-to-date (see [112] and [114]) with independence of the size and content of the hyperspectral image to be unmixed and the internal characteristics of the targeted GPU. The experimental results have been measured, using different evaluation metrics and different hyperspectral images, onto the GeForce GTX 680 and the Tesla K40c NVIDIA GPUs, which characteristics have been shown in Section 5.5.

5.7.4.1 Evaluation Metrics

In order to compare the goodness of the different unmixing processes in terms of accuracy, the hyperspectral image is reconstructed according to the linear mixing model with the extracted endmembers and the abundances calculated, obtaining the *Unmixed HyperSpectral* (UHS) image. This image is then compared with the *Real HyperSpectral* (RHS) image using three different quality measures, the *Normalized Root-Mean-Square Error* (NRMSE), the *Spectral Angle Mapper* (SAM) and the *Maximum Scaled absolute Difference Error* (MaxSDE). Values of NRMSE, SAM and MaxSDE close to zero indicate a high similarity between the UHS and RHS images.

The *Normalized Root-Mean-Square Error* (NRMSE), which evaluates the global accuracy of the unmixing process, is defined as:

$$\mathrm{NRMSE} = \frac{1}{N_b \cdot N_p} \sqrt{\frac{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (\mathrm{UHS} - \mathrm{RHS})_{i,j}^2}{\sum_{i=1}^{N_b} \sum_{j=1}^{N_p} (\mathrm{RHS})_{i,j}^2}} \tag{5.15}$$

where N_b and N_p refer to the number of bands and the number of pixels of the hyperspectral image, respectively.

The *Spectral Angle Mapper* (SAM), which evaluates the spectral distortions in the unmixing process, is defined as:

$$\mathrm{SAM}(\mathrm{UHS}, \mathrm{RHS}) = \frac{1}{N_p} \cdot \sum_{j=1}^{N_p} \mathrm{SA}(\mathrm{UHS}_j, \mathrm{RHS}_j) \tag{5.16}$$

where UHS_j and RHS_j refer to the jth pixel of the UHS and RHS images, respectively. SA represents the spectral angle between two hyperspectral pixels, and is defined as:

$$\mathrm{SA}(\mathrm{UHS}_j, \mathrm{RHS}_j) = \arccos \frac{\mathrm{UHS}_j \circ \mathrm{RHS}_j}{\|\mathrm{UHS}_j\| \cdot \|\mathrm{RHS}_j\|} \tag{5.17}$$

where "$\circ$" represents the inner product between two vectors, and "$\|\cdot\|$" represents the l_2 norm of these vectors.

The *Maximum Scaled absolute Difference Error* (MaxSDE) uses the mean of each pixel to normalize the error [182]. This metric is computed as the ratio of the maximum absolute difference between pixels of both images [RHS_i *and* UHS_i], and the average absolute spectrum of that pixel using the following expression:

$$\mathrm{MaxSDE} = \max_i \{ \frac{N_b \cdot \|\mathrm{RHS}_i - \mathrm{UHS}_i\|}{\sum_{i=1}^{N_p} \|\mathrm{RHS}_i\|} \} \tag{5.18}$$

5.7.4.2 Performance of the pFUN Algorithm

First of all, the time required by the CUDA implementation of the pFUN algorithm onto the GeForce GTX 680 NVIDIA GPU for processing hyperspectral images with different number of pixels, bands and endmembers, has been measured for evaluating how the image characteristics affect the execution time of the pFUN implementation. Figure 5.13 reflects that although the execution time increases as the number of endmembers, bands and pixels increases, the most significative increase is obtained when the number of endmembers of the image is increased. This is due to the fact that the number of endmembers is only known after the extraction of endmembers and only then, this number will be involved in launching kernels which will process some tasks in parallel. In this sense the number of endmembers is used in Figure 5.7 (see kernel in red colour) to obtain the number of thread blocks which will work in parallel to extract the endmembers from the indexes obtained during the extraction of endmembers of the CUDA implementation of the pFUN algorithm. Moreover, the number of endmembers is also used in Figure 5.12 to obtain the number of thread blocks which will work in parallel to obtain the duplication of endmembers needed to initiate the abundance calculation in the CUDA implementation of the pFUN algorithm.

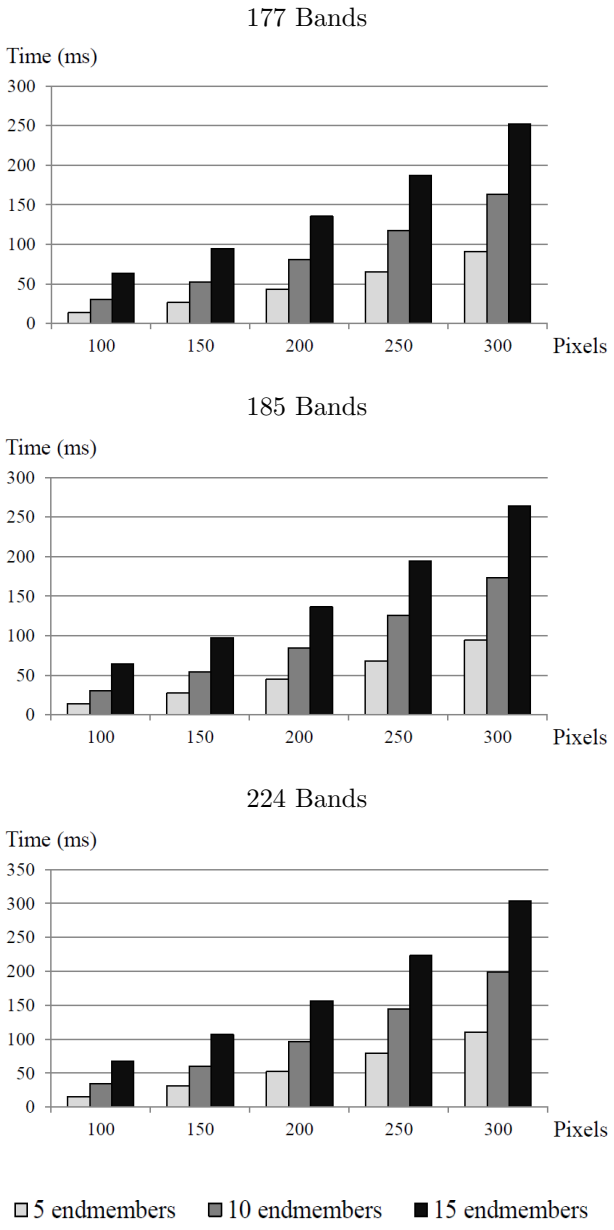


FIGURE 5.13: pFUN execution on a GeForce GTX 680 with different parameters for different synthetic images.

Table 5.6 shows the execution time for the parallel implementation of the two CUDA-based unmixing chains used as references in this work and the CUDA code of the pFUN algorithm with the hyperspectral images in Table 5.5 and the GeForce GTX 680 and Tesla K40c GPUs. To the best of author's knowledge, the two GPU-based unmixing chains used as references in this work represent the fastest state-of-the-art solutions based on NVIDIA GPUs capable of fully unmixing hyperspectral images. Both chains have

been heavily optimized by their authors for being executed onto NVIDIA GPUS, allowing for a more efficient exploitation of hyperspectral data in applications requiring a fast response. In Table 5.6 the first unmixing chain [112] performs the following algorithms to extract the endmembers and their abundances: HySime (Hyperspectral Signal identification by minimum error), PCA (Principal Component Analysis), N-Finder and LSU (Linear Spectral Unmixing), whereas the second unmixing chain [114] runs the VD (Virtual Dimensionality), SPP (Spatial Preprocessing), VCA (Vertex Component Analysis) and LSU to complete the endmember extraction and calculate the abundances. Hence, both proposals are based on well-known unmixing algorithms which have largely demonstrated their validity for linear spectral unmixing. The last two columns in Table 5.6 reflect the speed up of the unmixing process using the pFUN algorithm in relation to the unmixing chain 1 and 2, respectively. These results clearly prove that the pFUN algorithm is much faster than the two unmixing chains. In particular, the results obtained reveal that our GPU-based proposal is able to reduce the time required for unmixing a hyperspectral image in a percentage that approximately goes from a 50% (worst case) to a 85% (best case).

Table 5.6: Speedup for the parallel pFUN implementation in relation with two unmixing chains

GeForce GTX 680								
Image	Unmixing Chain 1		Unmixing Chain 2		pFUN		Speedup	
	endmembers	seconds	endmembers	seconds	endmembers	seconds	C. 1	C. 2
Cuprite	18	0.464	16	0.699	16	0.176	2.64	3.97
Indian Pines	12	0.288	28	0.747	18	0.137	2.09	5.45
World Trade Center	23	2.33	27	2.224	16	0.823	2.83	2.7
Tesla K40c								
Image	Unmixing Chain 1		Unmixing Chain 2		pFUN		Speedup	
	endmembers	seconds	endmembers	seconds	endmembers	seconds	C. 1	C. 2
Cuprite	18	0.484	16	0.894	16	0.155	3.12	5.78
Indian Pines	12	0.293	28	0.898	18	0.127	2.31	7.09
World Trade Center	23	2.638	27	2.8	16	0.668	3.94	4.18

Although the two GPU-based unmixing chains used as references in this work represent the fastest state-of-the-art solutions, it have been tried to go further and find the fastest possible combination with the algorithms being considered in these unmixing chains. For this purpose, the steps of the two above mentioned GPU-based unmixing chains which are equivalent have been identified, and the time consumed by each of these algorithms has been separately measured. Thus, the HySime algorithm, in the first unmixing chain, is equivalent to the VD algorithm, in the second unmixing chain, as both provide an estimation of the number of endmembers. Morever, the combination of PCA and N-FINDR algorithms of the first unmixing chain can be compared with the VCA algorithm of

the second unmixing chain (which internally incorporates a dimensionality reduction step) because they both are used to get the endmembers of the hyperspectral image.From this analysis, it can be observed that the GPU version of the VD algorithm published in [114] is faster than the GPU version of HySime published in [112], and similarly the combination of the GPU version of PCA and N-FINDR [112] is faster than the GPU version of VCA algorithm [114]. At this point, a new unmixing chain which is formed by the fastest algorithms of the two GPU-based unmixing chains used as references in this manuscript has been build. This new unmixing chain does not incorporate the spatial preprocessing algorithm which the second reference unmixing chain included. Table 5.7 reflects that the pFUN algorithm still provides better execution times than the new unmixing chain.

TABLE 5.7: Speedup for the parallel pFUN implementation in relation with a new unmixing chain

GeForce GTX 680					
Image	New Unmixing Chain		pFUN		Speedup
	endmembers	seconds	endmembers	seconds	
Cuprite	18	0.391	16	0.176	2.22
Indian Pines	12	0.288	18	0.137	2.01
World Trade Center	23	0.956	16	0.823	1.16
Tesla K40c					
Image	New Unmixing Chain		pFUN		Speedup
	endmembers	seconds	endmembers	seconds	
Cuprite	18	0.435	16	0.155	2.81
Indian Pines	12	0.311	18	0.127	2.45
World Trade Center	23	1.151	16	0.668	1.72

Furthermore, Table 5.6 also reflects that the pFUN algorithm obtains better execution times in a GPU with a higher number of cores and a lower clock frequency (Tesla K40c) in comparison with a GPU with less cores and higher clock frequency (GTX 680). This fact is supported because with a high parallelism, the image is processed in less clock cycles, and hence, the working frequency could be reduced without penalizing the processing time. In other words, the high parallelism of the pFUN algorithm greatly benefits from the high number of cores of the Tesla K40c despite of this GPU has a reduced clock frequency (745 MHz) with regard to the GeForce GTX 680 (1,058GHz). This reasoning proves the quality of the pFUN implementation from a parallelism point of view.

Moreover, the evaluation metrics presented in the subsection 5.7.4.1 have been calculated for the pFUN algorithm and the reference unmixing chains, as it is shown in Table 5.8. These evaluation metrics are mostly smaller for the parallel implementation of the pFUN algorithm than the other unmixing chains, demonstrating the goodness of the pFUN implementation also from an unmixing point of view.

TABLE 5.8: Evaluation metrics for the unmixing chains and the parallel pFUN implementation

	Cuprite			Indian Pines			World Trade Center		
	pFUN	U.C 1	U.C 2	pFUN	U.C 1	U.C 2	pFUN	U.C 1	U.C 2
NRMSE	0.008	0.107	0.071	0.007	0.010	0.040	0.021	0.015	0.280
SAM	0.478	0.414	1.204	0.408	0.564	1.760	2.394	2.003	9.169
MaxSDE	0.120	0.209	0.435	0.130	0.215	1.017	9.032	9.654	17.98

5.8 OpenCL Implementations of the pFUN algorithm

This section focuses on use of *Open Computing Language* (OpenCL) for implementing the pFUN algorithm onto different parallel computing platforms, the GeForce GTX 950 and the Tesla K40c NVIDIA GPUs, and the Bitware Stratix V Altera FPGAs, described in Section 5.5. The CUDA implementation of the pFUN algorithm already described in Section 5.7 is used as a reference for evaluating the suitability of OpenCL for developing applications for processing hyperspectral images under real-time constraints. The aforementioned CUDA implementation of the pFUN algorithm also targets the GTX 950 and the Tesla K40c NVIDIA GPUs, which allows to evaluate the efficiency of OpenCL for developing applications for NVIDIA GPUs against using its native programming language, CUDA.

5.8.1 OpenCL programing paradigm

OpenCL is a framework for writing programs that can be executed across heterogeneous computing platforms. OpenCL views a computing system as consisting of a number of *compute devices*, which might be *Central Processing Units* (CPUs) or 'accelerators' such as *Graphic Processing Units* (GPUs) or *Field Programmable Gate Arrays* (FPGAs), attached to a *host* processor (a CPU). OpenCL defines a C-like language for writing programs. Functions executed on the OpenCL devices are called 'kernels'. A single compute device typically consists of several *compute units*, which in turn comprises multiple *Processing Elements* (PEs). A single kernel execution can run on all or many of the PEs in parallel. However, how a compute device is subdivided into compute units and PEs is up to the vendor. The compute unit can be thought of as a 'core' but the notion of core is hard to define across all the types of devices supported by OpenCL.

In addition to its C-like programming language, OpenCL defines an *Application Programming Interface* (API) that allows programs running on the host to launch kernels on the compute devices and manage device memory, which is (at least conceptually) separated from host memory. While the OpenCL syntax and code may be portable from one platform to another, performance varies widely when the same code is run on different architectures. This results in the necessity of specifically developing the OpenCL kernels for the targeted device. Figure 5.14 graphically describes the OpenCL programming paradigm.

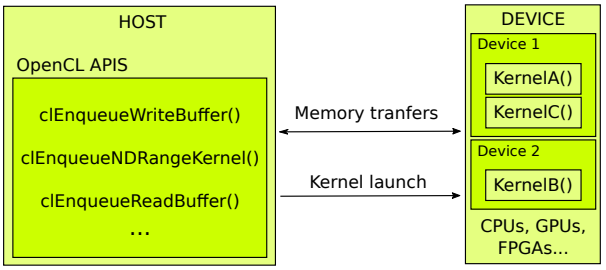


FIGURE 5.14: Diagram of the OpenCL programing paradigm

One of the main limitations of the Host-Device model is the need for intensive memory transfers, which are used for copying data from the host CPU memory to the devices global memory, which is typically carried out over the PCI Express bus. Hence, a good practice for increasing the overall performance is to reduce the number of transfers required between the host and the devices.

There exist multiple OpenCL based implementations for hyperspectral imaging analysis applications [22–24, 183]. However, there is no comparison between the results that could be obtained with OpenCL, for a particular hyperspectral imaging applications, using different parallel devices. Moreover, there is no evaluation of the efficiency of OpenCL for describing parallel hardware implementations for hyperspectral imaging applications under real-time constraints, related with other more specific hardware description platforms such as NVIDIA CUDA for NVIDIA GPUs. In this scenario, this work represents a step forward in the sense that it provides different comparisons for evaluating the suitability of OpenCL as a description language for implementing hyperspectral imaging algorithms using different computing platforms (GPUs and FPGAs). More concretely, the pFUN algorithm has been implemented in two different NVIDIA GPUs using OpenCL, with a similar structure than the same algorithm implemented in the same GPUs, but using NVIDIA CUDA. This allows the comparison between the achievable performance using OpenCL and using a more specific parallel programing platform for NVIDIA GPUs.

On the other hand, the exact same pFUN algorithm has also been implemented using OpenCL in an Altera FPGA, which allows the evaluation of the portability and efficiency of OpenCL among different architectures.

5.8.2 GPUs implementation of the pFUN algorithm using OpenCL

The pFUN algorithm has also been implemented and executed, using OpenCL, in two NVIDIA GPUs, the GeForce GTX 950 and the Tesla k40c. This implementation has been done using the same structure that was used for implementing the pFUN algorithm in the same GPUs but using NVIDIA CUDA, as described in Section 5.7, which represents the fastest implementation of the entire unmixing process. This provides fair comparisons between the performance obtained using OpenCL and one more specific parallel programing platform for NVIDIA GPUs, NVIDIA CUDA. Hence, the NVIDIA CUDA implementation of the pFUN algorithm described in Section 5.7 has been taken as reference in this Section.

The OpenCL code generation followed in this work for implementing the pFUN algorithm can be divided into three different parts:

1. **Kernels**
   The kernels, which were already written in CUDA, have been now written in OpenCL. The translation between the CUDA kernels and the OpenCL kernels is a straight forward step which requires just some minor changes. On the other hand, correctly developing the kernels is one of the most important steps since these kernels carry one the operations that will actually run in the GPUs.

2. **Libraries**
   There exist many libraries that implement many *Basic Linear Algebra Subprograms* (BLAS) for GPUs. Using these libraries for developing GPU code, when possible, usually leads to efficient results, since these libraries are widely tested and optimized [184]. In the reference CUDA version of the pFUN algorithm, the cuBLAS library was used, particularly the *General Matrix-matrix multiplication* (GEMM) operations. This library is implemented on top of NVIDIA CUDA runtime, and hence, it is not possible to use this library in an OpenCL implementation. Instead, clBLAS library has been used, since this library also provides the GEMM operations, but

implemented on top of OpenCL. However, it has been determined that the efficiency of the GEMM operations of the clBLAS library, for NVIDIA GPUs, is lower than the efficiency of the GEMM operations of the cuBLAS library [185].

3. **Host program**
   The program running in the host that was already written in CUDA, has been now written in OpenCL. This part summarizes the biggest differences between the CUDA and the OpenCL code. This is due to the fact that CUDA is specific for NVIDIA GPUs, and OpenCL allows targeting many different computing platforms. Hence, OpenCL host code requires executing some steps that are not necessarily needed in CUDA, such as setting the OpenCL platform, devices, context or queues. On the other hand, the overall OpenCL host structure developed in this work, related with the pFUN algorithm, follows the exact same structured used in the reference CUDA implementation.

While the model followed in the FPGA implementation of the pFUN algorithm, described in Section 5.8.3, tries to read each pixel just once, and perform all the operations that are to be done to that pixel before writing it back to the FPGA memory, the model followed in the GPU implementation does something very different. The main objective in the GPU implementation is that each single operation which is to be done for more that one element should be done at the same time for all the elements. This means that the pFUN algorithm has been divided into 'simple' operations that have to be repeated for many different elements, such as different pixels of the image or different bands of one pixel. Ideally, each of these operations should be executed in parallel for all the elements involved, as described in the ideal thread-level parallelism described in Figure 5.4. For doing that, one kernel (or clBLAS operation) is launched from the host to perform each of those operations for all the elements involved with the higher level of parallelism achievable, using GPU threads. The higher level of parallelism achievable is mainly limited by the data and operations dependences.

5.8.3 FPGA implementation of the pFUN algorithm using OpenCL

OpenCL has been used in this work for implementing the most computational demanding parts of the pFUN algorithm in a Bitware Stratix V Altera FPGA. The entire pFUN

algorithm is computed, using the Host-Device OpenCL model shown in Figure 5.15. In this figure, different blocks are shown, which correspond to different independent parts of the implementation of the entire pFUN algorithm.

It is important to mention that the FPGA resources are limited, and hence, it is crucial to find a balance for each part of the algorithm between the speed up that can be obtained in that particular part, the amount of resources required, and the time consumed in that part of the algorithm with respect to the time consumed in the entire algorithm. It is also important to highlight that some parts of the algorithm are executed many times, and hence, a speed up in that particular parts have a stronger impact in the time required for executing the entire algorithm. Moreover, there are parts of the algorithm that have a more parallel structure and can obtain more benefits of being implemented in the FPGA. Finally, it is important to reduce the number of data transfers between the host and the FPGA, since these data transfers usually decrease the overall performance.

It is mandatory to mention that the pFUN algorithm has been implemented in the FPGA using a single-precision floating point data type, for ensuring the exact same results that are obtained in the GPUs, allowing a fair comparison. However, while GPUs are traditionally designed for performing floating point operations, FPGAs are not, which results in a high consumption of the FPGA logic resources. This represents an extra challenge while trying to perform the implementation of the entire pFUN algorithm in the aforementioned Bitware Stratix V Altera FPGA board using OpenCL. On the other hand, some of the newest FPGAs are better prepared for floating point operations, and would be much more suitable for this kind of implementations.

The entire pFUN algorithm has been structured, according to these considerations and the model described in Section 5.4.1, as shown in Figure 5.15. The overall process can be divided in blocks described below, where labels H, D and T indicates code executed in the host, code executed in the device (FPGA), and data transfers between the host and the device, respectively.

H. **Mean pixel calculation.**
   This part of the algorithm is executed in the host, and calculates the centroid pixel of the image, which is needed for initializing the endmembers extraction process, as described in Line P.5 of the pseudocode Algorithm 5.2. This process is executed just one time, and its parallelization requires too many FPGA resources related with the speed up achieved, hence it does reports more benefits to compute it in the host.

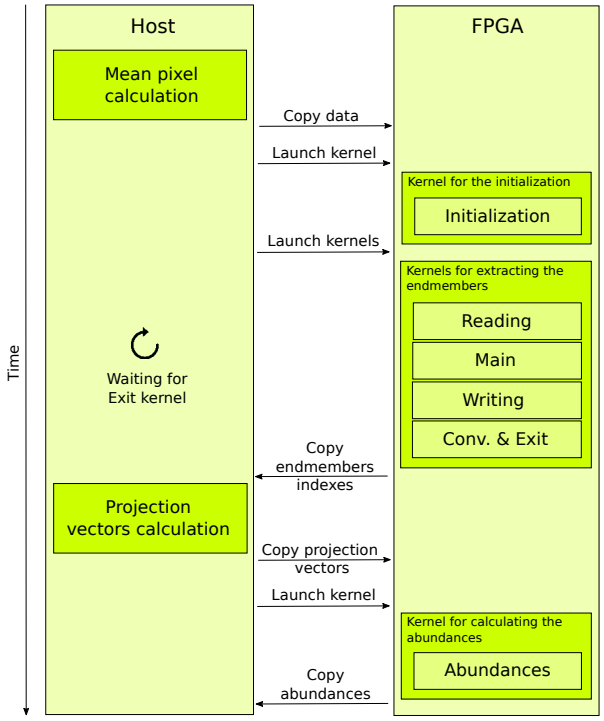


FIGURE 5.15: Structure of the pFUN algorithm implemented in the Bitware Stratix V FPGA Altera board using OpenCL.

T. **Copy data**.

The image and the centroid pixel are copied into the FPGA global memory, as well as a parameter which indicates the number of pixels in the image.

D. **Initialization.**

This block corresponds with a kernel executed in the FPGA. This kernel performs two different tasks. First, this kernel calculates the bright of each pixel of the image, automatically storing it in the global FPGA memory. Secondly, this kernel makes an extra copy of the image in the FPGA. Both the bright of each pixel and the extra copy of the image are needed in a further step of the pFUN algorithm.

D. **Endmembers extraction.**

This is the most tricky part of the pFUN algorithm and also the most computational demanding one, which corresponds with lines P.6 to P.29 of the pseudocode Algorithm 5.2. This part performs multiple iterations, one per endmember extracted, involving each of them processing, modifying and storing each pixel of the image. Four kernels and most of the FPGA resources have been used for this part of the pFUN algorithm. This endmembers calculation process will be deeply explained in

Section 5.8.3.1, since it corresponds with the most challenging part of the FPGA implementation work uncovered in this document.

T. **Copy endmembers indexes.**
Once that the endmembers have been extracted, they are copied back to the host, since this is one of the pFUN results, as shown in pseudocode Algorithm 5.2. On the other hand, the endmembers are also needed in the host for performing the calculation of the projection vectors.

H. **Projection vectors calculation.**
This part of the algorithm, executed in the host, corresponds with lines P.1 to AP.19 of pseudocode Algorithm 5.4. This part of the algorithm is not very computational demanding since it only works with the extracted endmembers. Moreover, it contains many dependences which are difficult to avoid in the FPGA without using too many resources in relation with the achievable speed up.

T. **Copy projection vectors.**
Once that the projection vectors have been computed, they are copied back to the FPGA global memory to perform the final step of the abundances calculation.

D. **Abundances calculation.**
This part of the algorithm, executed in the FPGA as one single kernel, performs the projection of the hyperspectral image using the projection vectors previously calculated in the host. This part correspond with line P.20 of the pseudocode Algorithm 5.4. This process is just a matrix multiplication, which is very easy to parallelize in the FPGA for obtaining a huge speed up, specially having in consideration the huge amount of data to be multiplied due to the size of the hyperspectral images to be processed. However, since most of the FPGA resources have been used for parallelizing the endmembers extraction process, which is the most challenging one, this part of the algorithm was just partially parallelized in this work. If more FPGA resources were available, it would be very easy to increase the parallelization level of this part of the algorithm, linearly increasing its performance.

T. **Copy abundances.**
Once that the abundances have been calculated, the abundances matrix is copied back to the host, since this is the another result provided by the pFUN algorithm as shown in pseudocode Algorithm 5.4.

5.8.3.1 FPGA endmembers extraction

The sequential process for simultaneously estimating the number of endmembers and extracting them is the most challenging part of the pFUN algorithm. This process corresponds with lines P.6 to P.29 of the pseudocode Algorithm 5.2. In each iteration of this sequential process, the operations shown in lines P.15 and P.16 are executed for each pixel of the hyperspectral image under analysis. Moreover, the values of ${s_j}^2$ obtained for each pixel in line P.16 have to be compared for identifying the pixel with the maximum ${s_j}^2$ value, as shown in line P.18. This part of the pFUN algorithm has been slightly modified for achieving a more pipeline-like structure, with the objective of reading and writing each pixel just once per endmember extracted, which leads to better results in the FPGA. This new structure of this part of the pFUN algorithm is shown in pseudocode Algorithm 5.5, in which lines M.2 to M.14 of the pseudocode Algorithm 5.5, which are the modified lines, correspond with lines P.11 to P.17 of the pseudocode Algorithm 5.2.

Algorithm 5.5: Process to extract the endmembers using pipeline

```
M.1  ...
M.2  p = 1; {Number of endmembers found}
M.3  j_max = 0;
M.4  s²_max = 0;
M.5  exit = 0;
M.6  while exit = 0 do
M.7      for j = 1 to N_p do
M.8          x_j = x_j − (x_j ∘ q_p) · u*_p;
M.9          s²_j = x_j ∘ x_j;
M.10         if s²_j > s²_max then
M.11             s²_max = s²_j;
M.12             j_max = j
M.13         end
M.14     end
M.15     if max(s²) · 100² ≤ α² · (r_{j_max} ∘ r_{j_max}) then
M.16         exit = 1;
M.17     else
M.18         p = p + 1;
M.19         q_p = x_{j_max};
M.20         u*_p = x_{j_max}/(x_j ∘ x_j);
M.21         e_p = r_{j_max};
M.22         Q = [Q, q_p];
M.23         U* = [U*, u*_p];
M.24     end
M.25 end
M.26 ...
```

The process for simultaneously estimating the number of endmembers and extracting them, according to the modification shown in pseudocode Algorithm 5.5 is as follows:

1. Read the pixel to be processed from the FPGA global memory.
2. Perform the operations shown in lines M.8 and M.9 of pseudocode Algorithm 5.5 to that pixel.
3. Write the processed pixel in the FPGA global memory.
4. Compare the s^2 value obtained for that pixel with the maximum s^2 obtained in the previously processed pixels. If the new s^2 value is larger, store it and store the pixel index too.
5. Once that all the pixels have been processed, the maximum s^2 obtained and the index of the corresponding pixel are used for computing the stop factor and evaluating if a new iteration of the algorithm needs to be performed.
6. If one more iteration of the algorithm is to be performed, the new orthogonal vectors q_p and u_p^* are computed and the process is restarted.

This process has been implemented in the Bitware Stratix V Altera FPGA board using OpenCL. For such purpose four different kernels have been designed, named *Reading*, *Writing*, *Main* and *Convergence and Exit*, as shown in Figure 5.15. Moreover, the hyperspectral image stored in the global memory of the FPGA has been replicated. These two copies of the image are shown in Figure 5.15 as *Img1* and *Img2*. The objective of replicating the image is that in each iteration the kernel will read the pixels from one image and will write them in the another one for optimizing the efficiency of the reading and writting actions through the DDR memory channel of the FPGA board. After processing all the pixels once, the image from which the pixels are read and written is switched, so that the pixels are read from the most recently modified image.

Kernel *Main* contains the operations for executing one entire iteration of the pFUN algorithm for extracting one endmember. This kernel corresponds to the lines M.7 to M.14 of the pseudocode Algorithm 5.5. Hence, it has to be executed multiple times, restarting each time that it finishes. For such purpose it has been designed as a persistent kernel. A persistent kernel is a kernel that does not terminate once it has been launched by the host. Instead, it is restarted every time it finishes. This kind of kernels cannot access the global memory of the FPGA, but instead, the inputs and outputs of this kernel are passed in this work using Altera Channels, which are implemented in the hardware as FIFOs making them relatively inexpensive and abundantly available. Due to this reason, the kernels

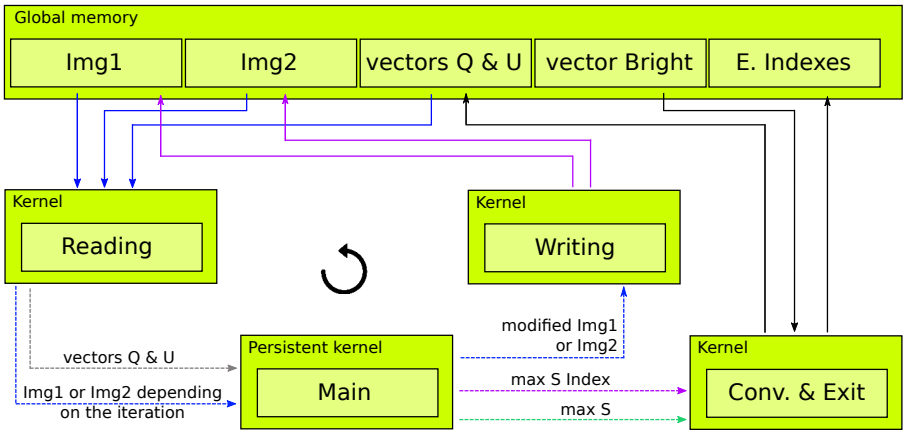


FIGURE 5.16: Endmembers extraction process implemented in the Bitware Stratix V FPGA Altera board.

Reading and *Writing* are needed. Kernel *Reading* gets the pixels from the global memory of the FPGA, reading it from the corresponding version of the hyperspectral image, *Img1* or *Img2*, according to the iteration; and sends them to the kernel *Main* through an Altera Channel, as shown in Figure 5.16. This kernel also reads vectors Q and U from the FPGA memory and sends them to the kernel *Main* through another Altera Channel. Similarly, kernel *Writing* receives the processed pixels from the kernel *Main* and stored them in the corresponding copy of the hyperspectral image, *Img1* or *Img2*, according to the iteration.

Finally, kernel *Main* sends the index of the pixel that corresponds with the maximum s^2 value and the maximum s^2 value itself to the kernel *Convergence and Exit*. This kernel uses these inputs to compute the exit condition of the algorithms, and decides if one more iteration is needed or not. If one more iteration is needed, this kernel computes the new orthogonal vectors q_p and u_p^*. The kernel *Convergence and Exit* is launched just one time from the host, and stalls in a loop that waits for each iteration of the kernel *Main* until the exit condition is achieved. When the exit condition is achieved, the kernel *Convergence and Exit* finishes, unleashing an event in the host, which indicates that all the endmembers in the hyperspectral image have been extracted and may be copied back to the host.

This process has been optimized for consuming all the bandwidth of one DDR channel for reading and writing in the global memory of the FPGA. Since the Bitware Stratix V Altera Board used in this work has actually two DDR channels, kernels *Reading*, *Main* and *Writing* have been duplicated, so each copy of these kernels works with half the pixels of the image. This way, kernel *Convergence and Exit* receives two maximum s^2 values and their corresponding pixel indexes, and makes the comparison between them for extracting

the actual maximum s^2 value and the index of its corresponding pixel before evaluating the exit condition.

Since Altera Channels work in such a way that the kernels stall when there is no data in the channels to be read, some extra Altera Channels have been used for synchronizing some parts of the process. In particular, each iteration of the kernel *Reading* waits for two Altera Channels, which ensures that the vectors q_p and u_p^*, as well as the hyperspectral image, are updated and ready to be read.

5.8.4 Experimental results

The experimental results have been measured for different hyperspectral images and different devices. The first experiment is for evaluating the efficiency of the OpenCL implementations (Section 5.8) against the NVIDIA CUDA implementation (Section 5.7) using for such purpose the pFUN algorithm. The second experiment aims to compare the efficiency of OpenCL for programing different kind of architectures, in particular two NVIDIA GPUs, the GeForce GTX 950 and the Tesla K40c, and the Bitware Stratix V Altera FPGA board.

5.8.4.1 Results using real hyperspectral images

The time consumed by the different implementations of the pFUN algorithm when unmixing the real hyperspectral images described in Section 5.6 is shown in Figure 5.17. This figure shows the time consumed by two different NVIDIA GPUs, the GeForce GTX 950 and the Tesla K40c, when programmed using NVIDIA CUDA or OpenCL. Moreover, this figure also shows the time consumed by the Bitware Stratix V Altera FPGA board, programmed using OpenCL.

The results obtained show that, as expected, the OpenCL versions of the pFUN algorithm for the NVIDIA GPUs are slower than the CUDA versions (already described in Section 5.7) for the same GPUs, but still competitive. At this point, it has to be taken into account that the pFUN algorithm has been implemented in the GeForce GTX 950 and the Tesla K40c NVIDIA GPUs using CUDA and OpenCL, as well as the cuBLAS and clBLAS libraries, respectively. In particular, all the operations of the pFUN algorithm that involve vectors and matrix multiplications have been implemented using these libraries. As already shown in Section 5.8.2, the cuBLAS library is better optimized for NVIDIA

GPUs and provide faster results than the clBLAS library. This may be the main reason why the OpenCL version of GPUs implementations of the pFUN algorithm is slower than its CUDA version for the same GPUs. It can also be observed than the results provided by the GeForce GTX 950 NVIDIA GPU are faster than the results provided by the Tesla K40c NVIDIA GPU, in both situations, programmed with CUDA and OpenCL. Whereas the Tesla K40c NVIDIA GPU has more cores than the GeForce GTX 950 (2880 and 788 respectively), its clock frequency is lower than the clock frequency of the GeForce GTX 950 VNIDIA GPU, 745 MHz and 1024 MHz, respectively. This suggests that both, the CUDA and the OpenCL implementations in both GPUs are not using all the GPUs cores, and hence, the higher frequency of the GeForce GTX 950 GPU provides more benefits than the higher number of cores of the Tesla K40c GPU in this particular situation.

On the other hand, the results obtained for the OpenCL implementation of the pFUN algorithm into the Bitware Stratix V Altera FPGA are similar, or even faster in some situations, than the results provided with CUDA implementation in the GeForce GTX 950 NVIDIA GPU. In particular, the more competitive FPGA results with respect to the fastest GPU implementation are for the smaller images. This is due to how the parallelism is inferred in both architectures. As described in Section 5.4.3, the pixels are processed in a pipeline structure in the FPGA, and hence, the time should linearly increase according to the increment of the number of pixels in the image. On the other hand, the pixels are processed in parallel in the GPUs, and hence, the time should be similar with independence of the number of pixels to be processed. These assumptions are not totally true because they are not having in consideration the time required for memory transfers, the time required for launching the kernels, and the dependences which make it impossible to achieve the ideal parallel implementation.

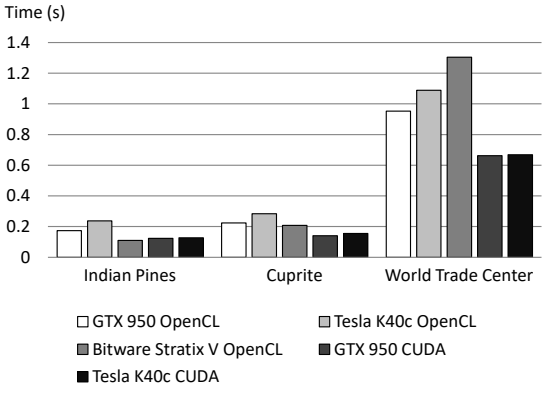


FIGURE 5.17: Comparison of the time consumed for unmixing different real hyperspectral images, using the Bitware Stratix V FPGA Altera board and the GeForce GTX 950 and Tesla K40c NVIDIA GPUs, programmed using OpenCL and NVIDIA CUDA.

5.8.4.2 Results using synthetic hyperspectral images

Figure 5.18 graphically shows how the time consumed by the OpenCL implementations on different parallel hardware architectures varies according to the number of pixels and the number of endmembers of the hyperspectral image under analysis. This figure shows the time consumed by two different NVIDIA GPUs, the GeForce GTX 950 and the Tesla K40c, and the Bitware Stratix V Altera FPGA board, programmed using OpenCL. This figure also shows the time consumed by the same two GPUs programmed using NVIDIA CUDA. The number of endmembers and pixels vary in these figures for exploring the performance of the different implementations in different situations.

First of all, the results shown in Figure 5.18 verify the increment of the time consumed by all the devices when the number of pixels and the number of endmembers in the image increase.

On the other hand, it can be seen that the increment in time caused by the increment of the number of pixels of the image is higher in the FPGA than in the GPUs. This is due to how the parallelism is inferred in both architectures. As described in Section 5.4.3, the pixels are processed in a pipeline structure in the FPGA, and hence, the time should linearly increase according to the increment of the number of pixels in the image. On the other hand, the pixels are processed in parallel in the GPUs, and hence, the time should be similar with independence of the number of pixels to processed. These assumptions are not totally true because they are not having in consideration the time required for memory transfers, the time required for launching the kernels, and the dependences which make it impossible to achieve the ideal parallel implementation.

Finally, it can also be seen that incrementing the number of endmembers in the image has a more negative impact in the GPUs, what makes sense according to the parallel implementations model described in Sections 5.4.1 and 5.4.2. This is due to the fact that in the FPGA implementation, the kernel which is persistently running in the FPGA executing the endmembers extraction process of the pFUN algorithm is launched just once, and the host just waits until all the endmembers have been extracted. On the other hand, the multiple kernels required to perform one single iteration of the pFUN algorithm for extracting the endmembers in the GPUs must be launched one time per endmember extracted. Moreover, the stop factor calculated in the GPUs is sent to the host, in each iteration, for evaluating if the process must stop or not.

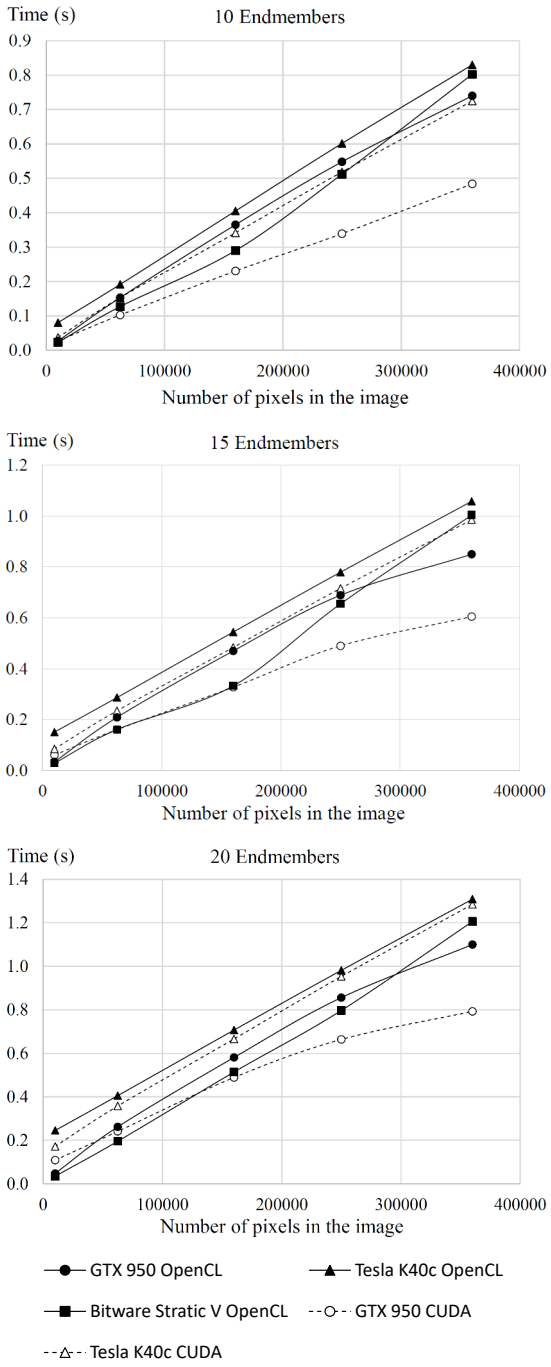


FIGURE 5.18: Comparison of the time consumed for unmixing different synthetic images, using the Bitware Stratix V FPGA Altera board and the GeForce GTX 950 and Tesla K40c NVIDIA GPUs, programmed using OpenCL and NVIDIA CUDA.

In order to provide a deeper analysis of the performance obtained in the different OpenCL implementations of the pFUN algorithm uncovered in this work, Figure 5.19 shows the proportion of the time consumed in the different parts of the process for images of different sizes and different number of endmembers. In particular, the proportion of time consumed for transferring data between the host and the devices has been measured, as well as the

time consumed in processing the data. In the developed GPU implementations of the pFUN algorithm, the hyperspectral image is copied into the GPU memory just once, and all the operations are executed inside the GPU. The host (CPU) just launches the kernels that are executed in the device (GPU), and controls when to stop the process. The percentage of time labeled as *device* in Figure 5.19 includes the time required for launching the kernels. Accordingly, the percentage of time spend in processing data in the host is null in the developed GPU implementation. On the other hand, the OpenCL implementation of the pFUN algorithm which targets the Bitware Stratix V Altera FPGA executes part of the unmixing process in the host, as described in Section 5.8.3 and graphically showed in Figure 5.15.

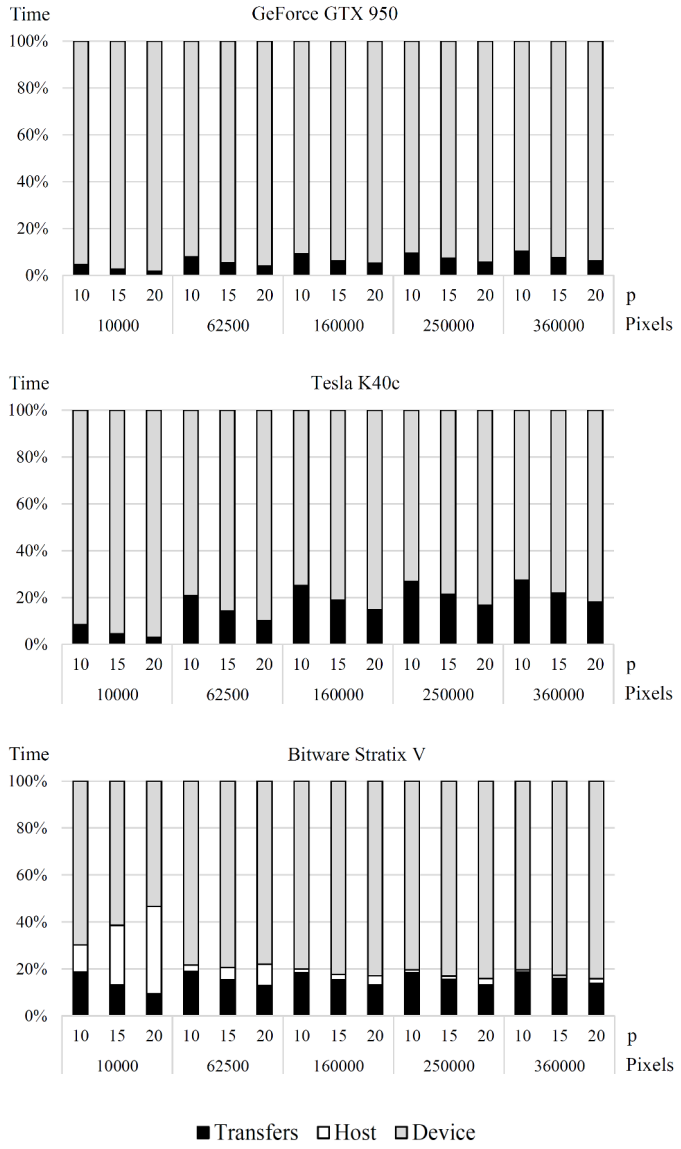


FIGURE 5.19: Percentage of time consumed in different parts of the process in the different devices and OpenCL implementations of the pFUN algorithm

Several conclusions can be dragged from the results shown in Figure 5.19. First of all, it can be observed that the impact of the memory transfers for images of a particular size is smaller when the number of endmembers increases in all the implementations and devices. This makes totally sense, since the computation required for unmixing a particular image directly depends on the number of endmembers of the image. On the other hand, the amount of data to be transfered between the host and the device mainly corresponds with the hyperspectral image and the obtained results (endmembers indexes and abundances). Within them, the hyperspectral image represents, by far, the biggest amount of data. Hence, processing images with the same size but different number of endmembers will produce a bigger difference in the amount of time required for processing the images than in the time required for transferring the data. Accordingly, the time required for transferring the data will proportionally decrease as the number of endmembers increases.

$$Number \quad of \quad operations:$$

$$Host \propto p^2 \cdot N_b \tag{5.19}$$

$$Device \propto p \cdot (N_p \cdot N_b) \tag{5.20}$$

Regarding the Bitware Stratix V FPGA implementation, we can observe that the proportion of time consumed for processing data in the host is higher for smaller images and larger number of endmembers. The most computationally demanding process executed in the host in this implementation of the pFUN algorithm corresponds with the orthogonalization of the extracted endmembers in order to obtain the set of vectors which is then used for calculating the abundances. This process is described in Algorithm 5.4. As shown in this pseudocode, the number of operations to be performed in the host is proportional to the square of the number of endmembers, as described in Equation 5.19. On the other hand, the number of operations to be computed in the device (Bitware Stratix V Altera FPGA) mostly depends on the number of endmembers and the number of pixels of the image, as described in Equation 5.20. Hence, increasing the number of pixels in the image will mainly increase the number of operations to be executed in the device, while increasing the number of endmembers in the image will have a higher impact in the host, specially for smaller images. This corresponds with the results shown in Figure 5.19.

5.8.4.3 Real-time processing capabilities of the proposed implementations

In order to evaluate the suitability of the developed OpenCL implementations of the pFUN algorithm in a real-time constrained scenario, the AVIRIS sensor data rate has been considered. Additionally, in order to evaluate the developed implementation in a more challenging scenario, the Next Generation Airbone Visible/Infrarred Imaging Spectrometer (NG-AVIRIS) data rate has also been taken into account [186]. Table 5.9 shows the size of the frames collected by these sensors as well as their achievable frame rate and data resolution.

TABLE 5.9: Specifications of the AVIRIS and NG-AVIRIS sensors

Sensor	Pixels per frame	Bands per frame	Frames per second	Data resolution
AVIRIS	677	244	12	12 bits
NG-AVIRIS	640	480	100	14 bits

In order to assess if real-time performance is achieved in the experiments, the time that would be required for collecting the different real and synthetic hyperspectral images used in the experiments, using both the AVIRIS and NG-AVIRIS sensors, is shown in Tables 5.10 and 5.11. Since all the real hyperspectral images used in the experiments have been collected by the AVIRIS sensor, and all the synthetic images used have 224 spectral bands, the time that would be needed for collecting these data using the AVIRIS sensor has been directly estimated as shown in Equation 5.21. On the other hand, the hyperspectral data collected by the NG-AVIRIS sensor have a different number of spectral channels and data resolution. Hence, the time required for collecting the different images has been estimated according to the amount of bits per second collected by the sensor and the total number of bits of the different images used, as shown in Equation 5.22.

$$Time_{AVIRIS}(s) = \frac{N_{pimage}}{677 \cdot 12} \tag{5.21}$$

$$Time_{NG-AVIRIS}(s) = \frac{N_{pimage} \cdot 224 \cdot 12}{640 \cdot 480 \cdot 14 \cdot 100} \tag{5.22}$$

According to the values shown in Tables 5.10 and 5.11 it can be observed that the time required by the classical AVIRIS sensor for collecting images of the same size than the hyperspectral images used in the experiments is much larger than the time required by the developed implementations for processing these images, with independence of the

TABLE 5.10: Time required for collecting the different real hyperspectral images used

Image	Pixels	Time (s) AVIRIS	Time (s) NG-AVIRIS
Indian Pines	21,025	2.59	0.13
Cuprite	47,750	5.88	0.30
World Trade Center	314,368	38.70	1.96

TABLE 5.11: Time required for collecting the different synthetic hyperspectral images used

Pixels	Time AVIRIS (s)	Time NG-AVIRIS (s)
10,000	1.23	0.063
62,500	7.69	0.39
160,000	19.69	1.00
250,000	30.77	1.56
360,000	44.31	2.25

number of endmembers present in the images. In this sense, Figure 5.20 compares the time required by the NG-AVIRIS sensor for collecting images with the same amount of data than the real images used in the experiments as well as the time required by the developed implementations of the pFUN algorithm for processing these images. Figure 5.21 graphically shows an equivalent comparison but using the 20 endmembers synthetic data sets used in the experiments. According to the obtained results it can be concluded that developed OpenCL implementations of the pFUN algorithms are able to provide real-time unmixing results, specially for large hyperspectral images, which represent the most challenging scenario.

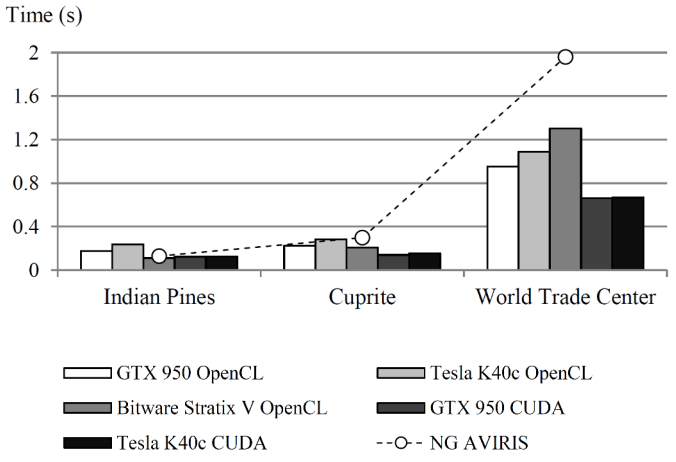


FIGURE 5.20: Comparison of the time required for processing real hyperspectral images using the different implementations of the pFUN algorithm and the time that would be required by the NG-AVIRIS sensor for collecting images with the same amount of data.

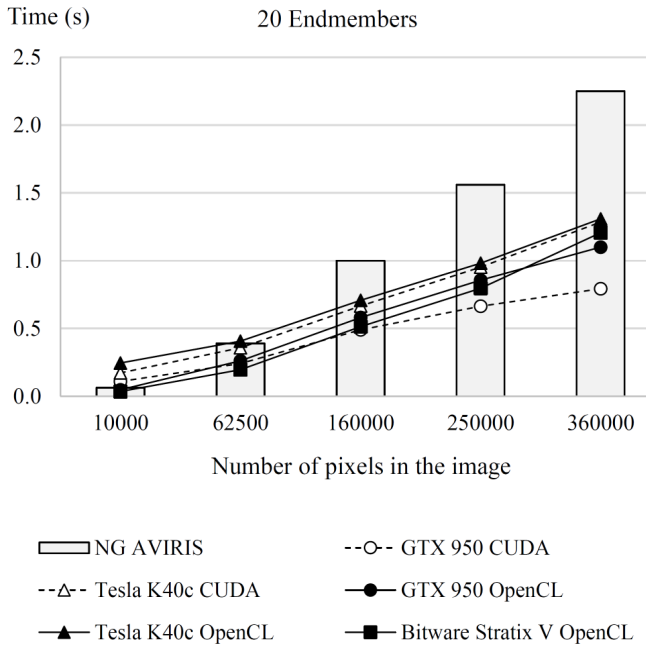


FIGURE 5.21: Comparison of the time required for processing different synthetic hyperspectral images, with 20 endmembers each, using the pFUN algorithm, and the time that would be required by the NG-AVIRIS sensor for collecting images with the same amount of data.

5.9 Conclusions

The FUN algorithm, has been implemented in different parallel computing platforms, namely GPUs and FPGAs, and using different implementation languages, namely NVIDIA CUDA and OpenCL. The accomplished implementations have been compared with the fastest state-of-the-art implementations for linearly unmixing hyperspectral images. The results obtained prove the suitability of the FUN algorithm for being implemented in parallel computing platforms, such as GPUs and FPGAs, using OpenCL as one single hardware description language for different computing platforms with efficient results. Moreover, since all the algorithms developed in this Thesis are based on orthogonal projections and make use of the modified Gram-Schmidt orthogonalization as their main operation core, it is assumed that the results and conclusions obtained for the implementations of the FUN algorithm could be transposed to the rest of the developed algorithms within this Thesis work.

The accomplished implementations has been tested using a wide variety of hyperspectral images, with different characteristics in terms of spatial and spectral resolution, size, and nature of the sensed scene (urban area, plain areas, mountains, dry areas, areas with

trees, water, etc.). This have provided very interesting comparisons between the different computing platforms and technologies as well as between the different hardware description languages and their suitability for different scenarios of hyperspectral imaging applications. The results obtained in the experiments verify the possibility of obtaining implementations for hyperspectral imaging applications under real-time constraints through the use of OpenCL for both GPUs and FPGAs devices. Furthermore, it is also observed that GPU implementations tend to be faster for processing very large images, while the smaller images are processed faster in the FPGA.

Chapter 6

Conclusions and further research

This Chapter summarizes the main contributions of this Thesis and proposes further research topics, which are expected to complement and enhance the future development for this work.

Hyperspectral imaging systems are considered nowadays one of the most powerful tools for acquiring surface information in many different fields. These systems are able to provide images in which single pixels have information from across the electromagnetic spectrum of the scene under observation. Whereas the human eye sees color of visible light in mostly three bands (red, green, and blue), hyperspectral imaging divides the spectrum into many more contiguous bands, which may be even out of the visible part of the electromagnetic spectrum. The main advantage of this technique is that certain objects leave unique 'fingerprints' in the electromagnetic spectrum, known as spectral signatures. These 'fingerprints' enable identification of the materials that make up a scanned object, providing a much better understanding of the scene under analysis.

Remote sensing is the field in which hyperspectral imaging techniques have traditionally acquired more relevance. In this field, hyperspectral sensors are tipically launched onboard satellites, for collecting spectral information of the Earth's surface or the different layers of the atmosphere. These data can be used for many different applications in the fields of oceanography, enviromental science, snow hydrology, geology, volcanology, soil and land management, atmospheric and aerosol studies, agriculture, and limology. On the other hand, many non-remote sensing applications for hyperspectral imaging have been gaining importance during the last years. Those applications are related with food processing, health care, mineralogy and surveillance, among others.

Many different algorithms able to produce accurate results have been developed for providing solutions to all those applications which make use of hyperspectral images as a source of information, which have increased the interest in this field. However, the algorithms applied for these applications require to process the huge amount of data provided by the hyperspectral sensors, which typically results in a high computational burden, preventing their use for applications under real-time constraints. The state-of-the-art solution consists on implementing the hyperspectral analysis algorithms into devices that are suitable for parallel computing, such as *Graphic Processing Units* (GPUs) and *Field Programmable Gate Arrays* (FPGAs). In fact, many works have been uncovered during the last years for implementing different algorithms for processing hyperspectral images into parallel hardware, with the purpose of speeding up the hyperspectral analysis process.

However, the algorithms for processing hyperspectral images have been typically developed with the purpose of producing the best hyperspectral analysis results, with independence of their mathematical complexity, and without having in consideration how these algorithms could be implemented into parallel computing platforms for providing results

in real-time. In example, many hyperspectral imaging algorithms use matrix operations which are very complex for large data sets, such as factorizations, the inverse of a matrix or the eigenvalues and eigenvectors calculation. Hence, when these algorithms are to be implemented into parallel devices for providing faster results, there is little room for speeding them up, due to their mathematical complexity as well as their so many data dependencies. As a result, many efforts have also been done during the last years for trying to modify or better adapt some hyperspectral image processing algorithms for being implemented into parallel computing devices in a easier and more efficient way. However, these approaches for speeding up the hyperspectral analysis are always limited by the inherent complexity of the algorithms and typically result in decreasing their accuracy.

This is why the development of more efficient hardware-friendly algorithms for processing hyperspectral images, which can be efficiently implemented in parallel hardware devices for speeding up the process is an important and exciting challenge. In this Thesis a new approach for hyperspectral imaging analysis applications under real-time constraints has been developed. This includes developing new parallel hardware-friendly hyperspectral imaging algorithms, as well as finding an efficient way of implementing the developed algorithms across different platforms.

In particular, three new hardware-friendly algorithms for processing hyperspectral images have been developed in this Thesis: the *Fast algorithm for linearly UNmixing hyperspectral images* (FUN), the *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI), and the *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA). These algorithms target three widely used hyperspectral imaging analysis processes: unmixing, hyperspectral and multispectral images fusion, and hyperspectral and ultraspectral image compression, respectively.

The FUN algorithm simultaneously performs the number of endmembers estimation and their extraction. Once that the endmembers have been extracted, this algorithm is also capable of calculating the abundances by doing just some additional operations. This algorithm is based on orthogonal projections, using the modified Gram-Schmidt method to perform them. This methodology allows the reutilization of information and a high level of parallelization. Moreover, the algorithm does not perform complex matrix operations, such as the inverse of a matrix or the extraction of eigenvalues and eigenvectors, which makes easier its ulterior hardware implementation.

In order to verify the behaviour of the FUN algorithm for the different parts of the unmixing process, and for different scenarios, real and synthetic hyperspectral images have

been used. In order to compare the results obtained with the FUN algorithm with some of the most relevant algorithms in the state-of-the-art for the same tasks, the entire unmixing process has also been performed using the unmixing chain formed by the HySIME algorithm, the VCA algorithm and the FCLSU algorithm for the same images. The experimental results demonstrate that the accuracy obtained with the FUN algorithm when estimating the number of endmembers and extracting them is similar or better than the accuracy of the HySIME algorithm and the VCA algorithm. Moreover, the accuracy of the abundances obtained with the FUN algorithm is also similar to the accuracy of the abundances calculated with the FCLSU, while the FUN algorithm calculates them much faster.

The CoEf-MHI performs the fusion of a *Low spatial resolution HyperSpectral* (LHS) image and a *High spatial resolution MultiSpectral* (HMS) image. This algorithm introduces the spatial details of the HMS image into the LHS image introducing minimal distortion, in order to obtain a *High spatial resolution HyperSpectral* (HHS) image. To do this, each pixel of HHS image is represented as a linear combination of some of its surrounding pixels. The CoEf-MHI algorithm has three important advantages with respect to the state-of-the art algorithms. First of all, each pixel of the HHS image is independently processed. Secondly, in order to perfectly represent a single pixel as a linear combination of its surrounding pixel, a low number of vectors is needed (typically no more than three pixels). Finally, the process independently performed for each pixel of the HHS image is also based on the modified Gram-Schmidt method, which has a low computational burden and does not perform complex matrix operations. All these facts turn the CoEf-MHI algorithm into a very suitable option for applications under real-time constraints.

In order to evaluate the goodness of the CoEf-MHI algorithm an extensive amount of simulations have been performed, comparing the results obtained using the CoEf-MHI algorithm with those provided by the most significant sate-of-the-art fusion algorithms up to this moment, i.e. the *Generalized Gram-Schmidt Adaptive* (GGSA) algorithm, the *Generalized Local Minimum Square Error* (GLMMSE) algorithm, the *Coupled Nonnegative Matrix Factorization* (CNMF) algorithm, the HySure algorithm, the *Bayesian Naive* (BN) algorithm and *Bayesian Sparse* (BS) algorithm. The results obtained demonstrate that the CoEf-MHI algorithm provides very good results for fusing hyperspectral and multispectral images, in terms of the spectral and the spatial quality of the fusion and with independence of the spatial and spectral characteristics of the images to be fused. In particular, the fused images obtained by the CoEf-MHI algorithm show, in average, a better spatial and spectral quality than the one obtained by the GGSA, GLMMSE, HySure,

and CNMF algorithms and a similar quality that the one achieved by the Bayesian Naive and the Bayesian Sparse algorithms under the same conditions. Moreover, the CoEf-MHI algorithm is able to complete the fusion process requiring a much lower computational complexity than all the aforementioned algorithms, which converts it in the most efficient fusion algorithm up to date.

The HyperLCA algorithm performs the spectrally decorrelation and reduction of the hyperspectral or ultraspectral data by projecting it onto a set of orthogonal vectors, similarly to all the *Kahrunen-Loève Transform* (KLT) approaches. However, the extraction of the orthogonal set of vectors is done by using a modified version of the Gram-Schmidt orthogonalization process too. This provides several advantages. On one side, the proposed methodology has a reduced computational burden in relation with KLT approaches, and can be efficiently and independently applied to small blocks of the data set with accurate compression results, what makes it more suitable for applications under real-time constraints, also providing error resilience. On the other side, the HyperLCA transform is specially design for preserving the anomalous pixels, or the most different pixels in the data set, through the compression-decompression process, what is crucial for many hyperspectral applications such as anomaly detection, tracking or classification.

An extensive amount of experiments have been performed in order to evaluate the goodness of the HyperLCA algorithm. The results obtained using the HyperLCA transform for spectrally decorrelating and reducing the data set have been compared with those provided by the *Principal Component Analysis* (PCA) transform, which provides the best results of the state-of-the-art approaches for this task. As a further step, the HyperLCA transform has been used in conjunction with the CCSDS121 entropy coder for performing the entire compression-decompression process. The results obtained have been compared with the results provided by the PCA transform in conjunction with the CCSDS122 compressor. Furthermore, the HyperLCA algorithm has also been tested for spectrally decorrelating and reducing ultraspectral images. Finally, the effect of the compression-decompression process within the HyperLCA transform (in conjunction with the CCSDS121 coder) on anomaly detection application has also been tested. As a further step, the computational burden of the HyperLCA algorithm has also been evaluated and compared with the computational burden of the PCA transform, for different images, measuring the FLOPs required for spectrally reducing the data sets. The results obtained in the experiments demonstrate that the HyperLCA efficiently a provides high compression performance. Moreover, it is specially suitable for applications under real-time constraints being able

to preserve the quality of the compressed pixels for hyperspectral image applications such as anomaly detection or classification.

The FUN algorithm has been implemented in different parallel computing devices, namely GPUs and FPGAs, and using different implementation languages, namely NVIDIA CUDA and OpenCL. The accomplished implementations have been compared with the fastest state-of-the-art implementations for linearly unmixing hyperspectral images. The results obtained prove the suitability of the FUN algorithm for being implemented in parallel hardware devices, such as GPUs and FPGAs, using OpenCL as one single hardware description language for different hardware devices with efficient results. Moreover, since the FUN, the CoEf-MHI and the HyperLCA algorithms are all three based on orthogonal projections and make use of the modified Gram-Schmidt orthogonalization as their main operation core, it is assumed that the results and conclusions obtained for the implementations of the FUN algorithm could be extrapolated to the rest of the developed algorithms within this Thesis work.

The accomplished implementations have been tested using a wide variety of hyperspectral images, with different characteristics in terms of spatial and spectral resolution, size, and nature of the sensed scene (urban area, plain areas, mountains, dry areas, areas with trees, water, etc.). The obtained results also demonstrate the suitability of OpenCL for developing real-time applications for hyperspectral imaging analysis under real-time constraints. Additionally, very interesting comparisons between the different computing devices and technologies as well as between the different hardware description languages and their suitability for different scenarios of hyperspectral images applications have been performed.

With the accomplished research work, the main objectives of this Thesis have been achieved. New hardware-friendly algorithms for processing hyperspectral images, thoughts from their conception for being implemented in parallel hardware devices for real-time applications, have been proposed. Different implementations of the developed algorithms on different parallel computing devices, GPUs and FPGAs, have also been achieved, using different implementation methodologies, namely CUDA and OpenCL. These implementations have provided very interesting comparisons between the architectures and development flows used, according to the targeted applications and the characteristics of the images to be processed. The accomplished experiments prove that real-time or near real-time results can be obtained in hyperspectral imaging applications by using algorithms thought from their conception to be implemented into parallel computing devices.

6.1 Further research work

Several topics are proposed which might be part of a future research work to achieve more efficient algorithms and implementations for hyperspectral imaging analysis in order to make this technology more suitable for real-time applications.

Regarding the algorithmic development for real-time hyperspectral imaging analysis, new different hyperspectral domains can be targeted. In particular, the information provided by hyperspectral images represents important advantages with respect to the traditional imaging systems for applications based on identifying or classifying targets in a particular scene. In this scenario, developing more hardware-friendly algorithms for domains such as classification, target detection or anomaly detection would have positive repercussions in hyperspectral imaging applications under real-time constraints.

On the other hand, there are processes which need to be applied to the raw data collected by the hyperspectral sensors before performing any further analysis of the hyperspectral data. These processes include different kind of calibrations and corrections which results strongly affect the ulterior hyperspectral analysis. Any improvement done to these process will have repercussion in hyperspectral imaging applications under real-time constraints, since they are actually the first step required in any hyperspectral imaging application.

Further research work is being carried out at this moment related with hyperspectral imaging lossy compression. The HyperLCA transform, developed in this Thesis for compression purposes, has demonstrated to provide an efficient performance for spectrally decorrelating and reducing hyperspectral images. However, this is just one of the compression stages, which should work in conjunction with an entropy coding. Due to the capability of the HyperLCA transform for independently processing small blocks of pixels of the hyperspectral image to be compressed, we are currently working on the development of an entire compression-decompression process for hyperspectral images which takes full advantage of this fact for real-time applications.

Regarding the implementation of the algorithms for processing hyperspectral images into parallel computing devices for speeding up the hyperspectral analysis process, it would be interesting to evaluate more development flows, platforms and architectures. In particular, manycore architectures, such as Kalray *Massively Parallel Processor Array* (MPPA), result a very interesting option for hyperspectral imaging processing, according to their relation between power and throughput. Furthermore, more design flows, such as Vivado

HLS for Xilinx FPGAs, should also be considered in order to better understand which are the best options for facing the development of new hyperspectral imaging applications under real-time constraints.

Furthermore, the evaluation of OpenCL for developing heterogeneous applications in which different devices are used in conjunction for providing better solutions for real-time hyperspectral imaging applications would also be interesting. In example, collecting and classifying hyperspectral video under real-time or near real-time constraints would require to execute several processes, such as collecting the data, correcting and calibrating it, classifying the hyperspectral frames and displaying the classification results. If different computing devices were able to more efficiently process different parts of this process, it could be useful to employ a combination of different devices for accomplishing the entire process. Additionally, the used of distributed computing through the used of *Remote Procedure Call* (RPC) tools such as *Internet Communications Engine* (Ice) [187] could also be considered.

Regarding the verification and validation of the developed hyperspectral application, the integration of the OpenCL design flow in the software/hardware in the loop validation would be important for speeding-up the whole design process. In this sense, the development of a verification platform that allows the management of input stimulus (hyperspectral images) and software models (or even hardware) is a very interesting future research line. Moreover, the extension of this platform to the management of heterogeneous hardware implementations should be a trending topic in the near future.

Appendix A

Sinopsis en español

Se presenta en este Capítulo una visión general del trabajo de investigación realizado en esta Tesis Doctoral, poniendo especial interés en las contribuciones en el campo del desarrollo de algoritmos para el procesado de imágenes hiperespectrales en tiempo real, así como su implementación en dispositivos paralelos a través de OpenCL.

A.1 Introducción

Los sistemas de adquisición de imágenes hiperespectrales son actualmente una de las herramientas más potentes para la obtención de información superficial en muchos campos de aplicación. Estos sistemas proporcionan imágenes en las que cada pixel posee información de la escena observada en muchas longitudes de onda. Los sensores hiperespectrales dividen el espectro electromagnético en muchas bandas contiguas, que pueden estar incluso fuera del espectro visible. La principal ventaja de las imágenes hiperespectrales es que los objetos dejan 'huellas' únicas en el espectro electromagnético, conocidas como firmas espectrales, que permiten distinguir entre materiales que podrían parecer iguales en imágenes RGB. Por consiguiente, las aplicaciones más importantes de la tecnología hiperespectral están relacionadas con distinguir o identificar materiales en una escena concreta.

La teledetección es el campo en el que tradicionalmente ha tenido más impacto la tecnología hiperespectral. En este campo los sensores hiperespectrales son colocados a bordo de satélites o aeronaves para obtener información espectral de la superficie terrestre o de distintas capas de la atmósfera. Los datos adquiridos por este tipo de sensores tienen aplicación en el campo de la oceanografía, la ciencia medioambiental, la hidrología, la geología, la vulcanología, la agricultura, etc., y permiten aplicaciones tales como la caracterización del suelo terrestre, el estudio de la atmósfera y de los gases que la componen, el estudio de la contaminación y realización de previsiones climatológicas.

Por otra parte, muchas aplicaciones industriales o de laboratorio que emplean sensores hiperespectrales han ganado importancia en los últimos años. Estas aplicaciones están relacionadas con el análisis de los alimentos, el cuidado de la salud, la minerología y la vigilancia. Una de las aplicaciones de la tecnología hiperespectral que ha despertado gran interés en los últimos años es la detección e identificación de tejidos tumorales en distintas partes del cuerpo humano. Ha sido demostrado que las imágenes hiperespectrales proporcionan información que permite distinguir entre el tejido sano y el tejido tumoral sin necesidad de emplear técnicas de detección más invasivas para el paciente.

Para el análisis hiperespectral es necesario emplear algoritmos capaces de extraer de las imágenes hiperespectrales la información de interés requerida por cada aplicación. Muchos algoritmos capaces de proporcionar buenos resultados han sido desarrollados durante los últimos años, motivando aun más el interés por la tecnología hiperespectral. Sin embargo, el desarrollo de algoritmos para el procesado de imágenes hiperespectrales se ha

llevado a cabo tradicionalmente con el objetivo de proporcionar los mejores resultados posibles, sin tener en cuenta la complejidad de los algoritmos. La complejidad de los algoritmos existentes, junto con la cantidad de datos a procesar, proporcionados por los sensores hiperspectrales, hacen muy complejo el empleo de la tecnología hiperespectral para aplicaciones que deban funcionar en tiempo real.

Con el objetivo de acelerar el análisis hiperespectral y hacerlo viable para aplicaciones que deban funcionar en tiempo real, los algoritmos desarrollados se han tratado de implementar en plataformas de cómputo paralelo, como GPUs y FPGAs. Sin embargo, la complejidad matemática de los algoritmos existentes, así como las dependencias de datos inherentes a dichos algoritmos, impiden el aprovechamiento eficiente de los recursos hardware, impidiendo así mismo su ejecución en tiempo real.

Por este motivo, el desarrollo de algoritmos altamente paralelizables, pensados desde su concepción para ser implementados eficientemente en plataformas de cómputo paralelo, es una tarea importante. En esta Tesis Doctoral se ha empleado una nueva metodología para el análisis hiperespectral en aplicaciones que deban funcionar en tiempo real. Esta metodología incluye el desarrollo de algoritmos altamente paralelizables que puedan funcionar en tiempo real, así como encontrar la forma de implementar de manera eficiente dichos algoritmos en diferentes plataformas de cómputo paralelo.

A.2 Objetivos y metodolgía de trabajo

El principal objetivo de esta Tesis Doctoral es proporcionar nuevas soluciones para el análisis hiperespectral para aplicaciones que deban funcionar en tiempo real. Este objetivo incluye desarrollar nuevos algoritmos altamente paralelizables que puedan ser implementados en dispositivos hardware de cómputo paralelo para su ejecución en tiempo real, así como encontrar una manera de implementar de forma eficiente los algoritmos desarrollados en distintos tipos de plataformas de cómputo paralelo. Este objetivo se ha dividido en los cuatro subobjetivos que se detallan a continuación:

- Desarrollar un nuevo algoritmo altamente paralelizable para el proceso de desmezclado espectral. El algoritmo desarrollado debe ser capaz de llevar todo el proceso de desmezclado espectral de tal forma que pueda ser implementado en plataformas de cómputo paralelo de forma eficiente para aplicaciones en tiempo real.

- Desarrollar un nuevo algoritmo altamente paralelizable para fusionar imágenes multiespectrales e hiperespectrales. El algoritmo desarrollado debe ser capaz de llevar a cabo el proceso de fusión evitando el empleo de operaciones matriciales complejas, así como las dependencias de datos, para que pueda ser implementado de forma eficiente en plataformas de cómputo paralelo para aplicaciones en tiempo real.

- Desarrollar un nuevo algoritmo para la compresión con pérdidas de imágenes hiperespectrales. El algoritmo desarrollado debe ser capaz de llevar a cabo la etapa de decorrelación y reducción espectral del proceso de compresión. Se desea además que el algoritmo tenga un bajo nivel de complejidad y un alto grado de paralelismo y escalabilidad, ya que estas características son especialmente requeridas por las aplicaciones que deben ejecutarse a bordo de satélites.

- Verificar la capacidad de los algoritmos desarrollados de proveer resultaods en tiempo real. Para ello los algoritmos desarrollados se implementarán en dispositivos de cómputo paralelo, FPGAs y GPUs. Además, se desea evaluar la eficiencia de OpenCL para el desarrollo de aplicaciones hiperespectrales que deban funcionar en tiempo real, en relacion con otras plataformas de desarrollo más específicas como puede ser NVIDIA CUDA para las GPUs de NVIDIA. Finalmente se desea evaluar la eficiencia de los distintos tipos de dispositivos de cómputo paralelo empleados en relación a las características de las imágenes y las aplicaciones objetivo.

A.3 Motivaciones.

La tecnología hiperespectral tiene muchas aplicaciones y cada una de ellas requiere ejecutar diferentes procesos de análisis hiperespectral. Por ejemplo, en aplicaciones cuyo objetivo es identificar tumores y diferenciarlos del tejido sano en distintas partes del cuerpo humano posiblemente se requiera la aplicación de procesos de clasificación. Sin embargo, los algoritmos de clasificación empleados pueden ser diferentes para distintas partes del cuerpo humano y pueden emplear imágenes hiperespectrales con distintas características, como distintas resolucion espacial y/o espectral, capturadas en distintos rangos del espectro electromagnético. Además, las imagenes pueden ser preprocesadas de acuerdo con las necesidades de cada aplicación en particular, usando algoritmos de reducción de ruido, corrección del brillo, llevando a cabo reducciones espaciales o espectrales del conjunto de datos, seleccionando las bandas espectrales más relevantes para las aplicaciónes, etc.

En esta Tesis Doctoral se han seleccionado tres procesos que son ampliamente empleados en el análisis hiperespectral, ya que simplificar y acelerar cualquiera de estos procesos tendrá repercuciones positivas en el procesado de imágenes hiperespectrales en general, y más concretamente en las aplicaciones que deban funcionar en tiempo real. Los tres procesos seleccionados son: desmezclado espectral, fusión de imágenes multiespectrales e hiperespectrales, y compresión de imágenes hiperespectrales y ultraespectrales.

Además, cuando los algoritmos empleados para el procesado de las imágenes hiperespectrales son paralelizables, el procesado de dichas imágenes puede acelerarse enormemente a través del uso de dispositivos de cómputo paralelos, como GPUs y FPGAs. También se ha estudiado la implementación de los algoritmos desarrollados en esta Tesis Doctoral, ya que encontrar una manera eficiente de implementarlos en dispositivos de cómputo paralelo para procesar rápidamente las imágenes traerá repercusiones positivas para el desarrollo de aplicaciones que requieran del análisis hiperespectral y deban funcionar en tiempo real.

A.3.1 Desmezclado espectral

Los sensores hiperespectrales capturan la reflectancia de los materiales en cada pixel de la escena fotografiada. La reflectancia de cada pixel es captada en el rango de longitudes de onda que es capaz de medir el sensor, obteniendo un espectro típicamente conocido como firma espectral del pixel. Dicha firma espectral puede ser empleada para identificar de forma inequívoca distintos materiales. Sin embargo, es habitual que coexistan varios materiales en un mismo pixel. Esto da como resultado píxeles cuya firma espectral no se corresponde con ninguna de las firmas espectrales de los materiales que componen dicho pixel, sino con una mezcla de ellas. El proceso de desmezclado espectral intenta identificar y diferenciar los distintos materiales presentes en la imagen, identificando para ello los materiales puros, también llamados *endmembers*, presentes en la imagen. El proceso de desmezclado espectral también evalua la abundancia de cada material en cada pixel de la imagen.

El proceso lineal de desmezclado espectral está basado en la idea de que cada pixel de una imagen hiperespectral puede ser representado como una combinación lineal de un set firmas espectrales puras, o *endmembers*, ponderados con unos factores de abundancia que definen la proporción de cada uno de los *endmembers* en el pixel en cuestión. El proceso lineal de desmezclado espectral asume que las reflexiones secundarias y los efectos de dispersión pueden ser despreciados en el proceso de adquisición de datos, y por lo tanto,

la firma espectral resultante de cada pixel puede ser expresada como la combinación lineal de las firmas espectrales de los materiales presentes en el pixel. En los casos en los que el impacto de las reflexiones secundarias y los efectos de dispersión sea relevante, es necesario recurrir a modelos de mezcla no lineal más complejos, los cuáles requieren de información sobre la geometría y las propiedades físicas de los objetos fotografiados. Esto resulta en una mayor complejidad computacional del proceso de desmezclado.

El proceso de desmezclado espectral lineal puede ser un paso preliminar a otros muchos procesos de análisis hiperespectral, como incremento de la resolución o fusión de imágenes, clasificación, detección de anomalías o de objetivos, etc. ya que permite un mejor entendimiento de la escena fotografiada. El proceso de desmezclado espectral tiene un mayor impacto en aplicaciones de teledetección, ya que las imágenes adquiridas desde satélites o aeronaves típicamente tienen una baja resolución espacial (> 5 m por pixel), lo que ocasiona que cada pixel sea una mezcla de diversos materiales.

A pesar de que muchas contribuciones diferentes se han llevado a cabo durante las últimas dos décadas, la mayoría de ellas están basadas en dividir el proceso de desmezclado espectral lineal en tres etapas consecutivas: (i) estimación del número de *endmembers* presentes en la imagen, (ii) extracción de dichos *endmembers*, y (iii) cálculo de la abundancia de cada uno de los *endmembers* extraídos en cada pixel de la imagen. A pesar de que esta cadena de desmezclado espectral ha demostrado ser capaz de producir resultados precisos para las imágenes hiperespectrales empleadas en teledetección, no deja de tener inconvenientes, como el hecho de que todas las combinaciones posibles de algoritmos requeridos para llevar a cabo el proceso de desmezclado lineal, de acuerdo con las tres etapas mencionadas, requieren un esfuerzo computacional elevado, que tiende a ser mayor cuanto mayor es la precisión de los resultados. Este hecho impide su utilización en aplicaciones que deban funcionar en tiempo real, en las cuales las imágenes hiperespectrales deben ser analizadas rápidamente. Por este motivo, existe una necesidad clara de aportar soluciones que permitan reducir el coste computacional requerido por las aplicaciones de desmezclado espectral, de forma que sea posible el aprovechamiento de los incuestionables beneficios que aportan la tecnología hiperespectral.

A.3.2 Fusión de imágenes hiperespectrales y multiespectrales

Los sensores espectrales a bordo de satélites y aeronaves capturan las imágenes espectrales midiendo la energía reflejada por la superficie de la Tierra. A partir de este proceso, distintos sensores espectrales son capaces de obtener imágenes con diferentes resoluciones espaciales, espectrales y distintos SNR. De acuerdo con la resolución espectral de las imágenes obtenidas, los sensores pueden ser clasificados como sensores multiespectrales, hiperespectrales e incluso ultraespectrales. Así mismo, las imágenes obtenidas por estos sensores tendrán desde unas pocas bandas espectrales típicamente próximas a la parte visible del espectro electromagnético (imágenes multiespectrales), a cientos o miles de bandas que normalmente cubren la parte visible e infrarroja del espectro (imágenes hiperespectrales y ultraespectrales). Sin embargo, la resolución espacial de los sistemas multiespectrales es normalmente mucho mayor que la resolución espacial de los sistemas hiperespectrales y ultraespectrales.

De acuerdo con esto, la principal ventaja de los sistemas hiperespectrales con respecto a los sistemas multiespectrales es que proporcionan más información espectral, lo que claramente beneficia muchas aplicaciones como: desmezclado espectral, clasificación o detección de objetivos. Sin embargo, una mejora en la resolución espacial de las imágenes hiperespectrales mejoraría los resultados obtenidos en estas aplicaciones. Con este objetivo en mente, y gracias a la existencia de sistemas de teledetección que incorporan tanto sensores hiperespectrales como multiespectrales, el proceso de fusión de imágenes multiespectrales e hiperespectrales se ha vuelto una tarea importante en la última década. Este proceso emplea algoritmos de fusión para obtener imágenes con una gran resolución espacial y espectral combinando de manera precisa la información adquirida por sensores multiespectrales e hiperespectrales.

Los algoritmos de fusión existentes pueden ser clasificados en métodos de sustitución de componentes, métodos de análisis multiresolución, y en métodos Bayesianos y de factorización de matrices, siendo estos últimos más recientes y complejos. Los métodos de sustitución de componentes normalmente proporcionan una buena fidelidad de los detalles espaciales en la imagen final, con una implementación relativamente fácil y rápida y cierta robustez a los errores de alineación de las imágenes. Sin embargo, estos métodos tienden a generar una significante distorsión espectral causada por la discrepancia espectral entre la imagen hiperespectral y la multiespectral. Los métodos de análisis multiresolución normalmente proporcionan consistencia y robustez a los errores de alineación de las imágenes si se aplican en condiciones adecuadas. Sin embargo, la carga computacional de estos

métodos es normalmente bastante mayor que la de los métodos de sustitución de componentes y su implementación es más compleja debido a la necesidad de emplear filtros espaciales. Por último, los métodos Bayesianos y los métodos basados en factorización de matrices para la fusión de imágenes multiespectrales e hiperespectrales son los métodos que proporcionan los mejores resultados a cambio de un incremento sustancial de su complejidad y carga computacional. Por estos motivos, el desarrollo de algoritmos de fusión más eficientes, entendiendo la eficiencia del algoritmo como la relación entre la calidad de los resultados de fusión que proporciona en relación con su complejidad computacional, representaría una importante ventaja para el análisis hiperespectral en general, y para las aplicaciones hiperespectrales que deban funcionar en tiempo real en particular.

A.3.3 Compresión de imágenes hiperespectrales y ultraespectrales

De forma similar a la compresión de imágenes RGB, la compresión de imágenes hiperespectrales y ultraespectrales consiste en reducir la cantidad de datos a enviar y guardar, obteniendo una imagen comprimida, de forma que la imagen hiperespectral o ultraespectral original puede ser recuperada a partir de la imagen comprimida, con o sin pérdidas de información, según se aplique compresión con o sin pérdidas. Sin embargo, mientras que en la compresión de imágenes RGB la información se comprime únicamente en la dimensión espacial (compresión 2D), en la compresión de imágenes hiperespectrales y ultraespectrales la información también puede ser reducida en la dimensión espectral (compresión 3D), lo que hace posible obtener mayores ratios de compresión.

El proceso de compresión de imágenes hiperespectrales y ultraespectrales resulta especialmente importante en aplicaciones de teledetección en las que el sistema de adquisición se encuentra a bordo de un satélite, dónde la capacidad de almacenamiento de información y la capacidad de transferencia de datos son limitadas. En estas circunstancias, ser capaces de comprimir rápidamente las imágenes capturadas, obteniendo ratios de compresión elevados con pocas pérdidas de información, representa una ventaja importante. Sin embargo, esto representa una tarea compleja, ya que los algoritmos empleados a bordo de satélites están sujetos a diversas restricciones, impuestas principalmente por las limitaciones de energía disponible en los satélites y por la capacidad de cómputo de los dispositivos capaces de operar en el espacio.

El incremento en la capacidad y velocidad de adquisición de datos de las nuevas generaciones de sensores hipersepectrales y ultraespectrales está haciendo más crítica la necesidad de obtener grandes ratios de compresión, haciendo necesario el empleo de técnicas de compresión con pérdidas. Los métodos de compresión con pérdidas para imágenes espectrales son típicamente evaluados de acuerdo con la relación entre el ratio de compresión alcanzado y las pérdidas obtenidas. Sin embargo, ha sido demostrado que la obtención de un nivel promedio de pérdidas reducido no garantiza que las aplicaciones hiperspectrales que hagan uso de dichas imágenes no se vean afectadas. En general, los métodos de compresión con pérdidas producen un filtro de paso bajo. Esto puede significar, por un lado, una reducción del ruido de las imágenes, afectando de forma positiva a algunas aplicaciones, pero también puede significar la destrucción de los elementos más atípicos de las imágenes, los cuales son importantes para aplicaciones como detección de anomalías, clasificación, desmezclado espectral o detección de objetivos. Por este motivo, el desarrollo de nuevas técnicas de compresión con pérdidas de imágenes hiperespectrales y ultraespectrales, que estén especialmente desarrolladas para que dichas pérdidas no afecten negativamente a las aplicaciones que empleen estos tipos de imágenes, y que puedan funcionar eficientemente a bordo de satélites, proporcionaría importantes beneficios para las aplicaciones de teledetección que deban funcionar en tiempo real.

A.3.4 Aceleración hardware. OpenCL

Las aplicaciones hiperespectrales que deben funcionar en tiempo real requieren procesar rápidamente una gran cantidad de datos. Cuando los algoritmos empleados para procesar dicha información son parelelizables, se pueden emplear dispositivos hardware de cómputo paralelo, como GPUs y FPGAs, para acelerar el procesado de los datos. Sin embargo, el desarrollo de implementaciones de algoritmos en FPGAs y GPUs requiere conocimiento y experiencia en el dispositivo a emplear, así como en las metodologías y herramientas de diseño que se pueden emplear para programar dicho dispositivo. En este ámbito, OpenCL representa una alternativa muy interesante ya que permite desarrollar implementaciones para diferentes tipos de dispositivos hardware de cómputo paralelo empleando un único lenguaje de síntesis de alto nivel.

El objetivo principal de esta Tesis Doctoral es aportar nuevas soluciones para el análisis hiperespectral para aplicaciones que deban funcionar en tiempo real. Este objetivo incluye desarrollar un nuevo set de algoritmos altamente paralelizables así como encontrar una

forma eficiente de implementarlos en distintos dispositivos hardware de cómputo paralelo. De acuerdo con estos objetivos, OpenCL representa una solución muy factible como lenguaje de descripción hardware para esta Tesis Doctoral, ya que permite programar distintos tipos de dispositivos empleando un único lenguaje con un alto nivel de abstracción, permitiendo invertir el mayor esfuerzo en el desarrollo de los algoritmos.

OpenCL interpreta el sistema de cómputo como un conjunto de aceleradores hardware tales como CPUs, FPGAs o GPUs, unidos a un *host* (una CPU). OpenCL define un lenguaje tipo *C* para escribir programas. Las funciones ejecutadas en los aceleradores se llaman *kernels*. Un acelerador se considera normalmente en OpenCL como un conjunto de unidades de cómputo, *compute units*, que a su vez pueden dividirse en muchos elementos de procesamiento, *processing elements* (PEs). Un único *kernel* puede ejecutarse en uno o muchos PEs de forma simultánea. Sin embargo, como se divide un dispositivo específico en PEs depende principalmente del fabricante de dicho dispositivo. De forma adicional al lenguaje de programación tipo *C*, OpenCL define una API que permite que los programas ejecutados en el *host* ejecuten *kernels* en los aceleradores y controlen sus memorias (las memorias de los aceleradores están separadas, al menos conceptualmente, de la memoria del *host*).

A.4 Contribuciones y conclusiones alcanzadas

Las contribuciones realizadas en esta Tesis Doctoral, de acuerdo con los objetivos establecidos son:

- Se ha desarrollado un algoritmo llamado *Fast algorithm for linearly UNmixing hyperspectral images* (FUN) capaz de llevar a cabo todo el proceso de desmezclado espectral. El algoritmo FUN está bassado en el concepto de proyecciones ortogonales y emplea operaciones muy simples que pueden ser aplicadas de forma independiente a cada pixel de la imagen. Estos factores posibilitan una implementación eficiente del algoritmo FUN para aplicaciones que deban funcionar en tiempo real.
- Se ha desarrollado un algoritmo llamado *Computationally Efficient algorithm for fusing Multispectral and Hyperspectral Images* (CoEf-MHI) para fusionar imágenes multiespectrales e hiperespectrales. Este algoritmo es capaz de procesar de forma independiente cada pixel de la imagen. El algoritmo CoEf-MHI está también basado

en el concepto de proyecciones ortogonales y emplea operaciones muy simples. Por estos motivos, este algoritmo puede ser implementado eficientemente en dispositivos hardware paralelos, siendo una muy buena opción para aplicaciones que deban funcionar en tiempo real.

- Se ha desarrollado un algoritmo llamado *Lossy Compression Algorithm for Hyperspectral image systems* (HyperLCA). Este algoritmo proporciona una solución eficiente para llevar a cabo la etapa de decorrelación y reducción espectral del proceso de compresión para imágenes hiperespectrales y ultraespectrales. La transformada HyperLCA está también basada en proyecciones ortogonales y emplea operaciones muy sencillas. Además, esta transformada está especialmente pensada para conservar los píxeles atípicos a través del proceso de compresión y descompresión. Todos estos factores convierten el algoritmo HyperLCA en una muy buena opción para la compresión de imágenes hiperespectrales y ultraespectrales en situaciones en las que el tiempo y los recursos hardware son limitados.

- Se ha implementado el algoritmo FUN en distintos dispositivos de cómputos paralelos, GPUs y FPGAs, empleando distintos metodologías, NVIDIA CUDA y OpenCL. Dado que los algoritmos FUN, CoEf-MHI e HyperLCA están todos basados en proyecciones ortogonales y emplean operaciones similares, se puede asumir que los resultados obtenidos en la implementación del algoritmo FUN, en términos de eficiencia, podrían ser extrapolados al resto de algoritmos desarrollados en esta Tesis Doctoral. Las implementaciones desarrolladas prueban que se pueden obtener resultados en tiempo real empleando algoritmos pensados desde su concepción para ser implementados de forma eficiente en dispositivos hardware de cómputo paralelo, como GPUs y FPGAs. De forma adicional se ha probado la utilidad de OpenCL para desarrollar implementaciones hardware para dispositivos de cómputo paralelo distintos, como GPUs y FPGAs, empleando un único lenguaje de descripción hardware, con resultados eficientes.

El trabajo de investigación realizado durante esta Tesis Doctoral ha llevado a alcanzar contribuciones adicionales que se detallan a continuación:

- Los algoritmos desarrollados en esta Tesis Doctoral han sido evaluados y comparados con los algoritmos que conforman el estado del arte para los distintos procesos del análisis hiperespectral seleccionados, con el objetivo de probar no solo la eficiencia

de los algoritmos desarrollados sino también la precisión de sus resultados. Estas comparaciones han dado lugar a la utilización y descripción de distintas métricas de evaluación y algoritmos del estado del arte.

- El algoritmo FUN ha sido implementado empleando distintas metodologías de diseño para un mismo dispositivo. Esto ha permitido comparar dichas metodologías, extrayendo sus beneficios y debilidades.

- El algoritmo FUN también se ha implementado en distintos tipos de dispositivos hardware (GPUs y FPGAs) empleando la misma metodología de diseño, OpenCL. Esto permite no solo verificar que dicho algoritmo puede ser empleado en aplicaciones que deban funcionar en tiempo real, sino que también ha dado lugar a comparaciones interesantes entre los distintos tipos de dispositivos, de sus ventajas e inconvenientes de acuerdo con las características de las aplicaciones y las imágenes empleadas. Además ha permitido probar la versatilidad de OpenCL como lenguaje de descripción hardware de alto nivel para dispositivos hardware heterogéneos.

La Figura A.1 resume de forma gráfica los principales objetivos y contribuciones de esta Tesis Doctoral.

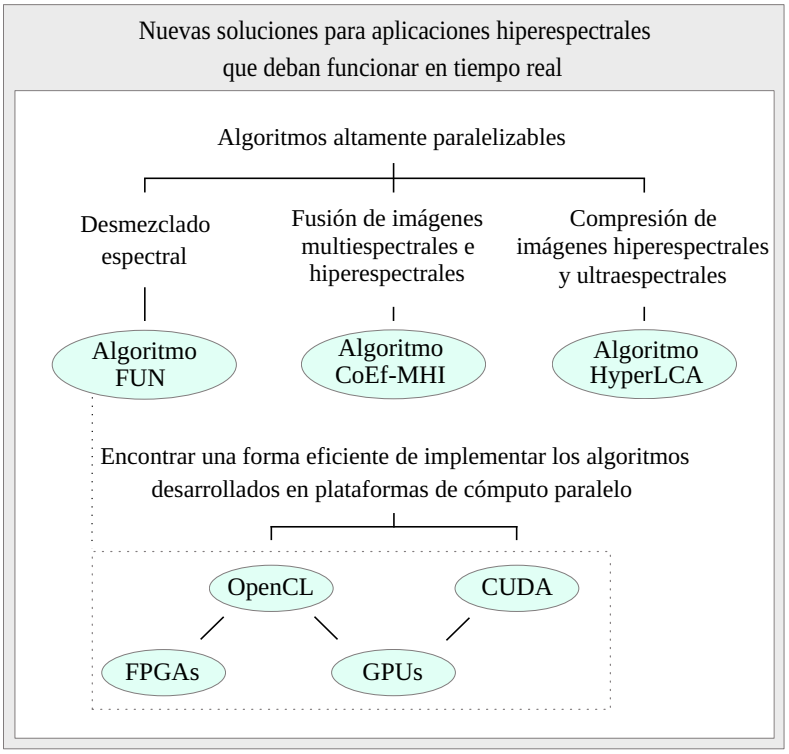


FIGURE A.1: Principales objetivos y contribuciones

Los resultados mostrados en los distintos capítulos de esta Tesis Doctoral demuestran la utilidad de todas las contribuciones realizadas para aplicaciones que empleen la tecnología hiperespectral y deban funcionar en tiempo real.

Appendix B

Publications

B.1 Journals

[1] Guerra, R., Santos, L., López, S., and Sarmiento, R. (2015). A new fast algorithm for linearly unmixing hyperspectral images. *Geoscience and Remote Sensing, IEEE Transactions on*, *53*(12), 6752-6765.

[2] Guerra, R., López, S., and Sarmiento, R. (2016). A Computationally Efficient Algorithm for Fusing Multispectral and Hyperspectral Images. *Geoscience and Remote Sensing, IEEE Transactions on*, *54*(10), 5712-5728.

[3] Martel, E., Guerra, R., López, S., and Sarmiento, R. (2016). A GPU-based processing chain for linearly unmixing hyperspectral images. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, *10*(3), 818-834.

[4] Guerra, R., Martel, E., Khan, J., López, S., Athanas, P., and Sarmiento, R. (2017). On the evaluation of different high-performance computing platforms for hyperspectral imaging: an OpenCL-based approach. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of.* Recommended for publication subject to satisfactory response to revisions suggested.

[5] Lozano, R., Madroñal, D., Desnos, K., Pelcat, M., Guerra, R., Fabelo, H., Ortega, S., López, S., Marrero, G., Juárez, E., Sanz, C. (2017). Porting a PCA-based Hyperspectral Image Dimensionality Reduction Algorithm for Brain Cancer Detection on a Manycore Architecture. *Systems Architecture, Elsevier Journal of.* Accepted for publication.

B.2 International Conferences

[1] Guerra, R., López, S., Callicó, G. M., López, J. F., and Sarmiento, R. (2013). Towards the acceleration of sequential endmember extraction algorithms for time critical applications. In *Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS 2013).*

[2] Guerra, R., López, S., Callicó, G. M., López, J. F., and Sarmiento, R. (2014). On the acceleration of the N-Finder algorithm for hyperspectral endmembers extraction. In *SPIE Sensing Technology + Applications (SPIE 2014).*

[3] Guerra, R., López, S., Callicó, G. M., López, J. F., and Sarmiento, R. (2014). On the use of Ritz Values for calculating the number of endmembers in hyperspectral images. In *Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS 2014).*

[4] Guerra, R., López, S., Callicó, G. M., López, J. F., and Sarmiento, R. (2014). A novel highly parallel algorithm for linearly unmixing hyperspectral images. In *SPIE Remote Sensing (SPIE 2014).*

[5] Guerra, R., Santos, L., López, S., and Sarmiento, R. (2015). A novel hardware-friendly algorithm for hyperspectral linear unmixing. In *SPIE Remote Sensing (SPIE 2015).*

[6] Guerra, R., Santos, L., López, S., and Sarmiento, R. (2015). GPU implementation of a linear hyperspectral unmixing algorithm. In *Conference on Design of Circuits and Integrated Systems (DCIS 2015).*

[7] Fabelo, H., Ortega, S., Guerra, R., Callicó, G. M., Szolna, A., Piñeiro, J. F., Tejedor, M., López, S., and Sarmiento, R. (2016). A Novel Use of Hyperspectral Images for Human Brain Cancer Detection using in-Vivo Samples. In *International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016).*

[8] Guerra, R., Melián, J., López, S., and Sarmiento, R. (2016). A new hyperspectral image compression paradigm based on fusion. In *SPIE Remote Sensing (SPIE 2016).*

[9] Lazcano, R., Madroñal, D., Desnos, K., Pelcat, M., Guerra, R., López, S., Juarez, E., and Sanz, C. (2016). Parallelism Exploitation of a Dimensionality Reduction Algorithm Applied to Hyperspectral Images. In *Conference on Design and Architectures for Signal and Image Processing (DASIP 2016).*

References

[1] Michael Theodore Eismann. Hyperspectral remote sensing. SPIE Bellingham, 2012.

[2] Zhongping Lee and Kendall L Carder. Hyperspectral remote sensing. In *Remote Sensing of Coastal Aquatic Environments*, pages 181–204. Springer, 2007.

[3] Éva Papp and Thomas Cudahy. Hyperspectral remote sensing. *Geophysical and Remote Sensing Methods for Regolith Exploration*, 144:13–21, 2002.

[4] Laura M Dale, André Thewis, Christelle Boudry, Ioan Rotar, Pierre Dardenne, Vincent Baeten, and Juan A Fernández Pierna. Hyperspectral imaging applications in agriculture and agro-food product quality and safety control: a review. *Applied Spectroscopy Reviews*, 48(2):142–159, 2013.

[5] Gregg Vane, Robert O Green, Thomas G Chrien, Harry T Enmark, Earl G Hansen, and Wallace M Porter. The airborne visible/infrared imaging spectrometer (aviris). *Remote Sensing of Environment*, 44(2):127–143, 1993.

[6] Hartmut H Aumann, Moustafa T Chahine, Catherine Gautier, Mitchell D Goldberg, Eugenia Kalnay, Larry M McMillin, Hank Revercomb, Philip W Rosenkranz, William L Smith, David H Staelin, et al. Airs/amsu/hsb on the aqua mission: Design, science objectives, data products, and processing systems. *Geoscience and Remote Sensing, IEEE Transactions on*, 41(2):253–264, 2003.

[7] Denis Blumstein, B Tournier, FR Cayla, T Phulpin, R Fjortoft, C Buil, and G Ponce. In-flight performance of the infrared atmospheric sounding interferometer (iasi) on metop-a. In *Optical Engineering+ Applications*, pages 66840H–66840H. International Society for Optics and Photonics, 2007.

[8] AA Gowen, CPo O'Donnell, PJ Cullen, G Downey, and JM Frias. Hyperspectral imaging–an emerging process analytical tool for food quality and safety control. *Trends in Food Science & Technology*, 18(12):590–598, 2007.

[9] D Lorente, N Aleixos, J Gómez-Sanchis, S Cubero, Or L García-Navarrete, and J Blasco. Recent advances and applications of hyperspectral imaging for fruit and vegetable quality assessment. *Food and Bioprocess Technology*, 5(4):1121–1142, 2012.

[10] Christian Fischer and Ioanna Kakoulli. Multispectral and hyperspectral imaging technologies in conservation: current research and potential applications. *Studies in Conservation*, 51(sup1):3–16, 2006.

[11] Guolan Lu and Baowei Fei. Medical hyperspectral imaging: a review. *Journal of biomedical optics*, 19(1):010901–010901, 2014.

[12] Hamed Akbari, Luma V Halig, David M Schuster, Adeboye Osunkoya, Viraj Master, Peter T Nieh, Georgia Z Chen, and Baowei Fei. Hyperspectral imaging and quantitative analysis for prostate cancer detection. *Journal of biomedical optics*, 17 (7):0760051–07600510, 2012.

[13] Hamed Akbari, Kuniaki Uto, Yukio Kosugi, Kazuyuki Kojima, and Naofumi Tanaka. Cancer detection using infrared hyperspectral imaging. *Cancer science*, 102(4):852–857, 2011.

[14] Svetlana V Panasyuk, Shi Yang, Douglas V Faller, Duyen Ngo, Robert A Lew, Jenny E Freeman, and Adrianne E Rogers. Medical hyperspectral imaging to facilitate residual tumor identification during surgery. *Cancer biology & therapy*, 6(3): 439–446, 2007.

[15] Zhi Liu, Hongjun Wang, and Qingli Li. Tongue tumor detection in medical hyperspectral images. *Sensors*, 12(1):162–174, 2011.

[16] José M Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(2):354–379, 2012.

[17] Antonio Plaza, Jon Atli Benediktsson, Joseph W Boardman, Jason Brazile, Lorenzo Bruzzone, Gustavo Camps-Valls, Jocelyn Chanussot, Mathieu Fauvel, Paolo Gamba, Anthony Gualtieri, et al. Recent advances in techniques for hyperspectral image processing. *Remote sensing of environment*, 113:S110–S122, 2009.

[18] Antonio Plaza, Pablo Martínez, Rosa Pérez, and Javier Plaza. A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data. *IEEE transactions on geoscience and remote sensing*, 42(3):650–663, 2004.

[19] Laetitia Loncan, Luis B de Almeida, José M Bioucas-Dias, Xavier Briottet, Jocelyn Chanussot, Nicolas Dobigeon, Sophie Fabre, Wenzhi Liao, Giorgio A Licciardi, Miguel Simoes, et al. Hyperspectral pansharpening: a review. *IEEE Geoscience and remote sensing magazine*, 3(3):27–46, 2015.

[20] Estanislau Auge, Josep Santalo, Ian Blanes, Joan Serra-Sagrista, Aaron Kiely, et al. Review and implementation of the emerging ccsds recommended standard for multispectral and hyperspectral lossless image coding. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*, pages 222–228. IEEE, 2011.

[21] Ian Blanes, Joan Serra-Sagrista, Michael W Marcellin, and Joan Bartrina-Rapesta. Divide-and-conquer strategies for hyperspectral image processing: a review of their benefits and advantages. *IEEE Signal Processing Magazine*, 29(3):71–81, 2012.

[22] Gustavo M Callicó, Sebastián López, Beatriz Aguilar, José F López, and Roberto Sarmiento. Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using OpenCL. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(8):3650–3659, 2014.

[23] S. Bernabé, G. Botella, J. M. R. Navarro, C. Orueta, M. Prieto-Matias, and A. Plaza. Parallel implementation of the simplex growing algorithm for hyperspectral unmixing using OpenCL. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 6153–6156, July 2016. doi: 10.1109/IGARSS.2016.7730607.

[24] Sergio Bernabé, Gabriel Martin, José MP Nascimento, José M Bioucas-Dias, Antonio Plaza, Guillermo Botella, and Manuel Prieto-Matías. A fast parallel hyperspectral coded aperture algorithm for compressive sensing using OpenCL. In *EUROCON 2015-International Conference on Computer as a Tool (EUROCON), IEEE*, pages 1–6. IEEE, 2015.

[25] Sergio Bernabé, Sergio Sánchez, Antonio Plaza, Sebastián López, Jón Atli Benediktsson, and Roberto Sarmiento. Hyperspectral unmixing on GPUs and multi-core

processors: A comparison. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1386–1398, 2013.

[26] Sebastián López, Tanya Vladimirova, Carlos Gonzalez, Javier Resano, Daniel Mozos, and Antonio Plaza. The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends. *Proceedings of the IEEE*, 101(3): 698–722, 2013.

[27] Sergio Bernabé, Sebastián López, Antonio Plaza, and Roberto Sarmiento. GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis. *Geoscience and Remote Sensing Letters, IEEE*, 10 (2):221–225, 2013.

[28] Sebastián López, Pablo Horstrand, Gustavo M Callico, José F López, and Roberto Sarmiento. A novel architecture for hyperspectral endmember extraction by means of the modified vertex component analysis (MVCA) algorithm. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(6):1837–1848, 2012.

[29] Mike Estlick, Miriam Leeser, James Theiler, and John J Szymanski. Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 103–110. ACM, 2001.

[30] Naoufal Amrani, Joan Serra-Sagristà, Valero Laparra, Michael W Marcellin, and Jesus Malo. Regression wavelet analysis for lossless coding of remote-sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 54(9):5616–5627, 2016.

[31] Ian Blanes and Joan Serra-Sagristà. Pairwise orthogonal transform for spectral image coding. *IEEE Transactions on Geoscience and Remote Sensing*, 49(3):961–972, 2011.

[32] Peg Shippert. Introduction to hyperspectral image analysis. *Online Journal of Space Communication*, 3, 2003. URL http://spacejournal.ohio.edu/pdf/shippert.pdf.

[33] William F Belokon, Spectral Imagery Training Center, and Logicon Geodynamics. *Multispectral imagery reference guide*. Logicon Geodynamics, 1997.

[34] José M Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and Remote Sensing Magazine*, 1(2): 6–36, 2013.

[35] Rob Heylen, Mario Parente, and Paul Gader. A review of nonlinear hyperspectral unmixing methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):1844–1868, 2014.

[36] Guoxia Yu, Tanya Vladimirova, and Martin N Sweeting. Image compression systems on board satellites. *Acta Astronautica*, 64(9):988–1005, 2009.

[37] Barbara Penna, Tammam Tillo, Enrico Magli, and Gabriella Olmo. Transform coding techniques for lossy hyperspectral data compression. *IEEE Transactions on Geoscience and Remote Sensing*, 45(5):1408–1421, 2007.

[38] Lucana Santos, Sebastián López, Gustavo M Callico, José F López, and Roberto Sarmiento. Performance evaluation of the h. 264/avc video coding standard for lossy hyperspectral image compression. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(2):451–461, 2012.

[39] Bruno Aiazzi, Luciano Alparone, and Stefano Baronti. Quality issues for compression of hyperspectral imagery through spectrally adaptive dpcm. In *Satellite Data Compression*, pages 115–147. Springer, 2012.

[40] Chulhee Lee, Sangwook Lee, and Jonghwa Lee. Effects of lossy compression on hyperspectral classification. In *Satellite Data Compression*, pages 269–285. Springer, 2012.

[41] Fernando García-Vílchez, Jordi Muñoz-Marí, Maciel Zortea, Ian Blanes, Vicente González-Ruiz, Gustavo Camps-Valls, Antonio Plaza, and Joan Serra-Sagristà. On the impact of lossy compression on hyperspectral image classification and unmixing. *IEEE Geoscience and remote sensing letters*, 8(2):253–257, 2011.

[42] Qian Du, Nam Ly, and James E Fowler. An operational approach to pca+ jpeg2000 compression of hyperspectral imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):2237–2245, 2014.

[43] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[44] David Kirk et al. NVIDIA CUDA software and GPU parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[45] Eric Monmasson and Marcian N Cirstea. FPGA design methodology for industrial control systems—a review. *IEEE transactions on industrial electronics*, 54(4):1824–1842, 2007.

[46] Grant Martin and Gary Smith. High-level synthesis: Past, present, and future. *IEEE Design & Test of Computers*, 26(4):18–25, 2009.

[47] Bertrand Le Gal, Emmanuel Casseau, Sylvain Huet, Pierre Bomel, Christophe Jego, and Eric Martin. C-based rapid prototyping for digital signal processing. In *Proceedings of 13th European Signal Processing Conference*, 2005.

[48] Zhiru Zhang, Y Fan, W Jiang, G Han, C Yang, and J Cong. High-level synthesis: From algorithm to digital circuit, 2008.

[49] Tom Feist. Vivado design suite. *White Paper*, 5, 2012.

[50] Felix Winterstein, Samuel Bayliss, and George A Constantinides. High-level synthesis of dynamic data structures: A case study using vivado hls. In *Field-Programmable Technology (FPT), 2013 International Conference on*, pages 362–365. IEEE, 2013.

[51] John E Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(1-3):66–73, 2010.

[52] Aaftab Munshi. The OpenCL specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

[53] Chein-I Chang and Qian Du. Estimation of number of spectrally distinct signal sources in hyperspectral imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 42(3):608–619, 2004.

[54] José M Bioucas-Dias and José MP Nascimento. Hyperspectral subspace identification. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(8):2435–2445, 2008.

[55] Michael E Winter. N-FINDR: An algorithm for fast autonomous spectral endmember determination in hyperspectral data. In *SPIE's International Symposium on Optical Science, Engineering, and Instrumentation*, pages 266–275. International Society for Optics and Photonics, 1999.

[56] José MP Nascimento and José M Bioucas Dias. Vertex component analysis: A fast algorithm to unmix hyperspectral data. *Geoscience and Remote Sensing, IEEE Transactions on*, 43(4):898–910, 2005.

[57] Chein-I Chang. Orthogonal subspace projection (osp) revisited: a comprehensive study and analysis. *Geoscience and Remote Sensing, IEEE Transactions on*, 43(3): 502–518, 2005.

[58] Joseph W Boardman. Automated spectral unmixing of aviris data using convex geometry concepts: In: Annual jpl airborne geosciences workshop, 4, pasadena, ca. *Summaries, JPL Publication*, pages 93–26, 1993.

[59] Joseph W Boardman, Fred A Kruse, and Robert O Green. Mapping target signatures via partial unmixing of aviris data. 1995.

[60] R Neville. Automatic endmember extraction from hyperspectral data for mineral exploration. In *International Airborne Remote Sensing Conference and Exhibition, 4 th/21 st Canadian Symposium on Remote Sensing, Ottawa, Canada*, 1999.

[61] Jun Li and José M Bioucas-Dias. Minimum volume simplex analysis: A fast algorithm to unmix hyperspectral data. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 3, pages III–250. IEEE, 2008.

[62] José M Bioucas-Dias. A variable splitting augmented lagrangian approach to linear spectral unmixing. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing, 2009. WHISPERS'09. First Workshop on*, pages 1–4. IEEE, 2009.

[63] Lucas C Parra, Clay Spence, Paul Sajda, Andreas Ziehe, and Klaus-Robert Müller. Unmixing hyperspectral data. In *NIPS*, pages 942–948, 1999.

[64] Saïd Moussaoui, Cédric Carteret, David Brie, and Ali Mohammad-Djafari. Bayesian analysis of spectral mixture data using markov chain monte carlo methods. *Chemometrics and Intelligent Laboratory Systems*, 81(2):137–148, 2006.

[65] Nicolas Dobigeon, Saïd Moussaoui, Jean-Yves Tourneret, and Cédric Carteret. Bayesian separation of spectral sources under non-negativity and full additivity constraints. *Signal Processing*, 89(12):2657–2669, 2009.

[66] Nicolas Dobigeon, Saïd Moussaoui, Martial Coulon, Jean-Yves Tourneret, and Alfred O Hero. Joint bayesian endmember extraction and linear unmixing for hyperspectral imagery. *IEEE Transactions on Signal Processing*, 57(11):4355–4368, 2009.

[67] Morten Arngren, Mikkel N Schmidt, and Jan Larsen. Unmixing of hyperspectral images using bayesian non-negative matrix factorization with volume prior. *Journal of Signal Processing Systems*, 65(3):479–496, 2011.

[68] Marian-Daniel Iordache, José M Bioucas-Dias, and Antonio Plaza. Sparse unmixing of hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 49 (6):2014–2039, 2011.

[69] Marian-Daniel Iordache, Antonio Plaza, and José Bioucas-Dias. On the use of spectral libraries to perform sparse unmixing of hyperspectral data. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2010 2nd Workshop on*, pages 1–4. IEEE, 2010.

[70] Derek M Rogge, Benoit Rivard, Jinkai Zhang, and Jilu Feng. Iterative spectral unmixing for optimizing per-pixel endmember sets. *IEEE Transactions on Geoscience and Remote Sensing*, 44(12):3725–3736, 2006.

[71] Daniel C Heinz and Chein-I Chang. Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 39(3):529–545, 2001.

[72] Dong Zhang, Ying ZHANG, and Huan LI. Methods for data preprocessing of hyperspectral satellite-remote sensing imagerey of coastal zone [j]. *Advances in Marine Science*, 1:013, 2009.

[73] IT Jolliffe. Principal component analysis. 1986. *Spring-verlag, New York*, 1986.

[74] L Schart Louis. Statistical signal processing: Detection, estimation and time series analysis. *Addision-Wesley Publishing company*, 1991.

[75] Andrew A Green, Mark Berman, Paul Switzer, and Maurice D Craig. A transformation for ordering multispectral data in terms of image quality with implications for noise removal. *IEEE Transactions on geoscience and remote sensing*, 26(1):65–74, 1988.

[76] Chein-I Chang and Su Wang. Constrained band selection for hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 44(6):1575–1585, 2006.

[77] Sylvia S Shen and Edward M Bassett III. Information-theory-based band selection and utility evaluation for reflective spectral systems. In *AeroSense 2002*, pages 18–29. International Society for Optics and Photonics, 2002.

[78] Xuesong Liu, Wei Xia, Bin Wang, and Liming Zhang. An approach based on constrained nonnegative matrix factorization to unmix hyperspectral data. *Geoscience and Remote Sensing, IEEE Transactions on*, 49(2):757–772, 2011.

[79] USGS digital spectral library. [online]. URL http://speclab.cr.usgs.gov/spectral-lib.html.

[80] Hisham Othman and Shen-En Qian. Noise reduction of hyperspectral imagery using hybrid spatial-spectral derivative-domain wavelet shrinkage. *IEEE Transactions on Geoscience and Remote Sensing*, 44(2):397–408, 2006.

[81] Damien Letexier and Salah Bourennane. Noise removal from hyperspectral images by multidimensional filtering. *IEEE Transactions on Geoscience and Remote Sensing*, 46(7):2061–2069, 2008.

[82] José M Bioucas-Dias, Antonio Plaza, Gustavo Camps-Valls, Paul Scheunders, Nasser M Nasrabadi, and Jocelyn Chanussot. Hyperspectral remote sensing data analysis and future challenges. *Geoscience and Remote Sensing Magazine, IEEE*, 1 (2):6–36, 2013.

[83] Laetitia Loncan, Luis B Almeida, José M Bioucas-Dias, Xavier Briottet, Jocelyn Chanussot, Nicolas Dobigeon, Sophie Fabre, Wenzhi Liao, Giorgio A Licciardi, Miguel Simões, et al. Hyperspectral pansharpening: a review. *arXiv preprint arXiv:1504.04531*, 2015.

[84] Zhao Chen, Hanye Pu, Bin Wang, and Geng-Ming Jiang. Fusion of hyperspectral and multispectral images: A novel framework based on generalization of pansharpening methods. *IEEE Geoscience and Remote Sensing Letters*, 11:1418–1422, 2014.

[85] Naoto Yokoya, Takehisa Yairi, and Akira Iwasaki. Coupled nonnegative matrix factorization unmixing for hyperspectral and multispectral data fusion. *Geoscience and Remote Sensing, IEEE Transactions on*, 50(2):528–537, 2012.

[86] Miguel Simões, José Bioucas-Dias, Luis B Almeida, and Jocelyn Chanussot. A convex formulation for hyperspectral image superresolution via subspace-based regularization. *Geoscience and Remote Sensing, IEEE Transactions on*, 53(6):3373–3388, 2015.

[87] Qi Wei, José Bioucas-Dias, Nicolas Dobigeon, and Jean-Yves Tourneret. Hyperspectral and multispectral image fusion based on a sparse representation. *Geoscience and Remote Sensing, IEEE Transactions on*, 53(7):3658–3668, 2015.

[88] Bruno Aiazzi, Stefano Baronti, and Massimo Selva. Improving component substitution pansharpening through multivariate regression of ms+ pan data. *Geoscience and Remote Sensing, IEEE Transactions on*, 45(10):3230–3239, 2007.

[89] Andrea Garzelli, Filippo Nencini, and Luca Capobianco. Optimal mmse pan sharpening of very high resolution multispectral images. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(1):228–236, 2008.

[90] Yifan Zhang and Mingyi He. Multi-spectral and hyperspectral image fusion using 3-d wavelet transform. *Journal of Electronics (China)*, 24(2):218–224, 2007.

[91] Daniel D Lee and H Sebastián Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[92] Lucien Wald, Thierry Ranchin, and Marc Mangolini. Fusion of satellite images of different spatial resolutions: assessing the quality of resulting images. *Photogrammetric engineering and remote sensing*, 63(6):691–699, 1997.

[93] Robert W Basedow, Dwayne C Carmer, and Mark E Anderson. Hydice system: Implementation and performance. In *SPIE's 1995 Symposium on OE/Aerospace Sensing and Dual Use Photonics*, pages 258–267. International Society for Optics and Photonics, 1995.

[94] Xiao Xiang Zhu and Richard Bamler. A sparse image fusion algorithm with application to pan-sharpening. *Geoscience and Remote Sensing, IEEE Transactions on*, 51(5):2827–2836, 2013.

[95] Andrea Garzelli, Filippo Nencini, and Luca Capobianco. Optimal mmse pan sharpening of very high resolution multispectral images. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(1):228–236, 2008.

[96] Hang Qian. Counting the floating point operations (FLOPs), Matlab central file exchange. *Iowa State Univ., Ames, IA, USA, Tech. Rep*, 50608, 2015. URL http://www.mathworks.com/matlabcentral/fileexchange/50608-counting-the-floating-point-operations--flops-.

[97] Lucana Santos Falcón. Hyperspectral image compression onboard next-generation satellites: implementation solutions on GPU and FPGAs. 2014.

[98] Barbara Penna, Tammam Tillo, Enrico Magli, and Gabriella Olmo. Progressive 3-d coding of hyperspectral images based on jpeg 2000. *IEEE Geoscience and remote sensing letters*, 3(1):125–129, 2006.

[99] Lena Chang, Ching-Min Cheng, and Ting-Chung Chen. An efficient adaptive klt for multispectral image compression. In *Image Analysis and Interpretation, 2000. Proceedings. 4th IEEE Southwest Symposium*, pages 252–255. IEEE, 2000.

[100] Pengwei Hao and Qingyun Shi. Reversible integer klt for progressive-to-lossless compression of multiple component images. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages I–633. IEEE, 2003.

[101] Qian Du, Wei Zhu, and James E Fowler. Anomaly-based hyperspectral image compression. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 2, pages II–974. IEEE, 2008.

[102] Qian Du and James E Fowler. Hyperspectral image compression using jpeg2000 and principal component analysis. *IEEE Geoscience and Remote Sensing Letters*, 4(2):201–205, 2007.

[103] David Taubman and Michael Marcellin. *JPEG2000 image compression fundamentals, standards and practice: image compression fundamentals, standards and practice*, volume 642. Springer Science & Business Media, 2012.

[104] Qian Du and James E Fowler. Hyperspectral image compression using jpeg2000 and principal component analysis. *IEEE Geoscience and Remote Sensing Letters*, 4(2):201–205, 2007.

[105] Ian Blanes, Joan Serra-Sagrista, Michael W Marcellin, and Joan Bartrina-Rapesta. Divide-and-conquer strategies for hyperspectral image processing: A review of their benefits and advantages. *IEEE Signal Processing Magazine*, 29(3):71–81, 2012.

[106] Consultative Committee for Space Data Systems (CCSDS). Blue books: Recommended standards. [online]. URL https://public.ccsds.org/Publications/BlueBooks.aspx.

[107] NASA Jet Propulsion Laboratory. Airbone visible infrared imaging spectrometer website. [online]. URL http://aviris.jpl.nasa.gov/aviris/index.html.

[108] U.S. Geological Survey and NASA. Earth observing 1, hyperion website. [online]. URL https://eo1.usgs.gov/sensors/hyperion.

[109] Jared A Herweg, John P Kerekes, Oliver Weatherbee, David Messinger, Jan van Aardt, Emmett Ientilucci, Zoran Ninkov, Jason Faulring, Nina Raqueño, and Joseph Meola. Spectir hyperspectral airborne rochester experiment data collection campaign. In *SPIE Defense, Security, and Sensing*, pages 839028–839028. International Society for Optics and Photonics, 2012.

[110] Dirk Borghys, Ingebjørg Kåsen, Véronique Achard, and Christiaan Perneel. Hyperspectral anomaly detection: Comparative evaluation in scenes with diverse complexity. *Journal of Electrical and Computer Engineering*, 2012:5, 2012.

[111] Stefania Matteoli, Marco Diani, and Giovanni Corsini. A tutorial overview of anomaly detection in hyperspectral images. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):5–28, 2010.

[112] Sergio Sánchez, Rui Ramalho, Leonel Sousa, and Antonio Plaza. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. *Journal of Real-Time Image Processing*, 10(3):469–483, 2015.

[113] Emanuele Torti, Giovanni Danese, Francesco Leporati, and Antonio Plaza. A hybrid cpu–GPU real-time hyperspectral unmixing chain. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(2):945–951, 2016.

[114] Luis Ignacio Jiménez and Antonio Plaza. Hypermix: An open-source tool for fast spectral unmixing on graphics processing units. *IEEE Geoscience and Remote Sensing Letters*, 12(9):1883–1887, 2015.

[115] Teresa G Cervero, Julián Caba, Sebastián López, Julio Daniel Dondo, Roberto Sarmiento, Fernando Rincón, and Juan López. A scalable and dynamically reconfigurable FPGA-based embedded system for real-time hyperspectral unmixing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8 (6):2894–2911, 2015.

[116] Teresa Cervero, Sebastián López, Gustavo Marrero Callicó, JF López, and Roberto Sarmiento. Scalable architectures for real-time hyperspectral unmixing. *Microelectronics Journal*, 45(10):1292–1303, 2014.

[117] Sebastián López, Gustavo M Callico, Anabella Medina, José F López, and Roberto Sarmiento. High-level FPGA-based implementation of a hyperspectral endmember extraction algorithm. In *Hyperspectral Image and Signal Processing (WHISPERS), 2012 4th Workshop on*, pages 1–4. IEEE, 2012.

[118] Carlos Gonzalez, Sebastián López, Daniel Mozos, and Roberto Sarmiento. A novel FPGA-based architecture for the estimation of the virtual dimensionality in remotely sensed hyperspectral images. *Journal of Real-Time Image Processing*, pages 1–12, 2015.

[119] Carlos González, Daniel Mozos, Sebastián López, and Roberto Sarmiento. FPGA implementation to estimate the number of endmembers in hyperspectral images. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–8. IEEE, 2015.

[120] Carlos Gonzalez, Sebastián López, Daniel Mozos, and Roberto Sarmiento. FPGA implementation of the hysime algorithm for the determination of the number of endmembers in hyperspectral data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2870–2883, 2015.

[121] José MP Nascimento, Mário Véstías, and Gabriel Martin. FPGA-based architecture for hyperspectral unmixing. In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, pages 1761–1764. IEEE, 2015.

[122] João Rosário, José MP Nascimento, and Mário Véstias. FPGA-based architecture for hyperspectral endmember extraction. In *SPIE Remote Sensing*, pages 924703–924703. International Society for Optics and Photonics, 2014.

[123] Carlos González, Daniel Mozos, Sebastián López, and Roberto Sarmiento. FPGA implementation to estimate the number of endmembers in hyperspectral images. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–8. IEEE, 2015.

[124] Sebastián López, Pablo Horstrand, Gustavo M Callico, José F López, and Roberto Sarmiento. A novel architecture for hyperspectral endmember extraction by means of the modified vertex component analysis (mvca) algorithm. *IEEE Journal of*

Selected Topics in Applied Earth Observations and Remote Sensing, 5(6):1837–1848, 2012.

[125] Antonio Plaza and Chein-I Chang. Clusters versus FPGA for parallel processing of hyperspectral imagery. *The International Journal of High Performance Computing Applications*, 22(4):366–385, 2008.

[126] Carlos Gonzalez, Daniel Mozos, Javier Resano, and Antonio Plaza. FPGA implementation of endmember extraction algorithms from hyperspectral imagery: pixel purity index versus n-findr. In *SPIE Remote Sensing*, pages 81830F–81830F. International Society for Optics and Photonics, 2011.

[127] Carlos González, Daniel Mozos, Javier Resano, and Antonio Plaza. FPGA implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis. *IEEE transactions on geoscience and remote sensing*, 50(2):374–388, 2012.

[128] Carlos González, Daniel Mozos, Javier Resano, and Antonio Plaza. FPGA design of the n-findr algorithm for spaceborne hyperspectral missions. In *Int. Conf. Space Technol.(ICST)*, pages 1–4, 2011.

[129] AP Cong Li, Lianru Gao, and B Zhang. FPGA implementation of a maximum volume algorithm for endmember extraction from hyperspectral imagery. In *IEEE Whispers Conf*, 2015.

[130] Cong Li, Lianru Gao, Antonio Plaza, and Bing Zhang. FPGA implementation of a maximum simplex volume algorithm for endmember extraction from remotely sensed hyperspectral images. *Journal of Real-Time Image Processing*, pages 1–14.

[131] Carlos Gonzalez, Javier Resano, Antonio Plaza, and Daniel Mozos. FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(1):248–261, 2012.

[132] Javier Morales, Nelson Medero, Nayda G Santiago, and Julio Sosa. Hardware implementation of image space reconstruction algorithm using FPGAs. In *Circuits and Systems, 2006. MWSCAS'06. 49th IEEE International Midwest Symposium on*, volume 1, pages 433–436. IEEE, 2006.

[133] Javier Morales, Nayda G Santiago, and Alejandro Fernandez. An FPGA implementation of image space reconstruction algorithm for hyperspectral imaging analysis.

In *Defense and Security Symposium*, pages 65651V–65651V. International Society for Optics and Photonics, 2007.

[134] Hongtao Du and Hairong Qi. A reconfigurable FPGA system for parallel independent component analysis. *EURASIP Journal on Embedded Systems*, 2006(1): 023025, 2006.

[135] José MP Nascimento, José M Bioucas-Dias, José M RodrJosíguez Alves, Vítor Silva, and Antonio Plaza. Parallel hyperspectral unmixing on GPUs. *IEEE Geoscience and Remote Sensing Letters*, 11(3):666–670, 2014.

[136] José M RodrJosíguez Alves, José MP Nascimento, José M Bioucas-Dias, Antonio Plaza, and Vitor Silva. Parallel sparse unmixing of hyperspectral data. In *Geoscience and Remote Sensing Symposium (IGARSS), 2013 IEEE International*, pages 1446–1449. IEEE, 2013.

[137] Sergio Sánchez, Gabriel Martín, Antonio Plaza, and Chein-I Chang. GPU implementation of fully constrained linear spectral unmixing for remotely sensed hyperspectral data exploitation. In *SPIE Optical Engineering+ Applications*, pages 78100G–78100G. International Society for Optics and Photonics, 2010.

[138] Sergio Sánchez and Antonio Plaza. Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs. *Journal of real-time image processing*, 9(3):397–405, 2014.

[139] Alexander Agathos, Jun Li, Dana Petcu, and Antonio Plaza. Multi-GPU implementation of the minimum volume simplex analysis algorithm for hyperspectral unmixing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(6):2281–2296, 2014.

[140] Sergio Sánchez, Abel Paz, Gabriel Martín, and Antonio Plaza. Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units. *Concurrency and Computation: Practice and Experience*, 23(13):1538–1557, 2011.

[141] Jorge Sevilla, Sergio Bernabé, and Antonio Plaza. Unmixing-based content retrieval system for remotely sensed hyperspectral imagery on GPUs. *The Journal of Supercomputing*, 70(2):588–599, 2014.

[142] Xianyun Wu, Bormin Huang, Antonio Plaza, Yunsong Li, and Chengke Wu. Real-time implementation of the pixel purity index algorithm for endmember identification on GPUs. *IEEE Geoscience and Remote Sensing Letters*, 11(5):955–959, 2014.

[143] A Barberis, Giovanni Danese, Francesco Leporati, A Plaza, and Emanuele Torti. Real-time implementation of the vertex component analysis algorithm on GPUs. *IEEE Geoscience and Remote Sensing Letters*, 10(2):251–255, 2013.

[144] Emilie Chouzenoux, Maxime Legendre, Saïd Moussaoui, and Jérôme Idier. Fast constrained least squares spectral unmixing using primal-dual interior-point optimization. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(1):59–69, 2014.

[145] Zebin Wu, Shun Ye, Jianjun Liu, Le Sun, and Zhihui Wei. Sparse non-negative matrix factorization on GPUs for hyperspectral unmixing. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(8):3640–3649, 2014.

[146] Antonio Plaza, Gabriel Martín, Javier Plaza, Maciel Zortea, and Sergio Sánchez. Recent developments in endmember extraction and spectral unmixing. In *Optical Remote Sensing*, pages 235–267. Springer, 2011.

[147] Sergio Sánchez, Abel Paz, and Antonio Plaza. Real-time spectral unmixing using iterative error analysis on commodity graphics processing units. In *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, pages 1767–1770. IEEE, 2011.

[148] Zebin Wu, Shun Ye, Jie Wei, Zhihui Wei, Le Sun, and Jianjun Liu. Fast endmember extraction for massive hyperspectral sensor data on GPUs. *International Journal of Distributed Sensor Networks*, 2013.

[149] Jaime Delgado, Gabriel Martín, Javier Plaza, Luis Ignacio Jiménez, and Antonio Plaza. Fast spatial preprocessing for spectral unmixing of hyperspectral data on graphics processing units. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(2):952–961, 2016.

[150] Eysteinn Már Sigurdsson, Antonio Plaza, and Jón Atli Benediktsson. GPU implementation of iterative-constrained endmember extraction from remotely sensed hyperspectral images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2939–2949, 2015.

[151] José M Rodriguez Alves, José MP Nascimento, José M Bioucas-Dias, Vítor Silva, and Antonio Plaza. Parallel implementation of vertex component analysis for hyperspectral endmember extraction. In *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pages 4078–4081. IEEE, 2012.

[152] Luis Ignacio Jiménez, Gabriel Martín, Sergio Sánchez, Carlos García, Sergio Bernabé, Javier Plaza, and Antonio Plaza. GPU implementation of spatial–spectral preprocessing for hyperspectral unmixing. *IEEE Geoscience and Remote Sensing Letters*, 13(11):1671–1675, 2016.

[153] Jaime Delgado, Gabriel Martin, Javier Plaza, Luis Ignacio Jimenez, and Antonio Plaza. GPU implementation of spatial preprocessing for spectral unmixing of hyperspectral data. In *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, pages 5043–5046. IEEE, 2015.

[154] José M RodrJosíguez Alves, José MP Nascimento, Antonio Plaza, Sergio Sánchez, José M Bioucas-Dias, and Vitor Silva. Vertex component analysis GPU-based implementation for hyperspectral unmixing. In *Hyperspectral Image and Signal Processing (WHISPERS), 2012 4th Workshop on*, pages 1–4. IEEE, 2012.

[155] Zebin Wu, Jianjun Liu, Shun Ye, Le Sun, and Zhihui Wei. Optimization of minimum volume constrained hyperspectral image unmixing on CPU–GPU heterogeneous platform. *Journal of Real-Time Image Processing*, pages 1–13, 2014.

[156] Sergio Bernabé, Antonio Plaza, Sebastián López, and Roberto Sarmiento. Parallel implementation of a hyperspectral unmixing chain: Graphic processing units versus multi-core processors. In *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pages 3463–3466. IEEE, 2012.

[157] Antonio Plaza, Javier Plaza, and Sergio Sánchez. Parallel implementation of endmember extraction algorithms using nvidia graphical processing units. In *Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009*, volume 5, pages V–208. IEEE, 2009.

[158] Jianwei Gao, Lianru Gao, Xu Sun, Yuanfeng Wu, and Bing Zhang. GPU implementation of ant colony optimization algorithm for endmember extraction from hyperspectral image. In *Hyperspectral Image and Signal Processing (WHISPERS), 2012 4th Workshop on*, pages 1–4. IEEE, 2012.

[159] David Valencia and Antonio Plaza. FPGA-based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm. In *International Conference on Computational Science*, pages 888–891. Springer, 2006.

[160] Thomas W Fry and Scott Hauck. Hyperspectral image compression on reconfigurable platforms. In *Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on*, pages 251–260. IEEE, 2002.

[161] Nazeeh Aranki, Didier Keymeulen, Alireza Bakhshi, and Matthew Klimesh. Hardware implementation of lossless adaptive and scalable hyperspectral data compression for space. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, pages 315–322. IEEE, 2009.

[162] Guoxia Yu, Tanya Vladimirova, and Martin N Sweeting. FPGA-based on-board multi/hyperspectral image compression system. In *Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009*, volume 5, pages V–212. IEEE, 2009.

[163] Didier Keymeulen, Nazeeh Aranki, Ben Hopson, Aaron Kiely, Matthew Klimesh, and Khaled Benkrid. GPU lossless hyperspectral data compression system for space applications. In *Aerospace Conference, 2012 IEEE*, pages 1–9. IEEE, 2012.

[164] Antonio Plaza. Towards real-time compression of hyperspectral images using Virtex-II FPGAs. In *European Conference on Parallel Processing*, pages 248–257. Springer, 2007.

[165] Carlos González, Javier Resano, Daniel Mozos, Antonio Plaza, and David Valencia. FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis. *EURASIP Journal on Advances in Signal Processing*, 2010(1):969806, 2010.

[166] Pasquale Corsonello, Stefania Perri, Paolo Zicari, and Giuseppe Cocorullo. Microprocessor-based FPGA implementation of SPIHT image compression subsystems. *Microprocessors and Microsystems*, 29(6):299–305, 2005.

[167] Ben Hopson, Khaled Benkrid, Didier Keymeulen, and Nazeeh Aranki. Real-time CCSDS lossless adaptive hyperspectral image compression on parallel GPGPU & multicore processor systems. In *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, pages 107–114. IEEE, 2012.

[168] Lucana Santos, Luis Berrojo, Javier Moreno, José F López, and Roberto Sarmiento. Multispectral and hyperspectral lossless compressor for space applications (HyLoC): A low-complexity FPGA implementation of the CCSDS 123 standard. *IEEE Journal*

of Selected Topics in Applied Earth Observations and Remote Sensing, 9(2):757–770, 2016.

[169] Lucana Santos, Enrico Magli, Raffaele Vitulli, José F López, and Roberto Sarmiento. Highly-parallel GPU architecture for lossy hyperspectral image compression. *IEEE journal of selected topics in applied earth observations and remote sensing*, 6(2):670–681, 2013.

[170] Lucana Santos, José F López, Roberto Sarmiento, and Raffaele Vitulli. FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool. In *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, pages 107–114. IEEE, 2013.

[171] Andrew G Schmidt and Matthew French. Fast lossless image compression with radiation hardening by hardware/software co-design on platform FPGAs. In *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, pages 103–106. IEEE, 2013.

[172] Vaclav Simek and Ram Rakesh Asn. GPU acceleration of 2d-dwt image compression in matlab with CUDA. In *Computer Modeling and Simulation, 2008. EMS'08. Second UKSIM European Symposium on*, pages 274–277. IEEE, 2008.

[173] Tom L Woolston, Gail E Bingham, Niel S Holt, and Glen Wada. An FPGA-based demonstration hyperspectral image compression system. In *SPIE Defense and Security Symposium*, pages 69661J–69661J. International Society for Optics and Photonics, 2008.

[174] Milosz Ciznicki, Krzysztof Kurowski, and Antonio Plaza. GPU implementation of JPEG2000 for hyperspectral image compression. In *SPIE remote sensing*, pages 81830H–81830H. International Society for Optics and Photonics, 2011.

[175] Li Li, Gang Zhou, Björn Fiethe, Harald Michalik, and Björn Osterloh. Efficient implementation of the ccsds 122.0-b-1 compression standard on a space-qualified field programmable gate array. *Journal of Applied Remote Sensing*, 7(1):074595–074595, 2013.

[176] Lucana Santos, Ian Blanes, Aday García, Joan Serra-Sagristà, José López, and Roberto Sarmiento. On the hardware implementation of the arithmetic elements of the pairwise orthogonal transform. *Journal of Applied Remote Sensing*, 9(1): 097496–097496, 2015.

[177] Maciel Zortea and Antonio Plaza. Spatial preprocessing for endmember extraction. *IEEE Transactions on Geoscience and Remote Sensing*, 47(8):2679–2693, 2009.

[178] Carlos Gonzalez, Sergio Sánchez, Abel Paz, Javier Resano, Daniel Mozos, and Antonio Plaza. Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. *INTEGRATION, the VLSI journal*, 46(2):89–103, 2013.

[179] Antonio Plaza, Javier Plaza, Abel Paz, and Sergio Sánchez. Parallel hyperspectral image and signal processing [applications corner]. *IEEE Signal Processing Magazine*, 28(3):119–126, 2011.

[180] Antonio Plaza, Sergio Sánchez, Abel Paz, and Javier Plaza. GPUs versus FPGAs for onboard payload compression of remotely sensed hyperspectral data.

[181] CUDA Toolkit Documentation. [online]. URL http://docs.nvidia.com/cuda/cublas/.

[182] Gabriel Martín and Antonio Plaza. Region-based spatial preprocessing for endmember extraction and spectral unmixing. *Geoscience and Remote Sensing Letters, IEEE*, 8(4):745–749, 2011.

[183] Jaime Ortegón Aguilar, Alejandro Castillo Atoche, Roberto Carrasco Alvarez, Javier Vázquez Castillo, Ivan Villalón-Turrubiates, and Omar Pérez-Martínez. Enhancement and edge-preserving denoising: An OpenCL-based approach for remote sensing imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(12):5482–5492, 2016.

[184] Fabian Tschopp. Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems. *arXiv preprint arXiv:1509.03371*, 2015.

[185] Michel Steuwer, Toomas Remmelg, and Christophe Dubach. Matrix multiplication beyond auto-tuning: rewrite-based gpu code generation. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 15. ACM, 2016.

[186] Next generation airborne visible/infrared imaging spectrometer (NG-AVIRIS). [online]. URL https://aviris-ng.jpl.nasa.gov/specifications.html.

[187] Ice Comprehensive RPC Framework. [online]. URL https://zeroc.com/products/ice.