Noname manuscript No.
(will be inserted by the editor)

Concurrent execution of lossy compression and anomaly detection of hyperspectral images on FPGA devices

Julián Caba · Jesús Barba · María Díaz · José Luis Mira · Sebastián López · Juan Carlos López

Received: date / Accepted: date

Abstract Hyperspectral sensors capture wide range of spectral data, making them crucial for Earth observation applications, but this fact poses significant challenges for embedded systems with limited resources. Nevertheless, most studies only perform one application at the same time, so multi-applications in the same device are not considered since high-performance and low hardware resources are limited. In this sense, this paper presents a hardware-friendly algorithm for concurrently execution of anomaly detection and lossy compression for hyperspectral imaging. The proposed algorithm reuses a hardware platform to perform both tasks in parallel, offering a validated hardware architecture designed for deployment on a cost-optimized FPGA device. The experimental results show that our hardware component can process hyperspectral images with a resolution of 825x1024 pixels and 160 bands in 0.53 seconds (486MB/s), with a power consumption of 1.08 watts (399MB/W).

Keywords hyperspectral imaging · lossy compression · anomaly detection · FPGA · adaptive computing

1 Introduction

Hyperspectral sensors capture rich spectral data across the electromagnetic spectrum, making them ideal for Earth observation applications like precision agriculture, geological mapping, and mineral exploration. However, the large data volume poses challenges for embedded systems with limited resources, exacerbated by advancements in hyperspectral sensor resolution [1,2].

Traditionally, only one application is used to process the information captured by hyperspectral sensors, tailored to the device platform on which the application is running. On this basis, the data captured by hyperspectral sensors is typically not processed on-board due to the limited computing performance and power capacity available, so low-power and cost-optimized devices are used for data sensing, but they lack high-performance capabilities [3]. In aerial platforms, such as Unmanned Aerial Vehicles (UAVs), images are generally stored onboard and processed after the flight mission is completed [4]. However, transferring large volumes of data creates a bottleneck in the downlink systems, which can impact overall performance and increase the energy consumption budget for transmission [5].

Recent studies suggest edge computing solutions that facilitate on-board image processing, minimizing the need for downlink bandwidth. Most hyperspectral imaging (HSI) applications require processing vast amounts of data using complex algorithms, leading to a significant computational burden. To overcome this challenge, massive parallel processing architectures, such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs), are often employed due to their high performance and the accuracy of the results they produce. However, HSI applications, including Artificial Intelligence (AI) models, are highly computationally demanding, requiring a substantial number of operations per second [6]. Many studies propose deploying FPGA- or GPU-based high-performance solutions at the edge [7–9]. However, these solutions are generally

J. Caba, J. Barba, J.L. Mira and J.C. López
University of Castilla-La Mancha (UCLM), 13071 Ciudad Real, Spain E-mail: julian.caba@uclm.es

M. Diaz
Plataforma Oceánica de Canarias (PLOCAN), 35214 Telde, Las Palmas de Gran Canaria, Spain

S. López
Institute of Applied Microelectronics (IUMA), 35017 Las Palmas de Gran Canaria, Spain

unsuitable for mobile embedded systems, such as UAVs, due to constraints related to power consumption, heat dissipation, and limited energy budgets. Instead, low-power and cost-optimized devices are often preferred for on-board data processing, but these devices face limitations in computing performance and power capacity. In this sense, achieving high performance while ensuring compatibility with available hardware resources requires significant engineering efforts. This process involves optimizing hyperspectral data processing algorithms and developing hardware accelerators to efficiently manage computational workloads.

This paper introduces a hardware-friendly algorithm for anomaly detection and lossy compression of hyperspectral data, utilizing a single FPGA platform for both tasks. It provides a low-power, cost-optimized solution for real-time processing in constrained environments. The FPGA-based architecture and experimental results are analyzed, comparing performance and resource utilization with state-of-the-art solutions.

2 Algorithm Description

In previous works, we introduced two FPGA-based algorithms: HLCA for lossy compression [10] and LbL-FAD for anomaly detection [11], each evaluated separately [12] [13] and designed for pushbroom/whiskbroom sensors; both algorithms follow a hardware-friendly approach, enabling parallelization on GPUs and FPGAs. This work takes a further step to demonstrate that the proposed methodology can be efficiently employed for the aforementioned HSI applications using a single architecture with the same set of core operations.

2.1 General Notation

Before starting with the description of the proposed algorithm, it is necessary to define some variables and terminology used throughout the remainder of this manuscript. A hyperspectral image, **HI**, is a sequence of nr hyperspectral frames or lines of pixels, $\mathbf{F}_i$, comprised by nc pixels with nb spectral bands. Pixels within **HI** are grouped in blocks of BS pixels, $\mathbf{M}_k$, being BS usually equal to nc, or a multiple of it, and k spans from 1 to $\frac{nr \cdot nc}{BS}$ hyperspectral frames. $\hat{\boldsymbol{\mu}}$ is the average pixel or *centroid* of each $\mathbf{M}_k$ block. $\mathbf{C} = \{\mathbf{c}_j, j = 1, ..., BS\}$ represents the centralized version of the input image block, $\mathbf{M}_k$. $\mathbf{E}_k = \{\mathbf{e}_n, n = 1, ..., p\}$ saves the p most different hyperspectral pixels extracted from each $\mathbf{M}_k$ block, whilst $\mathbf{B^*} = \{\mathbf{E}_k, k = 1, ..., n_f\}$ retains the subset of selected pixels $\mathbf{E}_k = \{\mathbf{e}_n, n = 1, ..., p\}$ within the first captured n_f $\mathbf{M}_k$. $\mathbf{V}_k = \{\mathbf{v}_n, n = 1, ..., p\}$ comprises p vectors of BS elements where each $\mathbf{v}_n$ vector corresponds to the projection of the BS pixels within $\mathbf{M}_k$ onto the corresponding n extracted pixel, $\mathbf{e}_n$. $\mathbf{Q} = \{\mathbf{q}_n, n = 1, ..., p\}$ and $\mathbf{U} = \{\mathbf{u}_n, n = 1, ..., p\}$ save p pixels of nb bands that are orthogonal among them.

Algorithm 1 Set of core operations

Inputs: $\mathbf{M}_k = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{BS}], \alpha$
Outputs: $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p]$ {Characteristic pixels}; $\hat{\boldsymbol{\mu}}$ {Average Pixel}; $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_p]$ {Orthogonal vectors}; $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_p]$ {Orthonormal vectors}; τ {Threshold}
Algorithm:
1: Average pixel: $\hat{\boldsymbol{\mu}}$;
2: Centralization: $\mathbf{C} = \mathbf{M}_k - \hat{\boldsymbol{\mu}}$;
3: **while** exit = 0 **do**
4: **for** $j = 1$ **to** BS **do**
5: Brightness: $\mathbf{b}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$;
6: **end for**
7: Maximum Brightness: $j_{max} = \text{argmax}(\mathbf{b}_j)$;
8: **if** $\frac{b_{j\max}}{(\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})} \cdot 100 < \alpha$ **then**
9: Stop Condition: exit = 1
10: **else**
11: Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{j_{max}}$;
12: $\mathbf{q}_n = \mathbf{c}_{j_{max}}$;
13: $\mathbf{u}_n = \mathbf{q}_n / b_{j_{max}}$;
14: Projection: $\mathbf{v}_n = \mathbf{u}'_n \cdot \mathbf{C}$;
15: Subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
16: $\tau = b_{j\max}$
17: **end if**
18: **end while**

2.2 Set of core operations

The HLCA and LbL-FAD algorithms use orthogonal subspace projections: HLCA maps the spatial domain into a transformation domain [2], while LbL-FAD selects distinct pixels from a background. Both coordinate the core operations implementing the Gram-Schmidt orthogonalization process [14]. The set of core operations, detailed in Algorithm 1, processes hyperspectral data in blocks of BS pixels, denoted as $\mathbf{M}_k$.

Average pixel. The first characteristic pixel, $\mathbf{e}_1$, is chosen based on the highest deviation from the average pixel, $\hat{\boldsymbol{\mu}}$, which is computed for the input block $\mathbf{M}_k$ (line 1, Algorithm 1).

Centralization. The image block $\mathbf{M}_k$ is then centralized by subtracting $\hat{\boldsymbol{\mu}}$ from each pixel, producing the centralized image $\mathbf{C}$ (line 2, Algorithm 1). Next, the p most representative pixels are extracted sequentially (lines 3–18, Algorithm 1). Using Gram-Schmidt orthogonalization, each pixel's projection onto the selected pixels, $\mathbf{e}_n$, is computed. New characteristic pixels are chosen based on the largest orthogonal projections, ensuring unique spectral information is captured. This iterative process continues until a predefined stop condition is met (line 8, Algorithm 1).

Brightness. Pixels are selected based on the highest self-dot product (lines 4–6, Algorithm 1), referred to as brightness in this document. The selected pixels, $\mathbf{e}_n$, are chosen from $\mathbf{M}_k$ based on maximum brightness in $\mathbf{C}$ (line 11, Algorithm 1). Their orthogonal projections, $\mathbf{q}_n$, and normalized versions, $\mathbf{u}_n$, are then computed (lines 12–13, Algorithm 1).

Projection. $\mathbf{C}$ are projected onto $\mathbf{u}_n$, generating the projection vector $\mathbf{v}_n$ (line 14, Algorithm 1).

Subtraction. The stored spectral information in $\mathbf{v}_n$ is subtracted from $\mathbf{C}$ (line 15, Algorithm 1), ensuring $\mathbf{C}$ retains only spectral data orthogonal that is not represented by previously selected spectra.

Regarding the estimation of the number of endmembers, the proposed methodology employs a *stop condition*. Each time a pixel e_n is selected, the spectral information that cannot be represented by the already extracted pixels remains in the image matrix $\mathbf{C}$. Consequently, when the image is reconstructed using the p selected e_n pixels, a small part of the spectral information is lost, which corresponds to the remaining information contained in $\mathbf{C}$. In this sense, the maximum brightness, b_{jmax}, after selecting the e_p, may be indicative of the spectral losses introduced by the unmixing process and, therefore, could be used to determine p. In this context, the endmember extraction process terminates when the loss, expressed as a percentage, is lower than an input parameter α, which represents the percentage of spectral information that will be considered as noise. This stop condition is implemented in line 8 of Algorithm 1, where $(r_{jmax} - \hat{\mu})$ represents the initial value of r_{jmax} in $\mathbf{C}$.

Algorithm 2 The HADeLOC algorithm.

Inputs: $\mathbf{HI} = [\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_{\frac{nr \cdot nc}{BS}}], n_f, \alpha$
Outputs: See each stage
Algorithm:
Stage 1:
Outputs: $\hat{\boldsymbol{\mu}}_{\mathrm{k}}$ {Average Pixel}; $\mathbf{E}_{\mathrm{k}} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p]$ {Characteristic pixels}; $\mathbf{V}_{\mathrm{k}} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_p]$ {Projections}
1: **for** k = 1 **to** n_f **do**
2: $\mathbf{E}_{\mathrm{k}} =$ **Algorithm 1**$(\mathbf{M}_{\mathrm{k}}, \alpha)$
3: $\mathbf{B^*} = [\mathbf{B^*}, \mathbf{E}_{\mathrm{k}}]$;
4: **end for**
Stage 2:
Outputs: $\hat{\boldsymbol{\mu}}_{\mathrm{B}}$ {Average Pixel}; $\mathbf{E}_{\mathrm{B}} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_b]$ {Characteristic pixels}
5: $[\hat{\boldsymbol{\mu}}_b, \mathbf{Q}, \mathbf{U}, \tau] =$ **Algorithm 1**$(\mathbf{B^*}, \alpha)$
Stage 3: Applied to each new received frame, $\mathbf{M}_k$, $k > n_f$
Outputs: $\mathbf{V}_{\mathrm{k}} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_p]$ {Projections}
6: **for** j = 1 **to** BS **do**
7: Centralization: $\mathbf{c}_j = \mathbf{r}_j - \hat{\mu}_b$
8: **for** n = 1 **to** p **do**
9: Projection: $v = \mathbf{U}'_n \cdot \mathbf{c}_j$
10: Subtraction: $\mathbf{c}_j = \mathbf{c}_j - \mathbf{Q}_n \cdot v$
11: **end for**
Stage 4:
Outputs: $\mathbf{AD} = [\mathbf{x}_{11}, \mathbf{x}_{12}, ..., \mathbf{x}_{kj}]$ {Anomaly map}
12: Brightness AD: $\mathbf{d}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$
13: **if** $\mathbf{d}_j \leq 1.5 \cdot \tau$ **then** $\mathbf{x}_{kj} = 0$
14: **else**$\mathbf{x}_{kj} = 1$
15: **end if**
Stage 5:
Outputs: $\mathbf{E}_{\mathrm{k}} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_a]$ {Characteristic pixels}; $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_a]$ {Projections};
16: **if** $\mathbf{x}_{kj} = 1$ **then**
17: **while** exit = 0 **do**
18: **for** $j = 1$ **to** BS **do**
19: Brightness: $\mathbf{b}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$;
20: **end for**
21: Maximum Brightness: $j_{max} = \mathrm{argmax}(\mathbf{b}_j)$;
22: **if** $\frac{b_{j\max}}{(\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})} \cdot 100 < \alpha$ **then**
23: Stop Condition: exit = 1
24: **else**
25: Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{jmax}$;
26: $\mathbf{q}_n = \mathbf{c}_{jmax}$;
27: $\mathbf{u}_n = \mathbf{q}_n / b_{jmax}$;
28: Projection: $\mathbf{v}_n = \mathbf{u}'_n \cdot \mathbf{C}$;
29: Subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
30: **end if**
31: **end while**
32: **end if**
33: **end for**

2.3 The HADeLoC algorithm

A shared mathematical framework enables lossy compression and anomaly detection in HSIs, utilizing the core operations previously described. This section explores the feasibility of running both processes simultaneously within the proposed methodology.

For hyperspectral lossy compression, the most distinct pixels, $\mathbf{E}$, and their projection vectors, $\mathbf{V}$, help decorrelate image blocks, $\mathbf{M}_k$, facilitating compression. In anomaly detection, $\mathbf{E}$, along with orthogonal counterparts $\mathbf{Q}$ and $\mathbf{U}$, define the background subspace to detect spectral anomalies. The process begins by extracting the most representative pixels from each block.

To do this, the core operations are applied to the first n_f blocks (Stage 1, Algorithm 2). The key difference between HLCA and LbL-FAD lies in the number of iterations needed to extract the p reference vectors. HLCA predetermines p during initialization, while LbL-FAD dynamically evaluates a stop condition at each iteration to ensure adequate representation of the processed block. Consequently, the core operations run n times, with outputs serving both applications, and whose n is dictated by their combined requirements.

For anomaly detection (Stage 2, Algorithm 2), the background must be estimated. The core operations are applied to the background using $\mathbf{B^*}{=}\mathbf{E}_k (k \leq n_f)$ as the input matrix $\mathbf{M}_k$. This yields the background's average pixel $\hat{\boldsymbol{\mu}}$ and the orthogonal vectors $\mathbf{Q}$ and $\mathbf{U}$, which are later used to detect anomalous spectra.

At this stage, the outputs of the core operations serve both applications, enabling their simultaneous execution. However, the complexity increases beyond this point. For anomaly detection, once the background pattern is modeled in Stage 2, the detection process continues in Stages 3 and 4 to generate the anomaly map. The

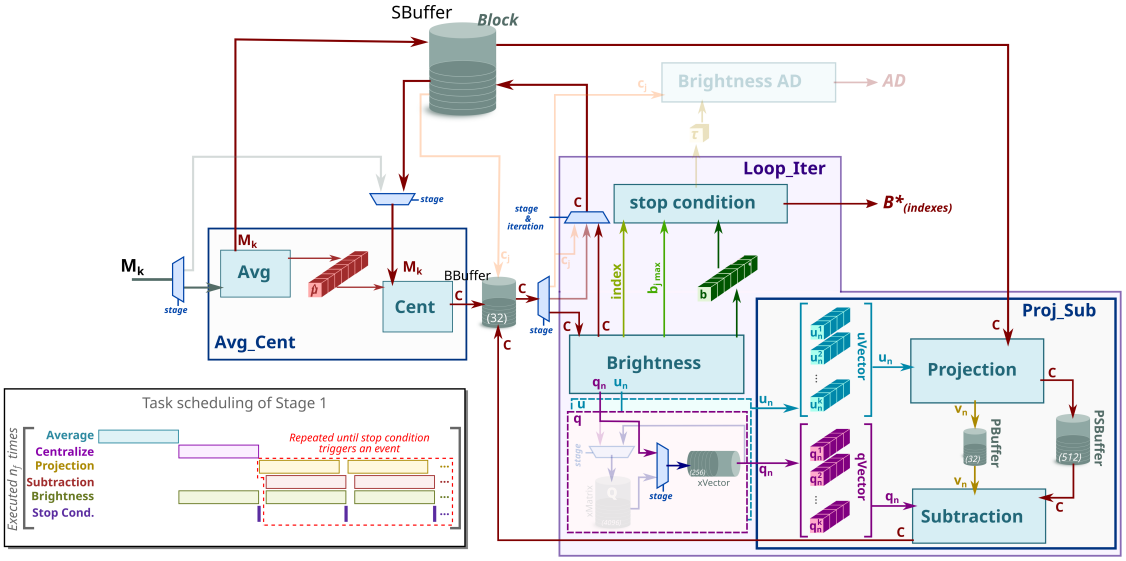


Fig. 1 Line-by-line extraction of the background reference spectra (Stage 1).

average pixel $\hat{\boldsymbol{\mu}}$ is used to centralize $\mathbf{M}_k$, eliminating the need to recalculate it. Then, the spectral information in $\mathbf{Q}$ and $\mathbf{U}$ is subtracted from $\mathbf{M}_k$ by repeating the projection and subtraction operations p times—matching the number of vectors in $\mathbf{Q}$ and $\mathbf{U}$. Notably, brightness computation is unnecessary here, as characteristic pixels were selected in the prior stage.

Meanwhile, the compression process diverges from its original version. If an input block $\mathbf{M}_k$ contains no anomalies, it is well represented using projection vectors $\mathbf{V}_k$ (Stage 3, Algorithm 2) and characteristic pixels $\mathbf{E}_B$ (Stage 2, Algorithm 2). In contrast, additional information must be extracted from blocks with anomalies by executing the brightness, projection, and subtraction operations p times. Thus, p is determined by a stop condition that ensures sufficient spectral representation for these blocks (Stage 5, Algorithm 2).

3 Hardware Architecture

The HADeLoC algorithm has been implemented on an FPGA-based computing platform using High-Level Synthesis (HLS) tools, which leverage High-Level Languages (HLLs) like C or C++ to define independent hardware accelerators (HWacc) for each computational stage. HLS technology reduces development time and complexity by translating high-level descriptions into lower-level RTL (Register Transfer Level) models. This approach streamlines development by modifying core operations to merge architectures that enable concurrent compression and anomaly detection.

Each core operation and anomaly detection function is modeled as an independent HWacc to meet requirements of HADeLoC. To coordinate these accelerators, a Finite State Machine (FSM) in VHDL orchestrates execution by activating selectors (blue trapezoids in Figures 1–4) and triggering HWacc at the right time. Synchronization is ensured via the standard handshake protocol provided by Vitis HLS tool, with intermediate memory buffers preventing stall scenarios.

3.1 Stage 1: Background reference spectra extraction

This stage corresponds to Stage 1 of the anomaly detector algorithm, where the set of core operations is applied to the first n_f image blocks, $\mathbf{M}_k$, to extract the most characteristic pixels, $\mathbf{E}$. It also aligns with the spectral transformation performed by the HLCA compressor, differing mainly in the number of pixels, p, to extract. In this proposal, a *stop condition* is introduced, based on the brightness ratio between the selected pixel in $\mathbf{M}_k$ and $\mathbf{C}$, which is evaluated iteratively.

For the sake of clarity, Figure 1 illustrates the process. First, the *Avg* module computes the average pixel ($\hat{\boldsymbol{\mu}}$) of $\mathbf{M}_k$ and stores the image block in internal memory. The *Cent* module then centralizes the block by subtracting $\hat{\boldsymbol{\mu}}$. A small FIFO buffer enables concurrent execution of *centralization* and *brightness* operations, reducing internal storage needs. The *brightness* module also stores the brightest pixel of the block, allowing the *stop condition* module to determine whether additional characteristic pixels are needed. At least two executions of the *brightness* module and one iteration of *projection* and *subtraction* are required, ensuring meaningful ratio comparisons in the *stop condition* module.

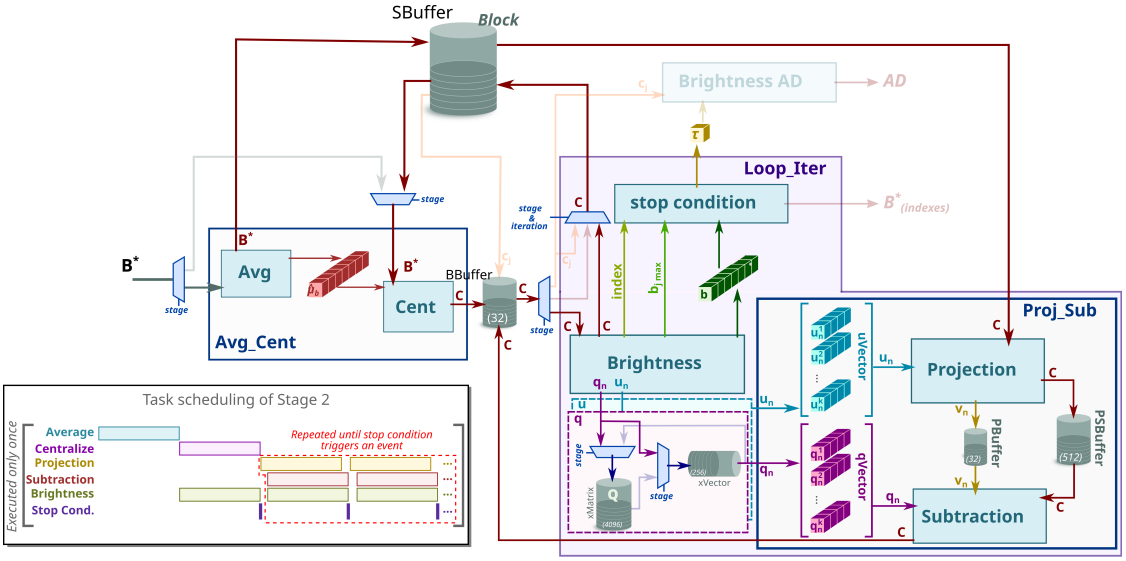


Fig. 2 Overall background subspace estimation (Stage 2).

Furthermore, the *brightness* computes the orthogonal projection vectors $\boldsymbol{q_n}$ and $\boldsymbol{u_n}$ using the brightest pixel (lines 12 and 13, Algorithm 1) and stores them in separate FIFOs (*qVector* and *uVector*, Figure 1). The *Projection* and *Subtraction* modules operate in parallel to compute projection vectors, $\mathbf{V}$, and remove spectral information from the image block. The *Projection* module reads the block from the *SBuffer* (line 14, Algorithm 1), while the *Subtraction* module processes the same pixel and its projection (*PBuffer*) to compute the subtracted image (line 15, Algorithm 2). Both modules read $\boldsymbol{u_n}$ and $\boldsymbol{q_n}$ using a custom array partition to enhance parallel execution.

The output at this stage resembles the HLCA compressor's results, comprising $\hat{\boldsymbol{\mu}}$, selected pixels $\mathbf{E}$, and projection vectors $\mathbf{V}$. However, p remains undetermined, depending on the *stop condition*. The decompression process receives a bit array and identifies vectors per iteration, n. To manage this, an extra bit is placed between each pair of vectors, $\mathbf{e}_n$ and $\mathbf{v}_n$; 0 denotes a new iteration, adding new vectors, while 1 marks the end of processing for $\mathbf{M}_k$ and the transition to $\mathbf{M}_{k+1}$. This synchronization barrier is also used in other data packing strategies.

3.2 Stage 2. Background subspace estimation

During this stage, the orthogonal subspace vectors $\mathbf{Q}$ and $\mathbf{U}$, modeling the background distribution, are estimated. The core operations are executed similarly to Stage 1 (Figure 2), but with the selected pixels extracted in Stage 1 as the input pixel block $\mathbf{B^*}$. Unlike Stage 1, this stage processes a single block instead of $\boldsymbol{n_f}$ blocks. However, the number of input vectors ($\mathbf{E}$) varies depending on the image. While software can handle variable block sizes easily, hardware implementation requires standardization to optimize resources. To address this, the block size is fixed at 1024 hyperspectral pixels, extending the background block by sequentially replicating its data.

Figure 2 illustrates the modified datapath. Internal memories *uMatrix* and *qMatrix* store the orthogonal projection vectors $\boldsymbol{u_n}$ and $\boldsymbol{q_n}$ extracted from the background block, which are then used by the *Projection* and *Subtraction* modules. Additionally, the *Stop condition* module stores the maximum brightness τ of $\boldsymbol{B^*}$ in a register without producing an output.

Since the data being processed is not part of the image intended for compression and reconstruction on Earth's surface, only the selected pixels, $\mathbf{E}$, and the average pixel, $\hat{\boldsymbol{\mu}}$, are transmitted in the output stream. These pixels are necessary for decompressing the blocks without anomalies, $\mathbf{M}_k$, as outlined in the subsequent algorithm stages.

3.3 Stage 3-4. Computation of the orthogonal subspace and anomaly detection

The following two stages focus on the detection of anomalous spectra, processed on the newly received image blocks, $\mathbf{M}_k$, similarly to the LbL-FAD algorithm. However, the compression process diverges from the original HLCA compressor. In the original HLCA, the most characteristic pixels within $\mathbf{M}_k$ were selected based on the highest brightness in matrix $\mathbf{C}$ during each iteration. Since the vectors $\mathbf{Q}$ and $\mathbf{U}$ from Stage 2 model

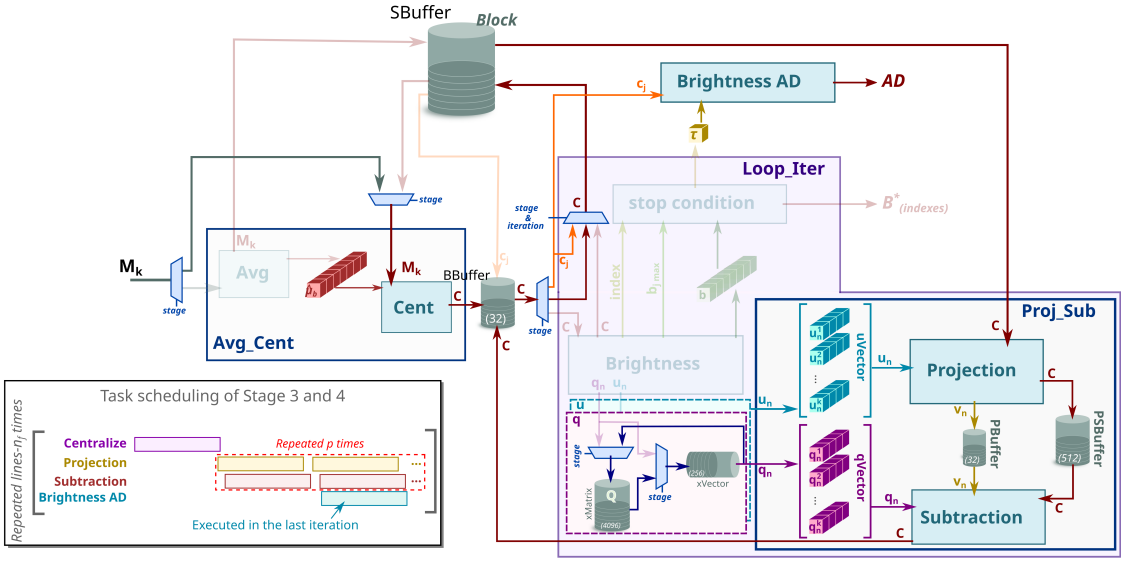


Fig. 3 Computation of the background subspace and anomalous target detection (Stage 3 and 4).

the background pattern, they effectively represent most of the spectral information in each pixel of $\mathbf{M}_k$. These vectors are used by the LbL-FAD algorithm to project image pixels onto them, retaining orthogonal information, which is crucial for detecting anomalous spectra.

Stage 3 is unique among the stages, involving only three hyperspectral operators (Figure 3). First, the new hyperspectral block, $\boldsymbol{M_k}$, undergoes *centralization* using the centroid $\hat{\boldsymbol{\mu}}_{\boldsymbol{b}}$ obtained in Stage 2. As a result, the *Avg* module is skipped, and the *Cent* module becomes the first operation. The centralized block is then stored in the *SBuffer* FIFO for later processing by the *Projection* and *Subtraction* modules (lines 8-11, Algorithm 2). The number of iterations is fixed, corresponding to the number of orthogonal vectors from Stage 2, denoted as p. The *Brightness* and *stop condition* modules are not used, meaning no new orthogonal vectors, $\boldsymbol{u_n}$ and $\boldsymbol{q_n}$, are generated.

An anomalous pixel is identified by a notably high $l^2 - norm$, also known as brightness, after subtracting the spectral information spanned by the background pattern. On this basis, the *Projection* and *Subtraction* operations are repeated for each of the p times using the orthogonal vectors selected in Stage 2 and stored in *qMatrix* and *uMatrix*. It is important to note that these orthogonal vectors must be preserved for subsequent blocks. Thus, the *qMatrix* and *uMatrix* memories have been tailored to store the data as it is being read, ensuring that the data are maintained within the same memory space and order. (see dark blue arrows in Figure 3). Moreover, to preserve the consistency with the preceding stages, the orthogonal vectors, $\boldsymbol{u_n}$ and $\boldsymbol{q_n}$, are replicated into *uBuffer* and *qBuffer*, respectively. This ensures that the data is fed through the same channel to the *Projection* and *Subtraction* modules. Furthermore, the output of the last subtraction operation, $\mathbf{c}_j$, is not discarded. Instead, it is used as the input for the *Brightness AD* module. In addition, $\mathbf{c}_j$ is also stored in *SBuffer* memory to extract spectral information when the block contains anomalous pixels. To do this, the $\mathbf{c}_j$ is duplicated to follow the two paths: feeding the *Brightness AD* module and storing the data in *SBuffer*, but the *SBuffer* will be drained when no anomalies are detected.

Regarding the compression process, a projection vector, v_n, is obtained in each iteration, n, and may be used for offline data reconstruction. Meanwhile, the orthogonal vectors stored in $\mathbf{Q}$ and $\mathbf{U}$ matrix are not included in the output stream at this algorithm stage. Therefore, the pixels selected in previous Stage 2 can perfectly represent the subsequent blocks. For this reason, projection vectors, $\mathbf{V}$, are part of the output of this stage.

Stage 4 identifies anomalous pixels, which differ from the background spectra, and can run in parallel with Stage 3. The subtracted image from the final iteration of Stage 3 is used as input for Stage 4 (orange arrow in Figure 3). Stage 4 consists of the *Brightness AD* module, which generates the anomaly map by computing the brightness of each pixel in the last subtracted image and comparing it with the threshold, $\boldsymbol{\tau}$, from Stage 2. The output is an anomaly map where 1 indicates an anomalous pixel and 0 indicates a normal pixel, with the map generated block by block in batch mode.

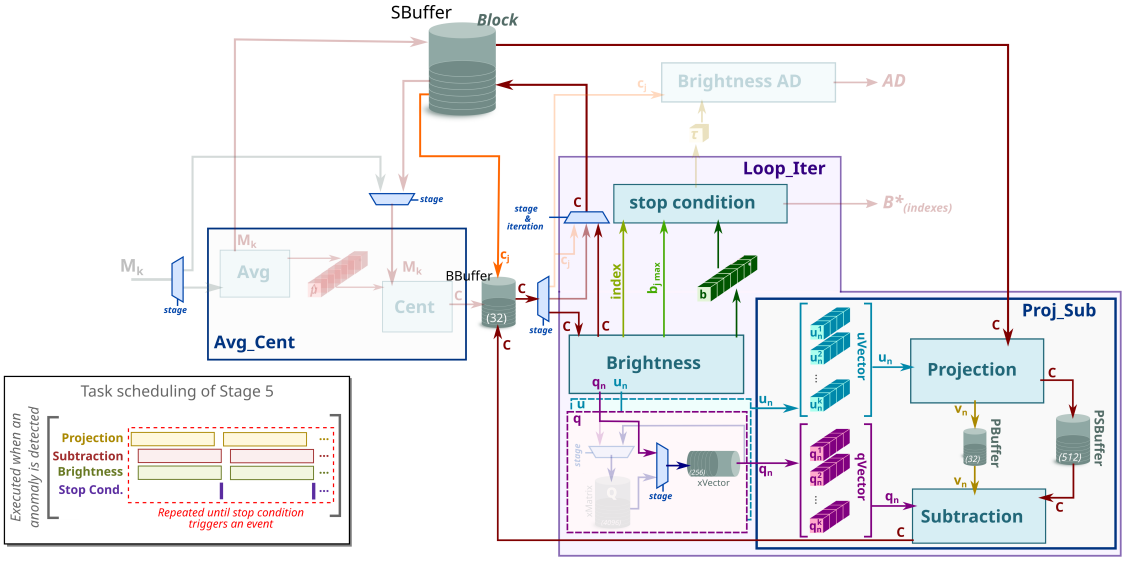


Fig. 4 Spectra extraction of anomalous blocks (Stage 5).

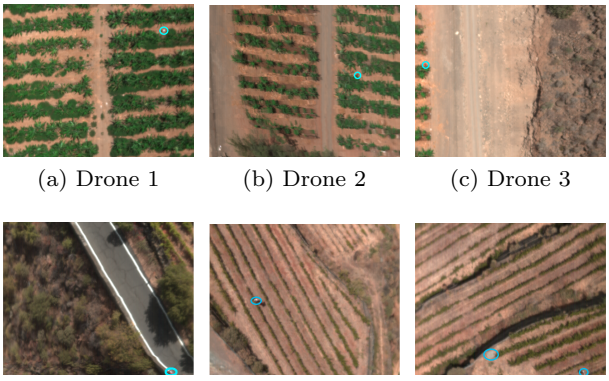
3.4 Stage 5. Spectra extraction of anomalous blocks

The HADeLoC algorithm incorporates an additional computing stage, because the anomalous pixels detected in the previous stage cannot be effectively reconstructed using the transmitted $\mathbf{V}$ vectors, as they only retain spectral information representative of the background pattern. In Stage 3, the latest subtraction output, i.e. p^{th} iteration, is stored in *SBuffer* and subsequently loaded by the *Brightness* module if the block contains anomalous pixels. Thus, the *Brightness* module then calculates new orthogonal vectors, $\mathbf{u}_{p+i}$ and $\mathbf{q}_{p+i}$, which are used by the *Projection* and *Subtraction* modules, respectively. Until the *Stop Condition* module halts the execution, the loop *Brightness-Projection-Subtraction* is performed. In this case, like Stage 1 the stop condition is determined by comparing the difference between brightness ratios of consecutive iterations, i.e. the brightness of $(p+i)^{th}$ and $(p+i+1)^{th}$ are compared to determine when the block is well represented.

The output stream differs slightly from the previously mentioned scenarios without anomalies. In this case, the output also includes the projection vectors, $\mathbf{V}$, of the image pixels onto the $\mathbf{Q}$ and $\mathbf{U}$ vectors, which represent the background pattern estimated in Stage 3. Additionally, it incorporates the extra selected $\mathbf{e}_n$ vectors and their corresponding $\mathbf{v}_n$ values, which are synchronized by barriers like the output of the Stage 1.

4 Results

This section evaluates the HADeLoC algorithm by analyzing its FPGA-based implementation on a ZC7Z020.



(a) Drone 1 (b) Drone 2 (c) Drone 3

(d) Drone 4 (e) Drone 5 (f) Drone 6

Fig. 5 RGB representation of the dataset with anomalous spectra highlighted in blue circles.

Performance metrics, including throughput and power consumption, are compared across different configurations with multiple parallel Processing Elements (PEs). Each PE processes a pixel band, enabling simultaneous processing of multiple bands. The architecture employs two levels of parallelization: (1) core operations run in parallel when data is available, and (2) fine-grain parallelization within each hyperspectral operator allows multiple bands to be processed concurrently.

The proposed FPGA-based architecture was evaluated using hyperspectral images captured by a custom aerial platform over vineyards in Gran Canaria, Spain. The dataset, consistent with previous studies, includes images from two vineyard areas at 27°59′35.6″N 15°36′25.6″W and 27°59′15.2″N 15°35′51.9″W.

A *Specim FX10* pushbroom hyperspectral camera mounted on a DJI Matrice 600 drone acquired the data, capturing 1024 spatial pixels per track and up to 224

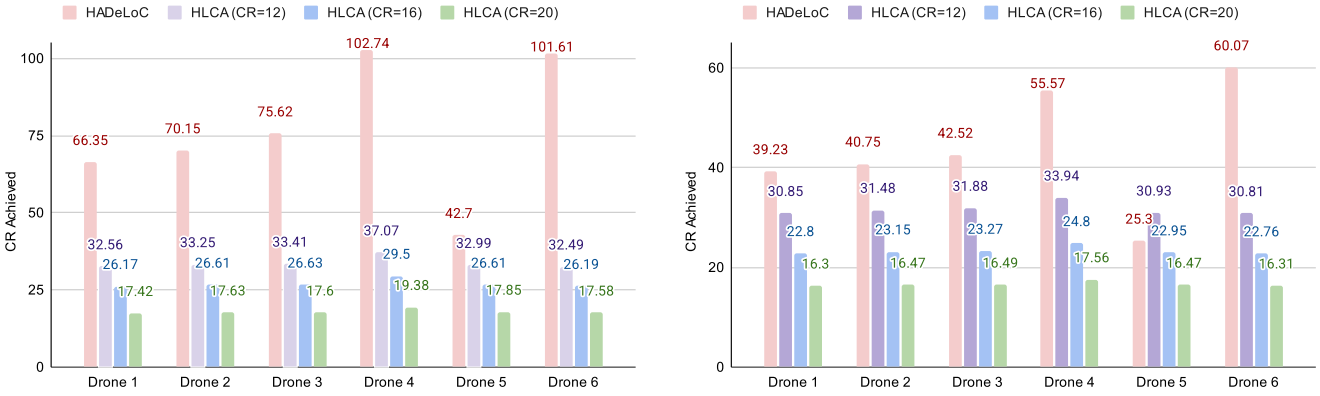


Fig. 6 Comparison among the compression ratios obtained by HLCA and HADeLoC. (a) HLCA: $N_{bits} = 12$; HADeLoC: $N_{bits} = 14$. (b) HLCA: $N_{bits} = 8$; HADeLoC: $N_{bits} = 10$.

spectral bands (400–1000 nm). However, only 160 bands were used for experiments, excluding bands with low spectral response. The dataset consists of 825 hyperspectral blocks (1024 pixels × 160 bands, 12-bit depth). The first 100 blocks are used to compute the background in Stage 1, while the remaining are analyzed for anomalies. All blocks are also used for lossy compression evaluation.

RGB representations of the dataset are shown in Figure 5. The images in Figures 5a -5c were captured at 72m altitude, 6m/s speed, and 125 FPS, resulting in a ground sampling distance of 5cm (GSD). Figure 5d corresponds to a second flight at 45m altitude, 4.5m/s speed, and 150 FPS, achieving a 3cm GSD. Figures 5e and 5f were captured at 45m altitude, 6m/s speed, and 200 FPS, also resulting in a 3cm GSD.

The images mentioned above were calibrated using white and dark calibration to derive reflectance values. However, neither orthorectification nor georeferencing processes were applied to the raw data. Instead, the images were constructed by arranging the subsequent captured hyperspectral frames alongside each other. This approach does not impact the quality of the experiments conducted in this study, as the algorithms tested do not rely on spatial information. It is worth mentioning the presence of anomalous artifacts in these images, such as humans and concrete construction, which have been highlighted with blue circles in Figure 5.

4.1 Compression performance analysis

Unlike the HLCA algorithm, the HADeLoC approach does not require a predefined compression ratio (CR); instead, the number of selected pixels (p) in Stages 1 and 2 is estimated based on a quality stopping condition. A block size (BS) of 1024 was used, aligning with the image acquisition system and the previous optimal HLCA results [12]. The experiments were carried out with $BS = 1024$ and $N_{bits} = [14, 10]$, increasing the values by two compared to prior works [12] [11] ($N_{bits} = [12, 8]$) to accommodate the scaled $\mathbf{V}$ vectors, which range between $[0, 4]$ after pixel brightness scaling.

Table 1 Compression performance results obtained with HADeLoC applied to the collected dataset.

Image		N_{bits}	CR	bpppb	SNR	MAD	RMSE	PNSR	SSIM
Drone 1	Fig. 5a	14	39.23	0.31	39.45	287.00	15.55	48.41	0.98
		10	66.35	0.18	39.32	286.00	15.79	48.28	0.98
Drone 2	Fig. 5b	14	40.75	0.29	38.03	317.00	16.38	47.96	0.96
		10	70.15	0.17	37.90	320.00	16.62	47.83	0.96
Drone 3	Fig. 5c	14	42.52	0.28	40.09	203.00	11.78	50.82	0.96
		10	75.62	0.16	39.76	205.00	12.24	50.49	0.96
Drone 4	Fig. 5d	14	55.57	0.22	21.49	470.00	49.68	38.32	0.62
		10	102.74	0.12	21.47	471.00	49.79	38.30	0.61
Drone 5	Fig. 5e	14	25.30	0.47	33.66	568.00	23.97	44.65	0.90
		10	42.70	0.28	33.58	567.00	24.18	44.58	0.90
Drone 6	Fig. 5f	14	60.07	0.20	30.19	371.00	33.48	41.75	0.85
		10	101.61	0.12	30.18	372.00	33.52	41.74	0.85

Table 1 presents the results obtained, including the results achieved for CR. Compression performance was assessed in two ways: (1) by evaluating the compression ratio and the average number of bits per pixel per band ($bpppb$) in compressed images, and (2) by analyzing data loss using five quality metrics: signal-to-noise ratio (SNR), mean absolute deviation (MAD), root mean square error ($RMSE$), structural similarity index measure ($SSIM$), and peak signal-to-noise ratio ($PSNR$).

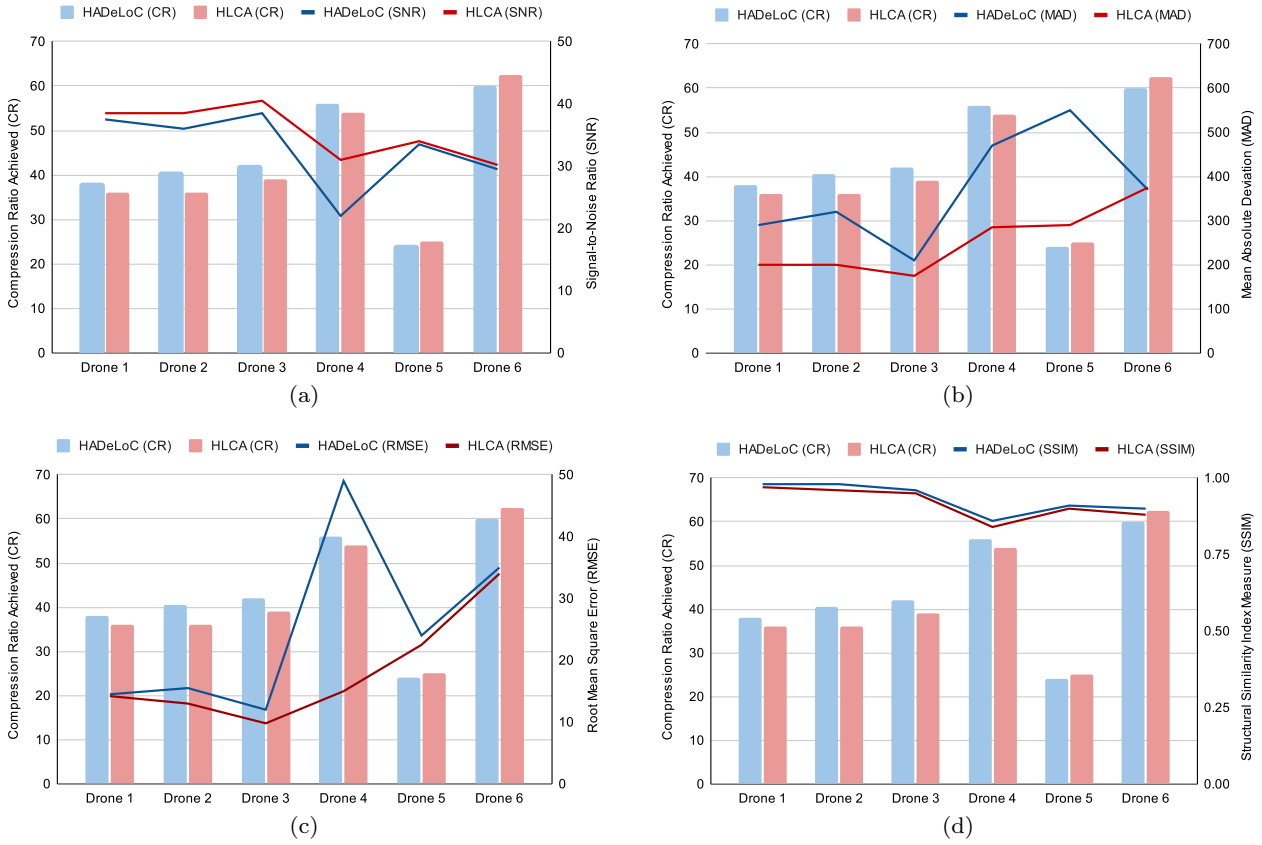


Fig. 7 Quality compression results of HLCA and HADeLoC algorithms. (a) SNR. (b) MAD. (c) RMSE. (d) SSIM.

Figure 6 compares the CR obtained with those of the HLCA algorithm [11], evaluated at three minimum compression ratios ($CR = [12, 16, 20]$). The comparison aligns $N_{bits} = 14$ with 12 bits and $N_{bits} = 10$ with 8 bits. The results highlight several key observations. First, the proposed algorithm dynamically determines the CR based on the spectral variability of the HSIs analyzed. As shown in Figure 6, the new approach consistently achieves significantly higher CRs than HLCA, even in its highest setting ($CR = 20$). This difference is particularly notable when $N_{bits} = 10$, where CR is 40-46% higher than with $N_{bits} = 14$. This occurs because the same number of pixels (p) is selected in Stages 1 and 2, and the same number of anomalous spectra is identified, while N_{bits} only affects the scaling and encoding of the $\mathbf{V}$ vectors. Consequently, compressed data using $N_{bits} = 14$ require more bits than those with $N_{bits} = 10$.

From a quality perspective, setting N_{bits} to 14 or 10 results in similar spectral distortions, but $N_{bits} = 10$ achieves a 60% higher compression ratio. However, the compression quality is lower than that of the HLCA algorithm due to the significantly higher CR values obtained. For *Drone 5* (Figure 5e), CR is comparable to HLCA at $CR = 20$, leading to similar SNR values. Adjusting the stopping condition to match HLCA's CR would yield more comparable quality metrics.

It is important to note that the dataset primarily consists of dark images that do not fully utilize the sensor's dynamic range. This is evident from the *PSNR* values in Table 1. *PSNR* represents the ratio between the maximum possible power of a signal and the power of corrupting noise, while *SNR* represents the ratio between the signal power and the noise power. The significantly higher *PSNR* values compared to *SNR* values indicate that the images were captured under low-light conditions. Consequently, the inherent image noise is comparable to the signal power, negatively impacting the *SNR* values. Additionally, the presence of undesirable distortions, such as sparkles, scattering, or the sweeping motion of the data acquisition platform, can lead to a misrepresentation of the background model estimated in Stages 1 and 2. This, in turn, affects the quality of compression in subsequent image blocks. This effect is particularly evident in *Drone 4* (Figure 5d), which exhibits the poorest reconstruction after the compression/decompression process. Furthermore, this specific image poses an additional challenge, as the pixels on the white road markings are nearly saturated, resulting in high l^2-norm values, i.e., high brightness levels.

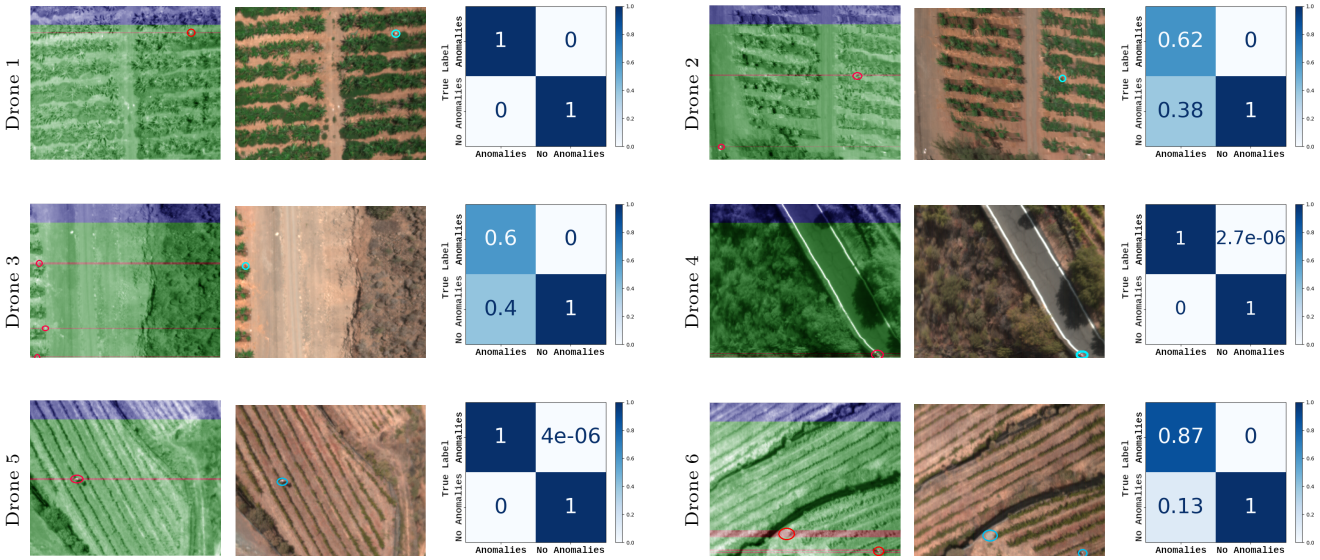


Fig. 8 Anomaly detection results obtained by the HADeLoC proposal.

Finally, regarding the Maximum Absolute Difference (*MAD*) assessment metric, it exhibits the greatest discrepancies compared to the results obtained with the HLCA compressor. The *MAD* represents the highest reconstruction error among all image pixels after the compression/decompression process. The pixels located on the rounded edges of anomalous entities primarily consist of background spectra mixed with the anomalous signature. As a result, they cannot be perfectly reconstructed through a linear combination of the background spectra, leading to the highest *MAD* values in the proposed methodology.

Beyond the previous analysis, the HADeLoC has also been compared with the proposed methodology under more similar conditions, i.e. when both HLCA and HADeLoC achieve comparable compression ratios (*CR*). Figure 7 presents four graphical representations showing the achieved *CR* and the quality of compression results through the *SNR*, *MAD*, *RMSE*, and *SSIM* metrics for both algorithms. In this scenario, the number of bits per pixel (N_{bits}) has been set to 14 for HADeLoC and 12 for HLCA, with similar behavior observed at lower N_{bits} configurations. The *MAD* and *RMSE* metrics indicate that the HLCA algorithm achieves better performance with lower information loss after the compression/decompression process. The key difference lies in Stage 3: HLCA centralizes each image block, $\mathbf{M}_k$, using its average pixel, $\hat{\boldsymbol{\mu}}$, and then selects pixels $\mathbf{E}$ to represent the block processing. In contrast, the HADeLoC algorithm relies on $\hat{\boldsymbol{\mu}}$ as well as the orthogonal vectors $\mathbf{Q}$ and $\mathbf{U}$ from the initial blocks n_f obtained in Stage 2. This means that the quality of the proposed method depends on background modeling. Consequently, the image *Drone 4* (Figure 5d) is not well reconstructed because the spectral information of the white road markings is treated as part of the background, leading to the highest *MAD* values. However, for the remaining images in the dataset, the reconstructions are closely aligned with those of the HLCA.

4.2 Anomaly detection performance analysis

The HADeLoC algorithm follows the processing stages of the LbL-FAD algorithm, yielding similar anomaly detection results. Figure 8 presents anomaly detection maps overlaid on panchromatic images, highlighting pixels with anomalies in red. Each scenario includes a confusion matrix to assess the detection performance.

The algorithm assumes that the first n_f hyperspectral frames sufficiently represent the background. However, in highly heterogeneous environments or in the presence of distortions (e.g., sparkles, scattering, or platform motion), this assumption may not hold. As a result, approximately 40% of anomalies in *Drone 2* and *Drone 3* are missed. In contrast, background models for other images are more representative, leading to better anomaly detection. *Drone 1* achieves full detection, while *Drone 4* and *Drone 5* exhibit near-accurate performance, though some anomalies are missed. In *Drone 6*, some pixels were incorrectly flagged as anomalous.

Table 2 Hardware utilization of HADeLoC algorithm for a ZC7Z020 SoC after post-implementation phase.

PEs	BRAM18K		DSP48E		FlipFlops		LUTs	
1	214	*(76.43%)*	14	*(6.36%)*	8,181	*(7.69%)*	6,859	*(12.89%)*
2	183	*(65.36%)*	22	*(10.00%)*	8,731	*(8.21%)*	7,624	*(14.33%)*
4	191	*(68.21%)*	38	*(17.27%)*	10,041	*(17.03%)*	9,261	*(17.41%)*
8	197	*(70.36%)*	70	*(31.82%)*	12,728	*(11.96%)*	12,576	*(23.64%)*
10	205	*(73.21%)*	86	*(39.09%)*	14,840	*(13.95%)*	14,846	*(27.91%)*
16	193	*(63.93%)*	134	*(60.91%)*	18,491	*(17.38%)*	18,970	*(35.66%)*
20	197	*(70.36%)*	166	*(75.45%)*	23,096	*(21.71%)*	23,959	*(45.04%)*

Table 3 FPS and FPS/W achieved by the different versions of the HADeLoC accelerator (Clock Frequency: 143MHz).

PEs	1	2	4	8	10	16	20
FPS	122	245	489	855	1006	1366	1552
HWacc (W)	0.473	0.512	0.496	0.688	0.802	0.920	1.214
FPS/W	259	478	986	1243	1254	1485	1278

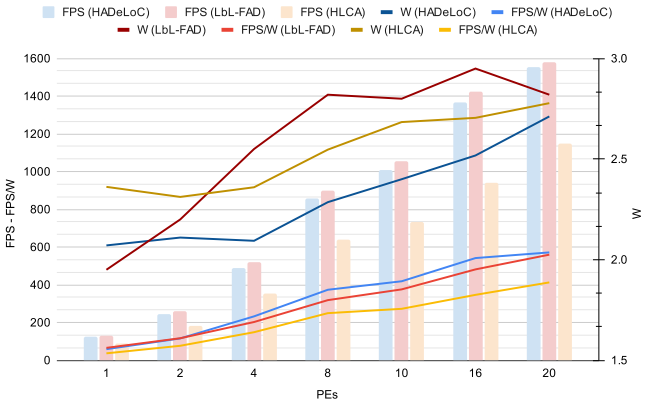


Fig. 9 Performance and power trade-off analysis of HLCA, LbL-FAD and HADeLoC proposals (Clock freq.: 143MHz).

4.3 Hardware and performance analysis

The HADeLoC architecture has been implemented on a ZC7Z020 FPGA for performance evaluation. This new version runs two applications simultaneously, making direct comparisons with previous implementations [12] and [13], but HADeLoC is considered a unique application in this evaluation. The selected FPGA (Artix) offers a balance between cost, power efficiency, and throughput compared to higher-end architectures like Kintex or Ultrascale+. However, the hardware limitations of ZC7Z020 require significant engineering efforts, as discussed previously.

Table 2 details the resource usage for different configurations based on the number of instantiated processing elements (PEs). Each PE processes multiple bands in parallel in batch mode. The block size is set to 1024 hyperspectral pixels, matching the sensor's maximum capacity, with 160 spectral bands. The FPGA supports up to 20 PEs, constrained by the 220 available DSPs, with this configuration using 166. Thus, DSPs are the primary limiting resource. The number of PEs must be a divisor of the total bands to optimize hardware efficiency. While other resources are not critical to scalability, BRAM utilization ranges from 64% to 77%, mainly allocated to the *SBuffer* FIFO for storing the hyperspectral block during processing.

The performance of the HADeLoC hardware accelerator is measured in frames per second (FPS), where each frame corresponds to a block or line of 1024 hyperspectral pixels. Table 3 presents the achieved FPS based on the number of instantiated PEs, with the FPGA operating at 143MHz. To ensure consistency, the same hyperspectral images (HSIs) used in previous implementations are employed for performance evaluation and comparison. The results demonstrate a linear scalability pattern, which means that performance increases proportionally with the number of PEs. Additionally, HADeLoC's performance is influenced by the selected background and the proportion of pixels not represented by it, i.e., anomalous pixels. This variability causes FPS fluctuations of up to ±1.5% compared to the values reported in Table 3. On this basis, the architecture with four processing elements is adequate for real-time processing at the highest spatial and spectral resolution captured by the Specim FX10 camera, as it can handle images at 489 FPS. Furthermore, a more advanced hyperspectral sensor, such as the Specim FX10+ camera, which increases the image speed to 705 FPS, can also be integrated with the proposed architecture. However, to achieve real-time processing with this sensor, the number of processing elements must be increased to 10. Table 3 also provides information on the power consumption of the FPGA-based implementation in various configurations of the HADeLoC accelerator on a ZC7Z020 device. From an energy efficiency point of view, the architecture performs well, with a power usage ranging from 0.5 watts for a single PE to 1.2 watts when scaling up to 20 PEs. However, the power consumption follows an exponential growth trend rather than a linear one, primarily due to the increased number of active hardware resources and clock regions.

Figure 9 shows the performance and power trade-off among HLCA, LbL-FAD, and HADeLoC. HADeLoC processes frames at slightly lower FPS than LbL-FAD but higher than HLCA, maintaining LbL-FAD's speed advantage. HADeLoC also consumes 6-30% less power than the other implementations. In higher PE configurations, HADeLoC outperforms in FPS-to-power ratio, while in lower PE setups, all three have similar FPS/W performance. This confirms HADeLoC's performance is similar to LbL-FAD, reflecting comparable computational strategies.

Table 4 Datasets/devices of state-of-the-art and HADeLoC.

	Prop.	Data Set	Lines	BS	Bands	Device	Price (€)
Compression	[15]	-	16	16	256	5VFX130	5,011.82
	[16]	Jasper Ridge	614	512	224	5VFX100T	4,819.07
	[17]	Jasper Ridge	614	512	224	XC7VX690T	19,029.66
	[18]	AVIRIS	512	512	256	XC7VX690T	19,029.66
	[19]	AVIRIS	512	512	256	XCZU9EG	4,832.37
	[12]	Drone	1024	1024	180	ZC7Z020	168.45
Anomaly Detection	[20]	HyMap	614	512	126	XC7VX690T	19,029.66
		WTC	614	512	224		
	[21]	HyMap	100	300	126	XC7VX980T	34,939.96
		Hydice Forest	64	64	169		
		Hydice Urban	80	100	175		
	[22]	San Diego	100	100	189	XC7K325T	1,820.75
		Urban-Beach 1	100	100	207		
		Urban-Beach 2	100	100	191		
		Urban-Beach 3	100	100	205		
		Urban-Beach 4	150	150	188		
		EI Segundo	250	300	224		
	[23]	Sentinel-2	120	120	10	XCZU9EG	4,832.37
	[13]	Drone	825	1024	160	ZC7Z020	168.45
	ours	Drone	825	1024	160	ZC7Z020	168.45

5 Discussion

The hardware resource utilization and performance of HADeLoC have been compared with state-of-the-art proposals using reconfigurable hardware. This analysis considers two HSI applications: compression and anomaly detection. Table 4 lists the datasets, device setups, and hyperspectral cube sizes used by each proposal, noting that HADeLoC processes the largest hyperspectral cube, requiring significant internal memory. The table also shows the selected devices and their costs. Unlike other proposals using high-cost devices with extensive resources, HADeLoC is designed for cost-effective devices, making it more suitable for embedded systems while still achieving high performance.

Table 5 compares the hardware resource utilization and throughput of state-of-the-art proposals on reconfigurable hardware. Throughput is analyzed using metrics like Megasamples per second (MSample/s), MB per second (MB/s), and data processed per second per watt (MBS/W). Power consumption is shown for each hardware accelerator, focusing only on the accelerator itself.

The HADeLoC algorithm has been compared with other FPGA-based compressors. For example, D. Bascones et al. [18] present an FPGA-based prediction-based compression architecture with a throughput of up to 170.33 MSamples/s and power consumption of 0.714 watts. While their power consumption is lower than HADeLoC's, the MBS/W of their design is slightly higher. In general, the performance of their architecture is similar to that of HADeLoC. L. Santos et al. and A. García et al. propose lossy compression algorithms for the ESA ExoMars mission [15], [16]. These solutions, developed using HLS (CatapultC), show lower performance as reflected in the throughput metrics in Table 5. Similarly, Y. Barrios et al. in [19] present a lossy compressor based on CCSDS with HLS, deployed on a reconfigurable platform (ARTICo). While scalable, this flexibility reduces throughput, achieving only 1.87 MSamples/s and higher power consumption per megabyte (3.96 MBS/W). In [17], Fernández et al. use PCA-based dimensionality reduction for HSI compression, but their throughput is much lower than HADeLoC's, even with a VHDL-based solution. The HADeLoC architecture, developed with a mix of HLS and VHDL, achieves better parallelization, lower power consumption, and higher throughput compared to these approaches.

The HADeLoC architecture has also been compared with several state-of-the-art anomaly detectors. J. Wu et al. in [21] optimize internal memory with FFs and LUTs, but their design only processes one pixel per 17 cycles, limiting FPGA parallelization. B. Yang et al. in [20] also use fixed-point precision but fail to achieve higher parallelism. J. Lei et al. in [22] create a pipeline entirely in HLS, which results in lower throughput. M. Coca et al. in [23] employ Vitis AI Framework for natural anomaly detection, but again, performance is limited by high-level tools.

Regarding hardware resource utilization, HADeLoC uses a significant portion of available resources due to the cost-optimized design, as opposed to oversized FPGA devices that provide unnecessary resources. While high-level frameworks reduce development time, they do not fully leverage FPGA potential, leading to lower throughput. The HADeLoC architecture outperforms state-of-the-art proposals in terms of MSample/s and MB/s, and even though it is slightly slower than previous versions, but it adds the benefit of performing two HSI applications simultaneously.

From an energy perspective, the HADeLoC architecture consumes 1.082 watts, positioning it in the middle of the state-of-the-art proposals. It is important to note that many of these proposals do not provide power consumption data, so estimates were made using the Xilinx Power Estimator (XPE) tool. The energy efficiency is evaluated through the MBS/W metric, which measures the number of megabytes processed per second per watt. In this regard, the HADeLoC architecture demonstrates a strong trade-off in terms of MBS/W, highlighting its high energy efficiency. The architecture by D. Bascones et al. in [18] is also energy-efficient, as it requires a low power budget for their HWacc.

Table 5 Hardware utilization, power consumption and throughput of the state-of-the-art and HADeLoC proposals.

	Proposal	Hardware Resources				Freq.	Power	Throughput		
		BRAMs	DSPs	FFs	LUTs	(MHz)	(Watts)	MSamples/s	MB/s	MBS/W
Compression	L. Santos et al. [15]	17 *(2.852%)*	4 *(1.250%)*	4,208 *(20.55%)*	7,836 *(5.978%)*	80.2	2.029*	19.25	36.71	18.09
	A. García et al. [16]	4 *(0.877%)*	25 *(9.766%)*	1,937 *(3.027%)*	7,746 *(12.10%)*	86.96	2.022*	27.37	52.21	25.82
	D. Fernández et al. [17]	897 *(30.51%)*	2,580 *(71.67%)*	57,564 *(6.644%)*	294,454 *(67.99%)*	75.99	2.46*	16.87	32.21	13.09
	D. Báscones et al. [18]	6 *(0.204%)*	5 *(0.139%)*	-	6,731 *(1.554%)*	355.3	0.714	170.33	324.87	455.00
	Y. Barrios et al. [19]	312 *(34.21%)*	26 *(1.032%)*	42,771 *(7.803%)*	56,800 *(20.72%)*	100	0.9*	1.87	3.56	3.96
	J. Caba et al. [12]	218 *(77.85%)*	202 *(91.82%)*	20,028 *(18.82%)*	19,415 *(36.49%)*	143	1.225	265.84	507.04	413.91
Anomaly Detection	B. Yang et al. [20]	240 *(8.163%)*	265 *(7.361%)*	28,245 *(3.260%)*	21,730 *(5.016%)*	200	0.641	36.34	69.31	108.13
		284 *(9.660%)*	459 *(12.75%)*	43,544 *(5.026%)*	33,997 *(7.848%)*	200	0.897	29.70	56.64	63.15
	J. Wu et al. [21]	4 *(0.133%)*	1,064 *(29.56%)*	772,164 *(63.09%)*	269,751 *(44.08%)*	100	2.72	10.08	19.23	7.07
		3 *(0.100%)*	460 *(12.78%)*	277,048 *(22.63%)*	140,362 *(22.93%)*	100	1.333	11.16	21.29	15.97
		3 *(0.100%)*	580 *(16.11%)*	414,673 *(33.89%)*	146,977 *(24.02%)*	100	1.619	11.29	21.53	13.30
	J. Lei et al. [22]	216 *(24.27%)*	12 *(1.429%)*	49,964 *(12.26%)*	33,969 *(16.66%)*	200	0.614	3.58	6.83	11.12
		282 *(31.68%)*	12 *(1.429%)*	57,962 *(14.22%)*	43,890 *(21.54%)*	200	0.687	3.74	7.14	11.39
		198 *(22.25%)*	12 *(1.429%)*	46,370 *(11.38%)*	31,681 *(18.98%)*	200	0.593	3.72	7.10	11.97
		282 *(31.69%)*	12 *(1.429%)*	57,929 *(14.21%)*	43,874 *(21.53%)*	200	0.686	3.71	7.07	10.31
		282 *(31.69%)*	12 *(1.429%)*	58,593 *(14.37%)*	44,906 *(22.03%)*	200	0.691	3.59	6.86	9.93
		525 *(58.99%)*	12 *(1.429%)*	83,713 *(20.54%)*	67,591 *(33.17%)*	200	0.911	2.14	4.08	4.48
	M. Coca et al. [23]	255 *(27.96%)*	710 *(28.17%)*	98,525 *(17.97%)*	51,351 *(18.74%)*	100	1.3*	0.206	0.392	0.302
	J. Caba et al. [13]	197 *(70.36%)*	166 *(75.45%)*	23,071 *(21.68%)*	23,493 *(44.16%)*	143	1.077	265.03	505.51	469.37
	ours	197 *(70.36%)*	166 *(75.45%)*	23,096 *(21.71%)*	23,959 *(45.04%)*	143	1.082	255.03	486.44	449.57

* Estimated with the Xilinx Power Estimator tool (XPE).

6 Conclusion

In recent years, significant efforts have been dedicated to on-board HSI processing for real-time applications, moving away from large high-performance computing systems. Although next generation commercial hardware devices evolve, offering improved spatial, spectral, and temporal resolutions, real-time onboard processing of HSIs remains challenging, particularly when multiple applications need to be executed concurrently on the same hardware. This requires the development of hardware-friendly algorithms and reconfigurable platforms to tackle such challenges in hyperspectral remote sensing.

This paper presents the HADeLoC algorithm, which enables real-time processing by executing multiple hyperspectral analysis techniques simultaneously. It reduces hardware acceleration time by reusing products across algorithms and allows concurrent task execution, minimizing computational cost and hardware requirements. Designed for embedded systems, especially autonomous UAVs, HADeLoC addresses energy and resource constraints by modifying the datapath at runtime. HADeLoC offers compression performance comparable to the HLCA algorithm and matches LbL-FAD's anomaly detection precision. It outperforms existing methods in compression quality, detection accuracy, processing time, and power efficiency, making it ideal for production on cost-optimized devices.

In this work, two main areas of improvement have been identified. First, we aim to enhance the background modeling process, as the initial captured lines play a crucial role in detecting anomalies throughout the rest of the scene. Second, we plan to explore energy-saving techniques for embedded systems, including dynamic power management, adaptive compression, and integration of energy-harvesting methods.

Acknowledgments

This research has been funded by the Spanish Ministry of Economy and Competitiveness under the OASIS (PID2023-148285OB-C4, subprojects 1 and 3) and T2C-BELIEF (PDC2023-145865-C32) projects and by the Spanish Ministry of Digital Transformation and Public Service through PERTE Chip Grants (UCLM Chair, TSI-069100-2023-0014), Next Generation EU Funds. Open Access funding provided by the University of Castilla-La Mancha.

References

1. A. Plaza, J. A. Benediktsson, J. W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, A. Gualtieri *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote sensing of environment*, vol. 113, pp. S110–S122, 2009.
2. A. Altamimi and B. Ben Youssef, "A Systematic Review of Hardware-Accelerated Compression of Remotely Sensed Hyperspectral Images," *Sensors*, vol. 22, no. 1, 2022.

3. M. Radosavljević, B. Brkljač, P. Lugonja, V. Crnojević, Ž. Trpovski, Z. Xiong, and D. Vukobratović, "Lossy compression of multispectral satellite images with application to crop thematic mapping: A hevc comparative study," *Remote Sensing*, vol. 12, no. 10, p. 1590, 2020.
4. F. Ortenberg, P. Thenkabail, J. Lyon, and A. Huete, "Hyperspectral sensor characteristics: airborne, spaceborne, hand-held, and truck-mounted; integration of hyperspectral data with lidar," *Hyperspectral Remote sensing of vegetation*, pp. 39–68, 2011.
5. E. Morin, M. Maman, R. Guizzetti, and A. Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," *IEEE Access*, vol. 5, pp. 7097–7114, 2017.
6. G. Benelli, G. Meoni, and L. Fanucci, "A low power keyword spotting algorithm for memory constrained embedded systems," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2018, pp. 267–272.
7. D. Gyaneshwar and R. R. Nidamanuri, "A real-time FPGA accelerated stream processing for hyperspectral image classification," *Geocarto International*, vol. 37, no. 1, pp. 52–69, 2022.
8. T. Yan, N. Zhang, J. Li, W. Liu, and H. Chen, "Automatic Deployment of Convolutional Neural Networks on FPGA for Spaceborne Remote Sensing Application," *Remote Sensing*, vol. 14, no. 13, 2022.
9. M. Xu, L. Chen, H. Shi, Z. Yang, J. Li, and T. Long, "FPGA-Based Implementation of Ship Detection for Satellite On-Board Processing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 9733–9745, 2022.
10. R. Guerra, Y. Barrios, M. Díaz, L. Santos, S. López, and R. Sarmiento, "A New Algorithm for the On-Board Compression of Hyperspectral Images," *Remote Sensing*, vol. 10, no. 3, 2018.
11. M. Díaz, R. Guerra, P. Horstrand, S. López, and R. Sarmiento, "A Line-by-Line Fast Anomaly Detector for Hyperspectral Imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 8968–8982, 2019.
12. J. Caba, M. Díaz, J. Barba, R. Guerra, J. A. d. l. T. López, and Sebastián, "FPGA-Based On-Board Hyperspectral Imaging Compression: Benchmarking Performance and Energy Efficiency against GPU Implementations," *Remote Sensing*, vol. 12, no. 22, 2020.
13. J. Caba, M. Díaz, J. Barba, R. Guerra, S. Escolar, and S. López, "Low-Power Hyperspectral Anomaly Detector Implementation in Cost-Optimized FPGA Devices," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 2379–2393, 2022.
14. Y. Saad, *Numerical methods for large eigenvalue problems*. Manchester University Press, 1992.
15. L. Santos, J. F. López, R. Sarmiento, and R. Vitulli, "FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool," in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, 2013, pp. 107–114.
16. A. García, L. Santos, S. López, G. Marrero, J. F. López, and R. Sarmiento, "High level modular implementation of a lossy hyperspectral image compression algorithm on a FPGA," in *2013 5th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2013, pp. 1–4.
17. D. Fernández, C. González, D. Mozos, and S. Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *Journal of Real-Time Image Processing*, vol. 16, 2019.
18. D. Báscones, C. González, and D. Mozos, "An FPGA Accelerator for Real-Time Lossy Compression of Hyperspectral Images," *Remote Sensing*, vol. 12, no. 16, 2020.
19. Y. Barrios, A. Rodríguez, A. Sánchez, A. Pérez, S. López, A. Otero, E. de la Torre, and R. Sarmiento, "Lossy Hyperspectral Image Compression on a Reconfigurable and Fault-Tolerant FPGA-Based Adaptive Computing Platform," *Electronics*, vol. 9, no. 10, 2020.
20. B. Yang, M. Yang, A. Plaza, L. Gao, and B. Zhang, "Dual-mode fpga implementation of target and anomaly detection algorithms for real-time hyperspectral imaging," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2950–2961, 2015.
21. J. Wu, Y. Jin, W. Li, L. Gao, and B. Zhang, "Fpga implementation of collaborative representation algorithm for real-time hyperspectral target detection," vol. 15, no. 3, p. 673–685, oct 2018. [Online]. Available: https://doi.org/10.1007/s11554-018-0823-7
22. J. Lei, G. Yang, W. Xie, Y. Li, and X. Jia, "A low-complexity hyperspectral anomaly detection algorithm and its fpga implementation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 907–921, 2021.
23. M. Coca and M. Datcu, "FPGA Accelerator for Meta-Recognition Anomaly Detection: Case of Burned Area Detection," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 16, pp. 5247–5259, 2023.