Journal of Engineering and Technology for Industrial Applications

ISSN ONLINE: 2447-0228

ITEGAM-JETIA

Manaus, v.12 n.58, p. 284-295. March/April, 2026.

DOI: https://doi.org/10.5935/jetia.v12i58.3026

RESEARCH ARTICLE OPEN ACCESS

LIFTING BASED BLOCK FRACTIONAL WAVELET FILTER COMPRESSION OF HYPERSPECTRAL IMAGES OVER WIRELESS MULTIMEDIA SENSOR NETWORK PLATFORMS

Purushottam Lal Nagar[1], Shrish Bajpai[2]

[1,2]Electronics & Communication Engineering Department, Faculty of Engineering & Information Technology, Integral University, Lucknow, Uttar Pradesh, India

[1]https://orcid.org/0000-0001-7713-5032; [2]https://orcid.org/0000-0001-5598-1940

E-Mail: plnagar88@gmail.com, shrishbajpai@gmail.com

ARTICLE INFO

Article History
Received: November 30, 2025
Reviewed: January 1, 2026
Accepted: January 6, 2026
Published: March 31, 2026

Keywords:
Hyperspectral Image,
Compression,
Energy Efficiency,
Wireless Sensor Network,
Low Memory Coding,
Lifting Wavelet Transform

ABSTRACT

In the rapidly development of remote sensing technology, the compression of Hyperspectral Images is a pivotal yet formidable task. Hindered by inherent limitations in hyperspectral imaging, enhancing the accuracy and efficiency of compression algorithm remains a critical and much-debated issue. Algorithms using set partition wavelet transforms excel in hyperspectral image compression due to their embedded nature, coding efficiency, and low complexity. Specifically, the Fractional wavelet-based zero memory set partitioned embedded block algorithm achieves high coding efficiency with lower memory demands, though its method of repeatedly comparing coefficients to a threshold is time-intensive. To solve this, a new algorithm has been developed that optimizes both computational and memory complexity. It employs a block-based fractional wavelet filter (BFrWF), which delivers the same accuracy as conventional transforms but requires far less memory. The Block-based Fractional Wavelet Filter is a low-memory technique for image transformation, but its high computational complexity makes it impractical for resource-constrained devices in IoT and Wireless Sensor Networks. Additionally, it produces blocking artifacts due to improper handling of block boundaries. This paper introduces a new lifting-based version of BFrWF that eliminates these artifacts by correctly overlapping image blocks. This new implementation with low complexity zero memory set partitioned embedded block (LC-ZM-SPECK) achieves higher coding efficiency, making it well-suited for resource constraint visual sensor nodes.

I. INTRODUCTION

Hyperspectral imaging (HSI) represents a significant analytical advancement as a non-destructive, integrative technology that merges the principles of imaging and spectroscopy to simultaneously capture both spatial and extensive spectral data from a target. Unlike conventional imaging that operates in broad bands, HyperSpectral (HS) image acquires detailed information across a contiguous series of tens to hundreds of narrow, adjacent wavelength bands for each pixel in an image, creating a complex three dimensional data structure known as a hypercube. This rich spectral signature enables the precise identification, quantification, and spatial mapping of chemical constituents (e.g., moisture, fat, pigments), physical properties, and structural characteristics within a sample without any preparatory alteration or destruction [1]. Implementing an efficient compression algorithm enhances sensor performance through multiple key benefits includes reduced coding complexity, lower power consumption, conserved communication bandwidth, faster browsing times, and decreased processing costs. The effectiveness of a HyperSpectral Image Compression Algorithm (HSICA) is evaluated based on three core metrics: coding efficiency, coding memory, and coding complexity [2].

- Coding efficiency is quantified using Compression Ratio (CR), Peak Signal-to-Noise Ratio (PSNR), and the Structural Similarity Index (SSIM).
- Coding memory refers to the amount of memory the algorithm consumes during compression.
- Coding complexity is measured by the time the algorithm takes to encode and decode data, excluding the time spent on the wavelet and inverse wavelet transforms [3].

HS image compression algorithms can be classified in two primary ways: based on the degree of data loss or based on the encoding technique used [2]. The first classification, based on data loss, consists of three categories:

- Lossless: No data is lost, and the original image can be perfectly reconstructed, but this results in a low CR.
- Near-Lossless: A minimal amount of data is lost, leading to a slightly higher CR than lossless methods. The reconstructed image is virtually indistinguishable from the original.
- Lossy: Some data is intentionally discarded, achieving the highest CR. It is known that the human eye typically cannot perceive image degradation beyond a PSNR of 40 dB.

The second classification is based on the encoding methodology, which divides algorithms into seven subgroups [4], [5]:

- Predictive Coding (PC) based HSICAs
- Vector Quantization (VQ) based HSICAs
- Transform Coding (TC) based HSICAs
- Tensor Decomposition (TD) based HSICAs
- Compressive Sensing (CS) based HSICAs
- Machine Learning (ML) based HSICAs
- Hybrid HSICAs

To eliminate undesirable redundancy and exploit the inherent correlation within HS images, TC based compression techniques utilize mathematical transforms. These transforms, such as Fourier, wavelet, cosine, curvelet, and shearlet, serve to project the HS image data into a more compact frequency-domain representation. The wavelet transform is widely recognized for its superior performance in this context, as it provides multi-resolution analysis and maintains localization information in both the original and transformed domains.

II. BACKGROUND

HS image compression is fundamentally achieved by eliminating inherent data redundancies which are categorized as spatial and spectral redundancy. Spectral redundancy arises from the high correlation between adjacent frequency bands, while spatial redundancy is due to the similarity between neighbouring pixels within a band. Research indicates that spectral redundancy contributes more significantly to the overall data volume than spatial redundancy [3]. Mathematical transforms, such as the 3D dyadic wavelet transform (3D-DWT), are highly effective at decorrelating these redundancies. The 3D-DWT is considered an optimal choice for this task due to its superior energy compaction properties across both spatial and spectral dimensions and is typically implemented as a series of one-dimensional wavelet transforms [2].

3D Wavelet transform based HSICAs that utilize set partitioning principles, such as 3D-SPECK [6] or 3D-SPIHT [7], exploit the energy clustering characteristics inherent to the wavelet transform. Their efficiency stems from a strategy that groups insignificant wavelet coefficients into sets. At higher bit planes, where the majority of coefficients are negligible, this set structure allows for their collective representation with a single bit, dramatically improving compression rates. This mechanism provides set partition-based algorithms with a distinct performance advantage over other transform-coding methods. But 3D wavelet transform had high transform memory requirement and high transform complexity exists. Thus, it is convenient to use the 2D wavelet transform instead of 3D wavelet transform to save the requirement of transform memory and low coding complexity [8].

Conventional 2D-DWT computation is characterized by high memory requirements, as the entire image must be loaded into memory to perform sequential row and column filtering and down sampling, producing the standard HH, HL, LH, and LL sub-bands [9]. This makes it impractical for visual sensor nodes in WMSNs/IoT. Although stripe-based, block-based, and line-based techniques have been developed to mitigate this by processing image sections, their memory footprint approximately 26 kB for a ‘512 x 512’ image remains too high. The Fractional Wavelet Filter (FrWF) and Segmented FrWF (SFrWF) represent a trade-off, achieving minimal memory use by processing individual lines or segments but at the cost of greatly increased computational complexity, which also renders them unsuitable for resource-constrained platforms [10].

III. RELATED WORK

This section is structured as follows: first section, we discuss the associated wavelet filter named as Lifting based Block fractional wavelet filter (LBFrWF) [11] and in second section, we introduce the linear indexing method; and finally, we provide a detailed description of the 2D-LC-ZM-SPECK algorithm [12].

III.1 LIFTING BASED BLOCK FRACTIONAL WAVELET FILTER

By [13] introduced a block-based implementation of the Fractional Wavelet Filter Transform (FrWF) for the gray scale images to reduce the original algorithm's computational and memory requirements. However, this approach processes blocks independently without adequate boundary overlap, resulting in significant blocking artifacts during coefficient encoding. Consequently, image coders based on BFrWF demonstrate inferior compression efficiency across all bitrates when compared to contemporary wavelet-based coders. Given the increasing demand for high-quality image transmission over Wireless Multimedia Sensor Networks (WMSNs) and IoT devices, the limitations of BFrWF render it an unsuitable choice for these platforms [8].

To address the limitations of BFrWF, we propose a lifting-based implementation of BFrWF termed LBFrWF. This approach not only retains the memory efficiency of BFrWF but also reduces its computational complexity by approximately 30% for every frequency frame of HS image. Crucially, by incorporating proper overlap processing at block boundaries, LBFrWF successfully eliminates blocking artifacts. The transform coefficients produced by LBFrWF are identical to those of the conventional DWT. To demonstrate its compatibility with modern coding standards, the coefficients were encoded using the advance wavelet transform based coders, a low memory state of the art algorithm for wavelet-based HS image compression algorithms.

The lifting scheme is an efficient, low-complexity method for computing the wavelet coefficients. It operates through four stages: split, predict, update, and scaling. Initially, the input signal is split into even and odd indexed samples. The predict step then generates the high-pass (known as H sub-band) or detail coefficients, which are subsequently scaled. Conversely, the update step produces the low-pass (kno]wn as L sub-band) or approximation coefficients, which are also scaled to complete the transformation [11].

III.2 2D-LOW COMPLEXITY ZERO MEMORY SET PARTITIONED EMBEDDED BLOCK (2D-LC-ZM-SPECK)

The 2D Low Complexity Zero Memory Set Partitioned Embedded algorithm [12] is a reduced complexity variant of the 2D-ZM-SPECK compression algorithm which is used for every frequency frame for the encoding/decoding process. Its efficiency stems from a pre-computation step during the transform phase, where the maximum coefficient magnitude for each sub-band is identified and stored. This approach drastically reduces the memory footprint, as only a single value per sub-band must be retained. For an L level BFrWF transform of a single frequency frame with spatial dimensions, the required coding memory is precisely '8N x (3L + 1)' bytes for 'N x (3L + 1)' coefficients. The encoding process initiates after this pre-computation. From a hierarchical perspective, the detail subbands are categorized into three orientation groups HL_n, LH_n, and HH_n for n = 1, 2, ..., L-1 and are systematically arranged into different sets denoted as as $\boldsymbol{S_\theta^n}$ and $\boldsymbol{I^n}$. The algorithm processes these sets for each frequency frame iteratively over a series of decreasing thresholds.

IV. ENCODING PROCESS OF PROPOSED ALGORITHM

The proposed hyperspectral image compression algorithm begins by transforming each frequency frame of unprocessed HS image into the frequency domain using a 2D LBFrWF (5-level) transform, a step that reduces the memory required for the transformed data. A Modified HyperSpectral (MHS) image cube, equal in size to the original HS image, is then constructed from groups of eight consecutive frames. Within each group, the first frequency frame remains unchanged, while each subsequent frame (2 through 8) is generated as the difference between the current HS frame and the previous one. This grouping and differential framing process repeats for the entire HS cube. The MHS cube is subsequently converted into a one-dimensional array via linear indexing scheme, and each of its frequency frames of MHS image cube is encoded using the 2D-LC-ZM-SPECK compression coder. This encoding proceeds frame by frame according to bit planes until a specified bit budget is consumed.

To evaluate the efficacy of the proposed Hyperspectral Image Compression Algorithm, a comprehensive comparative assessment is conducted using four distinct hyperspectral image datasets. This study benchmarks the performance of HSICA against nine existing compression algorithms including recent state of art compression algorithms, with a focused analysis on critical performance indicators including coding efficiency, memory utilization, and computational complexity. For the simulation, the MHS image cube was cropped to a size of '128 x 128' pixels with 128 spectral frequency frames. All simulation tests were conducted in MATLAB (simulation software) on a Windows 11 system with 20 GB of RAM and a 1.6 GHz processor.

A five-level Lifting based Block Fractional Wavelet Filter was applied to the spatial dimensions of each 2D image frame. A separate 1D wavelet transform was then applied along the spectral dimension. The resulting transform coefficients were quantized to integers and linearly indexed into a 1D array for encoding. The following compression algorithms were evaluated: 3D-SPECK (CA 1) [6], 3D-LSK (CA 2) [14], 3D-SPIHT (CA 3) [7], 3D-NLS (CA 4) [15], 3D-WBTC (CA 5) [16], 3D-LMBTC (CA 6) [17], 3D-ZM-SPECK (CA 7) [18], Fractional Wavelet Filter based 2D-ZM-SPECK (CA 8) [9], Block Fractional Wavelet Filter based 2D-LC-ZM-SPECK (CA 9) [8], and the proposed Lifting Block Fractional Wavelet Filter based 2D-LC-ZM-SPECK.

IV.1 TRANSFORM MEMORY AND COMPLEXITY

As evidenced by the data in Table 1, the Lifting based Block Fractional Wavelet Filter demonstrates the most efficient transform memory requirements. This efficiency is achieved through a buffering system that requires only a single input buffer to store a segment of the frequency frame and four additional buffers to compute the final four sub-bands [19]. For an HS image frequency frame of size 'N × N' divided into 'b' blocks, the input buffer dimension is $1 \times \frac{N}{b/2}$ while the remaining buffers have a dimension of $1 \times \frac{N}{b}$.

This minimal memory footprint, confirmed in Table 2, stands in stark contrast to the requirements of other transforms. Conventional 3D-DWT and 2D-DWT methods necessitate storing the entire HS image cube or complete frequency frames, respectively, leading to significantly higher memory demands [2], [20]. The use of fractional wavelet filters substantially reduces this need, an advantage that is further amplified by the BFrWF [21]. Consequently, the total transform memory required by the BFrWF is minimized to a level comparable with the near-zero coding memory of subsequent stages.

Table 1: Comparative Analysis between the different type state of art wavelet transform with Block Based Fractional Wavelet Transform on wavelet transform memory requirement.

Dimension of HS Image Cube / Type of Wavelet Transform	128	256	512
3D-Dyadic Wavelet Transform	38.34 MB	306.72 MB	2453.76 MB
2D- Dyadic Wavelet Transform	174.592 KB	698.368 KB	2793.472 KB
Fractional Wavelet Transform	3.123 KB	6.246 KB	12.493 KB
Block Based Fractional Wavelet Transform	1.5615 KB	3.123 KB	6.246 KB
Lifting based BFrWF	1.6125 KB	3.25 KB	6.4375 KB

Source: Authors, (2026).

Table 2: Average time requirement (for calculation of the transform complexity) of the different wavelet transform for different HS image size.

Dimension of HS Image Cube / Type of Wavelet Transform	256	512
3D-Dyadic Wavelet Transform	2.96 sec	18.47 sec
2D- Dyadic Wavelet Transform	5.65 sec	44.54 sec
Fractional Wavelet Transform	7.59 sec	47.24 sec
Block Based Fractional Wavelet Transform (4)	6.01 sec	37.06 sec
Lifting based BFrWF (LBFrWF)	5.22 sec	35.21 sec

Source: Authors, (2026).

Regarding computational complexity, measured as the time required to calculate transform coefficients, the BFrWF offers a lower complexity than the FrWF. However, as indicated in Table 2, it retains a higher complexity than standard 2D-DWT and 3D-DWT. It is important to note that 3D-DWT inherently benefits from a lower complexity than its 2D counterparts applied frame-by-frame. The complexity of the BFrWF decreases relative to 2D-DWT as the size of the HS image increases. The unique computation methodology of the BFrWF coefficients is the primary reason for its overall efficiency profile.

IV.2 CODING EFFICIENCY

A comparative analysis of the proposed algorithm and contemporary state-of-the-art compression techniques was conducted by calculating the average Rate-Distortion (RD) performance over a suite of HS images. The evaluation employs two complementary metrics: PSNR, a widely adopted objective measure of reconstruction error, and the Structural Similarity Index, which provides a quantitative assessment of perceptual quality by analyzing the preservation of structural details, edges, and textures [22], [23]. As shown in Tables 3 and 4, the proposed compression algorithm demonstrates superior RD performance and Structural Similarity scores, outperforming all other benchmarked algorithms. Its performance is virtually equivalent to the FrWF based 2D-ZM-SPECK and BFrWF based 2D-LC-ZM-SPECK compression algorithms, which are its closest competitor. This high efficiency is attributed to the algorithm's ability to identify and represent a large number of insignificant coefficients with a single bit during the early bit-plane encoding passes.

IV.3 CODING MEMORY

Coding memory is essential in hyperspectral image compression for tracking the significance of coefficients during bit-plane coding [2], [24]. In contrast to conventional methods, the proposed algorithm necessitates only a fixed, minimal memory allocation to store the global maximum magnitude value of the transformed image coefficients. This single value is efficiently shared with the decoder, eliminating the need for extensive bookkeeping data. The results in Table 5 demonstrate that only two other algorithms have a marginally lower memory requirement.

IV.4 CODING COMPLEXITY

A key advantage of the proposed compression algorithm is its reduction of coding complexity. This is achieved by storing the maximum magnitude of each set of transformed coefficients in a fixed memory allocation [18]. This method offers a significant efficiency improvement over the FrWF-based ZM-SPECK algorithm, whose requirement to check every coefficient against every threshold causes computational complexity, power demand, and processing time to increase substantially. Although derived from ZM-SPECK, the proposed algorithm implements a more efficient process. The resulting performance gains in encoding and decoding time are presented in Tables 6 and 7. Encoding time (the time to encode coefficients within a bit budget) is reduced.

Decoding is inherently less complex and thus faster, as the decoder does not perform the per coefficient threshold checks required in each bit plane during encoding. The visual representation of the HS image and reconstruct HS image after the compression process for Uncalibrated Yellowstone Scene 0 and Uncalibrated Yellowstone Scene 11 at CR = 1 is shown in Fig 1 and Fig 2. Fig 3 shows the 75th frame reconstruction at the four different bit rates. It has been clear from Fig 3 that there is significant improvement of the image quality of reconstructed HS image at the higher bit rates.

Table 3: Structural Similarity (SSIM) index between different HSICAs at fifteen different bit rates for YSUS 0.

Bit Rate	CA 1 [6]	CA 2 [14]	CA 3 [7]	CA 4 [4]	CA 5 [16]	CA 6 [17]	CA 7 [18]	CA 8 [9]	CA 9 [8]	Proposed Compression Algorithm
	YSUS 0 HS Image									
0.00625	0.39	0.38	0.38	0.38	0.39	0.38	0.37	0.41	0.4	0.41
0.0125	0.44	0.43	0.43	0.43	0.47	0.43	0.43	0.44	0.44	0.44
0.025	0.56	0.55	0.55	0.53	0.57	0.53	0.53	0.61	0.61	0.61
0.0375	0.62	0.62	0.61	0.61	0.62	0.61	0.62	0.67	0.66	0.65
0.05	0.66	0.65	0.65	0.65	0.66	0.65	0.65	0.71	0.7	0.7
0.1	0.75	0.75	0.75	0.74	0.75	0.74	0.75	0.8	0.79	0.8
0.2	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.87	0.86	0.86
0.3	0.87	0.88	0.87	0.87	0.87	0.87	0.87	0.9	0.89	0.89
0.4	0.91	0.9	0.9	0.9	0.9	0.9	0.89	0.92	0.92	0.91
0.5	0.93	0.93	0.92	0.92	0.93	0.92	0.92	0.94	0.94	0.94
0.6	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.96	0.96	0.96
0.7	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.97	0.97	0.97
0.8	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.98	0.98	0.98
0.9	0.97	0.97	0.97	0.97	0.97	0.97	0.96	0.98	0.98	0.98
1	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.98

Source: Authors, (2026).

Table 4: Numeric values of PSNR for proposed compression algorithm with the nine other on four different HS image dataset.

Bit Rate	CA 1 [6]	CA 2 [14]	CA 3 [7]	CA 4 [4]	CA 5 [16]	CA 6 [17]	CA 7 [18]	CA 8 [9]	CA 9 [8]	Proposed Compression Algorithm
	YSUS 0 HS Image									
0.00625	26.4	26.9	26.3	26.3	26.4	26.9	26.9	30.2	30.1	30.2
0.0125	27.6	27.5	27.4	27.4	27.6	27.5	27.5	31.5	31.4	31.4
0.025	29	28.4	28.8	28.3	29.1	28.3	28.4	32.8	32.7	32.7
0.0375	30	29.9	29.8	29.8	29.9	30	30	34.2	34.2	34.3
0.05	30.9	30.5	30.7	30.6	30.8	30.5	30.5	34.7	34.7	34.8
0.1	32.8	32.8	32.7	32.7	32.8	32.8	32.8	37	37	37.1
0.2	35.1	34.6	35.1	34.5	35.2	34.6	34.6	39.8	39.7	39.8
0.3	36.9	36.7	36.8	36.7	36.9	36.6	36.6	41.5	41.5	41.5
0.4	38.4	37.6	38.2	37.8	38.3	37.5	37.5	43.3	43.3	43.3
0.5	39.5	39.5	39.4	39.3	39.5	39.1	39.1	45	45	45
0.6	40.8	40.6	40.7	40.6	40.8	40.5	40.5	45.7	45.7	45.7
0.7	41.9	41.3	41.8	41.8	41.9	41.2	41.2	47	46.9	46.9
0.8	43	42.3	42.9	42.5	43.0	42.1	42.1	48.1	48	48
0.9	44	43.8	43.8	43.5	44.0	43.0	43	49.4	49.4	49.4
1	44.9	44.9	44.7	44.7	44.8	44.8	44.8	49.9	49.8	49.8
	YSUS 3 HS Image									
0.00625	30.5	30.5	30.3	29.8	30.5	30.5	30.5	32.4	32.3	32.4
0.0125	31.4	31.3	31.3	31.2	31.4	31.3	31.3	33.9	33.8	33.9
0.025	32.9	32.9	32.7	32.6	32.9	32.9	33	34.1	34	34.1
0.0375	34.0	33.8	33.8	33.8	34	33.8	33.8	36.7	36.6	36.7
0.05	34.9	34.7	34.6	34.4	34.8	34.6	34.7	37.9	37.9	37.9
0.1	37.2	37.1	37	37	37.2	36.9	36.9	41	41	41
0.2	40.4	40.1	40.2	40.1	40.4	40.1	40.1	44.2	44.1	44.2
0.3	42.9	42.8	42.7	42.4	42.9	42.4	42.5	46.9	46.8	46.9
0.4	45	44.6	44.7	44.6	45	44.5	44.5	49.2	49.2	49.2
0.5	47	46.7	46.8	46.3	47	46.1	46.2	51.5	51.5	51.5
0.6	48.9	48.9	48.5	48.5	48.9	48.9	48.9	53.7	53.6	53.7
0.7	50.7	50.2	50.3	50.2	50.7	50	50	54.8	54.8	54.8
0.8	52.1	52.2	51.8	51.7	52	51.4	51.4	56.4	56.3	56.4
0.9	53.8	53.8	53.4	53.4	53.8	53.8	53.8	58.5	58.5	58.5
1	55.4	55.1	55.0	54.9	55.3	54.9	54.9	59.9	59.8	59.9
	YSUS 10 HS Image									
0.00625	26.5	26.2	26.2	26.5	26.5	26.1	26.1	28.9	28.8	28.8
0.0125	27.6	27.5	27.5	27.4	27.5	27.5	27.5	29.6	29.6	29.6
0.025	28.5	28.2	28.2	28.2	28.7	28.2	28.3	30.8	30.7	30.7
0.0375	29.4	29.3	29.3	29.1	29.3	29.4	29.4	31.6	31.6	31.6
0.05	30.0	29.8	29.8	29.8	30	29.8	29.8	32.5	32.5	32.5
0.1	31.5	31.4	31.4	31.3	31.5	31.4	31.5	34.1	34	34.1
0.2	33.4	33.2	33.2	33.2	33.5	33.1	33.2	36.7	36.7	36.7
0.3	34.9	34.7	34.7	34.5	34.9	34.5	34.5	38.8	38.8	38.8
0.4	36.1	35.9	35.9	36	36.1	35.9	35.9	40.3	40.3	40.3
0.5	37.3	36.8	36.8	37	37.3	36.7	36.7	41.7	41.6	41.6

0.6	38.5	37.9	37.9	38	38.5	37.8	37.9	43.6	43.6	43.6
0.7	39.5	39.5	39.5	39.4	39.5	39.3	39.3	44.3	44.3	44.3
0.8	40.5	40.3	40.3	40.3	40.5	40.3	40.3	45	44.9	44.9
0.9	41.6	41.1	41.1	41.3	41.6	40.9	40.9	45.8	45.8	45.8
1	42.6	42.1	42.1	42.3	42.6	41.7	41.7	46.8	46.7	46.7
	YSUS 11 HS Image									
0.00625	30.4	30.4	30.2	30.2	30.4	30.3	30.4	31.9	31.9	31.9
0.0125	31.7	31.6	31.6	31.5	31.7	31.5	31.5	33.1	33	33.1
0.025	33	33	32.9	32.7	33	32.9	33	34.4	34.3	34.4
0.0375	34.1	33.9	33.9	33.8	34.1	33.8	33.8	35.8	35.7	35.8
0.05	34.9	34.4	34.6	34.5	34.8	34.4	34.4	36.7	36.7	36.7
0.1	36.9	36.6	36.7	36.7	36.9	36.5	36.6	39.1	39	39.1
0.2	38.9	38.8	38.8	38.8	38.9	38.8	38.8	42.2	42.1	42.2
0.3	40.5	40.1	40.4	40.2	40.5	40	40	44	44	44
0.4	41.9	41.7	41.8	41.5	41.9	41.3	41.4	46	45.9	46
0.5	43.2	43.1	43.1	43	43.2	43	43.0	47.7	47.7	47.7
0.6	44.5	43.9	44.3	44.3	44.4	43.8	43.8	48.6	48.6	48.6
0.7	45.6	45.1	45.4	45.1	45.7	44.8	44.8	50.1	50.1	50.1
0.8	46.7	46.7	46.6	46.5	46.7	46.1	46.1	51.5	51.5	51.5
0.9	47.8	47.8	47.7	47.7	47.8	47.8	47.8	52.1	52.1	52.1
1	48.8	48.5	48.6	48.5	48.8	48.3	48.3	53.1	53.1	53.1

Source: Authors, (2026).

Table 5: Coding memory requirement of nine compression algorithm for four different HS image datasets.

Bit Rate	**CA 1 [6]**	**CA 2 [14]**	**CA 3 [7]**	**CA 4 [4]**	**CA 5 [16]**	**CA 6 [17]**	**CA 7 [18]**	**CA 8 [9]**	**CA 9 [8]**	**Proposed Compression Algorithm**
	YSUS 0 HS Image									
0.00625	18	512	15	1024	8	12	0	0	0.018	0.018
0.0125	31	512	36	1024	23	12	0	0	0.018	0.018
0.025	62	512	71	1024	53	12	0	0	0.018	0.018
0.0375	109	512	111	1024	100	12	0	0	0.018	0.018
0.05	109	512	117	1024	108	12	0	0	0.018	0.018
0.1	237	512	249	1024	208	12	0	0	0.018	0.018
0.2	494	512	513	1024	489	12	0	0	0.018	0.018
0.3	597	512	615	1024	578	12	0	0	0.018	0.018
0.4	843	512	896	1024	844	12	0	0	0.018	0.018
0.5	1224	512	1222	1024	1184	12	0	0	0.018	0.018
0.6	1329	512	1354	1024	1230	12	0	0	0.018	0.018
0.7	1456	512	1461	1024	1356	12	0	0	0.018	0.018
0.8	1653	512	1680	1024	1554	12	0	0	0.018	0.018
0.9	1892	512	1906	1024	1876	12	0	0	0.018	0.018
1	2010	512	2072	1024	2022	12	0	0	0.018	0.018
	YSUS 3 HS Image									
0.00625	16	512	17	1024	8	12	0	0	0.018	0.018
0.0125	26	512	28	1024	20	12	0	0	0.018	0.018
0.025	71	512	73	1024	63	12	0	0	0.018	0.018
0.0375	78	512	81	1024	72	12	0	0	0.018	0.018
0.05	117	512	124	1024	109	12	0	0	0.018	0.018
0.1	205	512	204	1024	186	12	0	0	0.018	0.018
0.2	397	512	414	1024	378	12	0	0	0.018	0.018
0.3	678	512	682	1024	621	12	0	0	0.018	0.018
0.4	726	512	753	1024	708	12	0	0	0.018	0.018
0.5	947	512	947	1024	894	12	0	0	0.018	0.018
0.6	1119	512	1155	1024	1091	12	0	0	0.018	0.018
0.7	1234	512	1243	1024	1121	12	0	0	0.018	0.018
0.8	1343	512	1324	1024	1282	12	0	0	0.018	0.018
0.9	1439	512	1473	1024	1400	12	0	0	0.018	0.018
1	1498	512	1517	1024	1440	12	0	0	0.018	0.018
	YSUS 10 HS Image									
0.00625	19	512	15	1024	11	12	0	0	0.018	0.018
0.0125	33	512	37	1024	34	12	0	0	0.018	0.018
0.025	65	512	71	1024	64	12	0	0	0.018	0.018
0.0375	112	512	116	1024	101	12	0	0	0.018	0.018
0.05	163	512	123	1024	113	12	0	0	0.018	0.018
0.1	276	512	289	1024	282	12	0	0	0.018	0.018
0.2	448	512	489	1024	446	12	0	0	0.018	0.018

0.3	822	512	833	1024	826	12	0	0	0.018	0.018
0.4	920	512	881	1024	891	12	0	0	0.018	0.018
0.5	1038	512	1055	1024	1037	12	0	0	0.018	0.018
0.6	1352	512	1389	1024	1355	12	0	0	0.018	0.018
0.7	1634	512	1561	1024	1592	12	0	0	0.018	0.018
0.8	1681	512	1652	1024	1634	12	0	0	0.018	0.018
0.9	1741	512	1710	1024	1712	12	0	0	0.018	0.018
1	1825	512	1813	1024	1824	12	0	0	0.018	0.018
	YSUS 11 HS Image									
0.00625	15	512	16	1024	11	12	0	0	0.018	0.018
0.0125	26	512	29	1024	27	12	0	0	0.018	0.018
0.025	69	512	74	1024	71	12	0	0	0.018	0.018
0.0375	79	512	87	1024	80	12	0	0	0.018	0.018
0.05	112	512	121	1024	112	12	0	0	0.018	0.018
0.1	196	512	198	1024	196	12	0	0	0.018	0.018
0.2	467	512	486	1024	469	12	0	0	0.018	0.018
0.3	634	512	669	1024	633	12	0	0	0.018	0.018
0.4	993	512	1013	1024	998	12	0	0	0.018	0.018
0.5	1074	512	1102	1024	1076	12	0	0	0.018	0.018
0.6	1112	512	1102	1024	1112	12	0	0	0.018	0.018
0.7	1409	512	1431	1024	1407	12	0	0	0.018	0.018
0.8	1564	512	1560	1024	1547	12	0	0	0.018	0.018
0.9	1689	512	1678	1024	1640	12	0	0	0.018	0.018
1	1789	512	1821	1024	1790	12	0	0	0.018	0.018

Source: Authors, (2026).

Table 6: Encoding time (sec) requirement for the complexity analysis of nine different compression algorithms with the proposed compression algorithm for four different datasets.

Bit Rate	**CA 1 [6]**	**CA 2 [14]**	**CA 3 [7]**	**CA 4 [4]**	**CA 5 [16]**	**CA 6 [17]**	**CA 7 [18]**	**CA 8 [9]**	**CA 9 [8]**	**Proposed Compression Algorithm**
	YSUS 0 HS Image									
0.00625	1.1	0.2	1.3	0.3	1.1	0.3	0.6	0.6	0.5	0.5
0.0125	1.4	0.4	1.1	0.6	1.3	1.2	0.7	0.7	0.6	0.6
0.025	2.8	0.5	1.5	0.7	1.7	1.5	0.9	0.8	0.7	0.7
0.0375	4.3	0.6	2.2	0.8	2.2	1.7	1	1.1	0.9	0.9
0.05	7.3	0.7	3.4	0.9	3.5	2	1.2	1.3	1	1
0.1	18.5	0.9	6.7	1	6.2	3.1	1.8	2.1	1.6	1.6
0.2	89.7	1.1	31	1.4	17.6	4.9	2.8	3.7	2.4	2.4
0.3	195	1.5	67	1.8	60.7	7.1	3.9	5.4	4.1	4.1
0.4	249	1.8	96	2.1	118	8.6	4.9	7	5.3	5.3
0.5	340	2.1	118	2.4	173	10	5.8	8.4	7.2	7.2
0.6	692	2.6	257	2.9	376	13	7.1	9.9	7.9	7.9
0.7	961	2.8	463	3.2	657	14.2	8.1	11.6	8.9	8.9
0.8	1167	3.0	491	3.5	759	15.8	9.2	13.4	11.2	11.2
0.9	1304	3.3	513	4.1	855	16.8	9.9	15.3	12.9	12.9
1	1441	3.8	560	4.9	878	17.4	10.9	20.6	16.6	16.6
	YSUS 3 HS Image									
0.00625	0.9	0.2	1.0	0.4	0.5	0.4	0.4	0.9	0.7	0.7
0.0125	1.6	0.3	1.1	0.6	1.3	1.1	0.8	1.7	0.9	0.9
0.025	2.9	0.4	1.6	0.7	1.8	1.4	0.9	2.1	1.2	1.2
0.0375	5.0	0.5	2.2	0.8	2.4	1.7	1.3	2.2	1.6	1.6
0.05	6.7	0.6	4.3	0.9	2.7	2.1	1.4	2.4	1.9	1.9
0.1	18.2	0.8	9.9	1.4	7.1	3.2	1.9	3.3	2.2	2.2
0.2	57.3	1.2	24.3	1.9	25.0	5.5	3	5.1	3.7	3.7
0.3	97.1	1.4	39.9	2.1	47.9	6.7	3.9	6.8	4.6	4.6
0.4	206	1.8	104	2.4	122	9.3	5.1	8.4	6.8	6.8
0.5	303	2.2	131	2.9	177	10.5	6.1	10.1	7.5	7.5
0.6	349	2.5	141	3.2	190	11.5	7.2	11.8	10.1	10.1
0.7	706	2.7	338	3.5	528	14.3	8.6	13.5	11.5	11.5
0.8	835	3.0	408	3.8	601	15.7	9.5	15.2	12.8	12.8
0.9	981	3.5	414	4.5	633	16.8	10.7	16.9	14.1	14.1
1	1373	3.8	632	5.1	977	19.9	11.9	18.3	16.3	16.3
	YSUS 10 HS Image									
0.00625	1.8	0.2	0.6	0.4	0.8	0.9	0.4	0.6	0.5	0.5
0.0125	2.5	0.3	1.1	0.5	1.4	1.2	0.7	0.9	0.8	0.8
0.025	5.1	0.4	1.6	0.6	1.7	1.5	0.9	1.1	1	1

0.0375	5.6	0.5	2.2	0.7	2.1	1.7	1.1	1.3	1.1	1.1
0.05	8.5	0.6	3.3	0.8	3.5	2.1	1.2	1.4	1.2	1.2
0.1	18.8	0.7	8.1	0.9	5.9	3.2	1.7	2.2	1.8	1.8
0.2	83.5	1.1	28.9	1.3	23.6	5.3	3.7	3.7	3.6	3.6
0.3	110	1.3	50.3	1.7	32.5	6.6	3.9	5.3	4.2	4.2
0.4	326	1.7	181	2.0	199	9.2	5.3	7.2	5.5	5.5
0.5	472	2	241	2.3	294	11.5	7.2	10.8	8.9	8.9
0.6	588	2.2	264	2.5	363	11.3	8.4	11.4	9.5	9.5
0.7	678	2.5	281	2.7	398	12.5	7.8	12.6	10.6	10.6
0.8	1151	2.9	531	3.2	772	15.8	10	15.2	12.2	12.2
0.9	1745	3.1	883	3.4	1223	17.1	12	17.6	14.8	14.8
1	2558	3.3	1034	3.9	1517	18.3	13.7	18.5	15.8	15.8
	YSUS 11 HS Image									
0.00625	1	0.2	0.7	0.3	0.6	0.9	0.5	0.5	0.4	0.4
0.0125	1.6	0.3	1.0	0.4	2.3	1.8	0.9	0.6	0.5	0.5
0.025	3.0	0.4	1.5	0.5	3.1	1.7	0.9	0.8	0.6	0.6
0.0375	5.5	0.5	2.1	0.6	6.0	2.1	1.2	1.1	1	1
0.05	7.4	0.6	2.9	0.7	15.4	2.5	1.3	1.3	1.1	1.1
0.1	19.7	0.7	7.8	0.9	17.8	3.6	2.1	2.1	1.9	1.9
0.2	81.4	1.1	21.3	1.3	29.1	6.7	3.2	4.4	4.1	4.1
0.3	142	1.4	67.0	1.7	70.6	7.7	4.1	6.7	5.3	5.3
0.4	227	1.8	77.7	2.0	99.8	9.3	4.9	7.8	6.6	6.6
0.5	509	2.1	227	2.4	280	12.2	7.2	9.3	8.1	8.1
0.6	755	2.3	405	2.8	495	14.2	7.6	11.1	9.4	9.4
0.7	981	2.5	424	3.0	578	14.5	8.5	12.2	10.6	10.6
0.8	1018	2.8	449	3.2	646	14.7	9.1	16.5	11.3	11.3
0.9	1329	3.3	447	3.5	738	17.9	10.2	18.2	14.9	14.9
1	2073	3.6	873	4.1	1291	19.9	11.4	24.3	16.3	16.3

Source: Authors, (2026).

Table 7: Decoding time (sec) requirement for the complexity analysis of nine different compression algorithms with the proposed compression algorithm for four different datasets.

Bit Rate	**CA 1 [6]**	**CA 2 [14]**	**CA 3 [7]**	**CA 4 [4]**	**CA 5 [16]**	**CA 6 [17]**	**CA 7 [18]**	**CA 8 [9]**	**CA 9 [8]**	**Proposed Compression Algorithm**
	YSUS 0 HS Image									
0.00625	0.7	0.1	0.9	0.2	0.7	0.3	0.36	0.46	0.33	0.33
0.0125	0.7	0.3	0.3	0.4	0.4	0.5	0.7	0.5	0.4	0.4
0.025	1.5	0.4	0.7	0.8	0.7	0.7	0.8	0.7	0.6	0.6
0.0375	2.8	0.4	1.2	0.6	0.9	0.8	1	1	0.8	0.8
0.05	5.1	0.5	2.2	0.6	2.3	1	1.1	1.3	0.9	0.9
0.1	12.9	0.7	5.3	0.8	4.6	1.8	1.5	1.8	1.5	1.5
0.2	61.4	1.0	23.9	1.2	15.4	3.4	2.7	3.2	2.6	2.6
0.3	136	1.1	52.4	1.6	57.9	4.7	3.5	5	3.3	3.3
0.4	204	1.7	87.9	1.9	108	6.2	4.3	6.5	4.9	4.9
0.5	306	1.9	98.9	2.2	148	7.8	5.2	8	6.6	6.6
0.6	600	2.2	217	2.5	341	9.2	6.7	9.3	7.1	7.1
0.7	906	2.4	456	2.7	618	10.4	7.7	11.1	8.2	8.2
0.8	1005	2.5	486	3	727	13.8	8.5	12.5	10.4	10.4
0.9	1249	3.1	501	3.3	830	13.3	9.4	14.5	12.5	12.5
1	1389	3.6	512	4	832	14.5	10.6	19	15.1	15.1
	YSUS 3 HS Image									
0.00625	0.8	0.5	0.7	0.3	0.2	0.3	0.3	0.5	0.4	0.4
0.0125	0.9	0.3	0.8	0.5	0.4	0.5	0.6	0.7	0.7	0.7
0.025	1.6	0.4	1.2	0.6	0.6	0.7	0.8	0.9	0.8	0.8
0.0375	3.5	0.4	1.5	0.7	1.2	0.9	1.1	1.1	1	1
0.05	4.6	0.5	2.7	0.8	1.6	1.1	1.2	1.4	1.2	1.2
0.1	14.5	0.7	6.6	1.1	9.5	1.9	1.5	2.3	1.9	1.9
0.2	49.4	1	22.3	1.5	17.6	3.5	2.6	4.1	3.2	3.2
0.3	81.9	1.4	31.6	1.8	40.1	5.1	3.5	5.8	4.2	4.2
0.4	190	1.7	90.4	2.2	110	6.7	4.8	7.6	6.5	6.5
0.5	284	2.0	118	2.5	156	8.1	5.9	9.4	6.9	6.9
0.6	321	2.4	128	2.9	176	9.4	6.9	11.1	8.6	8.6
0.7	684	2.5	309	3.2	577	11.0	8.3	13	10.2	10.2
0.8	822	2.8	373	3.4	553	12.5	9.3	14.7	11.3	11.3
0.9	860	3.2	400	3.8	589	13.9	10.3	16.2	14.1	14.1
1	1258	3.4	589	4	849	15.6	11.5	18.1	15.7	15.7

	YSUS 10 HS Image									
0.00625	0.8	0.1	0.3	0.3	0.2	0.5	0.3	0.4	0.4	0.4
0.0125	1.9	0.2	0.3	0.4	0.4	0.6	0.6	0.6	0.6	0.6
0.025	2.3	0.3	0.7	0.5	0.7	0.7	0.8	0.8	0.7	0.7
0.0375	2.8	0.4	1.2	0.6	0.9	0.8	1	1	0.9	0.9
0.05	5.9	0.5	2.2	0.7	2.3	1	1.2	1.1	1	1
0.1	12.8	0.6	6.5	0.8	4.4	1.8	1.6	1.9	1.7	1.7
0.2	62.2	1	25.8	1.1	21.6	3.3	2.8	3.2	3.1	3.1
0.3	80.0	1.2	47.3	1.5	29.2	4.9	4	5.1	3.9	3.9
0.4	307	1.4	172	1.9	175	6.7	5.5	7	5.9	5.9
0.5	441	1.7	219	2.1	262	7.6	7.8	9.6	7.5	7.5
0.6	555	1.9	230	2.2	319	9.0	6.6	10.1	8.1	8.1
0.7	609	2.2	246	2.5	359	10.5	7.5	12	9.6	9.6
0.8	1080	2.5	496	2.8	717	11.8	9.2	14.3	11.4	11.4
0.9	1692	2.9	818	3.1	1130	13.3	10	16.5	12.9	12.9
1	2179	3.4	930	3.4	1384	14.7	11.5	17.2	14.8	14.8
	YSUS 11 HS Image									
0.00625	0.66	0.2	0.2	0.2	0.2	0.5	0.4	0.4	0.3	0.3
0.0125	0.93	0.3	0.3	0.4	0.8	0.6	0.7	0.5	0.4	0.4
0.025	1.63	0.4	0.6	0.4	0.9	0.8	0.8	0.7	0.6	0.6
0.0375	3.74	0.4	1.2	0.5	3.3	1.2	1.1	0.9	0.8	0.8
0.05	4.9	0.5	1.9	0.6	4.1	1.7	1.2	1.1	1	1
0.1	15.8	0.7	6.5	0.8	11.5	2.3	1.9	1.9	1.5	1.5
0.2	61.1	1.0	16.2	1.1	20.9	3.9	2.8	4.1	3.5	3.5
0.3	118	1.3	56.1	1.6	54.1	5.7	3.9	5.5	4.6	4.6
0.4	195	1.7	70.2	1.9	89.4	7.3	4.8	6.5	5.9	5.9
0.5	400	2.0	216	2.2	250	9.1	6.1	8	6.7	6.7
0.6	847	2.2	394	2.5	471	11.3	7.3	10.2	8.9	8.9
0.7	840	2.3	419	2.9	575	12.6	8.2	11.8	9.8	9.8
0.8	1059	2.6	436	3.1	621	13.4	8.7	15.6	10.6	10.6
0.9	1186	2.9	440	3.3	706	14.2	9.5	17.8	12.3	12.3
1	1901	3.4	867	3.9	1260	17.4	10.6	20.9	14.6	14.6

Source: Authors, (2026).

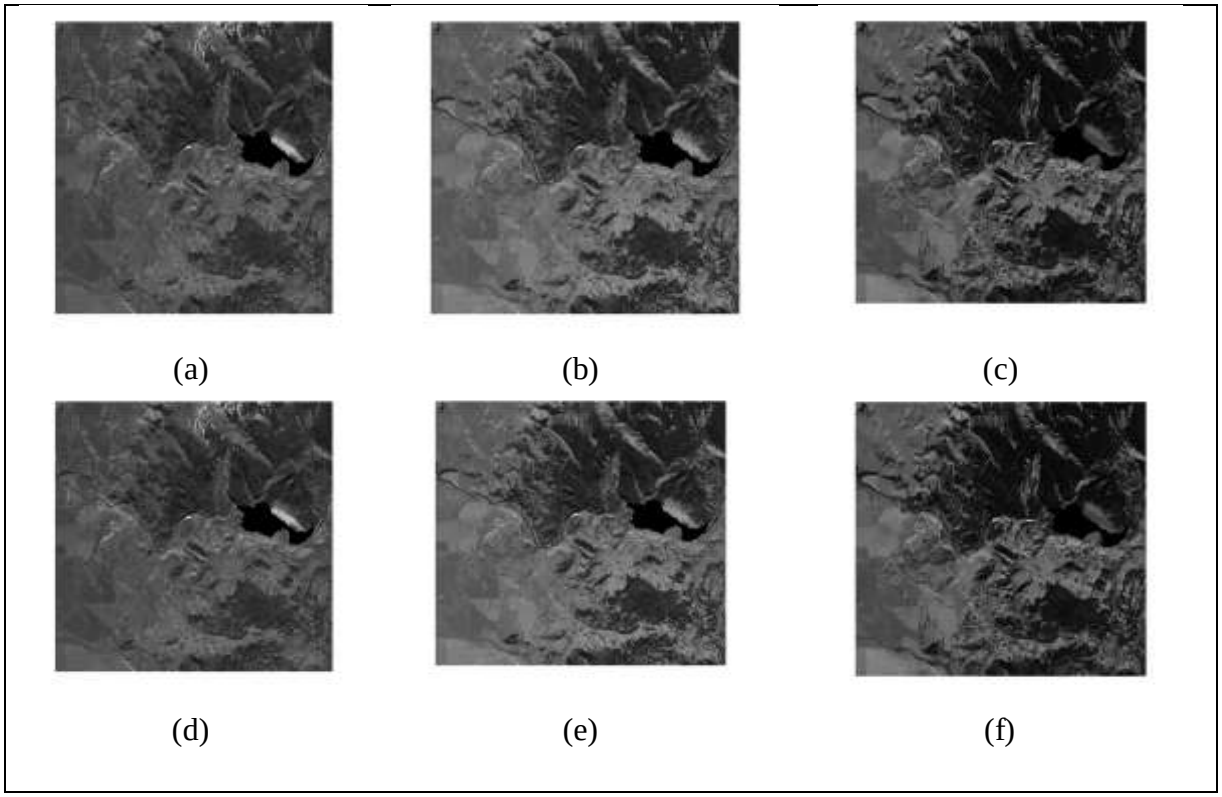

Figure 1: Original Uncalibrated Yellowstone Scene 0 (a) Frame 50 (b) Frame 100 (c) Frame 150 (d) Frame 200 Reconstructed Uncalibrated Yellowstone Scene 0 with CR=16 (d) Frame 50 (e) Frame 100 (f) Frame 150.
Source: Authors, (2026).

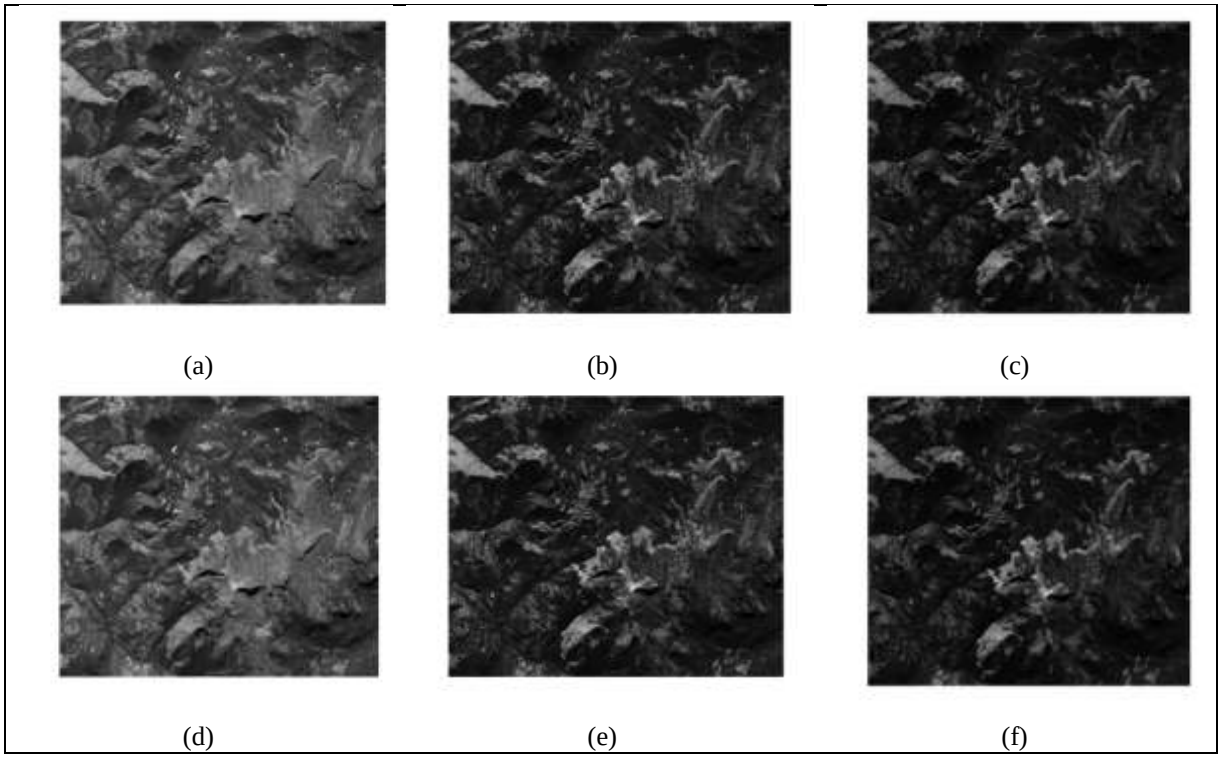

Figure 2: Original Uncalibrated Yellowstone Scene 11 (a) Frame 100 (b) Frame 150 (c) Frame 200 Reconstructed Uncalibrated Yellowstone Scene 11 with CR=16 (d) Frame 100 (e) Frame 150 (f) Frame 200.
Source: Authors, (2026).

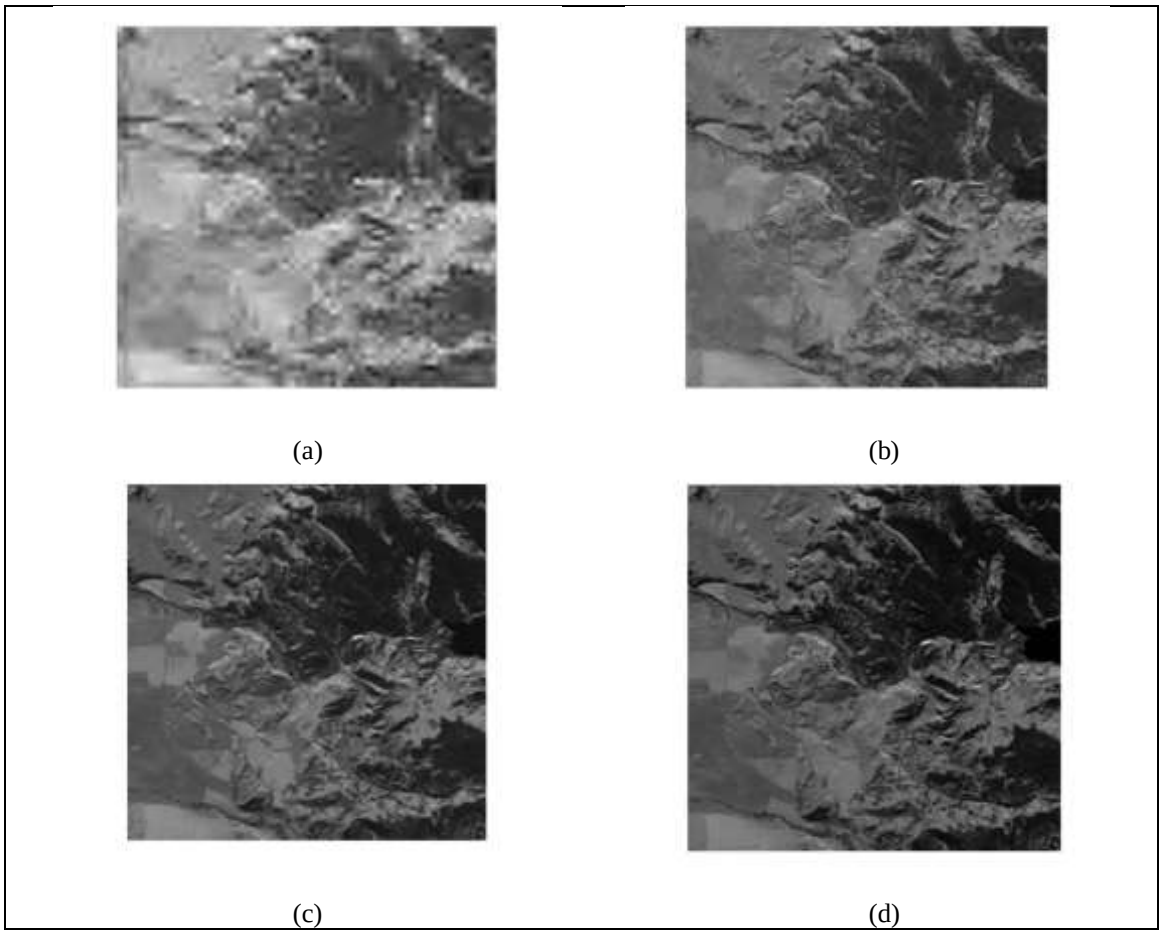

Figure 3: Reconstructed Uncalibrated Yellowstone Scene 0 with four bit rate (bpppb)
(a) 0.001 (b) 0.01 (c) 0.1 (d) 1.
Source: Authors, (2026).

V. CONCLUSION

This study introduces a lossy compression algorithm for HS images that is optimized for low memory and computational complexity, targeting resource-constrained onboard sensors. The proposed method enhances processing speed by minimizing memory access operations and capitalizes on the strong spectral-spatial correlations inherent in HS data to achieve high coding efficiency. The LBrWF is employed to reduce transform memory requirements, while the 2D-LC-ZM-SPECK algorithm eliminates coding memory and lowers coding complexity, leading to significantly faster compression compared to other transform-based techniques. Although the proposed algorithm operates in a listless manner, it exhibits a modest increase in computational time relative to existing listless coders. This is due to its frame-by-frame search for significant coefficients, which introduces a slight increment in processing overhead.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Purushottam Lal Nagar, Shrish Bajpai.
Methodology: Purushottam Lal Nagar, Shrish Bajpai.
Investigation: Purushottam Lal Nagar, Shrish Bajpai.
Discussion of results: Purushottam Lal Nagar, Shrish Bajpai.
Writing – Original Draft: Purushottam Lal Nagar.
Writing – Review and Editing: Shrish Bajpai
Resources: Purushottam Lal Nagar, Shrish Bajpai
Supervision: Shrish Bajpai
Approval of the final text: Purushottam Lal Nagar, Shrish Bajpai

VII. ACKNOWLEDGMENTS

We are sincerely thankful to the anonymous reviewers for their critical comments and suggestions to improve the quality of the paper. Authors want to express his gratitude to Integral University, Lucknow, Uttar Pradesh, India for providing manuscript number IU/R&D/2026-MCN0004264 for the present work.

VIII. REFERENCES

[1] Chutia, D., Bhattacharyya, D. K., Sarma, K. K., Kalita, R., & Sudhakar, S. (2016). Hyperspectral remote sensing classifications: a perspective survey. Transactions in GIS, 20(4), 463-490. doi : 10.1111/tgis.12164.

[2] Bajpai, S. (2024). 3D-listless block cube set-partitioning coding for resource constraint hyperspectral image sensors. Signal, Image and Video Processing, 18(4), 3163-3178. doi : 10.1007/s11760-023-02979-0.

[3] Sharma, D. (2024, November). Image Quality Assessment Metrics for Hyperspectral Image Compression Algorithms. In 2024 Second International Conference Computational and Characterization Techniques in Engineering & Sciences (IC3TES) (pp. 1-5). IEEE. doi : 10.1109/IC3TES62412.2024.10877577.

[4] Dua, Y., Kumar, V., & Singh, R. S. (2020). Comprehensive review of hyperspectral image compression algorithms. Optical Engineering, 59(9), 090902-090902. doi : 10.1117/1.OE.59.9.090902.

[5] Altamimi, A., & Ben Youssef, B. (2024). Lossless and near-lossless compression algorithms for remotely sensed hyperspectral images. Entropy, 26(4), 316. doi : 10.3390/e26040316.

[6] Tang, X., & Pearlman, W. A. (2004, October). Lossy-to-lossless block-based compression of hyperspectral volumetric data. In 2004 International Conference on Image Processing, 2004. ICIP'04. (Vol. 5, pp. 3283-3286). IEEE. doi : 10.1109/ICIP.2004.1421815.

[7] Tang, X., & Pearlman, W. A. (2006). Three-dimensional wavelet-based compression of hyperspectral images. In Hyperspectral data compression (pp. 273-308). Boston, MA: Springer US. doi : 10.1007/0-387-28600-4_10.

[8] Rajesh, Bajpai S., & Kidwai, N. R. Block-Based Fractional Wavelet Filter For Low-Complexity Coding Algorithm of Hyperspectral Image With Memory Constrained Wireless Multimedia Sensor, SSRG International Journal of Electronics and Communication Engineering, vol. 12, no. 8, pp. 257-271, 2025. doi : 10.14445/23488549/IJECE-V12I8P123.

[9] Bajpai, S., & Kidwai, N. R. (2024). Fractional wavelet filter based low memory coding for hyperspectral image sensors. Multimedia Tools and Applications, 83(9), 26281-26306. doi : 10.1007/s11042-023-16528-x.

[10] Tausif, M., Khan, E., Hasan, M., & Reisslein, M. (2017, October). SFrWF: Segmented fractional wavelet filter based DWT for low memory image coders. In 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON) (pp. 593-597). IEEE. doi : 10.1109/UPCON.2017.8251116.

[11] Tausif, M., Khan, E., Hasan, M., & Reisslein, M. (2020). Lifting-Based Fractional Wavelet Filter: Energy-Efficient DWT Architecture for Low-Cost Wearable Sensors. Advances in Multimedia, 2020(1), 8823689. doi : 10.1155/2020/8823689.

[12] Tausif, M., Khan, E., & Pinheiro, A. (2023). Computationally efficient wavelet-based low memory image coder for WMSNs/IoT. Multidimensional Systems and Signal Processing, 34(3), 657-680. doi : 10.1007/s11045-023-00878-8.

[13] Tausif, M., Jain, A., Khan, E., & Hasan, M. (2021). Memory-efficient architecture for FrWF-based DWT of high-resolution images for IoMT applications. Multimedia Tools and Applications, 80(7), 11177-11199. doi : 10.1007/s11042-020-10258-0.

[14] Ngadiran, R., Boussakta, S., Sharif, B., & Bouridane, A. (2010, May). Efficient implementation of 3D listless SPECK. In International Conference on Computer and Communication Engineering (ICCCE'10) (pp. 1-4). IEEE. doi : 10.1109/ICCCE.2010.5556843.

[15] Sudha, V. K., & Sudhakar, R. (2013). 3D listless embedded block coding algorithm for compression of volumetric medical images. J Sci Ind Res, 72, 735-748.

[16] Bajpai, S., Kidwai, N. R., & Singh, H. V. (2019). 3D wavelet block tree coding for hyperspectral images. International Journal of Innovative Technology and Exploring Engineering, 8(6C), 64-68.

[17] Bajpai, S., Kidwai, N. R., Singh, H. V., & Singh, A. K. (2019). Low memory block tree coding for hyperspectral images. Multimedia Tools and Applications, 78(19), 27193-27209. doi : 10.1007/s11042-019-07797-6.

[18] Bajpai, S., Kidwai, N. R., Singh, H. V., & Singh, A. K. (2022). A low complexity hyperspectral image compression through 3D set partitioned embedded zero block coding. Multimedia Tools and Applications, 81(1), 841-872. doi : 10.1007/s11042-021-11456-0.

[19] Rein, S., & Reisslein, M. (2011). Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks. Ad Hoc Networks, 9(4), 482-496. doi : 10.1016/j.adhoc.2010.08.004.

[20] Panna, B., Kumar, S., & Jha, R. K. (2019). Image encryption based on block-wise fractional fourier transform with wavelet transform. IETE Technical Review, 36(6), 600-613. doi : 10.1080/02564602.2018.1533892.

[21] Tausif, M., Khan, E., & Hasan, M. (2018, November). BFrWF: Block-based FrWF for coding of high-resolution images with memory-complexity constrained-devices. In 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON) (pp. 1-5). IEEE. doi : 10.1109/UPCON.2018.8597104.

[22] Zikiou, N., Lahdir, M., & Helbert, D. (2020). Support vector regression-based 3D-wavelet texture learning for hyperspectral image compression. The Visual Computer, 36(7), 1473-1490. doi : 10.1007/s11042-023-14738-x.

[23] Setiadi, D. R. I. M. (2021). PSNR vs SSIM: imperceptibility quality assessment for image steganography. Multimedia Tools and Applications, 80(6), 8423-8444. doi : 10.1007/s11042-020-10035-z.

[24] Nagendran, R., Ramadass, S., Thilagvathi, K., & Ravuri, A. (2025). Neural Reinforcement-Oriented Hyperspectral Image Compression: Adaptive Approaches for Enhanced Quality. Chemometrics and Intelligent Laboratory Systems, 105495. doi : 10.1016/j.chemolab.2025.105495.