2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)
GLA University, Mathura, Oct 26-28, 2017

SFrWF: Segmented Fractional Wavelet Filter Based Dwt For Low Memory Image Coders

Mohd Tausif [1], Ekram Khan[1], Mohd Hasan[1]and Martin Reisslein[2]
[1]Department of Electronics Engineering, A. M. U., Aligarh, India
mohdtausif32@gmail.com, ekhan67@gmail.com, mohd.hasan@amu.ac.in
[2]Schools of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, USA
reisslein@asu.edu

***Abstract*— The Discrete Wavelet Transform (DWT) is extensively used for image coding due to its excellent energy compaction property and its ability to simultaneously analyze images in space-frequency domains. However, conventional methods of computing the DWT coefficients of an image require large amounts of memory, thus making them unsuitable for memory-constraint low-cost portable devices. In this paper we propose a novel low memory approach named Segmented Fractional Wavelet Filter SFrWF to compute the DWT of high resolution images on low-memory devices. Evaluation results show that the SFrWF requires less than 10 kB of RAM for a gray-scale image of size 2048×2048 thus making the SFrWF suitable for low-cost visual sensor nodes.**

Keywords*— *Discrete Wavelet Transform, Fractional Wavelet Filter, low memory, visual sensor nodes.

I. INTRODUCTION

Most modern electronic gadgets, such as mobile-phones and personal digital assistants (PDAs), have built-in infrastructure to capture and share images wirelessly. A visual sensor is a wireless sensor node equipped with a camera. A Visual Sensor Network (VSN) is a network of visual sensors. VSNs are widely used for wildlife monitoring, vehicle traffic monitoring, and tracking objects of interest [1]-[5]. In most such applications, the deployment of large numbers of sensor nodes necessitates the use of low-cost visual sensors. The available random access memory (RAM) of most of the low-cost sensor nodes is only of the order of 10 kB [1]. Also, hand-held devices are mass market consumer products, and so their cost should be as low as possible. In order to maintain their low cost, the built-in RAM of these devices is generally kept small [6].

Furthermore, since the bandwidth of wireless links is limited, the captured images must be compressed, before their transmission [7]. An image coder mainly consists of a transform stage (for energy compaction) and coding stage (to quantize and encode the transformed coefficients). The transform and coding algorithms chosen in an image coder must satisfy the memory constraint of low-cost visual sensors. In this work we concentrate on the low-memory Discrete Wavelet Transform (DWT).

In order to facilitate transmission of images over band-limited wireless networks (or Internet), the transformed coefficients need to be coded using efficient image coding algorithm. The total memory requirement of an image coder will then be the larger of the required transform and coding algorithm memories. Various wavelet based image coding algorithms, such as the Low Memory Block Tree Coder (LMBTC) [7], Backward Coding of Wavelet Trees (BCWT) [8], the Wavelet Image two line coder (Wi2l) [9], and the Zero Memory Set Partitioned Embedded Block Coder (ZM-SPECK) [10] satisfy the memory-constraint of low cost visual sensor nodes. ZM-SPECK does not require any memory for its implementation (except for a few buffers), yet it needs to be combined with DWT, to design an image coder. In such case, the transform memory determines the overall codec memory and therefore the transform memory needs to be reduced to design a low-memory image coder.

The traditional approaches to compute the DWT require the whole image to be kept in system memory, which limits the implementation on memory-constraint devices. Also the memory requirement of the DWT increases linearly with the image size. Thus, it would be difficult to implement the DWT for high-resolution (HR) images [1].

To overcome this limitation, many low-memory implementations of the DWT have been proposed [6], [11]-[18], which can be categorized into three groups: line-based approaches [6], [11], [12], block-based approaches [13], [14], and strip-based approaches [15]-[18]. In line-based DWT, the image data is read line by line in a buffer and only the lines that are necessary to compute DWT coefficients are kept in the memory. In block-based approaches, the image is first partitioned into blocks and the wavelet transform is then applied on these fixed size blocks, rather than on the whole image. The strip-based DWT, also known as Z-scan DWT is analogous to line based DWT applied on wide blocks. The memory requirement of these methods for a typical 512×512 size gray-scale image is approximately 26 kB [1], which is still more than the available on-board memory of many low-cost sensor nodes. Recently, the Fractional Wavelet Filter (FrWF) has been proposed to compute the DWT with much smaller memory than the previous methods and is implementable on low cost sensor nodes [1], [19].

Although the FrWF satisfies the memory constraint of low-cost portable devices and sensor nodes for low resolution images, for HR images (images of dimensions greater than 1024×1024), the FrWF memory requirements are more than that available on typical sensor nodes. Since the demands for

978-1-5386-3004-4/17/$31.00 ©2017 IEEE

HR images are increasing day-by-day in applications such as medical imaging and space-borne imaging, there is a need to develop low-cost sensor nodes for such applications.

In this paper we propose the Segmented Fractional Wavelet Filter (SFrWF) to compute the DWT coefficients of an image. Instead of reading the complete image line into a buffer (as in case of the FrWF), the image lines are partitioned into segments and then each segment is read separately into the buffer. In fact, the SFrWF is a specifically designed combination of a general overlap-add method with the FrWF. Experimental results show that the SFrWF requires much smaller memory and can be used to compute the DWT of HR images using low-cost microcontrollers. To the best of our knowledge, this is the first attempt on reducing the transform memory beyond that of FrWF.

The rest of the paper is organized as follows. Brief overviews of the FrWF and the overlap-add method are presented in Section II. The proposed SFrWF algorithm is described in Section III. Experimental results and related discussions are presented in Section IV. Finally, the paper is concluded in Section V.

II. BACKGROUND

A. Fractional Wavelet Filter (FrWF)

The FrWF is a low-memory approach to compute the DWT of an image. In this approach, the original image and the sub-bands are stored in an external memory card. A vertical filter area (VFA) is defined which selects the number of rows equal to the length of the low pass filter (LPF) from the SD-card. The FrWF uses only three buffers, each capable of storing N coefficients, for an image of size $N \times N$. Only one row from the VFA is read into buffer $\mathbf{s}$ at a time. All operations required to calculate the DWT are performed on this line and the intermediate results are stored into two different buffers, namely *LL_HL* and *LH_HH*. For a gray-scale image, the memory requirement of FrWF using floating point arithmetic for '*lev*' levels of wavelet transform is given by:

$$`Bytes_{float} = \begin{cases} 9N, & lev = 1 \\ \frac{12N}{2^{(lev-1)}}, & lev > 1. \end{cases} \quad (1)$$

For further details on the FrWF, readers are referred to [1].

B. Overlap-add Method

The overlap-add method is a technique to compute the convolution of signals by partitioning the signals into different parts [20]. Although the overlap-add method has been developed for speech processing, it can be readily extended to image processing [21]. To understand the overlap-add method, let us partition the input signal $\boldsymbol{x}$ of length N into four equal parts $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$, and $\boldsymbol{x_4}$, each of size $L=N/4$. The signals $\boldsymbol{x}, \boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}$ and $\boldsymbol{x_4}$ are convolved separately by a filter $\boldsymbol{h}$ of length M resulting in outputs $\boldsymbol{z}$ (of length $N+M-1$), $\boldsymbol{z_1}$, $\boldsymbol{z_2}$, $\boldsymbol{z_3}$, and $\boldsymbol{z_4}$ (each of length $L+M-1$), respectively. The overlap-add method, as depicted in Fig. 1, ensures the linearity property of convolution, i.e. $\boldsymbol{z} = \boldsymbol{x} \otimes \boldsymbol{h} = (\boldsymbol{x_1}+\boldsymbol{x_2}+\boldsymbol{x_3}+\boldsymbol{x_4}) \otimes \boldsymbol{h} = \boldsymbol{z_1}+\boldsymbol{z_2}+\boldsymbol{z_3}+\boldsymbol{z_4}$. From Fig. 1 it is clear that when the highlighted terms of $\boldsymbol{z_1}$, $\boldsymbol{z_2}$, $\boldsymbol{z_3}$ and $\boldsymbol{z_4}$ are overlapped and added together, it gives the same result as $\boldsymbol{z}$. The highlighted terms are basically the overlapped and added terms. The last M-1 terms of $\boldsymbol{z_1}$, $\boldsymbol{z_2}$, and $\boldsymbol{z_3}$ are added consecutively to the first M-1 terms of $\boldsymbol{z_2}$, $\boldsymbol{z_3}$, and $\boldsymbol{z_4}$, respectively. The resulting $\boldsymbol{z_1}, \boldsymbol{z_2}, \boldsymbol{z_3}$, and $\boldsymbol{z_4}$ when concatenated together would give $\boldsymbol{z}$. This property of convolution is utilized in the proposed SFrWF.

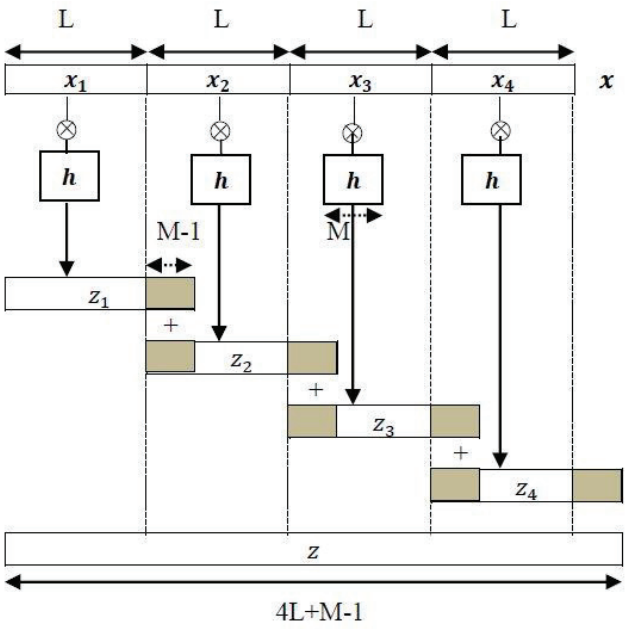

Fig. 1. Schematic diagram showing the overlap-add method: Partitions of length L of the original signal $\boldsymbol{x}$ are convolved (denoted by symbol $\otimes$) with a filter $\boldsymbol{h}$ of length M.

III. SEGMENTED FRACTIONAL WAVELET FILTER

The SFrWF is a novel approach to further reduce the memory requirement of the FrWF. The SFrWF partitions each image line into multiple (k, $k \geq 2$) segments and applies the FrWF to each segment. The segments are combined with the overlap-add method. The proposed SFrWF technique uses nine line buffers: an input buffer $\boldsymbol{s}$ of dimension $1 \times \frac{N}{k}$ (for an image of size $N \times N$, whereby each line is partitioned into k segments); four buffers $\boldsymbol{d_{LL}}$, $\boldsymbol{d_{LH}}$, $\boldsymbol{d_{HL}}$, and $\boldsymbol{d_{HH}}$ whereby each stores $\left(\frac{M-1}{2}\right)$ filtered coefficients, where M = length of filter; and four intermediate (temporary) buffers LL_t, LH_t, HL_t, and HH_t, each of dimension $\left(1 \times \frac{N}{2k} + \frac{M-1}{4}\right)$. In addition, the SFrWF requires one buffer to hold the convolution result (L or H). These nine line buffers and the single element buffer consume system memory. In order to explain the logic of the SFrWF, let us consider $k = 2$ segments, (though it can be generalized to

any value of *k)*. Further, let us focus on computing the DWT coefficients for one decomposition level. The original image and the final sub-bands LL, LH, HL, and HH, (each of dimension $\frac{N}{2}\times\frac{N}{2}$) are assumed to be stored on an external secure digital (SD) card. In order to apply the FrWF to each segment, a vertical area (VA) containing the number of rows equal to the number of coefficients in the low pass filter (LPF) is selected. A VA includes the number of columns equal to the segment size, making the dimensions of the VA equal to $M\times\frac{N}{k}$.

While calculating the DWT, at any given time only one row from the VA is read into the input buffer $\boldsymbol{s}$ and filtered, (according to the steps given in Table I) to obtain the coefficients. Prior to filtering, the input data (of the first segment) is symmetrically extended on the left-hand side. The extended signal is then jointly filtered and down-sampled. For instance, for a low-pass filter $\boldsymbol{l}$ of length M whose impulse response is centered at location zero, we sum over the index m = -(M-1)/2 to m = (M-1)/2 to compute the convolution:

$$L = \sum_m \boldsymbol{s}_{(2n+m)} \cdot \boldsymbol{l}_m \triangleq \text{conv}(\boldsymbol{s}, \boldsymbol{l}, 2n). \qquad (2)$$

This abbreviated representation of the convolution operation conv($\boldsymbol{s}$, $\boldsymbol{l}$, $2\boldsymbol{n}$) is employed in Line 6 of Table I and analogously for the high-pass filter in Line 9 of Table I. The low-pass filtered coefficients are then multiplied by a particular tap-gain of low-pass and high-pass filters and are stored in the LL_t and LH_t buffers, respectively. A similar process is applied to the high-pass filtered coefficients (output of conv($\boldsymbol{s}$, $\boldsymbol{h}$, $2\boldsymbol{n}$ +1), where $\boldsymbol{h}$ denotes high pass filter) which are stored in the HL_t and HH_t buffers, respectively. The VA is then slided horizontally to cover the columns of the next segment of the same rows, and the new VA is then processed similarly. Note that the pre-filtering symmetric extension will be applied to data belonging to the first (left-side extension) and the last (right-side extension) segments only. After covering one set of rows in all segments, the VA is moved down by two lines to achieve vertical downsampling by a factor of two.

The steps involved in implementing the SFrWF for k = 2 segments are summarized in Table I. From the implementation point of view, the buffer contents are transferred to the SD-card as follows. All the coefficients of the LL_t buffer (updated coefficients of first segment), except the last (highest indexed) four ($=\frac{M-1}{2}$; where M is the length of LPF, M =9 here) elements, are transferred to the SD-card, whereas remaining four coefficients of LL_t (denoted by $LL_{t(h\_4)}$) are transferred to buffer $\boldsymbol{d}_{LL}$ (Line 12, Table I), thereby emptying the LL_t buffer to be used for processing of the next updated by adding with the corresponding elements of $\boldsymbol{d}_{LL}$ (Line 18, Table I). Next, the contents of the LL_t buffer are concatenated with the previously stored data on the SD-card. The process is repeated for each segmented VA of the same rows, except the last segmented VA, for which the entire contents of LL_t are concatenated on the SD-card. After repeating the process for all VAs, the LL sub-band is obtained. The same procedure is executed for the intermediate buffers LH_t, HL_t, and HH_t to obtain the sub-bands LH, HL, and HH, respectively.

TABLE I: STEPS OF PROPOSED SFrWF, illustrated for k =2 segments

1.	For i = 0, 1, 2,…, N/2-1;
2.	Initialize temporary buffers LL_t, LH_t, HL_t, and HH_t to zero
3.	For j = -4, -3, …, 4: // for 9 tap filter
4.	Read first half of image line 2i + j from SD-card into $\boldsymbol{s}$
5.	For $\boldsymbol{n}$ = 0, 1,…, N/4-1.
6.	L= conv($\boldsymbol{s}$, $\boldsymbol{l}$, $2\boldsymbol{n}$)
7.	LL_t += $\boldsymbol{l}_j \cdot L$ // update LL_t
8.	LH_t += $\boldsymbol{h}_{(j-1)} \cdot L$ // update LH_t
9.	H= conv($\boldsymbol{s}$, $\boldsymbol{h}$, $2\boldsymbol{n}$ +1)
10.	HL_t += $\boldsymbol{l}_j \cdot H$ // update HL_t
11.	HH_t += $\boldsymbol{h}_{(j-1)} \cdot H$ // update HH_t
12.	$\boldsymbol{d}_{LL} \leftarrow LL_{t(h\_4)}$ // highest indexed four coeffic. of LL_t
13.	$\boldsymbol{d}_{LH} \leftarrow LH_{t(h\_4)}$; $\boldsymbol{d}_{HL} \leftarrow HL_{t(h\_4)}$; $\boldsymbol{d}_{HH} \leftarrow HH_{t(h\_4)}$
14.	Write remaining coeffic. of LL_t, LH_t, HL_t, and HH_t buffers as coeffic. of LL, LH HL, and HH to SD-card; clear buffers;
15.	For j = -4, -3, …, 4:
16.	Read second half of image line 2i+ j from SD-card into $\boldsymbol{s}$
17.	Repeat steps 5-11
18.	$LL_{t(l\_4)}$ += $\boldsymbol{d}_{LL}$ // lowest indexed four coeffic. of LL_t
19.	$LH_{t(l\_4)}$ += $\boldsymbol{d}_{LH}$; $HL_{t(l\_4)}$ += $\boldsymbol{d}_{HL}$; $HH_{t(l\_4)}$ += $\boldsymbol{d}_{HH}$
20.	Concatenate coefficients of LL_t, LH_t, HL_t, and HH_t buffers with previously stored coefficients of the corresponding lines of LL, LH HL, and HH, respectively, on SD-card

For each subsequent decomposition level, the steps in Table I can be repeatedly applied on the resulting LL sub-band of the previous decomposition level.

A. Memory and Complexity analysis of SFrWF

The memory requirement of the SFrWF depends on the number of elements and the size of each element stored in the nine buffers. The coefficients of input buffer $\boldsymbol{s}$ will be of one byte each (unsigned integer as they are directly read from the image) and the coefficients of the remaining eight buffers will be of four bytes each (to store real numbers obtained after convolution). Thus, the SFrWF memory requirement are:

$$Memory_{SFrWF} = \frac{9N}{k} + 12(M-1) \text{ Bytes} \qquad (3)$$

Where first term $\frac{9N}{k}$ bytes is the memory size of buffer $\boldsymbol{s}$ plus the $(1\times\frac{N}{2k})$ components of the four temporary line buffers; whereas $8(M-1)$ bytes of the second term are due to buffers $\boldsymbol{d}_{LL}$, $\boldsymbol{d}_{LH}$, $\boldsymbol{d}_{HL}$, and $\boldsymbol{d}_{HH}$ and the remaining $4(M-1)$ bytes of the second term are for the four temporary line buffers. Note that the same buffers may be used for computing higher DWT levels.

Following the steps in [19], the computational complexity of SFrWF in terms of number of additions and multiplications is:

TABLE II: MEMORY REQUIREMENT AND COMPUTATIONAL COMPLEXITY OF DWT, FrWF, AND SFrWF (for different numbers of segments k)

Transform (k)	Memory (kB)			Complexity (seconds)		
	512×512 (Image Size)	1024×1024 (Image Size)	2048×2048 (Image Size)	512×512 (Image Size)	1024×1024 (Image Size)	2048×2048 (Image Size)
DWT	2097.152	8388.608	37748.736	0.441	1.504	4.679
FrWF	4.608	9.216	18.432	0.800	1.820	5.396
SFrWF (2)	2.400	4.704	9.312	1.497	3.251	7.290
SFrWF (4)	1.248	2.400	4.704	3.119	6.371	13.583
SFrWF (8)	0.672	1.248	2.400	5.785	11.597	23.781

$$Compl._{SFrWF} = \left(\frac{81}{2}N^2 + \frac{NM(M-1)(k-1)}{4}\right)\boldsymbol{a} + \left(\frac{117}{4}N^2M\right)\boldsymbol{m} \qquad (4)$$

Where $\boldsymbol{a}$ and $\boldsymbol{m}$ represent the time required for computing one addition and one multiplication, respectively.

IV. RESULTS AND DISCUSSION

In this section the memory requirement and computational complexity (execution time in seconds) of the proposed SFrWF are compared with the FrWF and DWT in Table II. We report averages over twenty-five popular gray-scale test images of varying resolution obtained from the Waterloo Repertoire (http://links.uwaterloo.ca) and the standard image database (http://sipi.usc.edu/database). All the algorithms are implemented using MATLAB 7.0 and are executed on a Windows 8.1 Netbook with Intel atom CPU Z 3735F @ 1.33 GHz with 2 GB RAM and 32 GB MMC card. We observe from Table II that the SFrWF consumes much less memory than the FrWF. The SFrWF memory consumption can be further reduced by increasing the number of segments. Table II indicates that for an image of size 2048×2048, the FrWF consumes about 18.4 kB of memory, whereas the SFrWF requires only 4.704 kB and 2.400 kB of memory for $k = 4$ and $k = 8$, respectively.

Moreover, Table II indicates that the FrWF and SFrWF require more time than the conventional DWT for computing the transform coefficients. This is because the FrWF requires 2.89 times more add operations and 3.25 times more multiplication operations than the DWT [19]. The SFrWF is based on the FrWF concepts; however, due to the overlap-add process, the SFrWF uses more addition operations than the FrWF, which leads to a complexity increase. The SFrWF complexity increases linearly with the number of segments 'k' due to the additional overlap and add operations. Thus, the SFrWF presents a trade-off between memory and time complexity, as shown in Fig. 2.

Furthermore, the SFrWF can be readily extended to images of higher resolution. For example, for an image of size 8192×8192, the SFrWF would require only 4.704 kB and 2.400 kB of RAM for $k = 16$ and $k = 32$, respectively (from (3)), whereas the DWT and FrWF would require

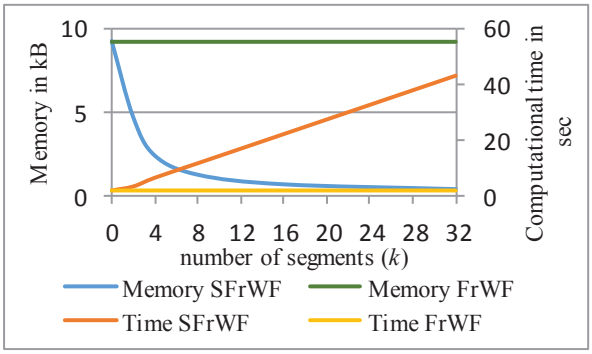

Fig. 2. Memory-computational complexity (time) trade-off of FrWF and SFrWF for image size 1024×1024: SFrWF enables flexible memory requirement vs computation time trade-off.

536.87 MB and 73.728 kB of RAM, respectively. Note that the SFrWF (with any value of k) when combined with any coding algorithm, such as LMBTC [7] or ZM-SPECK [10], gives exactly the same reconstructed image quality as using the same coding algorithm with the FrWF or conventional DWT.

V. CONCLUSION

In this paper the Segmented Fractional Wavelet Filter (SFrWF) is proposed to drastically reduce the memory requirements for computing the DWT. The SFrWF applies the FrWF on segmented image lines and then combines filtered segments using an overlap-add process. Experimental results demonstrate that the SFrWF can be used for computing the DWT of high-resolution images even on low-cost visual sensors and memory-constrained hand-held portable devices. In future research we aim to reduce the computational complexity of the SFrWF without increasing its memory requirements.

ACKNOWLEDGMENT

This work was supported in part by a grant from the Visvesvaraya PhD Scheme, Department of Electronics and Information Technology (DeitY), Government of India.

REFERENCES

[1] S. Rein and M. Reisslein, "Low-memory wavelet transforms for wireless sensor networks: A tutorial," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 2, pp. 291–307, Second Quarter 2011.

[2] X. Liu, D. Zhai, J. Zhou, X. Zhang, D. Zhao, and W. Gao, "Compressive sampling-based image coding for resource-deficient visual communication," *IEEE Trans. on Image Processing*, vol. 25, no. 6, pp. 2844-2855, June 2016.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "Wireless multimedia sensor networks: Applications and testbeds," *Proc. IEEE*, vol. 96, no. 10, pp. 1588–1605, October 2008.

[4] T. Melodia and I. F. Akyildiz, "Research challenges for wireless multimedia sensor networks," in *Distributed Video Sensor Networks*. London, U.K.: Springer, 2011, pp. 233–246.

[5] O. M. L. Granado, M. O. Martínez-Rach, P. P. Peral, J. O. Gil, and M. P. Malumbres, "Rate control algorithms for non-embedded wavelet-based image coding," *J. Signal Process. Syst*., vol. 68, no. 2, pp. 203–216, August 2012.

[6] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. Image Proc.*, vol. 9, no. 3, pp. 378–389, March 2000.

[7] M. Tausif, N. Kidwai, E. Khan and M. Reisslein, "FrWF-based LMBTC: Memory-efficient image coding for visual sensors," *IEEE Sensors Journal*, vol. 15, no. 11, pp. 6218-6228, November 2015.

[8] J. Guo, S. Mitra, B. Nutter, and T. Karp, "A fast and low complexity image codec based on backward coding of wavelet trees," *in Proc. Data Compress. Conf. (DCC),* pp. 292–301, March 2006.

[9] S. Rein and M. Reisslein, "Scalable line-based wavelet image coding in Wireless Sensor Networks," *Journal of Visual Communication and Image Representation*, vol. 40, pp. 418-431, July 2016.

[10] N. R. Kidwai, E. Khan and M. Reisslein, "ZM-SPECK: A fast and memoryless image coder for multimedia sensor networks," *IEEE Sensors Journal*, vol.16, no. 8, pp. 2575-2587, April 2016.

[11] J. Oliver and M. Malumbres, "On the design of fast wavelet transform algorithms with low memory requirements," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 2, pp. 237–248, February 2008.

[12] L. Ye, J. Guo, B. Nutter, and S. Mitra., "Low-memory-usage image coding with line-based wavelet transform.", *SPIE Journal of Optical Engineering*, vol. 50, no. 2, pp. 027005-1-027005-11, February 2011.

[13] C.-H. Yang, J.-C. Wang, J.-F. Wang, and C.-W. Chang, "A block based architecture for lifting scheme discrete wavelet transform," *IEICE Trans. Fundamentals,* vol. E90-A, no. 5, pp. 1062–1071, May 2007.

[14] Y. Bao and C.-C. J. Kuo, "Design of wavelet-based image coder in memory-constrained environment," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 5, pp. 642–650, May 2001.

[15] L. W. Chew, W. C. Chia, L. M. Ang, and K. P. Seng, "Very low memory wavelet compression architecture using strip-based processing for implementation in wireless sensor networks," *EURASIP Journal on Embedded Systems*, vol. 2009, no. 479281, pp. 1-16, December 2009.

[16] L. W. Chew, W. C. Chia, L. Ang, and K. P. Seng. "Low-memory video compression architecture using strip-based processing for implementation in wireless multimedia sensor networks," *Int. Journal of Sensor Networks*, vol. 11, no. 1, pp. 33-47, January 2012.

[17] W. C. Chia, L. W. Chew, L. Ang, and K. P. Seng, "Low memory image stitching and compression for WMSN using strip-based processing," *Int. Journal of Sensor Networks*, vol. 11, no. 1, pp. 22-32, January 2012.

[18] L. Ye and Z. Hou, "Memory efficient multilevel discrete wavelet transform schemes for JPEG 2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 11, pp. 1773-1785, November 2015.

[19] S. Rein, and M. Reisslein, "Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks," *Ad Hoc Networks*, vol. 9, no.4, pp. 482-496, June 2011.

[20] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing Principles, Algorithms and Applications*, 3rd ed., Prentice-Hall, 1996.

[21] J. S. Lim, *Two-dimensional Signal and Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1990.