Signal Processing: *Image Communication* 37 (2015) 58–74

Contents lists available at ScienceDirect

Signal Processing: *Image Communication*

journal homepage: www.elsevier.com/locate/image

Evaluation of the wavelet image two-line coder: A low complexity scheme for image compression ☆

Stephan A. Rein [a,*], Frank H.P. Fitzek [a], Clemens Gühmann [b], Thomas Sikora [c]

[a] *Aalborg University, Department of Electronic Systems, Mobile Device Group, Frederik Bajers Vej 7, 9220 Aalborg, Denmark*
[b] *Technische Universität Berlin, Department of Energy and Automation Technology, Chair of Electronic Measurement and Diagnostic Technology, Einsteinufer 17, 10587 Berlin, Germany*
[c] *Technische Universität Berlin, Department of Telecommunication Systems, Communication Systems Group, Einsteinufer 17, 10587 Berlin, Germany*

ARTICLE INFO

Article history:
Received 26 June 2012
Received in revised form
24 July 2015
Accepted 24 July 2015
Available online 1 August 2015

Keywords:
Wavelet image two-line coder (Wi2l)
Set-partitioning in hierarchical trees (SPIHT)
Backward coding of wavelet trees (BCWT)
Fractional wavelet filter
Fixed-point wavelet transform
Camera sensor node

ABSTRACT

This paper introduces the wavelet image two-line (Wi2l) coding algorithm for low complexity compression of images. The algorithm recursively encodes an image backwards reading only two lines of a wavelet subband, which are read in blocks of 512 bytes from flash memory. It thus only requires very little memory, i.e., a memory array for two wavelet subband lines, an array to store intermediate tree level data, and an array for writing binary data. A picture of 256×256 pixels would require 1152 bytes of memory. Computation time for the coding is derived analytically and measured on a real system. The times on a low-cost microcontroller for 256×256 grayscale pictures are measured as 0.25–0.6 s for encoding and 0.22–0.77 s for decoding. The algorithm can thus realize a low complexity system for compression of images when combined with a customized scheme for the wavelet transform; low complexity here refers to low memory, minimum write access to flash memory, usage of integer operations only, and low conceptual complexity (ease of implementation). As demonstrated in this paper, a compression performance similar to JPEG 2000 and the more recent Google WebP picture compression is achieved. The compression system uses flash memory (SD or MMC card) and a small camera sensor thus building an image communication system. It is also suitable for mobile devices or satellite communication. The underlying C source code is made publicly available.

© 2015 Published by Elsevier B.V.

1. Introduction

The storage, communication, and sharing of pictures is usually alleviated or even made possible by picture compression techniques. There exist different compression techniques, for example, the cosine transform used by JPEG or wavelet-based techniques used by JPEG 2000. The wavelet-based techniques generally give better compression performance, however, the available implementations are rather designed for PC-usage and not for low complexity devices such as wireless sensors. It is not convenient for users and developers to care about different image compression standards. The feature of JPEG is that it more or less runs on every device, as its complexity is rather low. A new compression technique that gives better performance is more likely an alternative (to JPEG) if it is applicable to a wide range of devices.

☆ A related conference paper appeared in the Proceedings of the IEEE Data Compression Conference (DCC), Snowbird, HT, March 2009, pp. 123–132.

* Corresponding author. Tel.: +45 9940 8723; fax: +45 9815 1583.
E-mail addresses: rein@gmx.net (S.A. Rein), ff@es.aau.dk (F.H.P. Fitzek), clemens.guehmann@tu-berlin.de (C. Gühmann), sikora@nue.tu-berlin.de (T. Sikora).

http://dx.doi.org/10.1016/j.image.2015.07.010
0923-5965/© 2015 Published by Elsevier B.V.

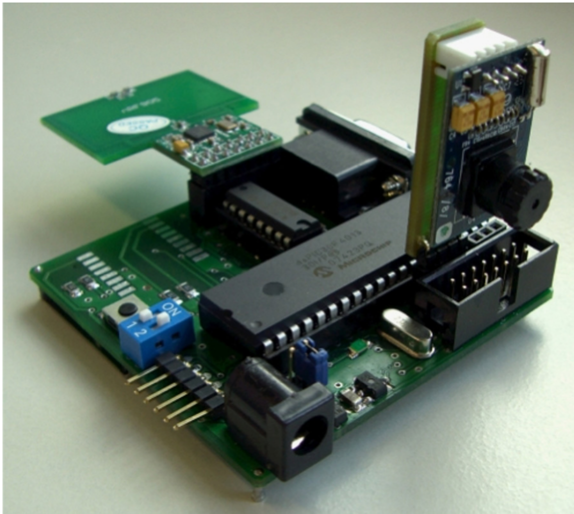

Fig. 1. Camera sensor node with the Microchip *dsPIC30F4013* controller that serves to evaluate the proposed coding algorithm.

Despite the compatibility issue, low complexity techniques save computational and memory requirements, which are usually limited or shared between different applications. A wise access of memory and minimum number of operations can save energy and improve user-responsiveness on mobile phones. The effective compression will also support the smart space, as the underlying tiny sensor nodes usually employ microcontrollers with little random access memory (RAM) in the range of a few kilobytes. A low complexity compression algorithm will allow to upgrade such nodes to a camera sensor node by connecting a small camera sensor to them. Moreover, phones and sensors both employ flash memory (multimedia or secure digital cards) for intermediate data storage, which use block-wise access to ideally be taken into account by the compression system. In this paper, we propose a wavelet-based image compression coding algorithm that gives state-of-the-art image compression, while the data access is line-wise and limited in space and time. The given memory requirements are in line with the limitations of the typical low-cost microcontrollers. A picture of 256×256 8 bit pixels can for instance be compressed using less than 1500 bytes of RAM.

The proposed coding algorithm requires a wavelet transform of the picture of interest. A picture wavelet transform results in a picture of the same dimension and allows a suitable coding technique to summarize the observed patterns, e.g., areas of zero-coefficients. In this paper we use a line-based wavelet transform and store the transformed image on flash memory. As the forward transform is regardless of the required compression rate, the intermediate storage of the transformed picture has the advantage that the transform can be reused for different quality versions of the image and is thus computed only once. On the receiver entity the picture is decoded similarly using the line-based algorithm and then inversely transformed to obtain the reconstructed image. Here the transform has to be repeated if the receiver requests for a new compression rate. For a sensor node, the line-wise coding gives the advantage that the binary data can be accessed directly from the flash memory. Thus, for encoding a picture, a very small buffer to store two subband lines is needed, and then the corresponding compressed binary stream can be written to the flash memory or directly be sent out.

In standard tree-based wavelet coding, such as performed with the Set-Partitioning in Hierarchical Trees (SPIHT) algorithm [1], the coefficients are scanned in a tree-based manner for each bit plane. This results in several scans of a transformed picture, and the access pattern is not line-wise. Furthermore, different lists to store the types of the coefficients due to the coding of previous bit-planes need to be maintained, which exceed the memory resources of a small sensor. The new algorithm encodes the image recursively and backwards, and thereby only needs a very small buffer for intermediate information between consecutive lines. Each image line is only accessed once. The backwards encoding starts to access the coefficients from the lowest wavelet level. To decode the image properly, the binary stream needs to be read at the receiver in the reversed order. The reversion can be performed on the sender or the receiver entity. We note that the reversal of the bit stream is not an explicit operation, but a rule that can be achieved by setting a reading pointer to the correct position and reading the data in the required order. A sensor node, for instance, if it saves the binary stream on the flash memory, can read the stored data reversely for the sending procedure. On the receiver the image is decoded in the usual order from the top wavelet level on. A drawback of the required data reversion is that it is not possible to perform an online data compression; the complete picture data needs to be encoded before the decoding can start.

The paper is structured as follows. In the next subsection related work is reviewed. In Section 2, we first shortly describe the so-called *fractional* wavelet transform, which is a computational scheme employed in this paper to transform the image with very little memory and integer calculations only. Then we give our notation for the tree-based coding and explain the principle of backwards wavelet coding. Specifically, we have implemented a recently introduced backwards algorithm as a reference for the introduced line-based algorithm. In Section 3, we describe the new algorithm, which is compared in Section 4 to the BCWT reference implementation, to SPIHT, to JPEG and JPEG 2000, and to the recent Google WebP format. We also implement the new algorithm on a 16 bit microcontroller and conduct measurements on the camera sensor node illustrated in Fig. 1 to estimate the required coding times. In the last section the paper is concluded.

1.1. Related work

In this section we motivate the selection of a wavelet-based coding technique as a starting point and review related work for low complexity coding of wavelet-transformed pictures. We aim for a lossy compression system for natural images that is of low complexity and practical to implement. While there exist many proposals for new algorithms for image compression, there only exist

a few number of well-known standards. From the JPEG committee, these are the JPEG image coding standard (ITU-T T.81 | ISO/IEC 10918-1), the more recent JPEG XR standard (ITU-T T.832 | ISO/IEC29199-2), and the JPEG 2000 standard (ITU-T T.800 | ISO/IEC 15444-1). Recently, there has also been proposed the WebP format from Google based on the VP8 image and video compression [2], which uses a block-based predictive coding. While the underlying methods of these methods are not necessarily designed for very low complexity platforms such as microcontrollers in wireless sensors, it yet makes sense to include them in our considerations as a reference for compression performance. The JPEG technique is more than 20 years old and does not give state-of-the-art compression any more [3]. The JPEG XR technique consistently outperforms JPEG and is very practical in terms of implementation; its performance is however slightly lower than with JPEG 2000 [4]. The WebP format is not yet extensively evaluated for a wide range of pictures, however, it is concluded from the measurements in [5] that its performance is close to JPEG 2000. A more important finding (that will be confirmed by our results in Section 4) is that for very high compression rates (such as 0.125 bpp), the JPEG and VP8 techniques do not provide support, and JPEG XR is outperformed by JPEG 2000. Another compression technique for still images that has only attained little attention is the AVC (H.264, MPEG-4 Part 10) intra-frame coding [6] based on a block-based integer DCT transform. It is evaluated in [7,8] to give similar or better compression performance for medium and high bit rates, while at very low bit rates JPEG 2000 can achieve better PSNR values. The employed block-based transform can result in visible artifacts which do not appear with the wavelet-based method, where the picture is transformed as a whole. The artifacts due to the block-based processing are addressed in [9], however, application on low-complexity target platforms is not aimed.

A low complexity sensor node can rather communicate with low data rates, thus high compression is a basic requirement. Another reason for high compression is the lower computational cost, which can save energy and time. As the literature states that the wavelet-based methods are very suitably for high compression, we select the wavelet transform for data pre-processing. Another reason for selecting the wavelets is that there already exist low complexity algorithms for the transform [10–12] and previous works on coding algorithms to be reviewed in the following, which give promising compression already without using entropy coding.

An established wavelet coding scheme for natural images that is included in most textbooks on data compression (e.g. in [13]) is the Set Partitioning in Hierarchical Trees (SPIHT) algorithm [1], which is based on the Embedded Zerotree Wavelet (EZW) [14]. Regarding its low complexity and our specific use-case that does not require the complete range of image types (including for instance artificial or medical images), the algorithm gives promising compression performance which is competitive to JPEG 2000, see the results in [3]. Entropy coding, as for instance arithmetic coding [15], is not needed. We therefore limit our considerations on previous work on wavelet coding to approaches that aim to reduce the complexity and memory requirements of SPIHT. These can be a few Megabytes only for the list entries, as the list approximately requires twice the memory as for the total number of coefficients [16]. We differentiate between three classes of literature aiming for (a) reduction of intermediate data, (b) reduction of picture data, and (c) improved speed and energy usage.

SPIHT maintains three lists: the list of insignificant pixels (LIP), the list of significant pixels (LSP), and the list of insignificant sets (LIS). Works of class (a) are given in [16–22], and the memory savings are due to reduction of list entries or cancellation of the lists. As these works assume the complete original picture to be kept in the memory, these works do not comply to our use-case of little RAM in the sensor node. Works of class (b) require a subset of the original image, and are reviewed in the following. In [23,24] coding systems for FPGA platforms are given, which require 15 kB of RAM for a picture dimension of 128×128 and 0.5 MB of RAM for a picture dimension of 512×512. The coder in [25] uses – at minimum – memory for eight picture lines with the use of a novel tree structure, and requires $256 \times 8 \times 2$ bytes RAM for a picture dimension of 256×256 using 16 bits per coefficient. The system is verified via a Matlab simulation. The more recent algorithmic in [26] performs single lines of the image that exceed the symmetrical part. These lines are transformed and coded differently thus giving a better compression performance, while no line-based solution for the symmetrical part is given.

Examples for works that relate to class (c) are the high-speed version of SPIHT in [27], which gives slightly lower performance, and the work in [28], which presents a wavelet-based technique called *SHPS* (Skipped High- Pass Sub-band) to distribute the computations over cooperating sensor nodes. SHPS can be parameterized to obtain different trade-offs between the image quality and the energy expended for compression and transmission of the image. A principle that is more in line with our low memory requirements in the range of a few kilobytes is the *backward coding wavelet tree* (BCWT) [29–31]. It belongs to the classes (b) and (c), as it reduces the need for memory by traversing the image using a small set of lines and thereby saves operational cost as each line is only accessed once. The savings result from the backwards scanning of the image, whereby each coefficient is entirely coded using a map of maximum quantization levels. While the compression performance of BCWT is the same as with SPITH, BCWT does not provide the progressive feature.

The memory reductions of BCWT over SPIHT are already promising. For a picture dimension of 256 it requires 42×256 coefficient entries (each one of 16 bits) and 3×256 entries for the map of quantization levels (each one of 8 bits), which results in roughly 22 kB. In this work the principle of backwards coding is resumed with the aim to further reduce the requirements by more than one order of magnitude. We review and evaluate the so-called Wavelet image two-line coder (Wi2l) [32], which builds in combination with a selected low complexity transform a system for image compression. The conceptual difference between the BCWT and the new Wi2l algorithm is that BCWT encodes the image in squared units of 16 coefficients, while Wi2l encodes a set of two wavelet

subband lines. BCWT encodes all units of a wavelet subband, and then the algorithm switches to the next higher wavelet level. The order in which the units are selected is not relevant (as long as it is in line with the order performed by the decoder). As in the next higher level, maximum quantization level information from the units of the previous level is needed, a list is employed to store all relevant information of a complete subband. In contrast to BCWT, the Wi2l algorithm reads two wavelet subband lines and only stores intermediate information regarding these two lines, thereby achieving the memory savings. The link between different levels of the wavelet transformed image is achieved via a proper selection of subband lines – in fact, Wi2l frequently switches between lines of different levels, while BCWT encodes a subband as a whole. To our best knowledge, the proposed scheme allows for the first wavelet-based picture compression system that is demonstrated to operate on a low-cost microcontroller.

2. Principles for wavelet coding and notation

In this section we describe (1) the low complexity scheme for the wavelet transform that has to be applied before the coding, (2) our notation for encoding single wavelet coefficients and significance levels, and (3) the principle of tree-based backwards coding, which we implement as a reference to verify the new coding algorithm to be introduced in Section 3. Note that throughout this paper, we assume squared input images in order to improve the readability, where N serves to denote the picture dimension (number of lines or columns).

2.1. Low complexity wavelet transform

The wavelet-based compression is two-fold in that first, a wavelet transform has to be performed, which rearranges the image information such that following summarizing techniques (the *coding* of wavelet coefficients) can utilize the correlations. In this work, the so-called *fractional wavelet filter* and the corresponding implementation provided in [33,12] is used to calculate the transform. The fractional filter is based on the Daubechies 9/7 transform, which gives state-of-the-art image compression and is similarly employed in the JPEG 2000 standard. It is a scheme to compute the wavelet transform by reading single lines of the input picture and writing the result as a pair of subband lines. With the help of flash memory

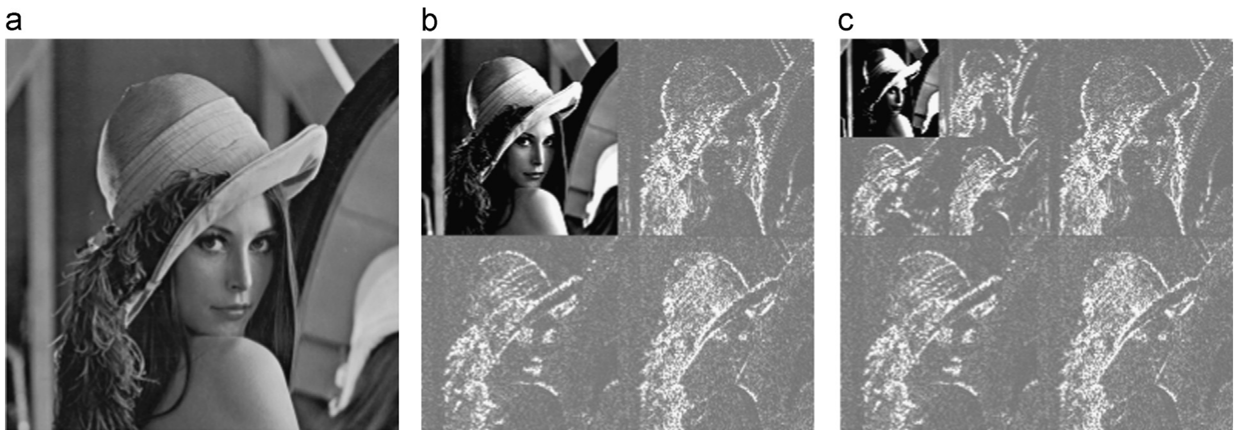

Fig. 2. Picture wavelet transform: Original image (a), one-level transform (b), and two-level transform (c).

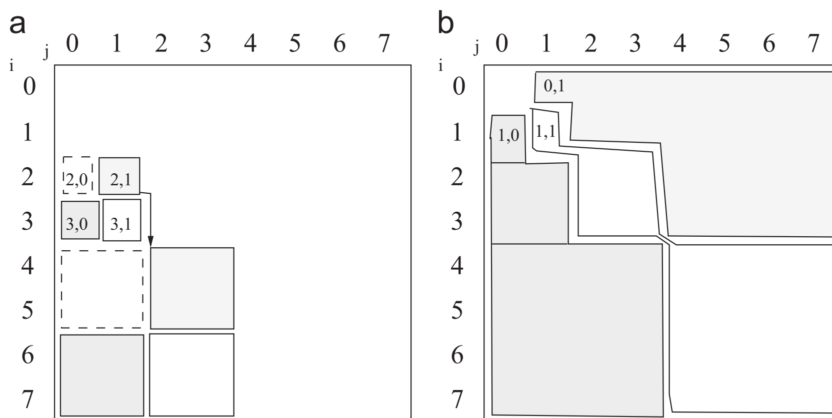

Fig. 3. Two-level quadtree in (a) and three-level tree (b), including the *grandchildren* nodes.

(used in this work in conjunction with the file system from [34]), large images can be transformed using extremely little RAM memory. As the filter is intended for low-complexity target platforms (16 bit microcontrollers), fixed-point arithmetic is used instead of floating-point filter computations. The loss of precision due to fixed-point calculations does not affect the compression performance in case of high compression rates, as in that case the least significant bits are cut off anyway.

Fig. 2 shows an example of a wavelet transform for different wavelet levels of the *Lena* image. The term *wavelet level* in this work refers to a wavelet decomposition level. In (a) the original image is shown, in (b) the one-level transform, and in (c) the two-level transform. We denote the four square matrices that result from a transform as *LL* (upper left), *HL* (upper right), *LH* (lower left), and as *HH* matrix. In this work the transform is computed up to wavelet level 6. The interested reader is referred to the tutorial in [35] for more details on the fixed-point fractional wavelet filter and the provided source code.

2.2. Notation for coding of wavelet coefficients

In this section our notation for encoding of wavelet coefficients is given. In wavelet based coding, the tree structure of the coefficients is utilized; thus it is necessary to traverse the tree and to address specific sets of coefficients. In the following we first give the notation for sets of descendant coefficients in the tree. Then we describe how single coefficients and quantization levels are addressed and finally, we give the notation for binary encoding and decoding of coefficients and levels.

2.2.1. Sets of wavelet coefficients

An elementary part of a typical wavelet tree is denoted as *quadtree*. A quadtree has a root node and four descendants which are referred as *child* nodes. Fig. 3(a) illustrates three examples of a quadtree, where the root node is defined by a line and a column index, which each start at index zero. (The position (0,0) relates to the node in the most upper left corner.) If the root node is defined by the indices (i,j), with i and j each corresponding to a line and column, respectively, the four child node locations are given by $(2i,2j)$, $(2i,2j+1)$, $(2i+1,2j)$, and $(2i+1,2j+1)$. The set of the four child positions is denoted as $C(i,j)$. The descendants of the child nodes are called *grandchild* nodes, and examples of trees including child and grandchild nodes are given in Fig. 3(b). The set of nodes that includes *all* descendants is denoted by $D(i,j)$, and the set of grandchild nodes is thus defined as $G(i,j)=D(i,j)-C(i,j)$. Table 1 summarizes the notation.

Table 1
Notation for the description of set of nodes.

$C(i,j)$	Set of four child nodes of the root (i,j)
$D(i,j)$	Set of all descendant nodes of root (i,j)
$G(i,j)=D(i,j)-C(i,j)$	Set of grandchild nodes of (i,j)

2.2.2. Description of wavelet coefficients and levels

Let $c(i,j)$ denote the numerical value of a wavelet coefficient at the position (i,j). In tree-based binary encoding, the numerical values of the coefficients are checked for their *quantization* level, thereby deciding if a coefficient or a set of coefficients is *significant* or *insignificant*. Only significant coefficients need to be encoded. A tree that has insignificant values only is called a *zerotree*. The quantization level $q(i,j)$ of a coefficient $c(i,j)$ is given as

$$q(i,j)=\begin{cases}\lfloor \log_2 |c_{i,j}| \rfloor, & |c_{i,j}|\geq 1\\ -1, & |c_{i,j}|<1,\end{cases} \tag{1}$$

where $\lfloor(\cdots)\rfloor$ rounds to the nearest integer lower or equal the input value. For the significance classification of a set of nodes it is necessary to find the *maximum* quantization level. The maximum quantization level for the set of four child nodes of position (i,j) is given as

$$q_C(i,j)=\max_{(k,l)\in C(i,j)}\{q(k,l)\}. \tag{2}$$

Similarly, for the set of the grandchildren of (i,j) the maximum quantization level is given with

$$q_G(i,j)=\max_{(k,l)\in G(i,j)}\{q(k,l)\}. \tag{3}$$

Of particular relevance for the backwards coding algorithms are the so-called *maximum quantization levels* (MQL) $m(i,j)$, which give the maximum quantization level for all values of the descendant nodes:

$$m(i,j)=\max_{(k,l)\in D(i,j)}\{q(k,l)\} \tag{4}$$

We denote the level q_{min} as the *minimum* quantization level, which decides about the compression rate for a complete picture. Table 2 summarizes the notation for quantization levels.

2.2.3. Coding/decoding of a number

In tree-based coding algorithms, there generally exists the need for binary encoding or decoding (i) an integer number referring to a wavelet coefficient and (ii) a quantization level referring to a maximum level of a group of coefficients. (For encoding of an integer number we take 16 bit variables as an input.) Generally, only a portion of bits of the binary representation of the number has to be

Table 2
Notation for the quantization levels of nodes.

$c(i,j)$	Value of the coefficient at (i,j)
$q(i,j)$	Quantization level of $c(i,j)$
$q_C(i,j)$	Maximum quantization level for the quadtree (i,j)
$q_G(i,j)$	Maximum quantization level of the grandchild nodes of (i,j)
$m(i,j)$	MQL node level for the quadtree at (i,j) and all descendants
q_{min}	Minimum quantization level for a picture

Table 3
Notation for the coding of coefficients and levels using the quantization levels Q_{min} and Q_{max}.

encode(c, Q_{min}, Q_{max})	Encode coefficient $c(\cdot)$
decode(c, Q_{min}, Q_{max})	Decode coefficient $c(\cdot)$
encodeL(q, Q_{min}, Q_{max})	Encode level $q(\cdot)$
decodeL(q, Q_{min}, Q_{max})	Decode level $q(\cdot)$

encoded. We define the portion with the minimum quantization level Q_{min} and the maximum quantization level Q_{max}. A quantization level of 0 refers to the least significant bit. We introduce the function **encode**(c, Q_{min}, Q_{max}) to encode a portion of the coefficient c starting at bit position Q_{min} and ending at position Q_{max}. Thus, there are $Q_{max}-Q_{min}$ bits encoded. In addition, a sign bit is coded. Similarly, the function **decode**(c, Q_{min}, Q_{max}) returns the decoded coefficient c. When decoding a coefficient, we use a rounding method to move the transformed value to the midpoint of the bin in which it lies, as it is proposed in [36].

A quantization level q is similarly encoded as a coefficient using a portion of bits from its binary representation, which is specified by a minimum level Q_{min} and a maximum level Q_{max}. The binary number generally has a sequence of zero bits and a single one bit, as it results from Eq. (1). The number is encoded starting from the bit position $\max\{q, Q_{min}\}$ to position Q_{max}. Note that the inclusion of q within the term $\max\{q, Q_{min}\}$ ensures that the least significant zero bits of q are not coded. The decoder accordingly starts decoding from the most significant bits and uses the single one as a stop bit. We define the functions **encodeL**(q, Q_{min}, Q_{max}) and **decodeL**(q, Q_{min}, Q_{max}) for the encoding and decoding of a quantization level q. When coding coefficients, Q_{min} generally equals the minimum quantization level q_{min} of the complete picture. A summary of the coding functions is given in Table 3.

2.3. Principle and reference implementation of backwards coding

In this work the principle of *backwards coding of wavelet trees* (BCWT) for wavelet-based picture compression is selected as a starting point. In the following we briefly review the BCWT method and describe our implementation of it, which serves as a reference to verify the new algorithm in Section 3. BCWT was introduced by Guo et al. [37] as a backward version of the *Set Partitioning in Hierarchical Trees* (SPIHT) algorithm [1]. Similarly than SPIHT, the BCWT algorithm utilizes the specific ordering of wavelet coefficients within trees. More specifically, coefficients tend to become smaller with more distance to the root of the tree. A part of the tree that only has insignificant values can be summarized, thereby achieving compression. The method starts with coding units of 16 wavelet coefficients within a subband from the first wavelet level on and moves up one level when all units of that subband have been encoded. (In spite of that method, the general forwards encoding starts from the top level.) When encoding one unit, it does not need to store the individual maximum quantization levels (MQL) of the quadtrees but only the maximum one. This MQL node is kept in the memory via a list to be retrieved when coding the next level. The encoded wavelet coefficients are not needed any more (for the detection of a maximum quantization level of a subtree) and the memory can be relieved. In the following we describe the encoding of a single BCWT unit. A BCWT unit includes a root node at (i,j), the four child nodes $C(i,j)$, and the set of the 16 grandchild nodes $G(i,j)$ to be encoded. The coding steps are given as follows:

1. Compute the (four) MQL levels $m(k,l), (k,l) \in C(i,j)$ with Eq. (4).
2. Compute q_G: $q_G(i,j) = \max_{(k,l) \in C(i,j)}\{m(k,l)\}$
3. Encode coefficients at $C(i,j)$ and the respective MQLs: If $q_G(i,j) \geq q_{min}$ do $\forall (k,l) \in C(i,j)$
   (a) if $m(k,l) \geq q_{min}$ do $\forall (u,v) \in C(k,l)$ encode $(c(u,v), q_{min}, m(k,l))$
   (b) encodeL $(m(k,l), q_{min}, q_G(i,j))$
4. Compute $m(i,j)$: $m(i,j) = \max\left\{q_{G(i,j)}, \max_{(k,l) \in C(i,j)}\{q(k,l)\}\right\}$ and put the result to the MQL list (needed for next level). The MQL nodes $m(k,l) \forall (k,l) \in C(i,j)$ can be deleted.
5. Encode the q_G level: If $m(i,j) \geq q_{min}$ encodeL $(q_G(i,j), q_{min}, m(i,j))$.

Decoding a BCWT unit works in the reverse way than for the encoding. Specifically, first the MQL level $m(i,j)$ is fetched from the list, then $q_G(i,j)$ is decoded. Then the $m(k,l), \forall (k,l) \in C(i,j)$ and the respective coefficients are decoded. Note that the $m(k,l)$ are put to the list to be retrieved when moving to the next lower level (the decoding starts at the top level). The decoder can detect a zerotree that prevents that the tree is scanned further a) with decoding $m(i,j) < q_{min}$ or b) with decoding $q_G(i,j) < q_{min}$. In our implementation we insert child MQL nodes of (i,j) into the list each time before a unit (i,j) is decoded and initialize them with -1 for a potential later detection of zerotrees.

For encoding of a complete wavelet picture, all units in the subbands of all levels have to be traversed using the described procedure for encoding of a BCWT unit. This is achieved by a main loop (for each subband *LH*, *HL*, and *HH*) that starts with the unit root node at wavelet level 3 (to span the levels 3,2,1) and goes up one level each time when all units in the current level are completed. The last three units that are encoded have their root nodes at $(0,1)$, $(1,0)$, and $(1,1)$. Using this loop, the 16 coefficients of the two highest levels (of all subbands) are not encoded. These coefficients can be encoded separately using a uniform quantization with the minimum quantization level q_{min} and the measured maximum quantization level of the complete wavelet picture. The measured maximum quantization level has to be encoded using a defined maximum level. Recall that the wavelet transform is usually not performed up to the highest possible wavelet level (which is given by $\log_2(N)$, where N gives the picture dimension). For instance, using the picture dimension $N=256$, we perform the transform up to level 6. In this case, the

remaining coefficients for which the uniform quantization should be performed relate to the 6th level *HH* subband.

The BCWT coding algorithm allocates memory for storage of the MQL list and needs to access the image within the defined units. A unit spans data of different wavelet levels, and does not accord to line-based access. Memory is required for $42 \cdot N$ coefficients and $3 \cdot N$ quantization levels. In the next section we introduce the wavelet image two-line coder, which similarly as BCWT encodes the image backwards but recursively reads lines of wavelet subbands to perform the compression, thereby reducing the memory requirements by an order of magnitude.

3. Wavelet image two-line coder

The standard backward wavelet coding traverses the image (wavelet-) level by level. It keeps a list of MQL level information for a complete subband in the memory which is retrieved when switching to the next higher level. In this section we introduce the *wavelet image two-line* coder (Wi2l) which uses a specific concept to line-wisely access the image in a recursive way. The concept allows to apply the wavelet coding with very little memory and is in line with the access attributes of flash memory. Similarly as with the BCWT algorithm, Wi2l assumes that the wavelet decomposition is already given (for instance using the low-complexity scheme mentioned in Section 2.1). In this section, we first introduce the basic principle of the proposed scheme. We define the required memory structures that link the level information between consecutively coded subband lines and extend the notation to access the respective information. Finally the function to encode two subband lines and a main procedure that calls this function are described.

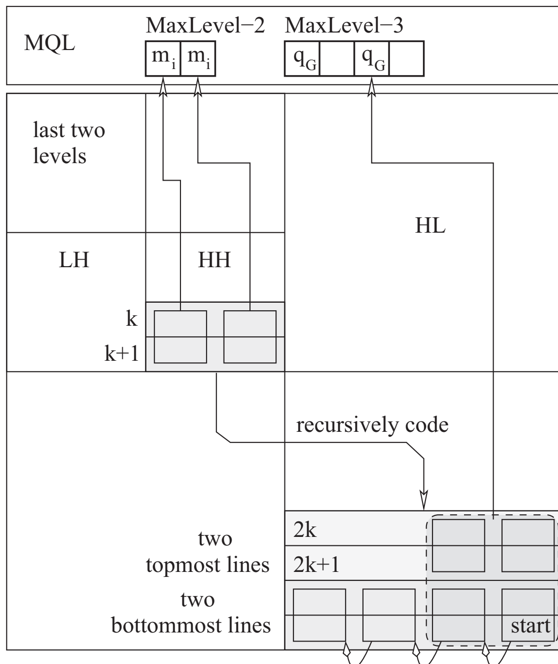

Fig. 4. Principle of the proposed Wi2l coder. The algorithm encodes blocks of four coefficients within two lines, where intermediate level information is stored in a small buffer.

3.1. Wi2l principle

The Wi2l scheme divides the image in units of *line quad sets*. Such a quad set is built of four consecutive lines of a wavelet subband (and contains the sets of 16 grandchild coefficients, see Section 2.2.1). We denote the lines of a quad set with the line index $l_4 = k \bmod 4$, where k denotes the absolute lines index within a wavelet subband (with $k=0$ denoting the topmost line). $l_4 = 0, 1$ thus denotes the two topmost lines of a quad set, and $l_4 = 2, 3$ the two bottommost lines. Fig. 4 illustrates the basic principle of encoding sets of two lines building a line quad set.

There is a difference in the encoding for the lines $l_4 = 2, 3$ and the lines $l_4 = 1, 2$. When encoding the lines $l_4 = 2, 3$, the scheme traverses the two lines in sets of four coefficients from right to the left. For each set, the MQL levels are computed. The coefficients of a set are encoded using the computed MQL levels, which are stored in the *MQL buffer*. Such a buffer exists for every wavelet level (the wavelet level here refers to a wavelet decomposition level, see Section 2.1). The lines $l_4 = 0, 1$ are traversed afterward, where sets of four coefficients are encoded similarly. The MQL buffer, however, does now serve to deliver the m_i levels of the previous two lines for computation of the q_G levels. The q_G levels in turn are stored in the MQL buffer (instead of the m_i levels) to be retrieved when encoding the corresponding two lines at the next higher wavelet level. Fig. 5 illustrates the usage of the MQL buffer to encode lines $l_4 = 2, 3$ (figure a)) and lines $l_4 = 0, 1$ (figure b)). A function call for encoding of two lines ensures via recursion that the required q_G levels are computed and stored in the MQL buffer. While the algorithm timely encodes the lines $l_4 = 2, 3$ before the lines $l_4 = 0, 1$, the

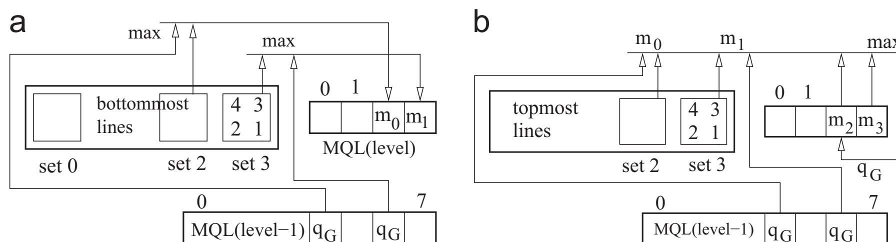

Fig. 5. Encoding of coefficients with the MQL level buffer, which in (a) for the bottommost two lines saves MQL level information, and in (b) for the topmost two lines allows to retrieve the previous levels to compute and save q_G levels.

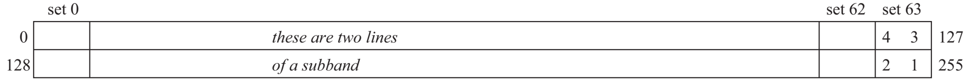

Fig. 6. Notation to access the coefficients of two lines for an example dimension of 256 coefficients.

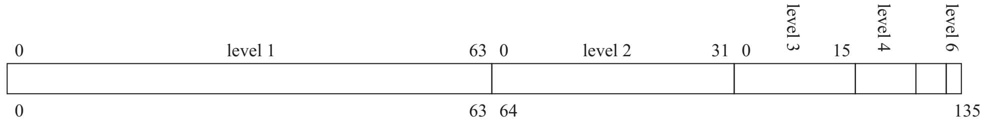

Fig. 7. Example of the data structure for the maximum quantization level (MQL) buffer for $N=256$. For each wavelet level i there exist $N/2^{i+1}$ entries of one byte.

Table 4
Extended notation to access the MQL and the wavelet coefficients. The little memory requirements of Wi2l are achieved via a linewise coefficient access pattern.

GetMQL(l, i)	Returns the quantization level for wavelet level l and $set(i)$
PutMQL(q, l, i)	Puts level q to the buffer at level l and $set(i)$
Read2Lines$(pic, band, k)$	Read two lines k and $k+1$ from picture pic at subband $band$
Write2Lines$(pic, band, k)$	Write two lines k and $k+1$ to picture pic at subband $band$

sets of two lines of a line quad set are not necessarily encoded in a sequence. The MQL buffer serves as a part of an optimization strategy to ensure that in the next wavelet levels, the quantization levels of the children do not need to be recomputed; instead they are found in the buffer. Note that in Fig. 5, the MQL level buffers of the previous wavelet level (indexed by $MQL(level-1)$) relate to different lines of the wavelet subband. In the following we give our notation to access subband lines and the MQL buffer.

3.2. Extension of the notation and memory requirements

The standard backwards coding technique traverses the image via units, which are defined by a root node (i,j). It encodes the set of coefficients $G(i,j)$. In contrast to the encoding of unit coefficients, Wi2l operates on two lines of a subband. We introduce a more suitable notation to access the coefficients of those lines. Fig. 6 illustrates the notation for two lines for an example dimension of 256 coefficients. Sets of four coefficients are accessed by the index $set(i), i=0,\dots,63$, where $set(0)$ relates to the most left set. The lines themselves are addressed by the line index k, where $k=0$ refers to the top line of a subband, and by a subband index $band, band=HL, LH, HH$. To access two lines of the picture pic from the line buffer we define the functions **Read2Lines**$(pic, band, k)$ and **Write2Lines**$(pic, band, k)$. The MQL buffer is addressed via the functions **GetMQL**(l, i) and **PutMQL**(q, l, i), where l refers to a wavelet level and i to a set i of four coefficients within the two lines. GetMQL() returns a quantization level q, while PutMQL() takes q as a parameter. An example data structure for an MQL buffer with $N=256$ is given in Fig. 7. It contains 135 different quantization levels, where for each wavelet level $l, l=0,\dots,6$, the set numbering starts with $set(0)$. Table 4 lists the defined functions of the extended notation.

The line and MQL buffers account for the memory requirements of Wi2l. The line buffer requires $N\cdot 2$ bytes, as two lines of a wavelet subband of a picture require $2\cdot N/2$ coefficients, where each coefficient takes two bytes (as it is motivated by the fixed-point number representation in Section 2). Note that this requirement is given by the subband lines of the lowest wavelet level, while for the higher levels only a part of the allocated memory is used. The dimension dim_{MQL} of the MQL buffer is given for a picture dimension N as

$$dim_{MQL} = \sum_{i=1}^{\log_2(N)-2} \frac{N}{2^{i+1}} = \cdots = \frac{1}{2}N. \tag{5}$$

(The expression converges as it is a geometric series of the form $\sum_{k=m}^{\infty} aq^k = aq^m/(1-q), |q|<1$.) The Wi2l coder also requires a binary buffer to access compressed data from the flash memory. At the sender side, the compressed bits are written to this buffer, and a full block is written to the flash memory. When the encoding process is finished, the compressed blocks are sent out in the reverse order. The intermediate storage on the flash memory allows to account for the unstable available wireless bandwidth. At the receiver side, such a buffer exists similarly, from which the compressed bits are read. We set the dimension of this buffer to 512 bytes to regard the block size of typical flash memory. The total memory requirements dim_{tot} for all buffers are thus given as

$$dim_{tot} = N\cdot 2 + \tfrac{1}{2}N + 512 = 2.5\cdot N + 512. \tag{6}$$

In the following a formal description of the Wi2l algorithm is given which uses the introduced notation to access subband lines and MQL level information.

3.3. Coding of two lines

The Wi2l principle of recursively encoding two wavelet subband lines is given in Fig. 8. The function *Code2Lines* takes a pointer pic to the picture, the wavelet subband ($band=HL, LH, HH$), the wavelet level l, and the line number $k, k=0,\dots,N$ as an input to encode the subband lines k and $k+1$. The steps for encoding are explained as follows.

Encode2Lines$(pic, band, l, k)$ {
//code line (k) and line (k+1):

1. if (level > 1)

   Encode2Lines$(pic, band, l-1, 2k+2)$

   Encode2Lines$(pic, band, l-1, 2k)$
2. Read2Lines$(pic, band, k)$
3. $DimMql = N/2^{l+1}$
   for $TwoSets = DimMql - 1, DimMql - 3, \ldots, 1$
   (a) for $i = 1, 0$ *//set 1 is right-hand set*
   $set(i) = TwoSets + i - 1$
      i. if $(l > 1)$ q_{Gi}=GetMQL$(l-1, 2 \cdot set(i))$
      else $q_{Gi} = -1$
      ii. $m_i = \max\left\{q_{Gi}, \max_{\forall j \in set(i)} \{q_j\}\right\}$
      iii. if $(m_i \geq q_{min})$
         A. If $(l > 1)$ encodeL(q_{Gi}, q_{min}, m_i)
         B. $\forall$ j in $set(i)$ do encode(c_j, q_{min}, m_i)
   (b) if $k/2$==odd *// bottommost lines*
      PutMQL$(m_0, l, TwoSets)$
      PutMQL$(m_1, l, TwoSets - 1)$
   else *// topmost lines*
      i. m_2 =GetMQL$(l, TwoSets)$
      m_3 =GetMQL$(l, TwoSets - 1)$
      ii. $q_G = \max\{m_0, m_1, m_2, m_3\}$
      iii. If $q_G \geq q_{min}$ for $i = 3, 2, 1, 0$ encodeL(m_i, q_{min}, q_G)
      iv. PutMQL$(q_G, l, TwoSets - 1)$

} *//end of function*

Fig. 8. Wi2l base function *Code2Lines* to recursively **encode two lines** of a wavelet subband.

Decode2Lines$(pic, band, l, k)$ { *//decode line (k+1) and line (k):*

1. $DimMql = N/2^{level+1}$
   $\forall j \in set(i)$ $c_j = 0, i = 0 \ldots DimMQL - 1$ *// Init coefficients*
2. for $TwoSets = 1, 3, \ldots, DimMql - 1$
   (a) if $k/2$==even
      i. q_G = GetMQL$(l-1, TwoSets - 1)$
      ii. If $q_G \geq q_{min}$ decodeL(m_i, q_{min}, q_G) for $i = 3 \ldots 0$
      else for $i = 3 \ldots 0$ $m_i = -1$
      iii. PutMQL$(m_2, TwoSets - 1)$
      PutMQL$(m_3, TwoSets)$
   else *//if k/2 is odd:*
      m_0 =GetMQL$(l, TwoSets - 1)$
      m_1 =GetMQL$(l, TwoSets)$
   (b) for $i = 0, 1$
      i. If $m_i \geq q_{min}$ do $\forall j \in set(i)$ c_j = decode(q_{min}, m_i)
      ii. If $(l > 1)$
         A. if $m_i \geq q_{min}$ q_{Gi} = decodeL(q_{min}, m_i)
         else $q_{Gi} = -1$
         B. PutMQL$(q_{Gi}, l-1, 2 \cdot (TwoSets + i - 1))$
3. Write2Lines$(pic, band, k)$
4. If $l > 1$

   Decode2Lines$(pic, band, l-1, 2k)$

   Decode2Lines$(pic, band, l-1, 2k+2)$

} *//end of function Decode2Lines()*

Fig. 9. Counterpart of the recursive Wi2l encoding function for **decoding two lines** of a wavelet subband.

In step (1) the recursive encoding of the lines $2k+2$ and $2k$ is performed. This ensures that the appropriate tree level information for the lines k and $k+1$ is gathered and stored in the MQL buffer. The selection of the line numbers $2k+2$ and $2k$ is due to the wavelet quadtree structure, which is denoted in Section 2. The lines to be encoded recursively contain the descendant coefficients. In step (2) the input lines k and $k+1$ are read into the line buffer. In step (3) the main loop for encoding of coefficients and levels is initiated. The variable *DimMql* is set for the current wavelet level as the number of MQL levels that exist for the two current coefficient lines, specifically, it equals half the dimension of a line as one MQL level relates to a set of four coefficients. The loop traverses the two lines in sets of eight coefficients from right to left. Such a set relates to one half of the grandchildren of a quadtree root node. The variable *TwoSets* relates to the right-hand set of four coefficients, and *TwoSets*−1 to the left-hand set. In (3a) these two sets denoted by $set(i)$ are processed, starting with the right-hand one. In (i) the corresponding maximum quantization level for the grandchildren of set i is denoted as q_{Gi} and retrieved via the function *GetMQL()*. We note that the maximum quantization level q_{Gi} relates to the notion of $q_G(i,j)$ in Table 2 with the difference that sets of four coefficients are denoted by the set index i (instead of the indices (i,j) for the root node of the set), see Section 3.2.

Similarly as with the recursive function call, the quantization level information q_{Gi} is found in the next lower level MQL buffer, where the set number is multiplied by two as there exist twice as much entries for that buffer, from which each second one contains a q_{Gi} level. In case of the first wavelet level the q_{Gi} does not exist, and it is set to minus one to signal the end of the tree. In (ii), the MQL level m_i for the current set of coefficients is calculated as the maximum of their four quantization levels and q_{Gi}. In case m_i is significant, the q_{Gi} of the previous level and the four set coefficients c_j are encoded in step (iii). For both, the m_i serves as an upper bound for the encoding.

The individual subunits of eight coefficients are encoded sequentially within the bottommost and the topmost two lines. Wi2l links the subunits via the MQL buffer to preserve the context of the larger units of 16 coefficients. In step (3b) it is detected if the lines k and $k+1$ refer to the bottommost or the topmost two lines. In case of the bottommost lines, the previously computed MQL levels m_1 and m_2 are saved in the MQL buffer at the positions *TwoSets* and *TwoSets*−1. In case of processing the topmost lines, the MQL levels stored for the bottommost lines are retrieved in (i) and (ii). Note that while the bottommost and the topmost lines relate to four consecutive subband lines, they are not necessarily encoded in a sequence. In step (iii) the maximum level q_G of the four MQL levels $m_i, i = 0, \ldots, 3$, are calculated. This level relates to the set of four lines, while the level $q_{Gi}, i = 1, 0$, encoded in (3a) (iii) relates to the next lower level (more deeply located in the tree). In step (3b) (iv) the levels $m_i, i = 3, \ldots, 0$, are encoded using q_G as an upper bound. In step (v) the q_G level is stored to the MQL buffer to be retrieved within coding of the next higher level.

3.4. Decode two lines

The Wi2l-function *Decode2Lines()* for decoding of two lines is given in Fig. 9. It takes similarly as the counter-

part encoding function a pointer *pic* to the image to be decoded, the wavelet subband *band* and level l, and the line index k as input parameters. The encoded bit stream is interpreted by this function in the reverse order, such that the decoding of the lines k and $k+1$ starts from the top level. In our implementation, the reversal is performed on the sender side when the compressed bit stream is read from the flash memory (which serves as an intermediate storage).

In step (1) the coefficients in the line buffer are initialized with zero. This ensures that insignificant sets of coefficients will have a defined value. In step (2) the main loop to decode all sets of coefficients within the two lines and related levels is given. The dimension *DimMql* of the MQL buffer is set as before in the encoding function, and the *TwoSets* variable takes the reverse range of values. In (2a), the MQL levels for the topmost two lines ($k/2$ is even) and the bottommost two lines ($k/2$ is odd) are retrieved. Note that we denote the sets of lines with *bottommost* and *topmost* the same way as with the encoding, and that therefore the decoding starts with the topmost lines. For the topmost lines, the q_G for the previous wavelet level is retrieved in (2a) (i). This level is used as the upper bound for decoding of the levels $m_i, i=3,\ldots,0$, in (ii). The levels m_0 and m_1 are stored to the MQL buffer in (iii). These are retrieved in (2a) in case of the bottommost lines ($k/2$ is odd), however, in that case they are denoted with m_0 and m_1.

In step (3b) the decoding actions for the sets $TwoSets-1$ ($i==0$) and $TwoSets$ ($i==1$) are taken. If m_i is significant, the coefficients $c_j, j \in set(i)$ are decoded and written to the line buffer. In step (ii), the q_{Gi} of the previous wavelet level is retrieved and saved in the MQL buffer. The decoded lines are written to the flash memory in (3). In (4), the recursion to decode the four lines $2k,\ldots,2k+3$ is performed. The recursion has to be the last step, as the decoding function which starts with the recursion operates the reverse way than the encoding function. The recursive functions decode coefficients within the next lower level and will find the required q_{Gi} information retrieved in step (2b) (ii) in the MQL buffer.

3.5. Main loop

The Wi2l coding functions each code the two lines k and $k+1$ of a subband. In the following we give a description of the initial calls of these functions to ensure that the complete picture is recursively encoded. A general consideration is that the recursive coding only makes sense within the typical wavelet tree structure – that means, to appropriately encode the image, the level until which the wavelet transform has been performed has to be regarded. As stated in Section 2, the transform is generally not computed until the highest possible level. The main procedure for encoding a complete picture is given as follows:

1. $list=\{HL, LH, HH\}$
2. for $i=1,2,3$
   (a) Encode2Lines ($list(i), MaxLevel, k=2$)
   (b) $q_G(i)$=Encode2Lines ($list(i), MaxLevel, k=0$)
3. EncodeLL ($q_G(i=1,2,3)$)

For instance, if the picture dimension is N=256, the maximum possible level would be $\log_2(256)=8$, and the third highest possible level would be given as $MaxLevel=8-2=6$. If the picture is transformed until the third highest level, there remain 16 coefficients at the remaining top levels. Therefore, the main encoding function traverses the lines $k=0,\ldots,3$ in the third highest subbands via two encoding function calls in step (1), which ensures that all successor lines are encoded. The procedure has to be performed for each subband HL, LH, and HH. If the transform is only computed until the next lower wavelet level (which in case of N=256 would be level 5), the lines $k=0,\ldots,7$ would have to be encoded. For each set of four lines, a maximum level $q_G(i)$ has to be returned. This level has to be encoded separately, as there is no recursive function that takes care of it. In step (2), the remaining coefficients of the highest LL subband and the $q_G(i)$ returned by the previous function calls are encoded via the function *EncodeLL()*. Both use the maximum quantization level of the complete picture as an upper bound. The maximum quantization level in turn has to be encoded using a defined maximum level given in the implementation.

Note that while the *Encode2Lines()* functions are called from the highest wavelet levels, the actual encoding of coefficients starts at the first wavelet level due to the recursion. This allows the decoding to start from the highest wavelet level by interpreting the reversed bit stream. The main decoding function first decodes the highest LL subband and for each subband $i=1,2,3$ a $q_G(i)$ level, for which we define the function *DecodeLL()*. For each set of four lines to be decoded in the following there exists such a q_G level. The decoding of the subband lines for each subband HL, LH and HH is performed in step (2). The main decoding procedure is given as follows:

1. $q_G(i=1,2,3)$=DecodeLL()
2. $list=\{HL, LH, HH\}$ for $i=3,2,1$
   (a) Write $q_G(i)$ to appropriate buffer position
   (b) Decode2Lines ($list(i), MaxLevel, k=0$)
   (c) Decode2Lines ($list(i), MaxLevel, k=2$)

Specifically, in (2a) the $q_G(i)$ level is written to the MQL buffer to be found by the decoding functions for the lines $0,\ldots,3$ called in b). In case the highest LL subband would be at the fourth highest possible wavelet level, two q_G levels would have to be stored in the buffer, and *Decode2-Lines()* would be called four times for each subband.

4. Evaluation of the Wi2l algorithm

In this section we evaluate the functionality and performance of the proposed Wi2l algorithm. As the Wi2l coding operations are performed on the wavelet-transformed image, we first measure in Section 4.1 the quality loss resulting from the fixed-point transform that is used in this work to verify the usability of Wi2l on a low complexity

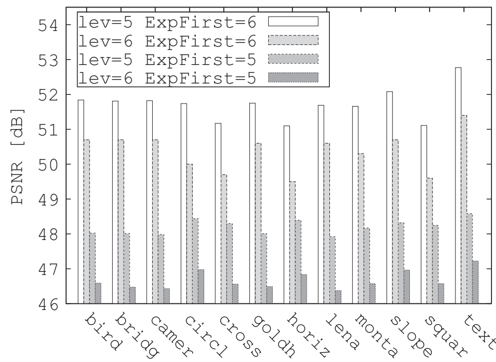

Fig. 10. PSNR performance of the low-memory **wavelet transform scheme** used in this work. The fixed-point calculations introduce a loss of PSNR quality, which is expected to be not relevant for high compression rates.

platform. In Section 4.2 our backwards coding reference implementation BCWT is checked for its proper operation. We have implemented the BCWT algorithm to verify the principle of backwards coding, which we use as a starting point to design a purely line-based algorithm. In Subsection 4.3 we compare the compression performance of the new Wi2l coder to the SPIHT algorithm. Then, in Section 4.4, the compression performance of Wi2l is compared to the standards JPEG, JPEG 2000, and the Google WebP format. Last, the encoding and decoding times are measured on a typical low-cost 16 bit microcontroller.

4.1. Performance of the fixed-point wavelet transform

In Section 2, we have shortly described the fractional wavelet filter – a low complexity wavelet transform that combined with the new Wi2l coding algorithm builds a compression system that can operate on very limited platforms. The fractional wavelet filter, while it allows for line-wise computation of the wavelet transform, introduces a quality loss as it transforms the floating-point filter operations to fixed-point arithmetic (thereby featuring fast computation on hardware without floating-point support). Fig. 10 shows the PSNR performance for the pictures from the *GraySet1* test images from the *Waterloo Repertoire* (available at http://links.uwaterloo.ca/). This set contains twelve grayscale images with 256×256 pixels ($N = 256$) and 8 bits per pixel. The images are given in the portable network graphics (PNG) format and converted to plain text numbers of the type *signed character* using the software suite *ImageMagick* and *Octave*. For each image, a forward transform is computed followed by a reverse transform. For each transform, four quality values are calculated for the maximum wavelet levels $lev = 5, 6$ and the fixed-point number representation format used for the first wavelet level is selected as ExpFirst=5,6. *ExpFirst* gives the number of bits used for the fractional portion of the number, as explained in Section 2.

From the figure it can be concluded that the choice of six bits for the fractional part (*ExpFirst*=6) results in less loss of image quality, as quality values in the range of 49.6 and 51.25 and 51.2 to 52.8 dB are achieved for the levels 6 and 5, respectively. Using five bits for the fractional part (*ExpFirst*=5) results in values between 46.2 and 47 dB for level 6 and 48 and 48.3 for level 5. The better qualities for more fractional bits are due to the higher precision of the calculations. We have included the *ExpFirst*=5 representation as especially for the wavelet levels $lev = 1, 2, 3$ the actual wavelet coefficients exceed the possible number range for the representation *ExpFirst*=6 (measurement results not given here). The quality gain with more fractional bits holds true as only a small number of coefficients exceed the range. Regarding the maximum level until the transform is computed, the higher level introduces more loss as more computations have to be performed for it.

In the following sections, the question will be answered if the higher level can provide better compression for the higher compression rates. Regarding the Wi2l compression, we expect that the quality loss due to the transform is in general not visible for the high compression rates, as the coding cuts of the least significant bits. For the low compression rates (high picture qualities), as it will be illustrated in Figs. 12 and 13 in Section 4.3, the loss due to the fixed-point arithmetic results in an image quality saturation for the Wi2l algorithm at approximately 50 dB.

4.2. Verification of the BCWT reference implementation

As the recent technique of wavelet based backwards coding seems to be promising in terms of memory resource consumption, we have implemented the BCWT algorithm to verify the basic principle and as a reference for the new Wi2l algorithm (which shall not give lower compression performance). In this section, the achieved PSNR quality with BCWT is compared to the quality achieved with SPIHT. BCWT is expected to give the same compression performance as SPIHT while only a smaller part of the image needs to be kept in the memory. Similarly as in the previous section, we use the *GraySet1* test images. However, instead of a low complexity wavelet transform, we use a floating-point transform (computed with the software *Octave*) as the SPIHT software similarly uses floating-point numbers. To perform the SPIHT compression we use the software provided by A. Said and W.A. Pearlman (available at http://www.cipr.rpi.edu/research/SPIHT), specifically the commands *fastcode* and *fastdecd*. A PSNR value for BCWT is computed via a forwards wavelet transform, an encoding with the BCWT implementation, a decoding with BCWT, a reverse transform, and a PSNR calculation using the original picture and the reconstructed one. Fig. 11 gives the compression performance for the wavelet levels *lev*=5,6 and the minimum quantization levels $q_{min} = 0, \ldots, 8$. The compression rate is given in bits per bytes (bpb) and is here equivalent to the bits per pixel (bpp) metric, as one image pixel is represented by eight bits. Recall that a larger q_{min} parameter results in higher compression (and lower PSNR quality).

From the figures it can be seen that BCWT gives the same compression performance than SPIHT. On one hand, it can be concluded that the principle of backwards coding fulfills the requirement of maintaining the same entropy than SPIHT, as the algorithm reorders the bit stream such that a compressed result can be generated for a set of picture input lines. On the other hand, it also verifies the

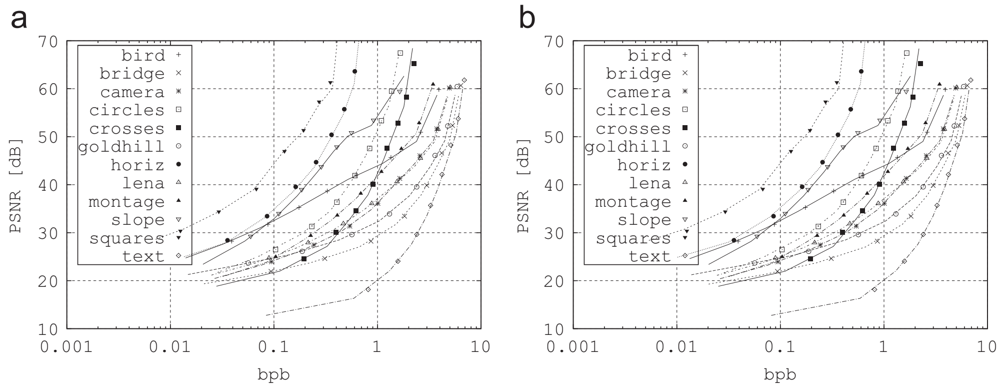

Fig. 11. PSNR picture quality over bits per byte (bpb) for the **BCWT** reference implementation (lines) compared to **SPIHT** (dots) with the different quantization levels $q_{min} = 0, \ldots, 8$ and the maximum wavelet levels $lev = 5$ and 6 in (a) and (b), respectively.

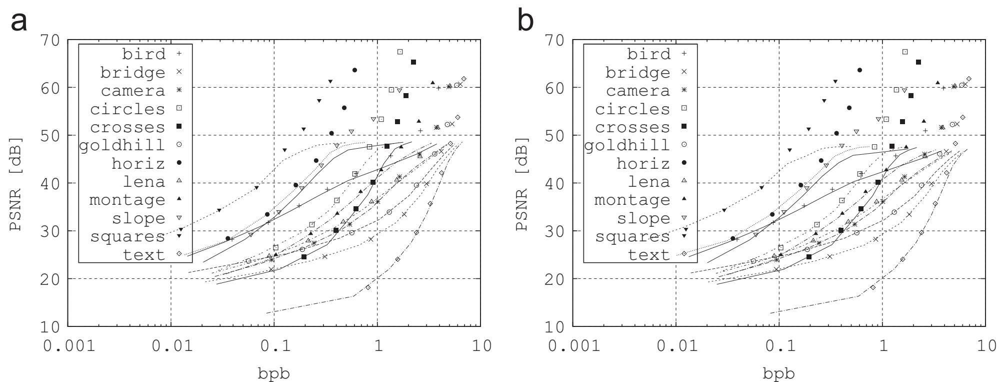

Fig. 12. Compression performance of the **Wi2l** coder (lines) compared to the performance of **SPIHT** (dots) for quantization levels $q_{min} = 0, 1, \ldots, 8$, the maximum wavelet levels $lev = 5$ and 6 in (a) and (b), respectively, and **five fractional** bits. The loss of quality is only due to the employed fixed-point transform – Wi2l results in the same entropy than SPIHT (and BCWT).

proper operation of our BCWT implementation, which we used as a reference to derive new schemes. In contrast to the given SPIHT software (given as an executable), the BCWT implementation allows for more flexibility, as different types of wavelet transforms can be used and parameterized. In the next section, the performance of the Wi2l scheme is compared to SPIHT.

4.3. Wi2l compression results in comparison to SPIHT

In this section, the PSNR quality achieved by our implementation of the proposed Wi2l coding scheme for different compression rates is measured and compared to the results achieved by SPIHT, while the same methodology than in Section 4.2 is used. That includes the forward transform, encoding for the different quantization levels q_{min}, $q_{min} = 0, \ldots, 8$, reverse transform, and decoding. The *GraySet1* test images are used again, and PSNR values computed as before. Fig. 12 illustrates the results for five fractional bits (*ExpFirst*=5) and the wavelet levels 5 and 6 in figures (a) and (b), respectively.

The compression performance of Wi2l until 43 dB is almost identical to SPIHT. At 48 and 47 dB the Wi2l compression stays constant, while SPIHT proceeds giving better qualities. This is due to the employed fixed-point wavelet transform; the maximal PSNR values are in line with the results for the transform given in Section 4.1. The higher wavelet level 6 gives marginal better qualities for the high compression rates, as particularly visible for the Squares image at 0.01251 bpb. Results for six fractional bits (*ExpFirst*=6) are given in Fig. 13.

The given PSNR values are similar as for five fractional bits with the difference that now higher qualities up to 52 and 51 dB for levels 5 and 6 are achieved. This is again caused by the fixed-point wavelet transform calculations, which can utilize more bits to achieve more precision. The results are confirmed by the PSNR values in Section 4.1. The performance loss of Wi2l versus SPIHT is thus due to the low complexity transform and not due to the proposed coding scheme. The specific transform is employed here to highlight a solution for a complete compression system, including transform and coding, which can be applied to a low complexity platform.

4.4. Wi2l compression results in comparison to JPEG, JPEG 2000, and WebP

In this section we give example results for the compression of Wi2l compared to JPEG, JPEG 2000, and the recent Google WebP format. The results for JPEG and JPEG

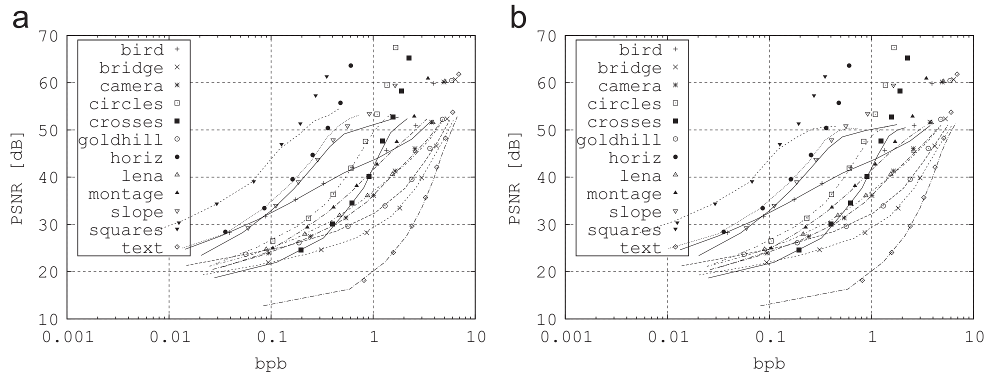

Fig. 13. Compression performance of the **Wi2l** coder (lines) compared to the performance of **SPIHT** (dots) for quantization levels $q_{min} = 0, 1, \ldots, 8$, the maximum wavelet levels $lev = 5$ and 6 in (a) and (b), respectively, and **six fractional** bits.

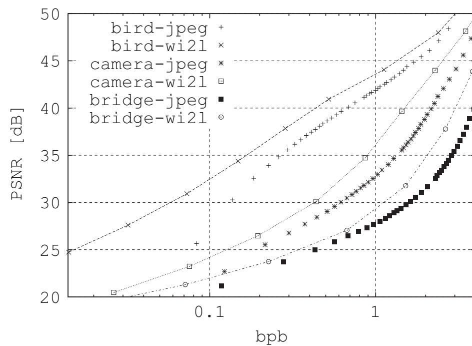

Fig. 14. Compression comparison of Wi2l (lines points) to **JPEG** (dots).

2000 were obtained using the *jasper* library, see http://www.ece.uvic.ca/mdadams/jasper/; the given results for JPEG 2000 are implementation specific. We have used jasper with the default configuration, which is not necessarily tuned for visual performance. For WebP a command-line executable is available from Google. Fig. 14 gives the PSNR values for Wi2l (lines) and JPEG (dots) for the selected example pictures Bird, Camera, and Bridge. There is a significant quality gain visible for Wi2l over the complete range of compression rates, which can be up to 5 dB. This is expected, as the JPEG algorithm is outdated (yet it is still used by most in practice). JPEG typically uses a quantization matrix that is optimized for visual and not for objective quality (PSNR); perceptual quality metrics may be more appropriate to illustrate the gain. The results are included to indicate the potential gain due to wavelet-based techniques especially in the sensor networks, where data rates and communication energy within the network are limited.

Fig. 15 gives the results for the comparison of Wi2l to JPEG 2000 in (a) and the Google WebP format in (b). The example pictures Bird, Lena, Camera, Goldhill, and Bridge from the GraySet1 are selected. For the Wi2l measurements, six fractional bits and wavelet level 5 are used. JPEG 2000 gives significantly less PSNR quality, especially for the bird image and the high compression rates in general. For the Bird and the Lena image, however, it is also visible that for the compression rates from 2 bpb on, JPEG 2000 gives slightly better results. We note that these examples are not representative, and that JPEG 2000 achieves for a wide range of resolutions and picture types (including artificial or medical images) in general better results than SPIHT or the in this paper advocated low complexity wavelet technique, see [3]. For a low-cost sensor node, however, Wi2l is the more suitable technique, as more likely low-resolution and natural images only will be applied. This is due to the camera sensors in this kind of marketplace, which are rather of low cost. In sensor networks it is also customary to use a larger set of low-cost sensors instead of a few more expensive high-resolution cameras.

Fig. 15(b) gives the PSNR results for WebP and Wi2l. WebP in general performs slightly better than Wi2l, especially for the Camera image, where the difference is roughly 1 dB over the entire range of compression values. For the Bird, the Goldhill, and the Bridge image, Wi2l gives slightly better results for high compression rates smaller than 0.3 bpb. The examples indicate that WebP is rather designed for web-usage, where lower resolution images are more likely. The WebP technique is based on the intra-frame-coding of the video VP8 codec and uses a block-based transform together with a modern entropy coding [2]. We note that Wi2l does not use entropy coding in order to save computations and energy, while in principle such a technique would also slightly improve the Wi2l compression performance. Furthermore, the given code for WebP is not applicable to sensor nodes as it is designed for PC-usage. In the next subsection the coding times on a microcontroller will be measured to verify the applicability of Wi2l.

4.5. Computational complexity analysis

In order to estimate the coding time of the Wi2l algorithm on an arbitrary platform we calculate the number of required operations for encoding of one picture with the dimension $N \cdot N$. We first consider the BCWT algorithm, which serves as a reference and then continue with the proposed Wi2l scheme. We note that the memory locality of the considered coding algorithms is not taken

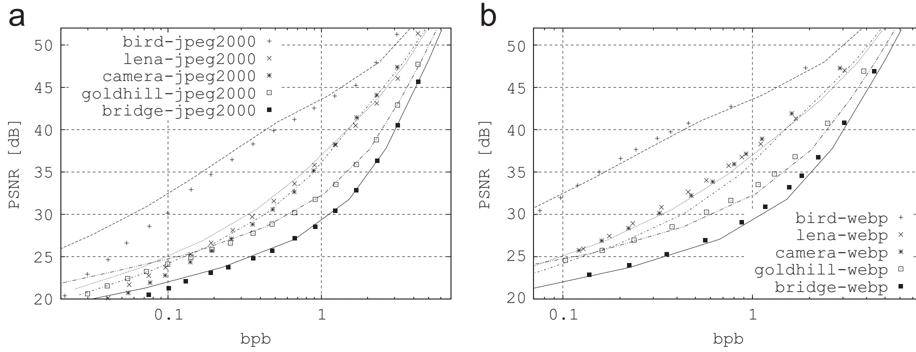

Fig. 15. Compression comparison of Wi2l (lines) to **JPEG 2000** (points) in (a) and **Google WebP** (points) in (b).

into account by the following analysis, while it can indeed effect the coding time. The aim of the given analysis is however to prove that Wi2l does not introduce additional coding bits when compared to BCWT. Regarding the (flash) memory locality, Wi2l should give advantages over BCWT due to its linewise processing.

BCWT and Wi2l both are low-complexity coding algorithms, thus the coding time of either algorithm is not determined by mathematical coefficient computations but by the specific access patterns for the coefficient and intermediate information. Retrieval of coefficients and writing of binary information is done through flash memory card access, which requires a software library for reading and writing of blocks of 512 bytes, while intermediate information is accessed via a specific memory structure (located in the RAM or heap memory). In order to cover the time-consuming coding operations, we denote r, c, and cl for *reading*, *coding* a coefficient and a *coefficient level*, which all relate to flash memory access. g and p serve to indicate the operation *get* and *put* for wavelet level information (one byte or less), which both refer to RAM memory access.

For the calculation of coding coefficients and levels we consider the worst case, that is, the image is not compressible at all. We note that the general way of processing for both algorithms is not affected by the image compressibility – that means, among others, the number of coefficient reading operations stays constant for any image.

4.5.1. BCWT coding algorithm

BCWT encodes units of $4\cdot4$ coefficients. We first consider (1) the effort for encoding a single unit, then use that result to estimate (2) the effort for encoding of a complete wavelet subband, and finally can estimate (3) the effort for encoding all levels of a subband and (4) the complete image with all subbands.

(1) Encoding of a single unit at root (i,j): According to the description in Section 2.3, BCWT requires read operations of $4\cdot4$ coefficients at $G(i,j)$ and four get operations (which result from coding operations at the next lower level) to compute the MQL levels of the coefficients. Then the 16 coefficients are coded ($16c+4cl$ operations). Last, the MQL level for the nodes at $C(i,j)$ is stored in the MQL list and the MQL level for the nodes at $G(i,j)$ is coded. The resulting computations $\varphi(u)$ for coding a unit are given as follows:

$$\begin{aligned}\varphi(u) &= 16r+4g+16c+4cl+4r+p+cl\\ &= 20r+4g+p+16c+5cl\end{aligned} \tag{7}$$

(2) Encoding of one level in a subband: A subband at level l has $N^2/(4^l)$ coefficients. The number of units (each of them with 16 coefficients) per level in a subband is given as $(1/16)(N^2/4^l)$. Thus, the operations $\varphi(l,N)$ for encoding of one level in a subband are given as

$$\varphi(l)(N) = \frac{1}{16}\frac{N^2}{4^l}\varphi(u) = \frac{1}{16}\frac{N^2}{4^l}(20r+4g+p+16c+5cl). \tag{8}$$

(3) Encode all levels of a subband: The operations $\varphi(N)$ for encoding of all levels $(l=1,\ldots,N-2)$ are given as

$$\begin{aligned}\varphi(N) &= \sum_{l=1}^{\log N-2}\varphi(l)(N) = \sum_{l=1}^{\log N-2}\frac{N^2}{4^{l+2}}(20r+4g+p+16c+5cl)\\ &= \frac{N^2}{16}(20r+4g+p+16c+5cl)\sum_{l=1}^{\log N-2}\frac{1}{4^l}\\ &\approx \frac{1}{3}\frac{N^2}{16}(20r+4g+p+16c+5cl).\end{aligned} \tag{9}$$

(4) Encoding all subbands: For the total number of operations $\phi_{\mathrm{BCWT}}(N)$ for encoding a complete image all the three subbands HL, LH and HH have to be considered:

$$\phi_{\mathrm{BCWT}}(N) = 3\varphi(N) = \frac{N^2}{16}(20r+4g+p+16c+5cl) \tag{10}$$

The small number of remaining coefficients in the LL subband (16 coefficients) is encoded separately and not relevant for our analytical considerations.

4.5.2. Wi2l coding algorithm

The description in Fig. 8 serves as a basis for the following considerations. First, the computations required for (1) the coding of two lines are derived. Then, the (2) coding of a complete level in a wavelet subband is

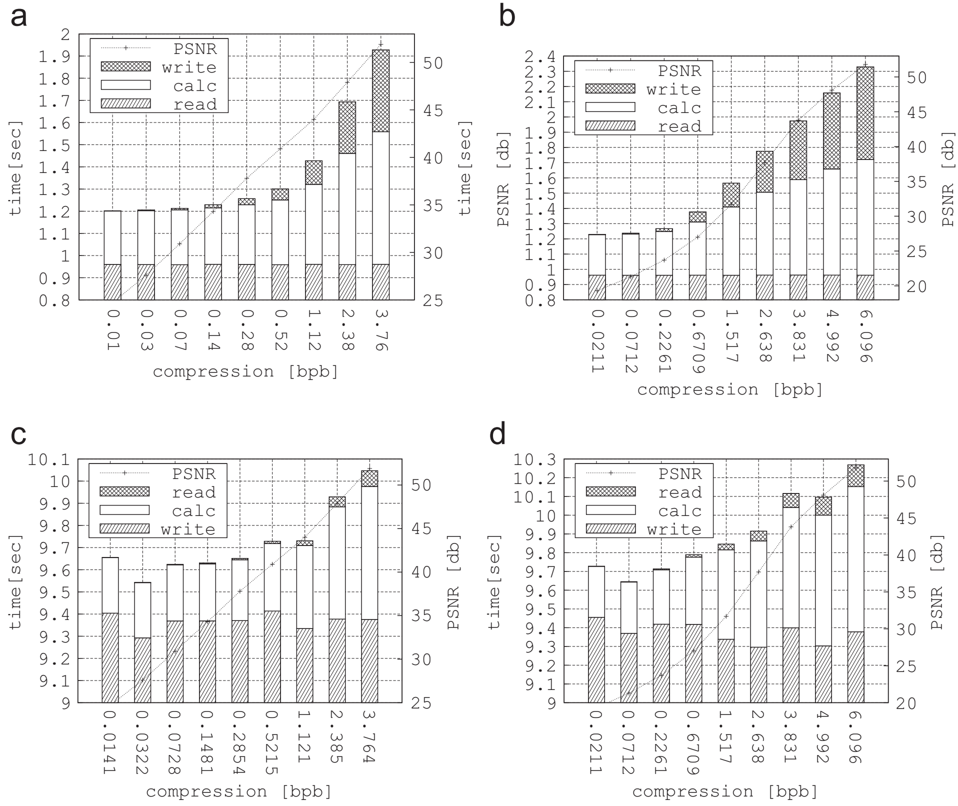

Fig. 16. Encoding (upper two plots) and **decoding times** (lower two plots) categorized by SD-card access and operational time for the images Bird in (a) and (c) and Bridge in (b) and (d). Measurements were conducted on a 16 bit microcontroller from the Microchip dsPIC family.

considered. Finally, the computations (3) for all levels of a wavelet subband and (4) for a complete image are given.

(1) Encoding of two lines: Let $\varphi(N,l)(l_4=2,3)$ denote the number of operations required to encode the lines $l_4=2,3$ of a wavelet subband with N coefficients in each line at wavelet level l. There exist $N/2^{l+1}$ sets of coefficients (with each set containing 4 coefficients) in the two lines, which require $2(N/2^l)$ *read* operations. For each set, a *get* operation for $q_{G_i}(l-1)$ is required to compute the MQL for the respective set in the previous level; then the retrieved quantization level $q_{G_i}(l-1)$ and each of the four coefficients are encoded. The MQLs for each set are stored in the quantization level memory (*put* operation). The total number of the respective operations for $l>1$ is given as

$$\varphi(N,l>1)(l_4=2,3)=2\frac{N}{2^l}r+\frac{N}{2^{l+1}}(g+cl+4c)+\frac{N}{2^{l+1}}p$$

$$=\frac{N}{2^{l+1}}(4r+g+p+4c+cl). \tag{11}$$

There is an exception for the first wavelet level: As there are no descendant coefficients in lower levels, there is no retrieval of previous level information and no coding of that level (see Fig. 8: (3a) (i) and (3a) (iii) A). Therefore, for $l=1$, Eq. (11) is modified as follows:

$$\varphi(N,l=1)(l_4=2,3)=\frac{N}{4}(p+4c+4r). \tag{12}$$

For the image lines $l_4=0$ and $l_4=1$ (topmost lines), there is a difference in the last additive term, see Fig. 8: (3b)): Instead of two *put* operations, there are now two *get* operations, a *cl* and one *put* operation. Thus, the computational requirements for $l>1$ are given as

$$\varphi(N,l>1)(l_4=0,1)=2\frac{N}{2^l}r+\frac{N}{2^{l+1}}(g+cl+4c)$$

$$+\frac{N}{2^{l+2}}(2g+4cl+p)$$

$$=\frac{N}{2^{l+2}}(8r+4g+6cl+8c+p), \tag{13}$$

and for $l=1$ as

$$\varphi(N,l=1)(l_4=0,1)=\frac{N}{2^3}(2g+4cl+8c+p+8r). \tag{14}$$

(2) Coding of a complete level in a wavelet subband: Let $\varphi(N,l)$ denote the total number of operations for encoding one complete level of a wavelet subband with $N/2^l$ lines. With Eqs. (11) and (13) the total number of operations for $l>1$ is given as

$$\varphi(N,l>1)=\frac{1}{4}\frac{N}{2^l}\varphi(N,l>1)(l_4=2,3)+\frac{1}{4}\frac{N}{2^l}\varphi(N,l>1)(l_4=0,1)$$

$$=\frac{N^2}{4^{l+2}}(6g+3p+16c+8cl+16r), \tag{15}$$

and with Eqs. (12) and (14) for $l=1$ as

$$\varphi(N, l=1) = \frac{N^2}{64}(16r + 2g + 3p + 16c + 4cl). \quad (16)$$

(3) For all levels l of a subband, that is, for $l = 1, \ldots, (\log_2 N) - 2$ (the last two wavelet levels with $8 \cdot 8 = 64$ coefficients are encoded separately as a single unit) the total number of operations $\varphi(N)$ is given as

$$\begin{aligned}\varphi(N) &= \sum_{l=2}^{\log_2 N - 2} \varphi(N, l > 1) + \varphi(N, l=1) \\ &= \frac{N^2}{16}(6g + 3p + 16c + 8cl + 16r) \sum_{l=2}^{\log_2 N - 2} \frac{1}{4^l} \\ &\quad + \frac{N^2}{64}(16r + 2g + 3p + 16c + 4cl) \\ &\approx \frac{N^2}{64}\left(4g + 4p + \frac{64}{3}c + \frac{20}{3}cl + \frac{64}{3}r\right). \end{aligned} \quad (17)$$

(4) Complete image: We denote $\phi_{\text{Wi2l}}(N)$ as the total number of operations required for the encoding of a complete image with dimension $N \cdot N$ including the 3 subbands HL,LH and HH:

$$\phi_{\text{Wi2l}}(N) = 3\varphi(N) = \frac{N^2}{16}(16r + 3g + 3p + 16c + 5cl). \quad (18)$$

4.5.3. Interpretation of the analytical results

A comparison of the final results for BCWT and Wi2l (Eqs. (10) and (18)) reveals that the number of coding operations is the same for both algorithms, that is, the coding includes N^2 coefficients and 5/16 levels per coefficient for each algorithm. Regarding the coefficient reading, BCWT reads each coefficient 20/16 times, while Wi2l reads each coefficient just one time. Wi2l can achieve this due to its different coding order with the help of a small buffer for quantization level information. Regarding the RAM access, Wi2l requires one *get* operation less and two *put* operations more than BCWT. We can conclude that the computational complexity for Wi2l is nearly the same than for BCWT. The advantage of Wi2l is that it works with much less RAM memory (small buffer for MQL information) and that the coefficient access via two image lines is suitable for fast utilization of flash memory: Wi2l reads and writes the data in sequential order to the flash memory; if the read and write access times for fixed data units are known for the target platform, the given analytical equation can be utilized to predict the coding time.

4.6. Wi2l coding time measurements

For the coding time measurements we apply the *Open-Sensor* platform [38] illustrated in Fig. 1, which employs a 16-bit microcontroller with 2 kB RAM from the so-called *dsPIC* family from Microchip (dsPIC4013) at 29 million instructions per second (MIPS). The sensor node has a slot for a standard secure digital/multimedia (SD/MMC) card. The wavelet-transformed pictures Bird and Bridge (level 5 and five fractional bits) are copied to the card and accessed via the C-library in [34]. (The low complexity wavelet transformed can also be computed on the sensor and the respective computation times are given in [12].) The wavelet picture is accessed line-wisely from the card in blocks of 512 bytes. The compressed binary stream is read and written similarly in 512 byte blocks. Fig. 16 shows the encoding and decoding times for the 256×256 images Bird and Bridge and the quantization levels $q_{min} = 0, \ldots, 8$. For this kind of evaluation the type of the image is of less importance, as all pixels have to be accessed. The different compressibility is however reflected in the results; specifically, the encoding time is shorter if less coefficients have to be encoded.

The processing times are given for SD-card read- and write-access and calculation times, and the bars in the plot are hatched accordingly. To give an indicator for the relevance of the coding times, the achieved PSNR qualities are also included in the plots. This allows to select a suitable compression rate with respect to the time requirements of the application.

As visible in the plots (a) and (b), the encoding times are in the range between 1.2 and 2.3 s. The reading time of approximately 0.95 s stays constant over the entire range of compression rates, while calculation and writing times increase with better PSNR qualities. The calculation takes between 0.25 and 0.7 s. Write access is in the range of 0–0.6 s.

The decoding operations are given in plots (c) and (d) and can take up to 10.26 s. More than 90% of the decoding time is taken by the write access of the SD-card. This is due to the recursive decoding scheme, which traverses selected blocks of two subband lines. As explained in [34], writing single and not consecutive blocks is relatively slow in contrast to a block-wise read operation. Note that the long writing times are related to the specific SD-card access library and that the decoding is not necessarily performed on the sensor node. The decoding calculation time is in the range of 0.22 and 0.77 s. As only the compressed stream has to be read from the SD-card, the read times are shorter than 0.12 s and only significant for the high PSNR qualities. Time results for the Lena, Camera, and Goldhill image will be with respect to their compressibility within the required time ranges of the Bird and Bridge images.

5. Conclusion

In this paper we have proposed and evaluated a wavelet-based algorithm that allows for effective image compression on very limited platforms. The algorithm builds on the principle of wavelet backwards coding, which encodes the picture starting from the lowest wavelet level (and not from the top-level), reverses the compressed bit stream, and decodes starting from the top-level. The proposed algorithm reorders the bit stream such that the coding can be conducted line-wisely. This is achieved by recursively scanning the wavelet tree such that the required quantization levels of the descendant child nodes of two lines are computed in advance and made retrievable via a specific buffer. The same buffer is used to compute quantization levels of two consecutive wavelet subband lines. Thereby, the buffer memory requirements of the algorithm are limited to the size of two wavelet subband lines ($2 \cdot N$ bytes using 16 bit coefficients, with N denoting the picture dimension) and a quantization level buffer of half the picture line

dimension (0.5N bytes). As the algorithm is in line with the typical block-access of flash memory, pictures can be read and written to SD- or MMC-cards.

From the evaluation we conclude that the coding algorithm achieves compression performance that outperforms the JPEG technique and is competitive to JPEG 2000 and Google WebP – in that it (a) gives for very high compression rates similar or even better quality and (b) for the specific application of lower resolution and natural images gives very similar performance than the state-of-the-art. We also verify the applicability of Wi2l on a 16-bit microcontroller and find that the computational time required by the algorithm is smaller than 300 ms for a 256×256 grayscale picture when a PSNR of 30 dB is sufficient. In the past, more specific platforms like digital signal processors have been regarded as mandatory for wavelet based compression on tiny devices. Moreover, the available implementations of JPEG 2000 and Google WebP are designed for PCs. The Wi2l algorithm, in contrast to these implementations, can run on a low-cost microcontroller, allowing typical sensor nodes to use modern compression via a software update. The underlying C source code is made publicly available in [39].

Future work may concern the improvement of the granularity of Wi2l, a modification of the algorithm to include the computation of the transform, or a scalability feature similar as it is provided by SPIHT.

References

[1] A. Said, W. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circuits Syst. Video Technol. 6 (1996) 243–250.

[2] J. Bankoski, P. Wilkins, Y. Xu, Technical overview of VP8, an open source video codec for the web, in: Proceedings of the IEEE International Conference on Multimedia and Expo, July 2011, pp. 1–6.

[3] D. Taubman, M. Marcellin, JPEG2000 – Image Compression, Fundamentals, Standard and Practice, Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[4] F. Dufaux, G. Sullivan, T. Ebrahimi, The JPEG XR image coding standard, IEEE Signal Process. Mag. 26 (2009) 195–199.

[5] J.-S.L.F. De Simone, L. Goldmann, T. Ebrahimi, Performance analysis of VP8 image and video compression based on subjective evaluations, in: Proceedings of the SPIE International Symposium on Optical Engineering, August 2011.

[6] T. Wiegand, G. Sullivan, G. Bjøntegaard, A. Luthra, Overview of the H.264/AVC video coding standard, IEEE Trans. Circuits Syst. Video Technol. 13 (2003).

[7] A. Al, B. Rao, S. Kudva, S. Babu, D. Sumam, A. Rao, Quality and complexity comparison of H.264 intra mode with JPEG2000 and JPEG, in: Proceedings of the International Conference on Image Processing (ICIP), 2004, vol. 1, pp. 525–528.

[8] M. Ouareta, F. Dufauxa, T. Ebrahimia, On comparing JPEG2000 and intraframe AVC, in: Proceedings of the SPIE International Symposium on Optical Engineering + Applications, August 2006.

[9] X. Peng, J. Xu, F. Wu, Line-based image coding using adaptive prediction filters, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'10), Paris, pp. 4221–4224.

[10] C. Chrysafis, A. Ortega, Line-based reduced memory wavelet image compression, IEEE Trans. Image Process. 9 (2000) 378–389.

[11] J. Oliver, M. Malumbres, On the design of fast wavelet transform algorithms with low memory requirements, IEEE Trans. Circuits Syst. Video Technol. 18 (2008) 237–248.

[12] S. Rein, M. Reisslein, Performance evaluation of the fractional wavelet filter: a low-memory image wavelet transform for multimedia sensor networks, Ad Hoc Netw. 9 (2011) 482–496.

[13] D. Salomon, G. Motta, D. Bryant, Handbook of Data Compression, 5th ed. Springer, London, 2009.

[14] J. Shapiro, Embedded image coding using zerotrees of wavelet coefficients, IEEE Trans. Signal Process. 41 (1993).

[15] I. Witten, R. Neal, J. Cleary, Arithmetic coding for data compression, Commun. ACM 6 (1987) 519–540.

[16] C.-Y. Su, B.-F. Wu, A low memory zerotree coding for arbitrarily shaped objects, IEEE Trans. Image Process. 12 (2003) 271–282.

[17] Y. Sun, H. Zhang, G. Hu, Real-time implementation of a new low-memory SPIHT image coding algorithm using DSP chip, IEEE Trans. Image Process. 11 (2002) 1112–1116.

[18] H. Arora, P. Singh, E. Khan, F. Ghani, Memory efficient set partitioning in hierarchical tree (MESH) for wavelet image compression, in: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 2, March 2005, pp. 385–388.

[19] M. Sakalli, W. Pearlman, M. Farshchian, SPIHT algorithms using depth first search algorithm with minimum memory usage, in: Proceedings of the 40th Annual Conference on Information Sciences and Systems, March 2006, pp. 1158–1163.

[20] J. Lian, K. Wang, J. Yang, Listless zerotree image compression algorithm, in: Proceedings of the 8th International Conference on Signal Processing, vol. 2, November 2006.

[21] Y. Singh, ISPIHT – improved SPIHT: a simplified and efficient subband coding scheme, in: Proceedings of the International Conference on Computing: Theory and Applications (ICCTA), March 2007, pp. 468–474.

[22] H. Pan, W. Siu, N. Law, Efficient and low-complexity image coding with the lifting scheme and modified SPIHT, in: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN), July 2008, pp. 1959–1963.

[23] M. Lanuzza, S. Perri, P. Corsonello, G. Cocorullo, An efficient wavelet image encoder for FPGA-based designs, in: Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation, November 2005, pp. 652–656.

[24] A. Kumarayapa, X.-F. Zhang, Y. Zhang, Simplifying SPIHT for more memory efficient onboard machine-vision codec and the parallel processing architecture, in: Proceedings of the International Conference on Machine Learning and Cybernetics, vol. 3, August 2007, pp. 1482–1486.

[25] L. Chew, L.-M. Ang, K. Seng, New virtual SPIHT tree structures for very low memory strip-based image compression, IEEE Signal Process. Lett. 15 (2008) 389–392.

[26] Z. Zhi-hui, Z. Jun, Unsymmetrical SPIHT codec and 1D SPIHT codec, in: Proceedings of the International Conference on Electrical and Control Engineering (ICECE), December 2010, pp. 2498–2501.

[27] N. Zhang, L. xu Jin, H. jiang Tao, K. Zhang, Image compression algorithm of high-speed SPIHT for aerial applications, in: Proceedings of the IEEE 3rd International Conference on Communication Software and Networks (ICCSN), May 2011, pp. 536–540.

[28] M. Nasri, A. Helali, H. Sghaier, H. Maaref, Energy-efficient wavelet image compression in wireless sensor networks, in: Proceedings of the International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), October 2010, pp. 1–7.

[29] J. Guo, B.N.S. Mitra, T. Karp, Backward coding of wavelet trees with fine-grained bitrate control, J. Comput. 1 (2006) 1–7.

[30] L. Ye, J. Guo, B. Nutter, S. Mitra, Memory-efficient image codec using line-based backward coding of wavelet trees, in: Proceedings of the IEEE Data Compression Conference (DCC), March 2007.

[31] L. Ye, J. Guo, B.S. Nutter, S.D. Mitra, Low-memory-usage image coding with line-based wavelet transform, SPIE J. Opt. Eng. 2 (2011).

[32] S. Rein, S. Lehmann, C. Gühmann, Wavelet image two-line coder for wireless sensor node with extremely little RAM, in: Proceedings of the IEEE Data Compression Conference (DCC'09), Snowbird, UT, pp. 252–261.

[33] S. Rein, S. Lehmann, C. Gühmann, Fractional wavelet filter for camera sensor node with external flash and extremely little RAM, in: Proceedings of the ACM Mobile Multimedia Communications Conference (Mobimedia), July 2008.

[34] S. Lehmann, S. Rein, C. Gühmann, External flash filesystem for sensor nodes with sparse resources, in: Proceedings of the ACM Mobile Multimedia Communications Conference (Mobimedia), July 2008.

[35] S. Rein, M. Reisslein, Low-memory wavelet transforms for wireless sensor networks: a tutorial, IEEE Commun. Surv. Tutor. 13 (2011) 291–307.

[36] K. Rao, P. Yip (Eds.), The Transform and Data Compression Handbook, CRC Press, Boca Raton, Florida, USA, 2001.

[37] J. Guo, S. Mitra, B. Nutter, T. Karp, A fast and low complexity image codec based on backward coding of wavelet trees, in: Proceedings of the IEEE Data Compression Conference (DCC), March 2006.

[38] S. Rein, C. Gühmann, F. Fitzek, Sensor networks for distributive computing, in: F. Fitzek, F. Reichert (Eds.), Mobile Phone Programming, 2007, pp. 397–409.

[39] S. Rein, Wavelet Image Two-Line Coder: C Source Code, 2015, Available online: ⟨http://www.mdt.tu-berlin.de⟩.