Ad Hoc Networks 9 (2011) 482–496

Contents lists available at ScienceDirect

Ad Hoc Networks

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

Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks

Stephan Rein [a], Martin Reisslein [b,*]

[a] Telecommunication Networks Group, Technical University Berlin, Berlin
[b] School of Electrical, Computer, and Energy Eng., Goldwater Center, MC 5706, Arizona State University, Tempe, AZ 85287-5706, United States

ARTICLE INFO

Article history:
Available online 13 August 2010

Keywords:
Image sensor
Image transform
In-network processing
Integer (fixed-point) arithmetic
Lifting computations
Wavelet transform
Wireless sensor network

ABSTRACT

Existing image wavelet transform techniques exceed the computational and memory resources of low-complexity wireless sensor nodes. In order to enable multimedia wireless sensors to use image wavelet transforms techniques to pre-process collected image sensor data, we introduce the *fractional wavelet filter*. The fractional wavelet filter computes the wavelet transform of a 256×256 grayscale image using only 16-bit fixed-point arithmetic on a micro-controller with less than 1.5 kbyte of RAM. We comprehensively evaluate the resource requirements (RAM, computational complexity, computing time) as well as image quality of the fractional wavelet filter. We find that the fractional wavelet transform computed with fixed-point arithmetic gives typically negligible degradations in image quality. We also find that combining the fractional wavelet filter with a customized wavelet-based image coding system achieves image compression competitive to the JPEG2000 standard.

1. Introduction

The processing unit of low-complexity sensor nodes is typically a 16-bit micro-controller, which has limited processing power and random access memory (RAM) in the range of 10 kbyte [1]. A multimedia sensor network can be formed by equipping nodes with a small camera [2,3]. Imaging-oriented applications, however, often exceed the resources of a typical low-cost sensor node. While it is possible to store an image on a low-complexity node using cheap, fast, and large flash memory [4–6], the image transmission over the network can consume too much bandwidth and energy [7]. Therefore, image processing techniques are required either (*i*) to compress the image data, e.g., with wavelet-based techniques that exploit the similarities in transformed versions of the image, or (*ii*) to extract the interesting features from the images and transmit only image meta data.

A modern pre-processing technique to inspect or compress an image is the discrete wavelet transform, which decorrelates the data. The decorrelation allows to extract the interesting features or to apply tree coding algorithms to summarize the typical patterns, e.g., the embedded zerotree wavelet (EZW) [8] or the set partitioning in hierarchical trees (SPIHT) scheme [9]. However, the memory requirements of existing image wavelet transform techniques exceed the limited resources on low-complexity micro-controllers.

In order to overcome this limitation we introduce the *fractional wavelet filter*, a novel computational method for the image wavelet transform. The fractional wavelet filter requires only random access memory (RAM) for three image lines, as analyzed in detail in Section 4.1. In particular, less than 1.5 kbyte of RAM are required to transform an image with 256×256 8-bit pixels using only 16-bit integer calculations. Thus, the fractional wavelet filter works well within the limitations of typical low-cost sensor nodes [1]. The fractional wavelet filter achieves this

* Corresponding author. Tel.: +1 480 965 8593; fax: +1 480 965 8325.
E-mail addresses: rein@tkn.tu-berlin.de (S. Rein), reisslein@asu.edu (M. Reisslein).
URL: http://mre.faculty.asu.edu (M. Reisslein).

1570-8705/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2010.08.004

performance through a novel step-wise computation of the vertical wavelet coefficients.

We conduct a comprehensive performance evaluation of the fractional wavelet filter. We first analyze the required random access memory and the computational effort for the fractional wavelet filter. We demonstrate how the fractional wavelet filter trades slightly increased computational effort for vastly reduced memory requirements compared to the conventional convolution approach. We further verify the functionality of the fractional wavelet filter and measure its timing requirements on a micro-controller with 2 kbyte of RAM, which is extended with a standard multimedia (flash memory, SD) card and a small digital camera.

We evaluate the image quality of the fractional wavelet filter, which is not lossless when computed with fixed-point arithmetic. Comparing images that are forward wavelet transformed using the fixed-point fractional wavelet filter and subsequently backward transformed with the original image, we find that the introduced image degradation is typically negligible for appropriate integer number formats. Employing the fractional wavelet filter in conjunction with a low-memory wavelet-based image coding system, we demonstrate equivalent compression performance compared to state-of-the-art image compression methods for high compression ratios.

This article is organized as follows. Section 2 discusses related work on low-memory wavelet transforms. Section 3 introduces the novel fractional wavelet filter. We evaluate the resource requirements (memory, computations, and time) of the fractional wavelet filter in Section 4. We evaluate the image quality of the fractional wavelet filter in Section 5. Section 6 concludes the article.

2. Related work

Implementations of the discrete two-dimensional image wavelet transform on a personal computer (PC) generally keep the entire source and/or destination image in memory and separately apply horizontal and vertical filters. As this approach is typically not possible on resource-limited platforms, the recent literature has explored the following memory-efficient implementations of the image wavelet transform.

A large part of the literature focuses on the implementation of the wavelet transform on field programmable gate arrays (FPGA), see for instance [10–13]. Similarly, complexity reductions aimed at efficient VLSI hardware implementations have been examined, e.g., [14–17]. The FPGA-platforms and VLSI hardware implementations are generally designed for one special purpose and are not an appropriate candidate for a multimedia sensor node that has to perform many tasks related to analysis of the surrounding area as well as communication [1,18]. Our work is different from the literature on FPGAs in that we consider the 9/7 image wavelet transform for a generic micro-controller with a very small RAM.

The general approach to build a multimedia sensor node has been to connect a second platform with a more capable processor to the sensor node, see e.g., [19]. This work is different from this traditional approach in that we connect a small camera directly to the micro-controller of the sensor node and perform the wavelet transform with the micro-controller.

The distributed nature of sensor networks as well as the correlation in the sensed data in neighboring sensors is exploited in a number of data and image compression approaches that combine the computations across several sensor nodes. For instance, partial wavelet coefficients are calculated in [20,21] and refined as the data travels toward the sink node. The approach in [22] employs lapped biorthogonal transform and requires sensor nodes to hold at least eight or 12 image rows in memory depending on their role in the sensor network. The distributed image compression approach in [23] employs the wavelet transform and strives to minimize energy consumption by distributing the compression computations across the nodes between a camera node and the sink node. In contrast, we compute the wavelet transform at an individual node.

As we show in this work, the low-complexity fractional wavelet transform can be combined with an image coding system, e.g., [24], to give compression performance competitive to the current JPEG2000 image compression system. To our best knowledge, only the approach in [25] compresses images directly on low-complexity hardware. However, [25] employs the general JPEG technique which gives significantly lower image compression performance than wavelet-based compression.

Closer related to our work is the line-based version of the image wavelet transform proposed in [26]. The line-based approach [26] employs a system of buffers where only a small subset of the coefficients is stored; thus, tremendously reducing the memory requirements compared to the traditional transform approach.

An efficient implementation of the line-based transform using the lifting scheme and improved communication between the buffers is detailed for floating-point arithmetic in [27] and implemented in C++ on a PC for demonstration. (Fixed-point arithmetic is not considered in [27].) The line-based approach [26,27], however, cannot run on a sensor node with very small RAM, as it theoretically requires memory for 12 image lines when lifting is employed (18 images lines without lifting) and, thus, uses approximately 26 kbyte of RAM for a six-level transform of a 512×512 image. In contrast, for a 512×512 image our fractional wavelet filter approach would only require roughly 4.6 kbyte of RAM using floating-point arithmetic (with fixed-point arithmetic only approximately 2.56 kbyte are required). To the best of our best knowledge, a 9/7 image wavelet transform for typical low-complexity sensor node hardware is for the first time presented and evaluated in this article.

A preliminary version of the fractional wavelet filter approach was presented in the conference paper [28]. This article extends [28] in two important ways. First, this article extends the fractional wavelet filter approach by incorporating the *lifting scheme*. The lifting scheme can compute the wavelet transform in place; thus, saving memory while significantly reducing the computational cost compared to the classical convolution approach. Second, the conference paper [28] presented only a preliminary performance evaluation of the fractional wavelet filter, whereas we present a comprehensive analysis of resource requirements as well as image quality evaluations for a range of image dimensions and accuracy levels in the integer arithmetic in this article. Moreover, the achievable compression when combining the fractional wavelet filter with an image coding system and the comparison with JPEG2000 is for the first time presented in this article.

Low-memory wavelet-based image coding (compression) that builds on wavelet transforms of images or integrates wavelet transform and compression is a research area related to the present study which focuses exclusively on low-memory image wavelet transforms. For completeness we briefly review this related research area, noting that to the best of our knowledge all techniques in this area have significantly higher memory requirements for the image wavelet transform than our approach. Building on [26], a strip based approached for low-memory image compression in proposed in [29]. In turn, building on [29], an approach for reducing the number of wavelet scales (transform levels) and an new transform tree structure to reduce the memory requirements for compression are proposed in [30,31]. Block-based wavelet transforms, which require similar amounts of memory as [26], were developed in [32,33] in support of block-based image coders. An enhanced cache memory management mechanism for the lifting scheme in the context of the block-based approach from [32] is developed in [34]. Low-memory on the fly computations of generic hierarchical transforms that do not need any re-loading of image samples and a related image coder are studied in [35]. The approach in [35] requires memory on the order of several tens or hundreds of image lines. A low-memory approach for set partitioning in hierarchical trees (SPHIT) image coding, which requires close to 100 kbyte for a 512 × 512 image is proposed in [36], while a low-memory image entropy coder employing zerotree coding and Golomb–Rice codes is developed in [37].

3. Fractional wavelet filter

In this section we introduce the fractional wavelet filter. We first give briefly preliminaries on wavelet transforms and introduce our main notations. We refer to [38,39] for more detailed background on the wavelet transform. We then introduce the floating-point version of the fractional wavelet filter, followed by the version for fixed-point numbers. Finally, we explain the lifting scheme for in-place computation of the fractional wavelet filter.

3.1. Preliminaries on wavelet transform

A one-dimensional discrete wavelet transform can be performed by low- and highpass filtering of an input signal vector $\mathbf{s}$ with dimension N, i.e., $\mathbf{s} = [s_0, s_1, \ldots, s_{N-1}]$, which can be a single line of an image. The two resulting signals are then sampled down, that is, each second value is discarded, and form the *approximations* and *details*, which are two sets of $N/2$ wavelet coefficients. We employ the Daubechies biorthogonal 9/7 filter, which is part of the JPEG2000 standard and is given by the filter coefficients

Table 1
9/7 Analysis wavelet filter coefficients in floating-point and Q15 fixed-point format.

Index j	Lowpass l_j		Highpass h_j	
	Floating-point	Q15	Floating-point	Q15
0	0.852699	27,941	0.788486	25,837
±1	0.377403	12,367	−0.418092	−13,700
±2	−0.110624	−3625	−0.040689	−1333
±3	−0.023849	−781	0.064539	2115
±4	0.037828	1240	0	0

in Table 1. More formally, the approximation vector $\mathbf{a}$ (of dimension $N/2$) is obtained by convolving the signal vector $\mathbf{s}$ (of dimension N, with symmetrical extension at the boundaries) with the vector of low pass filter coefficients $\mathbf{l} = [l_{-4}, l_{-3}, \ldots, l_4]$, i.e.,

$$\mathbf{a} = \mathrm{conv}(\mathbf{s}, \mathbf{l}). \tag{1}$$

In particular, the individual approximations are obtained as

$$a_i = \mathrm{convp}(\mathbf{s}, \mathbf{l}, 2i) = \sum_{j=-4}^{4} s_{2i+j} \cdot l_j, \quad i = 0, 1, \ldots, \frac{N}{2} - 1. \tag{2}$$

Analogously, the detail vector $\mathbf{d}$ is obtained through the convolution operation

$$\mathbf{d} = \mathrm{conv}(\mathbf{s}, \mathbf{h}), \tag{3}$$

and the individual details are given by

$$d_i = \mathrm{convp}(\mathbf{s}, \mathbf{h}, 2i+1) = \sum_{j=-3}^{3} s_{2i+1+j} \cdot h_j,$$
$$i = 0, 1, \ldots, \frac{N}{2} - 1. \tag{4}$$

A one-level image wavelet transform can be computed by first performing a one-dimensional transform for each line and then repeating this operation for all the columns of the result, that is, applying the low- and the highpass in the vertical direction. The horizontal transform results in two matrices with N rows and $N/2$ columns, which we denote by L and H for the approximations and details, respectively. The second operation leads to four square matrices of the dimension $N/2$ denoted by LL, HL, LH, and HH, which are the so-called *subbands*. The two operations can be repeated on the LL subband to obtain higher level subbands, e.g., HL_2 refers to the second level HL subband that is computed by first horizontally filtering LL_1 and then vertically filtering the result.

3.2. Floating-point fractional wavelet filter

The fractional wavelet filter computes the image wavelet transform on a sensor node with very small RAM memory and an attached flash memory (MMC card). The data on the MMC card can only be accessed in blocks of 512 bytes, thus a sample-by-sample access as easily executed with RAM memory on personal computers is not feasible. The fractional filter takes this restriction into account by reading the image samples line-by-line from the MMC card.

Table 2
Pseudo code of fractional wavelet filter for the first level forward wavelet transform.

```
1.  For $i = N/2-1, N/2-2, \ldots, 0$:
2.      For $m = 0, 1, \ldots, N-1$: $LL\_HL_m = 0$, $LH\_HH_m = 0$
3.    For $j = -4, -3, \ldots, 4$:
4.      Read line $\ell = 2i + j$ from flash memory into $\mathbf{s}$
5.      For $k = 0, 1, \ldots, N/2-1$:
6.          $L = \mathrm{convp}(\mathbf{s}, \mathbf{l}, 2k)$
7.          $LL\_HL_k \mathrel{+}= l_j \cdot L$    // update LL
8.          $LH\_HH_k \mathrel{+}= h_{j-1} \cdot L$    // update LH
9.          $H = \mathrm{convp}(\mathbf{s}, \mathbf{h}, 2k+1)$
10.         $LL\_HL_{k+N/2} \mathrel{+}= l_j \cdot H$    // update HL
11.         $LH\_HH_{k+N/2} \mathrel{+}= h_{j-1} \cdot H$    // update HH
```

For the first transform level, the algorithm reads the image samples line-by-line while it writes the subbands to a different destination on the MMC card. The *LL* subband contains the input data for the next transform level. Note that the input samples for the first level are of the type unsigned char (8 Bit), whereas the input for the higher level are either of type float (floating-point filter) or INT16 (fixed-point filter) format. The filter does not work in place on the MMC card and for each transform level a new destination matrix is allocated on the MMC card. Since the MMC card has abundant memory this approach does not affect the sensor's resources. This approach also allows reconstruction of the image from any transform level, which is useful for adaptive transmission mechanisms similar to [40].

The floating-point wavelet filter computes the wavelet transform with high precision, as it uses 32 bit floating-point variables for the wavelet and filter coefficients as well as for the intermediate operations. Thus, the images can be reconstructed essentially without loss of information. The fractional wavelet filter uses three buffers of the dimension N, namely $\mathbf{s}$ for the current input line, **LL_HL** for the *LL*/*HL* subband destination line, and **LH_HH** for the *LH*/*HH* subband destination line.

The filter computes the horizontal wavelet transform coefficients on the fly, while it computes a set of *fractions* (whereby we refer to part of a sum as a fraction) toward the vertical wavelet transform. More specifically, the input area which is centered (vertically) at index i moves up by two lines to achieve vertical down sampling, see Lines 1 and 4 in Table 2. For each position of the filter area, nine lines of input (indexed by j) are read, see Lines 3 and 4. The horizontal filtering is achieved on the fly through the loop over index k in Lines 5–11. Importantly, in this loop over k, the vertical filter index j stays constant as all subband rows are updated by the fractions contributed by the current horizontal filtering. Repeating this update procedure for the nine input lines (and the corresponding vertical filter coefficients) gives the final transform results in the subband rows **LL_HL** and **LH_HH** that are then written to the flash memory.

3.3. Fixed-point fractional wavelet filter

Many low-cost micro-controllers do not provide hardware support for floating-point operations. If floating-point operations are coded, the compiler translates them to integer operations, which often results in very long computing times, as illustrated in Section 4.3. Converting an algorithm from floating- to fixed-point arithmetic typically results in substantial time and energy savings at the expense of lower precision and thorough number range analysis. Thus, using fixed-point arithmetic for the fractional wavelet filter can help to significantly reduce the computational requirements and to reduce the RAM memory needed for the destination subbands. We refer to [39] for tutorial background on using fixed-point arithmetic for wavelet transforms.

We employ the standard *Qm.n* notation, where m indicates the number of bits allocated to the integer portion and n the number of bits for the fractional portion of a two's complement number. For the fixed-point fractional wavelet filter we transform the real wavelet coefficients to the Q0.15 format, see Table 1. For all subsequent computations, the possible range of the results should be smaller than the range of numbers that can be represented with the employed *Qm.n* format. As the number range of the wavelet transform (result) coefficients increases from level to level, the output data format has to be enlarged, i.e., m has to be incremented while n is decremented. We take the results of the study [10], which found that Q10.5 is appropriate for a one-level transform of an image with 8 bit integer samples, as a starting point for our own investigations in Section 5.2. For the first wavelet transform level, we use the input data format Q15.0 since the image samples are integer numbers.

3.4. Lifting scheme

For the lifting scheme the nine lowpass analysis filter coefficients and the corresponding seven lowpass synthesis coefficients [41] are factored into the parameters [42]

$$
\begin{aligned}
\alpha &= -1.5861343420693648,\\
\beta &= -0.0529801185718856,\\
\gamma &= 0.8829110755411875,\\
\delta &= 0.4435068520511142,\\
\zeta &= 1.1496043988602418.
\end{aligned}
\tag{5}
$$

Table 3 details the computation scheme for the 9/7 lifting scheme [43] using the operator notation where for instance $x \mathrel{+}= y$ assigns x the value $x + y$. In summary, the original signal is first separated into even and odd indexed values. Then follow four update/predict steps where the parameters α and γ are convoluted with the odd part and parameters β and δ are convolved with the even part of the signal (without signal extension at the boundaries).

Table 3
Lifting scheme for the forward wavelet transform.

```
$\mathbf{s}_e = [s_0, s_2, s_4, \ldots, s_{N-2}]$
$\mathbf{s}_o = [s_1, s_3, s_5, \ldots, s_{N-1}]$
$\mathbf{s}_o \mathrel{+}= \mathrm{conv}([\mathbf{s}_e, s_{N-2}], [\alpha, \alpha])$
$\mathbf{s}_e \mathrel{+}= \mathrm{conv}([s_1, \mathbf{s}_o], [\beta, \beta])$
$\mathbf{s}_o \mathrel{+}= \mathrm{conv}([\mathbf{s}_e, s_{N-2}], [\gamma, \gamma])$
$\mathbf{s}_e \mathrel{+}= \mathrm{conv}([s_1, \mathbf{s}_o], [\delta, \delta])$
$\mathbf{s}_e \cdot= \zeta$
$\mathbf{s}_o /= \zeta$
```

4. Resource requirements

In this section we evaluate the resource requirements of the fractional wavelet filter. We analyze the required random access memory (RAM), measure the computation times on an experimental sensor node, and analyze the reduction in computational complexity achieved with the lifting scheme.

4.1. Random access memory

We evaluate the memory requirements of the fractional wavelet filter through analysis of the buffer structure described in Section 3. For the first transform level $lev = 1$, the input buffer **s** holds N original 1 byte image samples. Furthermore, each of the buffers **LL_HL** and **LH_HH** holds N wavelet transform coefficients. These transform coefficients are float values of 4 bytes each for the floating-point filter and INT16 values of 2 bytes each for the fixed-point filter. Thus, the first transform level requires a total of $9N$ bytes for the floating-point filter and $5N$ bytes for the fixed-point filter.

For the higher transform levels $lev > 1$, the input buffer **s** has to hold the wavelet transform coefficient from the preceding LL subband. Note that the first-level transform of an image with $N \times N$ samples results in an LL subband with $N/2 \times N/2$ transform coefficients, i.e., the input buffer of the second-level transform has to hold $N/2$ transform coefficients. With each additional transform level, the number of input values for the subsequent transform level are halved. Further note that the coefficients are 4-byte float values for the floating-point filter and 2-byte INT16 values for the fixed-point filter. Thus, for a given transform level lev, $lev > 1$, the floating-point filter requires $12N/lev$ bytes and the fixed-point filter requires $6N/lev$ bytes of memory.

Overall, the maximum memory requirement for a multi-level transform of an $N \times N$ image is attained for the first transform level and is $9N$ bytes for the floating-point filter, and $5N$ bytes for the fixed-point filter. For instance, for 128×128 pixel images, the floating-point filter requires 1152 bytes, while the fixed-point filter requires 640 bytes. For a 256×256 image these memory requirements increase to 2304 bytes for the floating-point filter and 1280 bytes for the fixed-point filter.

The inclusion of the lifting scheme for in-place computation of the horizontal transform does not further reduce the random access memory requirements because the fractional wavelet filter still requires one input line buffer **s** and two destination buffers. The lifting scheme performs an in-place computation (in the buffer **s**) of the horizontal transform coefficients of an entire line with one execution of the lifting algorithm. This in-place computation requires that the buffer **s** can hold the transform result coefficients, which are in the INT16 format for fixed-point filter. Applying the lifting scheme for the first transform level would thus require a total of $6N$ bytes of memory for the fixed-point filter. Therefore, for the first transform level we do not use the lifting scheme, and use an input buffer holding N 1-byte image samples (for a total memory requirement of $5N$ bytes). For the second transform level, the dimension is $N/2$ and we employ the lifting scheme and utilize the input buffer **s** to hold $N/2$ INT16 input values (and then the corresponding result values). The lifting scheme, however, reduces the computational complexity, as evaluated in Section 4.3.

We note for comparison that the classical convolution approach for the wavelet transform requires memory for $2N^2$ pixels (with four bytes per pixel for floating-point and two bytes per pixel for fixed-point computation); with the lifting scheme for in-place computation this memory requirement is reduced to N^2 pixels. The existing line-based low-memory wavelet transform approaches [26,27] require memory for $18N$ pixels without the lifting scheme, and $12N$ pixels with the lifting scheme.

4.2. Computational requirements

In this section we give the computational requirements for (1) the wavelet transform using convolution, (2) the fractional wavelet filter, and (3) the lifting scheme. We recall from Section 3.1 that conv(**s**,**l**) and conv(**s**,**h**) denote the operations for computing the approximations and the details of the one-dimensional wavelet transform of a given image line **s** with the dimension N, respectively. We denote a, m, and d for the computational effort for an addition, multiplication, and division operation, with two operands in floating-point or fixed-point format (whereby the required concomitant bit-shift operations for fixed-point arithmetic are included). We denote Γ_l and Γ_h for the computational effort (which we will express in terms of the computational effort for elementary addition, multiplication, and division operations) of the conv(**s**,**l**) and conv(**s**,**h**) operations. We further recall that convp(**s**,**l**,·) and convp(**s**,**h**,·) denote the operations for computing a single approximation coefficient and a single detail coefficient and denote $\Gamma_{p,l}$ and $\Gamma_{p,h}$ for the computational effort of these operations.

For the computation of an approximation coefficient the convp(**s**,**l**,·) operation requires the scalar product of two vectors with the dimension nine, i.e., $l_{-4}s_1 + l_{-3}s_2 + \cdots + l_4 s_9$. Noting that the filter is symmetrical, the computation can be performed as $l_{-4}(s_1 + s_9) + l_{-3}(s_2 + s_8) + \cdots + l_0 s_5$. Hence, the convp(**s**,**l**,·) operation requires a computational effort of $\Gamma_{p,l} = 8a + 5m$. Similarly, the convp(**s**,**h**,·) operation which involves a symmetrical filter with seven coefficients requires a computational effort of $\Gamma_{p,h} = 6a + 4m$. The computational effort for the conv(**s**,**l**) operation is thus

$$\Gamma_l = \frac{N}{2}(8a + 5m) \tag{6}$$

while the effort for the conv(**s**,**h**) operation is

$$\Gamma_h = \frac{N}{2}(6a + 4m). \tag{7}$$

Note that throughout this section we consider the computational effort for a one-level wavelet transform of an image with dimension N. For each higher wavelet level the image dimension is halved.

4.2.1. Image wavelet transform using convolution

Recall that the image wavelet transform consists of the four subbands *LL*, *HL*, *LH*, and *HH*. In the standard convolutional approach, these subbands are computed by first applying a one-dimensional transform on each image line resulting in the *L*/*H* image, and then the *L* and the *H* subbands are vertically filtered to obtain the four subbands.

The computational effort is thus given as

$$\Gamma_{\text{img, conv}}(N) = \underbrace{N\cdot(\Gamma_l+\Gamma_h)}_{\text{compute } L/H} + \frac{N}{2}\left(\underbrace{\Gamma_l}_{LL}+\underbrace{\Gamma_h}_{LH}+\underbrace{\Gamma_l}_{HL}+\underbrace{\Gamma_h}_{HH}\right), \quad (8)$$

$$= 2N(\Gamma_l+\Gamma_h), \quad (9)$$

$$= N^2(14a+9m). \quad (10)$$

4.2.2. Fractional wavelet filter

We analyze the pseudo code of Table 2, to obtain the computational effort required for the fractional wavelet filter as

$$\Gamma_{\text{img, fracwave}}(N) = \underbrace{\frac{N}{2}9\frac{N}{2}}_{\text{lines } 1,\,3,\,5} \times \left(\underbrace{\Gamma_{p,l}}_{\text{line } 6} + \underbrace{2[m+a]}_{\text{lines } 7,\,8} + \Gamma_{p,h} + 2[m+a]\right), \quad (11)$$

$$= N^2\left(\frac{81}{2}a+\frac{117}{4}m\right). \quad (12)$$

The fractional filter thus requires $\frac{81}{2}/14 \approx 2.89$ times more add operations and $\frac{117}{4}/9 \approx 3.25$ more multiplication operations than the convolutional approach.

We provide additional insight into this relationship between the computational effort for the classical convolution approach and the fractional wavelet filter by analyzing the differences between the two approaches. For the convolutional approach, the computational effort can be expressed in two products as

$$\Gamma_{\text{img, conv}}(N) = \underbrace{N\cdot(\Gamma_l+\Gamma_h)}_{\text{compute } L/H} + \underbrace{N\cdot(\Gamma_l+\Gamma_h)}_{LL,\ LH,\ HL,\ HH}, \quad (13)$$

whereby the first product refers to the horizontal wavelet transform and the second product to the vertical filter operations. There are two contributions to the increased computational effort of the fractional wavelet filter approach. First, the convolution approach exploits the symmetry of the filter coefficients, i.e., filtering with nine filter coefficients requires only an effort of $4a$ for summing the signal values that are filtered by the same filter coefficient, $5m$ for multiplying the signal values with the filter coefficients, and $4a$ for summing the resulting products, i.e., $\Gamma_{p,l} = 8a + 5m$. In contrast, the fractional filter approach carries out these computations through update operations. Therefore, the fractional wavelet filter cannot exploit the symmetry of the wavelet filter coefficients. Instead, for each set of two destination lines **LL_HL** and **LH_HH**, the fractional filter reads nine input lines (see Line 3 in Table 2) and computes the horizontal transforms (see Lines 6 and 9 in Table 2). Further, note that the fractional filter moves up the filter area by two lines for each new set of two destination lines (see Line 4 in Table 2). Thus, there is overall a 9/2-fold increase in computational effort for computing the *L*/*H* image compared to the convolutional approach.

Second, for ease of readability and simplicity of the pseudo code, the fractional filter in Table 2 implies that there are nine highpass filter coefficients, i.e., that the highpass filter includes h_{-4} and h_4. These coefficients do not exist, i.e., have zero values and do not result in an actual update of the transform coefficients. We note that the compiler may take out the multiplication with zero coefficients; however, for completeness we include these operations. Therefore, the operations conv($\mathbf{s},\mathbf{l}$) and conv($\mathbf{s},\mathbf{h}$) require a computational effort of $\frac{N}{2}(9m+9a)$ in the fractional filter approach. Overall, the computational effort given by Eq. (13) for the convolution approach translates for the fractional wavelet filter approach to a computational effort of

$$\Gamma_{\text{img, fracwave}}(N) = \frac{9}{2}\cdot N\cdot(\Gamma_l+\Gamma_h) + N\left(\frac{N}{2}[9m+9a]+\frac{N}{2}[9m+9a]\right) \quad (14)$$

$$= \frac{9}{2}\cdot N\cdot\frac{N}{2}([8a+5m]+[6a+4m]) + N\frac{N}{2}(18m+18a) \quad (15)$$

$$= \frac{81}{2}N^2a+\frac{117}{4}N^2m, \quad (16)$$

which is the same result as previously computed in Eq. (12).

4.2.3. Lifting scheme

We extract the required number of operations for the lifting scheme from Table 3. The first update operation of the lifting scheme is $\mathbf{s}_o += conv([\mathbf{s}_e, s_{N-2}], [\alpha, \alpha])$ which requires $N-1$ additions and $N/2$ multiplications for the convolution as well as $N/2$ additions for updating the vector with dimension N, i.e., a total effort of $(\frac{3N}{2}-1)a+\frac{N}{2}m$. These update operations occur four times. Finally, the two vectors $\mathbf{s}_e$ and $\mathbf{s}_o$ are multiplied by ζ and divided by ζ, respectively, requiring $N/2$ multiplications and $N/2$ divisions. Thus, the total computational effort for transforming one image line is

$$\Gamma^{\text{lift}}_{\text{line}}(N) = 4\left(\left[\frac{3N}{2}-1\right]a+\frac{N}{2}m\right)+\frac{N}{2}m+\frac{N}{2}d, \quad (17)$$

$$= (6N-4)a+\frac{5}{2}Nm+\frac{N}{2}d. \quad (18)$$

With Eq. (8) we obtain the computational effort for transforming a complete image with the convolutional approach as

$$\Gamma^{\text{lift}}_{\text{img}}(N) = 2N\left([6N-4]a+\frac{5}{2}Nm+\frac{N}{2}d\right). \quad (19)$$

Toward evaluating the computational effort for the fractional wavelet filter with lifting for the horizontal transform we note that (18) gives the effort of transforming one image line with dimension N using the lifting scheme. We further note that in (11), the terms $\Gamma_{p,l}$ and $\Gamma_{p,h}$ account for the horizontal convolutional transform. Thus, we replace these two terms with the effort for the lifting scheme resulting in

$$\Gamma^{\text{lift}}_{\text{img, fracwave}}(N) = \frac{N}{2}9\frac{N}{2}(2[m+a]+2[m+a]) + \frac{N}{2}9\Gamma^{\text{lift}}_{\text{line}}(N), \quad (20)$$

$$= \frac{9}{2}N\left([8N-4]a + \frac{9}{2}Nm + \frac{N}{2}d\right). \quad (21)$$

Thus, for moderate to large N, the lifting scheme reduces the number of additions to approximately $72/81 \approx 90\%$ and the number of multiplications to $81/117 \approx 70\%$ of the numbers required for the fractional wavelet filter without lifting while requiring some additional divisions.

4.3. Time measurements

For measuring the time requirements of the fractional wavelet filter, we built a sensor node from the *Microchip dsPIC30F4013*, which is a 16 bit digital signal (micro) controller with 2 kbyte of RAM. We attached the camera module C328-7640 with a universal asynchronous receiver/transmitter (UART) interface to the micro-controller. We further attached an external 64 Mbyte multimedia card (MMC) as flash memory (MMC card) and access the data on the MMC card through the filesystem presented in [4]. Camera and MMC card both can be connected to any controller with UART and SPI ports. The system is designed to capture still images.

For the initial set of time measurements we considered the forward fractional wavelet filter without lifting scheme. The 2 kbytes of the sensor's RAM allowed to transform 128×128 images with the floating-point filter and 256×256 images with the fixed-point filter. We compute a six-level forward wavelet transform and report means from 20 independent measurements. Table 4 gives the times needed for the transform classified by read and write MMC card access times and processor computing times. The time values for the different categories were nearly constant across the different measurements with the exception of the write access times, which experience variability due to internal operations performed by the flash memory media before a block of data can be written. The floating-point transform takes 16.2 s for the 128×128 pixel image, whereas for the four times larger 256×256 pixel image the fixed-point transform takes 11.6 s. The floating-point algorithm is very slow because the processor does not support the required high-precision arithmetic. Instead, the operations are realized through heavy compiler support. We furthermore observe from Table 4 that the computing times make up the largest portion of the total time. That is, the flash memory is not the bottleneck of the fractional filter, even though there is large overhead in the read process, as the rows are read repeatedly. In most applications, the fixed-point algorithm will be preferable as the floating-point algorithm is much slower even for a four times smaller input image while it needs the same amount of RAM.

For further insight into the time requirements of the fixed-point filter, we examine the flash memory access read and write times and computation times for the individual transform levels 1–6 in Fig. 1. We observe from Fig. 1 that the first transform level consumes by far the most time. Each subsequent transform requires less than half the time of the preceding level. This is because the subsequent transform levels operate only on the transform coefficients in the *LL* subband of the preceding level. Recall that for an $N \times N$ input image, the first-level transform gives an *LL* subband with $N/2 \times N/2$ transform coefficients. With each successive transform level, the number of coefficients in the *LL* subband is reduced to a quarter of the number of input *LL* subband coefficients.

We further observe from Fig. 1 that compute times dominate for the first and second transform levels, but shrink to negligible values compared to the read and write times for transform levels 4–6. Nevertheless, aggregating all six transform levels, the compute times dominate the total time requirement, as also observed from Table 4. In summary, these timing results corroborate the underlying strategy of the fractional wavelet filter—to reduce the memory requirements by introducing some replication in the read process while ensuring that the write process has no replication. This is an important feature as the reading access on flash memory is generally very fast, whereas

Table 4
MMC card access and processor computing times for a six-level wavelet transform of an $128 \times 128 \times 8$ image with the floating-point filter and an $256 \times 256 \times 8$ image with the fixed-point filter. The floating-point filter is very slow because the compiler translates the computations to 16 bit integer operations.

	Time (s)			
	T_{read}	T_{write}	T_{compute}	T_{total}
float128	1.421	0.5425	14.22	16.18
fix256	2.839	1.085	7.716	11.64

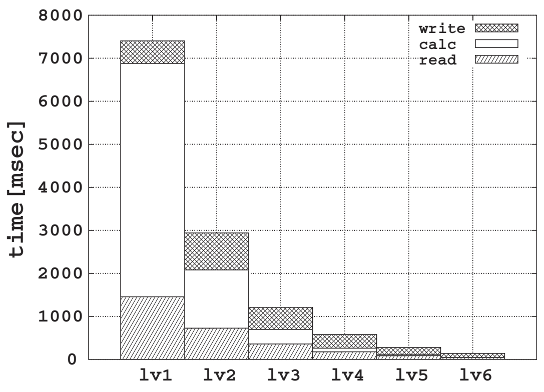


Fig. 1. Mean values of 20 time measurements for performing a fixed-point fractional wavelet filter transform of an 256×256 image for levels 1–6. The times are categorized by read and write access from the multimedia card and by computing times. The largest proportion is taken by the computing times.

writing times on flash memory fluctuate and repetitions in the write process should therefore be avoided, see the work on the filesystem [4]. Despite this "repetitive read" strategy, the total time requirement for the wavelet transform on memory constrained sensor nodes with attached flash memory is dominated by the compute times.

5. Image quality

5.1. Overview

For the quality evaluation of the fixed-point wavelet filter, we have selected 24 test images from the *Waterloo Repertoire* and the *standard image data base* (available at http://links.uwaterloo.ca and http://sipi.usc.edu/database). The images were converted to an integer data format with the range of $[-128,\ldots,127]$ using the *convert* command of the software suite *ImageMagick* (available at http://www.imagemagick.org) and the software *Octave*.

In the image quality evaluation we compare the original $N \times N$ image $f(j,k)$, $j,k = 1,\ldots,N$, with the reconstructed image $g(j,k)$, $j,k = 1,\ldots,N$. The reconstructed image is generated through a forward wavelet transform using the fractional fixed-point wavelet filter of the original image followed by an inverse wavelet transform (using the standard floating-point inverse transform). We compute the *mean squared error* (MSE) for the compared monochrome image matrices f and g:

$$\mathrm{MSE} = \frac{1}{N^2} \sum_{\forall j,k} (f(j,k) - g(j,k))^2. \tag{22}$$

The PSNR in decibels (dB) is calculated as

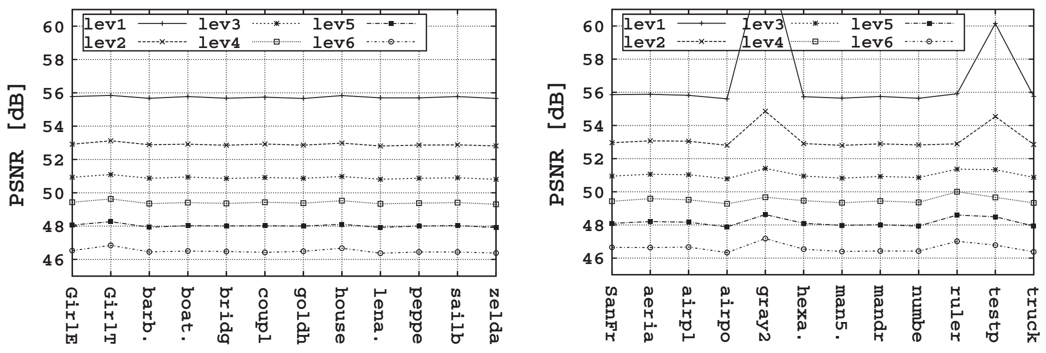
$$\mathrm{PSNR} = 10 \cdot \log_{10} \frac{255^2}{\mathrm{MSE}}. \tag{23}$$

Image degradations with a PSNR of 40 dB or higher are typically nearly invisible for human observers [44].

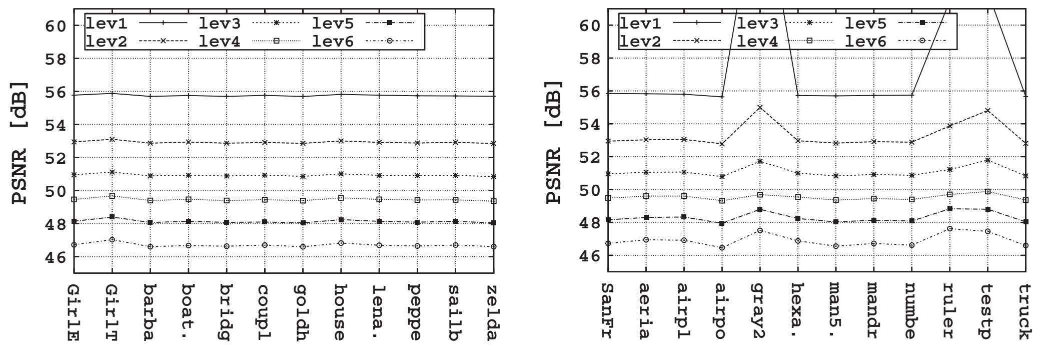
We denote the *Qm.n* format for the wavelet transform (result) coefficients of the first transform level by n_1 (as a short-hand for $Qm.n_1$). For instance, $n_1 = 5$ means that the result coefficients of the first transform level are in the *Q*10.5 format, which requires a division by 2^5 to obtain the coefficients in the *Q*15.0 format. Throughout, following [10], for each higher transform level, we switch the format to the next larger range.

5.2. Fixed-point filter without lifting scheme

We initially employ the fractional fixed-point wavelet filter without the lifting scheme and six wavelet transform levels for each image. We did not observe any visible qual-

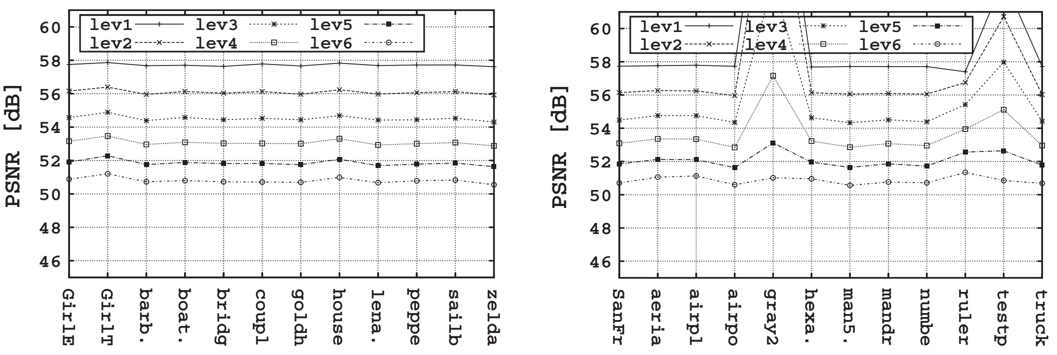


(a) $N = 256$

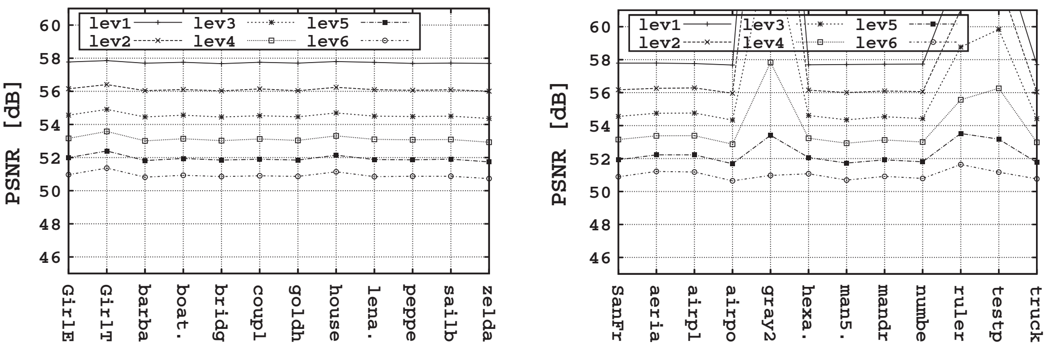


(b) $N = 512$

Fig. 2. PSNR image quality without lifting using the fractional fixed-point wavelet filter with $n_1 = 5$. The evaluation uses the fractional filter for the forward transform and a standard floating-point inverse transform. Even if there was no quality degradation visible, the transform is not lossless. The first level gives very high PSNR values as the transform input values are integers. The quality loss is typically negligible for a lossy compression algorithm that removes least significant bits.



(a) N = 256



(b) N = 512

Fig. 3. PSNR image quality without lifting using the fractional fixed-point wavelet filter with $n_1 = 6$.

ity degradation. However, the forward wavelet transform with the fractional wavelet filter is not lossless. Figs. 2 and 3 give the PSNR values for each of the maximum wavelet levels `lev1` through `lev6` for both 256 × 256 pixel images and 512 × 512 pixel images for $n_1 = 5$ and $n_1 = 6$, respectively. We observe from the comparison of Figs. 2 and 3 that the $n_1 = 6$ data format for the first transform level performs significantly better than the $n_1 = 5$ data format. This is due to the higher precision of the $n_1 = 6$ data format; however, the range of the wavelet transformed values is more limited with this format, which is not a problem for the considered test images. For the different image sizes there is only a small PSNR quality difference visible.

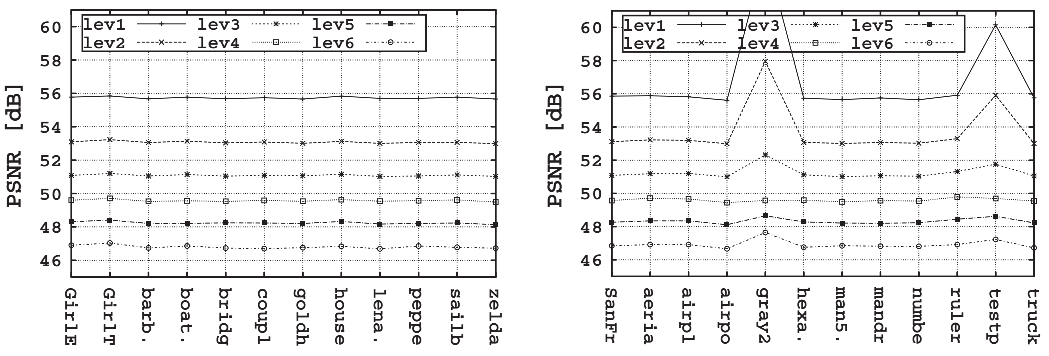
In additional evaluations for which we do not include plots due to space constraints, we have considered the data formats $n_1 = 4$ and $n_1 = 7$ for the first transform level. We found that $n_1 = 4$ consistently gives image qualities between 42 and 50 dB for transform levels one through five. For $n_1 = 7$, the image quality is very high (54–58 dB for all six transform levels) for images with relatively "soft" edges (e.g., `SanFr`, `aeria`, and `mandr`), while the image quality drops to values between 10 dB and 20 dB for images with sharp edges and "artificial" content (e.g., `numbers`, `ruler`, and `test pattern`). For the images with soft edges the wavelet transform (result) coefficients can be accommodated by the small number ranges with $n_1 = 7$, giving high image quality due to the large number of bits representing fractional values with fine resolution. For the images with sharp edges the wavelet coefficients overflow the number ranges with $n_1 = 7$, leading to large errors.

Overall, we find that for the computational methodology of the fractional wavelet filter, the $n_1 = 5$ format [10], which uses formats Q10.5 for the first transform level through Q15.0 for the sixth transform level robustly gives high image quality. The $n_1 = 6$ format achieves higher image quality for the considered test images which represent a wide range of image characteristics. However, the smaller number ranges of the $n_1 = 6$ format may lead to overflows for images with extreme characteristics and before use in specific application areas with extreme image characteristics should be tested for the application area.

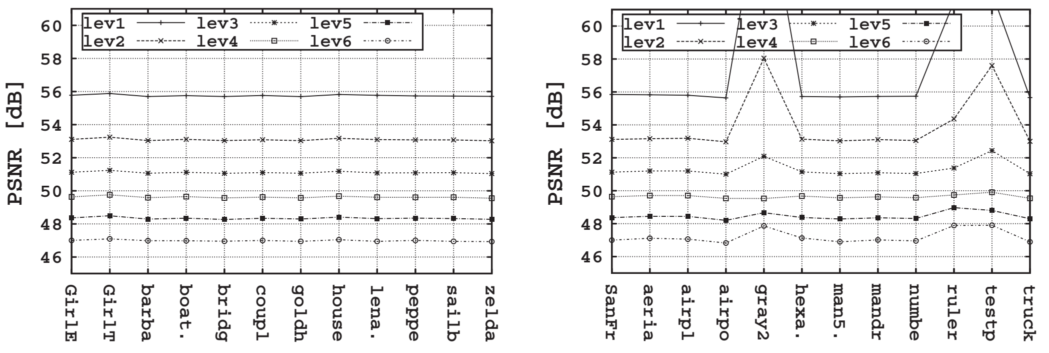
5.3. Fixed-point filter with lifting scheme

We next evaluate the image quality obtained when including the lifting scheme for in-place computation in the fixed-point filter. In Figs. 4 and 5 we plot the image qualities for transform levels one through six when the forward wavelet transform uses the fixed-point filter with the lifting scheme for $n_1 = 5$ and $n_1 = 6$, respectively. From comparisons of Figs. 2 and 3 with Figs. 4 and 5 we observe that the fixed-point filter with the lifting scheme gives generally the same or very slightly higher PSNR image qualities than the fixed-point filter without the lifting scheme.

An exception occurs for the `test pattern` image in Fig. 5a) and for the `numbers`, `ruler`, and `test pattern`



(a) $N = 256$



(b) $N = 512$

Fig. 4. PSNR image qualities for the fixed-point fractional wavelet filter with $n_1 = 5$ with incorporated lifting scheme, which is applied for the horizontal transform in levels 2–6. While the lifting scheme saves computation, the image qualities are nearly the same or slightly higher than with the fixed-point filter without the lifting scheme.

images in Fig. 5b) that have very sharp edges. For instance, in the `numbers` image, typed numbers are superimposed on the underlying `lena` image. For $N = 512$, $n_1 = 6$ for the `numbers` image the PSNR qualities are between 41.3 and 41.0 dB for transform levels 2–6; for the `test pattern` image they are between 44.6 and 43.9 dB. The sharp edges in these images give large coefficient values that result in a few overflows of the number range of the $n_1 = 6$ format during the lifting computations, as studied in detail in Appendix A. The $n_1 = 5$ format avoids these overflow problems and consistently achieves PSNR values above 46 dB for the six-level wavelet transform with lifting for these images.

Additional evaluations for $n_1 = 4$ gave very similar results as for the fractional wavelet filter without the lifting scheme, as summarized in Section 5.2. For $n_1 = 7$, in contrast to the results without lifting in Section 5.2, some overflows in the lifting computations caused the image quality to drop to values around 40 dB for most images with soft edges.

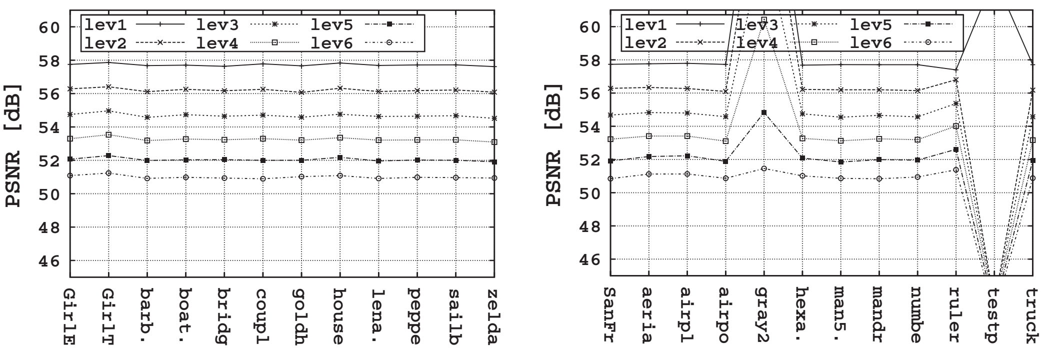
Overall, we conclude that the lifting scheme works robustly and achieves image qualities above 46 dB for $n_1 = 5$ for the considered wide range of image characteristics. With $n_1 = 6$, some overflows occur resulting in image qualities as low as 40 dB for sharp-edged images. The cost of the lifting scheme is the more complex and longer source code (which we provide). The longer source code adds on to the required program memory of the microcontroller (which was not bottleneck in our set-up as we did not run an operating system). Time savings are achieved through reductions of the computation steps for transform levels 2–6, as analyzed in Section 4.3.

5.4. Image compression performance

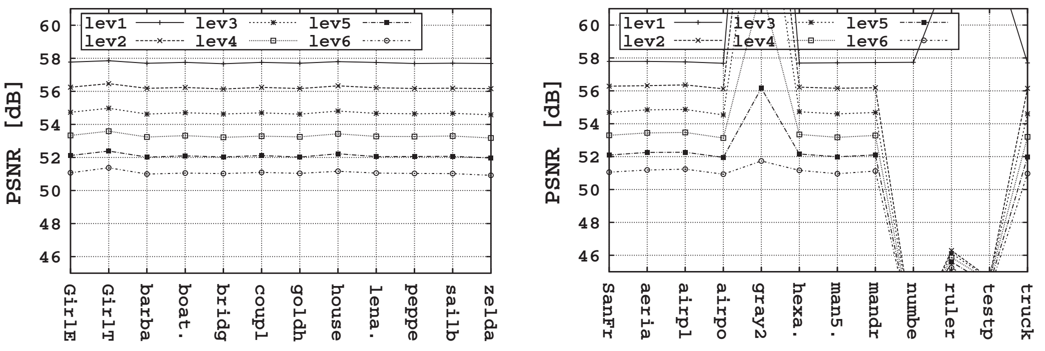
In this section we combine the fractional wavelet filter with a low-memory version of the SPIHT coder [24]. We demonstrate that the reductions in image quality observed in Sections 5.2 and 5.3 are negligible when the transformed image is compressed with a lossy wavelet compression algorithm.

We evaluate the compression performance for a wide set of 256 × 256 pixel images. We employ the fixed-point fractional wavelet filter with $n_1 = 6$ without the lifting scheme for the forward wavelet transform. In order to provide insight into the effects of the fractional wavelet filter and the low-memory image coder, we first plot in Fig. 6, analogous to the evaluation approach in Section 5.2, the image PSNR quality for fractional wavelet filter forward transform followed by conventional backward transform.

We next evaluate the combined fractional wavelet filter and low-memory wavelet transform in terms of the PSNR



(a) $N = 256$



(b) $N = 512$

Fig. 5. PSNR image quality with lifting using the fractional fixed-point wavelet filter with $n_1 = 6$.

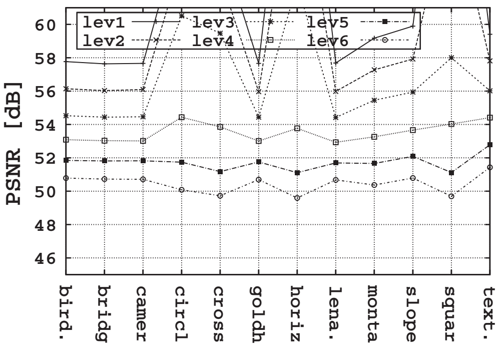


Fig. 6. PSNR image quality for fixed-point fractional wavelet filter forward transform with $n_1 = 6$ and no lifting and subsequent conventional backward transform. These 256×256 images are considered in the image compression evaluation in Fig. 7.

as a function of the compression ratio in bits per byte (i.e., per input pixel) denoted by `bpb` in Fig. 7. The PSNR is for the comparison between (*i*) the forward wavelet transformed, compressed, and subsequently uncompressed and backward transformed image compared with (*ii*) the original image. We denote the PSNR results obtained with the fixed-point fractional wavelet filter with $n_1 = 6$ without the lifting scheme for the forward transform in conjunction with the Wi2l image coder [24] by `Wi2l` in Fig. 7 We compare with state-of-the-art compression schemes, namely `Spiht` [9] obtained with the Windows implementation from Said et al. http://www.cipr.rpi.edu/research/SPIHT/spiht3.html, as well as `jpeg` and `jpeg2000` obtained with the JPEG and JPEG2000 implementations from the *jasper* project http://www.ece.uvic.ca/mdadams/jasper. These comparison benchmarks combine the image transform and image coding and are designed to run on personal computers with abundant memory.

We observe from Fig. 7 that the wavelet techniques Spiht, JPEG2000, and our fractional wavelet filter/Wi2l coder based scheme generally outperform the JPEG standard mainly because it is based on the discrete cosine transform. Especially for high compression rates, the wavelet techniques achieve significantly higher compression performance.

Importantly, we observe from Fig. 7 that the fractional wavelet filter/Wi2l coder based approach achieves state-of-the-art image compression that gives essentially the same PSNR image quality as JPEG2000 and SPIHT for the high compression rates (i.e., small bpb values). For the low compression rates (large bpb values), the fractional wavelet filter approach gives slightly lower image quality due to the loss in precision from the fixed-point arithmetic. However, the fractional wavelet filter/Wi2l approach achieves image qualities above 45 dB in the instances where it markedly drops below the other approaches (`circles`, `crosses`, `horizon`, and `squares` images) for low compression ratios. We observe from Fig. 6 that the fractional wavelet filter in isolation gives high image qualities for these images. This result suggests that these

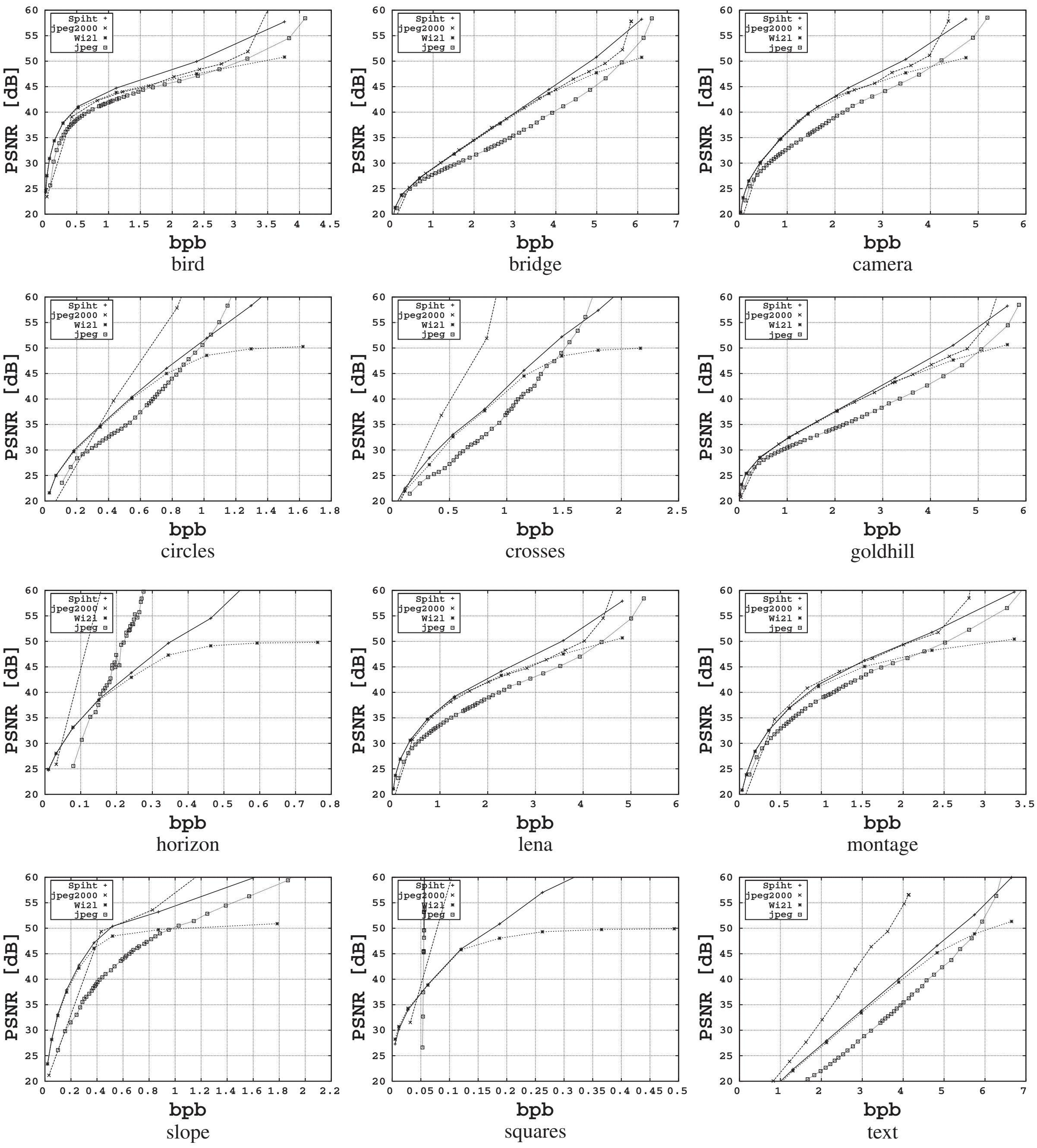


Fig. 7. The introduced wavelet filter combined with a low-memory image coder Wi2l gives nearly the same performance as JPEG2000, which reflects the state-of-the-art in image compression. While most implementations of JPEG2000 require a personal computer, the combination of fractional wavelet filter and image coder runs on a small 16 bit micro-controller with less than 2 kbyte of random access memory. Small wireless sensor nodes can thus perform image compression when extended with a flash memory, a small camera with serial interface, and a software update.

images may present unique challenges to the low-memory wavelet encoder. A more detailed evaluation of the low-memory wavelet coder is the topic of future research. Overall, we conclude from Figs. 6 and 7 that the combination of fractional wavelet filter and low-memory wavelet image coder [24] are well suited for sensor networks, which generally require a high compression (small bpb) due to the limited transmission energy and bandwidth.

6. Conclusions

We introduced and evaluated the fractional wavelet filter for computing the image wavelet transform on sensor nodes with small random access memory. The fractional wavelet filter computes the horizontal wavelet transform on the fly and computes the vertical wavelet transform through a novel iterative approach of computing and

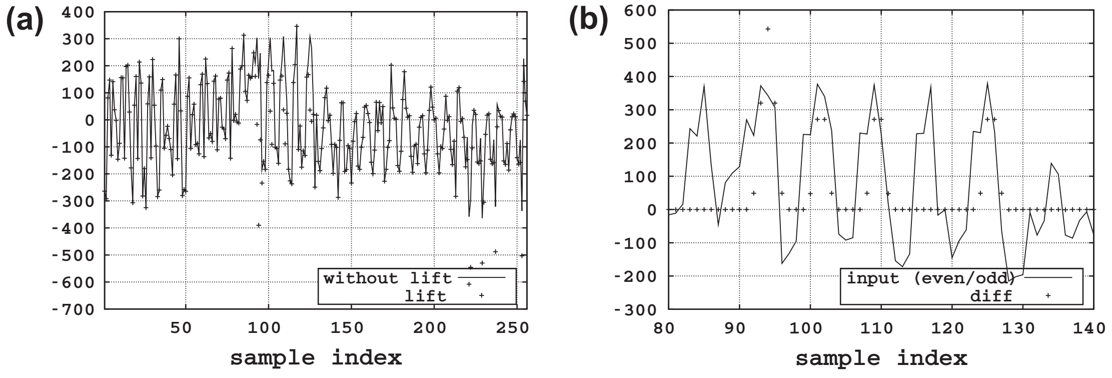


Fig. 8. (a) Comparison of fixed-point fractional wavelet filter without lifting, which has been verified to be correct, and with lifting. (b) Error in the result line (difference between correct results and results with lifting scheme) and the input samples sorted by even (lowpass values at left-hand side) and odd values (highpass values at right-hand side). The border line between even and odd values is at sample position 128.

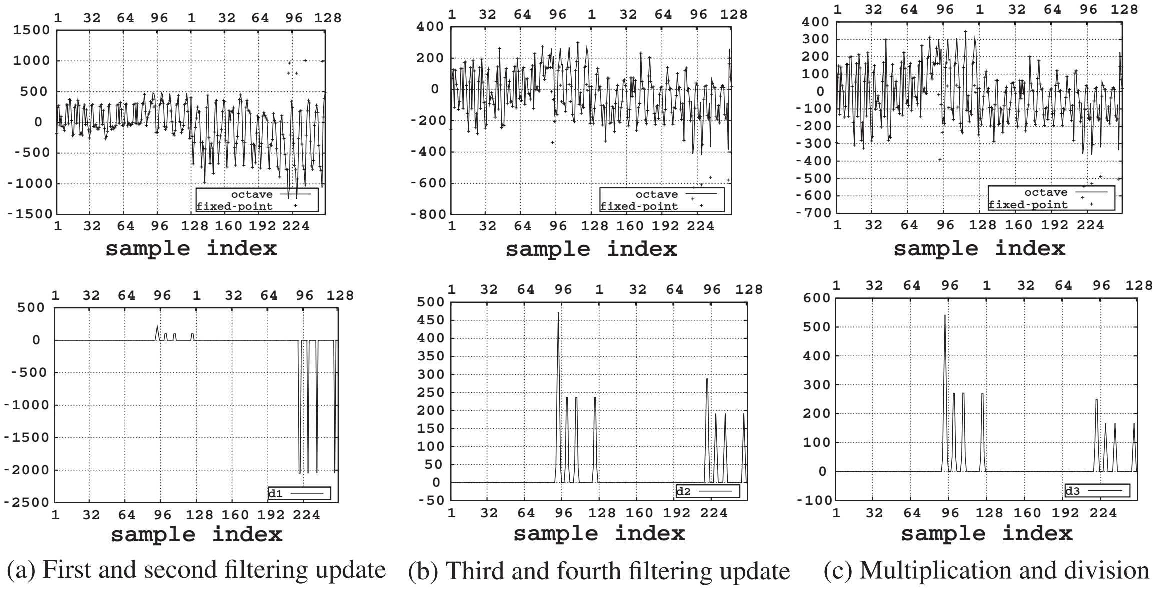


Fig. 9. Comparison of intermediate result values (top) and corresponding differences (bottom) between floating-point computation of lifting and fixed-point computation of lifting.

summing the fractional contributions made by the individual image lines. Only three image lines need to be kept in memory at any point during these computations. As a result, less than 1.5 kbyte of random access memory are required to transform a grayscale $N \times N = 256 \times 256$ image. A conventional data sensor node can thus be upgraded to a multimedia sensor node by adding some (external) flash memory, a small camera, and a software update. Upgrade costs and programming efforts are limited as the extension only requires a standard MMC card (flash memory card), a camera module, and a software update.

Aside from analyzing in detail the memory requirements, we have analyzed the computational complexity of the fractional wavelet filter. We have found that the reduction in memory from $2N$ images lines with the classical convolution approach to three image lines with the fractional wavelet filter comes at the expense of 2.89 more additions and 3.25 more multiplications. We demonstrate how this computational effort is substantially reduced with the lifting scheme for in-place computations. We also measured the time required for evaluating the fractional wavelet filter transform on a 16-bit micro-controller with integer arithmetic and detailed the contributions of computations as well as reading and writing from/to the flash memory to the total time.

The fractional wavelet filter is not lossless when evaluated with integer (fixed-point) arithmetic. However, our image quality evaluations indicate that the loss is not visible, as high PSNR values are obtained for appropriate integer number formats. In particular, we found that the $Q10.5$ format (with 5 bits for the fractional values) for the first transform level and one bit less for the fractional values for each successive transform level gives image qualities above 46 dB both without and with the lifting scheme.

We further combined the fractional wavelet filter with a customized low-memory image coder [24]. We found that for high compression ratios, this combined approach achieves image compression competitive with the state-of-the-art JPEG2000 compression.

One interesting direction for future research is to further reduce the memory requirements of the image wavelet transform by combining the fractional wavelet filter for the computation of the vertical transform with a partitioning of the horizontal transform. In particular, an image input line could be partitioned into two (or more) segments with appropriate overlap to avoid boundary effects. The fractional wavelet filter could then operate on these horizontal line segments.

Appendix A. Analysis of errors due to lifting scheme

In this appendix we analyze in detail the lifting scheme computations that lead to the lower image quality for the `numbers`, `rulers`, and `test pattern` images in Fig. 5 compared to the transform without lifting scheme in Fig. 3. We select the `numbers` image for our analysis. Since we employ the lifting scheme only for the horizontal wavelet transform of the second and higher transform levels we analyzed the result of the second level wavelet transform. Specifically, we examined the intermediate result after the horizontal second level wavelet transform is complete (which is then fed into the second level vertical transform), i.e., the L/H image of the second transform level. A comparison of this intermediate L/H transform result computed without the lifting scheme with the corresponding result computed with the lifting scheme revealed errors in line 248. As the vertical fractional filter computations did not introduce quality degradation for the fractional filter without lifting scheme, these errors must result from an incorrect lifting scheme calculation of the L and the H subband.

We extracted line 248 and computed the transform for this line using a conventional floating-point wavelet tool (Octave program) and compared with the fractional wavelet filter without the lifting scheme. Both approaches gave the same result, confirming that the fractional filter works correctly even for this “artificial” image with sharp edges.

We next compare in Fig. 8a) the correct values from the fractional filter without lifting scheme with the results from the fractional filter with the lifting scheme. The line has 256 wavelet coefficient values (samples), whereby the leftmost 128 samples are from the lowpass calculations and the rightmost 128 samples from the highpass calculations. We observe that incorrect values are computed with the lifting scheme as more clearly illustrated in Fig. 8b). In Fig. 8b) we plot the difference between the fractional filter with and without lifting as well as the original input samples. Interestingly, the first error occurs around sample 92, exactly at a very high input sample of the sample value 320. Clearly, the incorrect calculations result from overflow in the fixed-point format, as they appear after peaks of input data.

As the result coefficients of the lifting scheme are computed by several update iterations (see Section 3.4), we examine the intermediate results in Fig. 9. We compare verifying floating-point computations of the lifting scheme (with the Octave program) with the corresponding fixed-point computations in the top plots in Fig. 9; the bottom plots give the differences between both computations. Part (a) of Fig. 9 considers the first two filtering update operations, where the first operation gives the first intermediate highpass filtered result (128 rightmost samples), and the second update operation gives the first intermediate lowpass filtered result (128 leftmost samples). The horizontal scale on top of the plots refers to each of these filtering results. The four right-hand peaks in the lower plot in Fig. 9a result from the very large input peaks at positions 94, 101, 109, and 125, in Fig. 8b. This is due to the first filter update operation which is a convolution of the even values. The result of this convolution is not correct due to overflow. The incorrect result is carried over to the second filter update operation and results in the four smaller peaks on the left-hand side. The error is now propagated to the result of the next two filter update operations, as illustrated in Fig. 9b). Finally, the error is still visible after the multiplication and division with ζ, see Fig. 9c).

This error analysis illustrates that the few calculation (overflow) errors in the first update operation of the lifting scheme propagate to all following updates. Such an error propagation is not possible with the classical convolutional scheme for computation of the wavelet transform. The classical convolution separately computes each lowpass and highpass value. While the lifting scheme is an elegant computing methodology for the wavelet transform, computation (overflow) errors due to a few very large input values propagate and can result in an additional degradation of image quality, especially for artificial images with sharp edges.

References

[1] H. Karl, A. Willig, Protocols and Architectures for Wireless Sensor Networks, John Wiley & Sons, 2005.

[2] I.F. Akyildiz, T. Melodia, K.R. Chowdhury, A survey on wireless multimedia sensor networks, Computer Networks 51 (4) (2007) 921–960.

[3] S. Misra, M. Reisslein, G. Xue, A survey of multimedia streaming in wireless sensor networks, IEEE Communications Surveys and Tutorials 10 (4) (2008) 18–39. Fourth Quarter.

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

[5] A. Leventhal, Flash storage memory, Communications of the ACM 51 (7) (2008) 47–51.

[6] G. Mathur, P. Desnoyers, D. Ganesan, P. Shenoy, Ultra-low power data storage for sensor networks, in: Proceedings of the International IEEE/ACM Syposium on Information Processing in Sensor Networks (IPSN), 2006, pp. 374–381.

[7] D. Lee, S. Dey, Adaptive and energy efficient wavelet image compression for mobile multimedia data services, in: Proceedings of IEEE International Conference on Communications (ICC), 2002.

[8] J. Shapiro, Embedded image coding using zerotrees of wavelet coefficients, IEEE Transactions on Signal Processing 41 (12) (1993) 3445–3462.

[9] A. Said, W. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees, IEEE Transactions on Circuits and Systems for Video Technology 6 (3) (1996) 243–250.

[10] T. Fry, S. Hauck, SPIHT image compression on FPGAs, IEEE Transactions on Circuits and Systems for Video Technology 15 (9) (2005) 1138–1147.

[11] K.-C. Hung, Y.-J. Huang, T.-K. Truong, C.-M. Wang, FPGA implementation for 2D discrete wavelet transform, Electronics Letters 34 (7) (1998) 639–640.

[12] J. Chilo, T. Lindblad, Hardware implementation of 1D wavelet transform on an FPGA for infrasound signal classification, IEEE Transactions on Nuclear Science 55 (1) (2008) 9–13.
[13] S. Ismail, A. Salama, M. Abu-ElYazeed, FPGA implementation of an efficient 3D-WT temporal decomposition algorithm for video compression, in: Proceedings of the IEEE International Symposium on Signal Processing and Information Technology, December 2007, pp. 154–159.
[14] C.-H. Hsia, J.-M. Guo, J.-S. Chiang, Improved low-complexity algorithm for 2-D integer lifting-based discrete wavelet transform using symmetric mask-based scheme, IEEE Transactions on Circuits and Systems for Video Technology 19 (8) (2009) 1202–1208.
[15] W. Li, C. Hsia, J.-S. Chiang, Memory-efficient architecture of 2-D dual-mode discrete wavelet transform using lifting scheme for motion-JPEG2000, in: Proceedings of IEEE ISCAS, 2009, pp. 750–753.
[16] M. Vishwanath, The recursive pyramid algorithm for the discrete wavelet transform, IEEE Transactions on Signal Processing 42 (3) (1994) 673–676.
[17] B.-F. Wu, C.-F. Lin, A high-performance and memory-efficient pipeline architecture for the 5/3 and 9/7 discrete wavelet transform of JPEG2000 codec, IEEE Transactions on Circuits and Systems for Video Technology 15 (12) (2005) 1615–1628.
[18] S. Rein, C. Gühmann, F. Fitzek, Sensor networks for distributive computing, in: F. Fitzek, F. Reichert (Eds.), Mobile Phone Programming, Springer, 2007, pp. 397–409.
[19] B. Rinner, W. Wolf, Toward pervasive smart camera networks, in: H. Aghajan, A. Cavallaro (Eds.), Multi-Camera Networks: Principles and Applications, Academic Press, 2009, pp. 483–496.
[20] A. Ciancio, A. Ortega, A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting, in: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2005, pp. 825–828.
[21] A. Ciancio, S. Pattem, A. Ortega, B. Krishnamachari, Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet compression algorithm, in: Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN), 2006, pp. 309–316.
[22] Q. Lu, W. Luo, J. Wang, B. Chen, Low-complexity and energy efficient image compression scheme for wireless sensor networks, Computer Networks 52 (13) (2008) 2594–2603.
[23] H. Wu, A. Abouzeid, Energy efficient distributed image compression in resource-constrained multihop wireless networks, Computer Communications 28 (14) (2005) 1658–1668.
[24] 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), Snowbird, UT, March 2009, pp. 252–261.
[25] D.-U. Lee, H. Kim, M. Rahimi, D. Estrin, D. Villasenor, Energy-efficient image compression for resource-constrained platforms, IEEE Transactions on Image Processing 18 (9) (2009) 2100–2113.
[26] C. Chrysafis, A. Ortega, Line-based, reduced memory, wavelet image compression, IEEE Transactions on Image Processing 9 (3) (2000) 378–389.
[27] J. Oliver, M. Malumbres, On the design of fast wavelet transform algorithms with low memory requirements, IEEE Transactions on Circuits and Systems for Video Technology 18 (2) (2008) 237–248.
[28] 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 '08), July 2008.
[29] R.K. Bhattar, K.R. Ramakrishnan, K.S. Dasgupta, Strip based coding for large images using wavelets, Signal Processing: Image Communication 17 (6) (2002) 441–456.
[30] L. Chew, L.-M. Ang, K. Seng, New virtual Spiht tree structures for very low memory strip-based image compression, IEEE Signal Processing Letters 15 (2008) 389–392.
[31] L.W. Chew, L.-M. Ang, K.P. Seng, Low memory strip-based image compression for color images, in: Proceedings of IEEE International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2009, pp. 363–366.
[32] Y. Bao, C. Kuo, Design of wavelet-based image codec in memory-constrained environment, IEEE Transactions on Circuits and Systems for Video Technology 11 (5) (2001) 642–650.
[33] V. Ratnakar, TROBIC: two-row buffer image compression, in: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 1999, pp. 3133–3136.
[34] C.-K. Hu, W.-M. Yan, K.-L. Chung, Efficient cache-based spatial combinative lifting algorithm for wavelet transform, Signal Processing 84 (9) (2004) 1689–1699.
[35] X. Zhang, L. Cheng, H. Lu, Low memory implementation of generic hierarchical transforms for parentchildren tree (PCT) production and its application in image compression, Signal Processing: Image Communication 24 (5) (2009) 384–396.
[36] H. Pan, W.-C. Siu, N.-F. Law, A fast and low memory image coding algorithm based on lifting wavelet transform and modified SPIHT, Signal Processing: Image Communication 23 (3) (2008) 146–161.
[37] D. Zhao, Y. Chan, W. Gao, Low-complexity and low-memory entropy coder for image compression, IEEE Transactions on Circuits and Systems for Video Technology 11 (10) (2001) 1140–1145.
[38] B. Usevitch, A tutorial on modern lossy wavelet image compression: foundations of JPEG2000, IEEE Signal Processing Magazine 18 (5) (2001) 22–35.
[39] S. Rein, M. Reisslein, Low-memory wavelet transforms for wireless sensor networks: a tutorial, IEEE Communications Surveys and Tutorials (2010), in press. Preprint <http://mre.faculty.asu.edu/fwftut.pdf>.
[40] V. Lecuire, C. Duran-Faundez, N. Krommenacker, Energy-efficient transmission of wavelet-based images in wireless sensor networks, EURASIP Journal on Image and Video Processing 2007 (47345) (2007) 1–11.
[41] I. Daubechies, Ten Lectures on Wavelets, SIAM, 1992.
[42] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, Journal of Fourier Analysis and Applications 4 (3) (1998) 247–269.
[43] W. Sweldens, The lifting scheme: a construction of second generation wavelets, The SIAM Journal on Mathematical Analysis 29 (2) (1997) 511–546.
[44] K. Rao, P. Yip (Eds.), The Transform and Data Compression Handbook, CRC Press, 2001.

Stephan Rein studied electrical and telecommunications engineering at RWTH Aachen University and Technical University (TU) Berlin, where he received the Dipl.-Ing. degree in December 2003 and the Ph.D. degree in January 2010. He was a research scholar at Arizona University in 2003, where he conducted research on voice quality evaluation and developed an audio content search machine. From February 2004 to March 2009 he was with the Wavelet Application Group at TU Berlin developing text and image compression algorithms for mobile phones and sensor networks. Since July 2009 he is with the Telecommunication Networks Group at TU Berlin, where he is currently working on multimedia delivery to mobile devices.

Martin Reisslein is an Associate Professor in the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU), Tempe. He received the Dipl.-Ing. (FH) degree from the Fachhochschule Dieburg, Germany, in 1994, and the M.S.E. degree from the University of Pennsylvania, Philadelphia, in 1996. Both in electrical engineering. He received his Ph.D. in systems engineering from the University of Pennsylvania in 1998. During the academic year 1994–1995 he visited the University of Pennsylvania as a Fulbright scholar. From July 1998 through October 2000 he was a scientist with the German National Research Center for Information Technology (GMD FOKUS), Berlin and lecturer at the Technical University Berlin. From October 2000 through August 2005 he was an Assistant Professor at ASU. From January 2003 through February 2007 he served as Editor-in-Chief of the *IEEE Communications Surveys and Tutorials*. He currently serves as Associate Editor-in-Chief of the *IEEE Communications Surveys and Tutorials* and as Associate Editor for the *IEEE/ACM Transactions on Networking* and for *Computer Networks*. He has served on the Technical Program Committees of leading networking conferences, such as *IEEE Infocom*. He maintains an extensive library of video traces for network performance evaluation, including frame size traces of MPEG-4 and H.264 encoded video, at http://trace.eas.asu.edu. His research interests are in the areas of video traffic characterization, wireless networking, optical networking, and engineering education.