Fractional Wavelet Filter for Camera Sensor Node with external Flash and extremely little RAM

Stephan Rein
stephan.rein@tu-berlin.de

Stephan Lehmann
stephan.lehmann@tu-berlin.de

Clemens Gühmann
clemens.guehmann@tu-berlin.de

Wavelet Application Group
Department of Energy and Automation Technology
Chair of Electronic Measurement and Diagnostic Technology
Technische Universität Berlin

ABSTRACT

This paper introduces the *fractional wavelet filter* as a technique to compute fractional values of each wavelet subband, thus allowing a low-cost camera sensor node with less than 2 kByte RAM to perform a multi-level 9/7 picture wavelet transform. The picture dimension can be 256x256 using fixed-point arithmetic and 128x128 using floating-point arithmetic. The technique is applied to a typical 16 Bit sensor node architecture with external flash memory, which allows to line-wisely read and write picture data.

Categories and Subject Descriptors

I.4.10 [**Computing Methodologies**]: Image Processing and Computer Vision—*Image Representation*; B.3.2 [**Hardware**]: Memory Structures—*Design Styles*

General Terms

Algorithms, Performance, Measurement

Keywords

Fractional wavelet filter, Camera sensor node, Flash memory

1. INTRODUCTION

Camera sensor nodes are small wireless devices that gather, process and transfer environmental pictures. These devices can be stationary or mounted on moving objects. A possible application for a camera sensor network may be remote assistance for moving vehicles concerning localization and computer-assisted maintenance.

As the bandwidth is very limited in wireless sensor networks, data compression techniques are inevitable when large amounts of data, e.g., pictures, have to be transferred. The wavelet transform is an established decorrelating tool that allows coding algorithms to achieve superior compression gains among current picture compression techniques. A standard coding technique is the SPIHT algorithm [9] that especially outperforms other techniques for high compression rates while being conceptually fairly simple. However, as the wavelet transform itself is a memory intensive procedure, wavelet based compression has hardly arrived on the low energy consuming micro-controllers that drive low cost sensor nodes. Thus the design of low-memory wavelet transform schemes is still a current research activity, see section 1.1. This paper addresses this problem by introducing a wavelet filter system - that is, the *fractional wavelet filter*, for the Daubechies 9/7 wavelet (also called the *FBI-Wavelet*), which reads and writes the picture data row-wisely and roughly allocates 1.2 kBytes for a fixed-point wavelet transform with the picture dimension N=256. We do not consider the conceptually less complex 5/3 wavelet transform that works with less coefficients or the integer wavelet transform, as these generally give lower compression gains.

The fractional filter is applied and evaluated on an own sensor node architecture called *Spisa* (see `http://www.mdt.tu-berlin.de/research/wavelets`) or *OpenSensor* [8]. It uses the *Microchip dsPIC30F4013*, i.e., a 16 Bit digital signal controller with 2 kByte RAM, the camera module C328-7640 with an universal asynchronous receiver/transmitter (UART) interface (available at `http://www.comedia.com.hk`), and an external 64 MByte secure digital (SD) card as a flash memory, connected to the controller through a serial peripheral interface (SPI) bus. The data of the SD-card is accessed through an own filesystem [10]. Camera and SD-card both can be connected to any controller with UART and SPI ports. The system is designed to capture still images.

The paper is organized as follows. In the next subsection, related literature is reviewed. In section 2, the main principle of the picture wavelet transform and its implementation is outlined. In section 3, the fractional filter for the wavelet transform is detailed. We have implemented two forward transforms, one using floating-point numbers for high precision and another transform using fixed-point numbers that needs less memory while being computationally more suitable for a 16 Bit controller. In section 4, the two implementations for the fractional filter are evaluated on a 16 Bit signal controller sensor node with two kByte RAM concerning picture quality, flash memory access and computation

Mobimedia 2008 July 7-9, 2008, Oulu, Finland.
 ISBN 978-963-9799-25-7/08/07
DOI 10.4108/ICST.MOBIMEDIA2008. 4026

times. In the last section we conclude the paper and describe our future work.

1.1 Related Work

One major difficulty in applying the discrete two- dimensional wavelet transform to a platform with scarce resources is the need for large memory. Implementations on a personal computer (PC) generally keep the whole source and/or destination picture in memory, where horizontal and vertical filters are applied separately. As this is generally not possible on a resource-limited platform, the recent literature addresses the memory-efficient implementation of the wavelet transform. A very large part of the literature concerns the implementation of the wavelet transform on field programmable gate arrays (FPGA), see for instance [11, 3, 1, 4]. The FPGA-platforms are generally designed for one special purpose and are not an appropriate candidate for a sensor node that has to perform many tasks concerning communication and analysis of the surrounding area, see [5, 8] for details. This work is different from the literature on FPGAs in that it considers the 9/7 picture wavelet transform for a micro- or a signal-controller with very little RAM. Such a solution can easily be integrated in current sensor network platforms and offers much flexibility. Costs and programming efforts are limited as the extension only concerns a standard SD-card, a camera module, and a software module for the transform. The most related work to this paper is given in [2], where a line-based version of the wavelet transform is given. The authors describe a system of buffers where only a small subset of the coefficients has to be stored thus tremendously reducing the memory requirements compared to the traditional approach. A very efficient implementation of the line-based transform using the lifting scheme and improved communication between the buffers is detailed in [6], where the authors use a PC-based C++ implementation for demonstration. However, in the context of our requirements it is not applicable to a sensor node with extremely little RAM as it uses in ideal case 26 kByte RAM for a six-level transform of an 512x512 picture, whereas the approach given in this paper would only require roughly 5 kByte using the floating-point scheme. We could not find any paper where the 9/7 picture wavelet transform is implemented on a low-cost 16 Bit signal-/micro-controller.

2. WAVELET TRANSFORM

A tutorial on the picture wavelet transform and its features is given in [12]. We here just outline how the wavelet transform can be computed with filter operations while we start with one dimension to explain the basics for the picture transform. In the following subsections the fractional wavelet filter is detailed using floating-point and fixed-point numbers.

2.1 Wavelet Filtering

A one-dimensional discrete wavelet transform can be performed by low- and highpass filtering of an input signal with the dimension N, which can be a single line of a picture. The two resulting signals are then sampled down - that is, each second value is discarded, and then form the *approximations* and *details*, which are two sets of $N/2$ wavelet coefficients. In this work the Daubechies biorthogonal 9/7 filter is employed, which is part of the JPEG2000 standard and the basis for many wavelet compression algorithms, including

	lowpass		highpass	
index	real	Q15	real	Q15
0	0.852699	27941	0.788486	25837
±1	0.377403	12367	-0.418092	-13700
±2	-0.110624	-3625	-0.040689	-1333
±3	-0.023849	-781	0.064539	2115
±4	0.037828	1240	0	0

Table 1: 9/7 Analysis wavelet filter coefficients in real and Q15 data format.

the *embedded zerotree* (EZW) or the *set partitioning in hierarchal trees* (SPIHT) algorithm [12]. The approximations $L(i)$ can be computed with

$$L(i) = \sum_{j=-4}^{4} line_{pic}(i+j) \cdot Al(j), \qquad i = 0 \dots N-1 \quad (1)$$

and the details $H(i)$ with

$$H(i) = \sum_{j=-3}^{3} line_{pic}(index+i) \cdot Ah(j), \qquad i = 0 \dots N-1, \quad (2)$$

where $Al(j)$ and $Ah(j)$ denote the filter coefficients given in table 1, respectively. Note that the sample down operation can be incorporated by only computing the required coefficients, thereby avoiding useless computation. At the signal boundaries, we perform a symmetrical extension to avoid border effects, e.g., a signal $s(i), i = 0 \dots N-1$ is extended for the highpass filter as follows:
$s_{ext} = s(3)s(2)s(1)s(0)s(1) \dots s(N-1)s(N-2)s(N-3)s(N-4)$
The filter is slided over the signal in such a way that the center of the filter is located upside the i th signal value s(i), e.g., for $i = 1$:

Ah(-3)	Ah(-2)	Ah(-1)	Ah(0)	Ah(1)	Ah(2)	Ah(3)
s(2)	s(1)	s(0)	s(1)	s(2)	s(3)	s(4)

For the lowpass filter the center of the filter slides over the even values $i = 0, 2, \dots, N-1$, for the highpass it moves over the odd values $i = 1, 3, 5, \dots N$.

A one-level picture wavelet transform can be computed by first performing a one-dimensional transform for each line and then repeat this operation for all the columns of the result - that is, apply the low- and the highpass in vertical direction. The horizontal transform results into 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, HH, which are the so-called *subbands*. The two operations can be repeated on the LL subband to get the higher level subbands, e.g., $HL2$ refers to the second level HL subband that is computed by first horizontally filtering $LL1$ and then vertically filtering the result. An example of a two-level wavelet transform is given in figure 1.

3. FRACTIONAL WAVELET FILTER

In this section the fractional wavelet filter - that is, a computational scheme to compute the picture wavelet transform with very little RAM memory, is explained. The data on the SD-card can only be accessed in blocks of 512 Bytes, thus a sample-wise access as easily executed with RAM memory

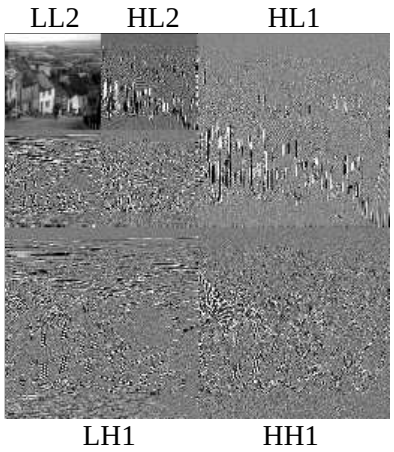

Figure 1: An example of a 2-level picture wavelet transform. Each one-level transform results into four subbands denoted by LL, LH, HL, HH. The contrast of each subband was adjusted to fill the entire intensity range.

on PCs is here not feasible. Even if it is possible to access a smaller number of samples of a block, the read/write time would significantly slow down the algorithm, as the time to load a few samples is the same as for a complete block. The fractional filter has to take this restriction into account.

For the first level, the algorithm reads the picture samples line by line from the SD-card while it writes the subbands line-wisely to a different destination on the SD-card. Two single lines of the LL/HL subbands and two lines of the LH/HH subbands build a destination line, respectively. For the next level the LL subband contains the input data. Note that the input samples for the 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 and for each level a new destination matrix is allocated on the SD-card. However, as the SD-card has plenty of memory, it does not affect the sensor's resources. This also allows to reconstruct the picture from any level (and not necessarily from the highest level, as it would be necessary for a transform illustrated in figure 1). The scheme is illustrated in figure 2.

3.1 Floating-Point Filter

The floating point wavelet filter computes the wavelet transform with a 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 pictures can be reconstructed without loss of information. The wavelet filter uses three buffers of the dimension N - one for the current input line and two buffers for two destination lines each of them building a row of the LL/HL subbands and a row of the LH/HH subbands. The filter computes the horizontal wavelet coefficients on the fly, while it computes a set of *fractions* (here we denote fractions as a part of a sum) of each subband destination line. Let $ll(i,j,k), lh(i,j,k), hl(i,j,k)$, and $hh(i,j,k)$ denote the fractional subband wavelet coefficients, where $i = 0, 2, 4, \ldots N/2-1$ and $j = -4 \cdots +4$ assign the current input line as $i*2+j$. $k = 0 \ldots N/2-1$ denotes

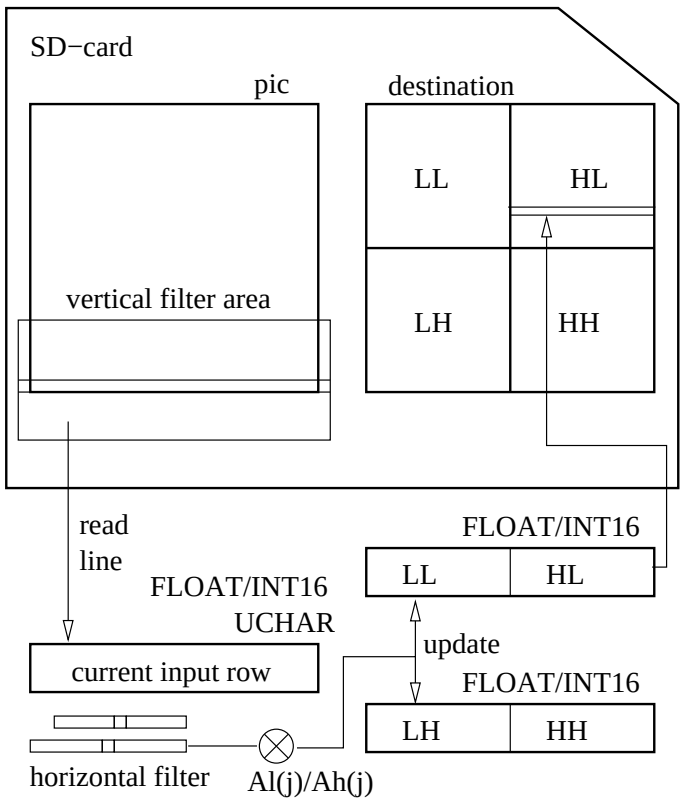

Figure 2: Proposed system for the picture wavelet transform. The horizontal wavelet coefficients are computed on the fly and employed to compute the fractional wavelet coefficients that update the subbands. For each wavelet level, a different destination object is written to the SD-card. Thus the picture can be reconstructed from any level.

the horizontal coefficient index. The fractional coefficients are defined as follows:

$$ll(i,j,k) = Al(j) \sum_{l=-4}^{4} pic(i*N+k+l) \cdot Al(l) \qquad (3)$$

$$lh(i,j,k) = Ah(j) \sum_{l=-3}^{3} pic(i*N+k+l) \cdot Al(l) \qquad (4)$$

$$hl(i,j,k) = Al(j) \sum_{l=-4}^{4} pic(i*N+k+l) \cdot Ah(l) \qquad (5)$$

$$hh(i,j,k) = Ah(j) \sum_{l=-3}^{3} pic(i*N+k+l) \cdot Ah(l) \qquad (6)$$

The final coefficients are computed by update operations and thus first have to be initialized: $LL(i,k) = LH(i,k) = HL(i,k) = HH(i,k) = 0 \mid_{\forall i,k}$. For $j = -4 \ldots 4$, the update operations are then given by

$$LL(i,k) += ll(i,j,k) \qquad LH(i,k) += lh(i,j,k) \qquad (7)$$

$$HL(i,k) += hl(i,j,k) \qquad HH(i,k) += hh(i,j,k) \qquad (8)$$

The special requirement for the fractional filter is that j stays constant for updating all subband rows. The process of updating the destination lines is repeated until the final

subband coefficients have been estimated. The pseudo-code of this procedure uses the horizontal filter functions *filtL* and *filtH*. The low- and highpass filter coefficients from table 1 are accessed through *al(j)* and *ah(j)*, where j denotes the filter index. The floating-point code for the first level is given as follows:

```
FLOAT LL_HL [N];                                                    1
FLOAT LH_HH [N];                                                    2
UINT8 line_pic [N];//current row                                    3
INT8 i;//current row index                                          4
INT8 j;//vertical filter index                                      5
INT8 k;//horizontal destination index                               6
for (i=N/2-1;i>=0;i--){                                             7
  //init the two destination buffers:                               8
  UINT8 b;                                                          9
  for(b=0;b<N;b++){LL_HL[b]=0;LH_HH[b]=0;}                          10
  for (j=-4;j<=4;j++){                                              11
    INT16 line_index=i*2+j;                                         12
    if (line_index<0) line_index*=-1;//sym.                         13
    else if (line_index>N-1)          //exten                       14
      line_index=2*N-2-line_index;    //sion                        15
    //get the current line from the SD-card:                        16
    GetFromSD(line_index*N,&line_pic,N,'pic');                      17
    for (k=0;k<N/2;k++){                                            18
      FLOAT L=filtL(&line_pic,k*2,N);                               19
      LL_HL[k]+=L*al(j);  //update LL                               20
      LH_HH[k]+=L*ah(j-1);//update LH                               21
      FLOAT H=filtH(&line_pic,k*2+1,N);                             22
      LL_HL[k+N/2]+=H*al(j);  //update HL                           23
      LH_HH[k+N/2]+=H*ah(j-1);//update HH                           24
    }                                                               25
  }                                                                 26
  //here write the two buffers 'LL_HL'                              27
  //and 'LH_HH' to the SD-card:                                     28
  WriteToSD(i*N,&LL_HL,N,'dest');                                   29
  WriteToSD((i+N/2)*N,&LH_HH,N,'WTpic');                            30
}                                                                   31
```

For each vertical filter area, nine input lines have to be read. As the filter moves up by two lines (implicit vertical down sampling), the total number of lines to be read is given as $N/2 \cdot 9$, where NxN is the picture dimension.

When the filter input area moves up, the input row of the last block could be used for the current filter area, thus reducing the number of repetitive readings. For an NxN picture, the number of line readings would reduce to to $N/2 \cdot 8$. For simplicity and because of time constraints, this is not yet implemented in the fractional filter.

For the higher levels the input data is not the original picture anymore but the previous LL-subband. Thus, the current input row (see line 3) has to be of the type float (32 Bit). The number of bytes required for a multi-level wavelet transform of a picture with the dimension N_{pic} is given as

$$Bytes_{float} = \begin{cases} N_{pic}/level \cdot 9, & level = 1 \\ N_{pic}/level \cdot 12, & level > 1, \end{cases}$$

where *level* refers to the required wavelet level.Table 3.1 gives the required Bytes for the floating point implementation with an 128x128x8 input picture for different levels.

3.2 Fixed-Point Filter

As many other low-cost processors, the dsPIC controller does not give any hardware support for floating-point operations. If they are coded anyway, the compiler translates them to integer operations. Switching an algorithm from floating- to fixed-point can result into time and power savings with the cost of less precision, need for a thorough number range analysis, and more programming work. Thus using a fixed-point format for the fractional filter can help to reduce the computational requirements and to reduce the RAM memory needed for the destination subbands. A tutorial on using the fixed-point format for wavelet filtering is given in [7]. We here just shortly review the methodology for fixed-point arithmetic. Fixed-point numbers are internally stored as an integer number (and the two- complement is used for the bit-wise representation). A binary N-bit fixed-point number denoted by its integer value a has $exp(a)$ fractional bits - that are the bits after the radix point, and $N - exp(a) - 1$ integer bits. Note that $exp(.)$ here refers to the exponent and not to the exponential function. A standard notation for such a format is the Texas Instruments Qm.n format, with $m = N - exp(a) - 1$ and $N = exp(a)$. The real number A to be interpreted by the user is then given as

$$A = a \cdot 2^{-exp(a)}. \quad (9)$$

The fractional fixed-point filter can then be realized by first transforming the real wavelet coefficients to the Q0.15 format, see table 1. The second requirement is that for all add and multiply operations the exponent has to be taken into account. For an add operation, the both operands must have the same exponent. A multiply operation will require a result exponent given by the sum of the input exponents. Both operations can be supported by an exponent change function that is realized by left or right bit-shift. Regarding these requirements, the fixed-point filter can be programmed similarly as the floating-point filter.

As the number range of the wavelet coefficients gets larger from level to level, the output data format has to be enlarged from level to level. The study in [11] on the required range reports the data formats in table 3.1 to be sufficient. (Note that there is a general difference when the usage of fixed-point numbers for wavelet transforms is discussed in the literature. Sometimes, the fixed-point numbers are only used for the representation of the wavelet coefficients, as it is done in [11]. In this work, when a fixed-point algorithm is discussed, this includes the coefficient representation as well as the internal calculations.) For the first wavelet level, the input data format is Q15.0, as the picture samples are integer numbers. Note that the L- and H-wavelet coefficients (computed by the functions *filtL and filtH*) are already computed in the data format of the final level (even if they may need a smaller range than the subband coefficients). The

	floating-point		fixed-point		
N	level	Bytes	level	Bytes	Format
256		-	1	1280	Q10.5
128	1	1152	2	768	Q11.4
64	2	640	3	640	Q12.3
32	3	320	4	512	Q13.2
16	4	160	5	384	Q12.2

Table 2: RAM Bytes required for each Wavelet Level using a 128x128 picture for floating-point and an 256x256 picture for fixed-point calculation. The data format for fixed-point calculation is given in the Texas Instruments Notation.

memory requirements for the fixed-point filter are given as

$$Bytes_{fixed} = \begin{cases} N_{pic}/level \cdot 5, & level = 1 \\ N_{pic}/level \cdot 6, & level > 1 \end{cases}$$

and are given in table 3.1 for the levels 1-5.

4. PERFORMANCE EVALUATION

4.1 Quality Measurements

The floating-point transform gives perfectly reconstructed pictures. The fixed-point transform introduces errors through the filter computations.

For the quality evaluation the *GreySet1* test images from the *Waterloo Repertoire* (available at `http://links.uwaterloo.ca/bragzone.base.html`) were chosen. This set contains 12 256x256x8 greyscale images in the graphics interchange format (GIF). The pictures were converted to the portable network graphics (PNG) format using the *convert* command of the software suite *ImageMagick* (available at `http://www.imagemagick.org`). Finally, the data was converted to plain text with unsigned char numbers using the software *Octave*. *ImageMagick* and *Octave* both are included in many free linux distributions.

The quality measurements evaluate the fractional fixed-point filter computing the peak signal-to-noise ratio (PSNR) to compare the original image with the reconstructed image, which is generated through a wavelet transform of the orginal image followed by an inverse wavelet transform. We have implemented a reference transform in C including a floating-point and a fixed-point format version, which were employed to develop the fractional wavelet filter for the camera sensor. The reference implementation gives the same results than the final forward fractional filter for the camera sensor and thus was employed for the PSNR measurements. The measurements include six wavelet levels for the given set of pictures. Even if for us there was no quality degradation visible, the picture reconstruction was not lossless. Figure 3 demonstrates that the PSNR values for the first level range from 60 to 87 dB, whereas the higher levels give much lower PSNR ranging from 47 to 57 dB. The reason for this difference may be that the input for the first level are integer numbers that are represented without loss of precision. For the levels 2-6 there is roughly a difference of 1-2 dB between the levels. The loss of precision affects the picture quality, but this may be of little interest when a lossy compression algorithm is employed, as for instance the bitplane coding technique SPIHT. In the lossy mode, SPIHT cuts the least significant bits of and thereby can give very high compression ratios.

4.2 Time Measurements

For the time measurements the forward fractional wavelet filter was employed on the camera sensor for a six-level forward wavelet transform. The 2 kBytes of the sensor's RAM did allow to transform a 128x128x8 picture with floating-point precision and a 256x256x8 picture with reduced fixed-point precision. Table 4.2 gives the times needed for the transform listed by read and write SD-card access times and processor computing times. The floating-point transform takes 16.18 seconds and the fixed-point transform 11.64 seconds. 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. The floating-point computations are about seven times slower than the fixed-point computations. This is even more remarkable, as the computing times take the largest amount of total time. Unexpectedly, the flash memory is not the bottleneck of the fractional filter, even if there is large overhead in the read process, as the rows are read repetitively. 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 picture while it needs the same amount of RAM.

	time [sec]			
	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

Table 3: SD-card access and processor computing times for a six-level wavelet transform. For the floating-point measurement an 128x128x8 picture and for the fixed-point measurement a 256x256x8 picture were transformed. The floating- point algorithm is very slow because the compiler translates the computations to 16 Bit integer operations.

5. CONCLUSION

In this paper the fractional wavelet filter technique was introduced as a computation scheme that allows the picture wavelet transform to be implemented with very low RAM requirements. The fractional wavelet is applied to a camera sensor node that uses a 16 Bit low-cost signal controller with 2 kByte of RAM. The floating-point fractional wavelet transform takes an 128x128x8 picture as an input and needs 16 seconds for six wavelet levels, whereas the fixed-point filter only roughly needs 12 seconds while transforming a 256x256x8 picture.

Even if we could not see any quality degradation in the pictures, the fixed-point reconstruction is not lossless. However, when a bitplane coding technique is applied the quality may not be affected in the lossy mode as the least significant bits of the coefficients are cut anyway. The fixed-point filter is a preferable choice as the wavelet based coding techniques especially give superior results with high compression ratios.

The fractional filter uses a SD-card to read the image and to write the wavelet subbands. An external flash memory is a very realistic extension to a camera sensor node, as the data of several pictures has to be stored anyway, including the original and the transformed picture. It is barely feasible to immediately send out the computed wavelet coefficients to the sensor network, as there might be network congestion or internal ongoing operations with higher priority. The fractional filter operates line by line and thus allows the usage of a filesystem that only can access blocks of 512 Bytes. Remarkably, as the time measurements show, the SD-card is not the bottleneck of the transform but the filter computations.

In our future work we will implement an inverse fractional filter and incorporate the lifting scheme. The lifting scheme cannot be applied to the vertical filtering technique of the fractional filter and also not to the horizontal transform of the first level, as the input lines of this level have to use an integer buffer array to not exceed the memory. The lifting scheme allows to compute the one- dimensional transform

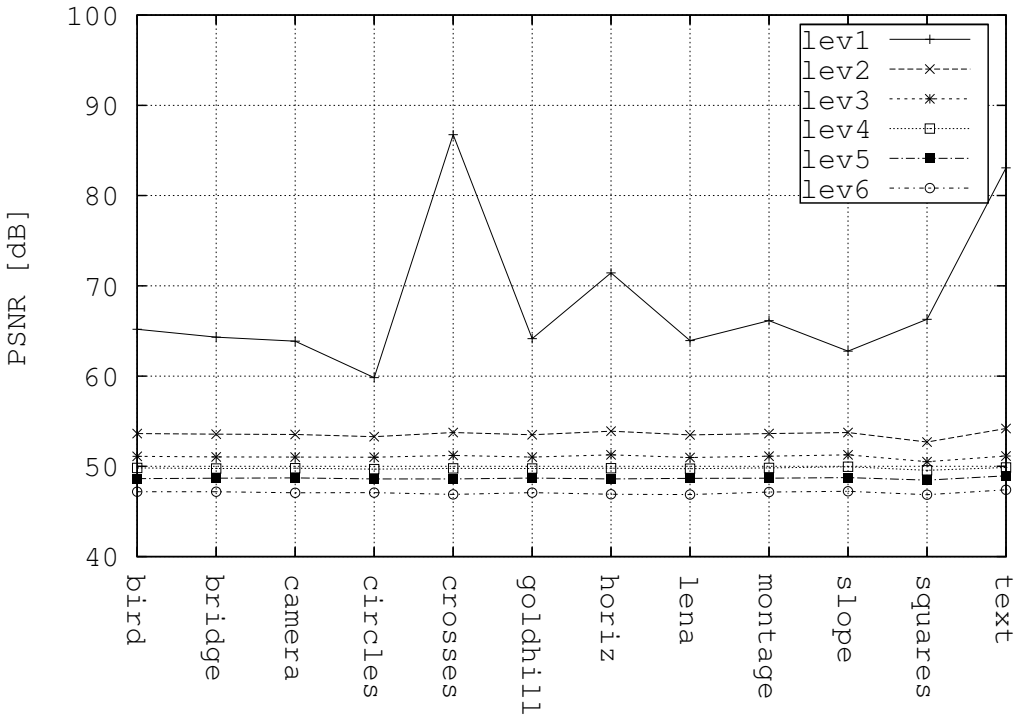

Figure 3: PSNR quality measurements for the fixed-point wavelet transform. The proposed system uses the fractional filter for the forward transform and a standard inverse transform. Even if there was no quality degradation visible, the transform is not lossless. The first level gives very high PSNR values as the transform input values are integers. The quality loss may not affect a lossy compression algorithm as least significant bits are cut anyway.

in place -that is,there is only one buffer for the input and output values, where the applied buffer must have the appropriate size to contain the final wavelet coefficients. It thus can be applied to the higher levels of the horizontal transform, as the input lines for these levels take variables with the larger data format anyway (and not just 8 Bit unsigned char as for the first level input). Another method to reduce the computational time and to offer an industrial solution may be to use assembly language filter operations.

Even if the transform is very slow and intended for still images, it might have many applications in current sensor networks, including robotics or space exploration. The transform can be a preprocessing technique for many compression algorithms that can help to reduce network congestion and overall energy. Our future work will also concern the development of a low-complexity version of SPIHT for a picture compression on the camera sensor.

As we already have discussed in section 1.1, the wavelet techniques for signal processing in sensor networks have not yet extensively been discussed in the literature. A reason for this may be that the micro controllers that are generally applied for these networks are regarded to not fulfill the computational and the memory requirements. The fractional wavelet filter shows that more complex signal processing algorithms can be applied to sensor networks and may be a good starting point for future investigations in this field.

6. REFERENCES

[1] J. Chilo and T. Lindblad. Hardware implementation of 1d wavelet transform on an fpga for infrasound signal classification. *IEEE Transactions on Nuclear Science*, 55(1):9–13, Feb. 2008.

[2] C. Chrysafis and A. Ortega. Line-based, reduced memory, wavelet image compression. *IEEE Transactions on Image Processing*, 9(3):378–389, Mar 2000.

[3] K.-C. Hung, Y.-J. Huang, T.-K. Truong, and C.-M. Wang. Fpga implementation for 2d discrete wavelet transform. *Electronics Letters*, 34(7):639–640, Apr 1998.

[4] S. Ismail, A. Salama, and M. Abu-ElYazeed. Fpga implementation of an efficient 3d-wt temporal decomposition algorithm for video compression. *IEEE International Symposium on Signal Processing and Information Technology*, pages 154–159, Dec. 2007.

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

[6] J. Oliver and M. Perez Malumbres. On the design of fast wavelet transform algorithms with low memory requirements. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(2):237–248, Feb. 2008.

[7] S. Rein. *Fixed-Point Arithmetic in C: A Tutorial and an Example on Wavelet Filtering*. Wavelet Application Group, Technische Universität Berlin, 2008. available at www.mdt.tu-berlin.de/research/wavelets.

[8] S. Rein, C. Gühmann, and F. Fitzek. *Mobile Phone*

Programming, chapter Sensor Networks for Distributive Computing, pages 397–409. Springer, 2007.

[9] A. Said and W. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 6, June 1996.

[10] S.Lehmann, S.Rein, and C.Gühmann. External flash filesystem for sensor nodes with sparse resources. submitted to *Mobimedia'08*.

[11] T.W.Fry and S. Hauck. SPIHT image compression on FPGAs. In *IEEE Trans. on Circuits and Systems for Video Technology*, volume 15, Sept. 2005.

[12] B. Usevitch. A tutorial on modern lossy wavelet image compression: Foundations of JPEG2000. In *IEEE Signal Processing Magazine*, Sept. 2001.