SANDIA REPORT
SAND2008-6566
Unlimited Release
Printed December 2008

Application Specific Compression: Final Report

David K. Melgaard, Dan S. Myers, Phillip J. Lewis, David S. Lee, Jeffrey J. Carlson, Raymond H. Byrne, Carol D. Harrison

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

SANDIA REPORT
SAND2008-6566
Unlimited Release
Printed December 2008

Application Specific Compression: Final Report

D. K. Melgaard, D. S. Myers, P. J. Lewis, D. S. Lee, J. J. Carlson,
R. H. Byrne, and C. D. Harrison
Data Analysis & Data Exploitation Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0576

ABSTRACT

With the continuing development of more capable data gathering sensors, comes an increased demand on the bandwidth for transmitting larger quantities of data. To help counteract that trend, a study was undertaken to determine appropriate lossy data compression strategies for minimizing their impact on target detection and characterization. The survey of current compression techniques led us to the conclusion that wavelet compression was well suited for this purpose. Wavelet analysis essentially applies a low-pass and high-pass filter to the data, converting the data into the related coefficients that maintain spatial information as well as frequency information. Wavelet compression is achieved by zeroing the coefficients that pertain to the noise in the signal, i.e. the high frequency, low amplitude portion. This approach is well suited for our goal because it reduces the noise in the signal with only minimal impact on the larger, lower frequency target signatures. The resulting coefficients can then be encoded using lossless techniques with higher compression levels because of the lower entropy and significant number of zeros. No significant signal degradation or difficulties in target characterization or detection were observed or measured when wavelet compression was applied to simulated and real data, even when over 80% of the coefficients were zeroed. While the exact level of compression will be data set dependent, for the data sets we studied, compression factors over 10 were found to be satisfactory where conventional lossless techniques achieved levels of less than 3.

Table of Contents

1 INTRODUCTION

Current threats to our national security include the proliferation of nuclear, chemical, and biological weapons of mass destruction; the growing availability of short-range, mid-range, and strategic missile delivery systems; broader access by foreign powers to space; international shifts of power resulting from multiple regional conflicts in limited or denied access areas; and efforts to deny or deceive collection against critical targets by existing methods. This escalation demands technologies that provide for continuously available situational awareness anywhere in the world at any time or that employ modes/domains not previously conceived. Current remote sensing and verification technologies have been critical to DOE's nonproliferation efforts as well as Department of Homeland Security and national security missions. The development of application-specific compression algorithms is critical for achieving significant gains in the performance of these systems.

Advances in semiconductor technologies have enabled high resolution sensors that can produce very high data rates. Unfortunately, the bandwidth of downlinks cannot be significantly increased. To maintain the same link margin at higher data rates, more radiated power is required. This leads to larger energy storage devices (e.g. batteries), larger collection devices (e.g. solar cells), and larger antennas – all of which may not be feasible given power and space limitations. One approach to reducing the downlink bandwidth is to perform some type of compression on all of the data. There are limitations to achievable lossless compression ratios because of the inherent entropy in the data, so other methods need to be developed to handle the increasing sensor bandwidths. Another alternative is to incorporate more autonomy onboard the satellite so that only appropriate data is collected and forwarded to the ground. The primary goal of this Laboratory Directed Research and Development (LDRD) project was to research potential lossy and lossless compression algorithms, and to evaluate the effects of lossy compression on automated detection algorithms. Lossy techniques can provide better compression ratios than typical lossless techniques and were a prime focus of this work. However, the downlink for the intended application will be composed of multiple data streams, some of which may require lossless compression. Therefore a combination of lossless and lossy techniques will likely be required for real applications. Since a significant portion of the data transmitted is often of little interest, this approach has the potential for tremendous reductions in bandwidth.

A main objective of the Application-Specific Compression LDRD project was to evaluate the feasibility of applying lossy data compression techniques to digitized signals transmitted from key national security assets. Another objective was to develop compression algorithms that are optimized with respect to mission requirements. Our goal was to develop algorithms that provide significant compression while not negatively impacting mission requirements. Currently, lossless Rice compression is used for compressing and transmitting data. The data reduction achieved with Rice compression is on the order of 3:1. Both the resolution and bandwidth requirements of future sensor technologies are expected to dramatically increase. Therefore, it is desirable to employ lossy compression algorithms that can provide data reduction far beyond that achievable with lossless algorithms. The channel capacity required to transmit the vast amounts of digital data extracted from these new sensor technologies make it mandatory to consider the application of lossy techniques.

2 THEORETICAL BACKGROUND

A large number of data compression applications are concerned with minimizing the number of binary digits (bits) used to represent a sampled signal. The benefits of compression arise in the storage and transmission of sampled signals resulting in vast amounts of data produced over short time periods. Compression objectives are to both minimize memory requirements for storage and the bandwidth requirements for transmission. The performance of a compression algorithm is reflected in its ability to compress data, minimize introduced distortions, and in its computational complexity. For transmission purposes, compression techniques are often greatly constrained by real-time requirements. These requirements can severely limit the size and complexity of the compression hardware. In our specific application, the data transmitted is subsequently uncompressed and processed by algorithms for the purpose of target detection and characterization. The data is also inspected by trained analysts for the same purpose. In addition, there exist separate mission requirements for remote sensing applications. It is possible that different missions will require different compression techniques.

Compression algorithms can be separated into lossless and lossy techniques. Lossless algorithms allow the original, uncompressed sampled signal to be exactly reconstructed from its compressed counterpart. Lossy algorithms, on the other hand, are generally only capable of reconstructing an approximation of the actual sampled signal from the compressed counterpart. That is, a decompressed signal will be distorted when compared to its original, uncompressed, counterpart. Acceptable levels of distortion can vary, depending on the specific application. Each compression category involves algorithms that exploit redundancy in the data. Redundancy is related to predictability, randomness, and smoothness in the sampled signal. For example, a sampled signal consisting of equal values for each sample is fully predictable from just one of the sample values. On the other hand, samples digitized from a white-noise process are totally unpredictable. Lossless compression algorithms attempt to represent a sampled signal, $S(n)$, as an array $S_c(m)$ which has no redundancy and from which $S(n)$ can be exactly reconstructed. Lossy algorithms attempt to remove nonessential components from $S(n)$ (e.g., noise or components that are not easily noticed by a human observer) prior to compression and consequently, $S(n)$ can generally only be approximately reconstructed from $S_c(m)$.

Shannon's seminal 1948 paper "A Mathematical Theory of Communication" [1], lays the mathematical foundation for the limits of lossless compression. The concepts and definitions put forth by Shannon on information, channel capacity, and entropy are important in our present work. For data compression, the concept of entropy is probably the most important. The entropy H of a sampled signal, $S(n)$, is defined as follows:

$$H = -K \sum_{i=1}^{n} p_i \log_2 p_i \tag{1}$$

where K is a positive constant (used for a choice of a unit of measure) and p_i is the probability of occurrence of the i^{th} symbol that compose $S(n)$. There are n possible choices for $S(n)$, and since the sum of all probability density functions must equal 1,

$$\sum_{i=1}^{n} p_i = 1 \tag{2}$$

The capacity C of a discrete noiseless channel is given by

$$C = \lim_{T \to \infty} \frac{\log N(T)}{T} \tag{3}$$

where $N(T)$ is the number of allowed signals of duration T.

The importance of entropy is conveyed in one of Shannon's famous theorems: *The Fundamental Theorem for a Noiseless Channel*. This theorem provides a bound on data compressibility. According to the theorem, it is possible to encode, without distortion, a sampled signal with entropy H (bits per symbol) and send it over a noiseless channel with capacity C (bits per second) at an average rate (symbols/second)

$$\text{Average Data Rate}_{max} = \frac{C}{H} - \epsilon \tag{4}$$

where ϵ is arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$. As the entropy of a signal increases, the data rate decreases and it takes longer to send the same amount of data. The main goal of lossless compression is to decrease the entropy of the signal via some form of encoding, and then send the signal over the channel. Equation 4 provides a theoretical maximum bound for how quickly one can send a signal over a channel with capacity C. For lossy compression, additional techniques may be applied that reduce the entropy of the signal even further, and increase the compression ratio at the expense of imperfect reconstruction (e.g. remove noise or high frequency components, quantization of coefficients, etc.).

3 LITERATURE SURVEY

One of the first steps of this effort was to conduct a literature survey of previous work that might relate to compression of multi-spectral sensor data. Efforts that focus on multispectral data, Synthetic Aperture Radar (SAR) data, or astronomical data were of particular interest. In [2], an approach for applying Generalized Laplacian Pyramid-based fusion to multispectral data is outlined. This approach has the largest benefit in reducing the entropy of the transmitted data when there are a large number of multispectral bands and a relatively high correlation between the data in the different bands. An approach for applying integer wavelet transforms to SAR data is described in [4]. Integer wavelet transforms have the advantage of yielding integer coefficients, usually at a slight performance loss compared to discrete wavelet transforms. This performance loss is amplified for SAR images which can have significant high frequency content. A novel approach to improve SAR performance is described, which is a new application of the lifting scheme to generate integer wavelet packets. An interesting comparison between two types of lossy compression applied to remote sensing data appears in [5]. Truncating followed by lossless compression (TLLC) is compared to lossy JPEG compression. The results show that lossy JPEG compression preserves radiometric data better than TLLC for three different data sets (a Landsat image and two images from a multi-spectral sensor).

An example of implementing the Haar* wavelet transform (an integer version of the Haar transform) in hardware is described in [6]. An easy-to-implement wavelet transform similar to the Haar transform, the S-transform, is described in [7]. The implementation only requires integer addition and bit shift operations. The performance of the S-transform is then improved with predictive coding. Progressive resolution transmission is supported by entropy encoding. Results for Huffman and arithmetic coding are described in the paper.

In [8], a new scheme named the contourlet is proposed as an improvement over traditional wavelets for two-dimensional piecewise smooth signals resembling images. The contourlet is based on previous work, the curvelet, which was intended for functions defined in the continuum space $\mathbb{R}^2$. The contourlet represents the development of a discrete version of the curvelet transform that can be applied to sampled images. The performance of the contourlet was found to be slightly better than a traditional 9-7 biorthogonal wavelet transform.

It has long been known heuristically that wavelet transforms, followed by some kind of thresholding of the coefficients, can reduce the amount of noise in the reconstructed data. "De-noising" is the term used to describe the informal schemes which attempt to reject noise by damping or thresholding in the wavelet domain. In [9], the authors prove that for additive Gaussian white noise, a specific type of soft thresholding of the coefficients yields near optimal mean-square error without the ripples or oscillations prevalent in other estimation approaches that only attempt to minimize the mean-square error. The elimination of blips or ripples is especially important for astronomical data where these artifacts may be interpreted as a scientifically significant structure. An approach for de-noising multiplicative noise instead of additive Gaussian white noise is outlined in [19].

Since wavelet coefficients also decompose the frequency content of a signal, the coefficients may be used in various detection schemes. In [10], a scheme for using the wavelet coefficients to detect faults in rotating machinery is described. A biorthogonal 3-9 wavelet transform is applied to data from an acoustic sensor. Four different types of estimators are presented, and de-noising of the coefficients is also employed. The paper claims that this approach is suitable for detecting different types of bearing faults in rotating machinery.

Sending data at a constant rate is important for applications where the bandwidth is limited (e.g. a downlink from a satellite). The compression ratio achieved with lossy wavelet transforms is a function of the image scene, the thresholding and quantization of the wavelet coefficients, and the approach used to encode the coefficients for transmission. A technique for obtaining the best image quality for a given bit rate is described in [11]. Their approach relies on using zerotrees to encode the significance map, which tells whether a sample has a zero or non-zero quantized value. At high compression ratios, a large number of the wavelet coefficients are set to zero (e.g. ~90%). A large fraction of the bit budget must be spent encoding the significance map. The zerotree approach assumes that a coefficient is insignificant if the absolute value of the coefficient is less than some threshold. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant (e.g. less than the threshold), then all wavelet coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant as well. The zerotree approach was developed in recognition of the difficulty of achieving greater bit rate reductions through additional prediction of significant coefficients. Rather, this approach focuses on reducing the cost of encoding the significance map so that for a given bit-rate budget, more bits are available to encode the expensive significant coefficients. This algorithm relies on adaptive arithmetic encoding for lossless encoding of the resulting coefficients.

Many wavelet transforms involve floating point coefficients (which are subject to rounding errors) and are computationally expensive to implement. A technique which addresses these two shortfalls is found in [12]. Any discrete wavelet transform with finite filters can be decomposed into a finite sequence of simple filtering steps, called lifting steps or ladder structures. This decomposition reduces the computational complexity by a factor of two, and allows the definition of wavelet-like transforms that map integers to integers. These integer to integer transforms still often require floating point arithmetic, but at least the resulting coefficients are integers. A more detailed analysis of integer to integer wavelet algorithms appears in [18].

The performance of various wavelet transform compression approaches is of great interest. The error rate of decay between the original image and the compressed image measured by an L^p norm for different wavelets is described in [13]. This paper also looks at the errors that are induced when coefficients are quantized. These authors make the argument that the L^1 norm is a better error measurement than the L^2 norm when looking at the spatial-frequency-intensity response of the human visual system.

A low-power, low-memory implementation of a wavelet compression algorithm for underwater imagery is described in [14]. A Daubechies 5/3 wavelet transform is used because it only uses integer arithmetic and division by two (shifting operations). Furthermore, the algorithm is expressed in lifting form to reduce the computational complexity by a factor of two. The authors also employ bit plane encoding to encode significant (greater than a threshold) coefficients. Results are presented for several underwater images.

Another application of the wavelet transform is to merge data with different spatial and spectral resolution. An approach for merging Landsat (28.5 to 100m resolution) and SPOT (10m resolution) data using a multiresolution wavelet decomposition is presented in [15].

Transmission bit errors are easy to detect, and can be easily corrected if the packets are protected with some type of error-correcting code (e.g. Reed Solomon encoding). There is an overhead penalty associated with adding error correcting codes. An alternative, discussed in [16] is to encode that data in a way that minimizes the effects of random bit errors. This approach is based on a modified embedded zerotree wavelet image compression algorithm.

If the signal to be compressed is known a priori, it is possible to construct an optimal wavelet algorithm that minimizes the upper bound of the L^2 norm of the error in approximating the signal up to the desired scale. Such an algorithm is described in [17].

Another area that was investigated was vector quantization techniques. Vector quantization techniques use a substitution method to compress data. First, data bytes are grouped into vectors of fixed length. These vectors are analyzed to produce the codebook, a relatively small selection of vectors that summarizes the input data; Teuvo Kohonen's self-organizing map is one well-known method of creating a codebook. Each data vector is then compared to the codebook, and the index of the closest matching codebook vector is recorded. The compressed data consists of the codebook vectors and indices. The data is reconstructed by replacing each index with its matching codebook vector. If the codebook vectors are very similar to the original input vectors, data loss will be minimal. Any vector that does not have a close match in the codebook will suffer more significant data loss.

Quantization techniques work well if the distribution of vectors can be adequately represented by the codebook. In general, images with regular backgrounds and minimal variations are very amenable to VQ. Unfortunately, our application-specific data does not meet these conditions. Though the image backgrounds are generally regular with small intensity variations, the targets are unpredictable, and cannot be easily represented by a small codebook. Further, targets generally occur in only one pixel of a vector, making it possible that targets will be modified or destroyed if their vector is quantized. Finally, codebook construction is a non-trivial task; constructing a new codebook for each image is not feasible, and constructing a single large codebook stored at the ground system for all images is also impossible, given the variety of data collected by the system.

A review of various standards and the approaches used for compression appears below:

JPEG: First developed by the Joint Photographic Expert's Group in 1991. Image data has its mean value removed and is then subjected to an 8x8 discrete cosine transform, followed by scaling and uniform quantization according to predefined tables. The mean value component is calculated block to block and the 63 other coefficients are then Huffman coded in amplitude and location along a zig-zag path running across the coefficient array from top left (DC term) to bottom right (highest frequencies) [3]. A disadvantage to this approach is that at low data rates block-structured artifacts appear in the reproduced image. Lossless coding results in 2 or 3:1 bit-rate reduction.

JPEG2000: switches to a wavelet transform instead of the DCT.

MPEG-I: The basic coding algorithm is a combination of motion-compensated inter-frame prediction and intra-frame transform-coding. The 8x8 blocks are combined into macro-blocks consisting of four 8x8 luminance blocks. Motion compensation is carried out over 16x16 luminance blocks [3].

MPEG-II: Uses the same basic algorithm as MPEG-I, except modified to handle higher data rates and different inputs. Added five profiles (from simple to high), and four levels (from low to high) to handle the large range of parameters available [3].

The next section discusses the areas that were the focus of the internal research and development effort.

4 OVERVIEW OF LDRD EFFORTS

Initial efforts focused on experimenting with unclassified data sets and using JPEG as a baseline approach while a parallel literature search was being conducted. The studies with JPEG are documented in section 6. Metrics were developed early on to compare the performance of different approaches. The metrics are discussed in more detail in section 5. Experiments with JPEG showed that at higher compression levels, the unwanted blocky artifacts started to cause problems for the detection algorithms manifested as an increased number of false alarms. Our literature survey had found that Haar wavelets are commonly used for astronomical data, which sometimes closely matches our data sets. In addition our initial studies on the effectiveness of the various wavelet indicated that the Haar wavelet performed well with our data. One attractive feature is that the Haar affects only a 2x2 pixel region, so it can be more adaptive to smaller features, then the other wavelets which utilize more points. Consequently since it could be implemented completely with integer arithmetic, the majority of the subsequent effort focused on deeper understanding of the performance of Haar wavelets on classified data sets. A separate classified report contains a case study of these results. The Haar wavelet algorithm was also prototyped in hardware to explore implementation issues. Implementation results are discussed in section 9. Additional efforts revisited all types of wavelets and their performance with low-level simulated targets superimposed on real sensor data. These results are summarized in section 8, and the complete results appear in [31]. Research and experiments were also conducted on various approaches for encoding the wavelet coefficients. These results are summarized in section 10.

5 PERFORMANCE METRICS

The metrics of greatest utility in determining the effect of lossy compression on mission data are those which characterize the impact upon target detection and extraction. As such, there are three different areas of interest which serve to assess this impact: probability of detection (Pd), false alarm rate (FAR), target extraction fidelity, and real-time human visual exploitation of frame data.

Lossy compression could increase the false alarm rate and decrease the probability of detection. Receiver operating characteristic (ROC) curves are typically used to quantify this impact. Data cubes necessary to build ROC curves would need a multitude of targets (e.g., hundreds) with a range of intensity values, spatially distributed across the scenes variation. The spatial distribution is necessary since Pd is a function of target location in the scene. In actuality, the preponderance of collected data cubes typically only has a small number of targets (e.g., 1-3). Consequently ROC curves from collected data (if the true events are known) can be misleading and statistically insignificant, but those built from simulated data or simulated targets inserted in real background data can be more useful if properly constructed. FAR-versus-detector-threshold curves are straightforward to create, but they do not address target detectability. To quantify the impact of various wavelet algorithms on small targets, ROC curves were generated for simulated targets superimposed on real sensor data. These results are summarized in section 10 and are described in more detail in [31]. Those generated using classified data can be found in the related classified report.

Acceptable degradation to target extraction fidelity and signature characteristics is low. Target intensity and location time-histories are used by analysts to discriminate between actual and false events and to characterize target phenomenology. The peak difference, max difference, root mean square error (RMSE), and peak signal-to-noise ratio (PSNR) are all of interest in analyzing extracted target data. Peak difference refers to the maximum difference between the peaks on the uncompressed extracted event and the compressed extracted event. Max difference, meanwhile, refers to the maximum difference at any specific temporal data point along the curve between the uncompressed and compressed extracted events. These are key metrics in determining target phenomenology, and thus great variation in peak or max difference would be unacceptable. The RMSE and PSNR are standard metrics used in compression literature.

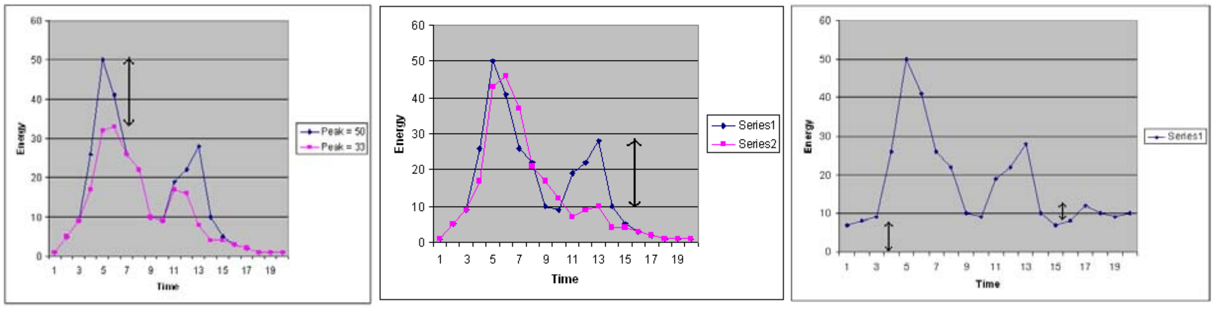

Figure 1: Examples of Metrics, Peak Difference (left), Max Difference (center), and Baseline Sigma/Elevation (right)

Analysts visually monitor real-time data for targets that may be missed by autonomous detection/extraction software. Data artifacts due to lossy compression can be distracting and impact visual exploitation efficacy (see discussion about Figure 8). Therefore, one consideration was the qualitative evaluation of the impact of data artifacts. This was accomplished by having subject matter experts review the compressed data.

6 SIMULATED DATA, JPEG COMPRESSION

Preliminary experimental tests were performed on an unclassified data set containing three simulated targets – two static and one dynamic. The test set consists of frames produced for a visible spectrum sensor. A top-hat function was used to generate the simulated targets. Though the target signatures are highly unrealistic, these simulations provide some insight into the effects of compression on target detection.

The performance of the detector was evaluated for JPEG quality factors of 100% (lossless compression) and 30%. The frame cube was first compressed using JPEG with the specified quality factor, then decompressed and analyzed using the detector. Detector thresholds were in the range of [10, 200], with increments of 10. In the terminology of the detector, a *trigger* occurs whenever a pixel in the input frame exceeds the detector threshold. Subsequent stages of the pipeline apply filtering to remove popcorn noise and other artifacts. A trigger is considered a *detection* after it passes a certain basic level of noise filtering. If the detection also passes tests for spatial and temporal correlation, it may be deemed an *event*, the ultimate output of the detector pipeline. Due to the unrealistic nature of the simulated targets, it was not productive to analyze the final events produced by the detector for this data set. For each threshold, two parameters: were calculated: the mean number of triggers per frame and the mean number of detections per frame.

Both of these metrics provide insight into the effect of compression artifacts on the detection software. If the number of triggers or detections is high, the probability of producing false events increases creating a higher processor load. Figure 2 shows plots for the mean number of triggers per frame for both JPEG quality factors. Figure 3 shows plots of the mean number of detections per frame.

Performance at low thresholds (less than 50) is clearly enhanced when using the higher quality factor. As the quality factor decreases, the JPEG artifacts begin to cause additional triggers and detections (false alarms). The nature of the blocky artifacts present with low JPEG quality factors was one of the motivations for exploring wavelet algorithms. Since JPEG is also optimized for visual consumption, these artifacts become even more of an issue with multi-spectral sensor data.

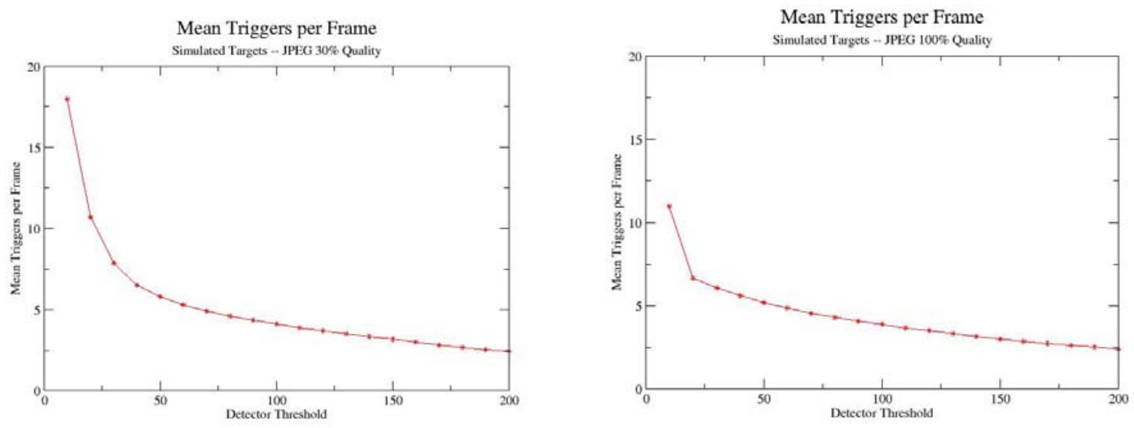

Figure 2: Mean Triggers per Frame for JPEG Quality Factors 30% and 100%

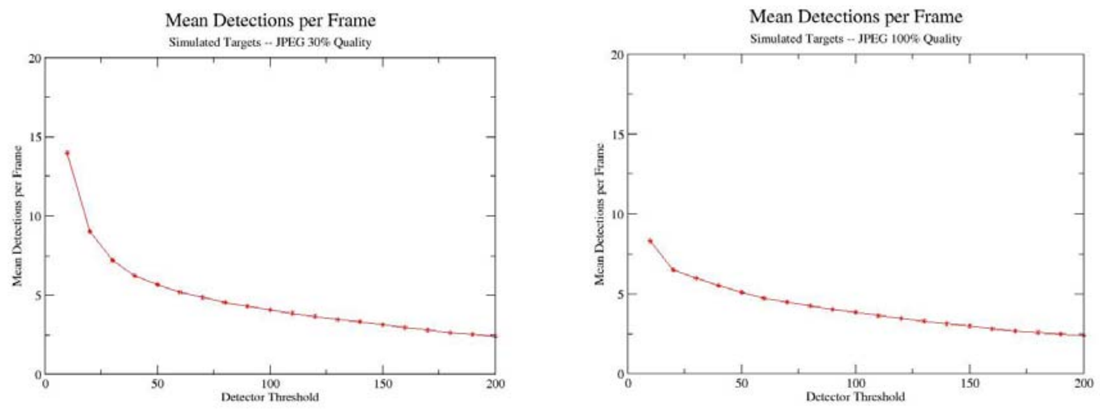

Figure 3: Mean Detections per Frame for JPEG Quality Factors 30% and 100%

Additional experiments were performed to explore the benefits of pre-processing sensor data prior to JPEG compression. Once again, unclassified data sets were used. Three different cases were tested:

- No pre-processing prior to JPEG compression
- Temporal smoothing prior to JPEG compression
- Temporal and spatial smoothing prior to JPEG compression

The results for these tests are shown in Figures 4, 5, and 6. Significant compression gains can be achieved by pre-processing the data. However the data quality was not adequate for our purposes.

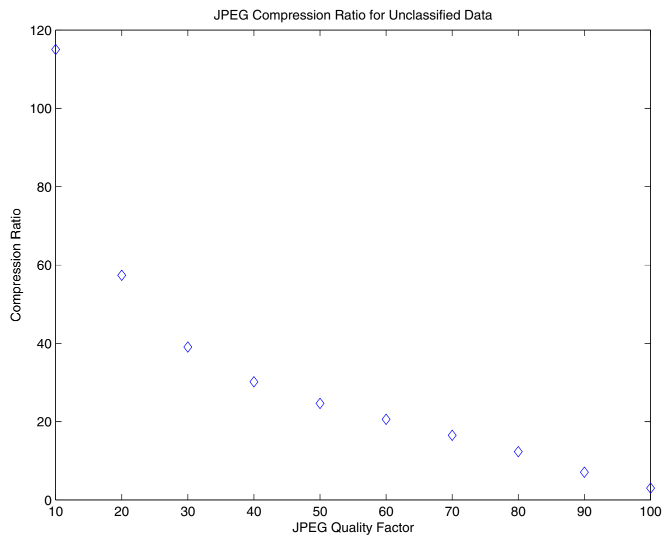

Figure 4: No Processing Prior to JPEG Compression

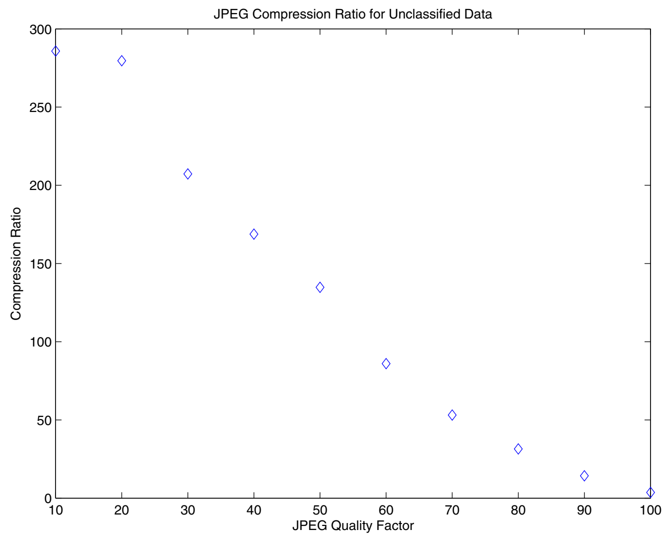

Figure 5: Temporal Smoothing Prior to JPEG Compression

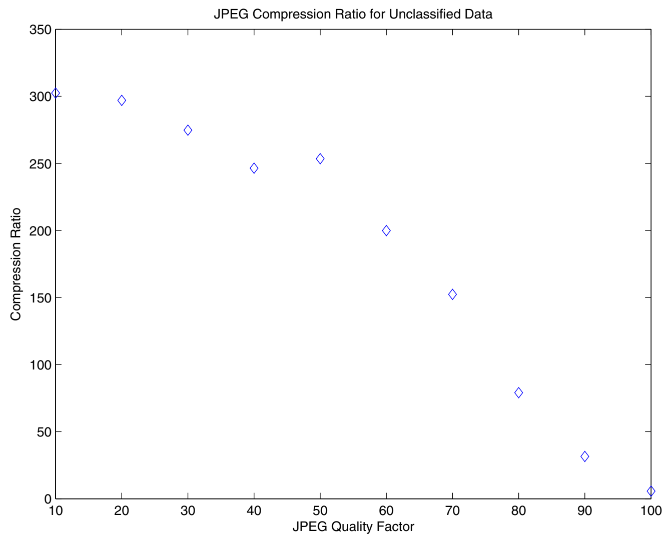

Figure 6: Spatial and Temporal Smoothing Prior to JPEG Compression

7 WAVELET COMPRESSION

After preliminary experiments with JPEG identified the undesirable effects caused by the discrete cosine transformation (DCT), the majority of the efforts focused on the wavelet transformation. Wavelet analysis is a popular and well-developed analytical tool, with widespread use in both signal and image processing. Compression is one of the many applications of wavelet processing, and wavelet-based methods are used in a number of compression algorithms, such as JPEG2000 [25]. A great deal of work has been done to understand the effect of wavelet compression on the visual interpretation, but we focused on automatic target detection from remotely sensed data, where the targets are generally single-pixel point sources modulated by the optics of the sensing system. For these applications, wavelet compression has a number of desirable properties; particularly its ability to preserve sharp transient features. However with increased compression those features can be distorted. This study examines the effects of using different wavelets for compressing data of this type and the amount of compression that is possible without affecting the target features. Results show that compression with an appropriate wavelet can increase signal-to-noise ratio and decrease false alarms, but will adversely affect probability of detection if the level of compression is high enough to significantly reduce the peak intensities.

Wavelets (from the French *ondelette*, “small wave”) are wave forms with compact support and a net sum of zero. Some examples of wavelets are shown in Figure 7. Wavelet analysis uses scaling and translation of wavelet functions to construct a basis set for a signal, capturing information that is localized in both space and frequency. The output of a discrete wavelet transform is actually the combination of high and low-pass filters applied to the image, providing an approximation (the low pass components) and details (the high frequency components) to describe the image. In multiresolution analysis, the approximation values are themselves recursively filtered, creating a set of approximation coefficients and detail coefficients capturing the structure of the image at different scales [26]. The process can be reversed to reconstruct the original image from the coefficients. Performed in this way, the discrete wavelet transform is lossless, but does not actually reduce the size of the data. However if the wavelet is well suited to the data, the entropy of the coefficient may be less than the entropy in the data, and the compression of the coefficients can achieve a higher compression ratio than compressing the data directly. In our study, we saw an increase in the compression ratio on the Haar wavelet coefficients over the frame data.

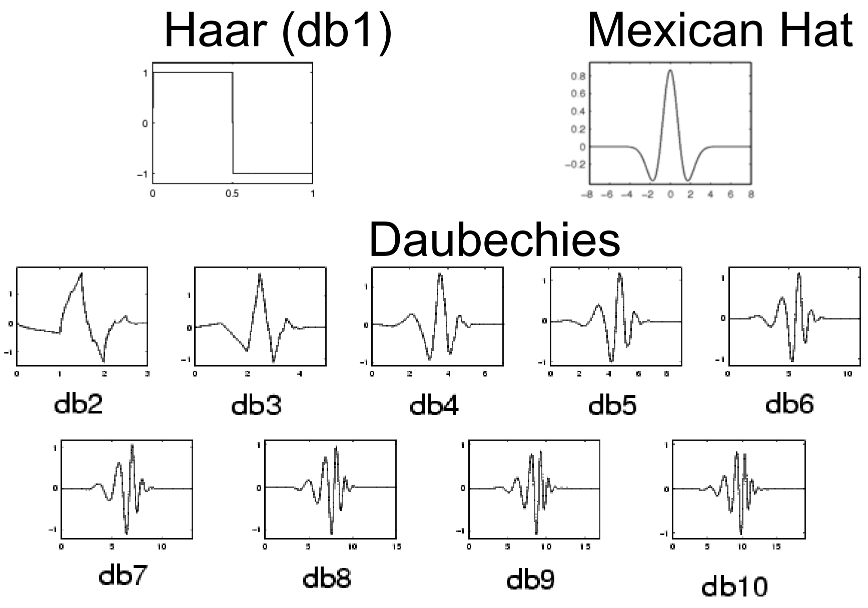

Figure 7: Example wavelets. Haar is also known as Daubechies 1.

To perform wavelet compression, first construct a discrete multiresolution analysis of the image. At that point there are at least two option for compression; 1) bit plane encoding which is useful for constant bit rate encoding and 2) zeroing the smallest detail coefficients and then taking advantage of the zeros to compress the coefficients. For our study we chose the second method but most of the conclusions in this report would apply to the other method. In many applications, including our experiments, we show the percentage of zeroed coefficients. That value can be approximately related to the compression ratios shown in Figure 25. Because the wavelet transform localizes coefficients in both space and time, compressing the coefficients does a better job of preserving high frequency transients than similar methods, such as the discrete cosine transform. For a more thorough discussion of the theory of wavelets see [27, 28, 29].

7.1 Wavelet Choice

A large number of wavelets have been developed so we first studied the impact of the various wavelets provided by Matlab as well as the Cohen-Daubechies-Feauveau 9/7, which has been utilized by the Consultative Committee for Space Data Systems (CCSDS). Since any of the wavelets could be used for the analysis and compression, the objective of the choice is to select the wavelet or wavelets which provided for the best compression with minimal impact on the target intensities. To evaluate the wavelets, we used 1) the entropy of the coefficients and 2) the target intensity difference.

While no wavelet consistently or significantly outperformed the others; the Haar, Biorthogonal 1.3, Daubechies 2, Symlets 3, Coiflets 3 and Reverse Biorthogonal 2.2 seemed to generally perform the best. Since the Haar was the choice selected by other researchers for compressing point source data and it was computationally simple, we focused primarily on it.

7.2 Frame Data Compression

A Matlab program was developed to examine the impact of wavelet compression on the frame data. The program read in data files in either Epro or HDF5 format and removed the estimated background from the data. It also provided options to select any of 48 wavelets, the number of levels of decomposition and compression threshold. The display showed the uncompressed and compressed images. In addition, the user could select an area of interest (AOI) to be displayed. If the AOI was displayed, the 3D intensity plots were also shown. The program could operate in run mode where the frames were displayed continuously running forward in time or backwards or could be stepped forward or backward to observe the impact of the compression on the data in greater detail. The figures in this section were taken from those displays and illustrate their utility.

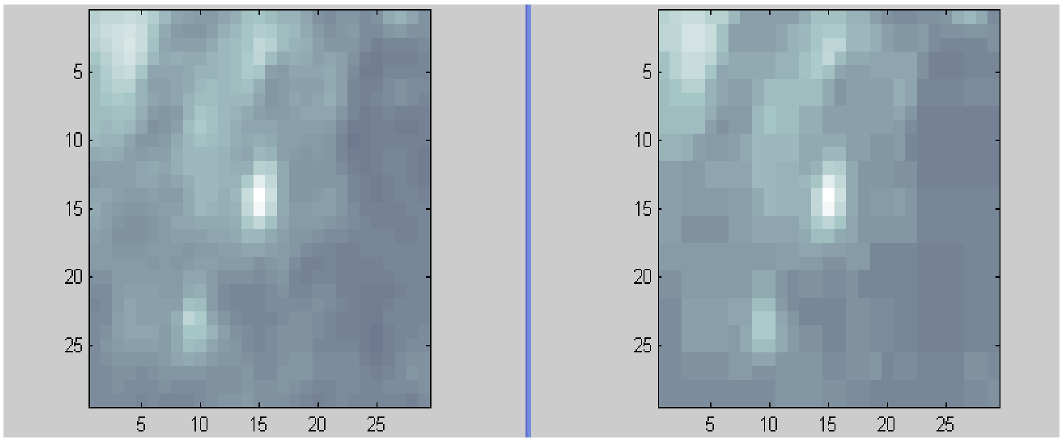

Figure 8: Uncompressed (left) and compressed (right) with 77% zeroed coefficients 30x30 pixel image illustrating blocky effect of compression.

After the wavelet analysis converted the frame data into wavelet coefficients, the small magnitude coefficients for the high frequency components were set to zero. The impact of this operation may cause the image to appear blockier in the areas of lower intensity. Note that this blocky effect is not the same as with the DCT. The DCT caused bright edges to appear which were leading to false alarms. The wavelet compression blocky effects do not increase false alarms but only the visual appearance. Figure 8 provides an enlarged portion of an image to illustrate this behavior. In the figure, the effect is relatively minor even with 77% zeros. That was also the case in the other data sets that we observed. Our primary concern was the impact on the target intensities. The 3-D display of the intensity values (displayed in Figure 9) show the impact was minimal.

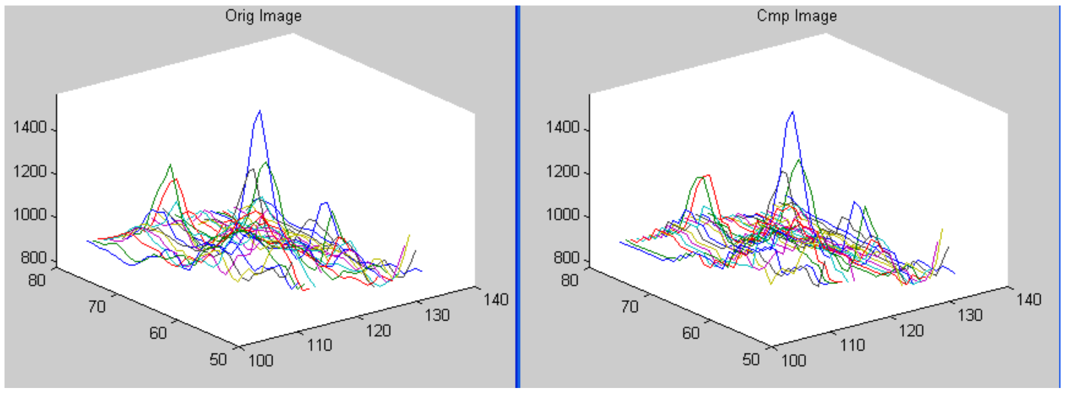

Figure 9: Uncompressed (left) and compressed (right) pixel intensities show differences only at the lower levels using a threshold of 50 resulting in 77% zeroed coefficients.

To understand the impact in greater detail we included a plot to show the target pixels. When the program was in AOI mode and single stepped through the frames, it would select the largest 5% of the pixels in the AOI, and display the uncompressed values in sorted order with the corresponding compressed values. The difference was also shown. Example results are shown in Figure 10. The plots illustrated the selective nature of wavelet compression where the lower value pixels showed larger differences on the order of the compression threshold (45 in this case) and the larger intensity pixels showed smaller differences. The figure also shows the bias that typically reduces the magnitude of the smaller pixels thereby increasing the SNR.

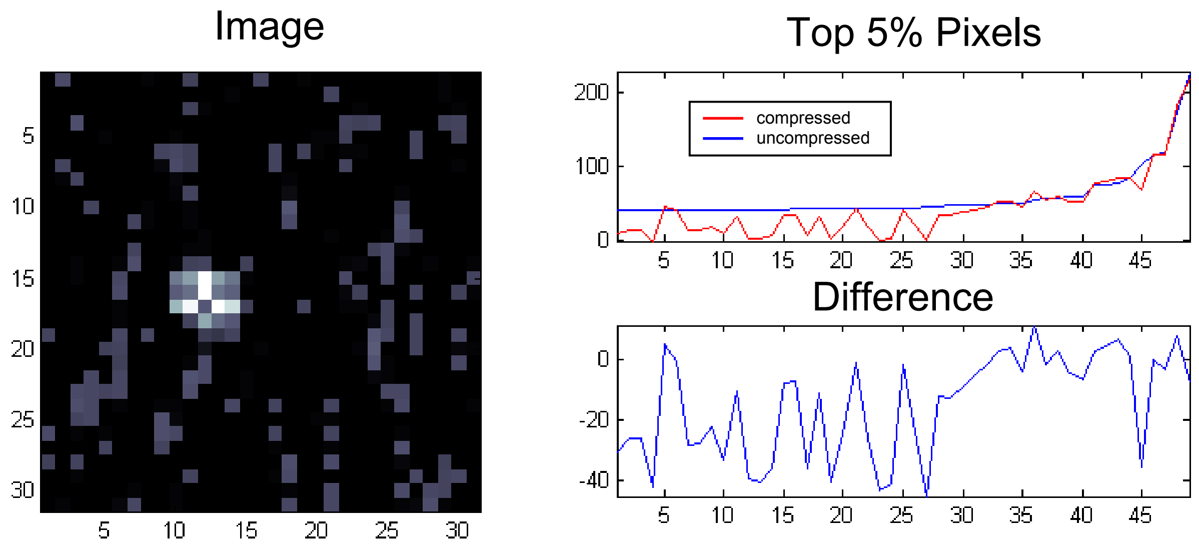

Figure 10: Image AOI with target on the left. The top 5% pixels for the uncompressed and the corresponding compressed values from a threshold of 45 on the right with the difference.

We employed the same program to compress real data using the Haar wavelet with 2 levels of decomposition and compression levels of approximately 60%, 70%, 80% and 90%. The data was then reconstructed and passed through our processing stream for evaluation. Usually there was no detectable difference between the compressed and the uncompressed data streams even at the higher compression levels. As expected, with dim targets the differences were more pronounced (particularly at the higher levels0 but at no time were the changes outside the tolerances used to evaluate targets and their detections. We refer you to the classified report for details.

8 SIMULATION RESULTS FOR SMALL TARGETS

In a parallel study, we examined detection of simulated targets to better understand the compression. We performed a study using simulated targets superimposed on real sensor data. Of particular interest was the study on dim targets with intensities close to the noise level. In the real sensor data we obtained, there were numerous types of features and types of noise, some of which directly affected our results. That would not have been the case had we simulated the data. Complete results appear in [31] and a summary of the main results are discussed in this section.

8.1 Data Frames and Simulated Targets

Because using simulated noise can lead to errors in analyzing the effects of compression, we collected real sensor data and then superimposed simulated targets on the frames. The frames contain characteristics of real sensors that are difficult to simulate (e.g. correlated noise, artifacts, and non-uniform focal plane responses) while providing control over the target size, shape, and duration required for our analysis. One hundred frames of data were taken from a large format sensor. A 256x256 region was extracted for use in further experiments. The simulated targets are single-pixel point sources, modulated by the point-spread of a hypothetical optical system, described by the Airy pattern for a circular aperture [32],

$$I(R) = \frac{1}{(1-\epsilon^2)^2}\left[\frac{2J_1(x)}{x} - \frac{2\epsilon J_1(\epsilon x)}{x}\right]^2. \tag{1}$$

ϵ is the annular aperture obscuration ratio $(0 \leq \epsilon < 1)$ and x is equal to

$$x = \frac{\pi R}{\lambda N}, \tag{2}$$

where R is the radial distance in the focal plane from the optical axis, $N$is the *f-number* of the optical system, and λ is the wavelength of light. To generate the targets, we assumed the parameters shown in Table 1.

The intensity of targets as a function of time was modeled by a decaying exponential, with an initial intensity of I_0. For a target that appears at time t_0, the intensity is given by

$$I(t) = \begin{cases} 0 & , \quad t < t_0 \\ I_0 e^{-(t-t_0)\Delta} & , \quad t \geq t_0 \end{cases} \tag{3}$$

where Δ is the time decay constant for the target.

Parameter	**Value**
λ	$1000nm$
f-number, *N*	5.0
Pixel spacing	$18u$
Obscuration ratio, ϵ	0

Table 1. Parameters for simulated targets

8.2 Target Detection Algorithm

This section describes the basic target detection algorithm used in all subsequent experiments. The detector has three main components: per-pixel noise estimation, signal-to-noise ratio calculation, and comparison to an ideal point-spread function.

The noise estimator computes an approximation of the noise present in each focal plane pixel. For this application, a single global noise estimate would be inappropriate, as some focal plane pixels are significantly noisier than average. Let *F* be the set of *N* total frames, and F_p^i be the value of pixel *p* recorded in the ith frame of *F*. The variance of the noise in *p* is estimated as

$$\sigma_p^2 = \frac{\sum_{i=2}^{N} \left(F_p^i - F_p^{i-1}\right)^2}{N-1}, \tag{4}$$

that is, the average squared difference in the values of *p* between consecutive frames. These noise estimates are computed as a pre-processing step on the set of frames before proceeding with the other target detecting steps.

After the noise estimation is complete, the target detector checks each pixel in the current frame, calculating intensity relative to noise and the shape of the energy surrounding the pixel. If the signal-to-noise ratio is sufficiently high and the energy shape approximates a point-spread function, the pixel is marked as a target.

The first step is a simple intensity threshold. If the intensity of pixel *p* is below the threshold, it is dismissed and no further processing takes place. If *p* passes the threshold test, the signal-to-noise ratio is computed as

$$SNR_p = \sqrt{\frac{\sum_{h \in H} F_h^2}{\sum_{h \in H} \sigma_h^2}}, \tag{5}$$

where *H* is the 3x3 neighborhood centered around and including *p*, and *F* is the frame currently being examined. If the SNR falls below a set threshold, the pixel is rejected.

For pixels with sufficient SNR, a final test ensures that the energy signature surrounding the pixel approximates a point spread function. First, compute the total energy in the 3x3 neighborhood,

$$E_p = \sum_{h \in H} F_h \ . \tag{6}$$

Next, compute the *area ratio*, a measurement of how the target energy is dispersed around the central pixel:

$$A_p = \frac{E_p}{F_p} \ . \tag{7}$$

For the Airy function used in these experiments, $A \approx 2.6$. Finally, compute the *time ratio*, a measurement of how quickly the target energy decays in the next frame:

$$T_p = \frac{E_p^{f+1}}{E_p^f} \ , \tag{8}$$

where E_p^f is the energy in the current frame *f* associated with pixel *p*. For these experiments, the ideal value of T_p is approximately .8. If A_p and T_p lie within set limits, then *p* is accepted and marked as a target. The empirically derived detector parameters are shown in Table 2.

Parameter	**Value**
$Threshold_{Intensity}$	2000
$Threshold_{SNR}$	1.25
A_{min}	1.0
A_{max}	4.0
T_{min}	.5
T_{max}	1.25

Table 2. Target detector settings.

8.3 Simulation Results

Because wavelet compression changes the intensities of individual pixels, it may have a serious effect on pixel-based target detectors. The effect of compression depends on both the wavelet used and the compression threshold. To assess the performance of different wavelets, the target detector described in Section 8.2 was tested against a data frame containing fifteen low-intensity targets. The target intensity was fixed at 3000; the mean uncompressed SNR was 1.87. Logically, any effect of wavelet compression will be most prominent on small targets resting just above the sensor noise floor. The noise estimates required by the detector were computed from a collection of one hundred consecutive frames.

The list of wavelets used in the experiment is shown in Table 3. Implementations were provided by the MATLAB® Wavelet Toolbox [7]. For each wavelet, the target frame was compressed with thresholds ranging from 0 to 1800, with increments of 100. With targets with an intensity of 3000, this range was large enough to show the performance of all the wavelets. The percentage of zeros produced by compression, the probability of successful target detection, and the false alarm rate were recorded for each threshold. Figure 11 shows plots of the probability of detection and figure 12 the false alarm rate vs. percentage of zeros for a representative subset of the wavelets. The probability of detection remains fairly constant and high up to about 70% zeros for all the wavelets. Then the probability for some wavelets begins to drop while for others it remains high up to about 80% zeros. The higher the percentage of zeros, the higher the compression ratio as discussed in section 10.4, so this difference can be significant. Note that the probability is always below 1. This indicates how dim the targets were compared to the noise in the frames. Two of the targets were affected by high noise values in close proximity and consequently were usually not detected. Of the criteria used in detection described in section 8.2; intensity threshold, SNR ratio and area ratio, the primary detection failure was due to low intensity. A significant part of the reason for this can be seen in figures 13 and 14. Figure 13 shows that for most of the wavelet types shown, the SNR increased with more zeroed coefficients. This pattern is to be expected as the first coefficients that are zeroed are related to the smallest high frequency components of the data that is primarily noise. Consequently the noise is being reduced while the target intensity largely remains the same or is reduced to a lesser degree. This effect was discussed earlier in section 7.2 and shown in Figure 10. For the simulated data, the general affect can be seen in Figure 14 where the mean intensity of all the targets is displayed. Those wavelet types that impact the intensity the most (db10 and sym8) show the lowest SNR values (Figure 13).

Wavelet	**Abbreviation**
Haar	haar
Daubechies 2,5,10	db2, db5, db10
5th, 8th order Symlets	sym5, sym8
2.4, 3.7, 4.4,6.4 Biorthogonal	bior2.4, bior3.7, bior4.4, bior6.8
2.4, 4.4, 5.5 Reverse Biorthogonal	rbio2.4, rbio4.4, rbio5.5
Cohen-Daubechies-Feauveau 9/7	CDF9.7

Table 3. Wavelet names and abbreviations

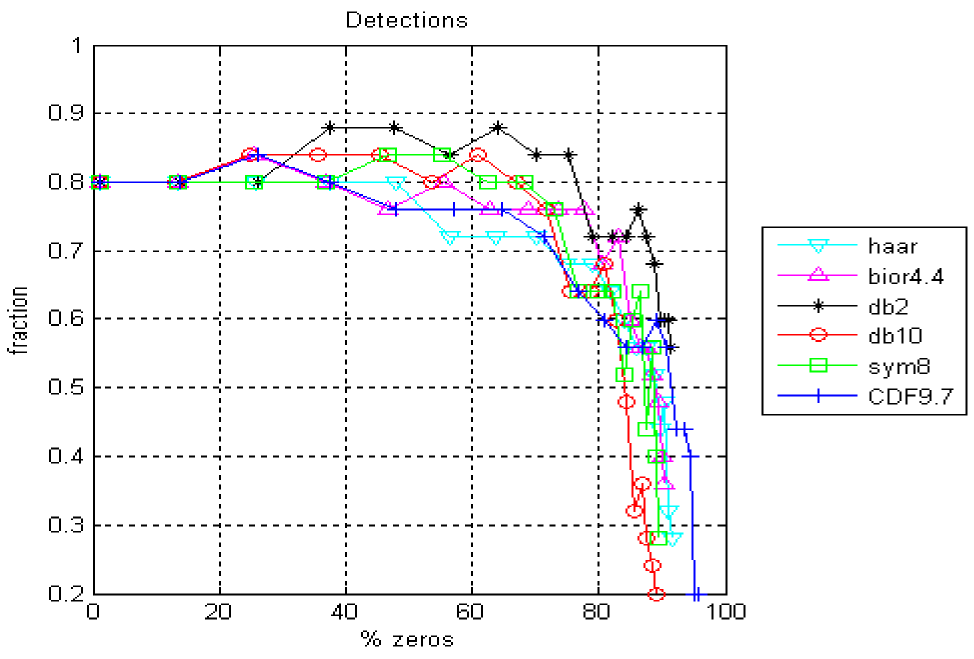

Figure 11: Probability of detections versus the % zero coefficients for a sample of the wavelets

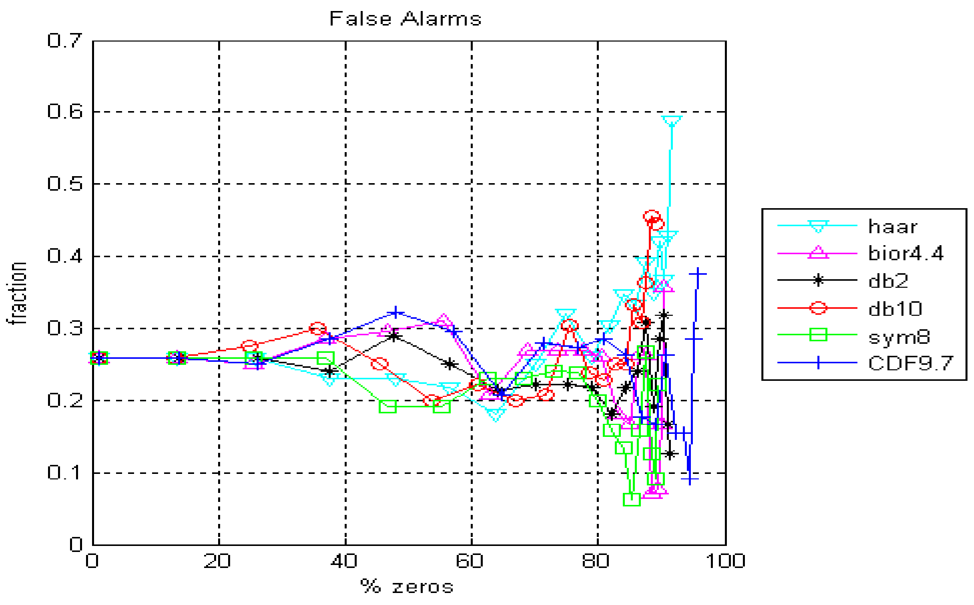

Figure 12: Probability of false alarms versus the % zero coefficients for a sample of the wavelets

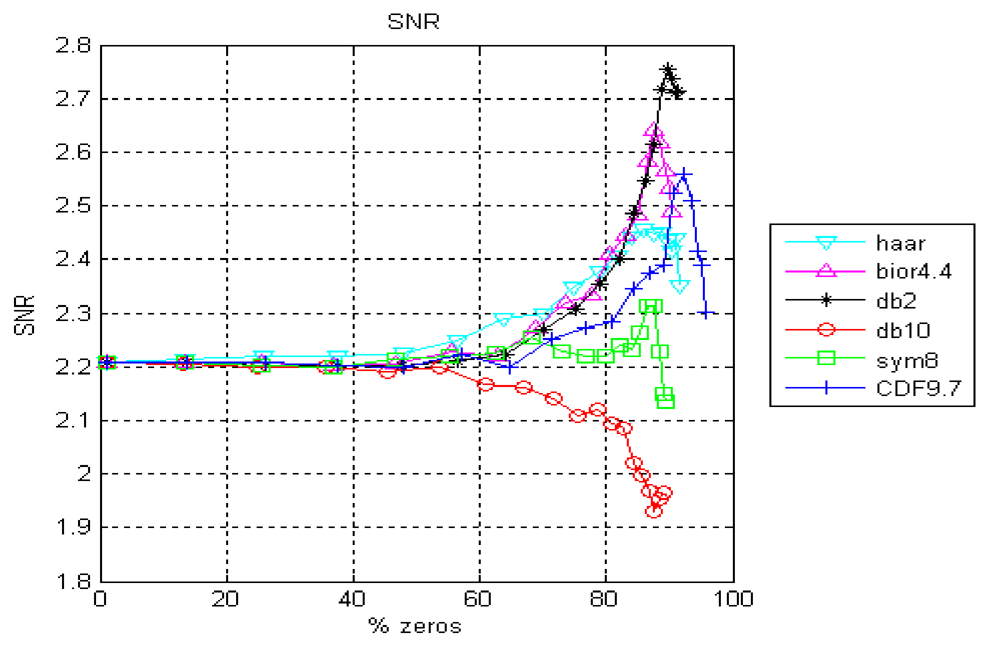

Figure 13: Signal to noise ratio versus the % zero coefficients for a sample of the wavelets

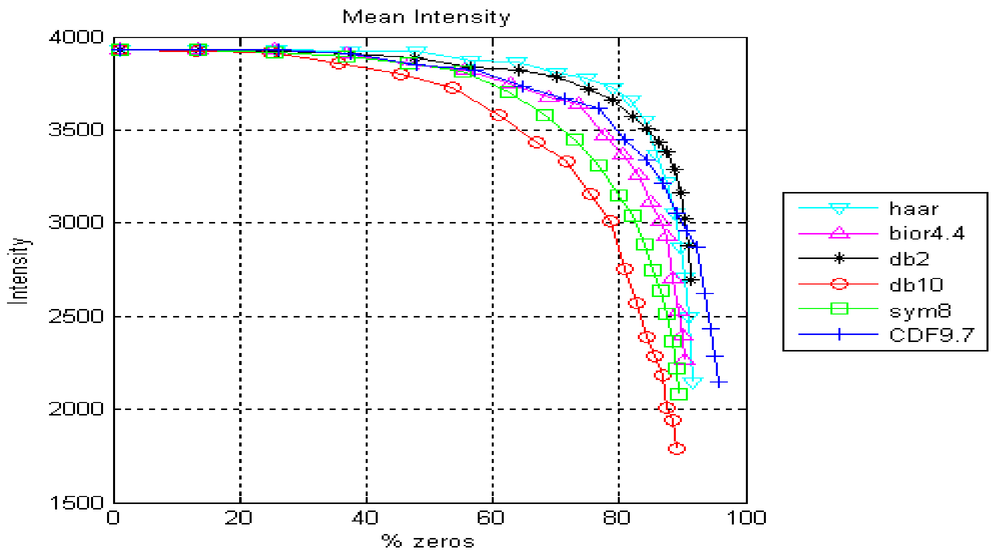

Figure 14: Mean Target Intensity versus the % zero coefficients for a sample of the wavelets

8.4 Simulation Conclusion

These experiments show the effect of wavelet compression on single-pixel targets modulated by an optical point-spread function. The experiments make use of real frame data collected from a sensor, thereby capturing sensor noise characteristics that are difficult to simulate. The results of the basic target detector show that the Daubechies 2 (db2) wavelet performs best for this type of data. It maintained a probability of detection better with more zeroed coefficients while keeping the false alarms rate low. It also showed the best SNR values. Compressing with more complex wavelets did not improve performance which was probably due to the relatively simple target shape. There are however a couple of caveats to be considered with this study. While the Daubechies 2 wavelet was best in this study, the shape of the target in real data may be different and consequently another form of wavelet may perform better. Also these were very dim targets compared to the noise level , so if the goal is to process larger targets it may be satisfactory to use the Haar wavelet which can compress using integer arithmetic, while the Daubechies 2 requires floating point operations. Still the improvements in SNR are worth noting.

Because wavelet compression generally decreases pixel intensities, it can decrease probability of detection when target pixel intensities drop below detector thresholds. However it reduces noise even more, resulting in an increased target signal-to-noise ratio. This suggests that an adaptive threshold based on increasing compression may be more appropriate for wavelet compressed data. With a properly designed detector, wavelet compression may actually enhance detection of low-SNR targets. Investigating this behavior will be a major focus of future study.

9 HARDWARE IMPLEMENTATION OF THE HAAR WAVELET

Compression of wavelet coefficients shows great potential for effective reduction of sensor data without compromising target detection abilities. Through simulation and software analysis, the compression achieved using the Haar wavelet demonstrated the capability to significantly reduce the data size, while maintaining confidence in target detection comparable to current systems. The goal of this portion of the LDRD study was to determine the feasibility of implementing the Haar wavelet in hardware.

The implementation of Haar is conceptually very straightforward. The Haar algorithm uses only integer operations, which sets it apart from most other wavelet algorithms that rely on floating point operations. Furthermore, the Haar algorithm can be implemented using only addition and subtraction, resulting in fast, resource-efficient operations.

The Haar algorithm processes 2x2 regions of pixel data and generates four values from those pixels: a vertical coefficient, a horizontal coefficient, a diagonal coefficient, and an approximation value (V, H, D and A respectively in Figure 14). When processing an entire image, corresponding coefficients are grouped together to keep all horizontal coefficients together, as well as all vertical coefficients, all diagonal coefficients, and all approximation values together.

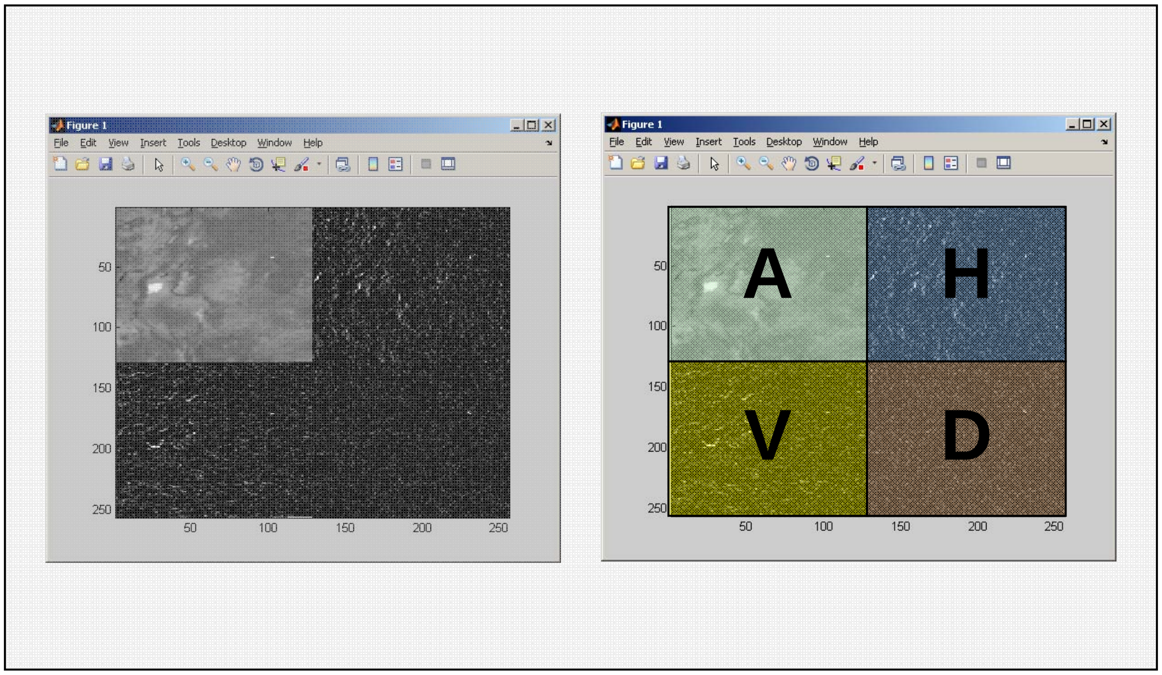

Figure 15: Haar generated coefficients (left) and regions (right).

There are numerous ways to generate the Haar coefficients from an input data stream. If memory is not an issue, the simplest implementation would buffer an entire image, iterate through the buffer reading out the appropriate pixels, and generate the coefficients into a new buffer. While neither optimal in time nor resource utilization, we chose this method since it allowed us to explore each step of the Haar algorithm individually and view the results after each operation.

The first iteration of our implementation took pairs of adjacent pixels on the same line and generated high-pass and low-pass filtered components (see Figure 16). This process involved sweeping through all pairs of adjacent pixels in the same row (pixel 1 and pixel 2; pixel 3 and pixel 4; etc.) and performing an addition and a subtraction operation on those operands. The addition of those values results in the generation of a low-pass component, and the subtraction yields the generation of a high-pass component.

The second iteration of our implementation, shown in Figure 16, takes these generated low-pass and high-pass components and examines pairs of adjacent pixels in the same column. The addition of adjacent low-pass values yields an approximation pixel, while the subtraction of these values yields a horizontal coefficient. The addition of adjacent high-pass values yields a vertical component, and the subtraction of those values yields a diagonal component. All pairs of pixels in the same column are processed to generate all the Haar coefficients.

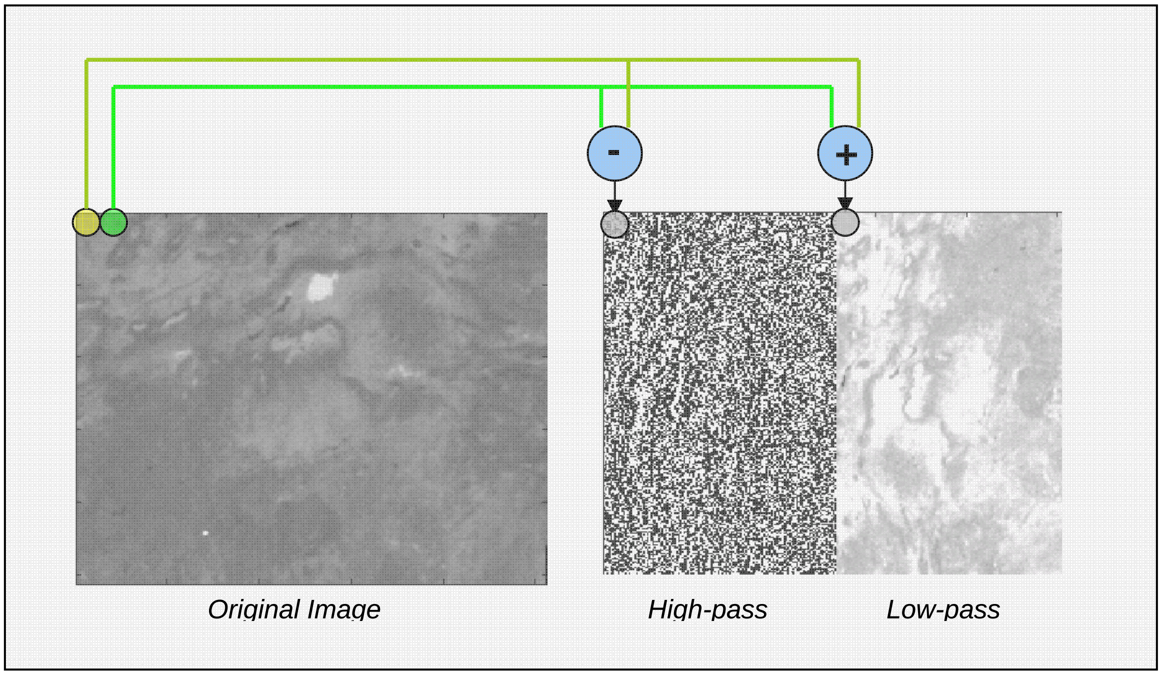

Figure 16: First iterative pass to generate high-pass and low-pass components. The "noise" in the high-pass section is an artifact of representing negative numbers in Matlab.

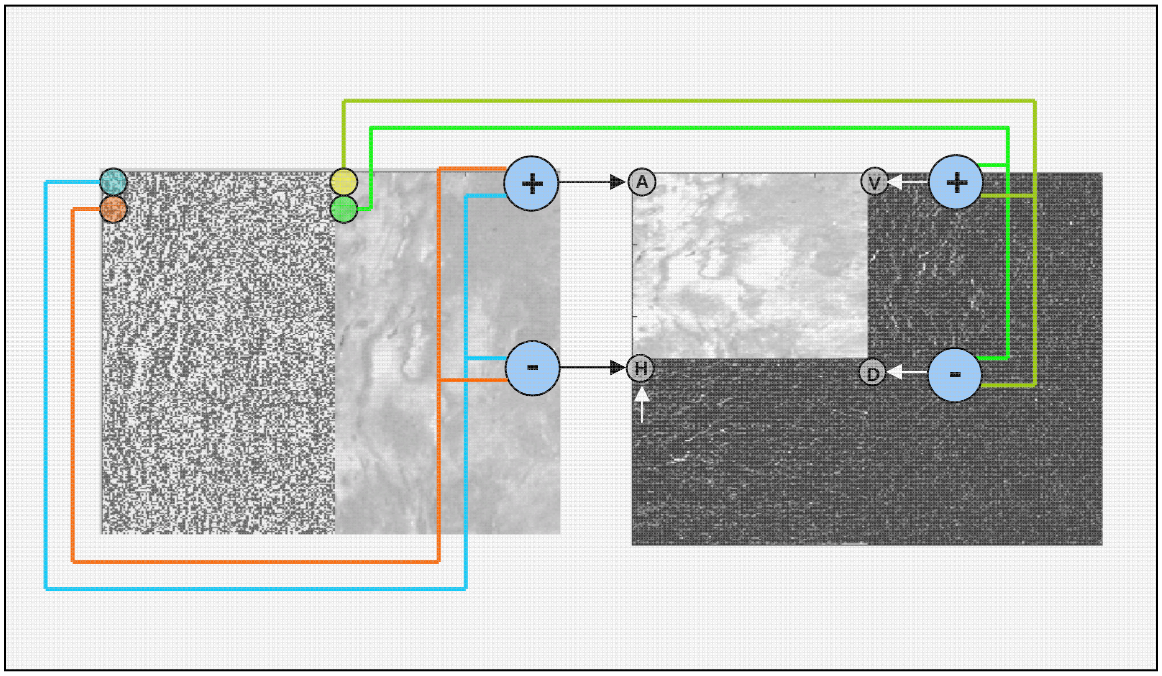

Figure 17: Second iteration which takes high-pass and low-pass components to generate Haar coefficients.

9.1 System Implementation

The hardware utilized in the implementation of the Haar coefficient generation involved three major components:

1. A PC with a gigabit Ethernet interface for transmitting data and a CameraLink interface for receiving data,
2. A Xilinx ML325 Hardware Prototype board with a Virtex-II Pro FPGA (XC2VP70), and
3. A CameraLink interface board developed by Ray Byrne and Joe Lyle at Sandia.

The sample image to be processed is sent to the FPGA over Ethernet, where it is stored in the FPGA on-chip BlockRAM. When the entire image has been received and processing begins, a state machine iteratively takes pixel values from the image buffer and calculates the high-pass and low-pass components, then Haar coefficients from those components. These coefficients are then returned back to the user via the CameraLink interface. Block diagrams for the hardware implementation are shown in Figure 18 and 19. It should be noted that much of this design leveraged hardware and HDL code developed for other projects at Sandia; thus, the variety of hardware utilized is partially due to our desire for design reuse, vastly decreasing the amount of development time needed for framework-related tasks (such as data transfer to and from the FPGA).

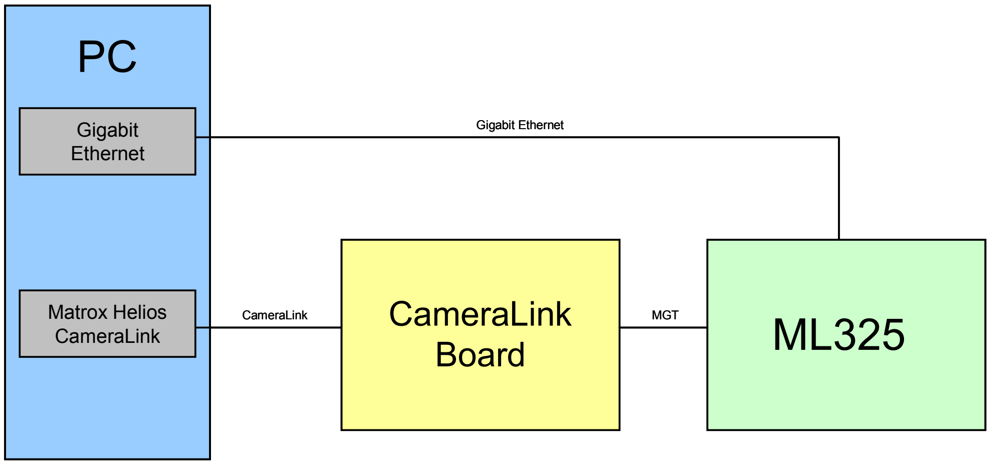

Figure 18: System Block Diagram

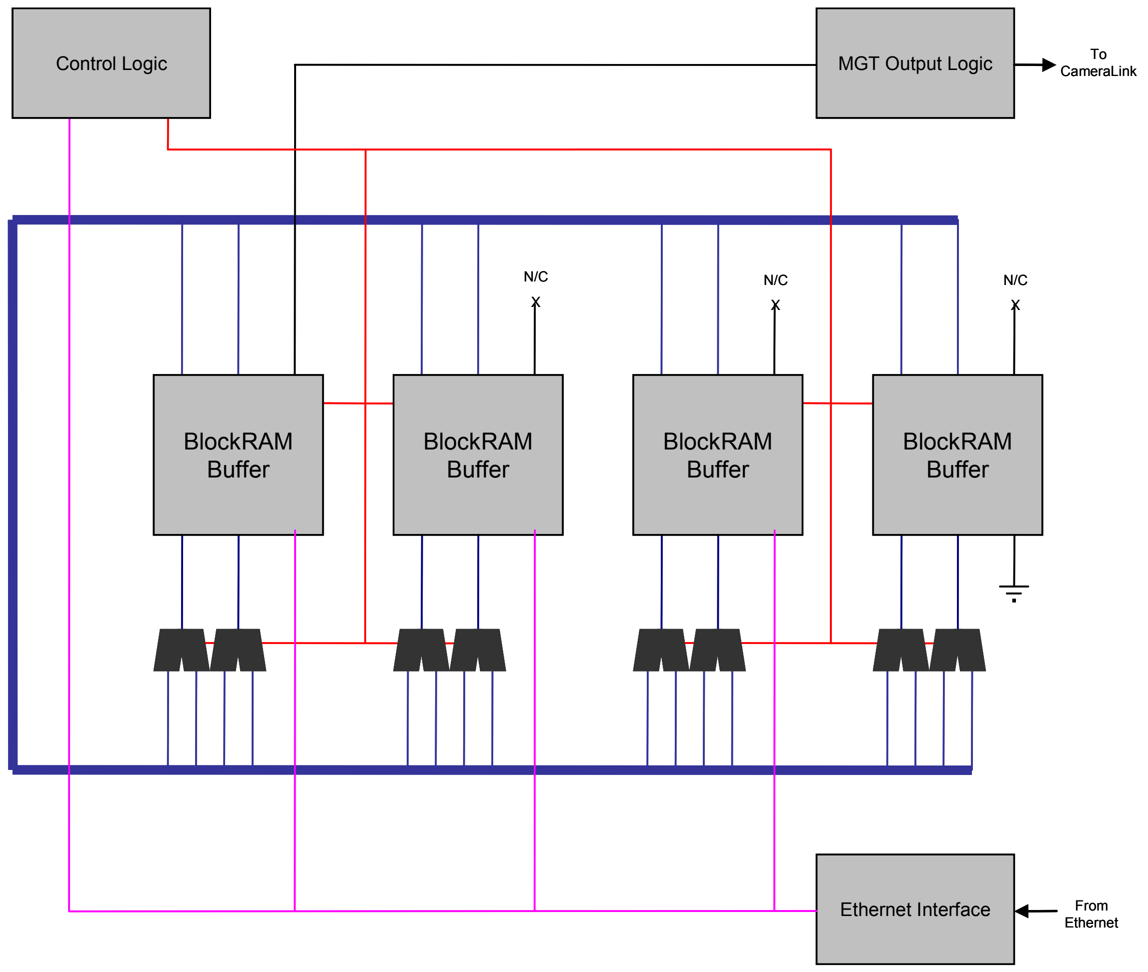

Figure 19: ML325 Block Diagram

9.1.1 Ethernet Interface

The Ethernet interface is the input mechanism to transfer image data from the PC to the FPGA for coefficient generation. The hardware consists of a Xilinx daughterboard which plugs into header pins present on the ML325 prototype board. The Ethernet daughterboard consists of two RJ45 8P8C connectors and two Ethernet PHY ICs (Marvell 88E1111). The inputs and outputs of the PHY are connected to multiple interfaces, selectable by jumpers on the daughterboard.

The interface was designed to be direct-cabled from the PC to the daughterboard with no intervening switches or routers (although this could be accommodated with some minor additional work). The Ethernet daughterboard handles all physical layer tasks for interfacing to the PC. The PC sends data to the FPGA in a packetized format, beginning with a sync word, followed by an image location, followed by the data at that location (represented as 16-bit signed data). Images are subdivided into packets with each packet containing the data for eight pixels of data.

As no full TCP implementation exists on the FPGA, the PC used to transmit the data must statically update its ARP table, at a minimum, to send data to the FPGA. The ARP tables should be updated to include an entry for IP address “10.0.10.1” – this is the IP address that is used by the C++ application to transfer data to the FPGA. The target hardware address may be user-defined to any valid MAC address. A full TCP implementation would be possible but was not pursued at this time.

9.1.2 FPGA BlockRAM Buffers

The BlockRAM buffers store image data in an array of on-chip RAM. Each buffer holds exactly 256x256x16, or a total of 1 Mbit of data, which is exactly the size of one image. Each buffer has two inputs and two outputs. When data is received via Ethernet and the system is not actively generating coefficients, the received data is written to three of the four buffers. The fourth buffer is not writable via Ethernet to provide one “non-volatile” or “persistent” image buffer if necessary (useful for some algorithms such as background removal). One set of output lines on one buffer is connected to a high-speed serial MGT to provide data to the CameraLink board at a data rate of 1.7 Gbps. When no other operations are pending, this buffer will continually output its data to the MGTs where it will ultimately be sent back to the PC.

The buffer outputs from each BlockRAM buffer are multiplexed to arithmetic operators (identified by the ALU-shaped components connected to the bottom of each buffer in the block diagram). This allows any combination of outputs from any number of buffers to be operated on in some fashion. Note that while the block diagram shows only two inputs for each ALU, in reality any number of operands may be used. As each BlockRAM buffer input can perform its own arithmetic function, buffers may operate on data in up to two different ways and write their results provided their target memory addresses do not collide.

Each BlockRAM buffer has a state machine with an address generator that takes inputs for starting row and column number, increment value for row and columns, and an ending row and column. The address generator, when signaled to start operating, will sweep through the valid addresses given the input parameters. This will cause the data in the BlockRAM to be output, where it may be operated upon outside of the BlockRAM buffer module. Alternately, the results of these operations may be sent to the BlockRAM inputs where, depending on the operation, data may be written to the BlockRAM (instead of read) using the addresses generated by the address generator. Control signals from the FPGA control logic determine address bounds and write enables.

9.1.3 FPGA Control Logic

The FPGA control logic is controlled mainly from the Ethernet interface. Ethernet commands may be sent to write to memory or to begin coefficient generation.

When coefficient generation begins, the appropriate address parameters are provided to the BlockRAM buffers to facilitate coefficient generation. The Haar algorithm is performed in three steps. The first step generates the high-pass and low-pass filter values. The second step generates the approximation and horizontal coefficients. The third step (which can and has previously been incorporated to run in parallel with the second step) generates the vertical and diagonal coefficients. If multiple iterations of the Haar algorithm are desired, the control logic will re-run the algorithm on the most recent approximation coefficient data and will derive the appropriate addresses as needed.

9.1.4 Matlab Software Interface

Initially the software interface on the PC was written using C++ libraries and compiled as standalone executables. Matlab's C-interface (MEX) was used to write Matlab-executable routines to perform the same functions, and as such the functions to send files to the FPGA, begin coefficient calculation, and retrieve processed results may now be called directly from Matlab.

9.2 Challenges and Potential Improvements to the Haar Algorithm Implementation

A few challenges were encountered during the implementation of this algorithm. The main drawbacks using the current implementation involve the allocation of resources. Specifically, the current implementation is not optimal in either resource utilization (memory) or in processing time (latency). However, using multiple image buffers to implement the Haar algorithm was chosen for its versatility in not only calculating Haar coefficients, but its ease in prototyping other algorithms (such as background removal or other filtering techniques) as well by simply changing a few values in the VHDL code.

One potential issue in implementing the Haar algorithm involves the bit-precision of intermediate values used in the Haar coefficient calculation. Using 16-bit inputs and the algorithm as described above, there is the possibility of two addition operations being performed on a 16-bit number. This conceivably could result in the final result being an 18-bit number, which cannot be adequately represented in the provided memory space. Our solution was to simply limit the input values to 14-bit, which was a suitable solution for the test data; however, for other input images this may not be an acceptable remedy. It should be noted that the actual Haar algorithm multiplies its data by a non-integer scaling factor during coefficient generation – this step was removed from this implementation and from the image reconstruction to avoid floating-point arithmetic.

Another consideration is the availability of memory. As the test images were relatively small (256x256x16), enough room existed for multiple image buffers in the memory available on the FPGA. However, for any images of reasonable size, larger off-chip memory will need to be utilized.

Also the processing time required to generate the Haar coefficients is related to the image size. For the small image sizes (256x256) the processing time required was not significant. However, for larger data sets, such as images from large focal plane arrays (FPAs), a significant amount of processing time is required. The method of storing the image in memory, then post-processing the data to generate the Haar coefficients, is probably not efficient enough for real-world systems. As images grow larger, proportionally more processing time will be needed for coefficient generation. This increased latency may affect the achievable frame rate for coefficient generation.

A pipelined approach for generating Haar coefficients may result in a much more streamlined operation with lower latency. This method could also theoretically reduce the amount of memory required for coefficient generation. If the input data could be reordered such that the data was presented in 2x2 blocks rather than straight scan lines, this could conceptually improve performance as well. Further exploration in this area is possible and warranted if the Haar wavelet is indeed considered for implementation on real-world systems.

10 ENCODING COEFFICIENTS

Once the wavelet transform has been performed, and the resulting coefficients have been appropriately manipulated (e.g. quantized, thresholded, de-noised, etc.), the next step is to apply some type of lossless encoding scheme to the coefficients. The subsequent sections address encoding methods for the zero and non-zero coefficients.

10.1 Zero Coefficients

There are two well established methods we considered for handling the zero coefficients: bit-level encoding (BLE) and run-length encoding (RLE). Both use a scheme to encode whether the coefficients are zero or not, and then maintain a separate list of the nonzero values. BLE uses a bit string where one bit is used for each value and the order of the bits corresponds to the order of values in the coefficient matrix. A ‘1’ is used to identify a nonzero number at that location and a ‘0’ indicates the value is zero. Since there is no need to encode the zero values, BLE requires 1 bit for each of the zeros. On the other hand, encoding the nonzero values requires one additional bit, since the bit in the string is needed as well as the encoded number. This scheme will only provide significant compression if the percentage of zeros is large. For instance if our data is

data = `[0 0 0 0 10 0 0 20 0 30 0 0 0],`

the location bit string and values set would be

location = [0 0 0 0 1 0 0 1 0 1 0 0 0]
values = [10 20 30].

RLE is a simple form of data compression where runs of repeated values are represented in a compressed format. There are many different variations on this general idea. The most frequent format is a pair of elements, one representing the value and the other representing the run length of that value. In our wavelet thresholded coefficients, runs of zeros are common. Therefore, we designed our run-length encoding method to efficiently compress runs of zeros and of nonzero values. Our method creates two vectors: a *counts* vector and a *values* vector. The *counts* vector is a series of integer values indicating the lengths of the runs. The values vector contains all the nonzero values. For example, consider the following sequence of coefficients:

data = `[0 0 0 0 10 20 11 0 30 0 0 0].`

This sequence will produce the following compressed *counts* and *values* vectors:

counts = [4 3 1 1 3]
values = [10 20 11 30].

We also considered a variation on RLE where the count is limited in size. This variation compresses the few long runs into shorter run values, reducing the number of possible count values. In our study we limited the counts to 31. So it encodes the following sequence:

data = [0 … 50 times … 0 10 20 11 0 30]

as

counts = [31 19 3 1 1]
values = [10 20 11 30].

The first 50 zeros are divided into the 31 and the 19. The next three nonzero values are then encoded as a 3 followed by two ones for the single zero and nonzero values. If the run happened to be 31 values long, a zero would follow it. In this paper, we refer to the first method as classic RLE (CRLE) and the second as limited RLE. (LRLE).

The efficiency of any RLE scheme depends on the distribution of the data. If long runs occur very frequently, RLE can achieve significant compression ratios. In the worst case, where there are no runs of more than one value, RLE can actually *increase* the number of bits required to encode the data. The size of the *counts* vector depends on the total number of runs in the coefficients and the size of the longest run. If there are long runs of zeros and nonzero numbers, the RLE method can represent the zeros with a fraction of a bit, outperforming BLE.

Since the order of the coefficients affects the compression in RLE, we investigated various ordering schemes to see how they impact the number of runs. Wavelet analysis gives the coefficients A_2, V_2, H_2, D_2, V_1, H_1, and D_1 in column major order. Most of A_2 is nonzero so reordering them was not considered, but the other six sets of coefficients were reordered. Two schemes shown in Figure 20 were implemented for reordering the coefficients. Since adjacent pixels are often similar, they were designed to place coefficients of adjoining pixels next to each other. The schemes take 4x4 blocks of pixels and reorders them. For instance, in the 2x2 scheme, the points are selected to group blocks of 2x2 pixels. Using coordinate notation (R,C) where R is the row and C is the column, the first nine points would be (1,1), (2,1), (2,2), (1,2), (1,3), (1,4), (2,4), (2,3) and (3,3). The diagonal scheme selects adjoining points moving from the upper left to the lower right in a serpentine fashion. The numbering for column major shown in Figure 20 is based on a 256x256 pixel image so the pixels in the first column go first, followed by the next column and so on. Both reordering schemes define the order of a 4x4 block of coefficients. The order of the 4x4 blocks over the entire image is shown in the Large Block table. After the first 4x4 block, the block to the right is done, followed by the one directly below the second and finally the block below the first block. That pattern is then repeated for the larger blocks of 8x8 and so on until the coefficient matrix is completely reordered. This reordering was done for each coefficient matrix, but the values in V_2, H_2, D_2, V_1, H_1, and D_1 were not mixed.

Column Major

1	257	513	769
2	258	514	770
3	259	515	771
4	260	516	772

2 x 2

1	4	5	6
2	3	8	7
15	14	9	10
16	13	12	11

Diagonal

1	4	5	16
2	3	6	15
9	8	7	14
10	11	12	13

Large (4x4, 8x8,…) Blocks

1	2
4	3

Figure 20: Coefficient Ordering Schemes by Area

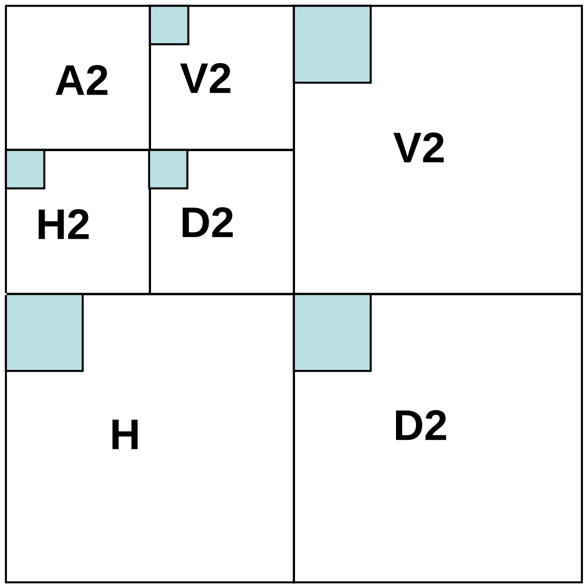

Figure 21: Multilevel wavelet decomposition coefficient groups

A different coefficient ordering scheme which grouped the coefficients by pixel was also considered. This ordering corresponds to the light blue rectangles shown in Figure 21 so the coefficients between the sets are interspersed. Using this ordering, the order of the first few coefficients is: V2(1,1), H2(1,1), D2(1,1), V1(1,1), V1(2,1), V1(1,2), V1(2,2), H1(1,1) , H1(2,1), H1(1,2), H(2,2), D1(1,1), D(2,1), D(1,2) ,D(2,2), V2(2,1), and H2(2,1). We refer to this reordering scheme as the pixel grouping.

10.2 Nonzero Coefficient Encoding

We considered three common approaches for lossless compression of the nonzero coefficients: Rice (RE), Huffman (HE), and Arithmetic (AE) encoding. A customized version of RE is employed on the current system, so it was a natural selection for this study. Huffman coding is a classic statistical method that is often used as a standard for compression. The more recent algorithm, AE often produces superior compression on image data and consequently has been included in the new JPEG 2000 standard.

The Rice compression algorithm used on the current system has been customized to include run-length encoding in addition to the standard Rice compression. The standard Rice compression takes advantage of the fact that most of the variation of the data occurs in the lower bits. Consequently, it will take a set of data and determine the optimal way to divide the data into upper and lower bits so as to minimize the encoding of both separately. For instance if we have the set of numbers 2995, 3003, 2997, and 3001, the binary codes are 0000 1011 1011 0011, 0000 1011 1011 1011, 0000 1011 1011 0101, and 0000 1011 1011 1001. The top ten bits are the same while there are differences in the lower six bits. The code for the numbers consists of the upper ten bits 0000 1011 10, followed by the lower bits for the numbers 11 0011, 11 1011, 11 0101, and 11 1001. Typically, groups of numbers are selected for compression. In the customized implementation, the values are grouped by sets of 16 for compression. RE is particularly effective when only a few bits are varying or the values are small.

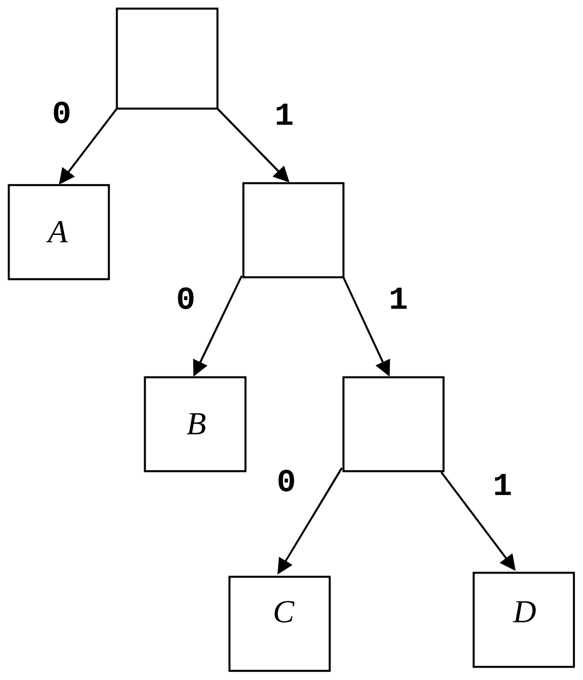

Figure 22: Example of a Huffman binary tree for encoding values.

HE is a lossless compression algorithm invented by David A. Huffman in 1952. The algorithm is based on *entropy encoding*. Each value in the input is paired with a compressed code. The most common values receive the shortest codes. Huffman codes are binary strings of variable length. The strings are constructed so that no code is a prefix of any other code, simplifying transmission and decompression. The Huffman coding algorithm works by constructing a binary tree of nodes, with one leaf node per value. At each internal node, the left branch is labeled with a zero and the right branch with a one. Thus, a path through the tree from the root to a leaf yields a unique sequence of zeros and ones that identifies the value associated with that leaf. An example tree in Figure 22 shows the unique codes for A (0), B (10), C (110), and D (111). HE has been shown to be an optimal entropy encoding method if the compression is done on a value-by-value basis.

The final method (AE) is similar to HE in that it uses probabilities, but it considers the entire data set at once. The method starts with the interval [0-1) and uses the probabilities of the successive values to narrow the interval. A narrow interval requires more bits to specify, so values with lower probabilities narrow the window more and conversely values with higher probabilities narrow it less. For example if we have three values, x, y and z, with probabilities 0.4, 0.5, and 0.1 respectively, we would encode “yyyz” as follows: the first y would convert the interval to [0.4-0.9); the next y to [0.6-0.85); the third y to [0.7-0.825), and the z to [0.8125-0.8250). The final code could be any number within that range. In our implementation, the process started with equal probabilities for all numbers and gradually refined the probabilities as the values were being read. This approach is simpler since it does not require separate processing to determine the probabilities. However, the coding of the data will be less efficient if the estimates of the probabilities are poor.

The advantage of the AE method over HE is that it does not require an integral number of bits to encode the values since it assigns one code for the entire image. If the sequence of value probabilities is negative powers of two, then the optimal number of bits for each value is integral, and HE and AE perform equally well. It is unlikely that wavelet coefficients will satisfy this condition.

10.3 Procedure

Before any of these methods could be applied, the form of the coefficients needed to be modified. The RE algorithm was designed for non-negative data, so the wavelet coefficients were converted to positive values by moving the sign bit from the most-significant bit to the least-significant bit. To do this, all positive numbers were multiplied by 2, making the lowest bit 0. Negative numbers were multiplied by -2 to make them positive, and one was subtracted from the result, thereby setting the lowest bit to 1. Using this scheme, [-1 1 -2 2 3 …] is mapped to [1 2 3 4 5 …].

For the compression using RE, these 16 bit numbers were encoded directly. For the HE and AE methods, we divided each 16-bit value into two 8-bit bytes. These two methods require probabilities for each possible value. With 16-bit numbers, there are over 65000 possible values, and most of those values will not appear in a given set of the wavelet coefficients. The conversion to 8-bit numbers reduces the total number of possible values to a more manageable 256.

Based on the preliminary indications from our earlier study, we chose to perform the wavelet analysis using Haar wavelets. The Haar wavelet is well suited to data with short spikes and discontinuities. We computed the compressed wavelet coefficients from 574 different frames that were selected from a set of data files covering a variety of images with varying noise characteristics. Of those frames, 514 had 256 rows and columns of pixels (64K values) and 60 were 128 rows and columns (16K values). The coefficient threshold was varied to provide a range of percentages of zero coefficients from ~45% to ~93%. The majority of cases had 75% to 90% zeros, since that appears to be the typical range for our applications.

In this study, we considered two basic compression approaches. First, we simply encoded all the wavelet coefficients using RE, HE, and AE. For the second approach, we used the bi-level methods BLE and RLE to separate the zero and nonzero values. The nonzero values including the counts and values vectors plus the remaining A_2 coefficients were then compressed using RE, HE and AE. Since the BLE is designed to use 1 bit per value (BPV) to encode the zero-nonzero order, we determined the BPV for RLE for comparison. As noted above, we split the 16-bit coefficients into two 8-bit byte for the HE and AE methods. This gave us another way to apply the bi-level methods. Since most of the coefficients are small (<256), their upper bytes are zero. That gave us a new set of zero and nonzero values to encode.

10.4 Results of the Encoding Methods

There were no significant differences in the number of runs between the original column major ordering and either of the area ordering schemes described in Figure 20. However, the pixel grouping scheme did result in fewer runs. The plots in Figure 23 show the percentage difference in the number of runs between the 2x2 ordering scheme (upper plot) and the diagonal scheme (lower plot) versus the pixel grouping scheme, respectively. The figure shows the pixel grouping scheme generally improved with higher numbers of zero coefficients.

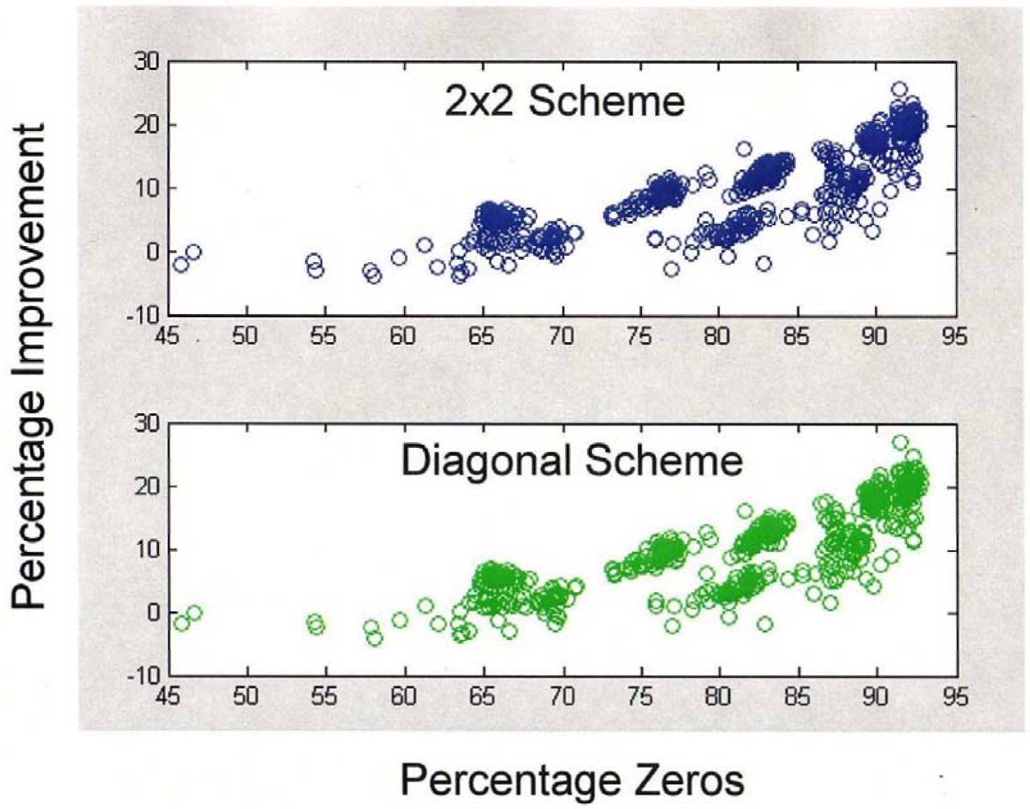

Figure 23: Percentage improvement in the number of strings with the pixel grouping scheme over the 2x2 and diagonal schemes.

The bits per value (BPV) versus the percentage of zeros using the two RLE methods is presented in Figure 24. The upper plot shows the BPV for LRLE and the bottom is the difference between LRLE and CRLE. As expected, the BPV decreased with increasing zeros as the run lengths increased. By design, the BLE algorithm uses one bit for each value. RE compressed the best, going below the one BPV level at about 70% zeros. The difference between LRLE and CRLE was small at lower percentages but CRLE was much better above 85% zeros.

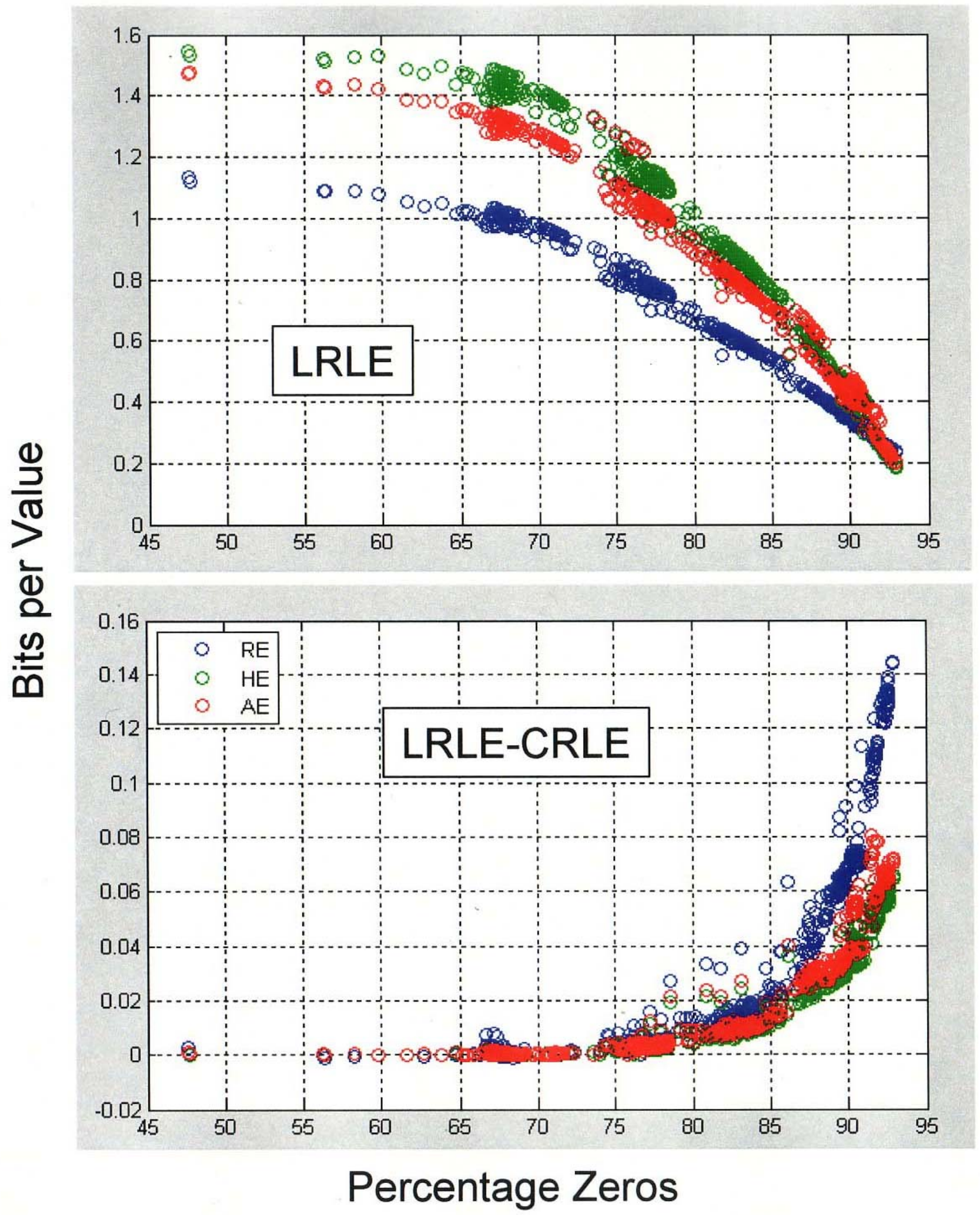

Figure 24: BPV needed for LRLE (above) and the difference between LRLE and CRLE (below) versus percentage of zero coefficients.

The average compression ratios for the various methods utilized in this study are summarized in Table 4 for frames with 64K values. The compression ratio was computed by dividing the size of the coefficients by the size of the compressed values. The averages were determined over 5% intervals from 65% to 90% zeros. Each 3x5 block of numbers in the table is the average compression ratio for the RLE method labeled at the top for the three nonzero compression methods (RE, HE and AE) over the 5 percentage ranges, 65-70, 70-75, 75-80, 80-85 and 85-90 percent zeros. The 3x5 block at the bottom labeled "Coefficients" has the ratios obtained by compressing the coefficients without using RLE first. Using RLE, RE consistently worked the best, followed by AE and then HE.

	LRLE Reordered		
% Zeros	RE	HE	AE
65-70	5.3	4.1	4.3
70-75	6.2	4.7	5.0
75-80	7.0	5.5	5.6
80-85	9.2	7.3	7.4
85-90	13.5	11.5	11.4

LRLE		
RE	HE	AE
5.1	4.0	4.3
6.0	4.7	4.9
6.8	5.4	5.5
8.9	7.1	7.2
13.3	11.1	11.0

LRLE Bytes		
RE	HE	AE
4.9	3.8	4.1
5.7	4.4	4.7
6.4	5.0	5.1
8.2	6.5	6.6
11.7	10.0	10.0

	CRLE Reordered		
% Zeros	RE	HE	AE
65-70	5.3	4.1	4.3
70-75	6.2	4.7	5.0
75-80	7.1	5.6	5.6
80-85	9.3	7.4	7.5
85-90	14.5	12.0	11.9

CRLE		
RE	HE	AE
5.1	4.0	4.3
6.0	4.7	4.9
6.8	5.4	5.5
9.0	7.1	7.2
14.0	11.3	11.3

CRLE Bytes		
RE	HE	AE
5.2	3.9	4.2
6.1	4.6	4.9
6.8	5.2	5.3
8.9	6.8	6.9
13.8	10.7	10.8

	Coefficients		
% Zeros	RE	HE	AE
65-70	3.5	4.0	5.3
70-75	4.1	4.4	6.2
75-80	3.9	4.6	6.8
80-85	4.6	5.1	8.6
85-90	6.5	5.8	12.2

Table 4: Average compression ratios for frames with 64K values over 5% ranges of zero percentages for LRLE in the top three 3x5 blocks on reordered coefficients, integer values, and byte values; CRLE in the middle blocks on reordered coefficients, integers, and bytes; and in the bottom block using RE, HE and AE without first using RLE.

The table shows the values for the various approaches we explored. The key findings were:

1. Reordering the coefficients by the pixel grouping scheme led to an improvement of about 3%,
2. LRLE was comparable to CRLE except at the highest zero percentages,
3. Compressing using 8-bit values instead of 16 bit values was worse for LRLE with all methods by around 7% and for CRLE with HE and AE by around 3%, and
4. Overall, the best method was CRLE on Reordered coefficients with RE for the final compression (left middle block).

	LRLE Reordered		
% Zeros	RE	HE	AE
70-75	8.3	6.0	5.6
75-80	9.1	6.6	6.2
85-90	11.6	9.9	9.1

LRLE		
RE	HE	AE
8.3	5.9	5.6
9.1	6.6	6.2
11.5	9.5	8.8

LRLE Bytes		
RE	HE	AE
7.5	5.6	5.3
8.2	6.1	5.8
10.2	8.5	8.0

	CRLE Reordered		
% Zeros	RE	HE	AE
70-75	8.3	6.0	5.6
75-80	9.1	6.6	6.2
85-90	12.2	10.2	9.4

CRLE		
RE	HE	AE
8.3	5.9	5.6
9.1	6.6	6.2
11.8	9.6	8.9

CRLE Bytes		
RE	HE	AE
8.2	5.9	5.6
9.0	6.6	6.2
11.6	9.1	8.5

	Coefficients		
% Zeros	RE	HE	AE
70-75	7.1	5.4	7.5
75-80	7.7	5.7	8.3
85-90	5.2	5.5	9.6

Table 5: Average compression ratios for frames with 16K values over ranges 70%-75%, 75%-80% and 85%-90% zeros for LRLE in the top three 3x5 blocks on reordered coefficients, integer values, and byte values; CRLE in the middle blocks on reordered coefficients, integers, and bytes; and in the bottom block using RE, HE and AE without first using RLE.

There were several differences between the 64K and the 16K frame compression ratios. The compression ratios for the 16K frames (Table 5) were higher for the smaller percentages but lower in the 85-90% range than the ratios computed for the 64K frames. RE changed the most at ~30% higher, HE was better by ~20% while AE was only better by ~12%. When encoding the coefficients directly, the difference for RE was even greater in the 75-80% range (7.7 vs. 3.9). In that range, RE was close to AE but in the 85-90% range the RE ratio dropped below the HE level. In contrast, RE was better in that range with the 64K frames. Whether the differences were caused by artifacts in the type of data for the 16K frames or related to the number of values is not known. It is worth noting, however, that the optimal compression method may be data size dependent.

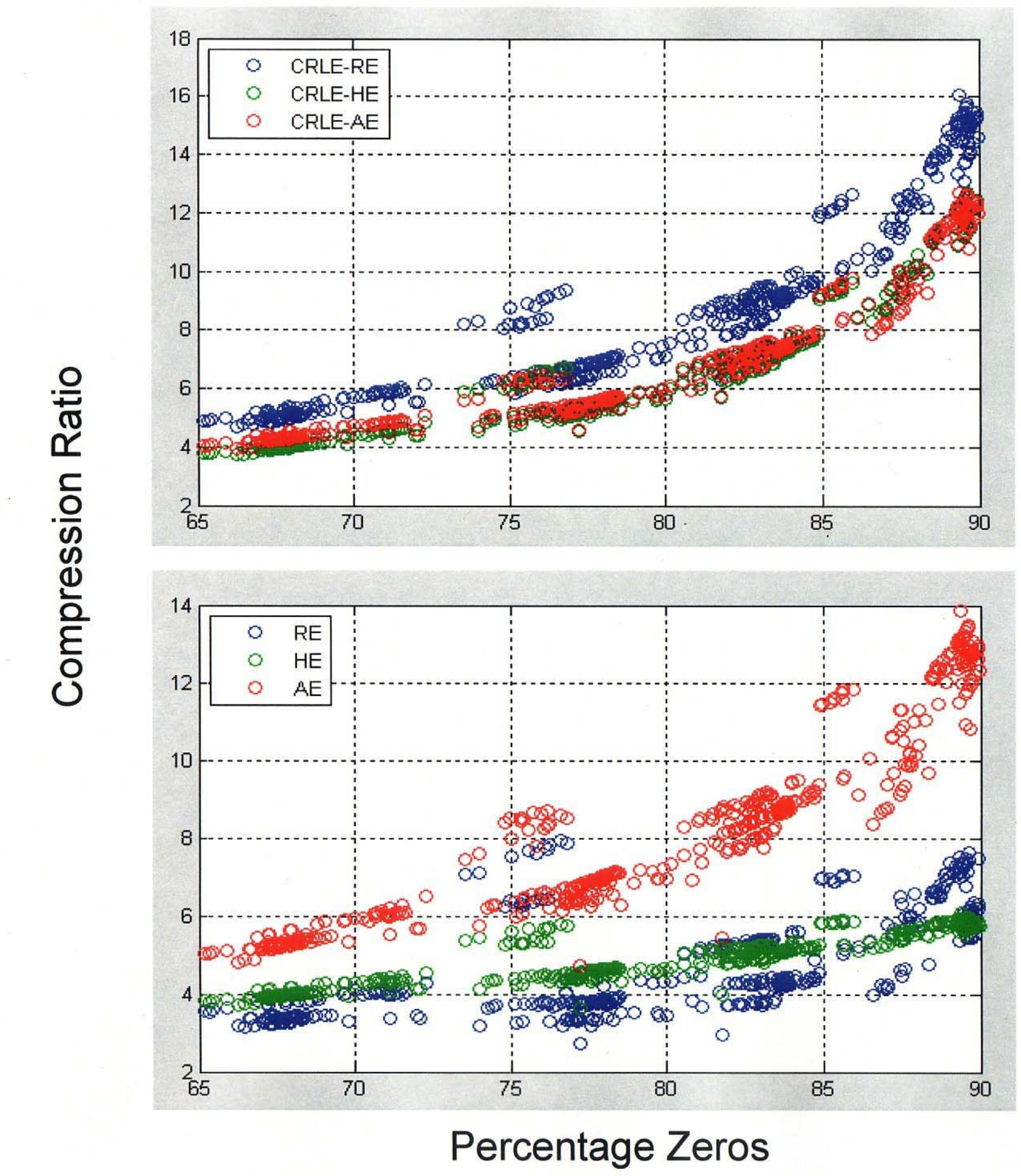

Figure 25: Compression Ratios on all coefficients using RE, HE and AE when using CRLE (upper plot) and when compressing the coefficients directly (lower plot).

For comparison, we applied RE, HE and AE to the coefficients directly. The upper plot in Figure 25 provides the compression ratios when CRLE was applied to the data before using RE, HE and AE and the lower plot has the ratios from compressing the coefficients directly. When the coefficients were compressed directly, AE provided superior compression ratios for all percentages shown. The compression ratios for the higher percentages were significantly higher than the compression ratio of around 3 currently experienced using Rice on the original data. HE provided higher compression ratios than RE except for the highest percentages. It is likely that the run-length zero encoding in the customized version was a factor in causing the switch. When applying CRLE to the data (upper plot), the compression ratios were similar to the lower plot values at the lower percentages. The ratios for RE and HE were ~2 times better at the highest percentages while the ratio for AE was worse when using RLE. Note that RE performed the best with CRLE but AE did the best when compressing the coefficients directly. We will address that issue in the next section.

10.5 Discussion of Encoding Methods

The causes for the differences in the compression ratios between the 64K and 16K are not presently known. We intend to probe the information in those frames to understand why they were different. Hopefully, that will provide insight into estimation of the compression ratios based on data characteristics, particularly on data set size.

As noted above, AE performed better then RE if RLE was not performed first but worse if it was. Also as shown in Table 5, AE performed worse after RLE then it did on the full set of coefficients. RLE generates several groups of values that need to be encoded: the run-lengths for the zeros and nonzero values, the nonzero values in the detail coefficients, and the A_2 coefficients. Each of these groups are separately encoded by RE, HE and AE to complete the compression. As noted above, in our implementation of AE, the probabilities were all equal at the beginning of the compression and were refined as the data was read. Since these sets are much smaller the probability estimates may not be as good. The best coding for AE can be obtained by making two passes on the data. The first is to determine the exact probabilities and the second does the coding. We intend to implement this method as well to see if the compression ratios for AE can be improved.

The best methods for the various options were:

1) CRLE-RE using pixel grouping reordered coefficients	(Reorder CRLE-RE),
2) CRLE followed by RE	(CRLE-RE),
3) CRLE with AE	(CRLE-AE)
4) AE of all the coefficients	(AE).

The results are displayed in Figure 26. The compression ratio difference between the methods increased as the percentage of zeroed coefficients increased. CRLE-RE on the reordered coefficients was consistently the best. The optimal choice for our applications may depend not only on the percentage of zeros in the coefficients but also on the overhead and computational issues associated with the methods. For instance, if we use the AE coding method in the previous paragraph, the 256 extra probability values would need to be transmitted to the decoding algorithm as well. However for a sequence of frames, that might not be necessary. Never-the-less, the compression ratios we achieved in this study are very encouraging since they are well above the compression ratios of approximately 3 achieved by the current method on the raw data. To validate our conclusions, we also compressed the data frames with the background removed using the same compression methods. AE compressed by compression ratio roughly between 2.2 and 3.0, RE by factors generally a little lower (between 2.0 and 2.5), while HE was the worst at between 1.7 and 2.3. Since these levels were at or below the typical compression ratios of the current system, we are confident the improvements we observed in the compression ratios are valid.

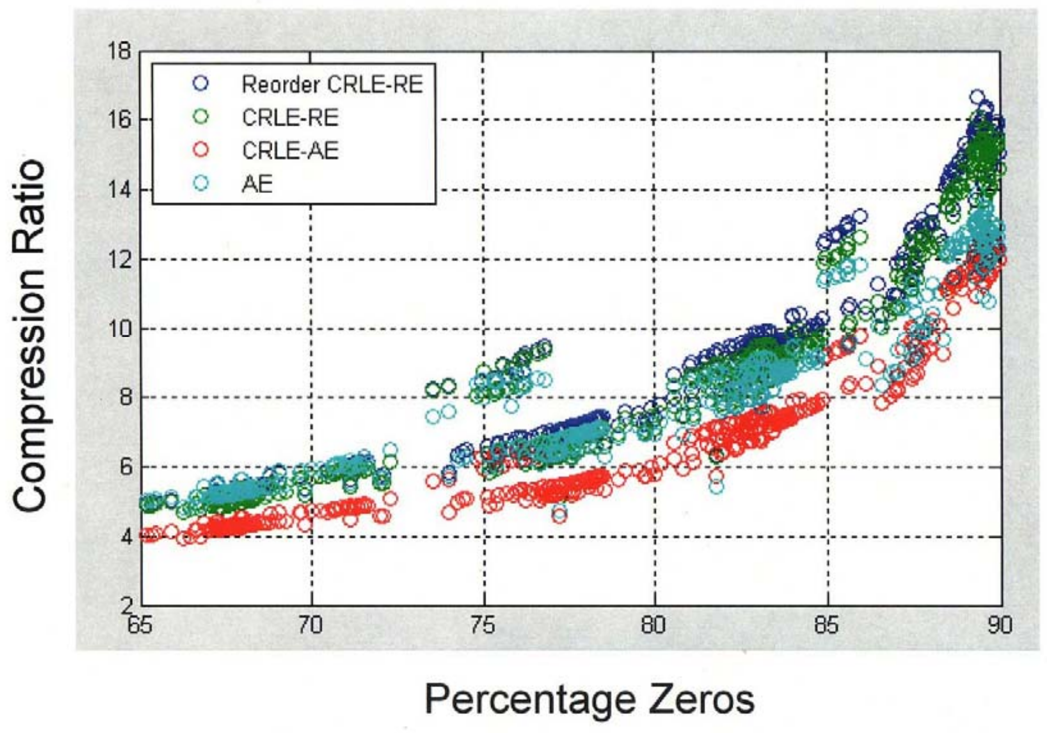

Figure 26: Compression ratios for the best methods: 1) CRLE and RE using reordered coefficients, 2) CRLE followed by RE, 3) CRLE with AE, and 4) AE of all the coefficients. Data was plotted against the percentage of zero coefficients.

11 SUMMARY AND CONCLUSIONS

As we considered the various approaches for lossy compression, we concluded that wavelet compression offered the best options. We examined many aspects of wavelet compression on real and simulated image frame data. Based on our investigation the Haar and Daubechies 2 were the most promising. The Haar has the advantage of being able to use integer arithmetic while Daubechies 2 showed advantages in maintaining target intensity and SNR in the simulated data with very dim targets. However on the real data where the Haar was used for compression, the detection using the current process stream performed comparably to detection on the uncompressed data. Even in the worst cases, the differences were well within acceptable levels. We also were able to successfully implement the Haar wavelet transform in hardware. In the final step of the process where the coefficients were compressed with lossless methods, we were able to obtain compression ratios of approximately 6, 8 and 16 for 70%, 80% and 90% zeros as compared to levels usually less than 3 using the same algorithms to compress the data directly. Since we used percentages up to 90% on the real data with minimal impact on detection, that level of compression might be acceptable. Based on our visual observations of the impact of various levels of zeros on target signatures, we found that levels around 80% had significantly less impact on the smaller targets than those around 90%, so we would suggest caution with the higher levels for some data sets. In this investigation, we did not find any significant concerns or problems with using wavelet compression on our current detection system. Since there are several parameters that can be selected with the compression, tuning would be required to suit particular applications in the future.

12 ACKNOWLEDGEMENTS

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000. This work was funded by Sandia's Laboratory Directed Research and Development (LDRD) Program.

13 REFERENCES

[1] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, Vol.27, pp.379-423, 623-656, July, October, 1948.

[2] B. Aiazzi, L. Alparone, S. Baronti, and M. Selva, “Lossy compression of multispectral remote-sensing images through multiresolution data fusion techniques,” *Proceedings of the SPIE*, vol. 4793, pp. 95-106, 2003.

[3] R. J. Clarke, “Image and video compression: a survey,” *International Journal of Imaging Systems and Technology*, vol. 10, no. 1, pp. 20-32, 1999.

[4] E. Magli, G. Olmo, and F. Sellone, “Integer wavelet packets and their application to a lossy compression system for SAR images,” *Proceedings of the International Conference Image Analysis and Processing, 1999*, pp. 780-785, Sept. 27-29, 1999.

[5] J. C. Tilton and M. Manohar, “Preserving radiometric resolution in remotely sensed data with lossy compression,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 35, no. 5, pp. 1171-1176, 1997.

[6] W. F. Turri, Waleed W. Smari, and F. A. Scarpino, “Integer division for quantization of wavelet-transformed images, using field programmable gate arrays,” *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems*, vol. 1, pp. 176 - 179, 2001.

[7] A. Said and W. A. Pearlman, “An image multiresolution representation for lossless and lossy compression,” *IEEE Transactions on Image Processing*, vol. 5, no. 9, pp. 1303-1310, 1996.

[8] M. N. Do and M. Vetterli, “Contourlets: a directional multiresolution image representation,” *Proceedings of the 2002 International Conference on Image Processing*, vol. 1, pp.357 -360, Sept. 22-25, 2002.

[9] D. L. Donoho, “De-noising by soft-thresholding,” *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613-627, 1995.

[10] Y. Feng, S. Thanagasundram, and F. S. Schlindwein, “Discrete wavelet-based thresholding study on acoustic emission signals to detect bearing defect on a rotating machine,” *Proceedings of the 13th International Conference on Sound and Vibration*, Vienna, Austria, July 2-6, 2006.

[11] J. M. Shapiro, “Embedded image coding using zerotrees of wavelet coefficients,” *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445-3462, 1993.

[12] I. Daubechies and W. Sweldens, “Factoring wavelet transforms into lifting steps,” *J. Fourier Anal. Appl.*, vol. 4, No. 3, pp. 247-269, 1998.

[13] R. A. DeVore, B. Jawerth, and B. J. Lucier, “Image compression through wavelet transform coding,” *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 719-746, 1992.

[14] J. S. Walker, T. Q. Nguyen, and Y. Chen, “A low-power, low-memory system for wavelet-based image compression,” *Optical Engineering, Research Signposts*, vol. 5, pp. 111-125, 2003.

[15] D. A. Yockey, “Multiresolution wavelet decomposition image merger of Landsat thematic mapper and SPOT panchromatic data,” Sandia National Laboratories Technical Report, SAND95-0439, Albuquerque, NM, 1995.

[16] C. D. Creusere, “A new method of robust image compression based on the embedded zerotree wavelet algorithm,” *IEEE Transactions on Image Processing*, vol. 6, no. 10, pp. 1436-1442, 1997.

[17] A. H. Tewfik, D. Sinha, and P. Jorgensen, “On the optimal choice of a wavelet for signal representation,” *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 747-765, 1992.

[18] I. Daubechies, “Recent results in wavelet applications,” *J. Electronic Imaging*, vol. **7**, no. 4, pp. 719-724, 1998.

[19] Y. Hawwar and A. Reza, “Spatially adaptive multiplicative noise image denoising technique,” *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1379-1404, 2002.

[20] V. T. Franques and V. K. Jain, “Subband coding of images using eigen designed quadrature mirror filters,” *Proceedings of the 1996 IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 145 - 148, May 12-15, 1996.

[21] R. W. Ives, N. Magotra, and C. Kiser, “Wavelet compression of complex SAR imagery using complex- and real- valued wavelets: a comparative study,” *Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 1294 - 1298, Nov. 1-4, 1998.

[22] J. S. Walker, “Wavelet-based image compression,” sub-chapter in *The Transform and Data Compression Handbook*, CRC Press, 2000.

[23] A. Munteanu, J. Cornelis, G. Van der Auwera, and P. Cristea, “Wavelet-based lossless compression scheme with progressive transmission capability,” *International Journal of Imaging Systems and Technology*, vol. 10, pp. 76-85, 1999.

[24] A. R. Calderbank, I. Daubechies, W. Sweldens, and B. Yeo, “Lossless image compression using integer to integer wavelet transforms,” *Proceedings of the 1997 International Conference on Image Processing,* pp. 596-599, 1997.

[25] JPEG 2000 Part I Final Committee Draft Version 1.0 ISO/IEC JTC1/SC29 WT1, eds. M. Boliek, C. Christopoulos, and E. Majani.

[26] S. Mallat, “A theory for multiresolution signal decomposition: the wavelet representation”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 2, no. 7, 1989.

[27] R. A. DeVore, B. Jawerth, and B. J. Lucier, "Image compression through wavelet transform coding," IEEE Trans. on Inf. Theory, vol. 38, 2, pp. 719-746, (1992).

[28] I. Daubechies, [Ten Lectures on Wavelets], Society for Industrial and Applied Mathematics, 1992.

[29] A. Graps, “An introduction to wavelets”, IEEE Computational Science and Engineering, vol. 2, no. 2, 1995.

[30] Pen-Shu Yeh, P. Armbruster, A. Kiely, B. Masschelein, G. Moury, C. Schaefer, and C. Thiebaut, “The new CCSDS image compression recommendation,” *Proceedings of the 2005 IEEE Aerospace Conference*, pp. 4138-4145, March 5-12, 2005.

[31] D. S. Myers, D. K. Melgaard, R. H. Byrne, P. J. Lewis, “Impact of wavelets on image data characterization during compression,” *Proceedings of the SPIE Optics & Photonics Conference*, San Diego, CA, August 10-14, 2008.

[32] C. Rivolta, “Airy disk diffraction pattern: Comparison of some values of f/No. and obscuration ratio,” Applied Optics, No. 25, vol. 14, pp. 2404-2408, 1986.

14 APPENDIX A

MATLAB Code

Appendix: Matlab Algorithm

The Haar implementation described in this document is based on the Matlab algorithm below:

```
function coef=Haar2DCoef(x,nLevel,val)

% 2-D Haar wavelet decomposition
%
% x             input matrix
% nLevel        number of levels of decomposition
% val           =1 if coefficients are not scaled
% coef          wavelet coefficients [a h v d]
% a=approximations, h=horizontal, v=vertical, d=details(diag)

    if nargin<3
        val=sqrt(2)/2;
    end
    hiFilter=[val -val];     % high pass filter
    loFilter=[val val];      % low pass filter
    coef=[];
    for iLevel=1:nLevel
        [s1,s2]=size(x);
        hiZ=zeros(s1,s2/2);
        loZ=zeros(s1,s2/2);
        j=0;
        %apply filters to columns
        for icol=1:2:s2
            j=j+1;
            for irow=1:s1
               loZ(irow,j)=x(irow,icol)*loFilter(1)+x(irow,icol+1)*loFilter(2);
               hiZ(irow,j)=x(irow,icol)*hiFilter(1)+x(irow,icol+1)*hiFilter(2);
            end
        end
        j=0;

        a=zeros(s1/2,s2/2);
        h=zeros(s1/2,s2/2);
        v=zeros(s1/2,s2/2);
        d=zeros(s1/2,s2/2);

        %apply filters to rows
        for irow=1:2:s1
            j=j+1;
            for icol=1:s2/2
                a(j,icol)=loZ(irow,icol)*loFilter(1)+loZ(irow+1,icol)*loFilter(2);
                h(j,icol)=loZ(irow,icol)*hiFilter(1)+loZ(irow+1,icol)*hiFilter(2);
                v(j,icol)=hiZ(irow,icol)*loFilter(1)+hiZ(irow+1,icol)*loFilter(2);
                d(j,icol)=hiZ(irow,icol)*hiFilter(1)+hiZ(irow+1,icol)*hiFilter(2);
            end
        end
        x=a;
        coef=[h(:)' v(:)' d(:)' coef];
        clear a d h v
    end
    coef=[x(:)' coef];
```

There are two key iterative loops in this algorithm that perform the following two operation sets:

```
 loZ(irow,j)=x(irow,icol)*loFilter(1)+x(irow,icol+1)*loFilter(2);
 hiZ(irow,j)=x(irow,icol)*hiFilter(1)+x(irow,icol+1)*hiFilter(2);
```

and

```
a(j,icol)=loZ(irow,icol)*loFilter(1)+loZ(irow+1,icol)*loFilter(2);
h(j,icol)=loZ(irow,icol)*hiFilter(1)+loZ(irow+1,icol)*hiFilter(2);
v(j,icol)=hiZ(irow,icol)*loFilter(1)+hiZ(irow+1,icol)*loFilter(2);
d(j,icol)=hiZ(irow,icol)*hiFilter(1)+hiZ(irow+1,icol)*hiFilter(2);
```

It should be noted that our FPGA logic is using strictly integer arithmetic. Therefore, the values of loFilter and hiFilter defined in the code:

```
hiFilter=[val -val];     % high pass filter
loFilter=[val val];      % low pass filter
```

The values of hiFilter equate to [1 -1] and the values for loFilter equate to [1 1]. As all of the multiplication in the Haar algorithm multiplies pixel values against elements in these two matrices, the logic for the Haar algorithm can be reduced using identity and inverse properties. By filling in the values for hiFilter and loFilter, we obtain:

```
loZ(irow,j)=x(irow,icol)*1+x(irow,icol+1)*1;
hiZ(irow,j)=x(irow,icol)*1+x(irow,icol+1)*(-1);
```

and

```
a(j,icol)=loZ(irow,icol)*1+loZ(irow+1,icol)*1;
h(j,icol)=loZ(irow,icol)*1+loZ(irow+1,icol)*(-1);
v(j,icol)=hiZ(irow,icol)*1+hiZ(irow+1,icol)*1;
d(j,icol)=hiZ(irow,icol)*1+hiZ(irow+1,icol)*(-1);
```

This may be further reduced to include only addition and subtraction operations:

```
loZ(irow,j)=x(irow,icol)+x(irow,icol+1);
hiZ(irow,j)=x(irow,icol)-x(irow,icol+1);
```

and

```
a(j,icol)=loZ(irow,icol)+loZ(irow+1,icol);
h(j,icol)=loZ(irow,icol)-loZ(irow+1,icol);
v(j,icol)=hiZ(irow,icol)+hiZ(irow+1,icol);
d(j,icol)=hiZ(irow,icol)-hiZ(irow+1,icol);
```

The for-loops around these two operations step through the pixels such that for each pair of pixels in adjacent columns, a loZ and hiZ value is generated. These loZ and hiZ values are then combined with loZ and hiZ values in adjacent columns to form the approximation, horizontal, vertical, and diagonal Haar coefficients.

Internal Distribution:

5	MS0576	John Feddema, 5535
1	MS1243	Ray Byrne, 5535
5	MS1243	Carol Harrison, 5534
15	MS0576	Dave Melgaard, 5534
5	MS0576	Phillip Lewis, 5534
2	MS0986	David Lee, 2664
1	MS0980	Steve Gentry, 5716
1	MS0980	Jay Jakubczak, 5710
1	MS0972	Kurt Lanes, 5550
1	MS0980	John Rowe 5550
1	MS0980	Dennis L. Eilers 5550
1	MS0509	Michael Knoll, 5330
1	MS0513	Matthew Brown, 5335
1	MS0503	Don Tolsch, 5339
1	MS0501	Steve Rohde, 5337
1	MS0986	Jeff Kalb, 2664
1	MS0980	Matthew Napier, 5571
1	MS0519	Joe Lyle, 5341
1	MS1108	Jeffrey J. Carlson, 6332
1	MS1072	Lyndon Pierson, 5629
1	MS1243	Kurt Larson, 5534
1	MS0576	Richard Wickstrom, 5531
1	MS0576	Scott Strong, 5531
1	MS0986	J. (Heidi) Ruffner, 2664
1	MS0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories