Low complexity lossless compression of underwater sound recordings

Mark Johnson[a)]
Scottish Oceans Institute, University of St. Andrews, Fife KY16 8LB, United Kingdom

Jim Partan and Tom Hurst
Woods Hole Oceanographic Institution, Woods Hole, Massachusetts 02543

(Received 1 February 2012; revised 1 October 2012; accepted 31 December 2012)

Autonomous listening devices are increasingly used to study vocal aquatic animals, and there is a constant need to record longer or with greater bandwidth, requiring efficient use of memory and battery power. Real-time compression of sound has the potential to extend recording durations and bandwidths at the expense of increased processing operations and therefore power consumption. Whereas lossy methods such as MP3 introduce undesirable artifacts, lossless compression algorithms (e.g., FLAC) guarantee exact data recovery. But these algorithms are relatively complex due to the wide variety of signals they are designed to compress. A simpler lossless algorithm is shown here to provide compression factors of three or more for underwater sound recordings over a range of noise environments. The compressor was evaluated using samples from drifting and animal-borne sound recorders with sampling rates of 16–240 kHz. It achieves >87% of the compression of more-complex methods but requires about 1/10 of the processing operations resulting in less than 1 mW power consumption at a sampling rate of 192 kHz on a low-power microprocessor. The potential to triple recording duration with a minor increase in power consumption and no loss in sound quality may be especially valuable for battery-limited tags and robotic vehicles.
 [http://dx.doi.org/10.1121/1.4776206]

PACS number(s): 43.30.Xm, 43.38.Md [AMT] Pages: 1387–1398

I. INTRODUCTION

There is growing recognition of the value of acoustic listening devices in studies of marine life. Many aquatic animals produce or respond to sound making this an important source of information about their presence, movements, and behavior. New statistical methods are enabling estimates of abundance from call counts in underwater sound recordings with applications to conservation management and mitigation of human impacts on species that are otherwise difficult to study (Mellinger *et al.*, 2007; Hastings, 2008; Marques *et al.*, 2009). To collect data for these analyzes, sound recording devices have been installed on platforms as diverse as long-term moorings (Wiggins, 2003; Clark and Clapham, 2004), underwater vehicles (Baumgartner and Fratantoni, 2008), and miniature animal tags (Burgess *et al.*, 1998; Johnson and Tyack, 2003). In most of these applications, memory and battery capacity are constrained leading to trade-offs among cost, size, recording bandwidth, and duration that impact the quality of sound recordings and the resulting science. In long-term moorings, service trips to replace batteries and hard drives represent an on-going expense that must be weighed against the value of wide-bandwidth or multi-channel recordings. Constraints are even more acute in gliders that must perform multi-month missions or in tags that must be kept small to minimize impact to the host animal. Some applications require radio transmission of underwater sound to a shore station (e.g., Clark *et al.*, 2007), and the limited bandwidth of the telemetry link places another constraint on recording quality. There is then a substantial benefit in compressing audio data to maximize the usage of recording or telemetry capacity, provided that this does not impact the power requirements of the device or compromise data quality. The issue of data quality is fundamental: Underwater sound recordings are a valuable scientific resource that may be analyzed in many different ways. It is difficult to anticipate the impact of data degradation on the inferences that will be drawn from recordings, and so the best policy is to record at the highest practical quality level.

Audio data compression has been researched intensely in the communications industries fueled by applications in telephones, music players, and the internet. Two classes of algorithms have resulted, termed lossy and lossless according to whether the original sampled signal can be recovered precisely from the compressed data. In lossy compressors, a high compression factor (i.e., the input data rate divided by the output rate) is paramount, and some distortion is accepted to achieve this. Lossy algorithms such as LPC, MP3 and AAC (Solomon, 2006) have been developed to deliver speech or music through limited-capacity communication channels with minimal perceptual degradation of the signal. These algorithms take advantage of psychoacoustic masking in human hearing to eliminate signal components that are largely inaudible (Solomon, 2006), but spectrographic examination of the recovered signals reveals substantial distortions and even new signal components (Liu *et al.*, 2008). These artifacts reduce the value of the compressed signal for the kind of detailed analyzes made by marine bioacousticians.

[a)]Author to whom correspondence should be addressed. Also at: Department of Bioscience, University of Aarhus, DK-8000 Aarhus C, Denmark. Electronic mail: markjohnson@st-andrews.ac.uk

In lossless compression, the requirement that the original data are recoverable leads to data-dependent compression factors. Lossless algorithms are used in "ZIP" compression programs to distribute software and documents, but applications in audio compression have been less extensive. One application in which lossless methods have excelled is in the exchange of music on the internet. Although the late Jerry Garcia of the Grateful Dead may have produced few redundant bits, the use of lossless compression in distributing concert recordings of this and other bands has resulted in the preservation of high quality historical archives. The success of a relatively simple lossless compressor, SHORTEN (Robinson, 1994), spurred the development of more sophisticated compressors such as FLAC and WAVPACK, which are now included in media players and multi-format data compression programs. These algorithms share the same basic structure (Hans and Schafer, 2001; Solomon, 2006) compressing sound by first removing the correlation between adjacent samples and then by efficiently coding the residual. But the need to adapt to a wide variety of music increases the complexity of these programs. In SHORTEN, a bank of filters and coders are tried in succession to find the best performer for each sound segment while both FLAC and WAVPACK use adaptive filters and data-dependent coders.

A consequence of the adaptability of lossless audio compressors is that these algorithms achieve surprisingly good compression with underwater sound recordings. But despite this, lossless compression appears to have had little traction among marine bio-acousticians. One reason for this may be the computational demands of algorithms such as FLAC and WAVPACK, which are significant for an embedded processor in a low-power recording instrument. To overcome this limitation, we have developed a low-complexity lossless audio compression algorithm specifically for underwater sound. This algorithm has been in use since 2002 in a miniature sound recording tag for marine mammals, the DTAG (Johnson and Tyack, 2003), but has not yet been reported in the literature. Here we describe the algorithm and evaluate it alongside other lossless compressors using a test set of sounds from drifting and moored recorders and from sound recording tags on cetaceans. The recordings include shallow- and deep-water sound environments sampled at rates from 16 to 240 kHz and so represent a broad cross-section of underwater sound signals. In the following section, we provide a brief review of the techniques used in lossless sound compression and consider the characteristics of underwater sound that make this signal suitable for compression. We then describe the new algorithm in Sec. III. The methods and data sets used to evaluate performance are described in Sec. IV and evaluation results are reported in Sec. V. We conclude the paper by discussing the performance attained by the algorithm and examine the extent to which this depends on the characteristics of the input signals and the recording system.

II. COMPRESSION OF UNDERWATER SOUND

A. Lossless audio compression

Although audio signals are usually complex and nonstationary, they contain two reliable sources of redundancy. First, there are often passages with low signal levels that do not require the full dynamic range of the digital representation and so can be coded with fewer bits. For example, a passage with a peak level of 10% of full scale can be coded with three fewer bits per sample. Second, audio signals are correlated from sample to sample, meaning that a part of the next sample can be predicted from previous samples. If this predictable component is removed, the residual signal will be smaller and so need fewer bits to represent. To take advantage of this redundancy, lossless audio compression algorithms generally incorporate two functional blocks (Hans and Schafer, 2001): A filter and a coder. The filter removes most of the inter-sample correlation by subtracting a prediction of the current sample obtained by filtering prior samples, i.e., by forming a residual signal $y_k = x_k - P(q^{-1})x_{k-1}$ where $P(q^{-1})$ is the prediction filter in the unit delay operator q^{-1}, and x_k is the input signal at sample k. The residual will be less predictable than the input signal and so more like white noise leading to another interpretation of this operation: The filter $W(q^{-1}) = 1 - q^{-1}P(q^{-1})$, which produces y_k from x_k, has the effect of flattening the spectrum of the input signal, leading to the term "whitening filter." As the spectral characteristics of the input signal vary with time, an adaptive filter is used in most compression schemes (Solomon, 2006) and the filter coefficients must be sent along with the residual data to enable decoding. The filtering operation is reversed in the decoder to recreate the original signal and so the filter must be stably invertible.

The residual signal after filtering has lower average power than the input signal and so can be represented with shorter binary words. Thus the second step in the compressor is to re-code the residual signal, replacing each value, y, by a code word with $n(y)$ bits. In an ideal coder, $n(y)$ is equal to the inverse of the probability of y, i.e., values that occur often are coded with few bits (MacKay, 2003). But it is computationally demanding to determine an optimal coder for arbitrary data, and so a fixed (i.e., data-independent) code is often used. This code will be most efficient if its implied distribution, $2^{-n(y)}$, is similar to the probability distribution of the whitened signal (MacKay, 2003). Some commonly used codes are shown in Table I. Rice and Elias codes are variable-length codes that allocate increasing numbers of bits to represent larger values. For example, the number 8 is coded in 17 bits in Rice-0 and 6 bits in Rice-3. As evident from the growth rate of code words with absolute signal level in Table I, each code has a different implied distribution: Rice-0 is effective for signals with low average levels while Rice-1 is better for stronger signals. One other code type, termed block floating point (BFP), is given in Table I. This is a run-length code in which each sample in a block is coded with a fixed number of bits chosen to fit the largest absolute value in the block. For example, if the absolute peak value in a block is 25, each sample can be represented by 6 bits allowing numbers between −32 and +31. The benefit of BFP codes is that the coded data will have no more bits than the input data (excluding the few extra bits needed to indicate the word length in each block). For this reason, BFP is useful for signals with high average levels that are not efficiently coded by low-order Rice codes.

TABLE I. Mappings of variable-length prefix codes for encoding small signed integers. The BFP-2 code is a 3-bit (level level bits and one sign bit) two's complement representation of the numbers −3 to 3. A BFP−3 code would have four bits per code word and cover the numbers −7 to 7. Note that the mappings for the Rice and Elias-gamma codes are biased relative to zero, i.e., negative numbers tend to have shorter code-words.

Number	Rice-0	Rice-1	Rice-2	Rice-3	Elias-gamma	BFP-2
0	1	10	100	1000	1	000
−1	01	11	101	1001	010	111
1	001	010	110	1010	011	001
−2	0001	011	111	1011	00100	110
2	00001	0010	0100	1100	00101	010
−3	000001	0011	0101	1101	00110	101
3	0000001	00010	0110	1110	00111	011
−4	00000001	00011	0111	1111	0001000	-
4	000000001	000010	00100	01000	0001001	-

Although the filter-coder structure is a common feature of lossless audio compressors, the way these are implemented varies (Hans and Schafer, 2001). Two popular algorithms, FLAC and WAVPACK, both use adaptive whitening filters followed by a coder and so are capable of tracking changing noise spectra. In FLAC, the filter is selected on a block-by-block basis from a battery of varying-length fixed and data-derived FIR filters (Solomon, 2006). The residual is coded with a varying-order Rice code. In WAVPACK, a single tap adaptive filter is used but the filter is passed over the data multiple times. The residuals are coded using Golomb codes, a super-set of the Rice codes (Solomon, 2006). The success of each method depends upon how well it is able to match the spectral and distributional characteristics of the input signal over time. In the following section, we consider what these characteristics are for underwater sound and the implications for compressor design.

B. Characteristics of underwater sound

The average underwater soundscape is dominated by noise from wind, surf, rain and, in many areas, distant shipping (Richardson *et al.*, 1995; Dahl *et al.*, 2007). Even though animal calls or discrete boat traffic are frequent in some areas, the duty-cycle of these noise sources, i.e., the proportion of time that they are detectable above the ambient noise, is generally low. For example, sounds from distant sperm whales are heard regularly in some deep-water habitats but, with a clicking rate of around 0.5 Hz (Madsen *et al.*, 2002) and a received call duration of a few milliseconds (Møhl *et al.*, 2003), the duty cycle of these transients is small. Choruses of fish and crustacea (e.g., snapping shrimp) may have a higher duty cycle but have a low average power. Thus for underwater sound, it is essential to focus on efficient compression of the ambient noise rather than specific sound sources.

Although ambient noise in the world's oceans varies temporally and spatially, its spectral characteristics have a consistent low-frequency emphasis as summarized in the Wenz curves (Wenz, 1962). This characteristic is partly a result of the increasing absorption of sound at higher frequencies. At low frequencies, sound absorption is approximately linear with frequency contributing to a roughly 4–6 dB/octave decrease in ambient noise from 100 Hz to about 5 kHz. Above this frequency, sound absorption increases more rapidly and the noise floor drops to a low at about 30 kHz before increasing gradually at higher frequencies due to thermal noise (Dahl *et al.*, 2007). In quiet areas, the ambient noise floor at 30 kHz is so low that it is challenging to build sound recorders capable of recording it, at least with low operating power. The system noise of the recorder may then dominate the background noise level above about 10 kHz. System noise arises from shot and Johnson noise generated by electronic components as well as quantization noise due to the finite-precision digital representation of sound. These noise sources are spectrally flat above a few hundreds of hertz. Thus from the viewpoint of compression, underwater sound generally comprises two components: (i) occasional transient signals that represent a small fraction of the average power and (ii) a slowly varying noise floor with a low-pass characteristic up to about 10 kHz and a flat spectrum above this. In the following sections, we show how these characteristics of underwater sound lead to a simple but effective compression algorithm.

III. A LOW-COMPLEXITY LOSSLESS COMPRESSION ALGORITHM

In designing a compression algorithm for underwater sound, our objective was to attain a moderate level of compression with very low processing effort. The resulting algorithm is dubbed X3 because it generally gives a compression factor of three or more. Like other audio compressors, X3 operates on blocks of data and comprises a whitening filter followed by a coder. But to minimize computation, the algorithm uses a non-adaptive filter and selects a coder from a limited selection of codes that can be implemented by look-up tables. Although other filters could be used, we will show that a first-order difference is effective in whitening underwater noise over a range of sampling rates. The lack of adaptation in the whitening filter is offset by allowing the coder to change from block to block, but the selection is made simply according to the magnitude of the signal in the block rather than via a computationally expensive best-fit procedure. The encoding and decoding algorithms for 16-bit audio data are described in the following sections. Parameter choices and extensions to other resolutions are discussed in later sections.

A. Encoder

The X3 encoder operates on blocks of N samples, performing a sequence of three operations on each block (Table II): (i) filtering using a first-order difference, i.e., $y_k = x_k - x_{k-1}$, where x_k is the kth sample of the incoming data and $k = 1,\ldots, N$, (ii) finding the largest absolute value, M, in the block of filtered data, i.e., $M = \max_k(|y_k|)$, and (iii) selecting and applying an appropriate coder given M. The filter is initialized using the last sample of the previous block as x_0 so that the filter effectively runs continuously from block to block. To select a coder, M is compared to three pre-determined thresholds, T_{1-3}. If $M \le T_1$, a Rice-0 encoder is chosen. If $T_1 < M \le T_2$, the coding is Rice-2,

TABLE II. Pseudo-code for the X3 encoding algorithm. Vector V contains the last input sample of the previous block along with the next N input samples. The function bit_write(n, x) expresses the number x in n bits and packs these to the output stream. If the number is signed, it is treated as an n-bit two's complement number. Function next_log2(x) returns the minimum number of bits required to represent a positive number, x. An additional bit is required to represent signed numbers between −x and x.

```
function    X3_encode(V,N)                  V is the data vector to be compressed; V(0) is filter state.
                                              N is the number of samples
M = 0
for k = 1:N,                                Filter the N samples and find max absolute value
  D(k) = V(k) − V(k − 1)                    Apply the whitening filter
  M = max(abs(D(k)), M)                     Find the largest abs filtered value
end

if M < T1,                                  Check if the block can be Rice-0 encoded
  bit_write(2, 1)                           If so, write the Rice-0 block header
  for k = 1:N,                              Code N samples using the Rice-0 code tables
    x = CODE0_tab(D(k))                     Lookup the Rice code for the kth sample
    n = LEN0_tab(D(k))                      Lookup the code length for the kth sample
    bit_write(n, x)                         Write the result to the output stream
  end

elseif M < T2                               Check if the block can be Rice-2 encoded
  bit_write(2, 2)                           Write the Rice-2 block header
  :                                         Code N samples using the Rice-2 tables
elseif M < T3                               Check if the block can be Rice-3 encoded
  bit_write(2, 3)                           Write the Rice-3 block header
  :                                         Code N samples using the Rice-3 tables
else                                        Otherwise, this is a BFP or pass-through block
  E = next_log2(M)                          Find the number of bits to use per code word
  bit_write(6, E)                           Write the BFP block header
  if E < 15,                                If E is less than 15 bits, use BFP encoding
    for k = 1:N,
      bit_write(E + 1, D(k))                Write (E + 1)-bit signed samples to the output stream
    end

  else                                      Otherwise, use pass-through encoding
    for k = 1:N,
      bit_write(16, V(k))                   Write 16-bit signed samples to the output stream
    end                                     using the unfiltered data
  end
end
```

where the code and length tables are:

CODE0_tab = [… 1, 1, 1, 1, 1, **1**, 1, 1, 1, 1,…] LEN0_tab = [… 8, 6, 4, 2, **1**, 3, 5, 7, 9,…]

CODE2_tab = [… 7, 5, 7, 5, **4**, 6, 4, 6, 4,…] LEN2_tab = [… 4, 4, 3, 3, **3**, 3, 4, 4, 5,…]

etc. from Table I. The lookup indices, D(k), are integers, so these tables are bi-directional. The value at the 0th index of each table is shown in bold and values to the left of this are accessed with negative indices, e.g., CODE2_tab(−2) = 7.

while Rice-3 is used if $T_2 < M \leq T_3$. If $M > T_3$, a BFP encoder is used. The BFP encoder finds the exponent, E, of M (i.e., the number of bits needed to represent M) and then left-truncates each filtered sample to $E+1$ bits (i.e., E bits plus a sign bit). If the input signal is close to full scale, it may not be possible to represent the output of the whitening filter in 16 bits (i.e., if $|x_k - x_{k-1}| > 32\,767$). To handle this, the unfiltered data are stored verbatim when $E = 15$ or if the subtraction overflows, and this is termed pass-through coding.

The compressed data block is represented by a header defining the coder followed by the packed output from the coder. Because of the low average level of underwater sound, Rice coders are likely to be selected far more often than the BFP coder, and so a short block header is used for Rice codes. A 2-bit header with values 1, 2, and 3 indicates Rice codes 0, 2, and 3, respectively. A header value of 0 indicates a BFP or pass-through block in which case the exponent, E, is coded in the following 4 bits to make a 6-bit header. Pass-through coding is indicated by $E = 15$.

Given that the whitening filter and the coders are fixed, the X3 algorithm has only four user-selected parameters: The block length, N, and the coder transition thresholds T_{1-3}. These could be selected based on the sampling rate and the expected ambient noise level with respect to the least-significant bit, but we show later that fixed parameter values work well over a wide range of conditions. Of the four parameters, only N is required to decode X3 data, and so this must be included in an information block accompanying each X3-encoded file.

B. Framing

As defined in the preceding text, each X3 data block comprises a code-selection header (2 or 6 bits long) followed

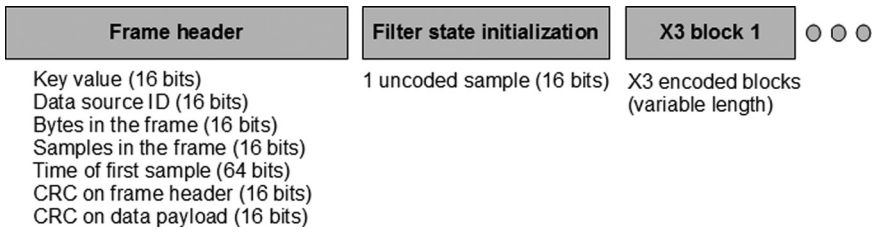

FIG. 1. Frame structure required for non-sequential decoding of X3 data. Frames can contain multiple X3-encoded blocks with no additional headers other than the 2 or 6 bit code selection header in each block. A known key value is placed in the frame header to help locate frames when searching through the recording. The 64-bit time value should have a resolution of at least one count per sampling rate to facilitate searching and the detection of missing samples.

by a variable number of bits encoding the N input samples. Consecutive X3 blocks can be concatenated in a file without additional headers. However, because of the varying block length, it is not possible to decode X3 data from a random starting point. Instead blocks must be decoded consecutively. To enable random access and to allow recovery from data errors (e.g., as can occur if Flash memory is used for data storage), groups of blocks are framed with a header defining the size of the data frame (Fig. 1). Decoding software can then skip from frame to frame until the required data segment is located. A 20-byte frame header is used containing the start time of the first sample, a data source identifier (allowing multiple data sources to be merged into a single file), the number of bytes and samples in the frame, CRC parity checks (Koopman and Chakravarty, 2004) on the header and data, and a key value to simplify searching for frame headers. One such header can frame several thousand samples (i.e., hundreds of blocks) keeping the overhead low.

TABLE III. Pseudo-code for the X3 decoding algorithm. The input is a stream of packed binary data containing a block of N encoded samples. The decoded output is assembled in vector V. At the start of the routine, the 0th element of V is assumed to contain the last output value from the previous decoded block (this is the filter state). Function bit_read(n) returns the next n bits from the input stream and signed_bit_read(n) reads the n bits as a signed number. Function skip_zero_bits() returns the number of 0-valued bits in the input stream until the next 1.

```
function V = X3_decode(V,N)                 V is the output data vector; V(0) is the filter state
                                              N is the number of samples in the block
c = bit_read(2)                             Read the block header
if c==1,                                    If this is a Rice-0 encoded block
for k = 1:N,                                For N samples
    n = skip_zero_bits()                    Find the # of leading zeros in next code word
    bit_read(1)                             Discard the leading 1
    d = INVRICE(n)                          Table lookup to convert to a signed number
    V(k) = d + V(k − 1)                     Invert the whitening filter
  end

elseif c==2,                                If this is a Rice-2 encoded block
  for k = 1:N,                              For N samples
    n = skip_zero_bits()                    Find the # of leading zeros in next code word
    bit_read(1)                             Discard the leading 1
    r = bit_read(2)                         Read the 2-bit suffix of the code word
    d = INVRICE(r + 4 n)                    Table lookup to convert to a signed number
    V(k) = d + V(k − 1)                     Invert the whitening filter
  end

elseif c==3,                                If this is a Rice-3 encoded block
  Decode N samples with a 3 bit suffix as above
else                                        This is a BFP or pass-through block
  E = bit_read(4)                           Read the rest of the block header
  if E == 15,                               If this is a pass-through block
    for k = 1:N,
      V(k) = signed_bit_read(16)            Read 16-bit words from the input
    end                                     No inverse filtering is needed

  else                                      Otherwise, this is a BFP-encoded block with
    for k = 1:N,                            E + 1 bits/word
      d = signed_bit_read(E + 1)            Convert the next E + 1 bits to a signed number
      V(k) = d + V(k − 1)                   Invert the whitening filter
    end
  end
end
```

where:INVRICE = [0,−1,1,−2,2,−3,3,−4,4,…] is a conventional table with indices 0, 1, 2,… The same lookup table is used for all Rice codes.

One other piece of information is needed to allow a frame of data to be decoded without reference to previous frames: The initial state of the filter used to whiten the data. Although the filter state could be reset to zero at the start of each frame, another alternative is to simply provide the first sample of each frame as an uncoded value (i.e., with pass-through coding). This can then be used to initialize the filter state for the remainder of the frame.

C. Decoding

An X3 frame can be decoded block by block using the pass-through sample at the start of the frame to initialize the filter state. Successive blocks are then decoded using the algorithm in Table III. For each block, the first two bits are inspected to determine which coder was used. For Rice-coded blocks, N samples are then decoded using the self-punctuating feature of these codes to identify each code word (MacKay, 2003). Code words are restored to signed values using a look-up table and then passed through the inverse of the whitening filter (i.e., an integrator, $y_k = y_{k-1} + x_k$) to recover the original data. For BFP-coded blocks, the additional four header bits are read to determine the word length, E, and then the N samples are read from the input stream as $(E+1)$-bit words. If E is 15, the block was uncoded and no further processing is needed. For $E < 15$, the code-words are restored to 16-bit values by sign extension and then passed through the integrator to reverse the whitening filter.

A drawback of using a differentiator for the whitening filter is that the inverse filter, an integrator, has infinite memory. This means that a bit error in the compressed data will cause every subsequent decoded value to be incorrect. The integrator state is re-initialized at the start of each frame so decoding errors will not pass beyond a frame. However, it is not generally possible to recover the data within the affected frame, suggesting that frames should be kept short if occasional data errors are possible. Frames with bit errors will be detected by performing a parity check.

IV. PERFORMANCE EVALUATION

A. Test data

The performance of the X3 algorithm was evaluated using a set of underwater sound samples covering a range of sampling rates and ambient noise conditions. The test samples were selected from archives of on-animal and far-field recordings (Table IV). Although most recordings were made near cetaceans, the soundscapes also include boats, fish-finders, surf, and crustaceans. The sampling rates extend over more than an order of magnitude (16–240 kHz), and the samples were recorded in shallow and deep water environments. Sample lengths of 5 min were chosen to make it practical to evaluate algorithms with a variety of settings but no special criteria were used to select samples from the raw recordings. All sound samples are available at http://soundtags.st-andrews.ac.uk where shorter segments (the first 15 s of each file) are also available.

On-animal recordings (Table IVa) were made by DTAG sound recording tags attached with suction-cups to the dorsal surface of cetaceans (Johnson and Tyack, 2003). These tags included a single-pole high-pass filter and a two-pole low-pass filter prior to 24-bit sigma-delta analog-to-digital converters (ADCs). According to the datasheet (Cirrus Logic Inc., 2006), the ADCs have a minimum dynamic range of 93 dB implying an effective number of bits of <16 (Orfanidis, 2010, also see Sec. VI) and so only the most significant 16 bits of the 24-bit samples were stored.

Off-animal recordings (Table IVb) were made with DMON digital acoustic monitors (Woods Hole Oceanographic Institute, Woods Hole, MA) suspended below drifting buoys (El Hierro and Pico Island) or floating above an

TABLE IV. Data sets used to evaluate the X3 algorithm. All recordings have 16-bit resolution. The clipping level is the maximum sound level in dB re 1 μPa that can be represented by the recorder. Samples GR48 and GI60 were produced by decimating 96 and 120 kHz recordings by 2 in MATLAB (Version 7, Mathworks Inc.) and then restoring the filtered data to integer values.

(a) On-animal recordings (DTAG)

Sample	Species	Recording depth (m)	Water depth (m)	Sampling rate (kHz)	Bandwidth (kHz)	Clipping dB re μPa	Activity
LI192	Cuvier's beaked whale (Liguria, Italy)	730	2000–3000	192	0.5–80	172	Foraging by echolocation
NO96	Sperm whale (Norway)	1170	1500–2000	96	0.5–47	182	Foraging by echolocation
GR48	Humpback whale (Greenland)	150	200–300	48	0.5–20	182	Lunge feeding

(b) Moored or drifting recorders (DMON)

Sample	Location	Recording depth (m)	Water depth (m)	Sampling rate (kHz)	Bandwidth (kHz)	Clipping dB re μPa	Sounds
PI240	Azores (Pico Island)	650	1500	240	1–100	182	Sperm whale
EH120	El Hierro (Canary Is.)	200	1500	120	0.1–51	170	Blainville's beaked whale, small boat, 50 kHz fish-finder
GI60	Goat Island (New Zealand)	5	10	60	0.1–26	170	Reef sounds, snapping shrimp, surf
GI16				16	0.01–7	170	

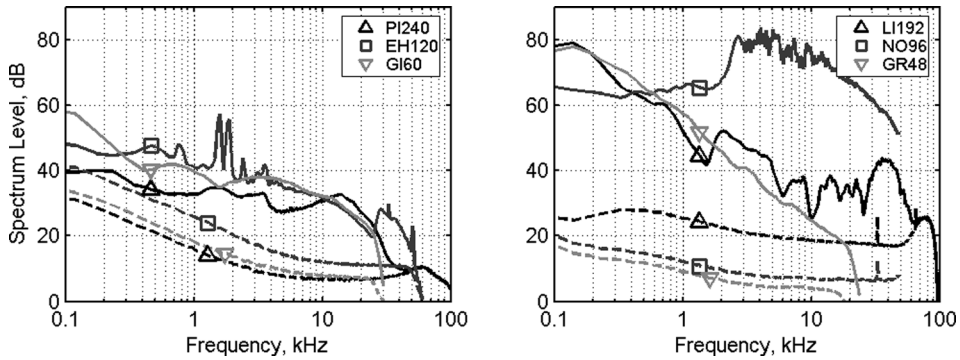

FIG. 2. Spectrum levels (i.e., power in 1 Hz bands in dB) of the test recordings relative to the 16-bit quantization noise power. The RMS quantization noise for 16-bit representation referred to a full-scale of ± 1 is $1/(2^{15}\sqrt{12})$. The moored and drifting recordings are shown in the left panel, and on-animal recordings are in the right panel. The solid lines show the spectrum level averaged over the 5 min samples while the dotted lines show the system noise level for each recording. System noise measurements were made with the device in a quiet room (tags) or by bypassing the hydrophone with a fixed capacitor equal to the capacitance of the hydrophone (moored/drifting recorders). GI16 is similar to GI60 over 0.1–8 kHz and is not shown.

anchor (Goat Island). The DMONs included a single-pole high-pass filter and a seven-pole Butterworth anti-alias filter prior to 18-bit successive approximation ADCs. The datasheet dynamic range for the ADCs is 93 dB (Analog Devices Inc., 2007) and so only the most significant 16 bits of each sample were retained. Signals were initially sampled at high rates (80, 240, or 480 kHz) and then decimated by a factor of 2–5 in the recorders using 36 - to 42-tap symmetric FIR filters producing the sampling rates given in Table IVb.

The power spectrum of each test sample is shown in Fig. 2. Sound samples from the moored/drifting recorders show the expected trend of decreasing energy at higher frequencies. Local noise sources provide some additional spectral features, e.g., boat noise (0.5–4 kHz) and an echosounder (50 kHz) in the EH120 sample. The low-frequency emphasis is also apparent in two of the on-animal recordings (LI192 and GR48) with water flow over the tag contributing to the low-frequency (i.e., <1 kHz) energy. High signal levels from sperm whale clicks dominate the NO96 spectrum even though these occupy only a small fraction of the recording time.

The occurrence rate of strong transients in the recordings influences their compressibility. Log-survivor plots of the minimum word length needed to represent the data in each recording (Fig. 3) provide an indication of how often transients occur. These plots are produced by finding the peak magnitude in each block of 50 samples and then converting this to a bit count, an operation that is effectively the same as BFP compression with no whitening filter. The log-survivor plots show the proportion of blocks that require at least a given number of bits per word. The high-sampling rate off-animal recordings contain few strong transients and so require fewer than 6-8 bits/word to code most blocks (i.e., a compression factor of 2-2.7 using filter-less BFP encoding). The low-frequency reef recording (GI16) requires about 11 bits/word because of frequent transients from surf noise. Flow-noise and calls from nearby animals in the on-animal recordings also lead to higher bit counts (e.g., 9–11 bits/word for LI192 and GR48) with more than 12 bits/word required to code blocks in NO96 that contain clicks from the tagged sperm whale.

B. Performance tests

The compression performance of X3 was compared to four public-domain lossless compression programs (Appendix). Two of these (LZ77 and PPMd) are intended for general data compression and form part of popular ZIP compression tools. These algorithms compress data by replacing commonly occurring word sequences by pointers into a library. A variant of LZ77 is the usual algorithm employed when compressing a file on a PC. The newer

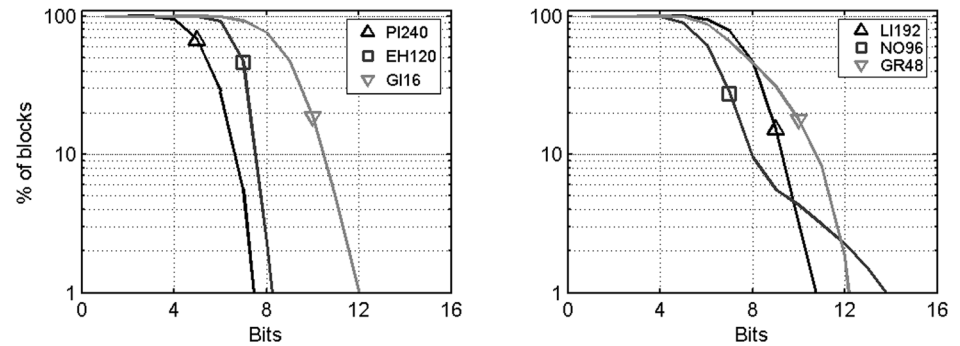

FIG. 3. Magnitude distributions of the test recordings shown as the log-survivor plots of the peak signal magnitude in 50-sample blocks, an indication of how frequently large signals occur in the data. Magnitude is expressed in bits, and the points on the curves indicate the percentage of blocks that require at least n bits per word to code, where n is the x-axis value, e.g., for GI16, $<20\%$ of blocks need 10 or more bits/word. The log-survivor plot for GI60 is similar to that for EH120 and is omitted for clarity.

PPMd (prediction by partial matching) algorithm codes the residuals after prediction using a dynamic library and so is a generalization of the filter-coder methods used in lossless audio compression. The two other methods evaluated (FLAC and WAVPACK) were specifically designed for audio compression as described earlier. The settings used with the public-domain algorithms are listed in the Appendix. For X3, a block length of 50 samples was used, and the code transition thresholds were set so that blocks with magnitude between 0 and 3 were coded with Rice-0, between 4 and 8 with Rice-2, between 9 and 20 with Rice-3, and above 20 with BFP.

Unlike FLAC and WAVPACK, X3 uses a fixed filter and so cannot adapt to the ambient noise spectrum. To examine the adequacy of the first-order differentiator used in X3, we estimated the compression performance that would have been attained with filters optimized to each test sample, assuming an ideal coder. These filters were least-square-error linear predictors calculated from the autocorrelation function of each test sample (Jackson, 1996). To focus the predictors on the ambient noise rather than infrequent but strong transients, the autocorrelation was only calculated on data segments of 1024 samples that had RMS signal levels <0.1% of full scale. The filters were implemented using floating point computations, but the residuals were restored to 16-bit integers for performance evaluation. The compression attainable with each filter was calculated from the estimated entropy, H, of the filtered signal:

$$H \approx \sum_{k=-32768}^{32767} -\frac{n_k}{N_s}\log_2\left(\frac{n_k}{N_s}\right) \quad (1)$$

where n_k is the number of occurrences of value k and N_S is the number of words in the sound sample (MacKay, 2003). The entropy is the number of bits/word needed to code the filtered data with an ideal coder. This was converted to a compression factor by dividing the incoming word length (16 bits) by the entropy.

V. RESULTS

A. Performance comparison

The compression factors obtained with the public-domain and X3 algorithms are given in Table V for each test sample. As expected, the LZ77-based GZIP algorithm performs relatively poorly giving compression factors of about 2, similar to those of a filter-less ideal coder (column 3 in Table V). LZ77 was designed primarily for text compression and so lacks the whitening filter that is critical for effective audio compression. FLAC and WAVPACK, which were expressly designed for audio, provide consistently higher compression factors (2.9–7.4) with FLAC performing better overall. Surprisingly similar results were obtained using PPMd, indicating that the predictive function of this algorithm is effective at decorrelating the sound prior to coding. The performance difference between these three methods is most pronounced for the PI240 sample. This sample has a very low average level to which FLAC and PPMd were able to adapt giving compression factors of more than 7 (i.e., just over 2 bits per 16-bit sample). The Golomb coder used in WAVPACK may be less capable of producing short codes resulting in a 5.4 compression factor.

TABLE V. Performance comparison of X3 against public-domain lossless compression algorithms. Shown are the measured compression factors (i.e., size uncompressed/size compressed) attained by each algorithm averaged over the 5 min data sets. The "No filter" column gives the theoretical maximum compression factor achievable without a whitening filter, calculated from the entropy of the signal [Eq. (1)].

Sample	File size (MB)	No filter	X3	FLAC	WAVPACK	PPMd	GZIP
GI16	9.2	1.6	3.1	3.2	2.9	3.0	1.8
GR48	27.4	1.9	3.4	3.9	3.9	3.3	2.0
GI60	34.3	2.5	3.0	3.1	3.1	3.0	2.2
NO96	54.9	2.3	4.0	4.2	4.4	4.3	2.6
EH120	68.8	2.5	3.8	4.0	3.9	4.0	2.6
LI192	109.8	1.9	3.5	4.0	3.8	3.6	2.0
PI240	137.6	2.8	7.2	7.4	5.4	7.5	4.6

The much simpler X3 algorithm gave compression factors of 3.0–7.2 with the test samples, attaining more than 87% of the compression obtained with FLAC. The code bank used in X3 was able to respond to both the low signal levels in PI240 and the high average levels in GI16 to give consistently high compression despite the non-adaptive algorithm. In the following, we explore how the design of the algorithm enables this high performance with underwater sound.

B. Whitening filter

A first-order differentiator was selected for the whitening filter in X3 because of its computational simplicity and because it roughly corrects the typical 4–6 dB/octave high frequency roll-off in underwater ambient noise. However, the test samples have considerably more complex noise spectra due to a mixture of animal sounds and ambient and anthropogenic noise (Fig. 2). Whitening filters matched to each data set might therefore be expected to enable better compression than the fixed first-order filter. Figure 4 shows

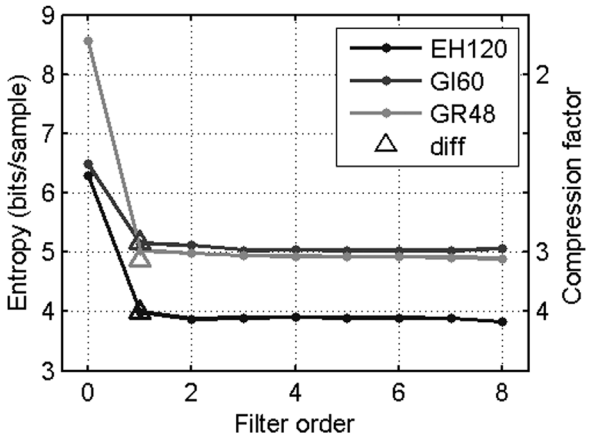

FIG. 4. Compressibility of three test samples with different whitening filters. The lines show the entropy (left-hand axis) and potential compression factor (right-hand axis) when the test samples are filtered with a whitening filter optimized for each test sample. The filters are 1st- to 8th-order least-squared-error predictors. The points at filter order 0 indicate the performance with no filter. The triangles indicate the performance with a fixed first-order differentiator.

the entropy of three of the test samples after filtering with whitening filters of up to eighth order matched to each test sample. Similar results were obtained with all seven test samples. As anticipated, there is an enormous benefit in filtering, and this is evidenced by a step change in entropy for filter orders >0. However, there is little additional benefit in using a high-order filter, even if it is tailored to the data, a conclusion that has also been made for music compression (Hans and Schafer, 2001). Using a fixed differentiator resulted in entropies within 5% of those attained with 8th order filters matched to each test sample, justifying the use of this much simpler filter in X3.

C. Coding efficiency

In X3, the coder used for each block of data is selected according to the peak signal level in the block. The coding efficiency of this method is very high, at least for the test samples, as can be seen by comparing the theoretical compression factors (Table VI, 3rd column) with the measured compressions obtained with X3 (Table V, 4th column). The X3 compression factors are within 5% of those predicted from entropy. In two samples (GR48 and NO96), X3 even narrowly exceeds the theoretical compression factor. This is a consequence of the block-oriented coding in X3: In block coding, a different code-book can be used for each block whereas the entropy calculation assumes a single code-book for the entire data sequence.

The code set used in X3 comprises three variable-length codes optimized for small (Rice-0) and medium (Rice-2 and Rice-3) signal levels. Blocks with high signal levels are compressed using a fixed-length code (BFP). For most underwater soundscapes, high signal levels are rare, and so the Rice codes will be used most often. In the test samples, BFP coding was used in <10% of blocks in four of the seven samples (Table VI, column 2). For these samples with low transient rates, it is critical to select a Rice code matched to the signal level in each block. To demonstrate this, Table VI shows the compression factors obtained using just one fixed code for variable-length blocks instead of the three codes available in X3. For each test sample, the best performing code provides most of the compression attained by X3, but the best code varies across the test samples, roughly in proportion to the average power in the whitened signal: Rice-0 performs best for PI240, the recording with the lowest signal level, but for the other recordings, higher-order codes provide better results. Thus automatic selection from a range of codes is a key factor enabling high compression over a variety of sampling rates and soundscapes.

TABLE VI. The compression factor achieved by individual coders for each test sample using a first-order differentiator as a whitening filter. For test samples with high average power, a high fraction of block floating point (BFP) blocks is needed, and the choice of variable-length code has correspondingly less influence on the overall compression factor. The "Entropic" column is the theoretical maximum compression factor derived from the entropy of the filtered signal. Shaded cells indicate the code that gives the highest realized compression factor for each sample. Codes marked * are not used in the X3 compressor and are provided for comparison.

Sample	Percentage BFP	Entropic	Elias-δ*	Rice-0	Rice-1*	Rice-2	Rice-3
GI16	48	3.2	2.6	2.0	2.6	2.9	2.9
GR48	25	3.3	3.0	2.4	3.1	3.4	3.2
GI60	64	3.1	2.6	2.0	2.5	2.8	2.8
NO96	8	3.9	3.7	3.5	4.1	4.0	3.5
EH120	2	4.0	3.3	2.5	3.6	3.9	3.6
LI192	6	3.6	2.8	1.9	2.9	3.4	3.4
PI240	0	7.6	6.4	7.2	6.4	5.1	3.9

D. Implementation

The X3 algorithm was originally designed for DTAG sound recording tags (Johnson and Tyack, 2003), which use Texas Instruments low-power digital signal processors (DSPs, part number TMS320VC5509A) to collect and store data. The X3 encoder has been implemented in a mixture of C code and assembly language on this processor while the decoder has been implemented in C to run on a PC. MATLAB and C functions for the encoder and decoder are available under the Gnu Public License and can be downloaded from: http://soundtags.st-andrews.ac.uk. The encoder implementation is streamlined by the parallel data paths, conditional arithmetic operations, and barrel shifter in the DSP chip. The average operation count for the algorithm is 8.2 CPU cycles per 16-bit input sample excluding function calls, which are anyway amortized over the block length. For 192 kHz single-channel 16-bit data, the complete algorithm requires 1.6 MOPs (million operations per second). Based on published power consumption figures for the processor (Bhatnagar, 2008), this operation rate adds a power consumption of approximately 0.9 mW if data and code are in internal memory amounting to <2% of the approximately 50 mW total power consumption of the DTAG.

In comparison, FLAC requires an estimated 95 CPU cycles to encode each 16-bit input sample when implemented on the same processor using assembly language and C code. This means an order of magnitude higher power consumption translating into a 20% reduction in battery life on the DTAG. A lower compression option (the −1 option) in FLAC reduces the operation count to 47 cycles/sample at the expense of a 2%–14% reduction in compression factor in the test samples. In addition to operation count, a significant factor for embedded implementations is the code size. The FLAC encoder involves about 16 500 lines of C code whereas X3 is implemented in <1000 lines. Although code length depends to some extent on programming style, a smaller program uses less processor resources and is easier to port and maintain. The operation counts for FLAC and X3 do not include CRC computation, which adds about 3.5 CPU cycles per compressed byte (i.e., one to two cycles per input sample) on the TI DSP using a lookup table implementation (Sarwate, 1988). The FLAC operation count includes MD5 signature calculation (a type of checksum used to uniquely identify recordings) which is not essential but is an integral part of the FLAC standard.

VI. DISCUSSION

Our objective has been to develop a low-complexity but effective lossless compression algorithm for underwater

sound. The method we propose is not especially novel as similar techniques are embedded in several popular public-domain audio compressors. However, these complex algorithms designed for music compression include a battery of compression strategies that are applied adaptively to track changing sound spectra and levels. This adaptation process makes the algorithms computationally demanding and so less suitable for low-power recording systems. Here we show that for a broad range of sampling rates and recording conditions, a simpler non-adaptive compressor provides reliable compression factors of three or more, attaining >87% of the compression of the more complex algorithms. In the following, we discuss the performance and limitations of such a simple approach and consider whether additional improvements may be possible.

A. Algorithm performance

Two assumptions about underwater sound underpin the design of the X3 algorithm. The first is that high-level transients, e.g., from animal calls or boats, are relatively rare and that the algorithm should therefore be optimized for ambient noise. The second assumption is that the ambient noise has a stable low-frequency emphasis that can be whitened with a low-order non-adaptive high-pass filter. The performance of the resulting algorithm with a range of test samples suggests that these assumptions are apt despite variability in the average power spectra across the samples (Fig. 2). The first-order differentiator used as a whitening filter is, in effect, a predictor in which the next sample is predicted by the current sample. The prediction error is then encoded in the second step of the algorithm. This simple predictor removes a large portion, but by no means all, of the correlation between adjacent samples. The remaining inter-sample correlations contribute to the performance loss of X3 as compared to FLAC. But the high compression factors nonetheless attained suggest that compression performance is quite robust to imperfect whitening. Undoubtedly there are limits to this, but we argue that the relative homogeneity in underwater ambient spectra creates a special class of sounds that can be effectively compressed without recourse to complex adaptive methods.

The test samples vary in average level as well as spectra. Although X3 is not adaptive, it is able to respond to different sound levels via code selection. The test samples include one with very low average levels (PI240) while three others had high levels due to snapping shrimp (GI60 and GI16) or flow-noise during lunge feeding (GR48). The inclusion of efficient coders for low and high signal levels in the algorithm allows it to adjust to these extremes, offering similar performance in each case to adaptive compressors.

Although we have selected test samples that span a range of sampling rates and recording conditions, the samples were produced by only two types of recording instruments. Different audio circuits and ADCs are used in these two devices but both were designed with the same general considerations. It is possible then that performance may differ when the algorithm is applied to other recording systems and environments. However, the consistent similarity between X3 and FLAC on the test samples suggests that the performance of X3 on other data can be predicted from the compression factor obtained with FLAC.

B. Parameters

Four parameters control the performance of X3: Three code-transition thresholds, T_{1-3}, and the block length, N. The code-transition thresholds are used to determine which coder to apply in each block. For computational simplicity, a sub-optimal selection criterion is used based on the maximum absolute value in the block rather than the distribution of signal levels. This assumes that blocks with a single outlier are rare and, in general, the peak value should be a fair indicator of the level distribution in a short block of samples. The performance of X3 in comparison to the more powerful coding methods used in FLAC and WAVPACK suggest that this is a reasonable assumption for underwater sound. The specific thresholds used (i.e., 3, 8, and 20) were arrived at by trial and error but are close to the best performing thresholds found in a simulation study using integerized Gaussian white noise with varying standard deviation. There may be little benefit, therefore, in changing these settings for different recording situations.

The block length controls the ability of the algorithm to track changing sound levels. If N is large, there is a high chance that a sound transient will occur in a given block meaning that a coder optimized to high signal levels will be chosen even though most of the block may contain low signal levels. In other words, the peak level becomes a poor proxy for the average levels in a block as N increases. This suggests using a small N for efficient compression but with very low N, the block header limits the overall efficiency. X3 uses a 2-bit header in the majority of blocks (i.e., those that are Rice encoded) allowing block lengths as short as 20 samples with only some 2% header overhead, enabling the algorithm to respond efficiently to changing sound conditions.

C. Resolution and gain

The X3 algorithm, as given in Table II, is intended for data with 16-bit resolution. This is a common word length, but the method can be readily adapted to shorter or longer words. To optimize performance with different word lengths, the code-transition thresholds and the BFP header length may need to be adjusted but few changes are otherwise required. The resulting compression factor, however, may change substantially: Longer word lengths will give poorer compression in any lossless compressor. As ADCs with 18–24 bits of resolution are increasingly being used in sound recorders, it is thus worth considering how many of these bits should be kept.

In underwater recordings, we are often interested in weak signals just above the ambient noise floor as well as occasional high level signals from nearby animals, necessitating both a low system noise and a high dynamic range. Dynamic range, DR, is defined as the power of the strongest sine-wave signal possible without clipping divided by the total noise power over the 0-to-Nyquist frequency band

(Madsen and Wahlberg, 2007). An ideal n-bit ADC generates noise due to its quantized representation of the input signal giving a DR of $1.8 + 6n$ dB (Orfanidis, 2010), suggesting that a 24-bit converter has a DR of 145 dB. But the dynamic range of real ADCs is typically less than the value predicted from the bit count. Low-power (i.e., $\ll$100 mW) audio ADCs suitable for battery-powered equipment almost invariably have a broadband dynamic range of $<$100 dB, irrespective of the number of bits and the type of ADC (successive approximation or sigma-delta). To add to the confusion, audio ADCs are often specified with an A-weighting (Dahl *et al.*, 2007) and the full-band (i.e., unweighted) dynamic range can be substantially lower than the value given in the data sheet. Thus a 24-bit 96 kHz ADC with 102 dB A-weighted dynamic range will likely have a full-band dynamic range of about 96 dB and so produces 7–8 bits that are below the ADC noise floor. These bits are essentially random and so limit the overall compressibility of the data stream without adding useful resolution.

No matter how the data will be processed, there is no benefit in recording more bits than are needed to match the DR of the ADC. Specifically, truncating the data at $DR/6 + 1$ bits, i.e., one more bit than implied by the dynamic range, will add $<$1 dB to the system noise floor from quantization. Thus 16- or 17-bit samples are usually appropriate for underwater sound recordings (Madsen and Wahlberg, 2007). Two options can then be considered when setting the recording gain (i.e., the gain of the preamplifier before the ADC). The lowest noise is obtained without sacrificing much DR by setting the gain so that the preamplifier noise is about 6–10 dB above the ADC noise floor over the frequency range of interest. Using a higher gain than this will only generate more bits of noise, reducing the DR and the compressibility of the data with negligible improvement in signal-to-noise ratio. At the other extreme, the best dynamic range and compressibility is obtained at the expense of signal-to-noise ratio by using a lower preamplifier gain such that the system noise floor is dominated by the ADC noise rather than the preamplifier noise, i.e., ADC noise is about 6–10 dB above the preamplifier noise. Even lower gain settings will reduce sensitivity without increasing DR. There are usually some 12–15 dB between the high gain setting which gives the lowest noise and the low gain setting that maximizes DR.

D. Multi-channel recordings

The X3 algorithm operates on a single channel of data and so must be applied to each channel separately in a multi-channel recorder, raising the question of whether a joint compression method might improve performance. Both FLAC and WAVPACK incorporate cross-channel prediction to take advantage of redundancy in stereo recordings of music. Similar redundancy should be present in signals from hydrophones that are placed close enough together. However, joint compression of multi-channel underwater sound may actually provide little advantage to justify the extra computational effort. Transients will be well correlated between channels but are typically infrequent making them an unrewarding target for compression effort. Low-frequency ambient noise will also be highly correlated between channels but is already de-emphasized by the whitening filter in the compressor without the need for cross-channel decorrelation. As a result, most of the coding effort in an audio compressor is invested in the residual high frequency noise that is dominated by system noise and so is uncorrelated between channels. To support this prediction, we evaluated the compression factor attained by FLAC and WAVPACK using cross-channel coding, on a two-channel version of the LI192 recording. This recording was made by a stereo DTAG with 3 cm hydrophone spacing. The mean-squared coherence (Carter *et al.*, 1973) between the channels is predictably high at low frequencies ($>$0.8 up to about 35 kHz), but, despite this, both FLAC and WAVPACK produced the same compression factor (4.0 and 3.8, respectively) for the two-channel sample as for the single channel sample, indicating that no additional effective redundancy was offered by the second channel.

E. Data formats

Data collected at sea are valuable and may be passed among researchers multiple times for different analyses. Maintaining metadata, timing information, and integrity checks with the data thus seems essential. However, the file format most often used for underwater sound recordings, the Microsoft WAV format, has no standard way of implementing any of these to the level required for archive-quality scientific recordings. Defining a standard for such recordings is an urgent community concern although one beyond the scope of this paper. Maintaining data integrity and precise timing relationships is even more important for compressed recordings. Compressed data are, by definition, more sensitive to errors than uncompressed data, and the use of variable length codes can make it difficult to recover from errors without losing timing precision by dropping samples. Here we recommend a minimum frame header (Fig. 1) to accompany compressed data packets that provides information for timing and error checking. This enables segments of data with errors to be detected during decompression and replaced by the correct number of zeros to maintain timing exactitude. Another useful feature of this header is a source identification field that allows frames of data from different sensors to be merged into the same file. This enables, for example, measurements of temperature or depth made throughout a recording to be stored along with the sound data, overcoming a short-coming of FLAC that only allows a single block of metadata at the start of each file.

An additional header is placed at the start of each file containing metadata such as the date and location of the recording, the equipment used, its sensitivity and frequency response, and the sampling rate and number of channels in the recording. We recommend a text-based mark-up format (i.e., XML) for embedded metadata allowing descriptive information fields akin to Dublin Core (Weibel, 1997). The benefit of this extra effort is that compressed files of underwater recordings become complete, compact data archives suitable for long-term storage and exchange.

VII. CONCLUSIONS

The low average power and consistent low-frequency spectral emphasis of underwater sound make it particularly suitable for lossless compression in digital sound recorders.

We have described a very low complexity compression algorithm specifically targeted for this application. The algorithm is simple enough to be implemented in real time on low-power microprocessors but offers compression factors of three or more with 16-bit audio over a wide range of ambient noise conditions and sampling rates. Despite the simplicity of the method, it achieves more than 87% of the compression attained with more complex adaptive methods on a test data set. With suitable data framing and error-correction coding, the compressed audio format is suitable for archiving and yet can be accessed at random locations as with a normal uncompressed audio format. For lossless compression to be effective, it is essential to choose gains in the recording system such that the lowest expected ambient noise, or the self noise of the recorder if this dominates, is no more than 10 dB above the quantization noise associated with the fixed point representation. Using this gain setting has a negligible impact on the ability to detect and characterize weak sounds but, compared to the higher gains often used in underwater recordings, results in both wide dynamic range and high compression factors.

ACKNOWLEDGMENTS

The authors express their gratitude to the creators of SHORTEN, FLAC, and WAVPACK who originated many of the ideas exploited here. Data used in this paper were collected by, or with the aid of, N. Aguilar de Soto, P. Madsen, M. Wahlberg, C. Oliveira, A. Bocconcelli, M. Simon, and many field helpers. Discussions with B. McConnell, D. Gillespie, P. Madsen and P. Tyack helped develop ideas in the paper which were further clarified by the anonymous reviewers. Algorithm development was supported by SERDP, ONR, US Navy (N45) and NOPP. M.J. was supported by the Marine Alliance for Science and Technology Scotland (MASTS).

APPENDIX: SOFTWARE USED

GZIP Version 1.3.12 (15 Oct. 2007) downloaded from: http://gnuwin32.sourceforge.net/packages/gzip.htm. Last accessed 20 September 2012. The default compression setting was used.

FLAC Version 1.2.1b (17 Sept. 2007) downloaded from: http://sourceforge.net/projects/flac/files/flac-win/flac-1.2.1-win. Last accessed 20 September 2012. The -5 compression option was used.

WAVPACK version 4.60.1 (no date given) downloaded from: http://www.wavpack.com/downloads.html. Last accessed 20 September 2012. The -h (high quality compression) option was used.

PEAZIP (PPMd implementation) Version 4.0 (no date given) downloaded from: http://www.peazip.org/. Last accessed 20 September 2012. The settings used were: Level (maximum), Method (PPMd), Dictionary (64MB), Word (16).

Analog Devices Inc. (2007). "Datasheet for AD7982, Rev. A," http://www.analog.com/en/analog-to-digital-converters/ad-converters/ad7982/products/product.html (Last viewed September 10, 2012).

Baumgartner, M. F., and Fratantoni, D. M. (**2008**). "Diel periodicity in both sei whale vocalization rates and the vertical migration of their copepod prey observed from ocean gliders," Limnol. Oceanogr. **53**, 2197–2209.

Bhatnagar, M. (**2008**). "TMS320VC5503/C5506/C5507/C5509A power consumption summary [Report SPRAA04C Texas Instruments]," http://www.ti.com/lit/an/spraa04c/spraa04c.pdf (Last viewed September 22, 2011).

Burgess, W. C., Tyack, P. L., Le Boeuf, B. J., and Costa, D. P. (**1998**). "A programmable acoustic recording tag and first results from free-ranging northern elephant seals," Deep-Sea Res., Part II **45**, 1327–1351.

Carter, G., Knapp, C., and Nuttal, A. (**1973**). "Estimation of the magnitude-squared coherence function via overlapped fast Fourier transform processing," IEEE Trans Audio Electroacoust. **4**, 337–344.

Cirrus Logic Inc. (**2006**). "Datasheet for CS5342 (DS608F1)," http://www.cirrus.com/en/pubs/proDatasheet/CS5342_F1.pdf (Last viewed September 10, 2012).

Clark, C. W., and Clapham, P. J. (**2004**). "Acoustic monitoring on a humpback whale (*Megaptera novaeangliae*) feeding ground shows continual singing into late spring," Proc. R. Soc. London **271**, 1051–1057.

Clark, C. W., Gillespie, D., Nowacek, D. P., and Parks, S. E. (**2007**). "Listening to their world: Acoustics for monitoring and protecting right whales in an urbanized ocean," in *The Urban Whale: North Atlantic Right Whales at the Crossroads*, edited by S. D. Kraus and R. M. Rolland (Harvard University Press, Cambridge, MA), pp. 333–357.

Dahl, P. H., Miller, J. H., Cato, D. H., and Andrew, R. K. (**2007**). "Underwater ambient noise," Acoust. Today **3**, 23–33.

Hans, M., and Schafer, R. W. (**2001**). "Lossless compression of digital audio," IEEE Signal Proc. Mag. **18**, 21–32.

Hastings, M. (**2008**). "Coming to terms with the effects of ocean noise on marine animals," Acoust. Today **4**, 22–34.

Jackson, L. B. (**1996**). "Filter design by modelling," in *Digital Filters and Signal Processing*, 3rd ed. (Kluwer Academic, Norwell, MA), Chap. 10.

Johnson, M., and Tyack, P. (**2003**). "A digital acoustic recording tag for measuring the response of wild marine mammals to sound," J. Ocean. Eng. **28**, 3–12.

Koopman, P., and Chakravarty, T. (**2004**). "Cyclic redundancy check (CRC) polynomial selection for embedded networks," in *Proceedings of the 2004 IEEE International Conference on Dependable Systems Networks*, pp. 145–154.

Liu, C. M., Hsu, H. W., and Lee, W.C. (**2008**). "Compression artifacts in perceptual audio coding," IEEE Trans. Audio, Speech, Lang. Process. **16**, 681–695.

MacKay, D. J. (**2003**). "Data compression," in *Information Theory, Inference and Learning Algorithms* (Cambridge University Press, New York), Chap. 1.

Madsen, P. T., Payne, R., Kristiansen, N. U., Wahlberg, M., Kerr, I., and Mohl, B. (**2002**). "Sperm whale sound production studied with ultrasound-time-depth-recording tags," J. Exp. Biol. **205**, 1899–1906.

Madsen, P. T., and Wahlberg, M. (**2007**). "Recording and quantification of ultrasonic echolocation clicks," Deep-Sea Res., Part I **54**, 1421–1444.

Marques, T. A., Thomas, L., Ward, J., DiMarzio, N., and Tyack, P. L. (**2009**). "Estimating cetacean population density using fixed passive acoustic sensors: An example with Blainville's beaked whales," J. Acoust. Soc. Am. **125**, 1982–1994.

Mellinger, D. K., Stafford, K. M., Moore, S. E., Dziak, R. P., and Matsumo, H. (**2007**). "An overview of fixed passive acoustic observation methods for cetaceans," Oceanography **20**, 36–45.

Møhl, B., Wahlberg, M., Madsen, P. T., Heerfordt, A., and Lund, A. (**2003**). "The monopulsed nature of sperm whale clicks," J. Acoust. Soc. Am. **114**, 1143–1154.

Orfanidis, S. J. (**2010**). *Introduction to Signal Processing*, Rutgers Univ., www.ece.rutgers.edu/~orfanidi/intro2sp (Last viewed December 5, 2011), pp. 61–65.

Richardson, W. J., Greene, C. R., Jr., Malme, C. I., and Thomson, D. H. (**1995**). *Marine Mammals and Noise* (Academic, San Diego), pp. 87–158.

Robinson, T. (**1994**). "SHORTEN: Simple lossless and near-lossless waveform compression," Report No. CUED/F-INFENG/TR.156, (Cambridge University Engineering Department, Cambridge, UK).

Sarwate, D. V. (**1988**). "Computation of cyclic redundancy checks via table look-up," Commun. ACM **31**, 1008–1013.

Solomon, D. (**2006**). "Audio Compression," in *Data Compression: The Complete Reference*, 4th ed. (Springer-Verlag, New York), Chap. 7.

Weibel, S. (**1997**). "The Dublin Core: A simple content description model for electronic resources." Bull. Am. Soc. Inf. Sci. Technol. **24**, 9–11.

Wenz, G.M. (**1962**). "Acoustic ambient noise in the ocean: Spectra and sources," J. Acoust. Soc. Am. **34**, 1936–1956.

Wiggins, S. M. (**2003**) "Autonomous acoustic recording packages (ARPs) for long-term monitoring of whale sounds," J. Mar. Technol. Soc. **37**, 13–22.