UNDERWATER VIDEO STREAMING USING ADAPTIVE FRAME DECIMATION

Gilad Avrashi[1,2], Shlomi Museri[1], Yaakov Bucris[2], Azriel Sinai[2] and Alon Amar[2]

[1]Signal and Image Processing Lab, Technion, Haifa, Israel
[2]Signal Processing Department, Acoustics Research Center, Rafael, Haifa, Israel

ABSTRACT

We introduce a low data rate video compression scheme for underwater acoustic communication. Recently, multi-carrier based underwater acoustic modems have been proposed and tested for data rates of up to tens of kilobits-per-second for about 1-4 kilometers transmission range. Online transmission of video data using these rates requires extreme compression. Herein, we propose pre-encoder and post-decoder processing in order to reduce the data rate. This is done by exploiting the characteristics of underwater videos for excluding frames. The algorithm was tested and analyzed in simulation environment with several video samples taken by divers in shallow waters of the Mediterranean. We show that using this technique, increased performance is achieved compared to standard H.264/AVC based codecs in both objective and subjective terms.

Index Terms— Video compression, underwater acoustic communication, orthogonal frequency division multiplexing.

1. INTRODUCTION

Underwater on-line video transmission is a growing need in military and commercial applications, such as autonomous underwater vehicles (AUV), oil and gas surveys, underwater pipe inspection and environmental monitoring. Current wired on-line video transmission enables high bit rate communications with the cost of range and mobility limitations. Wireless underwater acoustic modems have been developed in the past few years [1]. Using orthogonal frequency division multiplexing (OFDM), reliable communication with rates of up to tens of kilobits-per-second (kbps) was reached, enabling video transmission [4],[5]. The acoustic modem performance in terms of available data rates is dictated by the used bandwidth, which in turn affects the range of reliable communication. Hereafter, we take the maximal data rate to be 30 kbps, allowing us to consider reliable video streaming for ranges around 2 kilometers of horizontal transmission in shallow waters. In practical underwater acoustic communication (UAC), the bit error rate (BER) of the system is about 10^{-3}-10^{-2} which can not be tolerated when transmitting compressed video. Hence, error correction codes such as low density parity check and convolutional codes should be included, reducing the effective data rate furthermore.

At the video sensor, we consider gray scale 8 bits-per-pixel (bpp) digital data at a frame rate of 25 frames-per-second (fps). The bit rates required for real time transmission of such uncompressed video data is the order of tens to hundreds of Mbps (for standard resolution videos). Considering the UAC data rates described above, compression ratio of up to 3000 is needed in order to enable real time UAC video streaming. The H.264 standard [3] is an effective compression scheme even for low data rates but provides poor results for the underwater channel rates due to the high compression ratio.

The high level system block diagram is shown in Fig. 1. Herein, we focus on the video compression modules (enclosed by the dashed lines). We propose an on-line video compression codec, suited for data rates achieved with underwater acoustic OFDM systems. By exploiting unique characteristics of underwater photography, we propose pre-encoder and post-decoder processing stages for the H.264/AVC codec in order to reduce the data loss. Our pre-processor performs frame decimation based on the moving edge detection concept, proposed for surveillance video coding [6]. The post-processor performs frame interpolation in order to reconstruct the original video. The proposed algorithm [2] provides an improved quality underwater video stream compared to the standard H.264/AVC with the same rate. Another benefit is the distributed complexity and power usage between the encoder and the decoder, which is applicable for underwater resource limited sensors.

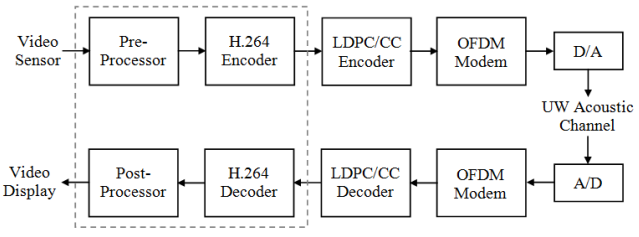

Fig. 1. Underwater video streaming block diagram

2. PRE-ENCODER PROCESSING

Underwater videos have unique characteristics that can be exploited when designing a video encoder. Here, we focus our scope on the fact that the relative speed between the video sensor (diver or AUV) and the object is usually small and the frames are slowly time-varying. The immediate implication is that the standard rate of 25 fps can be reduced, which in turn can be achieved with a frame decimation (FD) mechanism at the decoder. A straightforward approach is to exclude frames in pre-determined equally spaced times. This can be viewed as resampling the video stream using a frame decimation factor denoted by $D > 1$. Consequently, the new frame rate will be $25/D$ fps. The decimation factor can be pre-determined by the sensor movement. However, the sensor and object velocities may not be constant during the observation time, hence an adaptive method is expected to provide better results.

Consider L frames of a video stream (L could be the streamer input buffer size) where the frame rate is 25 fps. Our idea is to exclude the frames that hold negligible variations compared with previous ones. This is done in a pre-encoding process described in Fig. 2. The process is based on two steps: 1) Edge detection; 2) Motion detection between consecutive frames. The first step uses edge detection algorithm. In our simulations we used such as the Canny technique [7] (see Section 3). A range of other edge detection methods can be used in this step, such as discrete cosine transform (DCT) coefficient thresholding [6]. By applying edge detection on each frame, we construct a binary image. Denote by $\mathbf{F}_i$ the ith frame, the respective edge map is defined by

$$E_i(m,n) = \begin{cases} 1 & F_i(m,n) \text{ is included in an edge} \\ 0 & \text{belongs to an edge} \end{cases} \quad (1)$$

where (m,n) are the pixel indices. The goal of the second step is to determine whether significant movement exists between the ith frame (reference) and the jth frame (tested), where $j - i \geq 1$. This is done by comparing the normalized derivative of the edge maps to a threshold μ

$$\frac{\sum_{m,n} |E_j(m,n) - E_i(m,n)|}{\sum_{m,n} E_j(m,n) + E_i(m,n)} \underset{\text{not detected}}{\overset{\text{detected}}{\gtrless}} \mu \quad (2)$$

If no significant movement was detected the tested frame j will be skipped by the encoder. i.e, it will not be encoded by the H.264/AVC implementation. The reference is set to the last saved frame. This process produces a punctured video stream, where frames in non-constant times are missing. In order to reconstruct the video in the decoder side, the puncturing pattern mask is saved and sent as side information. Let $\mathbf{b} = [b_1, b_2, \ldots, b_L]^T$ denote the mask vector where

$$b_i = \begin{cases} 1 & i\text{th frame is saved} \\ 0 & i\text{th frame is skipped} \end{cases} \quad (3)$$

Some video encoders support frame time stamps, which can be used instead. Finally, the saved frames are sent to the encoder. Given the channel capacity (allowable bit rate) R_0 [kbps], the target rate of the encoder will be

$$R = \frac{R_0 L}{\|\mathbf{b}\|_0} \quad (4)$$

Where $\|\mathbf{b}\|_0$ denotes the 0-norm of the mask vetor, i.e, the number of non-zero elements of $\mathbf{b}$. Define the effective frame decimation factor

$$D_{\text{eff}} = \frac{L}{\|\mathbf{b}\|_0} \quad (5)$$

A trade-off exists - a sparse vector $\mathbf{b}$ will result in higher target rates for the encoder but will cause greater data loss and require more frame reconstruction at the decoder. A non-sparse $\mathbf{b}$ will cause the opposite results. By substituting (5) into (4), we get

$$R = R_0 D_{\text{eff}} \quad (6)$$

We can now understand the pre-processor encoder interface, illustrated in the right part of Fig. 2. The pre-processor produces a series of frames to be transmitted, defined as

$$S_k = [F_{i_1}, F_{i_2}, \ldots, F_{i_K}] \quad (7)$$

where the series size K is dictated either by the input or output buffer and the indices $i_1 \ldots i_K$ correspond to the non-zero elements of $\mathbf{b}$. At the same time, using the mask vector, the effective decimation factor $D_{\text{eff}}(S_k)$ is calculated. Once the output buffer is filled, S_k is sent to the encoder as well as the desired data rate, the buffer is cleared and the pre-processor turns to construct S_{k+1} and $D_{\text{eff}}(S_{k+1})$.

3. POST-DECODER PROCESSING

Given the H.264/AVC decoded video and the mask vector $\mathbf{b}$, we now turn to reconstruct the original video. Unlike videos with constant frame rate, it cannot be displayed as is, since its frame rate varies through time. Using $\mathbf{b}$, the missing frames are located and interpolated from the previous and next decoded frames. Assume, for example, that $b_i = b_{i+k} = 1$ while $b_{i+1}, \ldots, b_{i+k-1} = 0$ in accordance to the definition in (3). Consider the associated decoded frames $\mathbf{F}_i, \mathbf{F}_{i+k}$. The goal of the post-decoder processor, described in Fig. 3, is to reconstruct the missing frames $\mathbf{F}_{i+1}, \ldots \mathbf{F}_{i+k-1}$, using one of the methods listed below:

1. Zero order hold (ZOH): missing frames are replicated from the previous known one. $\mathbf{F}_{i+1}, \ldots \mathbf{F}_{i+k-1} = \mathbf{F}_i$. Note that this method is inherent in time stamp integrated decoders, in which case no post processing is needed.

2. Linear interpolation (LI): the (m,n)th pixel of the $(i+j)$th frame is given by

$$F_{i+j}(m,n) = \frac{(k-j)F_i(m,n) + jF_{i+k}(m,n)}{k} \quad (8)$$

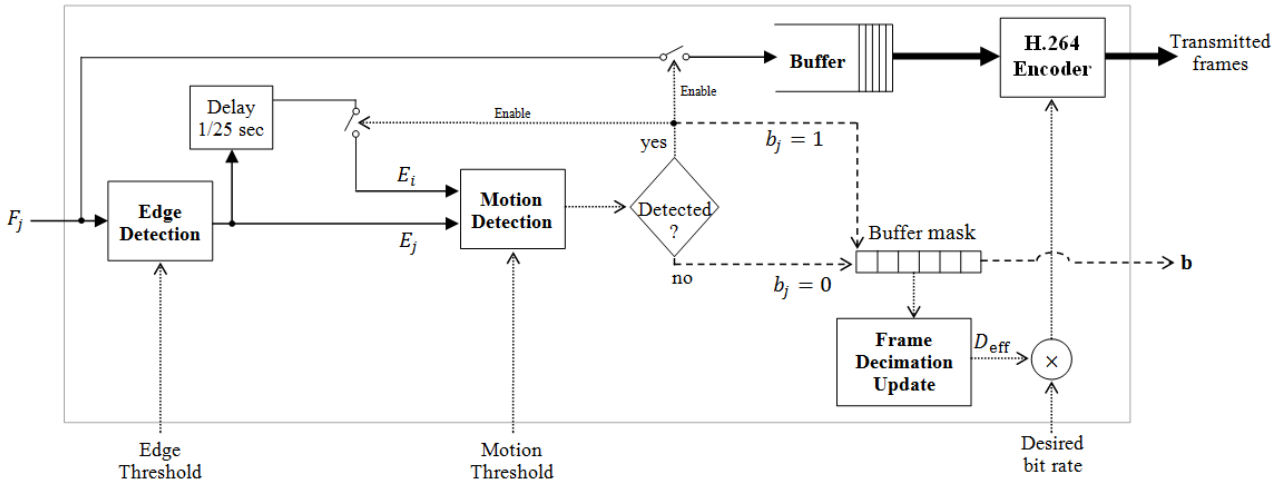

Fig. 2. Encoder block diagram. Solid lines represent the frame and edge maps, dashed lines represent the mask vector and dotted lines represent control operations

where $1 \leq j \leq k-1$ represents the index of a missing frame relative to the last received one.

3. Motion estimation (ME) based: $\mathbf{F}_i, \mathbf{F}_{i+k}$ are divided into P 16×16 blocks. A motion estimation algorithm is then used to find (x_p, y_p) $1 \leq p \leq P$, the 2D motion vector between the pth blocks of the given frames. In our post-decoder implementation we use the diamond search algorithm [8] to construct the motion vectors. Other algorithms, such as three and four step search methods [9]-[11] can also be used in this stage. Next, we estimate the motion vectors of the missing frames by linear interpolation. The resulting motion vectors of the jth frame are given by

$$(x_p^{(j)}, y_p^{(j)}) = \left(\frac{j}{k}x_p, \frac{j}{k}y_p\right), 1 \leq p \leq P \qquad (9)$$

The (m,n)th pixel of the reconstructed frame is then

$$F_{i+j}(m,n) = F_i(m - x_p^{(j)}, n - y_p^{(j)}) \qquad (10)$$

4. REAL DATA DEMONSTRATION

We tested our method on 10 video clips filmed by divers off the shore of Israel. The frame resolution was 640×480, 8 bits-per-pixel gray scale color depth, summing up to 60 Mbps data rate at 25 fps frame rate. Fig. 4 shows an example of decoded and reconstructed frames using the online available H.264/AVC implementation x264 decoder [12] and using the proposed FD methods. The achieved data rate of the x264 was 36 kbps, while for the FD, the target rate of 30 kbps was reached. As can be observed, the FD frames conserve more details than the x264 frame. This is clearly noticed looking

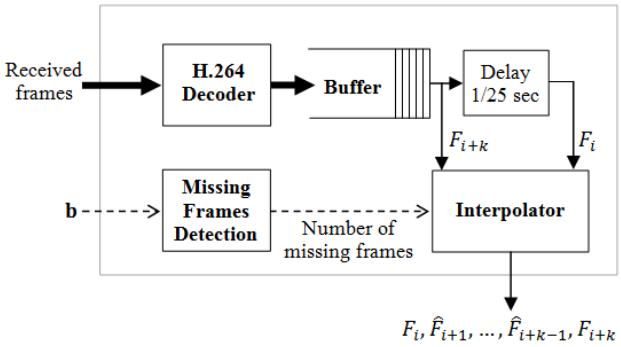

Fig. 3. Decoder block diagram

at the bottom of the structure, enlarged in Fig. 5, where several bolts are visible at the FD frames while indistinguishable at the x264 frames. Looking at the pipe line running from the top of the frame down towards the bottom right corner, a smudging effect can be observed at the so called linear interpolation frame. This phenomenon is noticeable in high contrast regions due to the averaging of pixel values as described in (8).

Objective results were evaluated in terms of the standard peak signal to noise ratio (PSNR) measure. Define the mean squared error

$$\text{MSE}_i = \frac{\sum_{m=1,n=1}^{M,N}[F_{i,o}(m,n) - F_{i,c}(m,n)]^2}{MN} \qquad (11)$$

where $F_{i,o}$,$F_{i,c}$ are the ith frames of the original and the reconstructed videos respectively, the size of $M \times N$ pixels

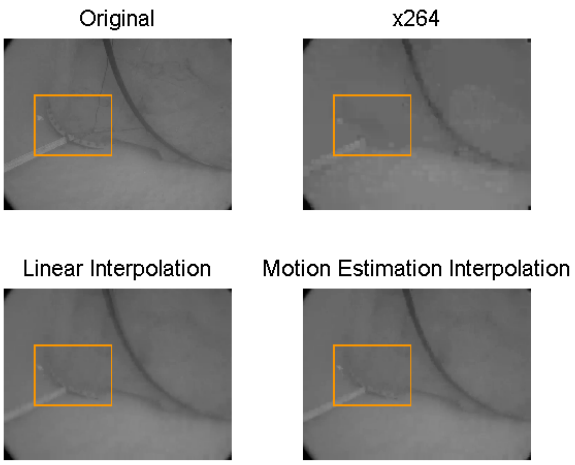

Fig. 4. Example of a reconstructed frame using different methods

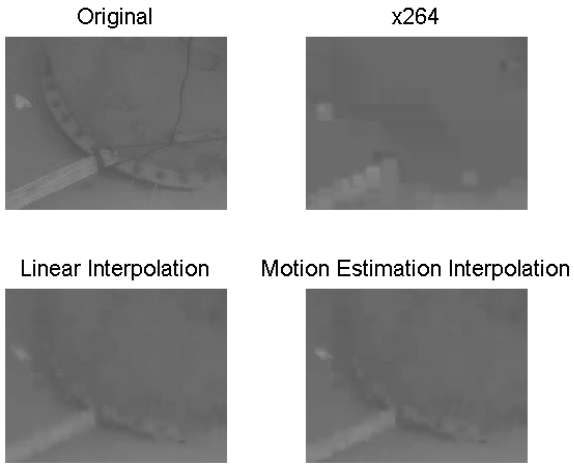

Fig. 5. Zoom-in on high detailed regions of Fig. 4

each. The PSNR of a video is given by

$$\text{PSNR} = 10\log_{10}\left(\frac{A^2}{\frac{1}{L}\sum_{i=1}^{L}\text{MSE}_i}\right) \qquad (12)$$

where A is the maximum possible pixel value (255 when using 8 bpp). It should be noted that while PSNR enables objective measurement of video quality, it suffers from distinct incompatibilities with human impression measurements. PSNR is calculated by pixel-wise comparison, making it highly sensitive to displacements while fairly optimistic for low contrast images. PSNR results are shown in Fig. 6. A set of 1000 frames was tested for a range of data rates using each one of the frame recovery methods and the x264 codec solely. For rates below 50 kbps, which are relevant for UAC in ranges of few kilometers, the FD method achieves better performance. The x264 codec in its standard configuration could not achieve rates below 36 kbps for the tested files. Comparing the frame recovery methods, it can be deduced, in contrast to the impression given by Fig. 4, that the linear interpolation method achieves better results than the other two. The reason is that the errors of the ME and ZOH methods are typically block and frame shifting respectively, while for the LI method the misplacements are smudged within the already small pixel value range.

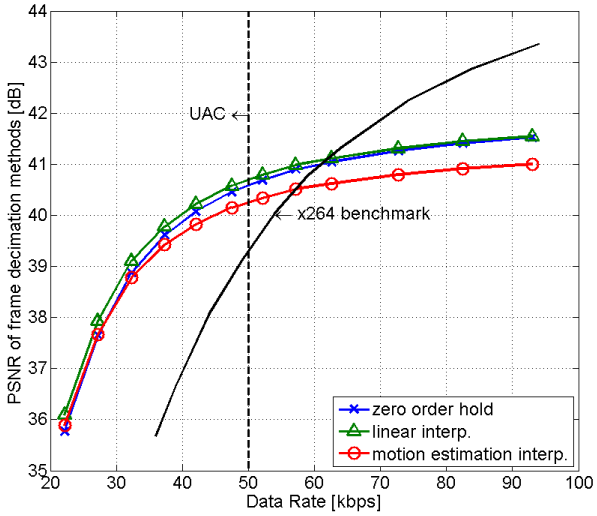

Fig. 6. PSNR vs data rate.

Finally, we explored the frame rate trade-off, explained in section 2. The proposed adaptive frame decimation algorithm was tested for a range of motion estimation and Canny method thresholds. Fig. 7 illustrates the PSNR performance for a range of frame rates achieved by altering these thresholds for a single test video with 30 kbps target data rate. Clearly, a trade-off behavior is visible with an optimum at around 10 fps (or an effective decimation factor of 2.5). As a reference, the estimated PSNR for the x264 codec at 30 kbps is plotted (rightmost square).

5. CONCLUSION

We have presented a novel underwater video compression technique, which exploits the unique features of underwater videos to achieve the UAC data bit rates. Improved PSNR values as well as good human impression results were achieved compared to a H.264/AVC standard codec. These results open a gate to online UW video streaming using acoustic communications, for ranges of several kilometers.

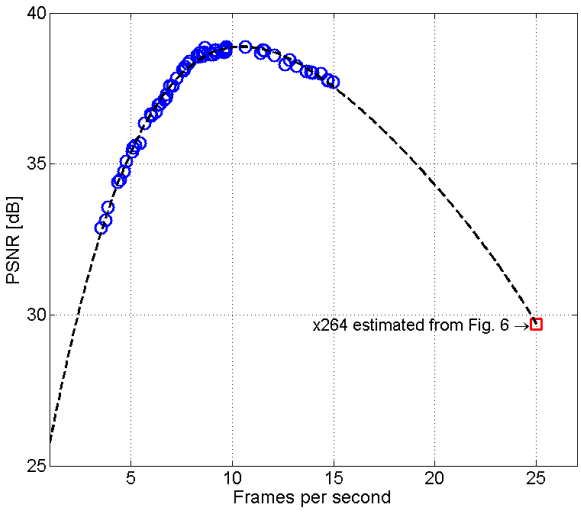

Fig. 7. PSNR vs frame rate.

6. ACKNOWLEDGMENTS

The authors would like to thank the Signal and Image Processing Lab (SIPL) at the department of Electrical Engineering at the Technion for providing the necessary facilities that helped to complete this work. Special thanks to Mr. Rami Cohen, member of the SIPL staff, for his thorough guidance and help during the work on this project.

7. REFERENCES

[1] B. Lee, S. Zhou, M. Stojanovic, L. Freitag, and P. Willett "Multicarrier Communication Over Underwater Acoustic Channels With Nonuniform Doppler Shifts", *IEEE Oceanic Engineering*, vol. 33, no. 2, Apr. 2008, pp. 198-209.

[2] G. Avrashi and S. Museri "Video Compression for Underwater Acoustic Communication", Technical Report P 9-1-12, Signal and Image Processing Lab, Electrical Engineering department, Technion, Israel, Dec. 2013.

[3] J. Osterman et. al. "Video Coding with H.264/AVC: Tools, Performance, and Complexity", *IEEE Circuits and Systems Magazine*, vol. 4, Apr. 2004, pp. 7-28.

[4] L.D. Vall, D. Sura and M. Stojanovic "Towards Underwater Video Transmission", *Wireless Underwater Networks (WUWNet) '11*, Dec. 2011.

[5] J. Ribas, D. Sura and M. Stojanovic, "Underwater Wireless Video Transmission for Supervisory Control and Inspection Using Acoustic OFDM", *IEEE Oceans '10*, Sep. 2010, pp. 1-9.

[6] C. Kim and N.E. O'Connor, "Low Complexity Video Compression Using Moving Edge Detection Based on DCT Coefficients", *MMM 2009*, vol. 5371, Apr. 2010, pp. 96-107.

[7] J. Canny, "Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, Nov. 1986, pp. 679-698.

[8] S. Zhu and K. Ma, "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation", *IEEE Transactions on Image Processing*, vol. 9, no. 2, Feb. 2000, pp. 287-290.

[9] R. Li, B. Zeng and M. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, Aug. 1994, pp. 438-442.

[10] J. Lu and M. Liou, "A Simple and Efficient Search Algorithm for Block-Matching Motion Estimation", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 2, Apr. 1997, pp. 429-433.

[11] L.M. Po and W.C. Ma, "A Novel Four-Step Search Algorithm for Fast Block Motion Estimation", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, Jun. 1996, pp. 313-317.

[12] VideoLan Organization, "x264 home page", (Link: www.videolan.org/developers/x264.html).