Implementing a Video Encoder and Decoder

Daniel Bonanno, *University of Malta*

***Abstract*— In this report, an overview of the design of a simple video coder and decoder in Matlab is given. Results obtained by varying parameters and techniques are also discussed.**

I. Introduction

The aim of this paper is to discuss the design and implementation of a simple video encoder and decoder in Matlab. Video frames have high correlation, in both space and time. This means that a lot of redundancy is present. By exploiting these redundancies video data can be compressed. This is the aim of video encoders, such as the one designed in this work. The rest of this paper is organised as follows. First the design of the Encoder and the Decoder are discussed in Sections II and III respectively. Section IV discusses the results obtained and Section V concludes the paper.

II. Encoder Design

A. Input

The encoder consists of inter-frame Differential Pulse Code Modulation (DPCM), a block quantizer, motion estimation and compensation and variable length coding. The overall design follows that of Figure 1 [1], which shows an MPEG-1 video encoder.

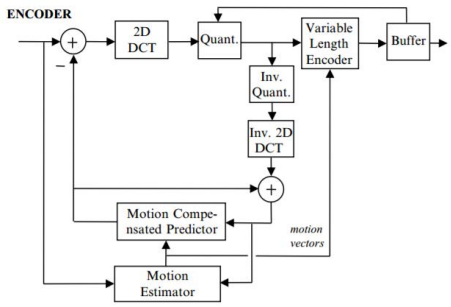

Figure 1: MPEG-1 Encoder [1]

The input to the encoder is the video, which is then split into the different frames. If the video is in colour, the three different channels are split into separate frames. It is noted that the human vision system is less sensitive to colour differences than it is to changes in luminance. Thus, chroma sub-sampling can take place to encode chroma information in a lower resulting. It is assumed that the input video is already chroma subsampled.

B. Residual Error

Next, the residual difference between the previous motion compensated frame, which is predicted from the data which is sent to the decoder, and the current input frame is calculated. This difference is called the residual error. This step, along with motion prediction and motion compensation which will be discussed later on, exploit temporal redundancies between adjacent frames. The greater the correlation between adjacent frames, the smaller the error to be encoded and thus, the better the compression that can be achieved, since shorter code words are sent.

In video compression, Intra-Coded frames (I-frames) do not exploit such redundancies. Thus, for these types of frames, the predicted frame is set to 0. This means that the residual error is equal to the frame that is inserted. Apart from the first frame of a video, I-frames occur at the start of a new Group of Pictures (GOP) and serve as a reference point, such that errors are not propagated in time. Having said that, such frames have lower compression, since the errors generated by this step are large and have a random distribution, which does not help with entropy encoding. An example of the error generated by the both an I-frame and a Predicted-Frame (P-frame), along with the histogram of the error values, can be found in Figure 2.

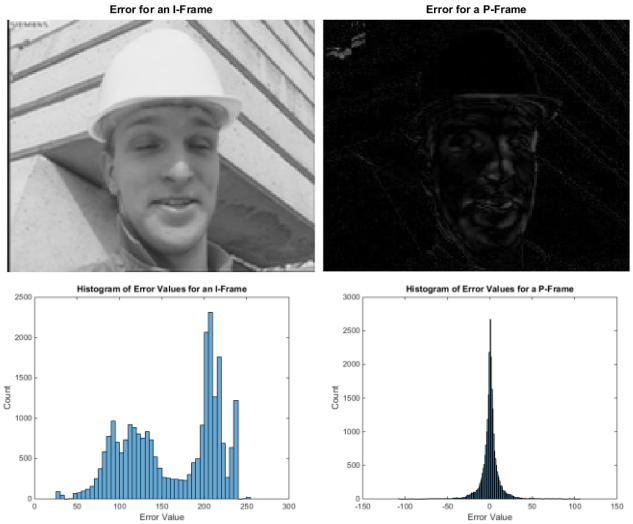

Figure 2: Error Values and Histogram for I-Frames and P-Frames

C. 2D DCT and Quantisation

The error is then split into 8x8 non-overlapping blocks and the Discrete Cosine Transform (DCT) is applied

on each block. The DCT helps to concentrate the energy into the top left part of a matrix of coefficients, which contains the lower coefficients for lower frequencies. Thus, it can be argues that the DCT exploits spatial redundancies in the frames – pixels which are close to each other generally have a similar error value, resulting in a lower frequency. This representation is better suited for compression, since this 8x8 block matrix is then fed to a block quantizer. The Human Visual system is less sensitive to higher frequency than to lower frequencies. This is exploited by the quantizer, which multiplies input block by quantisation matrices to perform non-uniform quantisation and reduce the dynamic range of the output. Quantisation can be performed by the equation below, where F(u,v) represents the original DCT matrix and Q(u,v) represents the quantisation matrix. Due to the rounding function, it must be noted that quantisation is irreversible and is what introduces the loss in the decoded video.

$$\hat{F}(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

The quantisation matrices used are provided in the assignment sheet and can be scaled to quantise more. The larger the value in the quantisation matrix, the more loss, but the lower the fidelity to the original representation.

D. Representation for DC and AC Values

The quantized DCT coefficients are then read in a zig-zag fashion, in order to increase the number of 0 values elements adjacent to each other. The DC coefficient is the coefficient with the lowest frequency in the DCT coefficient matrix. This is therefore the top-leftmost coefficient. By performing the zig-zag scanning of the matrix, the DC coefficient is the first value of the sequence. The remaining coefficients are called the AC coefficients. It can be noted that, the higher the frequency, the lower the coefficient value will be. Most AC coefficients will have a value of 0. Thus, through zig-zag scanning, larger runs of 0s will be present. This will result in larger compression when using Run Length Coding (RLC) on the AC coefficients. Figure 3 [2] shows the concept of zig-zag scanning, DC and AC coefficients.

After obtaining the zig-zag order of each block in the frame, the DC coefficients are encoded using differential pulse code modulation (DPCM). Due to the spatial redundancy in each frame, the DC coefficients in each frame are highly correlated. Thus, by using DPCM, the representation of the DC coefficients has lower magnitudes and smaller variances. This means that the DC coefficients can be encoded using less bits. A graphics representation of DPCM encoding of the DC coefficients is found in Figure 4 [2].

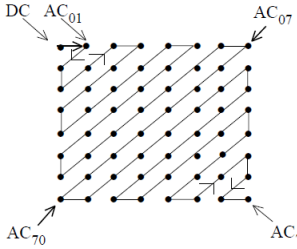

Figure 3: Zig-Zag scanning of the DC and AC Coefficients [2]

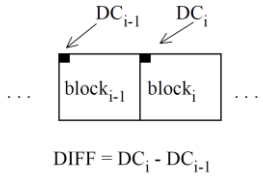

Figure 4: Representation of DPCM on adjacent DC Coefficients [2]

As previously mentioned, the AC coefficients are Run Length Coded on a block by block basis. This results in symbols in the form: (run-length, value), where 'run-length' is the number of 0 valued coefficients before 'value' coefficient. The values are written in hexadecimal. It is clear here why zig-zag scanning is important, so that there are as few symbols as possible. It must be noted that two special symbols exist: (F,0) represents a full run of 0s, since the maximum run of 0s allowed is 15, and (0,0) is used to represent the end of block.

E. Motion Estimation and Motion Compensation

As can be observed from Figure 1, the lower part of the encoder reverses the quantisation and DCT processes on the residual error and adds the predicted frame, so as to obtain once again a representation of the current frame that was input. Of course, due to the quantisation effects, this frame is not an exact replica of the input frame, but it is what the decoder would be decoding if there are no errors present. This frame is stored in the motion compensation block for the next frame input.

This process is done to generate motion vectors for the next frame to be input. By having this representation, the encoder has a copy of what the decoder would have up to this point. It then generates motion vectors using this frame for the next frame which the decoder would receive. Therefore motion vectors for frame f_{n+1} are generated from a lossy version of f_n , which would also be available to the decoder.

These motion vectors, generated via the motion estimation block, aim to estimate pixel values in the current frame from the previous frame. A motion vector is generated for every block and gives the vertical and horizontal distances between the current block and the closest block in the previous frame. This is done by defining a search area in the previous frame for every block in the current frame. Blocks in the search area are compared to the current block and the one which has the lowest cost is retained, generating a motion vector which points to it. The cost function used in this work is the mean square error.

It is important to note that the search area can be as big or as small as the user defines it to be. In general, a search area around a block can be defined as shown in Figure 5. In this work, the size of the block is a constant 8x8 and $p_{horizontal} = p_{vertical}$. The search area defined can have an impact on the results achieved, as will be discussed in Section IV.

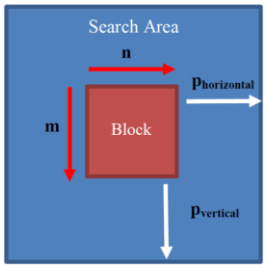

Figure 5: How a search area is defined for an arbitrary n×m block

Furthermore, the search area can be searched in various ways. A simple approach is the full search or exhaustive search, which checks all possible 8x8 blocks in the search area. This ensures that the block with minimal cost in the search area is chosen, however it is computationally expensive. Another technique which is implemented in this assignment is the 2D Logarithmic Search (TDL), which requires less steps. It defines a search pattern in the shape of a '+', with the final stage using 9 points in a square pattern. Thus, it is less computationally intensive, but might not find the minimum cost block.

The lossy version of f_n is also passed on to the motion compensated prediction block, along with the motion vectors generated. This block aims to generate the predictor frame for f_{n+1}, which will be used by the decoder to generate the residual error for f_{n+1}.

The motion vectors generated are also DPCM encoded similar to the DC coefficients, prior to being sent for entropy encoding. This is done since it can be noted that adjacent blocks have similar motion vectors. Furthermore, this is done since, as the search area increases, more bits would have to be used, since larger motion vector values are possible. For example, if the search area is small, the motion vectors generated may be in the range (±8,±8), which would require at most 5 bits. However, if the search area is increased, the motion vectors may now be in the range (±64,±64), which would require 7 bits. Thus, DPCM solves this problem since small magnitudes are required, which can be represented with smaller bits.

F. Entropy Coding

Entropy Coding provides further compression without introducing any further loss. This is done by using variable length codes. The more frequent a code word is, the shorter its binary representation. In this work, entropy coding is performed by using Huffman Coding and Huffman tables found in the ITU JPEG Standard [3]. It can be noted that different tables exist for DC and AC coefficients and for Luminance and Chrominance Frames. For the motion vectors, the same representation and tables as the DC coefficients is used.

For the DC Coefficients and Motion Vectors, a (Size, Amplitude) representation is used. 'Size' represents the number of bits that are used to encode 'Amplitude', which shows the actual amplitude of the DC coefficient or motion vector encoded in one's complement. The value for 'Size' and its binary representation are obtained from the tables. It can be noted that for an amplitude of 0, no binary value for 'Amplitude' value is sent, resulting in a more efficient binary representation of the video.

A similar approach is used for the AC values. However, the representation for the AC coefficients is in the form: (Run-length/Size, Amplitude). Again, 'Size' is first read from the tables based on the value of 'Amplitude'. The binary representation corresponding to the 'Run-length/Size' symbol obtained is found using tables and 'Amplitude' is again encoded using one's complement.

The final binary stream per block is in the form: *(DC_Binary, AC_Binary, MotionVector_Binary).*

III. Decoder Design

The decoder works in an opposite way to the encoder. It first starts by doing Huffman entropy decoding. It reads in the bit stream until it encounters a code word for the 'Size' value. This is done by using the Huffman tables. It then reads 'Size' bits and decodes them using ones complement. After doing so, it would have the DC coefficient. A similar procedure is repeated for the AC Coefficients, ensuring to revert the run length

coding process and the zig-zag scanning of the coefficient matrix. The Motion Vectors are read in a similar way to the DC Coefficients.

Once all the coefficients for a block are decoded, it performs inverse quantisation and the inverse DCT transformation. This process is repeated for all the blocks in a frame, after which the DPCM process can also be reversed. The decoder then feeds the motion vectors is read to the motion compensated predictor block. This block, similar to the one found in the encoder, reconstructs a predicted frame from the previous predictor frame using the motion vectors. Recall that the frame which has been decoded is the residual error, with some loss, from this predicted frame. Thus, by adding the residual error to this predicted frame, the actual video frame is reconstructed. This reconstructed frame is then sent back to the motion compensated predictor, which will use it as a predictor frame for the next frame. Figure 6 [1] shows a graphical representation of the decoder.

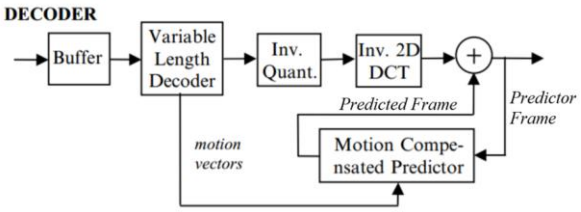

Figure 6: MPEG-1 Decoder [1]

Of course, to generate correct motion vectors, the encoder's motion compensated predictor must have stored the same frame that the decoder will have. Thus, to ensure correct operation of the implemented decoder, the output frames of the decoder, which will then be the predictor frames for the motion compensated predictor in the decoder, where compared to the frames stored inside the motion compensated predictor in the encoder. In every case tested, the frames were the same, meaning that the decoder was functioning correctly.

IV. Results

The above mentioned encoder and decoder were tested using 3 commonly used video test sequences downloaded from [4]. The test sequences are *Foreman*, *Suzie* and *Miss America*. Between them, they cover a range of motions, including fast motions and slow motions. All test sequences were already in YUV format, with the chroma subsampled using the 4:2:0 pattern. The resolution of all videos is QCIF. This means that the Y has an aspect ratio of 176x144 and the chroma components have an aspect ratio of 88x72. The results obtained are averaged out over all 3 videos.

The compression ratio is define as the ratio between the video's size before compression and after compression. It is used to determine how effective the codec is at reducing the bit rate. Furthermore, the Peak Signal to Noise Ratio (PSNR) is used to give information on the quality of the compressed video with respect to the original video. The PSNR can be calculated as follows:

$$PSNR = 10 \log \left(\frac{2^B - 1}{\frac{1}{XY} \sum_{x=1}^{X-1} \sum_{y=0}^{Y-1} (f[x,y] - \widehat{f[x,y]})^2} \right)$$

where:

- B = number of bits per pixel = 8
- X = number of pixels horizontally
- Y = number of pixels vertically
- $f[x,y]$ = original frame
- $\widehat{f[x,y]}$ = frame after compression

A. Varying the Search Area and Search Method

As already mentioned in previous sections, the search area and search method used have an effect on the results obtained. This section presents the results obtained when using both an exhaustive search and the TDL method are used. Furthermore, the size of search area was varied, with $p_{horizontal} = p_{vertical}$ varying from 8 to 64 pixels.

As can be seen in Figure 7, the Y-PSNR increases as the search area increases, irrespective of which search method is used. Having said that, the exhaustive search outperforms the TDL method in terms of quality. This is expected, since the exhaustive search checks every possible block. However, the exhaustive search is very computationally intensive, especially as the search area is enlarged. Thus, for large search areas, the TDL provides a good compromise between the time taken to encode and the final Y-PSNR.

Furthermore, it can be noted that the compression ratio decreases as the search area increases. This is expected, since it is more probable that a better matching block is found with a larger search area, meaning that the residual errors are small and therefore compressed better. However, it is interesting to note that the TDL algorithm provides a better average compression ratio that the exhaustive search method. This may be due to the fact that, with TDL, adjacent blocks have more correlation between the motion vectors generated, resulting in smaller DPCM encoded motion vectors.

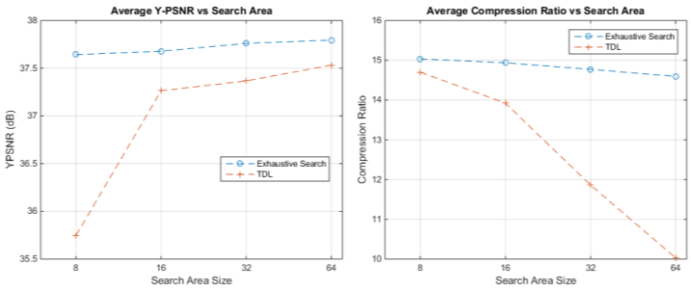

Figure 7: Y-PSNR and Compression Ratio vs Search Area for both the Exhaustive Search and the TDL Method

B. Scaling the Quantisation Matrices

Up to now, the quantisation matrices provided have been used with no scaling. However, they can be scaled by different values. Figure 8 shows the result of scaling the quantisation matrices by 2 and 4, keeping all other parameters (such as the search area and search method) equal. The quality drops as the scaling increases. This is expected: the larger the values in the quantisation matrices, the larger the loss incurred and therefore, the lower the reconstructed quality. With that being said, it can be noted that scaling the quantisation matrices also results in a different compression ratio: the larger the scaling factor, the larger the compression ratio. This is due to the fact that, if scaling is large, there are smaller values and longer runs of 0s in the quantised DCT coefficients, resulting in a more representation of the coefficients.

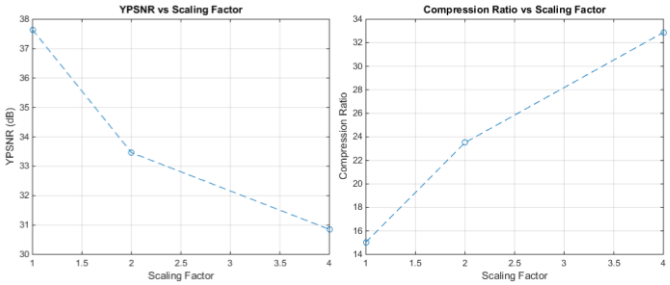

Figure 8: Y-PSNR and Compression Ratio vs Scaling Factor

C. Introducing Noise

As of yet, it has been assumed that all the bit send by the encoder were correctly received at the decoder. However, errors exist in channels between encoders and decoders. Thus, a bit error rate of 3×10^{-4} was simulated. In reality, if a bit error is encountered, the receiver could drop the whole packet, since it might mean that the entropy decoder might not be able to decipher it. In this case, however, the error was added after entropy decoding. This means that the error-less sequence was first entropy decoded, then, if an error was present in the sequence, the packet containing the error would be dropped, resulting in a black block in the frame. A packet was assumed to be 20 bytes (160 bits) long. Thus, a packet might have an effect over more than 1 block. Furthermore, since motion estimation is used, an error in a frame might be propagated to future frames which predict from the erroneous block. These concepts are demonstrated in Figure 9 below.

Figure 9: Errors due to the channel and their temporal propagation

The average PSNR for the video has of course dropped from 37.64dB to 29.74dB, since the errors cause a large reduction in quality. It is worth noting that this time, the PSNR was calculated for all the frames, not just the Y frames, since errors might be present in the chroma frames as well. Since the errors are random, calculating only the Y-PSNR might bias the results.

Up until now, only 1 I-Frame was sent – the first frame, with all the other frames using motion prediction from previous frames. Thus, an error in a frame might have a large effect, depending on how many future blocks predict from it. Therefore, the encoder was set up such that a GOP would be 10 frames long, that is, every 10^{th} frame, an I-frame would be inserted. This would help to stop the temporal propagation of errors. In fact, with this configuration, the PSNR changes to 32.58dB. This means that the loss in quality was significantly less. Having said that, I-frames are compressed less. In fact the improvement in quality causes the compression ratio to reduce from 15.04 to 13.85.

V. Conclusion

This work has discussed a simple video encoder and decoder which exploit redundancies found in frames to compress videos. The results obtained by varying different parameters were also discussed.

References

[1] L. Chiariglione, Ed., The MPEG Representation of Digital Media. New York, NY: Springer New York, 2012.

[2] G. K. Wallace, "The JPEG still picture compression standard," IEEE Trans. Consum. Electron., vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[3] International Telecommunication Union, "Terminal equipment and protocols for telematic services," Study Gr. VIII, ITU, Geneva, 1988.

[4] "YUV Video Sequences" [Online]. Available: http://trace.eas.asu.edu/yuv/. [Accessed: 13- May- 2017].