Audio and Video Coding - Assignment 3 Video Codecs

André Pinho
Email: andre.s.pinho@ua.pt
Nmec: 80313
DETI
UA

Diego Hernandez
Email: dc.hernandez@ua.pt
Nmec: 77013
DETI
UA

Margarida Silva
Email: margaridaocs@ua.pt
Nmec: 77752
DETI
UA

***Abstract*—Report of our implementation of the Assignment 3 of Audio and Video Coding. The proposal was to create lossless intra-frame and hybrid codecs and a lossy codec to achieve some level of compression.**

I. Compile instructions

The project contains a `CMakeLists.txt` on its root directory. In order to build the sources, a `build/` folder should be created on the same level as the cmake file. Inside the build folder, run `cmake ..` . This will create a `MakeFile` (and other files) that can be executed using `make` to generate the final binaries (madv_codecs).

II. Run instructions

The generated binaries can encode and decode video files by providing some arguments. When executing it without any parameters, it will print the usage guide. The valid parameters are stated in tables I, II, III and IV.

TABLE I
Mandatory flags for all encoders and decoders

Short flag	Long flag	Description	Valid values
-i	–in	Input file	Existing filename
-o	–out	Output file	Existing filename

TABLE II
Optional flags for all encoders

Short flag	Long flag	Description	Valid values	Default
-b	–blocksize	Block size in pixels	>0	8
-p	–predictor	Predictor mode	0-9	9

TABLE III
Optional flags for encoders 2 and 3

Short flag	Long flag	Description	Valid values	Default
-m	–macrosize	Macro block size in blocks	>0	2
-a	–searcharea	Search area for inter-frame	>0	4
-d	–searchdepth	Search depth for inter-frame	1-15	4
-k	–keyperiodicity	Periodicity of key frames	>=0	0

TABLE IV
Optional flags for encoder 3

Short flag	Long flag	Description	Valid values	Default
-q	–dct	Use DCT instead	N.A.	false
-y	–qualY	Quality level of the prediction residuals	>=0	10
-u	–qualU			
-v	-qualV			

III. Libraries

A. Bitstream, Golomb

Since the last project, these two libraries received minimal modification. The only alteration worth mentioning is that a new module was created (`mat_golomb_bitstream`). This module extends the functionality of `golomb_bitstream` by allowing it to read and write (signed and unsigned) OpenCV `CV_16x` Matrices.

B. DCT

The Discrete Cosine Transform (DCT) is a real and orthonormal transform. It is one of the most used transformations operations used in the context of image and video coding.

This library includes 2 other developed libraries, which are the following:

- Run-Length-Encoding (RLE)
- Zig-Zag

1) Run-Length-Encoding: The Run-Length-Encoding is a form of lossless data compression. It compresses data by specifying the number of times a value repeats followed by the respective value. It aims on reducing the number of bits used to represent a set of data. This library includes the operations of writing and reading run-length-code sequences. Operations such as:

- Having a vector of values, get the respective RLE represented by a vector of tuples of values, specifying the value and the number of times the same value appears in sequence.
- Given RLE data (a vector of tuples), gets the resulting sequence, thus the original data.

However, in order to get a more efficient compression, taking into account the resulting matrix from the DCT operation and its quantization, it is more effective to represent the

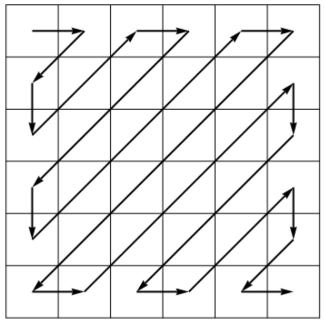

Fig. 1. Zig-zag scan.

compressed data as a vector of tuples, where each tuple has the information of preceding zeros, and the non-zero value. So, in a way, we still recurred to the RLE methodology, although adapted to the need for striving for a better compression. The library has operations capable of construction the pseudo-RLE Sequence from a vector of values, and vice-versa.

2) Zig-Zag: The Zig-Zag library is responsible to perform zig-zag scanning to encode quantized coefficients.

The use of this library allows the codec to obtain a vector of short values representing the sequence of values in a zig-zag scanning sequence (Fig. 1). The inverse operation is possible as well with the use of the *zigzag* library. This operations adapts not only for square matrix, but also for rectangular matrix, as long as its dimensions (number of rows and columns) are even and not odd. The library was developed this way to follow the same input restrictions as OpenCV's DCT module. These cases could be applied for the U and V blocks with rectangular dimensions, that go through the DCT process.

The basic idea of using this scanning is to group together the zero coefficients, using the variant run-length coding, allowing a more efficient representation.

3) Implementation: This library follows the sequential mode of JPEG and its inverse operation. The followed steps are:

- Calculation of the DCT: If an image is given (represented by cv::Mat), followed by the dimension of the block, the image is internally adjusted using padding, so the frame dimensions is multiple of the block dimensions, in order to partition the image in blocks. The programmer can also specify only the size of the blocks. Given the following it adapts the quantization matrix of JPEG (luminance) to the same dimensions of the specified block by cropping the matrix or resizing it, using Linear Interpolation.
  Given a block of a frame, at the calculation of the DCT, each pixel is subtracted to 128 and finally it is calculated the DCT 2D of each block.
- Quantization of the DCT coefficients: The DCT coefficients are quantized using the mentioned quantization matrix, and scaled by a compression quality factor. In the object initialization of the DCT it is needed to specify this quality factor. As default it is set to 1.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 129 & 101 \\ 71 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Fig. 2. Quantization matrix of JPEG (luminance).

$$\begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

Fig. 3. Quantization matrix of chrominance components.

This library implements for default two types of 8x8 matrices: The quantization matrix of JPEG applied to luminance channel (Fig. 2), and another quantization matrix applied for the chrominance components (Fig. 3). This matrices are different since the sensitivity of the human eye to the colors is different from that of the luminance. Thus they are different in order to have a more efficient coefficients quantization, reducing the perceptual redundancy.

C. Predictor

The predictor library allows the codecs to decorrelate the data to be encoded. The two tools available for this purpose are the motion compensation and spacial prediction.

1) Spatial prediction (Intra-Frame Prediction): The spatial prediction is based on both the 7 JPEG linear predictors and the predictor of JPEG-LS, looking at the neighbors that those standards use. The user can force either each of those predictors (0-7 for the JPEG linear predictors or 8 for JPEG-LS) or our auto mode (9). The auto mode applies the best predictor for each block.

The encoder was run for two different videos with prediction in auto mode and different block sizes. In table V we can see the percentage of blocks that were coded with LS and it's possible to observe that the bigger the block, the higher the chance of the LS predictor to be chosen as the best fitting. In table VI, we can see that the inclusion of LS in auto mode barely impacted compression ratio; on the other side, the encoding time significantly increased for a small block size. For this reason, we limited our auto mode to only account for JPEG-LS if the block size is greater than 16.

2) Motion compensation (Inter-Frame Prediction): On a natural video, sequential frames share a significant amount of information. Normally, chunk of pixels from a frame are shared with the next frame, with a offset and some minor

TABLE V
COMPARISON OF JPEG-LS PREDICTION USAGE ACCORDING TO BLOCK SIZE

File	Block size	LS-block percentage
foreman 4:2:0	8	16.4%
	16	23.9%
	32	36.3%
football 4:2:2	8	23.0%
	16	32.5%
	32	46.2%

TABLE VI
ENCODING DUCKS_TAKE_OFF (1080P) IN AUTO MODE WITH AND WITHOUT JPEG-LS

Has LS	Block size	Encoding time	Decoding time	Output size	Compression ratio
No	8	4m10s	1m29s	804MiB	1.64
Yes	8	5m21s	1m36s	803.9 MiB	1.64
No	32	3m55s	1m43s	821.5MiB	1.61
Yes	32	3m40s	1m21s	821.4MiB	1.61

Fig. 4. Motion prediction on *football_422_cif.y4m*

changes. This module tries to exploit those features by searching for matches from a certain frame with the N previous ones. This search is done with square blocks (macroblocks) in a range of pixels from the previous frames (Fig. 4).

This search is done exhaustively (all blocks are tested within a range). We initially planned to develop a logarithmic search (similar to a binary tree search) where we tested four blocks (in four corners) and then select the best one and search its neighbours. But, because of time constraints, we only implemented the trivial approach.

Each candidate is scored by the sum of the absolute value of the residuals of that block with the block to predict. The lower the score, the better.

D. Y4M

Y4M is a simple video (no audio) format where each frame is stored without any compression. It uses the YUV colour space and has chroma sub-sampling. It has 3 types:

- *4:4:4* - No chroma sub-sampling;
- *4:2:2* - U and V channels' width are cut by half; horizontally;
- *4:2:0* - U and V channels' dimensions are cut by half.

Our Y4M module is capable of parsing and storing all those modes, and is able to pad the frames to a specified size. This last feature allows the codecs to split the frames in blocks.

IV. Y4M PLAYER

Our Y4M Player is based on the one developed for the first labwork assignment. It uses our module Y4M to load the files, and OpenCV to show the images. It is possible to feed more than one video, and they will be played sequentially. Pressing the arrow keys will make the player jump to the next or previous video.

V. CODECS

Each codec has two independent binaries: one for encoding, and another for decoding. Additionally each codec has an unique file structure, which is described inside the source folder of each codec (`file_format.txt`).

A. Lossless Intra-Frame Encoder

This encoder only applies the prediction as stated in the previous section. The block size argument only has effect if the auto-predictor is enabled, in which case the best predictor will be found for each of the blocks. In tables VII and VIII, we can see the results obtained after running the encoder with the several prediction modes. As expected, the auto mode displays the best compression ratio, at the cost of a longer encoding time. It's also possible to see that for a fixed predictor for the whole image, the JPEG-LS holds slightly better results. For this reasons, the default predictor is the auto mode.

Similarly to our last project, the Golomb module adapts to the data and recalculates the 'm' parameter on the fly. As a way to improve flexibility of our codecs, each video channel has it's own Golomb object that works independently from each other.

TABLE VII
COMPRESSION RESULTS FOR FOREMAN_CIF.Y4M (4:2:0) USING DIFFERENT PREDICTORS

Prediction type	Encoding time	Decoding time	Output size	Compression ratio
0	4.752s	2.572s	55.2MiB	0.79
1	5.063s	2.704s	27.8MiB	1.56
2	4.396s	2.487s	26.2MiB	1.66
3	4.496s	2.755s	29.4MiB	1.48
4	5.320s	2.727s	27.0MiB	1.61
5	4.434s	2.836s	26.7MiB	1.63
6	4.434s	2.836s	26.5MiB	1.64
7	4.382s	2.646s	26.4MiB	1.64
LS	5.692s	3.007s	25.9MiB	1.67
Auto	9.680s	2.651s	21.6MiB	2.01

B. Lossless Hybrid Encoder

This codec incorporates motion compensator into the codification process. Each frame is always separated into blocks, and there are two types of frames, the *I-Frame* and the *P-Frame*:

- *I-Frame*: These frames are complete frames, represented in intra-mode. They are split into blocks, and have the best predictor is chosen for each block (or selected by default from the arguments). By default, the codification

TABLE VIII
COMPRESSION RESULTS FOR FOOTBALL_422_CIF.Y4M (4:2:2) USING DIFFERENT PREDICTORS

Prediction type	Encoding time	Decoding time	Output size	Compression ratio
0	11.169s	4.487s	86.5MiB	0.80
1	7.772s	4.104s	44.9MiB	1.55
2	7.191s	4.010s	41.2MiB	1.69
3	7.730s	4.080s	48.4MiB	1.44
4	7.435s	4.159s	42.2MiB	1.65
5	7.351s	4.804s	42.2MiB	1.65
6	7.519s	4.131s	42.3MiB	1.65
7	7.189s	4.081s	42.8MiB	1.63
LS	7.615s	4.075s	41.2MiB	1.69
Auto	15.173s	4.556s	35.6MiB	1.96

process only codes the first frame of the video as an *I-Frame*. This happens since these usually take more space to represent than *P-Frame*s. But, it is possible to force the encoder to place an *I-Frame* within a certain interval of frames (flag `-k` of the encoder);

- *P-Frame*: This kind of frame are represented using past frames (inter-mode). The frame is broken down into macroblocks (junction of, by default, 2x2 blocks). For each macroblock, it is calculated it's motion from the last N frames. But, if the score for this representation of the macroblock is higher than a threshold, it is assumed that the macroblock would better be represented on I-Mode. In these cases, the macroblock is once again split into blocks, and are coded just like if they were on a *I-Frame*.

Unlike blocks, macroblocks don't have to be complete. On the right and on the bottom of the frame, the macroblocks of the frame might be missing blocks. In these cases, the macroblock is cropped. The advantage of this approach is higher compression ratio since we aren't storing more padded data.

C. Lossy Encoder

For the regular quantization mode, it's possible to choose a "quality" level for each of the channels. That level is the constant for which the original prediction residuals (both the spatial and motion residuals) will be divided for, so the lower the level, the higher the output quality. We experimented with trying different values among the three channels, but the gains were not significant so we opted for leaving the three defaults equal.

In table IX we can see the compression ratio, maximum per pixel error and SNR for the same video but with different quantization levels. The decoding time was left out because it was similar for all cases. The higher the level, the greater the compression ratio, but also the lower the SNR. We considered that a SNR greater than 30 was good enough, so we opted for leaving 10 as the default level.

If the DCT flag is toggled, the quantization step used while coding is the one provided by the DCT module.

At the encoding process, for each block of each channel, it is applied the calculation of the DCT, quantization of the DCT coefficients and applied RLE. In the same process it is done the reverse operation and written into the frame channel where the block belongs. This is done in order for the encoder to synchronize with the error inserted by quantization. Otherwise, the decoder would skew away from the encoder.

TABLE IX
MPPE AND SNR FOR FOOTBALL_422_CIF.Y4M (4:2:2) USING DIFFERENT QUANTIZATION LEVELS

Quant level	Output size	Compression ratio	Max Per Pixel Error	Min SNR	Max SNR	Avg SNR
6	14.5MiB	4.8	22	34.3	37.5	35.8
8	13MiB	5.4	31	31.7	35.2	33.3
10	12MiB	5.8	39	30.2	33.8	31.5
12	11.4MiB	6.1	47	28.4	31.9	29.8
14	11MiB	6.3	55	27.7	31.1	28.6
16	10.7MiB	6.5	64	26.4	30.4	27.3

D. Alternative Lossy Encoder (Codec3V2)

An additional encoder and decoder was developed in order order to achieve even greater compression. It uses *yuv* library to extract the Y, U and V channels of each frame of the video. In comparison to the last mentioned Lossy encoder, this one only makes of use the sequential mode of JPEG, using the *dct* library which comprises the calculation of the DCT and other steps mentioned in its dedicated subsection of this report. It is also used Golomb Variable Length Coding as a final step to compress the resulting values of the RLE process. It's important to note that no kind of prediction is used, the data feed to the DCT is the raw frames (in YUV).

The inverse operation is implemented in the decoder, thus using the same libraries in order to recover the video with a perceptual redundancy reduced.

This codec resembles the codec of the Motion JPEG standard.

VI. RESULTS

A. Compression ratio and times

Codec3v2 is one of the fastest alternatives to compress and decompress the video files, having a considerably high compression ratio. However, it does not compress as good as Codec3 does with default quantization parameters. Comparing this codec with the others lossy codecs, this one has a similar Average SNR results and Perceptual Errors are not highly noticeable.

Codec3 with the default quantization parameters behaves as a better alternative to compress a video file, if time restrictions are not as relevant as the compression ratio. It compresses information better than any other lossy codec.

For the codec3 using the DCT parameters, its compression ratio are slightly worst than most of the lossy codecs and its the the slowest of them all to conclude the encoding process, due to all the involving operations.

For the lossless codecs, the hybrid codec serves better results on the smaller videos, but seems to get slightly worse compression on bigger ones. We believe this happens because of the threshold we set to switch between I-Frames and P-Frames. To improve these results, we would have to code for every macro block the I-Block equivalent and compare the scores to decide the best one.

TABLE X
COMPRESSION RESULTS FOR CODEC 1 USING DEFAULT PARAMETERS

File	Encoding time	Decoding time	Output size	Compression ratio
foreman 4:2:0	10s	3s	21.6MiB	2.01
football 4:2:2	15s	5s	35.6MiB	1.96
akiyo 4:2:0	10s	3s	16.9MiB	2.57
ducks_take_off 1080p	7m08s	1m49s	932MiB	1.59
ducks_take_off 720p 4:4:4	4m10s	1m29s	804MiB	1.64
park_joy 1080p	7m48s	1m36s	965MiB	1.53

TABLE XI
COMPRESSION RESULTS FOR CODEC 2 USING DEFAULT PARAMETERS

File	Encoding time	Decoding time	Output size	Compression ratio
foreman 4:2:0	20s	3s	20MiB	2.18
football 4:2:2	26s	5s	33.9MiB	2.05
akiyo 4:2:0	11s	2s	10.7MiB	4.07
ducks_take_off 1080p	18m24s	2m01s	947.5MiB	1.57
ducks_take_off 720p 4:4:4	10m04s	1m29s	815.4MiB	1.62
park_joy 1080p	18m21s	1m54s	939.6MiB	1.58

TABLE XII
COMPRESSION RESULTS AND SNR FOR CODEC 3 USING DEFAULT QUANTIZATION PARAMETERS

File	Encoding time	Decoding time	Output size	Compression ratio	Avg SNR
foreman 4:2:0	20s	3s	7.4MiB	5.88	32.7
football 4:2:2	28s	4s	12.0MiB	5.80	31.5
akiyo 4:2:0	22s	2s	6.3MiB	6.90	33.7
ducks_take_off 1080p	13m17s	1m.5s	287.7MiB	5.16	30.3
ducks_take_off 720p 4:4:4	6m44s	59s	242.1MiB	5.45	30.4
park_joy 1080p	13m08s	1m.3s	332.2MiB	4.47	30.8

TABLE XIII
COMPRESSION RESULTS AND SNR FOR CODEC 3 V2 (DCT)

File	Encoding time	Decoding time	Output size	Compression ratio	Avg SNR
foreman 4:2:0	4s	4s	9.8MiB	4.44	34.4
football 4:2:2	6s	5s	14.2MiB	4.90	32.1
akiyo 4:2:0	4s	5s	7.2MiB	6.04	34.7
ducks_take_off 1080p	2m44s	3m40s	439.5MiB	3.38	30.1
ducks_take_off 720p 4:4:4	1m42s	1m36s	249.7MiB	5.28	28.9
park_joy 1080p	3m09s	2m51s	458.6MiB	3.24	30.7

TABLE XIV
COMPRESSION RESULTS AND SNR FOR CODEC 3 USING DEFAULT DCT PARAMETERS

File	Encoding time	Decoding time	Output size	Compression ratio	Avg SNR
foreman 4:2:0	18s	2s	9.4MiB	4.63	28.4
football 4:2:2	34s	4s	15.0MiB	4.64	30.9
akiyo 4:2:0	23s	2s	9.1MiB	4.78	32.8
ducks_take_off 1080p	18m26s	2m54s	554.6MiB	2.68	29.9
ducks_take_off 720p 4:4:4	8m3s	1m09s	301.8MiB	4.37	27.2
park_joy 1080p	17m34s	2m44s	513.9MiB	2.89	30.8