Audio and Video Coding Lab work n^{o}3 Universidade de Aveiro

Miguel Maia, *Universidade de Aveiro, 76434*
Silvério Pereira, *Universidade de Aveiro, 76505*
Joao Amaral, *Universidade de Aveiro, 76460*

***Abstract*—This reports describes the design and implementation of video encoders and decoders developed for the realization of the third practical project of the Audio and Video Coding course. The purpose of the project was to study and explore existing techniques of encoding video files and to develop our own intra-frame and inter-frame encoders, with and without loss of information, and in the end evaluate the obtained compression results.**

I. Exercise 1

The goal of this exercise was to develop a video player with capabilities of reproducing video in the formats RGB, YUV/YCbCr 4:4:4, 4:2:2 and 4:2:0. In order to do so we adapted the code already supplied by the teachers, that loaded a video file and displayed it, and extended it to work with the previous formats. In order to better structure and abstract the code, the class *FrameParser* was created. In this class the following methods were implemented in order to process and transform data of a given frame:

- *planarToPackedRGB*
- *planarToPackedYUV*
- *getPlanarChannels*
- *getPlanarResiduals*
- *yuv2rgb*
- *rgb2yuv*
- *parseHeader*
- *mat2vec*

A. FrameParser class

The class is initialized using the parseHeader method that takes a string corresponding to the header of the video file that contains the necessary metadata do process it. Here, the size of the file (number of rows and columns) is processed along with its stored format and respective frames per second. After processing the header, it is calculated the size of the matrices that will be used to store the channels values. Depending on the format, the size varies according to:

```
ySize = yCols * yRows;
if(format == "444"){
    frameSize = ySize * 3;
    uvCols = yCols;
    uvRows = yRows;
}
else if(format == "422"){
    frameSize = ySize * 2;
    uvCols = yCols / 2;
    uvRows = yRows;
}
else{
    frameSize = ySize * 3/2;
    uvCols = yCols / 2;
    uvRows = yRows / 2;
}
uvSize = uvCols * uvRows;
```

In the *PlayerYUV.cpp* file, after processing the header and obtaining the frame size, the matrix is initialized. Due to the fact that the video is stored in YUV planar mode but OpenCV uses packed mode it is necessary to convert to packed mode, which is done in the following segment of code:

```
buffer = (uchar*)img.ptr();
for(int i = 0 ; i < yRows * yCols * 3; i += 3)
{
    parser.planarToPackedRGB(imgData, i, r, g, b);

    buffer[i] = b;
    buffer[i + 1] = g;
    buffer[i + 2] = r;
}
```

After reading and storing the frame channels values in packed mode it is now possible to create matrix that can be visualized as an image, allowing to reproduce all the frames of the file sequentially obtaining an approximate reproduction of the video.

B. Code execution

The command used to play a given video file (input file) is as follows:

```
./PlayerYUV <video_file>
```

II. Exercise 2

A. Exercise description

The objective of this exercise was to develop a lossless intra-frame video encoder with the requirements that it would implement the linear predictor of the JPEG-LS algorithm followed by entropy coding using Golomb codes.

B. Lossless intra-frame algorithm

A lossless frame encoder allows, as the name suggests, the encoding and posterior reconstruction of the original video frames , while not losing any information (bits) during this process. Being an intra-frame enconder results in the encoder exploiting only spatial redundancy, using only the information contained in the frame being process to perform the compression. The lossless predictive algorithms are usually characterized by the predictors they implement. For this exercise, the linear predictor of the offical norm of the JPEG-LS algorithm was used, as it was one of its requirements. The predictor can be represented as follows:

$$x' = \begin{cases} min(a,b) & if c \geq max(a,b) \\ max(a,b) & if c \leq min(a,b) \\ a+b-c & otherwise \end{cases} \quad (1)$$

	c	b	
	a	x	

Fig. 1: JPEGLS predictor

The simbol x represents the current pixel being processed while a,b and c represent its neighboor pixels. The predictor uses this information in order to make a prediction of x, x', which is then subtracted to x in order to obtain the residual value that will then be encoded using Golomb codes, and its resulting values written into the encoded outpout file.

C. Code description

In order to implement the intra-frame lossless algorithm specified in the last section, the class FrameParser, specified in Exercise I, was used and extended to support additional methods.

In the encoding operation the Golomb class is initialized and the video header metadata written into the output stream. Then the frames will be parsed and the method getPlanarChannels() will be used in order to obtain the y,u and v channels values from the file and store them as a matrix data structure. With the channels now extracted then each one will be encoded. In the first stage of the encoding, padding with zeroes is applied to the first row and first column pixels, since these had originally no neighboors, which that are necessary to calculate the prediction. The prediction is then obtained, the residuals calculated and followed by Golomb encoding. The m parameter of Golomb is estimated at the beginning, after parsing the first frame of the video, and is used repeatedly for every following frame. The encoded values are the one written to the output bitstream in order generate encoded file.

In the decoding operation the residuals of the y,u and v channels are read from the encoded binary file using the getPlanarResiduals() method. With the values of the channels, the prediction is then calculated, again using the JPEG-LS predictor, and the original values will be reconstructed.

D. Code execution

The command used to encode a given video file (input file), and generate its corresponding encoded binary file (output file) using the intra-frame predictor is as follows:

```
jpnegls -e  <video_file> <output_file>
```

The command used to decode a given binary file (input file), and originate the reconstructed original file (output file) , is the following:

```
jpnegls -d  <encoded_file> <video_file>
```

III. Exercise 3

A. Exercise description

The objective of this exercise was to develop a hybrid video encoder (inter and intra-frame encoding) with the intracoding peridocity, block size and and search area given to the program as parameters.

B. Hybrid compression algorithm

Hybrid video compression involves the classification and compression of two different types of video frames: I frames and P frames.

Type I frames refer to the frames that are meant to be compressed as intra-frame, using only in the process information local to the frame, compressed in isolation from the remaining video frames. For the exercise the already developed lossless intra-frame codec was used to compress type I frames.

Type P frames are frames that are temporally encoded, using a previously encoded type I frame in the process, refered to as the reference frame.

The periodicity, a tunable parameter of the encoder, is the number of frames of type P encoded between I type frames. An example of the frame classification scheme, with periodicity 3, can be seen as follows:

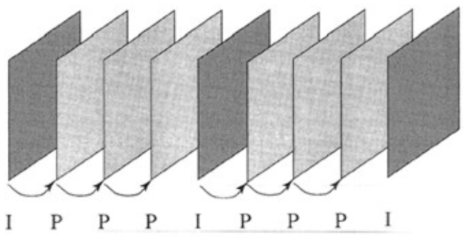

Fig. 2: Inter and intra frame classification

Another concept that is introduced in this class of encoders is that the frames, instead of being considered as a single unit that were the reference of the encoder algorithm, they are now divided into a grid, where each resulting square of the grid is named a macroblock. The macroblock is then considered as the unit used by the codec for the inter-frame process.

What this means is that now when a type P frame is to be encoded, the macroblocks will be parsed individually and matched with the corresponding macroblock in the reference frame (type I). The objective of this technique is to find any pixel displacement in the macroblocks being compared. In case there is a match and they are simillar, we can encode that macroblock referencing the previous intra-encoded one. On the other hand if there is any displacement of macroblocks in the neighboring pixels being compared, it is considered that there was movement in the video and a vector, known as motion vector, is calculated based on the unit displacement of the blocks being processed. This allows for better performance and compression rate since we are now storing the vector and a reference to the matched macroblock instead of the whole macroblock. This technique is called motion compensation.

The other two parameters of the encoder are the block size and the search area. The block size refers to the size of the macroblocks (squares) that will split the frames being processed and the search area refers to the number of pixel displacements in the reference macroblocks neighboors pixels the algorithm will use to extend its search.

An example of this encoding scheme can be seen in Figure 3. which depicts the technique used for the common hybrid video codec H.261.

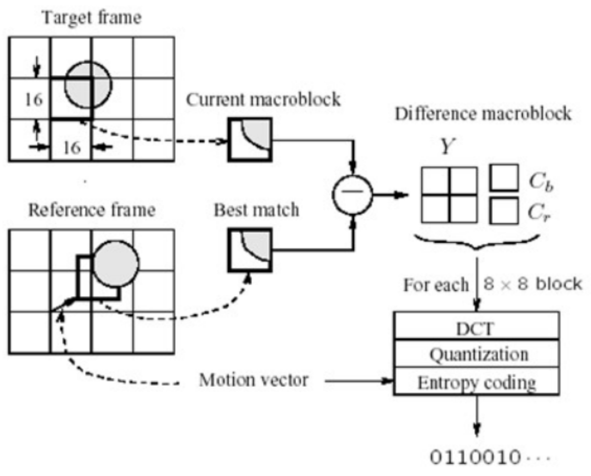

Fig. 3: H.261 inter-frame compression scheme

C. Code description

In order to implement the hybrid video encoder previously specified using motion compensation, the class FrameParser was, once again, extended to support the necessary new methods.

In the encoding operation, the encoder parameters block size, search area and periodicity are given by the user as input of the codec. The FrameParser class is initialized with the input file header metadata and the frames will be sequentially processed until the end. The channels of the frame are read and in the case it corresponds to an frame of type I (intra-frame) they are encoded with the encodeLS() method (described in Exercise II). Otherwise it will be an type P frame and the frame channels will be split in macroblocks accordingly with the given block size parameter.

With the frame now split in macroblocks, each one will be processed individually in the motionCompensationAllChannels() method. Here for each channel the motion vectors will be calculated and for every vector the residual calculated followed by the best error estimation for each channel.

The golomb M parameter is estimated for every P type frame encoded.

D. Code execution

The command used to encode a given video file (input file), and generate its corresponding encoded binary file (output file) using the hybrid encoder is as follows:

```
lhe -e|--encode <video_file> <encoded_file>
<block_size> <search_area> <periodicity>
```

The command used to decode a given binary file (input file), and originate the reconstructed original file (output file) , is the following:

```
lhe -d|--decode <encoded_file> <video_file>
```

IV. Exercise 4

A. Exercise description

The objective of this exercise was to extend the previously developed lossless intra-frame video codec in order to allow lossy compression. The program receives three new parameters indicating the quantization steps used for quantizing the prediction residuals of the three color components.

B. Lossy intra-frame compression algorithm

The quantization is performed for each of the color components and it's done in a similarly to the previous assignment

C. Code execution

```
jpneg_lossy -e|--encode <video_file>
<encoded_file> <y_quant> <u_quant>
<v_quant>

jpneg_lossy -d|--decode <encoded_file>
<video_file>
```

V. Compression statistics - lossless intra-frame encoder (Exercise 2)

File Name	Size of original File	Size of encoded file	Ratio	Encoding time	Decoding time
akiyo_cif.y4m	44M	21M	2.1	6.378 s	7.44 s
bowing_cif.y4m	44M	24M	1.8	6.469 s	7.333 s
carphone_qcif	14M	8M	1.8	2.374 s	2.753 s
ducks_take_off_420_720p50	660M	437M	1.5	110.954	124.085 s

VI. Compression statistics - hybrid encoder (Exercise 3)

Tests were done with a fixed periodicity of 3, a search area of 4 and a block size of 16

File Name	Size of original File	Size of encoded file	Ratio	Encoding time	Decoding time
akiyo_cif.y4m	44M	14M	3.2	58.211 s	6.829 s
bowing_cif.y4m	44M	19M	2.3	59.841 s	7.215 s
carphone_qcif	14M	7M	2	17.330 s	2.409 s
ducks_take_off_420_720p50	660M	445M	1.5	2019.008 s	324.912 s

VII. Compression statistics - lossy intra-frame encoder (Exercise 4)

Done with a quantization reduction of 2 for all channels

File Name	Size of original File	Size of encoded file	Ratio	Encoding time	Decoding time
akiyo_cif.y4m	44M	11M	4	5.118	6.260 s
bowing_cif.y4m	44M	11M	4	4.990 s	6.205 s
carphone_qcif	14M	5M	2.8	2.058 s	2.400 s
ducks_take_off_420_720p50	660M	304M	2.17	90.994 s	110.447 s

VIII. Distribution of work

Silvério Pereira - 50%
Miguel Maia - 20%
João Amaral - 30%