# Audio and Video Coding Lab work nº2
# 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 audio encoders and decoders developed during the realization of the second practical project for the Audio and Video Coding course. The purpose of the project was to study existing techniques of encoding audio files with and without loss of information, done by encoding the original file using less bits to represent the same information, and evaluate the obtained compression results.**

## I. Exercise 1

### A. Exercise description

The goal of this exercise was to develop an entropy coding method using the Golomb algorithm, with both encoding and decoding functionalities.

### B. Golomb algorithm

The Golomb algorithm is a common data compression method that is highly used for situations where the input stream follows a geometric distribuition and where it is optimal. It uses a tunable parameter M to divide the input value to be encoded, N, into two parts : q which represents the result of the division by M and r, the remainder of such division. These operations are as follows:

$$q = \left\lfloor \frac{N}{M} \right\rfloor, \quad r = N - qM \tag{1}$$

One particular concern that needed to be addressed was that the Golomb's scheme was designed to encode only sequences of non-negative numbers. However this problem is easily solved by the application of an overlap and interleave scheme technique to represent the digits being encoded, both negative and non-negative, which are reassigned to some positive number in a unique and reversible way. It works as follows:

$$x' = \begin{cases} 2x & , x \geq 0 \\ 2x - 1 & , x < 0 \end{cases} \tag{2}$$

During the encoding process the interleave is applied at the beginning of the encoding flow and the reverse operation is done at the end of the decoding process.

One other aspect that needs to be addressed the selection of the M value, since it is responsible for obtaining better compression ratios. For our approach the M calculation is later explained in this document in section II C.

### C. Code description

In the developed code this functionality is implemented by the Golomb class in the golomb.h file, which can be seen in Figure 1. It is composed by a BitStream custom class, developed during the previous project, that is used to write or read the encoded values to and from the file and is set during the class initialization. The tunable parameter M is also specified during initialization. It contains methods to encode and decode single as well as multiple values (endode(), decode(), encodeN(), decodeN()), to apply interleave and to reverse it (interleave(), reverse_interleave()), to write and read metadata information from the file (readMetadata(), writeMetadata()) and to set the value of M that changes from block to block that is being encoded (setM()).

```
class Golomb{

  private:

    BitStream bs;
    size_t m;
    size_t b;
    bool isMPow2;
    unsigned int t;
```

Fig. 1: Golomb class attributes

## II. Exercise 2

### A. Exercise description

The objective of this exercise was to develop and test a lossless linear predictive encoder, exploring temporal and channel redundancy. Its output values are then encoded with golomb code.

### B. Lossless linear predictive algorithm

In order to develop the lossless encoding predictive algorithm, a linear predictive approach was implemented. A lossless audio encoder allows, as the name suggests, the encoding and the reconstruction of the original audio signal, while not losing any information (bits) during the process. It has an average compression rate of 2:1 to 3:1 and is used in a wide area of applications such as archiving, transcoding,

distribution, online music stores, portable music players, DVD-Audio, and so on. The lossless predictive algorithms are usually characterized by the predictors they implement. For our developed solution, the most widely used class of predictors, linear predictors, were used. This predictor tries to estimate the value of the sample currently being processed , $x_k$, by the means of a linear combination of the M previous samples, $x_{k-1}, x_{k-1}, ..., x_{k-M}$. M is called the prediction order, and the prediction is denoted by $\hat{x}_k$ . The simplest predictors are fixed polynomial predictors, which assume the signal can be modeled by a small order polynomial. The most common polynomials are the following:

1) order 1, $\hat{x}_k$ = $x_{k-1}$
2) order 2, $\hat{x}_k$ = $2x_{k-1}$ - $x_{k-2}$
3) order 3, $\hat{x}_k$ = $3x_{k-1}$ - $3x_{k-2}$ + $x_{k-3}$

### C. Code description

In order to implement the linear predictive algorithm referenced in the last section, the class Predictors was created and is present in the file predictors.h. It contains four types of predictors: three of temporal type and their corresponding polynomial coefficients (defined as it's kernel) and one predictor of type inter-channel. It's structure can be seen in the following segment of code extracted from predictors.h:

```
class Predictor {

    public:
    std::vector<int> kernel;
    std::string name;
    const PredictorType type;
    const bool isTemporal;
    (...)
    switch(type){
        case PredictorType::TemporalOrder1:
            kernel = std::vector<int>{1};
            (...)
        case PredictorType::TemporalOrder2:
            kernel = std::vector<int>{2,-1};
            (...)
        case PredictorType::TemporalOrder3:
            kernel = std::vector<int>{3,-3,1};
            (...)
        case PredictorType::InterChannel:
            (...)
    }
   (...)
}
```

With the predictor class established the next step was to define the algorithm operations. The encoding operation was specified in the LPAencode.cpp file. It has two modes of operation: one for lossless and other for lossy encoding. The latter will be described in the next section. Before processing the data of the audio file the predictor is initialized with type of predictor and order, in case of temporal predictor, to be used in the encoding operation. The next step is to generate a handler (sndFileHandler) for the input audio file, through which its metadata will be extracted. In this step the histogram demonstrating the results of the operation will also be initialized with the number of channels of the audio file. The Golomb class, previously described, will also be initialized with the name of the output file and an output stream. The metadata previously extracted will then be written into the destination file header. This is done so there is information that facilitates the decoding process of the file without any errors or file corruptions. With the preprocessing steps done the next step is to process the audio frames. The audio file frames are read and processed in blocks, and for each frame read a estimate (precition) value will be calculated based on the order of the predictor, which will be subtracted from the current sample being read, resulting in a residual value. Each residual value obtained it stored in the "residuals" vector. Once a block is processed and all residuals have been calculated, the histogram is updated with these values. The value of Golomb parameter M is also here calculated, being estimated from the gathered residual values of the processed block. It is important to remember that, as we've seen before, M is a tunable parameter of the Golomb algorithm and its choice greatly influences the compression ratio obtained by the Golomb encoding operation. Its estimation works as follows:

1) Iterate through each residual value obtained through the processing of the block.
2) For each residual:
   a) Calculate the residual interleave value (remember that Golomb does not deal with negative values, so this step solves that problem)
   b) Sum and store all of the interleave resulting values
3) Obtain the number of residuals
4) Calculate the inverse average residual value of the block, denominated as p
5) the steps above effectively do parameter estimation for the p parameter of a geometric distribution, where $P(X = k) = (1 - p)^k * p$
6) The M value is then obtained by the following expression:

$$M = \left\lceil \frac{-1}{\log_2(1-p)} \right\rceil \tag{3}$$

With the value of M of the block now calculated, it is written into the destination file header, as block metadata, and the previously calculated residuals are ready to be encoded using the Golomb encodeN() operation which processes the residuals, calculates their interleaved values and writes the result into the destination binary file.

As for the decoding operation of the temporal prediction approach it is present in the LPAdecode.cpp file. Before processing the data of the binary input file the Golomb class is initialized with an input stream to read its content. The file metadata is then extracted from the file header and from this data the predictor is intialized accordingly to the predictor used in the enconding operation. Then a sound file handler (SndfileHandler) is created using the output file name and the metadata of the original file. The file contents will then be processed simillarly to the encoding approach: Process audio samples block by block and sample by sample of each corresponding block. In the block processing loop, it's metadata is read, the whole block is decoded using the Golomb

decodeN() method. Then, for each decoded frame, the estimate is obtained (prediction) and it is added to the current residual in order to obtain the original frame prior to encoding. Once a block is fully decoded it is then written into the destination decompressed audio file. The whole process is repeated until all the blocks have been processed.

One important detail that is worth noting is that, in both encoding and decoding operations of the temporal linear predictor, the frames from different audio channels are processed independently.

### D. Interchannel predictive algorithm

In this algorithm what differs is that to predict, we use values from the left and right channel.

X results from the subtraction of the sample from the left channel and the sample from the right channel divided by 2, Y results from the subtraction of the sample from the left channel and the sample from the right channel.

```
X = (samples[i] + samples[i+1])/2;
Y = samples[i] - samples[i+1];
```

After, the calculation of X and Y, these values are saved in the residuals array and sent to golomb.

```
residuals[i] = X;
residuals[i+1] = Y;
```

In the decoding process, the values are read from the residuals array and then, if y is odd then one must be added to X. If Y is even, the opposite calculation is done, multiplying X by 2.

```
if(Y % 2 != 0)
    LR = 2*X+1;
else
    LR = 2*X;
```

The values of the samples are recalculated reversing the operations in the encoding process.

```
samples[i+1] = (LR - Y)/2;
samples[i] = Y + samples[i+1];
```

After that, the values are written in the file.

### E. Code execution

The command used to encode a given audio file (input file), and generate its corresponding encoded binary file (output file) using the temporal linear predictor is as follows:

```
LPAencode --lossless <predictor order>
<input file> <output file>
```

The command used to decode a given binary file (input file), and originate the reconstructed original file (outputfile) , using the temporal linear predictor is the following:

```
LPAdecode <input file> <output file>
```

## III. Lossy predictive algorithm

### A. Exercise description

For this exercise it was intended to add an option to the previous encoder to compute lossy compression based on residual quantization and to compare these results obtained with the ones obtained in the previous project.

### B. Lossy predictive algorithm

In contrast with the lossless algorithms that were described in the previous sections, the lossy approach works by removing data from the audio samples being encoded, in a way that there is a balance between removed information while keeping the file still listenable for the user. This approach is widely used since it allows for the best ratios of compression while keeping the audio seamlessly intact of changes, when in reality a lot of information has been removed.

### C. Code description

The mode of operation both for the encoding and the decoding processed is very similar to the linear temporal approach. The major change here is that, in the encoding operation, the calculated residual suffers a transformation where bits are removed from its original representation value. It is done by right shifting the value by the result of the operation: ( 16 - quantsize ). Quantsize is a given program parameter and represents the amount of bits to be removed from the original value being encoded.

```
residual = residual >> (16 - quantsize)
```

Another additional step needed is that now we also remove representation bits from the calculated prediction value that will be used to estimate and obtain the original value in the decode process. It is as follows:

```
prediction + (residual << (16 - quantsize))
```

The remaining operations are the same as before, which means process the frames on the block and at the end of each block encode the values with Golomb and write them to file.

As for the decoding operation, the major change is, again, the representation of the values, that need to be restored to its original before compression. In order to do so it only requires to shift the read residual value and left shift the same amount of bits as in the encoding operation.

```
sample = (residual << (16-quantsize)) + prediction
```

Note that the residual values being read have already been decoded by the Golomb decode operation. The sample is then obtained normally by just adding the prediction with newly obtained residual.

### D. Code execution

The command used to encode a given audio file (input file), and generate its corresponding encoded binary file (output file) using the lossy linear predictor is as follows:

```
LPAencode --lossy <predictor order>
<input file> <output file> <quantisize>
```

The command used to decode a given binary file (input file), and originate the audio file (outputfile) using the lossy linear predictor is as follows:

```
LPAdecode <input file> <output file>
```

## IV. Compression rate - lossless predictive (Exercise 2)

| Predictor type | Size of original File | Size of encoded file | Ratio | Encoding time | Decoding time |
|---|---|---|---|---|---|
| Inter Channel | 5176796 (bytes) | 4376340 (bytes) | 0.845 | 2,532 s | 1,194 s |
| Temporal of order 1 | 5176796 (bytes) | 3806990 (bytes) | 0.735 | 2,033 s | 1,158 s |
| Temporal of order 2 | 5176796 (bytes) | 4284836 (bytes) | 0.827 | 2,490 s | 1,174 s |
| Temporal of order 3 | 5176796 (bytes) | 4388447 (bytes) | 0.847 | 2,717 s | 1,413 s |

## V. Compression rate - lossy coding (Exercise 3)

| QuantSize | Predictor type | Size of original File | Size of encoded file | Ratio | Encoding time | Decoding time |
|---|---|---|---|---|---|---|
| 10 | Inter Channel | 5176796 (bytes) | 2435140 (bytes) | 0.470 | 1.274 s | 0,705 s |
| 10 | Temporal of order 1 | 5176796 (bytes) | 1866491 (bytes) | 0.735 | 1,118 s | 0,360 s |
| 10 | Temporal of order 2 | 5176796 (bytes) | 2343567 (bytes) | 0.452 | 1,450 s | 0,798 s |
| 10 | Temporal of order 3 | 5176796 (bytes) | 2447541 (bytes) | 0.472 | 1,452 s | 0,915 s |
| 15 | Inter Channel | 5176796 (bytes) | 4052789 (bytes) | 0.783 | 2,141 s | 1,174 s |
| 15 | Temporal of order 1 | 5176796 (bytes) | 3483439 (bytes) | 0.673 | 1,818 s | 0,973 s |
| 15 | Temporal of order 2 | 5176796 (bytes) | 3961287 (bytes) | 0.765 | 2,316 s | 1,239 s |
| 15 | Temporal of order 3 | 5176796 (bytes) | 4064902 (bytes) | 0.785 | 2,463 s | 1,300 s |

| QuantSize | Predictor type | SNR Channel 0 | SNR Channel 1 |
|---|---|---|---|
| 10 | Inter Channel | 37.5405 | 43.5994 |
| 10 | Temporal of order 1 | 40,5419 | 40.6398 |
| 10 | Temporal of order 2 | 40.5419 | 40.9398 |
| 10 | Temporal of order 3 | 40.5419 | 40.9398 |
| 15 | Inter Channel | 71.8376 | 74.8923 |
| 15 | Temporal of order 1 | 71.8376 | 74.8923 |
| 15 | Temporal of order 2 | 74.7947 | 74.8942 |
| 15 | Temporal of order 3 | 74.7947 | 74.8942 |

## VI. Histograms

Histograms for the residuals for with various predictors on sample01.wav for which the orignal samples' histogram can be seen in 2, all of them have the histogram for the mono version (average) of the channels.

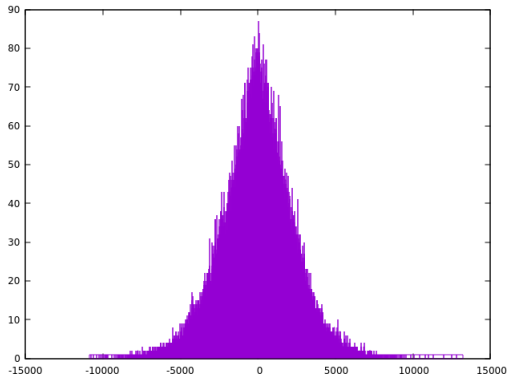


Fig. 2: Original samples

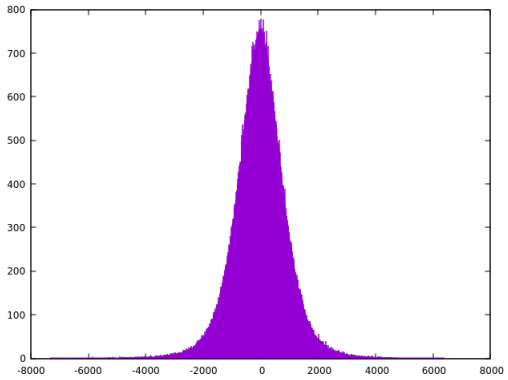


Fig. 3: Lossless spatial predictor residuals

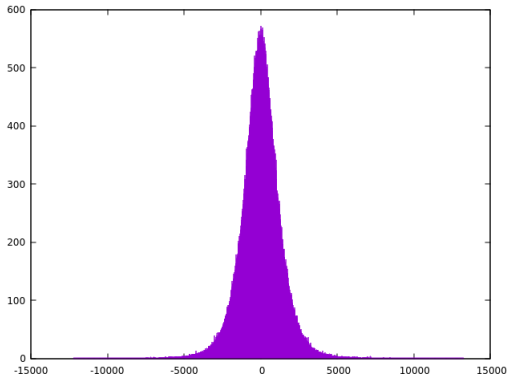


Fig. 4: Lossless temporal2 predictor residuals

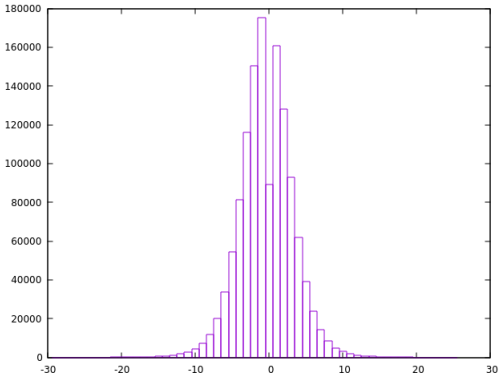


Fig. 5: Lossy spatial predictor residuals with 8 quantsize

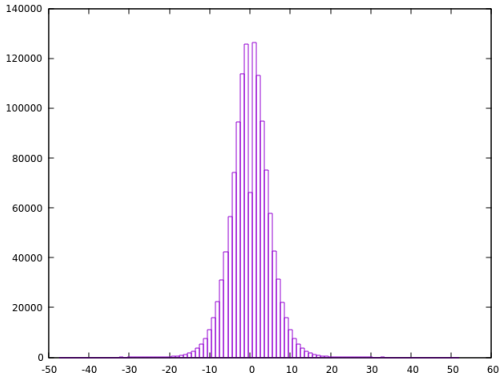


Fig. 6: Lossy temporal 2 predictor residuals with 8 quantsize

## VII. Distribution of work

Silvério Pereira - 40%
Miguel Maia - 30%
João Amaral - 30%