# Project IC - Third Project

Universidade de Aveiro

Bernardo Marçal, Ricardo Machado, Rui Campos

# Project IC - Third Project

Informação e Codificação

Universidade de Aveiro

Bernardo Marçal, Ricardo Machado, Rui Campos
(103236) bernardo.marcal@ua.pt, (102737) ricardo.machado@ua.pt
(103709) ruigabriel2@ua.pt

07 de janeiro de 2024

# Índice

# Capítulo 1

# Introduction

In the pursuit of efficient video compression, the implementation of a video codec using the Golomb coding algorithm emerges as a significant endeavor. This codec is specifically designed for grayscale video sequences and is characterized by its reliance on block-based motion compensation and spatial predictive coding. The chosen approach necessitates the utilization of uncompressed videos, where videos are represented in the uncompressed YUV4MPEG format.

To tailor the codec for grayscale functionality, an initial step involves the extraction of the luminance component from the original videos. This preparatory stage ensures that subsequent encoding processes focus solely on the grayscale information, optimizing the codec's performance for monochromatic video content. As the implementation progresses, the integration of Golomb coding, block-based motion compensation, and spatial predictive coding will collectively contribute to a robust video codec, offering efficient compression for grayscale video sequences.

The project's repository can be viewed here: https://github.com/ruicampos2/IC$_{P}roject$3.

All the command codes to test are available at the README file.

# Capítulo 2

# Project Workthrough

## 2.1 Exercise 2

### 2.1.1 Source Code

We began by crafting a class named Reader.h to retrieve information from the video file we are about to process. The primary goal is to extract details such as color space, aspect ratio, frame rate, and frame dimensions.

```
Reader(const std::string& file_name)
  : file_(file_name, std::ios::in | std::ios::binary) {
      if (!file_.is_open()) {
          throw std::runtime_error("Cannot open file: " + file_name);
      }

      // Parse the header of the file to extract the width, height, and color space.
      std::string line;
      // read first line
      std::getline(file_, line);

      // extract width and height
      sscanf(line.c_str(), " W%d H%d F%d:%d", &width_, &height_, &frame_rate_1_, &frame_rate_2_);
      // check if 'C' is in the line
      if (strchr(line.c_str(), 'C') == NULL) {
          color_space_ = "420";
      } else {
          // extract only the color space started by 'C'
          color_space_ = line.substr(line.find('C') + 1);
          std::cout << "here" << std::endl;
      }
      // check if 'I' is in the line
      if (strchr(line.c_str(), 'I') == NULL) {
          interlace_ = "p";
      } else {
          // extract only the interlace started by 'I' with the next character
          interlace_ = line.substr(line.find('I') + 1, 1);
      }
      // check if 'A' is in the line
      if (strchr(line.c_str(), 'A') == NULL) {
          aspect_ratio_1_ = 1;
          aspect_ratio_2_ = 1;
      } else {
          // extract only the aspect ratio started by 'A'
          std::string aspect_ratio = line.substr(line.find('A') + 1);
          // extract the first number
          aspect_ratio_1_ = std::stoi(aspect_ratio.substr(0, aspect_ratio.find(':')));
          // extract the second number
          aspect_ratio_2_ = std::stoi(aspect_ratio.substr(aspect_ratio.find(':') + 1));
```

Fig.1: *Reader.h* Code

Our intra-frame encoder operates on a frame-by-frame basis, meticulously processing each frame by identifying the key marker "FRAME."The encoder then systematically collates the frame's information into vectors of appropriate sizes, contingent upon the color space utilized in the video. This intricate process involves essential computations, ultimately culminating in the creation of an encoded file that encapsulates all pertinent details required for seamless video reconstruction, ensuring no loss of information.

```
//M VECTOR CALCULATION
for (long unsigned int i = 0; i < Yresiduals.size(); i++) {
    if (i % blockSize == 0 and i != 0) {
        int sum = 0;
        for (long unsigned int j = i - blockSize; j < i; j++) sum += abs(Yresiduals[j]);
        int mean = sum / blockSize;
        int m = calc_m(mean);
        if (m == 0) m = 1;
        Ym_vector.push_back(m);
        if (i < Cbresiduals.size()) {
            int sumCb = 0;
            int sumCr = 0;
            for (long unsigned int j = i - blockSize; j < i; j++) {
                sumCb += abs(Cbresiduals[j]);
                sumCr += abs(Crresiduals[j]);
            }
            int meanCb = sumCb / blockSize;
            int meanCr = sumCr / blockSize;
            int mCb = calc_m(meanCb);
            int mCr = calc_m(meanCr);
            if (mCb == 0) mCb = 1;
            if (mCr == 0) mCr = 1;
            Cbm_vector.push_back(mCb);
            Crm_vector.push_back(mCr);
        }
    }
    if (i == Yresiduals.size() - 1) {
        int sum = 0;
        for (long unsigned int j = i - (i % blockSize); j < i; j++) sum += abs(Yresiduals[j]);

        int mean = sum / (i % blockSize);
        int m = calc_m(mean);
        if (m == 0) m = 1;
        Ym_vector.push_back(m);
    }

    if (i == Cbresiduals.size() - 1) {
        int sumCb = 0;
        int sumCr = 0;
        for (long unsigned int j = i - (i % blockSize); j < i; j++) {
            sumCb += abs(Cbresiduals[j]);
            sumCr += abs(Crresiduals[j]);
        }
        int meanCb = sumCb / (i % blockSize);
        int meanCr = sumCr / (i % blockSize);
        int mCb = calc_m(meanCb);
        int mCr = calc_m(meanCr);
        if (mCb == 0) mCb = 1;
        if (mCr == 0) mCr = 1;
        Cbm_vector.push_back(mCb);
        Crm_vector.push_back(mCr);
    }
}

Golomb g;
int m_index = 0;
for (long unsigned int i = 0; i < Yresiduals.size(); i++) {
    if (i % blockSize == 0 and i != 0) {
        Ym.push_back(Ym_vector[m_index]);
        m_index++;
    }
    Yencoded += g.encode(Yresiduals[i], Ym_vector[m_index]);
    if (i == Yresiduals.size() - 1) {
        Ym.push_back(Ym_vector[m_index]);
    }
}
```

Fig.2: *program2enc.h Code*

Upon the completion of encoding residuals, a binary file is generated. The initial step in this phase involves the creation of a header, a crucial component containing indispensable information necessary for the comprehensive reconstruction of the video file. This includes parameters such as height, width, frames per second, aspect ratio, and other fundamental attributes. Additionally, optimal values of 'm' and all residuals vital for accurate frame reconstruction are inscribed in the header.

Conversely, on the decoding end, the process commences with a thorough examination of the header within the encoded file. The header serves as a repository of indispensable information that acts as a blueprint for the meticulous reconstruction of the video file, ensuring the preservation of all original data. Once all requisite readings are completed, the decoding process unfolds in tandem with the execution of the inverse prediction process, a crucial step in restoring the original video frames. This inverse process is intricately tailored to the specifics of the pixel in use, mirroring the encoding counterpart. Notably, the inverse prediction process may exhibit slight variations contingent upon the color space employed, particularly considering the nuanced presence or absence of U and V pixels at specific positions within the frame.

### 2.1.2 Results

The implementation of a lossless image codec for grayscale images, leveraging Golomb coding of prediction residuals, promises to deliver a robust solution for efficient compression. By strategically employing appropriate predictors and fine-tuning the values of 'm' for Golomb codes, the codec aims to achieve a high level of compression while ensuring the preservation of image fidelity. Golomb coding, known for its simplicity and effectiveness, plays a pivotal role in encoding prediction residuals, contributing to the overall compression efficiency of the codec.

The selection of suitable predictors is critical in enhancing the codec's performance. By employing predictors that effectively capture the inherent patterns and correlations within grayscale images, the codec can exploit redundancies and achieve optimal compression ratios. Additionally, the careful adjustment of parameter 'm' in Golomb coding further refines the compression process, striking a balance between compression efficiency and computational complexity.

Ultimately, the envisioned outcome of this endeavor is a lossless image codec that excels in compressing grayscale images while maintaining the integrity of the original data. The incorporation of Golomb coding, coupled with well-chosen predictors and parameter values, is anticipated to result in a powerful and versatile compression solution, meeting the demands for both efficiency and accuracy in the realm of grayscale image compression.

Analyzing the outcomes, it becomes evident that an increase in block size correlates positively with a higher compression ratio, with the optimal block size aligning with the video's width. Generally, smaller block sizes may yield more accurate representations of the original data but might result in larger file sizes due to the elevated quantity of residuals requiring quantization and enco-

ding. Conversely, larger block sizes may produce coarser data representations but contribute to smaller file sizes due to a reduced number of residuals. Additionally, it was observed that as the video file size expands, the compression ratio also increases, as larger videos tend to encompass more redundancy exploitable by the compression algorithm, thereby achieving more efficient compression. It is important to acknowledge that the compression ratio is contingent upon the video's content and may vary across different video formats not explored in this project and report.

## 2.2 Exercise 3

### 2.2.1 Source Code

Similar to the intra-frame encoder, the initial step in our process involves validating all user inputs. This crucial phase ensures that the input parameters provided by the user adhere to the required specifications and constraints. By meticulously verifying user inputs, we establish a foundation for the subsequent stages of the process, promoting accuracy and reliability throughout the execution of the encoder. This verification step is essential for safeguarding the integrity of the encoding process and facilitating a seamless and error-free operation.

```
// frames readed
while(!feof(input)){
    numFrames++;
    fgets(line, 100, input);
    for(int i = 0; i < width * height; i++){
        Y[i] = fgetc(input);
        if(Y[i] < 0) {
            numFrames--;
            finish = true;
            break;
        }
    }
    if (finish) break;
    for(int i = 0; i < width * height / 4; i++) U[i] = fgetc(input);
    for(int i = 0; i < width * height / 4; i++) V[i] = fgetc(input);
```

Fig. 3 : *program3enc.cpp* Code

The initial stage involves the common process of reading the input file and gathering information into the YUV vectors, akin to the intra-frame encoder. However, this program introduces a distinction in predicting future values. Guided by the keyframe period input, every keyframe frames encoded triggers the utilization of intra-frame prediction for the subsequent frame. For all other frames, inter-frame prediction is employed to capitalize on the similarity between adjacent frames in a video. Inter-frame prediction strategically leverages a reference frame, or keyframe, to predict the content of other frames, termed prediction frames. This involves dividing frames into blocks of specified sizes (blockSize) and searching for the best-matching block in the keyframe within a designated search area (searchArea), governed by additional input variables.

```
if(frameIndex==0){
    keyYmat = YMat.clone();
    keyUmat = UMat.clone();
    keyVmat = VMat.clone();
}
else if(frameIndex % keyFramePeriod == 0){
    keyYmat = YMat.clone();
    keyUmat = UMat.clone();
    keyVmat = VMat.clone();
}


if(frameIndex==0 || (frameIndex % keyFramePeriod==0)){
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            int Y = YMat.at<uchar>(i, j);
            int U = UMat.at<uchar>(i, j);
            int V = VMat.at<uchar>(i, j);
            int Yerror = 0;
            int Uerror = 0;
            int Verror = 0;
            if (i == 0 && j == 0) {
                Yerror = Y;
                Uerror = U;
                Verror = V;
                Yresiduals.push_back(Yerror);
                Cbresiduals.push_back(Uerror);
                Crresiduals.push_back(Verror);
            } else if(i==0){
                //if its the first line of the image, use only the previous pixel (to the left)
                Yerror = Y - YMat.at<uchar>(i, j-1);
                Yresiduals.push_back(Yerror);
                if (j < (width/2)) {
                    Cbresiduals.push_back(U - UMat.at<uchar>(i, j-1));
                    Crresiduals.push_back(V - VMat.at<uchar>(i, j-1));
                }
            } else if(j==0){
                //if its the first pixel of the line, use only the pixel above
                Yerror = Y - YMat.at<uchar>(i-1, j);
                Yresiduals.push_back(Yerror);
                if (i < (height/2)) {
                    Cbresiduals.push_back(U - UMat.at<uchar>(i-1, j));
                    Crresiduals.push_back(V - VMat.at<uchar>(i-1, j));
                }
```

Fig.4: *program3enc.cpp* Keyframe Code

Post prediction, three vectors encapsulate the residuals of all predicted frames, whether through inter-frame or intra-frame prediction. Following a process similar to the intra-frame encoder, optimal m values for Golomb coding are computed, and these values, along with motion vectors and residuals, are encoded into a binary file. The header of this file contains all necessary information for recreating the original video file.

On the decoding side, the decoder reverses the encoding process. After reading header information, motion vectors, optimal m values, and decoding residuals from the binary file, the decoder reconstructs the original frames, resulting in a video file identical to the initial one. During the inter-frame decoding phase, values are converted back to a vector, and each channel is appropriately arranged in the output file.

```
if(frameIndex==0){
    keyYmat = YMat.clone();
    keyUmat = UMat.clone();
    keyVmat = VMat.clone();
}
else if(frameIndex % keyFramePeriod == 0){
    keyYmat = YMat.clone();
    keyUmat = UMat.clone();
    keyVmat = VMat.clone();
}


if(frameIndex==0 || (frameIndex % keyFramePeriod==0)){
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            int Y = YMat.at<uchar>(i, j);
            int U = UMat.at<uchar>(i, j);
            int V = VMat.at<uchar>(i, j);
            int Yerror = 0;
            int Uerror = 0;
            int Verror = 0;
            if (i == 0 && j == 0) {
                Yerror = Y;
                Uerror = U;
                Verror = V;
                Yresiduals.push_back(Yerror);
                Cbresiduals.push_back(Uerror);
                Crresiduals.push_back(Verror);
            } else if(i==0){
                //if its the first line of the image, use only the previous pixel (to the left)
                Yerror = Y - YMat.at<uchar>(i, j-1);
                Yresiduals.push_back(Yerror);
                if (j < (width/2)) {
                    Cbresiduals.push_back(U - UMat.at<uchar>(i, j-1));
                    Crresiduals.push_back(V - VMat.at<uchar>(i, j-1));
                }
            } else if(j==0){
                //if its the first pixel of the line, use only the pixel above
                Yerror = Y - YMat.at<uchar>(i-1, j);
                Yresiduals.push_back(Yerror);
                if (i < (height/2)) {
                    Cbresiduals.push_back(U - UMat.at<uchar>(i-1, j));
                    Crresiduals.push_back(V - VMat.at<uchar>(i-1, j));
                }
```

Fig.4: *program3enc.cpp* InterFrame Code

Throughout the process, the program distinguishes between keyframes and non-keyframes, executing relative predictions accordingly. Despite undergoing a lossless codec that significantly compresses the file, the end result is a video file identical to the original, showcasing the effectiveness of the compression while preserving the integrity of the content.

### 2.2.2 Results

To enhance the codec to support inter-frame prediction, several key requisites must be addressed. Firstly, the encoder should incorporate the periodicity of intra-frames as an input parameter, allowing users to specify how frequently keyframes are inserted within the video sequence. Additionally, the codec

needs to accommodate the block size and search area as input parameters specifically tailored for inter-frame coding. These parameters play a crucial role in determining the granularity of motion estimation and the spatial extent within which the encoder searches for the best matching block during the inter-frame prediction process.

Furthermore, the encoder should implement a mechanism for estimating the coding mode for P frames dynamically, based on the bitrate produced. This entails determining whether the current block should be encoded in intra or inter mode, optimizing the trade-off between the two modes to achieve efficient compression while preserving video quality. This adaptive approach ensures that the codec can intelligently select the optimal coding mode for each block within P frames, allowing for a flexible and bitrate-conscious compression strategy. In summary, the desired outcome is a codec that effectively integrates these features, providing users with enhanced control over intra-frame periodicity, inter-frame coding parameters, and dynamic coding mode selection based on bitrate considerations.

Taking these factors into account, it becomes evident that the observed results align with expectations. In a video characterized by continuous movement, the challenge of finding similar blocks increases significantly when comparing a frame with one that is 10 or 15 frames old, resulting in a diminished compression ratio. Conversely, in videos with less movement and static elements it is anticipated that zones of similarity will be present when comparing with frames 5, 10, or 15 frames behind.

Interestingly, the impact of the search area on compression ratio appears negligible, particularly when there is minimal pixel shift between consecutive frames. Moving forward, as the file size expands, the encoding time decreases. While this observation challenges conventional expectations, a more comprehensive examination with a larger sample of video files would be necessary for a conclusive determination. Our hypothesis suggests that larger file sizes correspond to reduced video complexity have less complexity and movement. This conjecture proposes that a video's complexity influences processing time, where more complex videos demand additional time to identify suitable matching blocks, explaining the observed behavior.

To conclude, it is essential to recognize that intra-frame encoding outpaces inter-frame encoding in terms of speed. This discrepancy arises because inter-frame encoding involves not only predicting values but also calculating motion vectors. Consequently, a longer keyframe period, which means more instances of inter-frame encoding, results in increased encoding time. Interestingly, keyframe period and search area values exhibit minimal impact on decode time, as these parameters primarily facilitate the identification of the best and most similar blocks during the encoding process. During decoding, the decoder already possesses this information, rendering these parameters inconsequential to decoding time.

## 2.3 Exercise 4

### 2.3.1 Source Code

This task involves extending our codec to incorporate basic quantization of residuals, leading to a lossy output file. The modification applied to both encoders primarily entails quantizing the residuals post-calculation but prior to encoding. This addition introduces a controlled level of loss, as the precision of the residuals is reduced through quantization. The process involves mapping the continuous range of residual values to a discrete set of quantization levels, thereby introducing a degree of approximation.

By implementing simple quantization in this manner, the codec now introduces a trade-off between compression efficiency and the fidelity of the reconstructed video. The degree of quantization applied will determine the extent of loss in video quality, with higher levels of quantization leading to more compression but potentially compromising visual fidelity. This enhancement enables users to tailor the codec's performance based on their specific priorities regarding compression ratio and output video quality.

```
Golomb g;
int m_index = 0;
for (long unsigned int i = 0; i < Yresiduals.size(); i++) {
    if (i % blockSize == 0 and i != 0) {
        Ym.push_back(Ym_vector[m_index]);
        m_index++;
    }
    //only quantize the residuals that are not in the keyframe
    if((frameIndex !=0) && (frameIndex % keyFramePeriod != 0)){
        Yresiduals[i] = Yresiduals[i] >> quantization;
    }
    Yencoded += g.encode(Yresiduals[i], Ym_vector[m_index]);
    if (i == Yresiduals.size() - 1) {
        Ym.push_back(Ym_vector[m_index]);
    }
}
```

Fig.5: *program4interenc.cpp* InterFrame Encoder Code

```
for (long unsigned int i = 0; i < Yresiduals.size(); i++) {
    Yresiduals[i] = Yresiduals[i] >> quantization;
}
```

Fig.6: *program4intraenc.cpp* IntraFrame Encoder Code

In both encoders, the decision was made to exclusively quantize the channel responsible for conveying luminosity information in pixel composition. Through various tests, we explored the trade-off between quantizing all three channels (Y, U, and V) and solely quantizing the luminance channel. Surprisingly, the results indicated that the compression ratio remained nearly unchanged, whether information was removed from all three channels or solely from the luminance channel. This revelation provided a valuable flexibility to truncate more bits without significantly compromising the quality of the output file, as the decoded video retained the original colors, preserving crucial information in keyframes.

In the case of the inter-frame lossy encoder, a more nuanced strategy was employed. Quantization was applied exclusively to the information in non-keyframe frames of the video. This strategic choice was informed by the inter-frame encoding algorithm, where residuals are calculated based on a reference frame. The concern was the potential propagation of errors introduced by removing small bits of information. To mitigate this, keyframes were ensured to be original, containing all necessary information. Subsequent frames, however, were intentionally deprived of some information during reconstruction. The reverse operation, aimed at restoring correct values, then involved dealing with errors in only one of the contributing factors— the residual values.

In summary, the approach of selectively removing information solely from the luminance channel and restricting quantization to non-keyframe frames in the inter-frame encoding process allowed for the achievement of higher compression ratios. Remarkably, this optimization was achieved while maintaining better quality in the video's content, in contrast to a blanket removal of information from all frames across all channels.

```
Golomb g;
vector<int> Ydecoded = g.decodeMultiple(Yencodedstring, Ym, blockSize);
vector<int> Cbdecoded = g.decodeMultiple(Cbencodedstring, Cbm, blockSize);
vector<int> Crdecoded = g.decodeMultiple(Crencodedstring, Crm, blockSize);

int frameIndex = 0;
int total = padded_height*padded_width;

for (long unsigned int i = 0; i < Ydecoded.size(); i++){
    if (i % total == 0 and i != 0) {
            frameIndex++;
    }
    if((frameIndex !=0) && (frameIndex % keyFramePeriod != 0)){
        if(quantization != 1){

            Ydecoded[i] = Ydecoded[i] << 1;
            Ydecoded[i] = Ydecoded[i] | 1;
            Ydecoded[i] = Ydecoded[i] << (quantization - 1);
        }else{
            Ydecoded[i] = Ydecoded[i] << 1;
        }
    }
}
```

Fig.7: *program4interdec.cpp* InterFrame Decoder Code

```
for (long unsigned int i = 0; i < Ydecoded.size(); i++){
    if(quantization != 1){
        Ydecoded[i] = Ydecoded[i] << 1;
        Ydecoded[i] = Ydecoded[i] | 1;
        Ydecoded[i] = Ydecoded[i] << (quantization - 1);
    }else{
        Ydecoded[i] = Ydecoded[i] << 1;
    }
}
```

Fig.8: *program4intradec.cpp* IntraFrame Decode Code

When the quantization value deviates from 1, there is an opportunity to optimize error reduction in a more nuanced manner. This optimization involves

inserting half of the quantization cut value, as opposed to assigning all bits a value of 0. This strategic approach results in a value that is approximately closer to the original value before undergoing quantization, minimizing the introduced error.

However, when the quantization factor is precisely 1, it signifies that only 1 bit of information was discarded during the quantization process. In this scenario, the recovered value for that bit is set to 0. This distinction in handling quantization values different from 1 versus exactly 1 allows for a more tailored and nuanced approach to error reduction, optimizing the trade-off between compression and fidelity in the quantization process.

### 2.3.2 Results

Upon analyzing the results, it is evident that the encoding time exhibits a decrease with an increase in the number of quantized bits. Despite processing the same number of frames during encoding, the data load can still vary depending on the number of quantized bits. Quantizing bits lowers the resolution of residuals, resulting in reduced information. Consequently, fewer non-zero residuals post-quantization require representation, leading to a potential decrease in the data that needs processing during encoding, thereby contributing to a decrease in encode time.

The achieved compression ratio increases proportionally with the number of removed bits, as expected. Generally, quantizing more bits diminishes the precision of data, leading to a corresponding reduction in file size. Consequently, the compression ratio may exhibit a linear decrease as the number of quantized bits increases.

Interestingly, the amount of bits removed does not significantly impact encoding time. This is attributed to the fact that the computational intensity lies primarily in comparing the current frame to the reference frame and generating residuals, with this process outweighing the relatively swift quantization step.

When examining the effect of quantization on the achieved compression ratio, a logarithmic increase is observed. Beyond a certain point, the compression ratio experiences minimal improvement. This phenomenon arises from the fact that, with fewer bits, the values being quantized often already approach 0. Consequently, the impact of cutting more bits is primarily felt by residuals with higher values, which are fewer in quantity. This trend contrasts with intra-frame encoding, mainly due to the superior predictive algorithm in inter-frame encoding, resulting in smaller residual values that are more significantly affected by quantization.

### 2.3.3 Conclusions

In the culmination of this project, a comprehensive exploration of a video codec utilizing Golomb coding, block-based motion compensation, and spatial predictive coding for grayscale video sequences has unfolded. The following key insights and conclusions emerge from the implementation and evaluation of the codec:

1. **Golomb Coding Efficiency**: The Golomb coding algorithm showcased commendable efficiency in representing residual data, resulting in a notable reduction in file size. Its adaptability to grayscale video content proved valuable in achieving effective compression.

2. **Motion Compensation Impact**: Block-based motion compensation significantly contributed to the compression process by mitigating temporal redundancies. The codec's ability to analyze and compensate for motion within video sequences enhanced overall compression efficiency.

3. **Visual Quality Assessment**: The reconstructed video's visual quality was subject to examination, considering the inevitable loss of information during compression. Balancing the achieved compression ratio with perceptual video quality is crucial for evaluating the codec's practical utility.

4. **Compression-Quality Trade-off**: The project delved into the trade-off between compression ratio and visual quality, demonstrating that altering the number of quantized bits directly influenced the codec's performance. Striking an optimal balance between compression gains and perceptual losses is a key consideration.

5. **Codec Flexibility**: The codec's adaptability to diverse grayscale videos was assessed, emphasizing its flexibility in handling varied content and motion patterns. The implementation's robustness across different scenarios is vital for its applicability in real-world use cases.

6. **Comparison with Other Codecs**: While the project may not explicitly compare the developed codec with other established codecs, future work could explore benchmarking against industry standards to ascertain its competitiveness in terms of compression efficiency and visual fidelity.

7. **Encoding and Decoding Times**: Evaluating the time complexity of the encoding and decoding processes revealed insights into the codec's computational efficiency. Considerations such as block size, video complexity, and quantization levels played pivotal roles in determining these times.

# Capítulo 3

# Repository and Contributions

If there is any doubts about any part of the code that was put into this report, here is the github repository that has every part of the code that we have worked on and every file:

https://github.com/ruicampos2/IC$_{P}roject3$

All the compilation methods are inserted in the Readme in the main page of the repository. The contributions of the group work to this project are the following:

Bernardo Marçal (103236) - 33%
Ricardo Machado (102737) - 33%
Rui Campos (103709) - 33%