Universidade de Aveiro

Informação e Codificação

Bruno Lemos 98221, Tiago Marques 98459, João Viegas 98372

22 de janeiro de 2023

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório visa descrever a resolução do Projeto 3 desenvolvido no âmbito da unidade curricular de Informação e Codificação. O código desenvolvido para o projeto encontra-se disponível em :
https://github.com/brunolemos06/IC_Project3.

Para executar os comandos seguintes é necessário estar no diretório:
IC_Project3/src
Para compilar todos os programas
make
Para eliminar os executáveis
make clean

Executar o exercício 1
Encode: ../bin/codec_video_tests encode <YUV420 or YUV422> <video_to_encode.y4m> <out_encoded.txt>
Opt. encode args: <-mode [1, 8]> <-p [4, 32768]>
Decode: ../bin/codec_video_tests decode <encoded_frames_in.txt> <decoded_out.y4m>

Executar o exercício 2
Encode: ../bin/ex2_codec_video <input.y4m> <out.bin>
Opt. encode args: -blocksize <value> -searcharea <value> -keyframe <value>
Decode: ../bin/ex2_decoder_video <out.bin> <out.y4m>

Executar o exercício 3
Encode: ../bin/codec_video_tests encode <YUV420 or YUV422> <video_to_encode.y4m> <out_encoded.txt> <-lossy [1,7]>
Decode: ../bin/codec_video_tests decode <encoded_frames_in.txt> <decoded_out.y4m>

../bin/video_comp <input.y4m> <input.y4m> <max value of signal> <YUV420/YUV422>

Importante notar que os exercícios compilados de cada programa são guar-

dados na pasta **bin** e o código fonte na pasta **src**.

Capítulo 2

Parte I

Esta parte é dedicada a implementação de um codec intraframe para o formato de *YUV4MPEG2 (.y4m)* que pode estar comprimido com os seguintes formatos de compressão *YUV420* e *YUV422*.
Este codec possuí opções para escolher entre 8 modos de previsão de valor entre píxeis e qual a periodicidade de atualização do valor de m.

2.1 Formatos YUV suportados

2.1.1 YUV420

Neste formato de compressão, no espaço de cores planar YCbCr, cada bloco 2x2 de píxeis numa imagem é representado por 4 amostras de Y, 1 para cada pixel, mas todos os 4 píxeis partilham a mesma amostra de Cb e de Cr, ou seja, para a representação de 4 píxeis são necessários 6 valores.

Codificação e Descodificação

Para o processo de codificação do vídeo, é em primeiro lugar criado um *header* com todas as informações necessárias para realizar o processo de descodificação e de seguida é feita a codificação de cada frame desse vídeo. Assim sendo o processo de codificação de uma só *frame* consiste em codificar primeiro os valores do espaço Y, começando pela primeira coluna e primeira linha são codificados os valores originais pertencentes a essa região, enquanto que para os restantes valores desse espaço de cor é calculado o resíduo do valor de Y do pixel em questão e codificado o resultado.
Por sua vez, como blocos de 2x2 píxeis partilham o mesmo valor de Cb e Cr, a frame é agora divida em blocos dessa dimensão e lemos cada um desses blocos como se tratasse de um só pixel, na codificação desse tal 'pixel' é primeiro codificado o valor de Cb e depois o valor de Cr. Assim sendo, tal como na codificação dos valores do espaço Y, são primeiramente codificados os valores dos píxeis da primeira coluna (Cb e Cr), e logo de seguida os valores dos píxeis

da primeira linha(Cb e Cr). Por fim é realizado o cálculo residual dos valores (Cb e cR) dos píxeis restantes e codificados os valores resultantes.
Acabado o processo de codificação é gerado um ficheiro que contém o *header* e a informação codificada.

Figura 2.1: Exemplificação de codificação dos valores Cb e Cr dos píxeis no formato YUV420.

Para o processo de descodificação é lido o *header* do ficheiro codificado, que contém todas as informações necessárias para a descodificar o ficheiro e de seguida é iniciado o processo de descodificação *frame* a *frame*. Neste processo de descodificação de uma *frame*, são inicialmente lidos os valores originais da primeira coluna e primeira linha do espaço de cor Y, e consequentemente através da realização das operações inversas aos do cálculo residual de cada pixel obtemos assim o valor original de Y desse pixel. Para os espaços de cor Cb e Cr o processo é igual ao do espaço de cor Y, com a diferença que o processo de descodificação desses dois canais é feito em simultâneo, devido ao facto que ao ler um valor do canal Cb o valor seguinte pertence ao canal Cr, e que cada par de valores Cb e Cr corresponde a um bloco de 2x2 píxeis.
Ao acabar este processo de descodificação as *frames* resultantes são utilizadas para recriar o vídeo codificado anteriormente.

2.1.2 YUV422

Neste formato de compressão, no espaço de cores planar YCbCr, cada bloco 1x2 de píxeis numa imagem é representado por 4 amostras de Y, 1 para cada pixel, mas os 2 píxeis partilham a mesma amostra de Cb e de Cr, ou seja, para a representação de 2 píxeis são necessários 4 valores.

Codificação e Descodificação

Para o processo de codificação do vídeo, é em primeiro lugar criado um *header* com todas as informações necessárias para realizar o processo de descodificação e de seguida é feita a codificação de cada frame desse vídeo. Assim sendo o processo de codificação de uma só *frame* consiste em codificar primeiro os valores do espaço Y, começando pela primeira coluna e primeira linha são codificados os valores originais pertencentes a essa região, enquanto que para os restantes valores desse espaço de cor é calculado o resíduo do valor de Y do pixel em questão e codificado o resultado.
Por sua vez, como blocos de 1x2 píxeis partilham o mesmo valor de Cb e Cr, a frame é agora divida em blocos dessa dimensão e lemos cada um desses blocos como se tratasse de um só pixel, na codificação desse tal 'pixel' é primeiro codificado o valor de Cb e depois o valor de Cr. Assim sendo, tal como na codificação dos valores do espaço Y, são primeiramente codificados os valores dos píxeis da primeira coluna (Cb e Cr), e logo de seguida os valores dos píxeis da primeira linha(Cb e Cr). Por fim é realizado o cálculo residual dos valores (Cb e cR) dos píxeis restantes e codificados os valores resultantes.
Acabado o processo de codificação é gerado um ficheiro que contém o *header* e a informação codificada.

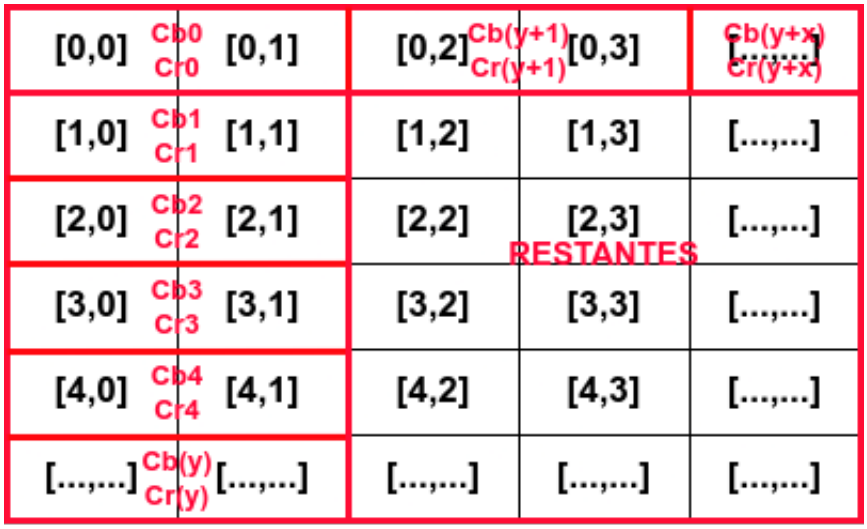


Figura 2.2: Exemplificação de codificação dos valores Cb e Cr dos píxeis no formato YUV422.

Para o processo de descodificação é lido o *header* do ficheiro codificado, que contém todas as informações necessárias para a descodificar o ficheiro e de seguida é iniciado o processo de descodificação *frame* a *frame*. Neste processo de descodificação de uma *frame*, são inicialmente lidos os valores originais da

primeira coluna e primeira linha do espaço de cor Y, e consequentemente através da realização das operações inversas aos do cálculo residual de cada pixel obtemos assim o valor original de Y desse pixel. Para os espaços de cor Cb e Cr o processo é igual ao do espaço de cor Y, com a diferença que o processo de descodificação desses dois canais é feito em simultâneo, devido ao facto que ao ler um valor do canal Cb o valor seguinte pertence ao canal Cr, e que cada par de valores Cb e Cr corresponde a um bloco de 1x2 píxeis.
Ao acabar este processo de descodificação as *frames* resultantes são utilizadas para recriar o vídeo codificado anteriormente.

2.2 Cálculo do Residual

O codec possui 8 formas de calcular o valor previsto, sendo elas:
Mode 1 -> $\hat{X} = a$
$Mode2 -> \hat{X} = b$
$Mode3 -> \hat{X} = c$
$Mode4 -> \hat{X} = a + b - c$
$Mode5 -> \hat{X} = a(b - c)/2$
$Mode6 -> \hat{X} = b + (a - c)/2$
$Mode7 -> \hat{X} = (a + b)/2$
$Mode8 ->$

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

Figura 2.3: Cálculo do Valor Previsto no Mode 8.

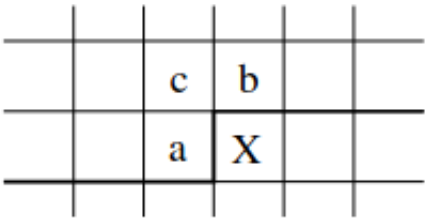


Figura 2.4: Posição relativa dos píxeis em torno do pixel X

Após calculado o valor previsto este é subtraído ao valor real do pixel e é obtido o valor residual.

$$Residual = X - \hat{X}$$

Este processo é realizado três vezes por pixel, visto que cada pixel contém 3 canais.

2.3 Resultados

2.3.1 *Compress Ratio*

O *Compress Ratio* é uma representação numérica simples do "poder de compressão" de codecs ou técnicas de compressão específicas. É uma medida da redução relativa no tamanho da representação de dados produzida por um algoritmo de compressão de dados. Normalmente, é expresso como a divisão do tamanho não compactado pelo tamanho compactado.

$$\frac{UncompressedSize}{CompressedSize}$$

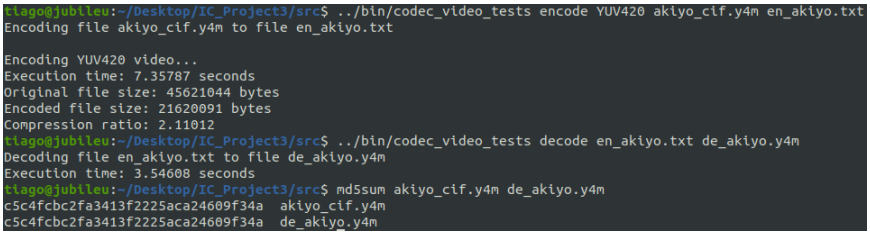


Figura 2.5: Codificação e descodificação do ficheiro akiyo_cif.y4m

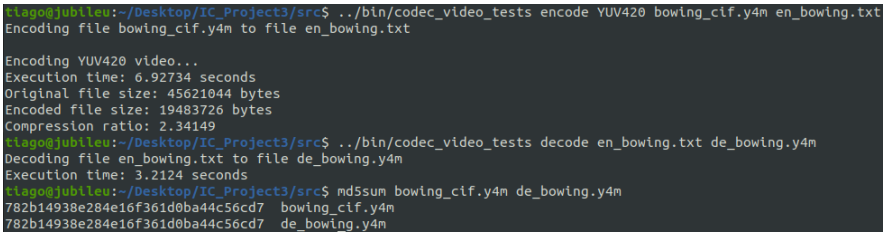


Figura 2.6: Codificação e descodificação do ficheiro bowing_cif.y4m

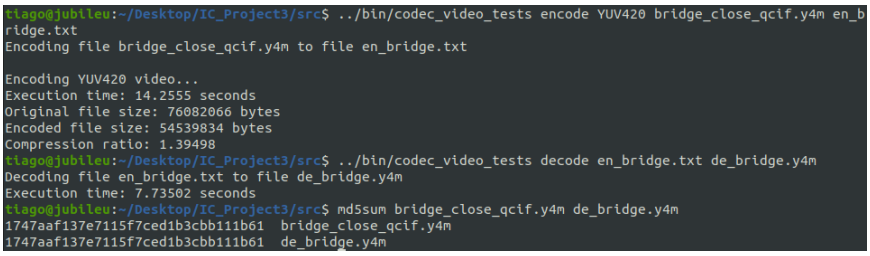


Figura 2.7: Codificação e descodificação do ficheiro bridge_close_qcif.y4m

Observando as imagens acima é nos visivél uma grande variação do valor do *Compress Ratio*, isto acontece devido à natureza do vídeo gravado, ou seja, se um vídeo possuir muito movimento e/ou uma palete de cores muito distintas este ira possuir uma taxa de compressão menor do que outro onde existe pouco movimento e tenha poucas cores. Este efeito acontece devido a este *codec* se basear em *intraframe*.
A maior taxa de compressão observada pertence ao ficheiro bowing_cif.y4m (figura 2.6).

2.3.2 Modo do cálculo do residual

Como já foi referido anteriormente este *codec* possui 8 maneiras diferentes de realizar o cálculo do valor previsto de um píxel, valor esse que depois de calculado será usado para obter o valor do resíduo. As imagens abaixo demonstram os resultados obtidos na utilização de alguns desses modos.

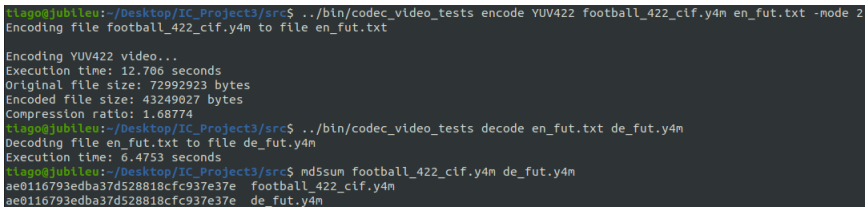


Figura 2.8: Codificação e descodificação do ficheiro football_422_cif.y4m com modo 2

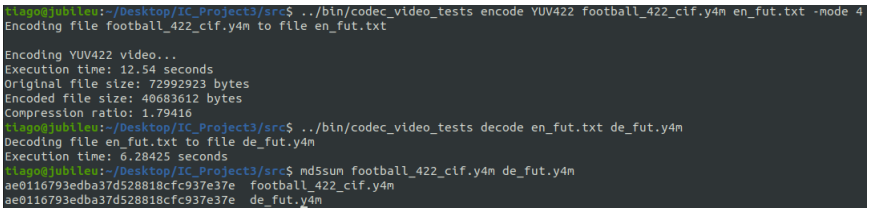


Figura 2.9: Codificação e descodificação do ficheiro football_422_cif.y4m com modo 4

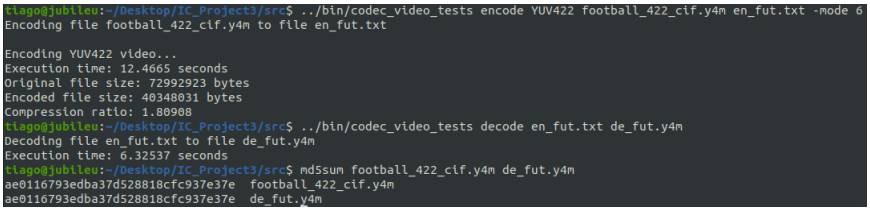


Figura 2.10: Codificação e descodificação do ficheiro football_422_cif.y4m com modo 6

Figura 2.11: Codificação e descodificação do ficheiro football_422_cif.y4m com modo 8

Em primeiro lugar, analisando a taxa de compressão, podemos observar que dependendo do modo usado o valor da taxa aumenta para modos de números mais altos e é mais pequena para modos mais baixos, isto deve-se à complexidade dos cálculos realizados para determinar o valor previsto. A escolha do método de cálculo do valor previsto também está relacionado com o tempo de codificação e descodificação visto que cálculos mais complexos exigem mais tempo a ser realizados que cálculos mais básicos.

2.3.3 Periodo

Neste *codec* existe um parâmetro 'p' que está relacionado com a periodicidade com que se atualiza, dinamicamente, o valor de m, valor esse usado no código de golomb para a codificação e descodificação. Ao realizar a operação periódica do cálculo de um novo m é assim possível obter uma taxa de compressão maior ao custo do tempo de execução para codificar/descodificar um vídeo. Assim sendo, podemos concluir, de acordo com as imagens abaixo que um período mais alto (menor valor de p) irá contribuir para uma maior taxa de compressão mas também para um tempo codificação/descodificação mais alto, devido ao serem realizadas mais operações de atualização do valor m. O contrário irá acontecer, menor taxa de compressão e menor tempo de codificação/descodificação, para um período mais alto (maior valor de p).

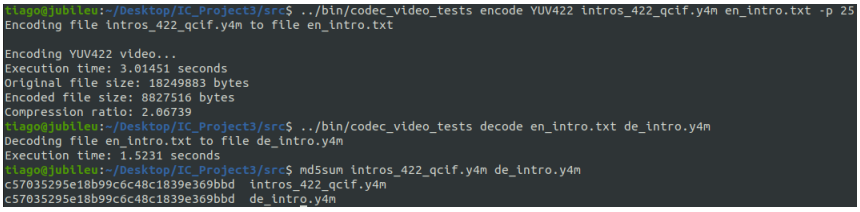


Figura 2.12: Codificação e descodificação do ficheiro intros_422_qcif.y4m com periodo 25

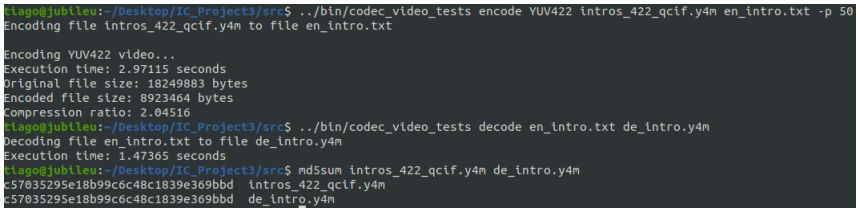


Figura 2.13: Codificação e descodificação do ficheiro intros_422_qcif.y4m com periodo 50

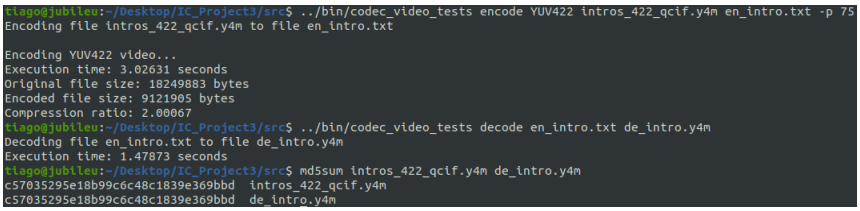


Figura 2.14: Codificação e descodificação do ficheiro intros_422_qcif.y4m com periodo 75

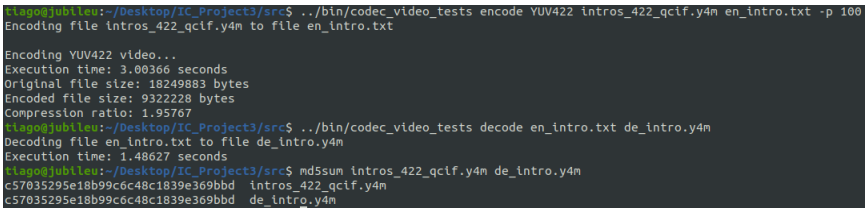


Figura 2.15: Codificação e descodificação do ficheiro intros_422_qcif.y4m com periodo 100

Nota: O valor dado a p refere-se ao número de valores des/codificados e não a uma unidade de tempo.
Exemplo: Se p=50 então a cada 50 valores des/codificados é atualizado o valor de m.

Capítulo 3

Parte II

Motion compensation é uma técnica utilizada para fazer compressão de vídeo através do movimento do vídeo. A ideia por trás da compensação de movimento é prever o movimento futuro do vídeo e ajustar a imagem de modo a minimizar o efeito do movimento no vídeo final.

3.1 Argumentos e suportes

Os argumentos obrigatórios para codificar são o ficheiro de codificação no formato y4m e o ficheiro de saída. (exemplo: out.bin)
../bin/ex2_codec_video <input.y4m> <out.bin>
Os argumentos opcionais são:

- -blocksize <default 16>
  Tem de ser divisível pelo *height* e *width* da imagem
- -searcharea <default 4>
- -keyframe <default 2>

Para este exercício foi suportado apenas o suporte para o Y420.

```
┌─(~/Projects/UA-ECT/4ano/IC/IC_Project3/src)──
└─(04:17:07 on main)─> ../bin/ex2_codec_video akiyo_cif.y4m out.txt --blocksize 16 --searcharea 8 --keyframe 4
Video info:
        Width: 352
        Height: 288
        FPS: 29
        Frames: 300
        Frame type: 16
        FourCC: 808596553
        Mode: 8
        Intra(1)/Inter(0): 1
        Period: 100
        Keyframe: 4
        BlockSize: 16
        Search Area: 8
Header: YUV4MPEG2 W352 H288 F30000:1001 Ip A128:117
```

Figura 3.1: Argumentos do codificador ex2

3.2 Algoritmo

3.2.1 Codificador

Este algoritmo basea-se em codificar de keyframes em keyframes com o modo *intra-frame* as restantes são codificadas através do modo *inter-frame prediction*, também conhecido como *motion compensation*.

Inicialmente é codificado o *height*,*width*, número de frames, header original, keyframe e blocksize. Estes valores são essenciais para posteriormente usarmos na descodificação.

No modo *inter-frame* a frame é **dividida em blocos** com o tamanho de **blocksize x blocksize** (Fig 3.1).

De seguida para cada bloco da frame atual é lhe aplicado uma search area (Fig 3.2) e é procurado um bloco mais parecido na search area da frame referência (keyframe). Para calcular o bloco mais semelhante foi usada SDA, "Soma de Diferenças Absolutas". SDA é uma técnica comumente utilizada na comparação de imagens. Ela envolve o cálculo da diferença absoluta entre os pixels correspondentes nas duas imagens, e então a soma dessas diferenças para obter um único valor que representa a diferença geral entre as imagens.

Figura 3.2: Divisão das frames em blocos

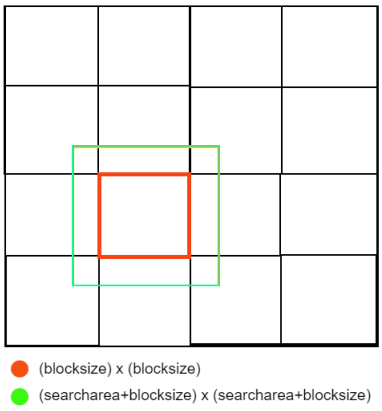


Figura 3.3: Search área em relação ao bloco

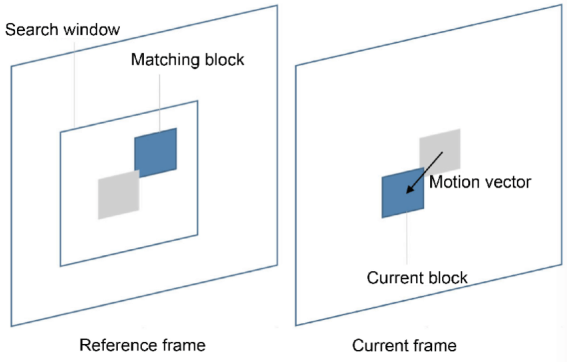
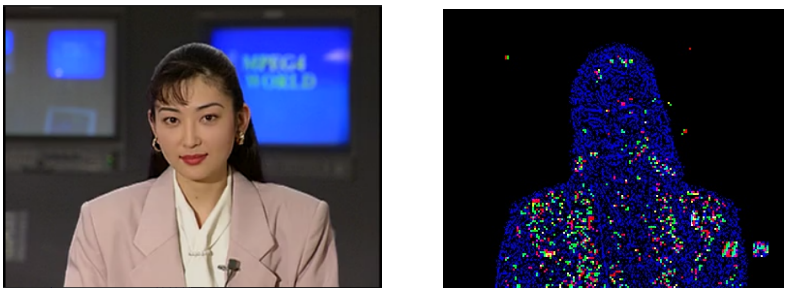


Figura 3.4: Estimativa do bloco dado uma search area

3.2.2 Descodificador

No descodificador o processo é todo invertido. Inicialmente é descodificado todos os valores iniciais para começar a descodificação. Esses valores são *height*,*width*, número de frames, header original, keyframe e blocksize. Todos os keyframes são descodificados com o modo intra-frame. No que toca às outras frames são descodificados todos os vetores associados à frame e de seguida é descodificada a *residual frame*. Para calcular a Frame Real através dos vetores e da *residual frame* é necessário para bloco aplicar a fórmula seguinte:

$$RealBlock = keyFrameblock \ + \ vector \ - \ Residualblock$$



(a) Key Frame (b) Residual Frame

Figura 3.5: Keyframe and ResidualFrame

3.3 Resultados

A descodificação de motion compensation é uma técnica amplamente utilizada na codificação de vídeo para reduzir a quantidade de dados necessária para transmitir ou armazenar vídeo. No entanto, é importante avaliar os resultados para garantir que a qualidade da imagem não é comprometida e que o tempo de execução é aceitável. Neste capítulo, apresentaremos os testes com diferentes conjuntos de dados para avaliar o tempo de execução e o impacto dos níveis de compressão na qualidade da imagem.

3.3.1 Tempos de execução

Foram calculados vários tempos de execução na codificação do vídeo *akiyo_ cif.y4m*. Os valores que foram variando foram a keyframe, blocksize e search_area.

Os valores default foram keyframe = 2, blocksize = 8 e search_area = 2 para cada um dos testes. Para os dados do keyframe foram alterados apenas os valores do keyframe e para os restantes o mesmo raciocínio.

Os tempos neste capítulo são expressos em segundos.

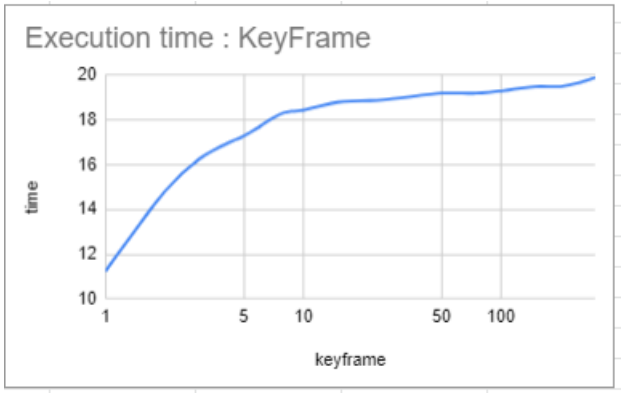


Figura 3.6: Gráfico da variação keyframe

keyframe	time
1	11,23
2	14,81
3	16,28
4	16,9
5	17,28
6	17,69
7	18,08
8	18,33
9	18,4
10	18,44
15	18,8
25	18,9
50	19,2
75	19,2
100	19,3
150	19,5
200	19,5
250	19,67
300	19,9

Figura 3.7: Resultados da variação keyframe

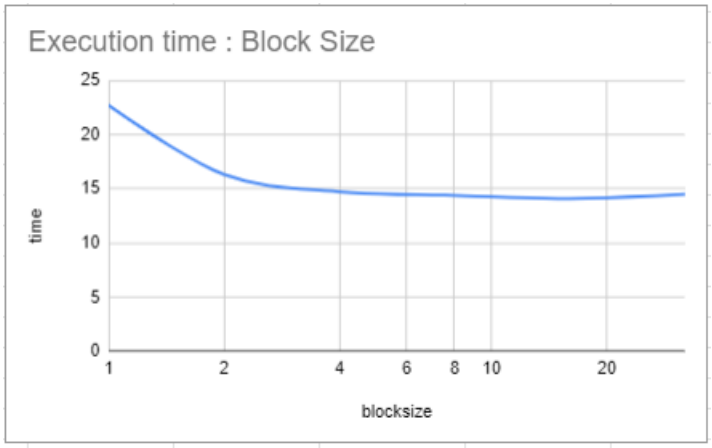


Figura 3.8: Gráfico da variação blocksize

blocksize	time
1	22,69
2	16,31
4	14,73
8	14,39
16	14,1
32	14,5

Figura 3.9: Resultados da variação blocksize

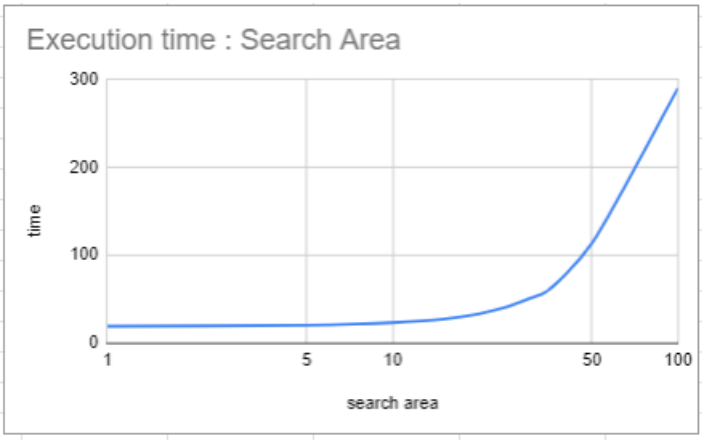


Figura 3.10: Gráfico da variação search_area

search area	time
1	18,97
5	20,12
10	23,08
15	26,95
20	33,04
25	40,6
30	50,29
35	59,33
50	113,3
100	290

Figura 3.11: Resultados da variação search_area

De seguida são apresentados os tempos de execução do decoder, podemos reparar numa primeira impressão que este tem menos computação associada e posteriormente um menor tempo de execução comparando ao encoder.

Vídeo	Intra Frames	Inter Frames	Total Frames	% of inter	% of intra	time
akiyo_cif.y4m	149	151	300	50,33	49,67	4,36
bridge_close_qcif.y4m	1001	1000	2001	49,98	50,02	9,7
bowing_cif.y4m	228	72	300	24,00	76,00	4,77
news_cif.y4m	202	98	300	32,67	67,33	5,1
carphone_qcif.y4m	320	62	382	16,23	83,77	1,92

Figura 3.12: Resultados da variação search_area

3.3.2 Níveis de compressão

Foram testados alguns vídeos para verificar o nível de compressão. Abaixo estão apresentados de acordo com os argumentos. De acordo com o vídeo de acordo com os argumentos podemos obter melhores resultados comparando com outros vídeos.

Vídeo	Configuração			Intra Frames	Inter Frames	Total Frames	Ratio	% of original
	Keyframe	Search Area	Block Size					
akiyo_cif.y4m	10	8	32	118	182	300	2,340	42,74
bridge_close_qcif.y4m	10	8	16	508	1493	2001	1,622	61,64
bowing_cif.y4m	3	4	32	203	97	300	2,468	40,52
paris_cif.y4m	2	4	32	648	417	1065	1,750	57,14
news_cif.y4m	2	8	32	181	119	300	2,140	46,73
carphone_qcif.y4m	2	3	16	352	30	382	1,788	55,93

Figura 3.13: Resultados sobre Nível de compressão

3.3.3 Validação

Para validarmos os resultados usámos um comando no linux - md5sum. Assim podemos comparar as hash's do ficheiro original e do ficheiro gerado através do compressor. Se as hash's são iguais os ficheiros são iguais e podemos comprovar que é um codificador lossless, sem qualquer perda de bits.

```
Execution time: 2.98308 seconds
┌─(~/Projects/UA-ECT/4ano/IC/IC_Project3/src)──────────────────────────
└─(03:35:58 on main)─> md5sum carphone_qcif.y4m out.y4m
2faa58f132677e9c6dc8650bd6d69f8f  carphone_qcif.y4m
2faa58f132677e9c6dc8650bd6d69f8f  out.y4m
```

Figura 3.14: Validação dos resultados - md5sum

3.4 Conclusões

A escolha entre o modo intra e o modo inter depende da quantidade de movimento presente no vídeo. Se houver pouco movimento, o modo intra é mais adequado, enquanto que se houver muito movimento, o modo inter é mais eficiente. No nosso codec temos os modos de forma dinâmica, e a escolha é feita em *real-time* de acordo com o tamanho que ocuparia cada modo.

E reparámos que em alguns casos pode ser vantajoso usar então uma combinação dos dois tipos de compensação para obter o melhor desempenho possível,

por outro lado se quisermos algoritmos mais rápidos podemos optar apenas por intra-frames.

Capítulo 4

Parte III

Neste capítulo vamos abordar a quantização. Quantização é um processo utilizado em codecs de vídeo para reduzir a quantidade de dados necessário para representar o conteúdo de um vídeo. Em codecs de vídeo, a quantização é usada para codificar os componentes de vídeo, como brilho e cores.

4.1 Quantização

Quando vamos a calcular o valor residual temos de quantizar o valor da seguinte forma:

$$Residual = (ValorReal - Predicao) >> Quantization$$

De seguida é necessário substituir o valor real da frame atual para o valor suposto depois pelo decoder, assim evitamos a propação de erros no resto da frame.

$$ValorReal = (Residual << Quantization) + Predio$$

Desta forma continuamos a codificar os valores sem ter mais perdas de bits de forma linear.

Podemos verificar nas imagens seguintes o que acontece quando a quantização é toma os devidos valores e reparar que a imagem vai perdendo as suas características á medida que aumentamos a quantização.

Figura 4.1: Vídeo akiyo_cif.y4m com quantização a 1

Figura 4.2: Vídeo akiyo_cif.y4m com quantização a 2

Figura 4.3: Vídeo akiyo_cif.y4m com quantização a 3

Figura 4.4: Vídeo akiyo_cif.y4m com quantização a 4

Figura 4.5: Vídeo akiyo_cif.y4m com quantização a 5

Figura 4.6: Vídeo akiyo_cif.y4m com quantização a 6

Figura 4.7: Vídeo akiyo_cif.y4m com quantização a 7

4.2 Video Compare

Para conseguirmos uma melhor perceção da qualidade das sequências de vídeo que foram criadas com o nosso codec de video lossy, desenvolvemos um programa videocmp. Este programa compara dois vídeos frame por frame e calcula a relação sinal ruido. Para calcular esta relação, utilizamos as fórmulas demonstradas a baixo.

$$\mathrm{PSNR} = 10 \log_{10} \frac{A^2}{e^2},$$

Figura 4.8: Formula de cálculo relação sinal-ruido

$$e^2 = \frac{1}{NM} \sum_{r=1}^{N} \sum_{c=1}^{M} [f(r,c) - \tilde{f}(r,c)]^2,$$

Figura 4.9: Formula de cálculo do erro médio do frame reconstruido

A seguir, apresentamos uma tabela com os resultados do cálculo da relação sinal-ruído entre quatro vídeos.

PSNR	akiyo	bowing	Footbal	Intros	
Y	8.92	08.09	17.15	17.17	
U	17.58	18.24	17.15	17.11	2 Bits
V	17.86	18.00	17.15	17.13	
Y	02.04	0.78	3.67	3.68	
U	3.90	6.78	3.58	3.25	4 Bits
V	4.58	5.15	3.45	3.93	
Y	-15.13	-9.48	-11.06	-11.74	
U	-9.17	-5.72	-8.98	-9,55	6 Bits
V	-5.85	-8.46	-9.50	-9.24	

Figura 4.10: Valores de relação sinal-ruido

Como podemos observar na tabela acima, os vídeos com uma quantização de 2 bits apresentam uma relação sinal-ruído superior em relação aos videos com uma quantização de quatro e seis bits, o que indica que ele possui uma qualidade de imagem um bastante melhor. Isso pode ser explicado pelo fato de que a relação sinal-ruído é uma medida da qualidade de imagem de um vídeo, quanto maior essa medida, melhor será a qualidade do vídeo. Portanto, podemos concluir que quanto mais bits de quantização menor é a qualidade do video.

Contribuições dos autores

Todos os autores participaram de forma igual na divisão, desenvolvimento e discussão deste trabalho pelo que a percentagem de contribuição para cada aluno fica:

- Bruno Lemos - 33.3%
- Tiago Marques - 33.3%
- João Viegas - 33.3%