Data Compression and Video Coding

Complementos Sobre Linguagens de Programação

Tiago Melo 89005

João Nogueira 89262

Introdução

Este relatório visa descrever o trabalho desenvolvido na realização deste projeto, dando conta das linguagens utilizadas, classes e funcionalidades desenvolvidas, e respetivos programas de teste e sua utilização.

Classes e funcionalidades desenvolvidas

Todo o projeto foi desenvolvido exclusivamente em Python, não tendo sido feito qualquer código funcional em C++, de acordo com as prioridades e objetivos estabelecidos ao longo do tempo na cadeira de CSLP, face às circunstâncias.

O projeto foi assim trabalhado de acordo com as fases estabelecidas no guião, que iremos agora abordar.

Data Compression – Bitstream

A classe **Bitstream.py** contém os métodos desenvolvidos para ler e escrever bits para ficheiros binários. Esta classe é inicializada com o nome do ficheiro e um de dois modos, mutuamente exclusivos, o de 'READ', para leitura, e o de 'WRITE', para escrita.

É então possível a escrita de um ou mais bits (writebit() e writebits()), a leitura de um ou mais bits (readbit() e readbits()), a leitura ou escrita de um valor (numérico) com N número de bits (write_n_bits() e read_n_bits()), abortando o programa caso não seja possível escrever dado valor com esse número de bits, e ainda a escrita de texto, com 8 bits, método útil para escrever headers com as informações dos vídeos codificados.

A classe foi utilizada com sucesso para ler e escrever bits respetivos aos pixéis de cada frame do vídeo codificado, na fase de Video Coding.

Está ainda presente um programa de teste unitário da classe, 'bitstream_test.py'.

Data Compression – Golomb

A classe **Golomb.py** contém métodos respetivos para codificar números e descodificar sequências de bits, sendo inicializada com um parâmetro M, respetivo ao fator do Golomb.

A implementação da codificação é feita com a sequência respetiva ao quociente, codificada em unário (com X número de 1's e um 0 como terminador) seguida da sequência respetiva ao resto, codificada em binário, tendo em atenção que o número de bits dessa sequência deve ser igual ao logaritmo de 2 do fator inicializado no Golomb.

É de notar que não foi implementada a codificação/descodificação em truncated binary encoding, para parâmetros de M!=2^x (bem como qualquer método de análise para o melhor valor de M baseado no vídeo apresentado), sendo que o Golomb só deve ser então utilizado para valores de M que sejam potências de 2.

A classe foi utilizada com sucesso para codificar e descodificar os bits da Bitstream, relativos aos pixéis de cada frame dos vídeos codificados, na fase de Video Coding.

Está ainda presente um programa de teste unitário da classe, ‘golomb_test.py’.

Video Coding – Video Player

A classe **VideoPlayer.py** contém os métodos necessários para ler o ficheiro relativo a um vídeo em YUV, e reproduzir o mesmo no terminal.

Para tal, toda a informação relativa ao vídeo, é lida e guardada em estruturas apropriadas, sendo os valores dos pixéis guardados em arrays de NumPy com shape adequada consoante o color space e subsampling dos componentes Y U V do vídeo.

Antes da conversão para RGB, é necessário, caso estejamos perante os formatos 422 ou 420, o resizing dos arrays respetivos de cada componente do vídeo (um array de arrays NumPy para cada componente Y,U,V com tamanho igual ao número de frames) fazendo com que os arrays das componentes U e V tenham o mesmo formato e quantidade de informação como a da componente Y. É para isso utilizado o método ‘resize’ da biblioteca NumPy. São então convertidos todos os píxeis de cada frame do vídeo para RGB, através de contas aritméticas, e mostrados no ecrã através da biblioteca OpenCV do Python.

Por questões de desenvolvimento do projeto, cada frame do vídeo está a ser mostrada durante 1 segundo, antes de passar para o seguinte, podendo isto ser mudado ao alterar o valor ‘1000’ no método ‘play_video()’.

A classe foi utilizada com sucesso para mostrar vídeos YUV no terminal, em qualquer dos formatos (444,422,420) e ainda para verificar visualmente a correta codificação e descodificação de vídeos nas fases posteriores do projeto, de Intra e Inter Frame Encoding (o programa contém para este efeito uma segunda forma de inicialização em que é passada a classe Python com o ficheiro descodificado, em vez do path para o ficheiro).

Está ainda presente um programa de teste da classe, ‘video_player_test.py’.

Video Coding – Intra-Frame Encoder

Na classe **IntraCodec.py** foi implementada uma versão de um codificador intra-frame sem perdas, analisando os pixéis adjacentes de cada pixel, escolhendo o melhor e codificando a sua diferença.

O funcionamento da classe é o seguinte, o vídeo é lido do ficheiro original e armazenado nas estruturas do programa. O vídeo é então codificado, começando pela escrita do header, que contém as informações originais do ficheiro como width e height, e outros parâmetros necessários à descodificação com o parâmetro de inicialização do Golomb. Cada pixel de cada frame é então iterado, sendo decidido o pixel adjacente ‘predicted’ consoante a implementação da fórmula do algoritmo do preditor *JPEG-LS non-linear,* sendo codificado a diferença entre estes dois pixéis,

através do Golomb e da Bitstream. Todos os valores são precedidos por um bit 0 ou 1 indicativo se o valor é positivo ou negativo.

Na descodificação está presente o processo inverso, desde a leitura e descodificação do header, até à leitura e descodificação dos pixéis através do Golomb, Bitstream e reconstrução dos mesmos através da soma da diferença codificada com o pixel que foi predicted.

A classe foi utilizada com sucesso para codificar e descodificar os três formatos de vídeo (444,422,420) reduzindo significativamente o espaço ocupado pelos mesmos, e verificada a integridade dos valores dos pixéis antes e depois da conversão.

Está ainda presente um programa de teste da classe, 'intra_codec_test.py'.

Video Coding – Hybrid-Frame Encoder

Na classe **HybridCodec.py** está presente a nossa implementação de um codificador Inter-Frame híbrido, fazendo sempre codificação Intra-Frame na primeira frame do vídeo, e Inter-Frame nas restantes.

O programa começa por ler o vídeo original do ficheiro indicado. A seguir começa a codificar o vídeo. É acrescentado ao header a informação do tamanho dos blocos (quadrados) e da área de pesquisa de blocos. A primeira frame é codificada de forma similar à indicada no módulo anterior. Em todas as restantes frames, são criados arrays (matrizes) relativos às posições e conteúdos dos respetivos blocos de pixéis em cada frame. Para cada bloco da frame presente é encontrado o bloco da frame anterior mais similar a este, sendo que esse bloco tem que estar dentro da range definida em termos de posição (argumento search_area). O bloco mais similar é aquele cuja soma dos valores absolutos das diferenças entre pixéis dos dois blocos, é menor. O processo é agora similar ao do módulo anterior, é codificado em primeiro lugar o vetor respetivo à posição relativa na frame do bloco mais similar, seguido de todos os pixéis desse bloco de diferenças. Do lado do descodificador, os vetores e blocos de diferenças de pixéis são descodificados, e reconstruídas as frames.

A classe foi utilizada com sucesso para codificar e descodificar os três formatos de vídeo (444,422,420) reduzindo significativamente o espaço ocupado pelos mesmos, e verificada a integridade dos valores dos pixéis antes e depois da conversão.

Está ainda presente um programa de teste da classe, 'hybrid_codec_test.py'.

Video Coding – Lossy Coding

A funcionalidade de **Lossy Coding** nas duas vertentes de codificadores (Intra e Inter) foi implementada através da quantização dos valores das componentes de cor do vídeo (Y U V).

São passados três argumentos na inicialização do programa, referentes ao fator de quantização de cada componente. Os valores são assim divididos por esse fator durante o encoding com a Bitstream, e multiplicados pelo mesmo fator no decoding. Devido ao floor() utilizado na divisão durante o encoding, ocorre perda da informação do vídeo, sendo esta maior quanto maior for o fator de quantização.

Para tentar corrigir a propagação do erro, cada pixel codificado é atualizado durante a codificação, passando a ter o valor desse mesmo pixel, somado com a diferença com que está a ser codificado. Contudo, isto piorava significativamente a qualidade da imagem, e não foi encontrada uma solução para resolver esta problema da forma como seria esperado.

Sendo assim, a versão que está implementada, aplica a quantização das componentes, deixando a propagação do erro.

Está presente um programa de teste desta funcionalidade, na vertente de Intra-Frame Encoding, no programa, 'lossy_intra_coding_test.py'. Não foi feito um programa de teste de lossy coding para o codificador híbrido, dado que a funcionalidade de lossy coding não foi implementada com sucesso.

Nota final:

É de notar que todos estes programas apresentam um tempo de execução considerável para grande número de frames, podendo este número ser customizado nos argumentos de execução dos programas. No desenvolvimento do projeto foram usados valores pequenos de 1 a 3 frames, sendo no entanto óbvia a correta execução dos programas para qualquer número de frames.

Execução dos programas de teste

bitstream_test.py- Este programa, de teste unitário da Bitstream, escreve uma sequência de bits num ficheiro, e ainda um valor numérico com N número de bits (caso seja possível). Posteriormente lê os mesmos valores e apresenta-os no terminal.

Corrido sem argumentos (python3 bitstream_test.py) utiliza valores default, a sequência 0010101, e o número 49 com 6 bits.

Se corrido com 3 argumentos extra (python3 bitstream_test.py 10010 37 8) irá escrever a sequência 10010 de bits, e escrever o número 37 com 8 bits.

python3 bitstream_test.py <sequenciaDe1'sE0's> <valorNumerico> <NumeroBits>

---

golomb_test.py- Este programa, de teste unitário do Golomb, codifica uma série de números presentes no ficheiro testFile.txt, e descodifica-os novamente, mostrando os mesmos no terminal.

Corrido sem argumentos (python3 golomb_test.py) utiliza um valor default para o argumento do Golomb, m=32.

Se corrido com 1 argumentos extra (python3 golomb_test.py 4) irá usar o valor 4 como parâmetro de inicialização do fator do Golomb.

python3 golomb_test.py <fatorMGolomb>

---

video_player_test.py- Este programa de teste do Video Player, converte e mostra no ecrã X frames do vídeo YUV.

Precisa necessariamente de um argumento, por exemplo, (python3 video_player_test.py 3) irá mostrar as primeiras 3 frames do vídeo. Podemos escrever 'all' em vez um valor númerico para converter e mostrar todas as frames do vídeo (embora não seja aconselhável devido ao elevado tempo de execução).

A seguir será pedido ao utilizador a escolha de um dos 3 vídeos presentes no nosso repositório.

python3 video_player_test.py <NumeroDeFramesLimite>

---

intra_codec_test.py- Este programa de teste do Intra Codec, codifica, descodifica e mostra ainda no ecrã X frames do vídeo.

Necessita de 2 argumentos, por exemplo, (python3 intra_codec_test.py 3 4) irá codificar/descodificar/mostrar no ecrã 3 frames do vídeo, usando o parâmetro de Golomb M=4.

A seguir será pedido ao utilizador a escolha de um dos 3 vídeos presentes no nosso repositório.

python3 intra_codec_test.py <NumeroDeFramesLimite> <ParametroGolomb>

---

hybrid_codec_test.py- Este programa de teste do Hybrid Codec, codifica, descodifica e mostra ainda no ecrã X frames do vídeo.

Necessita de 4 argumentos, por exemplo, (python3 intra_codec_test.py 3 4 8 1) irá codificar/descodificar/mostrar no ecrã 3 frames do vídeo, usando o parâmetro de Golomb M=4, com tamanho dos blocos de pixéis=8 e área de pesquisa do melhor bloco=1.

A seguir será pedido ao utilizador a escolha de um dos 3 vídeos presentes no nosso repositório.

python3 hybrid_codec_test.py <NumeroDeFramesLimite> <ParametroGolomb> <blockLength> <searchArea>

---

lossy_intra_coding_test.py- Este programa de teste da funcionalidade de Lossy Coding codifica, descodifica e mostra ainda no ecrã X frames do vídeo.

Necessita de 5 argumentos, por exemplo, (python3 lossy_intra_coding_test.py 3 4 0 2 2) irá codificar/descodificar/mostrar no ecrã 3 frames do vídeo, usando o parâmetro de Golomb M=4. Irá usar como parâmetros de quantização para as componentes YUV, respetivamente os valores 0, 2 e 2 (0 → não há quantização).

A seguir será pedido ao utilizador a escolha de um dos 3 vídeos presentes no nosso repositório.

python3 lossy_intra_coding_test.py <NumeroDeFramesLimite> <ParametroGolomb> <quantizationStep1> <quantizationStep2> <quantizationStep3>

Link para o repositório github

https://github.com/Joao-Nogueira-gh/video-compressin

Todo o código desenvolvido encontra-se no repositório do link acima mencionado (pasta src), juntamente com a devida documentação gerada pelo Doxygen (pasta docs), os vídeos exemplo e ficheiros utilizados (pasta res), e ainda um ficheiro 'requirements.txt' com as bibliotecas python e respetivas versões usadas no desenvolvimento do projeto.

É de notar que se não for usada a versão indicada do numpy (1.15.4), funções como a de lossy coding **não irão funcionar**, e o programa irá dar erro.