Centro de Competências de Ciências Exatas e da Engenharia

Compressão de imagem e vídeo com controlador ARM

Projeto para obtenção do grau de Mestre em Engenharia de Telecomunicações e Redes de Energia

Orientação:

Doutor João Dionísio Simões de Barros

Eng. Luís Miguel Vieira Fernandes de Aguiar

Linda Natércia Calaça Sousa

fevereiro de 2015

Resumo

Este trabalho foi desenvolvido num estágio na empresa ABS GmbH sucursal em Portugal, e teve como foco a compressão de imagem e vídeo com os padrões JPEG e H.264, respetivamente. Foi utilizada a plataforma LeopardBoard DM368, com um controlador ARM9.

A análise do desempenho de compressão de ambos os padrões foi realizada através de programas em linguagem C, para execução no processador DM368. O programa para compressão de imagem recebe como parâmetros de entrada o nome e a resolução da imagem a comprimir, e comprime-a com 10 níveis de quantização diferentes. Os resultados mostram que é possível obter uma velocidade de compressão até 73 fps (*frames per second*) para a resolução 1280x720, e que imagens de boa qualidade podem ser obtidas com rácios de compressão até cerca de 22:1.

No programa para compressão de vídeo, o codificador está configurado de acordo com as recomendações para as seguintes aplicações: videoconferência, videovigilância, armazenamento e *broadcasting*/*streaming*. As configurações em cada processo de codificação, o nome do ficheiro, o número de *frames* e a resolução do mesmo representam os parâmetros de entrada. Para a resolução 1280x720, foram obtidas velocidades de compressão até cerca de 68 fps, enquanto para a resolução 1920x1088 esse valor foi cerca de 30 fps.

Foi ainda desenvolvida uma aplicação com capacidades para capturar imagens ou vídeos, aplicar processamento de imagem, compressão, armazenamento e transmissão para uma saída DVI (*Digital Visual Interface*). O processamento de imagem em *software* permite melhorar dinamicamente as imagens, e a taxa média de captura, compressão e armazenamento é cerca de 5 fps para a resolução 1280x720, adequando-se à captura de imagens individuais. Sem processamento em *software*, a taxa sobe para cerca de 23 fps para a resolução 1280x720, sendo cerca de 28 fps para a resolução 1280x1088, o que é favorável à captura de vídeo.

Palavras-chave: Compressão de vídeo, compressão de imagem, processamento de imagem, controlador ARM, JPEG, H.264.

Abstract

This work was developed under an internship at the company ABS GmbH sucursal em Portugal, and it is focused in image and video compression with JPEG and H.264 standards, respectively. The platform used for that was LeopardBoard DM368, with an ARM9 controller.

The compression performance analysis, to both JPEG and H.264 standards, was made through programs developed in C language, running at DM368 processor. The program referring to image compression has as inputs the image name to compress and its resolution and outputs several versions of that image compressed with 10 different quantization levels. The results show that compression speed can reach 72 fps to 1280x720 resolution, and images with good quality can be obtained with compression ratios until 22:1.

In video compression program, the encoder is configured according to recommendations for the following applications: videoconferencing, surveillance, storage, and broadcasting/streaming. The configurations to each coding process, the video name and resolution, and the frames quantity are the inputs. For 1280x720 resolution it was measured the frame rates until 68 fps, and for 1920x1088 resolution it was measured frame rates until 30 fps.

It was also developed an application to capture images and videos, apply image processing, compress it and store it or transmit it to a DVI output. When image processing is applied by software, it is possible to dynamically enhance the images, and the frame rate related to 1280x720 resolution is about 5 fps, which is suitable to capture still images. Without image processing by software, frame rate increases until 23 fps to 1280x720 resolution, and it is about 28 fps to 1920x1088 resolution, which is suitable to video capture.

Keywords: Video compression, image compression, image enhancement, ARM controller, JPEG, H.264.

Agradecimentos

Com a conclusão deste trabalho, concretizo um dos grandes objetivos da minha vida, o qual fiz com grande dedicação. Tenho a agradecer, acima de tudo, aos meus queridos pais que sempre me apoiaram e incentivaram a seguir este caminho, e muito se esforçaram para me proporcionar tudo o que necessitei para alcançar este objetivo.

Tenho muito a agradecer também a um conjunto de outras pessoas:

À empresa ABS GmbH sucursal em Portugal por proporcionar este estágio, e disponibilizar os equipamentos necessários à realização do trabalho, bem como apoio técnico. Agradeço especialmente ao Eng. Luís Aguiar, o meu coorientador, pelos conhecimentos transmitidos, pelo apoio, paciência e orientação em todos os momentos ao longo da realização desta tese.

Ao meu orientador, professor Dionísio Barros, pela disponibilidade apresentada sempre que precisei de orientação, apoio e motivação para prosseguir com este trabalho.

Ao professor Sílvio Velosa, pela ajuda no tratamento de dados estatisticos.

Ao meu namorado, Diogo, pela paciência e apoio prestados ao longo deste último ano.

Aos meus amigos, pela força, companheirismo, amizade e momentos de descontração que me proporcionam. Agradeço em especial ao Pedro, à Tatiana, à Ana, à Lisandra e ao Everaldo pela constante atenção que me dão, pela preocupação, pelos conselhos, por todo o apoio dado, cada um à sua maneira.

Aos meus colegas, nomeadamente aos que acompanharam durante o mestrado, pelo companheirismo e pela ajuda mútua nas horas de estudo e na realização de trabalhos, que dessa forma também contribuíram para que alcançasse esta fase.

Índice

Lista de acrónimos

API - *Application Programming Interface*

ASCII - *American Standard Code for Information Interchange*

ASO - *Arbitrary Slice Order*

AVC - *Advanced Video Coding*

AWB - *Auto White Balance*

bS - boundary strength

CABAC - *Context Adaptive Binary Arithmetic Coding*

CAE - *Context Based Arithmetic Encoding*

CAVLC - *Context Adaptive Variable length Coding*

CBR - *Constant Bit Rate*

CCD - *Charge-Coupled Device*

CM - Compensação de Movimento

CMYK - *Cyan, Magenta, Yellow, Black Key*

CPU - *Central Processing Unit*

DCT - *Discrete Cosine Transform*

DDR2 - *Double Data Rate*

DFT - *Discrete Fourier Transform*

DLCD - *Digital LCD Controller*

dpi - dots per inch

DRAM - *Dynamic Random Access Memory*

DSP - *Digital Signal Processor*

DV - *Digital Video*

DVI - Digital Visual Interface

DVR - *Digital Video Recorder*

DWT - *Discrete Wavelet Transform*

EDMA - *Enhanced Direct Memory Access*

EM - Estimação de movimento

EQM - Erro Quadrático Médio

EZW - *Embedded Zerotree Wavelet*

FDCT - *Forward* DCT

FOURCC - *Four Character Code*

FPGA - *Field Programmable Gate Array*

fps - *frames per second*

FQ - Fator de Qualidade

GIF - *Graphic Interchange Format*

HD - *High Definition*

HDTV - *High Definition Television*

HDVICP - *High Definition Video and Imaging Co-Processor*

HEVC - *High-Efficiency Video Coding*

HSI - *Hue, Saturation, Intensity*

HSV - *Hue, Saturation, Value*

IPIPE - *Image Pipe*

IPIPEIF - *IPIPE Interface*

ISIF - *Image Sensor Interface*

ISO - *International Standardization Organization*

ITU - *International Telecommunication Union*

JBIG - *Joint Bi-level image Experts Group*

JCT-VC - *Joint Colaborative Team on Video Coding*

JPEG - *Joint Photographic Experts Group*

KLT - *Karhunen-Loeve Transform*

LCD - *Liquid Crystal Display*

MF – *Multiplication Factor*

MJCP - *MPEG JPEG Co-Processor*

MJPEG - *Motion* JPEG

MOS - *Mean Opinion Score*

MP - Mega Píxel

MPEG - *Motion Pictures Experts Group*

MSE - *Mean Square Error*

MVD - *Motion Vector Difference*

NAL - *Network Abstraction Layer*

NTSC - *National Television System Committee*

OSD - *On Screen Display*

PAL - *Phase Alternation Line*

PC - *Personal Computer*

PDF - *Portable Document Format*

PQ - Parâmetro de Quantização

PSNR - *Peak Signal-to-Noise Ratio*

RGB - *Red, Green, Blue*

SD - *Secure Digital*

SDK - *Software Development Kit*

SDTV - *Standard Definition Television*

SECAM - *Sequentiel Couleur Avec Mémoire*

SoC - System on Chip

SRAM - *Static Random Access Memory*

STB - *Set-up Box*

TI - *Texas Instruments*

TIFF - *Tagged Image File Format*

UART - *Universal Asynchronous Receiver/Transmitter*

UHD - *Ultra High Definition*

VBR - *Variable Bit Rate*

VBV - *Video Buffering Verifier*

VCL - *Video Coding Layer*

VENC - *Video Encoder*

VLSI - *Very Large Scale Integration*

VM - Vetor de Movimento

VPBE - *Video Processing Back End*

VPFE - *Video Processing Front End*

VPSS - *Video Processing Subsystem*

WHT - *Walsh-Hadarmd Transform*

xDAIS - *eXpress DSP Algorithm Interoperability Standard*

xDM - *xDAIS for Digital Media*

Índice de figuras

Índice de tabelas

1. Introdução

Um vídeo é constituído por uma sequência de múltiplas imagens, sendo cada uma delas designada por *frame* [1]. Por sua vez, as imagens capturadas por um sensor apresentam-se num formato designado por *raw* [2]. Neste formato, os dados são representados por um elevado número de *bits.* Tal apresenta-se como um problema quando se pretende transmitir estes dados através de canais de comunicação de largura de banda limitada, como também para as aplicações de armazenamento e processamento de dados. Neste sentido as técnicas de compressão apresentam-se como um fator de grande importância, uma vez que permitem a compactação dos dados [2].

A aplicação de técnicas de compressão de sinais de vídeo tem contribuído para o aumento da quantidade de serviços de comunicação de dados multimédia através da rede de telecomunicações e de acesso de dados através da internet, onde os dados de imagem ocupam uma porção significativa [3]. Nos últimos anos, os requisitos crescentes por elevada qualidade de imagem, de realidade virtual e de jogos envolventes têm conduzido ao aperfeiçoamento das técnicas de compressão no sentido de se adequarem a sequências de elevada definição e a rácios de compressão elevados, como também para poder suportar sistemas 3D e multivistas [4].

Neste trabalho, foi implementado um sistema de codificação de imagem e vídeo para execução no processador DM368, que é um SoC (*System on Chip*) que inclui um controlador ARM. Este sistema permite capturar, aplicar processamento de imagem, e comprimir imagens ou vídeos para posterior armazenamento ou transmissão para saída DVI (*Digital Visual Interface*).

1.1 Motivação

Este trabalho foi desenvolvido no âmbito de um estágio na empresa ABS GmbH. Esta empresa desenvolve, produz e vende sistemas de câmara digital personalizados. Alguns destes sistemas, que usam sensores de imagem, geram sinais de vídeo que dificultam a transferência de dados na maior parte dos protocolos de comunicação industriais. Neste sentido, o desenvolvimento de um sistema de compressão em tempo real é de elevada importância para a empresa, uma vez que permite a redução da quantidade de dados, possibilitando a transmissão em tempo real pelos sistemas industriais.

Atualmente já existem *frameworks* que dispõem de *pipelines* a partir das quais é possível fazer a compressão de imagem e vídeo em SoCs, e que permitem até a configuração dos parâmetros de codificação. Contudo, o desenvolvimento de uma aplicação própria é mais interessante na medida em que proporciona liberdade para explorar as capacidades do processador e da plataforma que o integra. Para além disso, permite personalizar a aplicação conforme os requisitos do sistema, e proporciona flexibilidade de adaptação da mesma a diferentes contextos de utilização. Tal proporciona, ainda, potencial para constante aperfeiçoamento conforme o desenvolvimento tecnológico e as necessidades do mercado.

A utilização de controladores ARM, da família DaVinci da Texas Instruments, foi uma opção feita pela empresa. Estes vêm com *hardware* de aceleração para compressão de imagem e vídeo, suportando padrões como H.264, MPEG4, MPEG2, MJPEG e JPEG. Os padrões de interesse para a empresa são o JPEG e o H.264.

1.2 Objetivos

Neste trabalho de estágio foram definidos quatro objetivos. Esses objetivos são:

- Analisar e comparar padrões de compressão de imagem e vídeo, mais especificamente o JPEG e o H.264;
- Implementar um sistema de câmara usando o controlador ARM da Texas Instruments com compressão de vídeo;
- Fazer processamento de imagem (baseado no histograma ou outros) antes da compressão da imagem;
- Testar o desempenho da compressão de imagem com o padrão JPEG, e da compressão de vídeo com o padrão H.264, usando o controlador ARM.

Os procedimentos tomados com foco no alcance destes objetivos começaram pela aquisição dos conceitos acerca de imagem digital, bem como das técnicas de processamento das mesmas. Seguidamente procurou-se conhecer e compreender as técnicas de compressão de imagem e vídeo.

Uma vez que estes objetivos visam a implementação num controlador ARM, procurou-se também compreender a arquitetura típica de um sistema para compressão de vídeo com base nestes controladores. Foi necessário adquirir conhecimento acerca do desenvolvimento de aplicações usando o sistema operativo Linux embebido, o qual está associado à plataforma de desenvolvimento utilizada.

1.3 Estrutura do trabalho

Este trabalho está organizado em 6 capítulos. O presente capítulo, Introdução, faz uma introdução ao tema do trabalho, apresenta as motivações e os objetivos tidos em conta na elaboração do trabalho, e indica qual é a estrutura do documento.

No segundo capítulo, Fundamentação teórica, apresenta-se os fundamentos teóricos, tidos como base para os procedimentos tomados no sentido de se atingirem os objetivos propostos. Este capítulo encontra-se dividido em três seções, sendo a primeira dedicada a noções de imagem digital e de processamento da mesma, a segunda dedicada à compressão de imagem e vídeo, e a terceira é dedicada à apresentação da arquitetura típica de sistemas de compressão de vídeo, bem como à apresentação de algumas implementações comerciais atuais.

No capítulo seguinte, Compressão de imagem e vídeo num controlador ARM, primeiramente é feita uma apresentação à plataforma utilizada no desenvolvimento deste trabalho, bem como aos restantes recursos materiais, e aos codecs integrados nas aplicações desenvolvidas. Para além disso, são apresentados os procedimentos realizados no desenvolvimento das aplicações para análise dos padrões de compressão, e os algoritmos implementados para processamento de imagem antes da compressão.

Os resultados obtidos das análises realizadas através das aplicações apresentadas no capítulo três são analisados e discutidos no quarto capítulo, Avaliação do desempenho da compressão de imagem e vídeo num controlador ARM.

No quinto capítulo é descrito o sistema de compressão de imagem e vídeo realizado, bem como é realizada uma análise ao desempenho do mesmo.

O sexto e último capítulo, Conclusões e trabalhos futuros, são apresentadas as conclusões do trabalho, bem como alguns aspetos a ter conta em trabalhos futuros.

2. Fundamentação teórica

Este capítulo é dedicado à revisão do estado da arte das tecnologias de compressão de imagem e vídeo em processadores ARM.

Como um dos objetivos deste trabalho refere-se ao melhoramento da imagem antes da compressão, é fundamental o conhecimento das noções acerca de imagem digital como também das técnicas de processamento da mesma. Seguidamente são apresentados alguns métodos de compressão de imagem e vídeo, bem como alguns padrões.

Por fim, apresenta-se uma revisão do estado da arte dos sistemas de compressão de imagem e vídeo.

2.1. Imagem digital

Uma imagem pode ser definida como uma função de duas variáveis, $f(x,y)$, onde x e y são as coordenadas do plano espacial. Num determinado ponto de coordenadas (x,y), no caso de se pretender uma imagem em escala de cinza, $f(x,y)$ é uma quantidade escalar positiva que representa a amplitude da intensidade ou nível de cinza nesse ponto. Já se se pretender uma imagem colorida, $f(x,y)$ representa um vetor de três elementos [5], cujos valores representam componentes diferentes do espaço de cor utilizado (ver secção 2.1.3).

Assim sendo, quando uma imagem é gerada a partir de um processo físico, os valores de f são proporcionais à energia radiada pela fonte física, de modo que devem ser finitos e diferentes de zero, ou seja [6],

$$0 < f(x,y) < \infty. \tag{2.1}$$

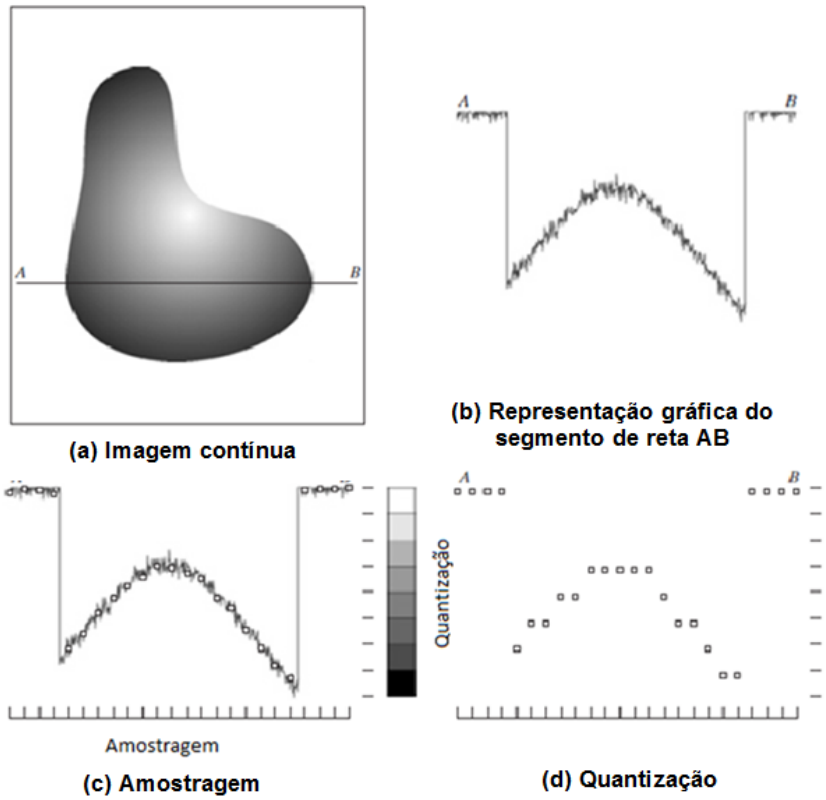
Esta função é caraterizada por duas componentes: a iluminação, que é a quantidade de iluminação da fonte incidente na cena visualizada; e a refletância, que é a quantidade de iluminação refletida pelos objetos da cena [6].

Esta é uma função contínua, tanto relativamente às coordenadas espaciais x e y, quanto à amplitude f. Para se obter a imagem digital é necessário, portanto, converter tais dados contínuos para a forma digital, o que é realizado essencialmente a partir de dois processos: amostragem e quantização. A amostragem refere-se à discretização dos valores das coordenadas espaciais, enquanto a quantização refere-se à discretização dos valores das amplitudes [6]. Tais processos encontram-se ilustrados de uma forma geral na Figura 2.1.

Tendo-se uma imagem contínua, cujas cores se encontrem na escala de cinza, como a da Figura 2.1 (a), pode ser representada por uma função unidimensional como a da Figura 2.1 (b), que é um gráfico com os valores do nível de cinza da imagem contínua ao longo do segmento de linha AB da Figura 2.1 (a). A amostragem desta função é realizada tirando amostras igualmente espaçadas ao longo da linha AB, como se pode ver na Figura 2.1 (c). A função amostrada é representada pelo conjunto das localizações discretas das amostras, as quais encontram-se identificadas pelas marcas verticais na parte inferior da figura.

Quanto à localização vertical de cada uma das amostras, representadas na Figura 2.1 (c) pelos pequenos quadrados brancos, refere-se ao respetivo nível de cinza. Como se pode observar, no lado direito desta figura é mostrada a escala de níveis de cinza dividida em oito níveis discretos, definidos pelas marcas horizontais, numa gama desde o preto até ao branco. A quantização é realizada atribuindo um dos oito níveis de cinza a cada amostra, em função da proximidade vertical da mesma à marca horizontal.

A Figura 2.1 (d) mostra as amostras digitais resultantes dos processos de amostragem e quantização, de uma linha da imagem. Se se repetirem estes processos para as restantes linhas da imagem é produzida uma imagem digital bidimensional, como a que se apresenta na Figura 2.2.



(a) Imagem contínua

(b) Representação gráfica do segmento de reta AB

(c) Amostragem

(d) Quantização

Figura 2.1. Amostragem e quantização de uma imagem.

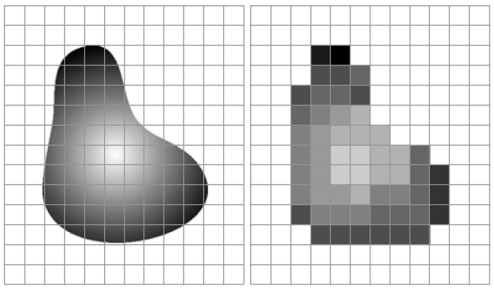


Figura 2.2. (a) Imagem contínua projetada num sensor; (b) Resultado dos processos de amostragem e quantização [6].

As amostras que compõem uma imagem digital, cada uma com uma localização e um valor específicos, são denominados por píxeis (*PICture ELements*) ou *pels.* Píxel é o termo mais utilizado para denotar os elementos de uma imagem digital [6].

Quanto aos valores das coordenadas espaciais dos *píxeis*, apenas é requerido que sejam inteiros positivos. Já o número de níveis de intensidade, L, é determinado pelo número de *bits* com que são codificados tais níveis de intensidade. Devido às caraterísticas do *hardware* relativamente ao processamento, armazenamento e amostragem, L é tipicamente uma potência inteira de 2 [6], ou seja

$$L = 2^k. \tag{2.2}$$

É comum definir-se uma imagem que possa ter L níveis de intensidade como "imagem de *k-bit*" ou "imagem com *k-bits* de profundidade". Por exemplo, uma imagem com 256 possíveis valores para o nível de intensidade é chamada imagem de 8-*bit*. Um caso especial é o das imagens binárias (*bi-level*), ou imagens de 1-*bit,* cujos píxeis são codificados com apenas 1 *bit,* de modo que apenas podem ter dois valores possíveis: preto ou branco [7].

2.1.1 Resolução

O número de píxeis que constituem uma imagem com M colunas e N linhas de píxeis é também conhecido por resolução da imagem digital, o qual é apresentado na forma $M \times N$ [8]. Na Tabela 2.1 são apresentadas algumas resoluções muito utilizadas.

Tabela 2.1.**Resoluções de vídeo.**

Resolução	Número de píxeis (*M*x*N*)
CIF (*Common Intermediate Format*)	352x288
VGA (*Video Graphics Array*)	800x600
HDTV (*High-Definition Television*)	1280x720
FHD (*Full High-Definition*)	1920x1080
4K UHD (4000 píxeis*, Ultra High-Definition*)	3840x2160

Contudo, o verdadeiro significado do termo *resolução* é outro, e está relacionado com o processo de amostragem espacial [8]. A resolução espacial de uma imagem digital refere-se à proximidade entre os *píxeis* que constituem a imagem, ou seja, ao número de *píxeis* por unidade de área, sendo dada por *pontos por polegada* (em inglês: *dots per inch* – *dpi*) [9]. A resolução espacial permite assim estabelecer a relação entre o número de *píxeis* de uma imagem e a dimensão real da sua representação num suporte físico.

2.1.2 Imagem digital no domínio da frequência

A representação de uma imagem no domínio da frequência permite conhecer a taxa com que ocorrem alterações nos valores das intensidades das cores [10]. Normalmente não é possível fazer associações diretas entre componentes específicos de uma imagem e a sua transformada. Contudo, é possível tirarem-se algumas conclusões acerca das caraterísticas da imagem a partir da sua representação no domínio das frequências. Sabe-se, por exemplo, que as maiores frequências estão associadas a zonas onde ocorrem transições rápidas de cor, que as menores frequências ocorrem em zonas de sombras [11].

No caso de imagens constituídas por padrões regulares, cada linha e/ou coluna está associada a uma forma de onda com frequência fixa. Contudo, tal já não acontece para imagens com padrões irregulares ou sem qualquer padrão, nas quais cada linha e/ou coluna é representada por uma forma de onda complexa [11].

Segundo *Fourier*, estas formas de onda complexas podem ser decompostas em várias componentes de frequência. A transformada de *Fourier* permite separar cada uma dessas componentes, bem como a passagem da função no domínio dos tempos para o domínio das frequências. Embora imagens digitais sejam sinais discretos no domínio dos tempos, a sua passagem para o domínio das frequências torna-os contínuos. Assim sendo, para que possam tornar-se discretos também no domínio da frequência, e portanto poderem ser processados, devem ser amostrados neste domínio. Para tal, deve ser aplicado um caso particular da transformada de

Fourier, mais precisamente a transformada discreta de *Fourier* (DFT – *Discrete Fourier Transform*) [12]. A DFT direta de uma imagem com M linhas e N colunas definida por uma função de duas variáveis, $f(x,y)$, é dada por

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, \tag{2.3}$$

enquanto a respetiva DFT inversa é calculada a partir de

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \tag{2.4}$$

Note que no caso de $f(x,y)$ representar uma imagem, o valor da transformada na origem, ou seja $F(0,0)$, representa o valor médio do nível de brilho da imagem. Este é chamado por coeficiente DC, podendo fazer-se uma analogia com os sistemas elétricos, em que a componente DC pode ser comparada com a corrente direta [10].

A DFT gera, portanto, uma matriz de coeficientes que indicam as amplitudes dos componentes de frequência. Nesta representação, diversas operações de tratamento de sinal tornam-se mais simples de processar, do que na representação espacial ou temporal [10].

A respetiva transformada inversa é aplicada no domínio da frequência, de forma a se obter a imagem no domínio espacial.

2.1.2.1 Aliasing

O teorema de amostragem de *Nyquist* [10] diz que se uma função for amostrada a uma taxa igual ou superior a duas vezes a sua mais alta frequência, é possível recuperar completamente a função original a partir das suas amostras por período. Caso não seja respeitado o teorema de amostragem, a função é afetada por um fenómeno denominado por *aliasing*, em que são introduzidas componentes em frequência à função amostrada, e portanto, a função reconstruída a partir das amostras não será uma reprodução perfeita da original [6]. Quando tal acontece a funções relativas a imagens digitais, dá origem a artefactos que poderão ser visíveis na imagem no domínio espacial, tais como pequenas áreas de cores incorretas ou auras artificiais em torno dos objetos.

Quando é realizada amostragem a uma taxa reduzida relativamente àquela referida pelo teorema de amostragem de *Nyquist*, cada amostra é referente a uma área maior da imagem, a qual pode conter mais que uma cor. Como apenas uma cor

pode ser representada pela amostra, então é atribuída uma única cor a essa área, determinada com base numa média, de modo que o limite entre as cores envolvidas é perdido. Nas áreas da imagem em que tal acontece, pode surgir uma espécie de blocos de cor sólida (efeito conhecido por *blockiness,* ou blocagem), como se pode observar na Figura 2.3, e os limites ou contornos podem tornar-se irregulares (surgindo os conhecidos *jaggies*), como se pode observar na Figura 2.4 [11].

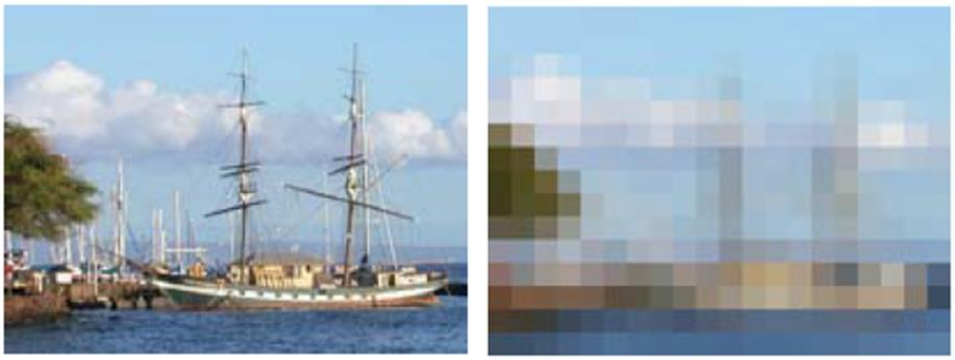


Figura 2.3. Efeito de blocagem: (a) imagem original; (b) imagem com blocagem, devido à reduzida taxa de amostragem [11].

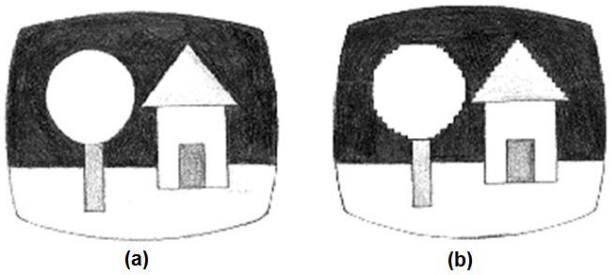


Figura 2.4. Efeito *jaggies*: (a) imagem original; (b) imagem com *jaggies* [13].

Um outro efeito que pode surgir nas imagens devido ao *aliasing* refere-se aos padrões de *Moiré.* Estes são bem visíveis em imagens com padrões complexos associados a elevadas frequências, e em imagens que contenham um padrão que apresente um determinado ângulo de inclinação relativamente ao ponto a partir do qual é realizada a amostragem [11]. Tal encontra-se ilustrado na Figura 2.5. Como se pode ver, a grelha mostra os blocos de amostragem, havendo blocos que contêm ambas as cores. Aqueles que possuem maioritariamente a cor branca tornam-se completamente brancos, caso contrário, tornam-se completamente pretos, resultando portanto, num padrão de *Moiré*, diferente do padrão original. Um exemplo real é apresentado na Figura 2.6.

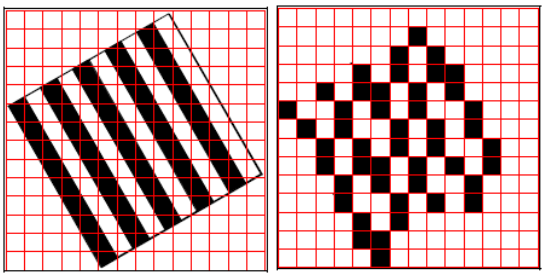
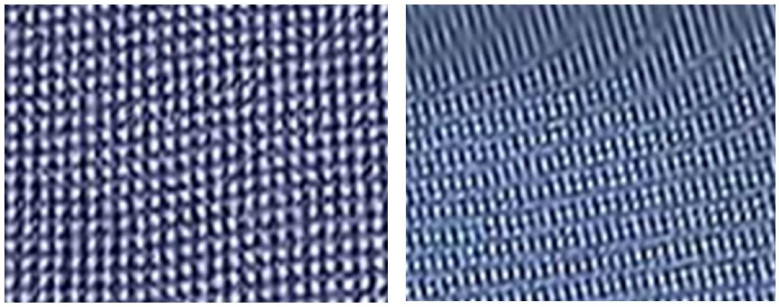


Figura 2.5. Esquema ilustrativo do processo a partir do qual surge o efeito de *Moiré* [11].



(a) **(b)**

Figura 2.6. (a) Padrão original da imagem, sem efeito de *Moiré*; (b) padrão com efeito de *Moiré* [11].

Os efeitos de *aliasing* podem surgir logo aquando da captura de uma imagem digital, bem como quando uma imagem digital resulta de uma digitalização a partir de um dispositivo como um *scanner*, pois ambos os processos envolvem uma operação de amostragem.

2.1.3 Espaços de cor

Um espaço de cor é uma representação matemática de um conjunto de cores. Este conceito surgiu devido à necessidade de criar padrões para a especificação de determinadas cores de entre um conjunto de cores detetáveis [10], [14].

Todos os espaços de cor podem ser derivados da informação RGB (*Red, Green, Blue*) fornecida pelos dispositivos de aquisição de imagens de digitais, como câmaras e *scanners* [14]. Os modelos de cores mais utilizados são o RGB (usado na computação gráfica), o YIQ (*In phase, Quadrature*), YUV e o YCbCr (usados em sistemas de vídeo), e o CMYK (*Cyan, Magenta, Yellow, Black Key*) (usado na impressão de cores). Nenhum destes espaços de cor está diretamente relacionado com as noções intuitivas de tonalidade, saturação e brilho, sendo estes parâmetros especificados por modelos como o HSV (*Hue, Saturation, Value*) e o HSI (*Hue, Saturation, Intensity*). Estes são mais utilizados para simplificar a programação, processamento e a manipulação por parte do utilizador final [14].

2.1.3.1 Espaço de cor RGB

O espaço de cor RGB, tal como indica o nome, tem como cores primárias o vermelho, o verde e o azul. Estas são mapeadas num sistema de coordenadas cartesiano tridimensional, resultando num cubo 3D como o representado na Figura 2.7.

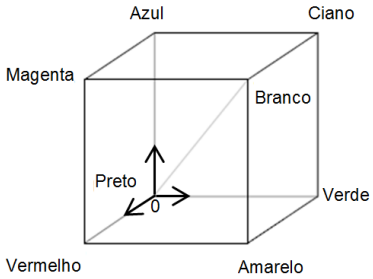


Figura 2.7. Espaço de cor RGB representado num sistema de coordenadas tridimensional.

Este espaço de cor é o mais utilizado nos gráficos de computador devido aos monitores usarem o vermelho, o verde e o azul para criar a cor desejada, simplificando portanto a arquitetura e o desenvolvimento do sistema. Para além disso, os sistemas desenvolvidos usando este espaço de cor podem ainda tirar vantagem de um largo número de rotinas de *software* já existentes, uma vez que este espaço de cor já é usado há vários anos [14].

Por outro lado, para a codificação de vídeo este não é o espaço de cor mais adequado, uma vez que existe uma alta correlação entre os três sinais de cor, resultando em uma certa redundância quando usada tal representação [15]. Para além disso, em diversas operações de processamento de imagem são realizadas alterações na componente luminosidade, tornando-se necessário calcular esta componente se os dados estiverem no espaço de cor RGB. Naturalmente que este passo pode ser descartado se o sistema tiver acesso a uma imagem guardada diretamente no formato de luminosidade e cor (tal como YUV).

Tais razões levaram a que muitos padrões de vídeo adotassem outros espaços de cor. Os mais comuns são o YUV, o YIQ e o YCbCr [14].

2.1.3.2 Espaços de cor YUV, YIQ e YCbCr

O espaço de cor YUV é usado pelos padrões de vídeo PAL (*Phase Alternation Line*), NTSC (*National Television System Committee*) e SECAM (*Sequentiel Couleur Avec Mémoire*) [14]. Este pertence a uma família de espaços de cor nos quais a informação da luminosidade é codificada separadamente da informação da cor. Tal como no RGB, neste são utilizadas três componentes para representar qualquer cor: Y, também chamado *luma*, representa o brilho ou

luminância; U e V, também chamados *croma* ou crominância, transportam a informação relativa à cor.

Esta separação da componente do brilho das componentes de cor representa uma vantagem, pois, como o sistema visual humano (HVS – *Human Visual System*) é mais sensível à intensidade da luz que às cores, é possível utilizar uma resolução menor na amostragem das componentes crominância sem sacrificar a qualidade subjetiva da imagem [15]. Tal processo é conhecido por subamostragem (em inglês: *subsampling*), e permite compactar a informação, e assim reduzir os requisitos de armazenamento e largura de banda de canal [1]. Duas das técnicas mais conhecidas são apresentadas adiante, na secção 2.1.3.3.

A componente Y é derivada de uma cor RGB fazendo uma média pesada dos componentes vermelho, verde e azul, enquanto as componentes U e V são derivados pela subtração do valor Y às componentes azul e vermelha. As equações utilizadas para fazer a conversão entre estes espaços de cor são [15], [16]

$$\begin{cases} Y = W_R R + W_G G + W_B B, \\ U = \dfrac{1}{2} \cdot \dfrac{B - Y}{1 - W_B}, \\ V = \dfrac{1}{2} \cdot \dfrac{R - Y}{1 - W_R}, \end{cases} \tag{2.5}$$

$$\begin{cases} R = Y + 2V(1 - W_R), \\ G = Y - U\dfrac{2W_B(1 - W_B)}{W_G} - V\dfrac{2W_R(1 - W_R)}{W_G}, \\ B = Y + 2U(1 - W_B). \end{cases} \tag{2.6}$$

Os coeficientes $W_R = 0{,}299, W_B = 0{,}114$ *e* $W_G = 0{,}587$ representam o peso que cada uma das componentes R, G e B tem no cálculo da luminosidade. A sua aplicação na equação (2.5) reflete o facto de que o olho humano é mais sensível a certos comprimentos de onda de luz do que a outros, o que afeta o brilho percebido de uma cor. A cor azul é a menos percetível, a verde aparece mais brilhante, e o vermelho aparece no intermédio [10]. Para dados com profundidade de 8 *bits,* R, G, B e Y variam de 0 a 255, enquanto U e V variam de -127 a 128.

O espaço de cor YCbCr é uma versão reduzida e deslocada do YUV, muito utilizada quando se lida com vídeo computorizado. Neste espaço de cor, tendo em conta uma profundidade de 8 *bits,* Y está definido para ter uma gama nominal de 16 a 235, enquanto Cb e Cr têm uma gama de 16 a 240 [14]. Segundo as recomendações ITU-R BT.601 [17], as equações de conversão entre dados RGB, na gama de 16 a 235, e YCbCr são

$$\begin{cases} Y = 0{,}299R + 0{,}587G + 0{,}114B, \\ C_b = -0{,}172R - 0{,}339G + 0{,}511B + 128, \\ C_r = 0{,}511R - 0{,}428G - 0{,}083B + 128, \end{cases} \quad (2.7)$$

$$\begin{cases} R = Y + 1{,}371(C_r - 128), \\ G = Y - 0{,}698(C_r - 128) - 0{,}336(C_b - 128), \\ B = Y + 1{,}732(C_b - 128). \end{cases} \quad (2.8)$$

O espaço de cor YIQ é opcionalmente usado pelo padrão de vídeo NTSC. As componentes I e Q significam, respetivamente, *in phase* e *quadrature* (quadratura, que é o método de modulação utilizado para transmitir a informação da cor), e são obtidos por uma rotação de 33º dos eixos U e V [14].

2.1.3.3 Técnicas de amostragem aplicadas ao YUV (YUV subsampling)

Estas técnicas são aplicadas para amostrar as componentes *luma* e *croma* de diferentes formas. No caso de serem amostradas com a mesma taxa, significa que não há subamostragem, e que cada píxel é composto por uma amostra de cada componente Y, U e V, segundo o posicionamento apresentado na Figura 2.8 (a). Esta técnica de amostragem é denotada por 4:4:4, sendo usados 24 *bits* por píxel no caso das imagens de 8-*bit* [15].

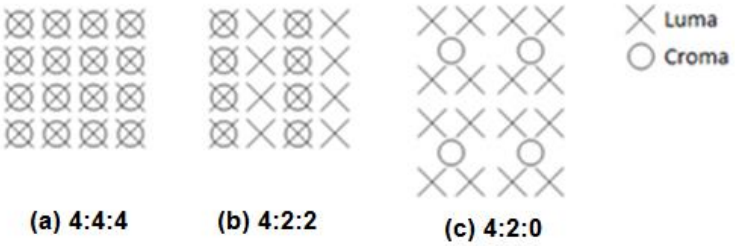


Figura 2.8. Diferentes técnicas de amostragem utilizadas pelos espaços de cor YUV.

No entanto, existem esquemas nos quais as componentes *croma* são amostradas a uma taxa menor que as componentes *luma*, o que acontece nas técnicas 4:2:2, 4:2:0 e 4:1:1. Na técnica 4:2:2 é feita uma amostragem horizontal 2:1, ou seja, por cada duas amostras Y horizontais, é utilizada uma amostra U e uma V (Figura 2.8 (b)). Quando a amostragem é feita também verticalmente, na mesma proporção 2:1, a técnica utilizada é a 4:2:0. Esta técnica apresenta algumas variantes ao que diz respeito à disposição dos diferentes componentes, estando uma delas representada na Figura 2.8 (c). Uma técnica menos comum é a 4:1:1, na qual a amostragem é feita apenas horizontalmente com uma razão de 4:1, ou seja, existem quatro amostras Y para cada U e V [1].

Quanto à organização das componentes de luminância e crominância num ficheiro, pode ser realizado de acordo com diferentes formatos. Cada um destes

formatos está associado a um código FOURCC (*Four Character Code*), que é um inteiro positivo de *32-bit* resultante da concatenação de quatro carateres ASCII (*American Standard Code for Information Interchange*) [18].

A Figura 2.9 ilustra a forma como os componentes são organizados em dois formatos largamente utilizados.

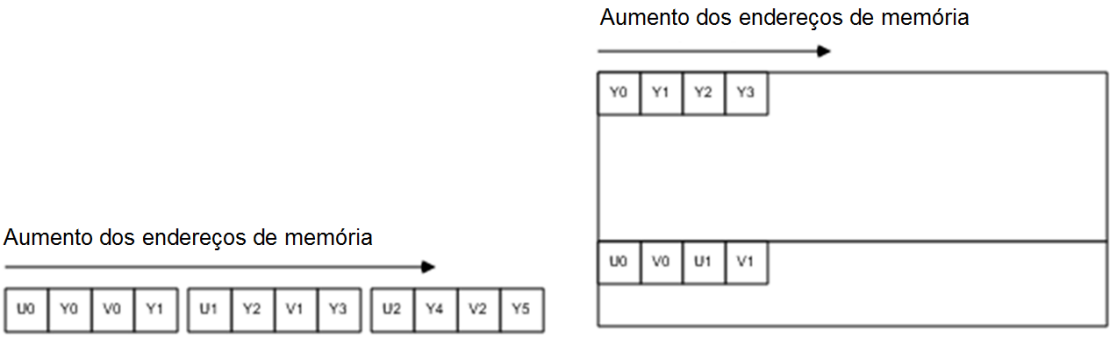


Figura 2.9. Formatos YUV de 8 *bits*: (a) UYUV; (b) NV12.

O formato na Figura 2.9 (a) é adequado à subamostragem 4:2:2, pois faz uso de 16 *bits* por píxel e tem como código FOURCC UYUV. Já o formato na Figura 2.9 (b) é adequado à subamostragem 4:2:0, que faz uso de 12 *bits* por píxel e tem como código FOURCC NV12.

2.1.4 Histogramas

O histograma apresenta-se como uma ferramenta importante quando se lida com imagens digitais. A partir dele é possível conhecerem-se diversas caraterísticas da imagem que representa, assim como também é possível alterar uma imagem a partir da manipulação do histograma [3].

Existem diversos tipos de histogramas, uns mais adequados para análise de determinadas caraterísticas do que outros, no entanto o princípio pelo qual são obtidos é sempre o mesmo. Os mais comuns apresentam a frequência relativa de ocorrência dos vários níveis de brilho, ou vários níveis de luminância, na imagem. Para uma imagem digital cuja luminância varie no intervalo de [0, *L*-1], o histograma é uma função discreta dada por

$$p(r_k) = \frac{n_k}{N}, \tag{2.9}$$

onde r_ké o k-ésimo nível de luminância, n_k é o número de *píxeis* na imagem com aquele nível de luminância, N é o número total de *píxeis* na imagem, e k toma valores inteiros desde 0 até *L*-1 [3].

2.1.4.1 Gama dinâmica (Dynamic range)

A gama dinâmica de uma imagem é entendida como a quantidade de níveis de brilho que compõem uma imagem [7]. É importante que a gama dinâmica de uma imagem seja elevada, pois permite uma menor degradação na qualidade da imagem durante o processamento da imagem e compressão. No caso ideal, a gama dinâmica abrange todos os níveis de brilho desde o valor mínimo encontrado na imagem até o valor máximo encontrado na imagem.

2.1.4.2 Contraste

O contraste de uma imagem é facilmente lido diretamente a partir do histograma, pois é definido pela gama dinâmica e pela diferença entre os níveis de brilho máximo e mínimo encontrados na imagem [7]. Uma imagem com muito contraste faz uso efetivo de todos os possíveis níveis de brilho do histograma, tornando-o largo. Já numa imagem com pouco contraste, o histograma é estreito, pois um grande número de píxeis faz uso de apenas uma pequena porção de níveis de brilho, e como a diferença entre esses níveis é muito reduzida, resulta num efeito tipo nevoeiro.

Assim sendo, o contraste de uma imagem pode ser alterado esticando ou encolhendo o seu histograma, ou seja, atribuindo a cada píxel um novo valor de brilho de modo a alterar a diferença entre os níveis de brilho máximo e mínimo encontrados na imagem. Uma técnica usada para alterar o contraste, conhecida por "contraste automático", consiste em aplicar a equação

$$p_n = (p - p_{min}) \times \frac{g_n}{g} \tag{2.10}$$

ao valor do brilho, p, de cada píxel [7], [19]. Nesta equação, ilustrada na Figura 2.10 (a), p_{min} é o valor de brilho mínimo da imagem, p_n é o novo valor do brilho do píxel, g_n é a gama de níveis de brilho para o novo histograma, e g a gama de níveis de brilho do histograma atual (entenda-se por gama de níveis de brilho como a diferença entre os valores máximo e mínimo de brilho). Uma outra técnica conhecida é a utilizada no *Photoshop* [8], a qual é dada pela equação

$$p_n = (p - 128) \times \frac{g_n}{g} + 128. \tag{2.11}$$

Neste método apenas os píxeis cujo brilho é de 128 não sofrem alterações, como se pode ver na Figura 2.10 (b). Uma variação deste método consiste em alterar o valor 128 para o valor do brilho dos píxeis que não se pretendem alterar.

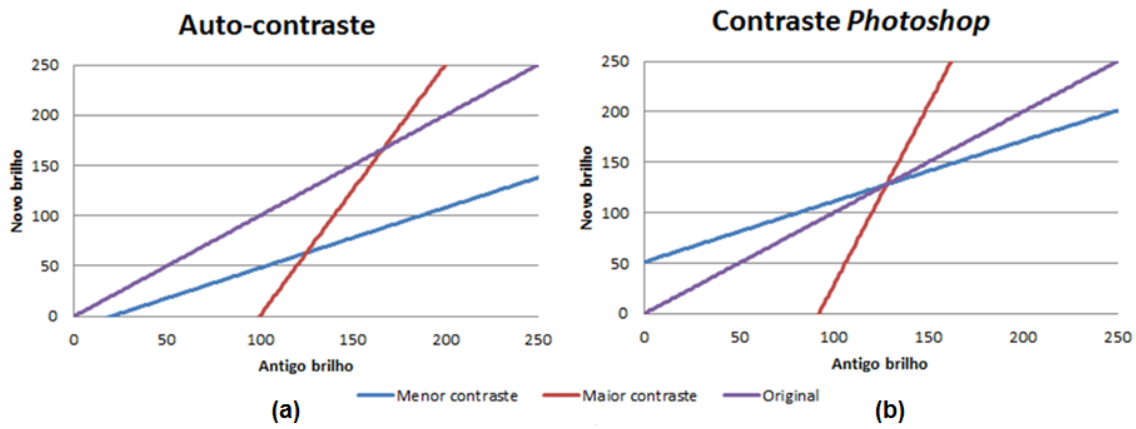


Figura 2.10. Controlo do contraste pelos métodos (a) "auto-contraste" e (b) contraste *Photoshop*.

As imagens da Figura 2.11 apresentam a forma como uma imagem é afetada pela alteração do contraste.

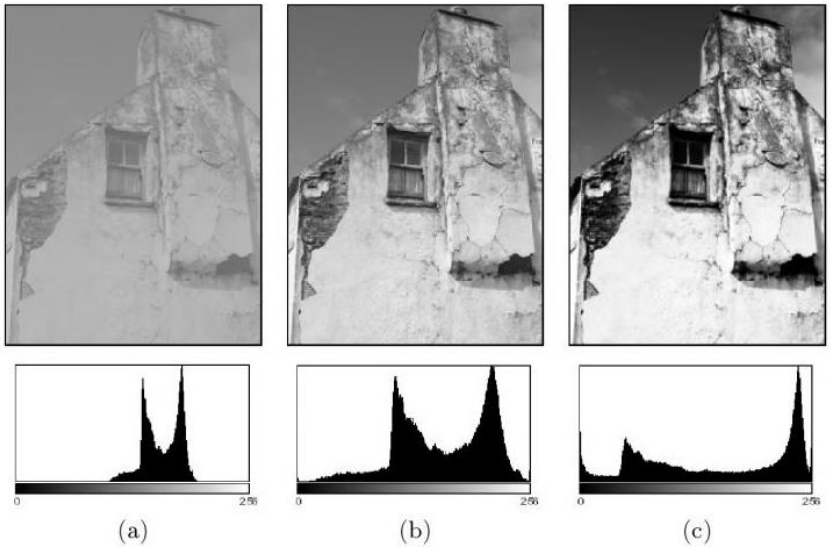


Figura 2.11. Imagem com diferentes contrastes: a) baixo contraste; b) contraste normal; c) elevado contraste [7].

Como se pode observar, num histograma estreito os níveis de luminância não diferem muito uns dos outros, dando origem a uma imagem com um certo efeito de nevoeiro. Já uma imagem com um histograma largo possui os níveis de luminosidade mais dispersos por toda a gama possível, contendo zonas muito claras em oposição a zonas muito escuras.

2.1.5 Balanço de brancos

O balanço de brancos (*white balancing*) refere-se ao processo de corrigir tonalidades de cor erróneas de imagens digitais, causadas principalmente pelas condições de aquisição (em particular, devido à geometria de iluminação e cor

iluminante). Uma solução implementada em muitas câmaras digitais refere-se à utilização de algoritmos que fazem o balanço de brancos automático (mais conhecido por AWB – *Auto White Balance*). Tais algoritmos permitem que a câmara seja capaz de, dinamicamente, detetar a temperatura da cor da luz ambiente e compensar conforme os seus efeitos, ou determinar a partir do conteúdo da imagem a correção de cor necessária devido à iluminação [20], [21].

A partir de uma perspetiva computacional, o balanço de brancos automático é um processo realizado em duas etapas: a iluminância é estimada e as cores da imagem são depois corrigidas com base nesta estimativa. A correção gera uma nova imagem da cena como se tivesse sido tirada sobre uma iluminância padrão conhecida [22].

Um método que é muito utilizado é o *Gray World,* o qual procura equalizar o valor médio dos canais vermelho, verde e azul, diga-se R, G e B, respetivamente. Este é bastante eficiente na prática, exceto em situações onde uma determinada cor é dominante, como por exemplo a tonalidade azul do céu. Para estas situações existe um outro método mais eficiente, conhecido por *White Patch* ou teoria *Retinex,* o qual, como o próprio nome indica, é baseado na teoria *Retinex* da constância da cor visual [23]. Tais algoritmos são descritos no Anexo A.

Algumas abordagens baseadas na correlação também têm sido propostas, contudo apresentam elevada complexidade computacional, não sendo adequadas para aplicações em tempo-real [22].

2.2 Compressão de imagem e vídeo

O processo de compressão de imagem e vídeo envolve um par de sistemas complementares, constituído por um compressor (ou codificador) e um descompressor (ou descodificador): o codificador converte os dados da fonte numa forma comprimida antes da transmissão ou armazenamento, e o descodificador converte a forma comprimida de volta numa representação dos dados de imagem ou vídeo originais. O par formado pelo codificador e descodificador é chamado de *codec* (*enCOder/DECoder*) [2]. O *codec* deve permitir representar os dados usando o menor número de *bits* possível e com uma fidelidade tão alta quanto possível, contudo é difícil satisfazer ambos os requisitos simultaneamente, uma vez que um menor número de *bits* tipicamente produz uma imagem de reduzida qualidade no descodificador [2].

A remoção do número de *bits* no processo de compressão dá-se à custa da remoção de redundância na representação de dados, ou seja, remoção de

componentes que não são necessários para a reprodução fiel dos dados [2]. Existem essencialmente quatro tipos de redundância num sinal de vídeo [1]:

- Espacial: ocorre onde píxeis vizinhos numa imagem ou *frame* estão relacionados, como acontece por exemplo, num objeto de uma única cor;
- Temporal: ocorre quando a relação é entre *frames* consecutivas, ou seja, quando os dados não mudam entre *frames* consecutivas;
- Percetiva: está relacionada com o sistema visual humano, pois este não trata toda a informação visual com a mesma relevância. Por exemplo, o facto do sistema visual humano ser mais sensível à informação de luminância que aquela de cor, permite eliminar redundância percetiva pela utilização dos esquemas de subamostragem referidos na secção 2.1.4;
- Estatística: ocorre devido aos parâmetros surgirem numa imagem com probabilidades diferentes, de modo que uma forma de remover esta redundância é codificando com menos *bits* os parâmetros que aparecem com maior frequência.

As técnicas de compressão podem ser com perdas ou sem perdas. Quando a compressão é sem perdas, os dados reconstruidos a partir do descodificador são uma cópia perfeita dos dados originais, contudo os dados são comprimidos numa quantidade reduzida comparativamente com aquela alcançada pelas técnicas com perdas [2]. Neste tipo de compressão, os dados descomprimidos não são exatamente iguais aos originais, tal como acontece na compressão sem perdas, de modo que os elevados rácios de compressão são atingidos à custa de uma perda de qualidade visual [2].

Um codificador de vídeo consiste em três unidades funcionais principais: um modelo temporal, um modelo espacial e um codificador de entropia [2]. O fluxo dos dados ao longo destas unidades encontra-se representado na Figura 2.12**.** Já o codificador de imagem, possui apenas o modelo espacial, por onde entra a imagem a codificar, e o codificador de entropia.

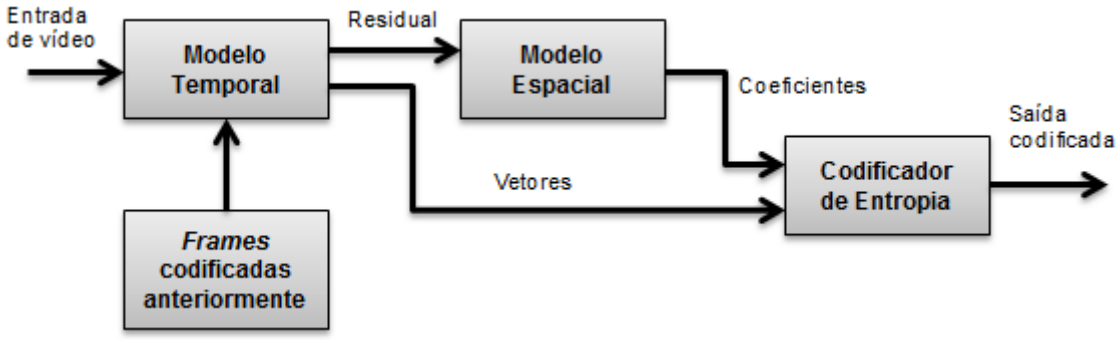


Figura 2.12. Diagrama de blocos de um codificador de vídeo.

No processo de descodificação o fluxo dos dados ocorre no sentido contrário ao da codificação. Isto é, os dados codificados começam por ser descodificados por um descodificador de entropia, seguindo-se a descodificação do modelo espacial. No caso da descodificação de imagem, a imagem reconstruida é obtida na saída do descodificador espacial, enquanto no caso da descodificação de vídeo é ainda necessário descodificar o modelo temporal.

2.2.1 Modelo temporal

As mudanças entre *frames* são causadas pelo movimento dos elementos da imagem (movimento de elementos rígidos, como um carro, ou elemento deformável, como um braço), pelo movimento da camara (como aquando da captura de um panorama, ao realizar uma inclinação, zoom ou rotação), por regiões que são descobertas (como uma porção da cena de fundo sendo descoberta devido a um objeto que se move), por mudanças na luminosidade, entre outras causas. No exemplo ilustrado na Figura 2.13, observa-se um cenário em que o fundo é estático, e as diferenças entre as *frames* devem-se apenas ao movimento do humano. O modelo temporal, ou codificação *interframe*, como também é conhecido, é responsável pela codificação destas diferenças entre *frames* [1].

Nalguns padrões de compressão de vídeo, podem se distinguir 3 tipos de *frames* em função da forma como a *frame* é codificada. Quando uma *frame* transporta todo o seu conteúdo, não precisando da informação de outras *frames* aquando da descompressão, como é caso da primeira *frame* da Figura 2.13, diz-se que a *frame* é do tipo I. Já uma *frame* que transporte as diferenças em relação a uma *frame* do tipo I, e precise dessa *frame* para a sua correta descodificação, então é do tipo P, como é o caso da segunda e terceira *frames* da Figura 2.13. Já as *frames* do tipo B transportam as diferenças em relação a duas ou mais *frames* do tipo I ou P [2].

Figura 2.13. Redundância temporal entre *frames* consecutivas [24].

No codificador de vídeo, o modelo temporal recebe como entrada uma sequência de vídeo não comprimida, e procura reduzir a redundância temporal normalmente pela realização de uma predição da *frame* de vídeo atual. A saída é uma *frame* residual, obtida pela subtração da *frame* predita à *frame* atual real, e um conjunto de parâmetros do modelo, tipicamente um conjunto de vetores de movimento descrevendo como o movimento foi compensado [2].

Algumas técnicas usadas na implementação deste modelo são:

- Predição a partir da *frame* anterior [2];
- Compensação de movimento (CM) baseada no fluxo ótico [2]
- Estimação e compensação de movimento baseada em blocos [2];
- Estimação e compensação de movimento sub-píxel [2];
- Compensação de movimento baseado na região [2].

Para a situação ilustrada no exemplo da Figura 2.13, que é muito comum, as técnicas com compensação do movimento entre *frames* normalmente são as que proporcionam uma melhor predição [2]. A predição com estimação e compensação de movimento baseada em blocos é um método largamente usado, e encontra-se descrito no Anexo B.

2.2.2 Modelo espacial

O modelo espacial é responsável pela remoção das redundâncias espaciais. Na codificação de vídeo, atua sobre a *frame* residual proveniente do modelo temporal, enquanto na codificação de imagem atua diretamente na imagem original a codificar. A codificação realizada por este modelo também é conhecida por codificação *intraframe* [1].

Existem diferentes abordagens para a codificação deste modelo, tais como a codificação preditiva de imagem [2] e a codificação em planos de *bit* [10]. Uma implementação típica que envolve perda de dados é a baseada em transformadas [2], a qual é descrita com mais detalhe no Anexo C. Esta implementação é realizada em três fases:

1. Transformação: A transformada permite representar os dados num domínio diferente, e no contexto da compressão, tem como objetivo descorrelacionar os dados de forma a facilitar a compressão dos mesmos [25]. De entre as mais utilizadas para compressão estão a transformada discreta do cosseno (*Discrete Cosine Transform,* DCT) [2], [3], e a transformada *wavelet* discreta (*Discrete Wavelet Transform,* DWT) ou simplesmente *wavelet* [2], [3].

2. Quantização: A quantização refere-se ao processo de representar ou mapear um conjunto de valores num outro conjunto mais pequeno [25]. Esta é uma forma de reduzir a redundância percetiva, pela remoção dos coeficientes pouco significativos que transportam pouca informação, como aqueles resultantes da DCT ou *wavelet* próximos de zero [2]. É uma operação irreversível, resultando portanto, nalguma perda de informação. Evidentemente, este bloco não é incluído nos sistemas de compressão sem perdas.
3. Reordenação: A reordenação é realizada para agrupar os coeficientes quantizados diferentes de zero numa ordem linear, e representar os coeficientes de valor nulo eficientemente, permitindo uma maior compactação aquando da codificação da entropia. O caminho de reordenação ótimo depende essencialmente da distribuição dos coeficientes de valor zero [2], [25].

2.2.3 Codificação da entropia

No codificador de entropia os elementos da imagem ou sequência de vídeo representados por uma série de símbolos são convertidos num fluxo de dados (ou *stream*) adequado à transmissão ou armazenamento [2]. Para tal, é criado um código de comprimento fixo ou variável, e os símbolos são mapeados de acordo com esse código. Os símbolos de entrada podem incluir coeficientes da transformada quantizados, vetores de movimento, marcadores (códigos que indicam um ponto de sincronização na sequência), cabeçalhos (cabeçalhos de macrobloco, de imagens, de sequências, etc.) e informação suplementar (informação paralela que não é essencial para a descodificação correta). No Anexo D são apresentados alguns métodos utilizados para a codificação da entropia.

2.2.4 Medidas de desempenho

O desempenho de um codec de imagem ou vídeo pode ser analisado de diversas formas. Este pode ser avaliado de pontos de vista como a complexidade computacional, a quantidade de compressão e a semelhança dos dados reconstruidos com os originais.

Uma forma básica para medir o quanto foram comprimidos os dados por um dado algoritmo é calculando o rácio de compressão (RC) [26], que é dado pelo rácio entre o número de *bits* necessários para representar os dados originais, considere-se Q, e o número de *bits* necessários para representar os dados reconstruidos, considere-se P, ou seja

$$RC = \frac{Q}{P} \ . \tag{2.12}$$

Quando a compressão é com perdas, os dados reconstruidos diferem dos originais. Assim sendo, é conveniente determinar a eficiência do algoritmo tendo em conta a diferença entre os dados originais e os reconstruidos. Esta diferença é conhecida por distorção [25], e determina a qualidade ou fidelidade dos dados reconstruidos. A distorção pode ser avaliada de forma objetiva, através de formulações matemáticas, ou de uma forma subjetiva com base na avaliação visual do utilizador.

A avaliação da qualidade com base na observação por parte de um utilizador, de forma precisa e quantitativa, é uma tarefa dificultada pela subjetividade que lhe está inerente. Os resultados deste tipo de avaliação podem ser influenciados por múltiplos fatores, entre os quais, o ambiente onde é realizada a avaliação, o estado de espirito do observador, a fidelidade espacial (o quão óbvia é a distorção, ou o quão claramente podem ser vistas as partes de uma cena), e a fidelidade temporal (o quão natural e "suave" parece o movimento) [2]. Alguns métodos para a realização deste tipo de avaliação foram publicados pela ITU (*International Telecommunication Union*), e podem ser encontrados nos seguintes documentos:

- *ITU-R Recommendation BT.500-13: Methodology for the Subjective Assessment of the Quality of Television Pictures* [27]*;*
- *ITU-T Recommendation P.910: Subjective Video Quality Assessment Methods for Multimedia Applications* [28]*;*
- *ITU-R Recommendation BT.710: Subjective Assessment Methods for Image Quality in High-definition Television* [29]*.*

Nalguns métodos sugeridos nestes documentos é pedido ao observador para fazer uma avaliação quanto à qualidade e uma quanto à distorção, de acordo com as categorias apresentadas na Tabela 2.2 (a) e na Tabela 2.2 (b), respetivamente.

Tabela 2.2. Categorias usualmente utilizadas na avaliação da qualidade e distorção [27], [28], [30].

(a) Qualidade

Número	Categoria de qualidade
5	Excelente
4	Boa
3	Razoável
2	Pobre
1	Má

(b) Distorção

Número	Categoria de distorção
5	Impercetível
4	Percetível mas pouco perturbante
3	Ligeiramente perturbante
2	Perturbante
1	Muito perturbante

Uma forma comum de tratar os dados obtidos através destes métodos é calcular a média das avaliações realizadas pelos observadores através do *Mean Opinion Score* (MOS) [31], [30], dado pela equação

$$MOS = \frac{\sum_{i=1}^{k} n_i C_i}{\sum_{i=1}^{k} n_i}, \tag{2.13}$$

onde C_i é o valor numérico correspondente à categoria i, n_i é o número de apreciações nessa categoria e k é o número de categorias na escala [30].

A avaliação objetiva da distorção também pode ser realizada de diferentes formas. Uma delas é pelo número médio de *bits* necessários para representar uma única amostra (N_b) [32]. Este valor é normalmente chamado de taxa, e pode ser calculado por

$$N_b = \frac{P}{N}, \tag{2.14}$$

em que N é o número total de píxeis que representam a imagem, e P é o número de *bits* necessários para representar os dados reconstruidos. Em [32] é sugerida uma forma de classificar a imagem em função do valor de N_b, a qual encontra-se ilustrada na Tabela 2.3.

Tabela 2.3. Classificação da qualidade de imagem em função da taxa.

N_b (*bits*/píxel)	Qualidade da imagem
0,25 – 0,5	Moderada a boa qualidade
0,5 – 0,75	Boa a muito boa qualidade
0,75 – 1,5	Excelente qualidade
1,5 – 2,0	Normalmente indistinguível da original

A forma objetiva mais comum de medição da distorção é calculando a relação sinal-ruído de pico (*Peak Signal-to-Noise Ratio,* PSNR), dada por [33]

$$PSNR = 20 \log_{10} \frac{max|P_i|}{\sqrt{MSE}}, \tag{2.15}$$

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (P_i - Q_i)^2, \tag{2.16}$$

em que P_i e Q_i são os valores do píxel i reconstruido e original, respetivamente, $max|P_i|$ é o valor máximo que pode tomar P_i, de acordo com o número de *bits* que define a profundidade da imagem (no caso de uma imagem de 8 *bits* $max|P_i|$ é 255), e MSE significa *Mean Square Error*, ou seja, Erro Quadrático Médio (EQM).

Apesar das técnicas descritas serem largamente utilizadas, é preciso ter em atenção que os resultados das técnicas objetivas nem sempre estão de acordo com

as subjetivas, de modo que a utilização de apenas uma delas para a medição da distorção de uma imagem ou vídeo não é completamente fiável.

Note ainda que existe uma relação de compromisso entre o rácio de compressão e a qualidade da imagem ou vídeo. Altos rácios de compressão estão associados a menor qualidade. Uma forma comum de analisar o desempenho de um codec e comparar com o de outro é exatamente pela computação da curva caraterística taxa-distorção, na qual o PSNR é dado em função da taxa de *bits* codificados [2].

2.2.5 Padrões de compressão

Os padrões de compressão definem um conjunto de métodos que devem ser utilizados para a codificação e descodificação de um dado tipo de dados. Os codecs desenvolvidos de acordo com um padrão internacional (tal como os padrões sancionados pela *International Standardization Organization* (ISO)) normalmente usam tecnologia patenteada, estando uma autoridade de licenciamento responsável pela cobrança das taxas apropriadas em nome dos detentores das patentes [1]. Por outro lado, algumas companhias optam por desenvolver os seus próprios codecs, designados por padrões proprietários [1], [34].

Na Tabela 2.4 encontram-se descritos alguns dos padrões de compressão mais conhecidos.

Tabela 2.4. Padrões de compressão de imagem e vídeo.

Padrão	Descrição	Aplicações
JBIG/JBIG1 (*Joint Bi-level image Experts Group*), 1993	• Compressão de imagens, sem perdas [10].	• Imagens binárias; • Imagens de tom contínuo até 6 *bits*/píxel
JBIG2, 1995	• Sucessor do JBIG1; • Compressão com ou sem perdas [10].	• *Internet*; • FAX.
JPEG (*Joint Photographic Experts Group*), 1994	• Compressão de imagens, com ou sem perdas [3]. • É um dos métodos mais populares para a compressão de imagens na Internet [10].	• Imagens de qualidade fotográfica.
JPEG-LS, 1998	• Compressão de imagens, com ou sem perdas [25].	• Imagens de qualidade fotográfica.
JPEG2000, 2000	• Sucessor do JPEG; • Pode ser com ou sem perdas; • Permite maior compressão de imagens de qualidade fotográfica [35].	• Imagens de qualidade fotográfica.

Padrão	Descrição	Aplicações
DV (*Digital Video*), 1998	• Compressão de vídeo [10].	• Aplicações e equipamento de produção de vídeo semi-professional e doméstico (exemplo: *camcorders*).
MPEG-1 (*Motion Pictures Experts Group*), 1993	• Padrão para compressão de vídeo progressivo • As taxas de *bits* alcançadas variam aproximadamente de 400 *kbits*/s a 1,5 *Mbits*/s [1] • Suporta resolução de 352x288 a 25 fps para sistemas PAL, e 352x240 a 30 fps para sistemas NTSC [1].	• CD-ROM
MPEG-2, 1994	• Extensão do padrão MPEG-1 [10] • As taxas de *bits* alcançadas variam aproximadamente de 1,5 *Mbits*/s a 15 *Mbits*/s [1]. • Suporta resolução 720x480 [1]. • Suporta vídeo interlaçado e HDTV [10].	• Televisão *broadcast;* • DVD.
MPEG-4, 1998	• Extensão do padrão MPEG-2 • As taxas de *bits* alcançadas variam aproximadamente de 28,8 *kbits*/s a 500 *kbits*/s [1]. • Suporta resoluções 176x144 e 352x288 [1]	• *Streaming*.
H.261, 1988-1990	• Padrão original para compressão de vídeo progressivo [1]; • Desenvolvido para atuar sobre linhas ISDN (*integrated services digital network*); • Suporta resoluções 352x288 e 176x144, chamadas CIF (common intermediate format) e QCIF (Quarter CIF), respetivamente; • Permite reduzidos tempos de atraso; • Amostragem de 4:2:0; • As taxas de *bits* alcançadas variam aproximadamente de 384 *kbits*/s a 2 *Mbits*/s [1].	• Videoconferência.
H.263, 1992	• Versão melhorada do H.261; • As taxas de *bits* alcançadas variam aproximadamente de 28,8 *kbits*/s a 768 *kbits*/s [1]. • Para além das resoluções CIF e QCIF, suporta resoluções adicionais: SQCIF (*Sub-QCIF*, 128x96), 4CIF (704x576) e 16CIF (1408x512) [1] [10].	• Videoconferência • Aplicações associadas a baixas taxas de *bits*
H.264 / MPEG-AVC (*advanced video coding*), 2002	• Extensão H.263; • Muito flexível em termos de taxas de *bits* e de resoluções, tornando-o adequado para uma larga gama de aplicações.	• Videoconferência; • Videovigilância; • Televisão *broadcast*; • Etc.

Padrão	Descrição	Aplicações
MJPEG (*Motion JPEG*)	• Formato de compressão no qual cada *frame* é comprimida independentemente usando o JPEG [10].	• Videovigilância; • Etc.
VC-9 (*video codec* 9)	• Versão otimizada do MPEG-4 para maior compressão • Alcança taxas de *bits* inferiores a 1 *Mbits*/s • Suporta resoluções até 2048x1536 [1]	• Aplicações de alta definição • *Streaming*
H.265/ MPEG-HEVC (*High-Efficiency Video Coding*), 2013	• Versão melhorada do H.264/MPEG-AVC • Suporta resoluções UHD (ultra high definition) de 4K UHD (3840x2160) e 8K (7680x4320), e resoluções até 8192x4320 [36].	• Videoconferência; • Videovigilância; • Televisão *broadcast*; • *Home cinema;* • Etc.
VP9, 2013	• Codec de vídeo *open-source* [34] • Suporta resoluções até 4K UHD.	• Videoconferência; • Etc.

De entre os padrões de compressão de imagem apresentados na Tabela 2.4**,** verifica-se que o JBIG1 e o JBIG2 são os mais adequados a imagens com poucos *bits* de profundidade. Mais precisamente, o JBIG2 apresenta-se como uma melhor opção que o JBIG1, uma vez que permite rácios de compressão 2 a 4 vezes superior que os alcançados pelo JBIG1 [37].

Já para imagens de qualidade fotográfica, associadas a uma maior profundidade de *bits*, são apresentadas três opções. O JPEG-LS proporciona uma compressão sem perdas, e portanto, sem perda de qualidade, contudo implica reduzidos rácios de compressão (reduções em fatores de cerca 3 a 4) [2], limitando muito as aplicações em que pode ser utilizado. Para aplicações em que sejam pretendidos maiores rácios de compressão, o JPEG e o JPEG2000 são os mais adequados.

O JPEG foi desenvolvido com o intuito de suportar uma variedade de aplicações com compressão de imagens de tom contínuo, em diversos tamanhos de imagem, e em qualquer espaço de cor. Permite alcançar rácios de compressão ajustáveis pelo utilizador, associados a boa a excelente qualidade de reconstrução. Outro objetivo deste padrão é proporcionar complexidade computacional adequada para uma variedade de implementações práticas [3]. Para tal, dispõe de quatro diferentes modos de operação. Três deles são baseados na DCT, e portanto, são com perdas, enquanto o outro é sem perdas [3]. Uma das implementações mais comuns é baseada na DCT, e encontra-se descrita no Anexo E. O padrão JPEG proporciona desempenho excelente a taxas acima de 0,25 *bits*/píxel, contudo a taxas menores ocorre uma rápida degradação na qualidade da imagem reconstruida [25].

Quanto ao JPEG2000, foi desenvolvido para proporcionar um sucessor mais eficiente que o JPEG, proporcionando maiores rácios de compressão, ao mesmo tempo que não exibe artefactos de blocagem, como acontece nos métodos baseados na DCT (ver Anexo C) [2]. Para além disso, veio a proporcionar flexibilidade acrescida tanto à compressão de imagens de tom continuo como ao acesso dos dados comprimidos, uma vez que porções de uma imagem comprimida JPEG2000 podem ser extraídas para retransmissão, armazenamento, visualização e/ou edição [10].

O JPEG2000 usa a transformada *wavelet* discreta como método de codificação básico, a quantização de coeficientes é adaptada às escalas e sub-bandas individuais e os coeficientes quantizados são codificados aritmeticamente numa base de plano de *bit* [10]. Apesar do JPEG ser uma tecnologia mais antiga e apresentar um sub-desempenho em relação ao JPEG2000 e a muitos formatos de compressão proprietários, ainda é largamente usado para o armazenamento de imagens em camaras digitais, PCs (*Personal Computer*) e páginas WEB [2].

Relativamente aos padrões de compressão de vídeo, é apresentada uma maior variedade, contudo os mais relevantes na evolução dos mesmos são o H.262/MPEG-2 e o H.264/MPEG-4 AVC (ver Anexo F), desenvolvimento dos quais foi levado a cabo pela parceria conhecida como *Joint Colaborative Team on Video Coding* (JCT-VC). A primeira versão do padrão H.264/MPEG-4 AVC foi desenvolvida para satisfazer a necessidade crescente por maior eficiência de codificação, especialmente no que diz respeito à televisão de definição padrão (*Standard Definition Television,* SDTV) e à transmissão de vídeo sobre canais com baixa taxa de dados. Como resultado, foi obtido um padrão com uma eficiência de compressão (em termos de taxa de *bits*) cerca de 50% superior em relação ao H.262/MPEG-2 [34]. A melhoria em relação aos outros padrões desenvolvidos anteriormente a este também é significativa.

No gráfico da Figura 2.14**,** obtido em [24], é realizada uma comparação da taxa de *bits*, dado o mesmo nível de qualidade de imagem, entre os padrões H.264/MPEG-4 AVC, MPEG-4 sem compensação de movimento, MPEG-4 com compensação de movimento, e o MJPEG. Como se pode observar, a taxa de *bits* apresentada pelo H.264/MPEG-4 AVC é de cerca 50% inferior à do MPEG-4 com compensação de movimento, cerca de 3 vezes inferior à do MPEG-4 sem compensação de movimento, e cerca de 6 vezes inferior à apresentada pelo MJPEG.

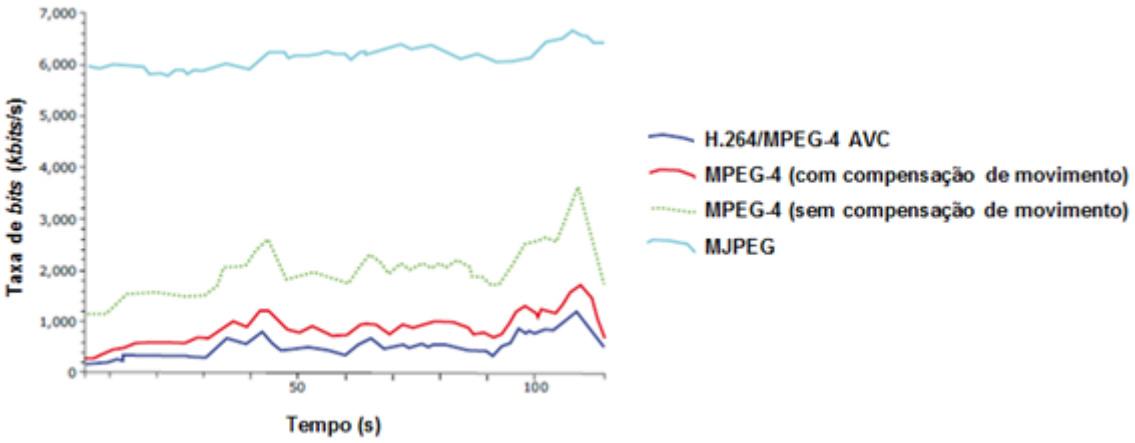


Figura 2.14. Comparação das taxas de *bits* alcançadas na compressão do mesmo vídeo por diferentes padrões [24].

O padrão H.264/MPEG-4-AVC não suporta conteúdo de vídeo de ultra alta definição (*ultra high definition*, UHD), o que impulsionou o desenvolvimento do padrão H.265/MPEG-HEVC. Este padrão foi desenvolvido para ser aplicável em quase todas as aplicações existentes para o H264, dando enfase à codificação de vídeo de alta resolução. O desenvolvimento do codec VP9, proprietário da Google Inc., também já teve em conta o suporte para conteúdo de vídeo 4K UHD. Em [34] é realizado um estudo para comparar o desempenho dos codificadores H.264/MPEG-4-AVC, H.265/MPEG-HEVC e VP9, tendo o H.265/MPEG-HEVC proporcionado melhorias na taxa de *bits* na ordem dos 43,3% e 39,9% relativamente ao VP9 e ao H.264/MPEG-AVC, respetivamente. Os resultados obtidos neste estudo mostraram também que, apesar do VP9 apresentar maiores taxas de *bits* que o H.264/MPEG-AVC, está associado a tempos de codificação típicos mais de 100 vezes superiores aos medidos na codificação H.264/MPEG-AVC. Já em relação aos tempos de codificação associados ao H.265/MPEG-HEVC, apresentaram-se superiores aos associados ao VP9, por um fator médio de 7,35.

Os elevados tempos de compressão medidos para o H.265/MPEG-HEVC em relação aos medidos para o H.264/MPEG-4-AVC e para o VP9, revelam também que o aumento do desempenho se dá à custa de uma elevada complexidade computacional, o que representa um obstáculo à implementação em determinadas aplicações com limitações de consumo de potência.

2.3 Sistemas de compressão de imagem e vídeo

Na secção anterior mostrou-se que o avanço nos padrões de compressão de imagem e vídeo têm permitido maiores eficiências de codificação, as quais têm implicado também aumentos na complexidade computacional. Quando o processamento de sinais de imagem e vídeo multidimensionais é realizado em

tempo real, é necessário ainda o tratamento de fluxos de grandes volumes de dados contínuos. Tendo em conta que as aplicações de codificação de imagem e vídeo estão cada vez mais associadas a dispositivos móveis, a procura por reduzido consumo de potência e por custos reduzidos também é crescente. Geralmente, o desenvolvimento destas aplicações baseia-se numa das seguintes opções: implementação em *software* numa plataforma de sistema geral, implementação em FPGA (*Field Programmable Gate Array*) e implementação em SoC.

As soluções em *software* estão entre as primeiras a serem desenvolvidas quando um padrão está em fase de projeto. Isto por apresentarem a vantagem de poderem ser rapidamente alteradas no sentido de cumprirem requisitos específicos [38].

Apesar destas soluções estarem associadas a custos moderados quando comparadas com as baseadas em FPGAs e SoCs, tendem a ser aquelas com o menor desempenho. Tal está relacionado com o facto de requererem tempo significativo para a codificação, descodificação e reprodução de vídeo (principalmente alta resolução), não se adequando a aplicações em tempo real. Para além disso, estas implementações tendem a ser alojadas em sistemas de tamanho semelhante aos pequenos PCs [38].

Já as soluções baseadas em FPGAs apresentam melhores desempenhos que aquelas baseadas em *software*, e adequam-se às aplicações em tempo real [38]. As implementações deste género podem ser muito diferentes entre si, pois a possibilidade de alteração do *hardware* dá liberdade aos fabricantes para personalizarem as capacidades do codificador. Tal é útil para casos em que os clientes requerem novas funcionalidades ou em que haja alterações nos padrões de codificação. A atualização dos produtos pode ser realizada pela reprogramação da FPGA [38].

Estes produtos estão associados a elevados custos, os quais se devem tanto ao uso de FPGAs, como também às despesas de aquisição das competências e recursos necessários para o desenvolvimento do algoritmo de codificação. Por este motivo, as soluções baseadas em FPGAs são mais direcionadas para mercados menos sensíveis ao preço, como o *broadcasting* de ponta. Costumam estar disponíveis nas fases iniciais de desenvolvimento dos padrões de codificação [38].

As implementações baseadas em SoCs normalmente surgem numa fase mais avançada, quando o desenvolvimento de um padrão de codificação já estabilizou e se encontra bem definido. Estes produtos são desenvolvidos pelos fabricantes de semicondutores, e são altamente integrados com um processador de alto

desempenho e funcionalidade específica para uma aplicação. Nomeadamente para o caso da codificação de vídeo, a solução mais encontrada no mercado passa pela utilização de núcleos ARM para a realização das operações complexas associadas às funções de codificação [38]. Existem, contudo, outros fabricantes que produzem processadores com o mesmo propósito, como é o caso da Intel (ver Tabela G.1 no Anexo G). Dispositivos como *set-up boxes* (STBs), STBs com DVR (*Digital Video Recorder*) integrado e câmaras de vídeo, utilizam SoC com integração suficiente *on-chip* que satisfaz os requisitos completos de uma dada aplicação [38].

O elevado nível de integração associado aos SoCs oferece numerosos benefícios, de onde se destacam: melhor desempenho para codificação, menores dimensões, menor potência de consumo e menores custos. Até mesmo os fabricantes das soluções baseadas em *software* e FPGAs, normalmente adotam estes produtos, uma vez que permitem que outros produtos proliferem no mercado. Apesar das alterações de *firmware* permitirem a atualização das funções que são executadas no processador *on-chip,* a funcionalidade mantém-se fixa, sendo esta a principal desvantagem dos SoCs [38].

Nesta secção é dado foco às soluções em *hardware,* sendo apresentada uma arquitetura básica de *hardware* direcionada para aplicações de codificação de imagem e vídeo. Posteriormente são apresentadas algumas implementações comerciais, e é realizada uma comparação entre elas.

2.3.1 Arquiteturas de *hardware* para codificação de imagem e vídeo

A arquitetura do *hardware* de um sistema pode ser classificada como dedicada ou programável. No caso de ser dedicada, significa que cada um dos diferentes módulos ou blocos que a constituem têm uma arquitetura completamente adaptada a tarefas específicas. Estas permitem minimizar os custos no *hardware* e baixar o consumo de potência, no entanto esta dedicação apresenta como desvantagem a impossibilidade de posteriores extensões para outras tarefas. Por outro lado, as arquiteturas programáveis são aquelas em que cada módulo oferece flexibilidade para permitir a execução de várias tarefas no mesmo *hardware* apenas modificando o *software*. Consequentemente, a flexibilidade deste *hardware* implica tanto maiores custos como maior consumo de potência relativamente a uma arquitetura dedicada [39].

Porém algumas aplicações estão associadas a esquemas híbridos, cujas tarefas apresentam diferentes caraterísticas computacionais. De modo a processar eficientemente as várias tarefas individuais, as arquiteturas de sistema têm se tornado hibridas, as quais são compostas tanto por módulos de arquitetura dedicada

como por módulos de arquitetura programável. Contudo, continuam a ser classificadas como dedicadas ou programáveis, conforme o tipo dominante de arquitetura entre os diferentes módulos constituintes do sistema.

A seleção entre arquiteturas de sistema programáveis ou dedicadas depende essencialmente do campo de aplicação do dispositivo. As arquiteturas programáveis são adequadas para aplicações com demandas computacionais e algoritmos muito variáveis, enquanto arquiteturas dedicadas são adequadas para aplicações bem definidas com funcionalidade fixa.

No caso dos algoritmos de codificação de imagem e vídeo, podem ser considerados esquemas de codificação híbridos uma vez que envolvem diferentes procedimentos, os quais, em conjunto, reduzem as redundâncias estatísticas, espaciais e temporais dos sinais de imagem e vídeo. Ainda assim, esta é considerada uma aplicação específica, de modo que, para que se obtenham as mais eficientes soluções em *hardware*, uma arquitetura dedicada é a mais indicada.

O desenvolvimento de uma arquitetura dedicada eficiente envolve o mapeamento das tarefas individuais em diferentes módulos, de forma a identificar o tipo de arquitetura mais adequada para cada um deles, e a otimizá-la em termos de requisitos de desempenho, área e potência, alcançando portanto uma arquitetura de sistema dedicada com o mais alto grau de adaptação. Tal implica ainda uma análise detalhada aos algoritmos pretendidos, de modo a que as caraterísticas computacionais especiais inerentes aos mesmos possam ser exploradas [39].

Na Figura 2.15 é apresentada a arquitetura básica de um sistema de comunicação multimédia, a qual também inclui a codificação de imagem e vídeo.

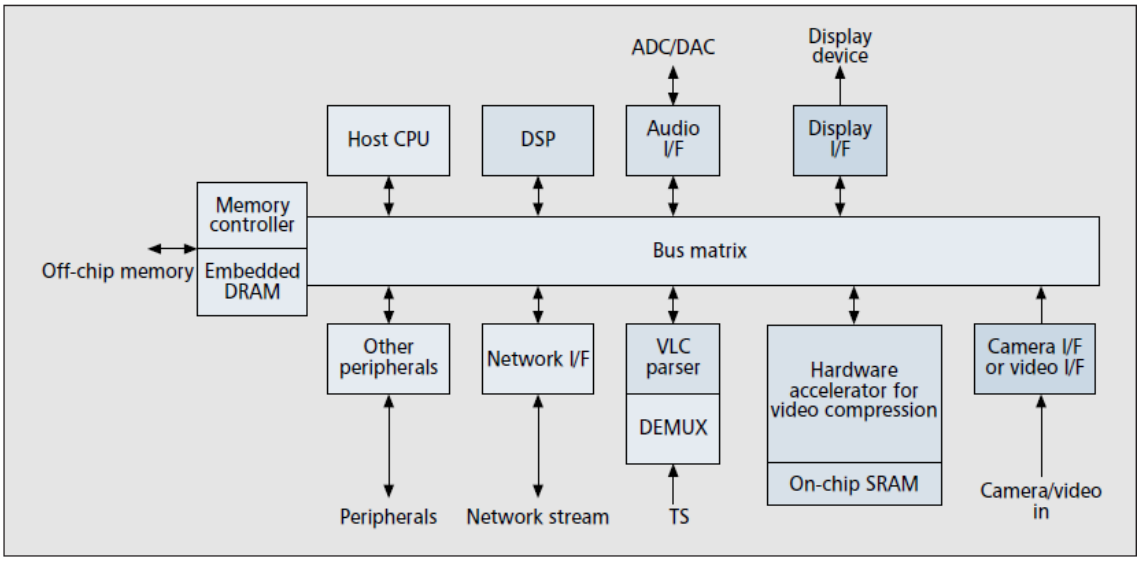


Figura 2.15. Arquitetura de *hardware* geral para um sistema de comunicação multimédia [40].

O CPU (*Central Processing Unit*) hospedeiro (*Host CPU*) desempenha um papel importante em todo o sistema, pois é responsável pelo controlo de todos os módulos bem como pela alocação de memória para os mesmos. Em aplicações mais complexas este pode correr um sistema operativo que serve de ponte entre o *software* da aplicação e o *hardware*.

A DSP (*Digital Signal Processor*) é normalmente integrada no sistema para codificação de áudio, podendo também implementar alguns módulos de compressão de vídeo, como as transformadas e a CM. A interface de áudio (*Audio I/F*) serve para receber ou fornecer dados áudio, e está associada a DACs e a ADCs para conexão a microfones e a altifalantes.

A interface de visualização (*Display I/F*) é responsável pela conexão aos dispositivos de reprodução dos dados de vídeo, e pode tornar-se um módulo complexo quando integra funções gráficas 2D e 3D, como o redimensionamento de *frames* necessário para alguns dispositivos de visualização. O canal de comunicação, para uma rede sem fios por exemplo, é implementado através da interface de rede (*Network I/F*).

O analisador de codificação de comprimento variável (*VLC parser*) e o desmultiplexer (*DEMUX*) são responsáveis pela descodificação do fluxo de transporte (*transport stream, TS*) e pela separação em fluxos de vídeo, áudio ou de sistema. A aquisição de *frames* é realizada através da interface de câmara ou vídeo (*Camera I/F or vídeo I/F*), as quais podem ser provenientes de um sensor de imagem ou de um sinal de vídeo. Este módulo pode se tornar complexo se implementar funções de processamento de sinal de imagem [40].

O acelerador em *hardware* para compressão de vídeo (*Hardware accelerator for vídeo compression*), tal como o nome indica, representa um módulo dedicado para a realização de compressão de imagem e vídeo. Este é constituído por uma memória *on-chip* e por módulos mais pequenos, os quais podem ser conectados uns aos outros através de um *bus* local e controlados por um controlador local, que por sua vez pode comunicar com o CPU hospedeiro para obter as instruções. Outra alternativa é conectar os vários módulos diretamente ao *bus* do sistema. A escolha da mais adequada arquitetura varia conforme a aplicação final [40].

Para além dos módulos descritos, esta arquitetura também inclui um controlador de memória e um *bus matrix.* A *bus matrix* é responsável pela conexão entre os vários módulos, os quais transferem entre si grandes quantidades de dados. Por sua vez, o controlador de memória é responsável pelo acesso a esses dados através de *frame buffers,* e pelo seu transporte entre os vários módulos e uma

memória. Este pode ser um processo lento e pode implicar uma elevada potência se for utilizada uma memória *off-chip.* Felizmente, a tecnologia de integração em muito larga escala (*very large scale integration,* VLSI) tem permitido a integração de DRAM (*Dynamic Random Access Memory*) ou SRAM (*Static Random Access Memory*) como *frame buffer on-chip,* permitindo assim aumentar significativamente o desempenho e reduzir o consumo de potência [39].

A *bus matrix* também pode ser modificada de acordo com as aplicações finais. Uma compreensão mais profunda dos padrões de comunicação entre módulos é essencial para desenvolver uma *bus matrix* eficiente, a qual pode proporcionar maior largura de banda e consumir menor potência. Pode-se considerar que existe um compromisso entre a flexibilidade e a eficiência da *bus matrix*: a utilização de um *bus* global proporciona maior flexibilidade mas menor eficiência, enquanto a utilização de um *bus matrix* particionada em vários *links* de dados, em que cada um é dedicado com total adaptação a um algoritmo específico, proporciona mais alta eficiência, uma vez os módulos conectados aos diferentes *links* não interferem uns com os outros, no entanto perde alguma flexibilidade [39].

Esta arquitetura pode ser integrada numa FPGA ou num único chip, como um SoC, sendo geralmente esta a chave para alcançar os desejados requisitos de baixo consumo energético, alto desempenho e baixo custo dos produtos de multimédia atuais [40].

2.3.2 Implementações comerciais

A arquitetura descrita na secção anterior representa a base para uma larga gama de aplicações de codificação de vídeo, entre as quais, *broadcasting* e câmaras de vídeo. Contudo, várias alterações podem ser realizadas no sentido de satisfazer os requisitos específicos de cada aplicação, o que vai determinar os fatores como custo, potência de consumo e desempenho. No Anexo G encontra-se a Tabela G.1 que apresenta as caraterísticas principais de alguns SoC direcionados para a codificação de vídeo disponíveis no mercado. A análise desta permite verificar que as opções vão desde arquiteturas que incluem apenas um processador principal e um acelerador de *hardware*, até aquelas que também incluem DSPs ou processadores de múltiplos núcleos. Por ser uma indústria altamente competitiva, alguns fabricantes não disponibilizam muitos detalhes acerca dos componentes e da arquitetura, de modo que a caraterização realizada nesta tabela, é mais detalhada para alguns casos que noutros.

Existe uma grande variedade de fabricantes deste tipo de produto, sendo que alguns os produzem para uma determinada aplicação específica, enquanto outros apresentam maior flexibilidade neste campo.

A *Texas Instruments*, por exemplo, possui uma família de processadores otimizados para sistemas de vídeo digital, denominada por *DaVinci*. Contudo, dentro desta família, os produtos estão divididos em 5 categorias em função da constituição da unidade de processamento. Esta pode ser constituída, portanto, por apenas um processador ARM9, um processador ARM Cortex-A8, uma DSP, uma DSP e um processador ARM9, ou uma DSP e um processador ARM Cortex-A8, com diferentes combinações resultando em produtos com diferentes caraterísticas [41].

Os produtos que integram apenas uma DSP são adequados a aplicações em que os requisitos em termos de qualidade de vídeo sejam reduzidos, uma vez que usam apenas o codificador H.264 BP, e suportam resoluções só até D1 (720x480). Por outro lado suportam uma gama de temperaturas relativamente grande, o que os torna adequados para aplicações industriais. As outras opções já suportam uma maior variedade de codecs, bem como maiores resoluções. As melhores caraterísticas estão associadas aos produtos que usam uma DSP e um processador ARM Cortex-A8, pois possuem as maiores frequências de processamento, maior número de aceleradores em *hardware* e uma grande variedade de interfaces de comunicação, permitindo alcançar resolução 1080p a 60 fps com o padrão H.264, bem como diversas funcionalidades de processamento de vídeo e imagem. Tais caraterísticas fazem desta opção adequada para uma grande gama de aplicações.

Já a Ambarella desenvolve câmaras de uso geral como também produtos para aplicações mais específicas, nomeadamente soluções para *broadcast*, para cenários de desporto, para câmaras automotivas e câmaras IP de segurança. Todos os produtos integram pelo menos um processador ARM, uma DSP para imagem e uma DSP para vídeo, sendo que a restante constituição bem como as funcionalidades suportadas variam de acordo com a aplicação específica do produto.

A solução para *broadcast* é a mais lenta em termos de velocidade de processamento (apenas 256 MHz), não suporta ligação Wi-Fi, e não contém tantos periféricos como os outros produtos. Contudo suporta codificação H.264 e MPEG-2 de resoluções até 1080p a 60 fps, possui 4 interfaces para MPEG-4 TS, e executa diversas funcionalidades de transcodificação, controlo de taxa de *bits*, conversão de espaços de cor, entre outras. Também possui um módulo para criptografia.

Já para aplicações de câmaras automotivas os requisitos são muito diferentes, de modo que as caraterísticas do SoC também o são. Este suporta a codificação de resoluções até 3 MP a 30 fps, a codificação simultânea de 2 *streams* de vídeo 1080p a 30 fps, possui múltiplas portas de saída de vídeo, um maior número e variedade de periféricos, não suporta descodificação, tendo sido as funcionalidades de transcodificação substituídas por filtros de outras funcionalidades de processamento de imagem. Naturalmente que o CPU utilizado é mais rápido, e a tecnologia de 32 nm que possui permite aumentar o desempenho e reduzir o consumo de potência.

As soluções para câmaras IP de segurança apostam em interfaces flexíveis para a entrada de vídeo, quer a partir de sensores quer de sinais de vídeo, de modo a suportar diferentes tipos de câmaras. A codificação de vídeo pode ser realizada simultaneamente sobre um máximo de 8 *streams,* cuja resolução pode ir até 4K UHD a 30 fps, utilizando os padrões H.264 e MJPEG. Para além disso, permitem uma grande variedade de funcionalidades para processamento de imagem e para análise de vídeo inteligente, permite o controlo da taxa de *bits*, e possui diversos periféricos de armazenamento e de comunicação, incluindo *ethernet* e para ligação sem fios. Para tal, estas soluções podem contar com processadores de dois núcleos e com um módulo para criptografia, além das DSPs de imagem e vídeo. Utilizam tecnologia 45 nm e 32 nm, permitindo o consumo de menos de 1,5 W para codificação de vídeo 1080p a 60 fps, e menos de 2,5 W para vídeo de resolução 4K UHD.

As soluções para câmaras para desporto são as mais exigentes em termos de processamento, uma vez que o elevado movimento a que estão associadas implicam que os dados tenham de ser tratados a uma maior velocidade de modo a ser obtida boa qualidade de vídeo. A frequência dos seus processadores pode ir até 1 GHz. O SoC iOne Smart, por exemplo, está preparado para integração num dispositivo móvel Android, e é composto por um processador ARM Cortex-A9 de dois núcleos, um processador ARM11, duas DSPs de vídeo e uma DSP de imagem [42].

O SoC mais adequado a vídeos com elevado movimento, o A9, tem uma constituição semelhante, mas possui uma DSP para vídeo e um módulo para criptografia, ao invés de duas DSPs para vídeo, e ainda assim está associado a um reduzido consumo de potência (menos de 1 W para codificar vídeo 1080p a 60 fps, e menos de 2 W para codificar vídeo de resolução 4K UHD), uma vez devido à tecnologia 32 nm. A capacidade da interface para o sensor suportar um fluxo de até 700 MP/s permite a captura de vídeo de resolução 4K UHD a 30 fps, 1080p a 120

fps e 720p a 240 fps. Para além disso, de entre as opções para a entrada de vídeo estão incluídas duas interfaces para sensores. Possui diversas interfaces para memória e para periféricos, e suportam várias funcionalidades para processamento dos dados dos sensores e para processamento de imagem.

Como se pode ver na Tabela G.1 no Anexo G, no mercado também já existem arquiteturas que integram processadores ARM de 4 núcleos, naturalmente, com velocidades de processamento superiores, para além de diversos aceleradores em *hardware,* a custos relativamente baixos. Alguns SoC mais recentes, como o Snapdragon 805 e Snapdragon 810 da Qualcomm, ou o MT6595M e MT6795 da MediaTek, para além de possuírem processadores de 8 núcleos, e permitirem frequências de até 2,7 GHz, até já possuem suporte em *hardware* para codificação e descodificação de vídeo com o mais recente padrão de codificação de vídeo (H.265, ou HEVC), para resoluções até ultra HD. O MT6595M está disponível em *smartphones* [43], [44]*,* enquanto o Snapdragon 805 encontra-se implementado em dispositivos como *smartphones, tablet,* e num *smart glass* [45]*.*

As plataformas de desenvolvimento permitem tirar proveito das capacidades dos processadores que integram. Contudo, a análise de diversos produtos permitiu verificar-se que nalguns casos nem todas são aproveitadas, mas por outro lado, existem casos em que a plataforma implementa capacidades e funcionalidades que o processador só por si não suporta. Um exemplo disso é a BeagleBoard-xM que suporta ligação Ethernet 10/100 Mb/s, apesar do DM3730 não possuir interface para tal.

3. Compressão de imagem e vídeo num processador ARM

A análise da compressão de imagem e vídeo foi realizada recorrendo ao controlador DM368. Foram desenvolvidas 3 aplicações em linguagem C, as quais realizam compressão de acordo com os padrões JPEG e H.264, instanciando codecs para o efeito. Os codecs utilizados para tal foram os da *Texas Instruments* para o controlador DM368, e a integração dos mesmos nas aplicações foi realizada com recurso ao Codec Engine [46]. O Codec Engine é um conjunto de APIs (*Application Programming Interface*) que pode ser usado para instanciar e correr algoritmos xDAIS (*eXpress DSP Algorithm Interoperability Standard*) [47]. Este também implementa uma interface xDAIS-DM ou xDM (*eXpress DSP Algorithm Interoperability Standard for Digital Media*) [47], que é uma extensão ao padrão xDAIS que proporciona suporte para codificadores, descodificadores e codecs multimédia digital.

3.1 Plataforma de desenvolvimento

A plataforma usada para o desenvolvimento deste trabalho foi a LeopardBoard com o controlador DM368 [48]. Este possui um núcleo ARM926, cuja frequência de operação máxima é cerca de 432 MHz, tal como se pode verificar na Tabela 3.1. De entre os fatores que a tornam adequada para tal estão o baixo custo e o suporte em *hardware* para compressão de imagem e vídeo segundo diversos padrões.

O diagrama de blocos relativo ao controlador DM368 apresenta-se na Figura 3.1. Os blocos responsáveis pela execução dos codecs são o HDVICP (*High Definition Video and Imaging Co-Processor*) e o MJCP (*MPEG JPEG Co-Processor*). O bloco VPSS (*Video Processing Subsystem*) é responsável pelas interfaces de entrada e saída de imagem e vídeo. Já na Figura 3.2 pode ser observada a LeopardBoard juntamente com outro equipamento utilizado no desenvolvimento deste trabalho, nomeadamente: sensor CCD (*charge-coupled device*) (Cosmic (Pentax) TV Lens 1,4/16 mm) já montado na respetiva plataforma (LI-5M03 para câmara de 5 *Mega-pixel),* adaptador do cabo série ligado na interface UART (*Universal Asynchronous Receiver/Transmitter*), cartão SD (*Secure Digital*) e adaptador para saída DVI (LI-DVI1).

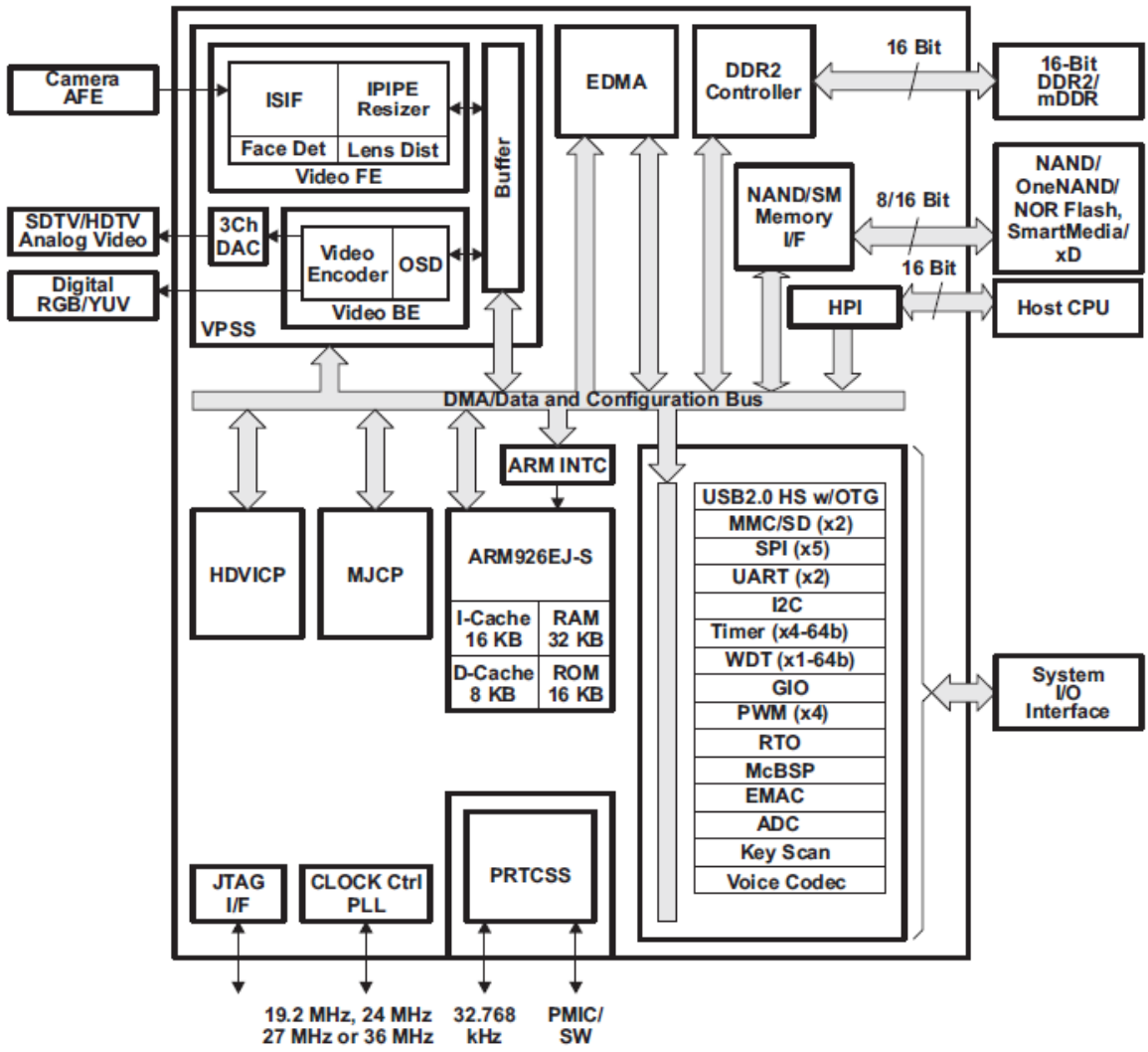


Figura 3.1. Diagrama de blocos do controlador DM368 [48].

Tabela 3.1. Frequências máximas de operação de alguns sub-sistemas do DM368.

Componente	Frequência máxima (MHz)
ARM926	432
Co-processador HDVICP	340
Co-processador MJCP	340
DDR2 (*Double Data Rate 2*)	340
Bloco lógico VPSS	340
***Bus* do sistema periférico e EDMA (*Enhanced Direct Memory Access*)**	170
VPBE (*Video Processing Back End*) – VENC (*Video Encoder*)	74,25
VPFE (*Video Processing Front End*)	120

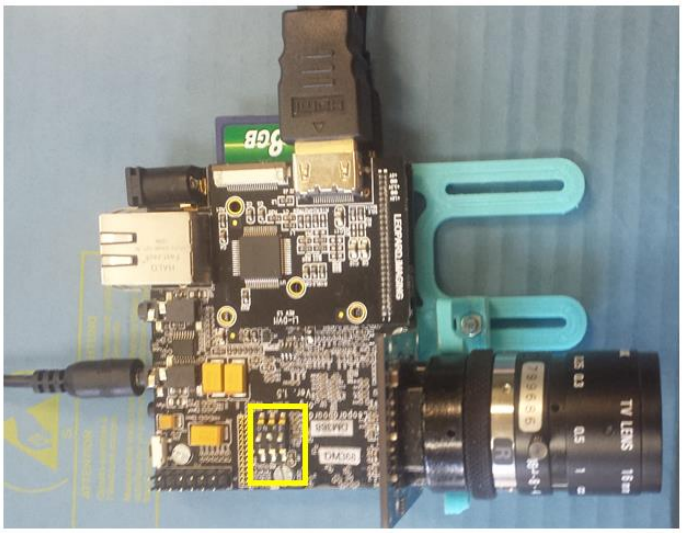


Figura 3.2. LeopardBoard DM368.

O SDK (*Software Development Kit*) utilizado foi o *LeopardBoard DM365 EVAL SDK Turrialba,* uma versão de teste disponibilizada pela *RidgeRun* [49]. De entre os aspetos que o caraterizam destacam-se o *kernel* Linux 2.6.32.17 – que tem suporte para diversos *drivers –,* uma *toolchain* baseada no gcc 4, a capacidade de integração de *hardware* adicional na placa, e a integração com a *toolkit* da *Texas Instruments* para as plataformas *DaVinci* (DVSDK 4.00.02.06). A *RidgeRun* indica que este SDK foi desenvolvido e testado no sistema operativo Ubuntu [50], e requer no mínimo a versão 12.04, de modo que optou-se pela utilização da mesma. Mais precisamente, utilizou-se este sistema operativo sobre uma máquina virtual *Oracle VM VirtualBox*. No Anexo H encontram-se os procedimentos a seguir no sentido de instalar e compilar o SDK, como também para proceder à inicialização da placa.

O módulo denominado por VPSS (*Video Processing Subsystem*), apresentado na Figura 3.1, é responsável pelas interfaces de entrada e saída de imagem. Na sua constituição inclui um módulo denominado por VPFE (*Video Processing Front End*) que proporciona uma interface de entrada para periféricos externos de imagem, tais como sensores de imagem e descodificadores de vídeo. Inclui também um outro módulo, denominado por VPBE (*Video Processing Back End*), que proporciona uma interface de saída para dispositivos de visualização como monitores SDTV analógicos, painéis LCD (*Liquid Crystal Display*) digitais e codificadores de vídeo HDTV (*High Definition Television)* [51].

3.1.1 Interface de entrada de imagem no DM368

O VPFE é a interface de entrada de imagem no controlador DM368, e é constituído por 4 módulos: ISIF (*Image Sensor Interface*), IPIPE (*Image Pipe*),

IPIPEIF (*Image Pipe Interface*) e um mecanismo de deteção facial em *hardware* [48]. A Figura 3.3 apresenta o diagrama de blocos deste componente.

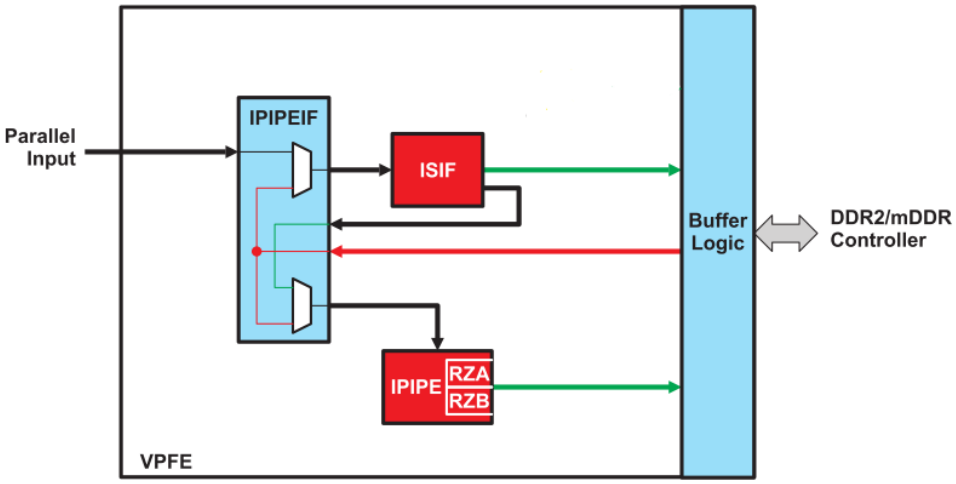


Figura 3.3. Diagrama de blocos do VPFE [51].

O ISIF é responsável por receber dados de imagem/vídeo *raw* (não processados) a partir de um sensor (CMOS ou CCD) [48]. O ISIF também pode receber dados de vídeo YCbCr 4:2:2 em numerosos formatos, tipicamente a partir dos chamados dispositivos de descodificação de vídeo. No caso de receber dados *raw*, a saída do ISIF requer processamento de imagem adicional para transformar a imagem de entrada *raw* numa imagem processada final. Isto tanto pode ser realizado *on-the-fly* no IPIPE, como também em *software* no processador ARM ou nos subsistemas de coprocessamento MPEG/JPEG e *HD Video Image*.

O IPIPEIF é o módulo de interface de sinais de sincronização e de dados para o ISIF e IPIPE. A fonte de dados deste módulo é a porta paralela para o sensor, ISIF ou SDRAM e os dados selecionados são enviados para o ISIF ou IPIPE [48].

O IPIPE é um módulo de processamento de imagem em *hardware* programável que gera dados de imagem nos formatos YCbCr 4:2:2 e 4:2:0 a partir de dados *raw* do sensor CCD/CMOS. O IPIPE também pode ser configurado para operar num modo apenas de redimensionamento, que permite que dados nos formatos YCbCr 4:2:2 ou 4:2:0 sejam redimensionados sem processamento em cada um dos módulos do IPIPE [48].

3.1.2 Interface de saída de imagem no DM368

A interface de saída de imagem no DM368 é o módulo VPBE, o qual é composto pelo módulo OSD (*On Screen Display*) e pelo VENC/DLCD (*Video Encoder / Digital LCD Controller*) [48].

A função principal do módulo de OSD é recolher e misturar os dados de vídeo e dados de exibição, e passá-los para o codificador de vídeo (VENC) num formato YCbCr. Estes dados são lidos a partir da memória externa DDR2/mDDR (*Double Data Rate*), gerando vídeo analógico na saída do VENC, ou dados digitais RGB/YCbCr na saída do DLCD [48].

A saída de vídeo analógico é compatível com formatos como vídeo composto, S-Video, e vídeo componente em HD, enquanto na saída do DLCD são suportados formatos YCbCr 4:2:2 de 8 e 16 *bits,* e RGB de modo série e paralelo [48].

3.2 Codecs de compressão

O controlador DM368 possui suporte em *hardware* para diferentes padrões de compressão, entre os quais H.264 e JPEG [48], cujo desempenho se pretende analisar neste trabalho. Para tal, recorreu-se aos codecs disponibilizados gratuitamente pela *Texas Instruments* (TI), de modo que a análise a realizar encontra-se sujeita às limitações apresentadas pelos mesmos. Tais limitações, bem como outras caraterísticas são apresentadas seguidamente. Quanto aos procedimentos necessários para proceder à integração destes codecs numa dada aplicação, são indicados no Anexo I.

3.2.1 Codec JPEG da TI

O codificador JPEG da Texas Instruments implementa o processo de codificação sequencial baseado na DCT. De entre as limitações a que está associado, encontram-se: o uso das mesmas tabelas de *Huffman* e de quantização para ambos os componentes U e V; permitir o uso apenas de tabelas de *Huffman* definidas por defeito; apenas suportar imagens de 3 componentes; apenas suportar o modo *baseline* [52]. Uma outra limitação significativa no âmbito da análise do processo de codificação refere-se ao facto de que as referidas tabelas de *Huffman* e de quantização não são especificadas na documentação, sabendo-se apenas que as tabelas de quantização são fixas com um fator de qualidade que varia desde 2 (pior qualidade) até 97 (melhor qualidade), em função do ajuste do parâmetro de quantização [52].

A documentação deste codificador, referenciada em [52] também indica que, relativamente aos ficheiros de entrada, suporta dados YUV nos formatos planares 4:2:0 e 4:2:2, no formato intercalado 4:2:2 (UYVY) e no formato semi-planar 4:2:0 (NV12). Os ficheiros codificados podem estar num dos formatos planares 4:2:0 ou 4:2:2. Em [53], [54] e [55], para um controlador DM36x, é indicado um desempenho típico de 66 MP/s (megapixéis/segundo) e de 60 fps para a resolução de 720p, se

apenas for executado o codificador ou o descodificador, e um desempenho de 30 fps para a mesma resolução se ambos os codificador e descodificador forem executados simultaneamente. Também é indicado que o melhor desempenho é alcançado quando se utiliza na entrada o formato 4:2:2 (UYVY) e na saída o formato 4:2:0.

Já o descodificador correspondente, referenciado em [56], também implementa o processo sequencial *baseline,* com limitações como o uso das mesmas tabelas de *Huffman* e de quantização para ambos os componentes U e V, e apenas suporta imagens de 1 ou 3 componentes. Note-se que o codificador apenas suporta imagens de 3 componentes, o que significa que não é adequado para imagens só com componentes luminância, ao contrário do descodificador.

O processo de descodificação referido é compatível com imagens comprimidas em diversos formatos YUV 4:2:0, 4:2:2 e 4:4:4, enquanto as imagens que reconstrói podem estar nos formatos NV12 ou UYVY [56].

3.2.2 Codec H.264 da TI

O codificador H.264 da Texas Instruments permite a utilização de qualquer um dos 3 perfis definidos por este padrão, bem como diferentes níveis (até o nível 5 no perfil *high*). Também permite utilizar o filtro de remoção de artefactos (*Deblocking filter),* escolher o método de codificação de entropia (CABAC ou CAVLC), utilizar transformadas de 8x8, entre outros. Por outro lado, não suporta *frames* do tipo B nem permite usar mais que 1 *frame* de referência [57]. Este codificador encontra-se disponível em dois modos:

- Modo compatível com a versão 1.1, o qual oferece desempenhos de até 720p a 30 fps na DM365/DM368, e resolução máxima de 2048x2048;
- Modo Platinum, que oferece desempenhos de até 1080p a 30 fps na DM368, e resolução até 4096x4096, e desempenho de até 720p a 30 fps no DM365 (que opera a 300 MHz) [54].

Estas configurações podem ser realizadas essencialmente através de 4 diferentes estruturas de dados. O Anexo J contém uma tabela que apresenta os campos que constituem as referidas estruturas de dados, bem como uma breve descrição dos mesmos.

3.3 Compressão de imagens com codec JPEG

A análise do desempenho do controlador DM368 na compressão de imagens de acordo com o padrão JPEG foi realizada através de uma aplicação em linguagem C (Anexo K) que obtém uma imagem a partir de um ficheiro, e comprime-a com

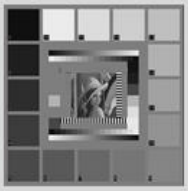
diferentes parâmetros de quantização. Tal aplicação encontra-se descrita na secção 3.3.2.

3.3.1 Seleção dos ficheiros alvo de compressão

Começou-se por definir qual o tipo de dados mais apropriados para os ficheiros de testes. De entre os 4 formatos no espaço de cor YUV que o codec JPEG dispõe, optou-se por um 4:2:2 (por estar associado a melhor qualidade visual que um 4:2:0), nomeadamente o UYVY.

Para os testes a este codec procuraram-se ficheiros com diferentes resoluções e com caraterísticas próprias de determinadas cenas, como aquelas associadas a cenas naturais (com transições de cor suaves) e como aquelas associadas a cenas artificiais (com contornos bem definidos e transições rápidas de cor). Procurou-se escolher imagens tipicamente utilizadas nas áreas de análise e processamento de imagem, como é o caso de “Lena”, “Red” (também conhecido por “Splash”) e “Testpat” [58]. A Tabela 3.2 apresenta uma breve descrição das imagens utilizadas.

Tabela 3.2. Ficheiros usados para a análise do codec JPEG.

Nome	Lena	Red	Testpat	Azores	Builds
Imagem					
Resolução	512x512	512x512	1024x1024	1920x1200	1920x1200
Descrição	Imagem de cor, baixa resolução, mas com detalhes.	Imagem de cor, baixa resolução e pouco detalhe.	Imagem em escala de cinza, com padrões finos e grosseiros, e com Lena (em escala de cinza) no centro.	Imagem de cor com transições suaves. Possui reduzidas componentes nas altas frequências.	Imagem de cor com muitos detalhes, muitas componentes nas altas frequências. Típico de imagens artificiais.

A razão pela qual se utilizou também as restantes duas imagens, que não são tipicamente utilizadas no campo da análise de imagens, prende-se com a resolução das mesmas, pois pretendeu-se também analisar o desempenho do codec para diferentes resoluções.

3.3.2 Aplicação desenvolvida para analisar o codificador JPEG

O funcionamento da aplicação responsável pela gestão dos processos de compressão e descompressão JPEG é apresentado no fluxograma da Figura 3.4. Esta aplicação lê uma imagem UYVY, cujo nome e resolução lhes são indicados, e faz a compressão da mesma com diferentes níveis de quantização. Uma vez que não se tem acesso direto ao níveis de quantização, estes foram alterados através do parâmetro fator de qualidade, (FQ) mais especificamente, para os valores 97 (máximo, $FQ_{máx}$), 82, 72, 62, 52, 42, 32, 22, 12 e 2 (mínimo, FQ_{min}), sendo cada uma destas imagens guardada num ficheiro diferente. As imagens comprimidas voltam a ser descomprimidas de modo a que o EQM possa ser calculado.

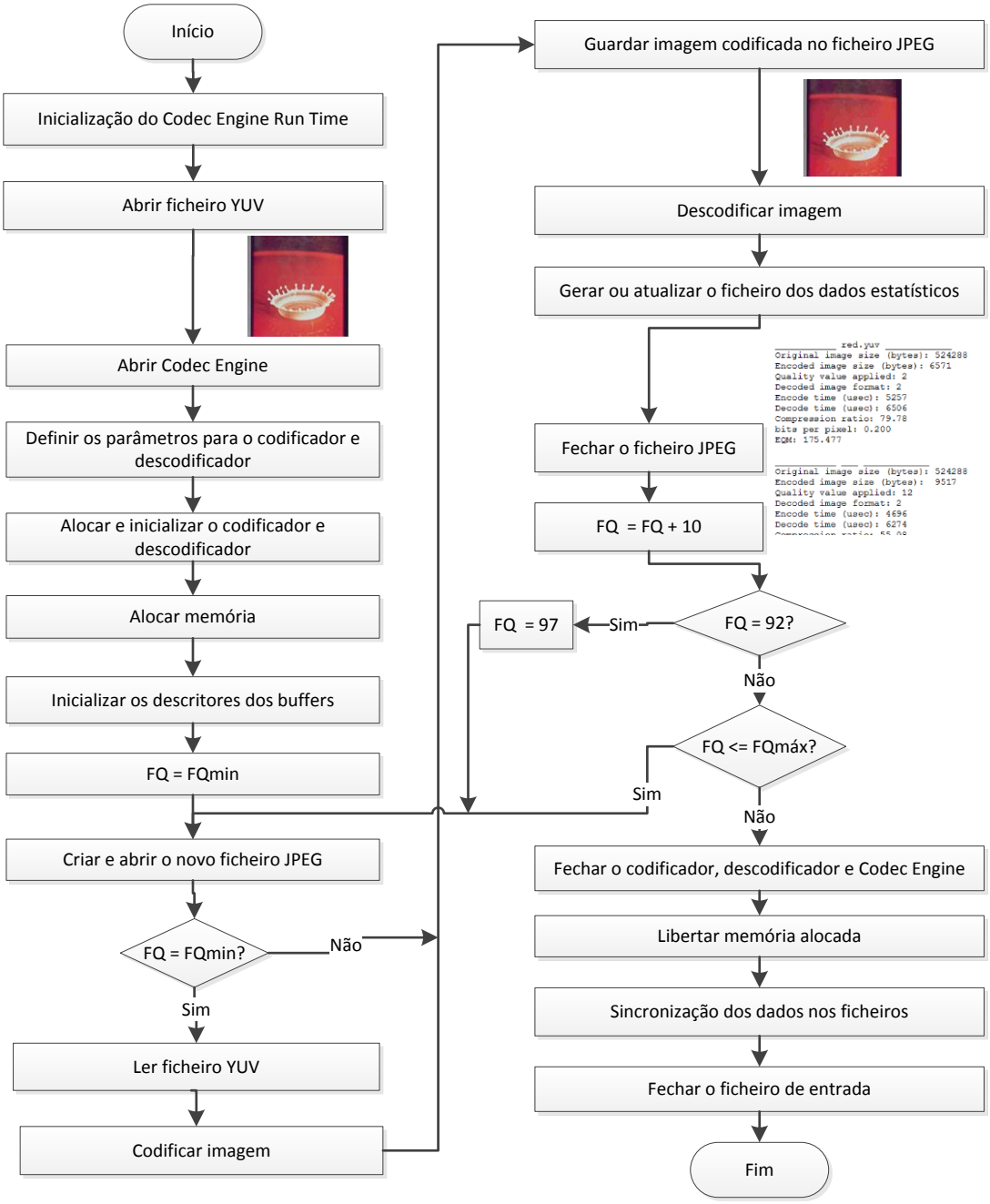


Figura 3.4. Fluxograma da aplicação de compressão e descompressão de imagens com o padrão JPEG.

Para cada imagem analisada, esta aplicação cria um ficheiro de texto onde, para cada fator de qualidade aplicado, regista o tamanho em *bytes* da imagem original e da imagem comprimida, os tempos (em µs) dos processos de compressão e descompressão, o rácio de compressão, o número de *bits*/píxel (N_b) da imagem comprimida e o EQM entre a imagem original e a imagem descomprimida.

O estudo do padrão JPEG foi realizado com base nas referências [3], [59] e [60]. No Anexo E é feita uma descrição deste padrão de compressão.

3.4 Compressão de vídeo com codificador H.264

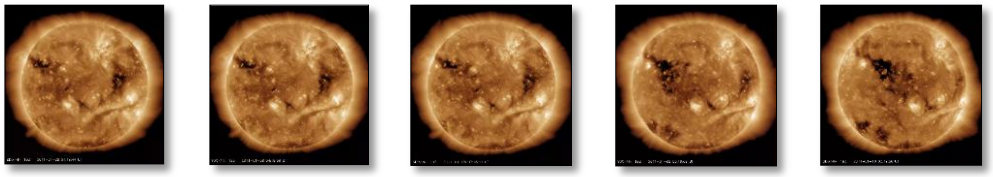
Para analisar o desempenho do controlador DM368 na compressão de vídeo de acordo com o padrão H.264, foi desenvolvida uma aplicação em linguagem C (Anexo K) para ser executada na LeopardBoard. Devido às limitações do sensor utilizado (ver secção 3.3) os vídeos comprimidos nesta aplicação são obtidos a partir de ficheiros.

3.4.1 Seleção dos ficheiros alvo de compressão

Para testar o H.264 procuraram-se ficheiros com diferentes resoluções e com diferentes níveis de complexidade, por exemplo em termos de quantidade de movimento. Relativamente à resolução, a máxima testada foi de 1440x1440. A Tabela 3.3 apresenta uma breve descrição dos ficheiros utilizados, os quais podem ser encontrados no Anexo L.

Tabela 3.3. Descrição dos ficheiros de vídeo utilizados nos testes.

Nome	Resolução	Duração	Descrição
Stefan	CIF (352x288)	300 *frames* (30 fps)	Vídeo com muito movimento e muito detalhe.
Hall	CIF (352x288)	300 *frames* (30 fps)	Câmara parada, em ambiente interior. Cena com pouco movimento. Vídeo típico de videovigilância.
Davinci	720P (1280x720)	100 *frames* (30 fps)	Vídeo computorizado, com fundo de cor constante, a qual muda gradualmente entre *frames*. Cena com pouco movimento.

Nome	Resolução	Duração	Descrição
BigBuckBunny	1920x1088	100 *frames* (30 fps)	Vídeo computorizado, com fundo estático. Tem pouco movimento, mas algum detalhe.
			
SolarCoronalHoles	1440x1440	103 *frames* (30 fps)	Imagem estática, com o objeto a executar movimento lento de rotação.
			

Todos os ficheiros foram utilizados no formato NV12, uma vez que este é o único suportado pelo codificador [57].

3.4.2 Aplicação desenvolvida para analisar o codificador H.264

O estudo do padrão H.264 foi realizado com base nas referências [2], [15], [24], [25], [33], [34], [57], [61]–[63], a partir do qual foi realizado um resumo que pode ser analisado no Anexo F. A partir deste resumo (e como já se referiu na secção 3.2.2) verifica-se que o codificador H.264 permite configurar uma multiplicidade de parâmetros (ver Anexo J). Neste trabalho procurou-se analisar como varia o desempenho da codificação em função de alguns deles, nomeadamente: a utilização do filtro de remoção de artefactos, o nível de quantização, a periodicidade de *frames* do tipo I, o algoritmo de controlo da taxa de *bits,* e a taxa de transmissão de *bits* do canal. Para além disso, procurou-se também avaliar o desempenho de codificador configurado segundo as recomendações sugeridas pela TI em [61], para os seguintes diferentes tipos de aplicações: *broadcasting/streaming*, videovigilância, videoconferência e armazenamento.

O *broadcasting* consiste na transmissão de sinais de áudio e vídeo contendo programas para uma dada audiência. Aplicações *broadcast* podem transmitir em "tempo real", como eventos de desporto, ou podem transmitir dados pré-gravados, como filmes e séries [61]. O *streaming* distingue-se do *broadcast* na medida em que a transmissão ocorre para apenas um cliente, o qual reproduz o conteúdo recebido sem que o armazene no dispositivo recetor [61]. Um servidor de *streaming* distribui

conteúdo como vídeo "*on demand"*, eventos em direto e listas de reprodução de conteúdo pré-gravado.

Uma vez que o conteúdo transmitido nestes tipos de aplicações é variável, pode incluir sequências de vídeo com muitas mudanças de cena (muito movimento, por exemplo). Assim sendo, as configurações recomendadas para estes tipos de aplicações incluem *frames* do tipo I em intervalos curtos, e a utilização da transformada de blocos 8x8 na codificação das *frames* do tipo I.

Quanto às aplicações de videovigilância, são mais exigentes em termos de requisitos quando comparadas com outras aplicações de vídeo. Estas requerem elevada compressão devido às limitações de armazenamento e de largura de banda, baixa complexidade de forma a reduzir os custos, e escalabilidade para servir vários clientes na rede [61]. As aplicações de videovigilância envolvem tipicamente dois *streams,* um para a reprodução da sequência de vídeo em tempo real e outro de alta fidelidade para armazenamento ou análise [61]. No primeiro caso normalmente são usadas baixas taxas de *bits* e são requeridos atrasos reduzidos, enquanto no segundo caso são permitidas elevadas taxas de *bits* e maiores atrasos.

Numa aplicação de videoconferência a comunicação é interativa, permitindo a transmissão simultânea de sinais de áudio e vídeo entre duas ou mais localizações. Difere da videochamada na medida em que é desenvolvido não só para a comunicação entre 2 dispositivos, mas também para servir as necessidades para conferência, que envolve mais que 2 indivíduos [61]. São requeridos atrasos reduzidos, e a taxa de *bits* também é condicionada pelas limitações da rede. A recomendação da TI inclui também a utilização de fatias para evitar os erros de transmissão e para melhor se ajustar aos limites dos pacotes na rede.

Aplicações de armazenamento, como DVR, gravam vídeos no formato digital para um disco ou meio de memória local dentro de um dispositivo como STB e leitores de multimédia portáteis [61]. Uma vez que o conteúdo comprimido destina-se ao armazenamento local, não é necessário efetuar o controlo da taxa de *bits* para que seja constante, e são permitidos elevados atrasos. Quanto se tratam de dispositivos *high-end* como um DVR, o armazenamento é menos limitado que aquele em dispositivos *low-end,* como é o caso de *smartphones,* de modo que a taxa de *bits* máxima alcançável deve ser superior.

Para a realização dos vários testes foram criados "modos de compressão", nos quais os diferentes parâmetros são configurados de forma específica, conforme as caraterísticas de codificação desejadas para cada teste. Os diferentes modos de compressão encontram-se descritos no Anexo L.

O fluxograma da Figura 3.5 apresenta o funcionamento da aplicação em linguagem C desenvolvida para analisar o desempenho do padrão H.264 no controlador selecionado. Esta aplicação começa por abrir um ficheiro cujo nome e dimensões (incluindo o número total de *frames*, *NumFrames*) lhes são indicados, e a compressão é realizada de acordo com os parâmetros do modo de compressão selecionado pelo utilizador.

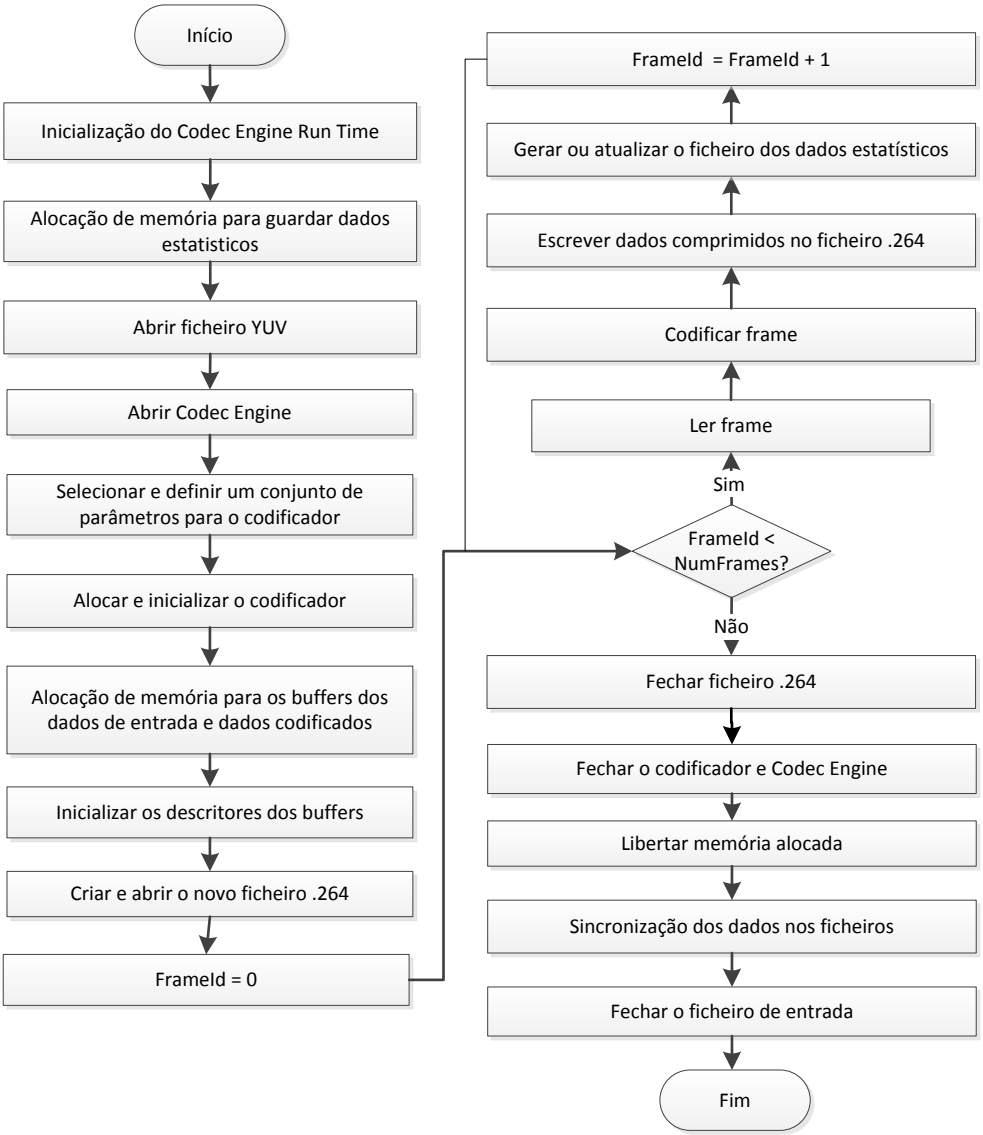


Figura 3.5. Fluxograma descritivo da aplicação desenvolvida para analisar o codificador H.264.

Para cada vídeo, esta aplicação cria um ficheiro de texto onde regista os diferentes modos com que o mesmo vídeo foi comprimido durante as diversas vezes que a aplicação foi corrida. Para cada modo com que o vídeo foi comprimido, são registadas também algumas caraterísticas desse modo, bem como o tempo que demorou a comprimir todo o vídeo, o tempo dedicado a cada *frame* e o tamanho de cada *frame* codificada.

O Matlab [64] foi utilizado para calcular o EQM entre um dado vídeo original e cada versão comprimida do mesmo. O ficheiro original e uma versão descomprimida são abertos, o EQM entre cada *frame* é calculado e o valor é guardado num ficheiro de texto. O código desenvolvido pode ser analisado no Anexo K.

Os vídeos foram descomprimidos recorrendo à ferramenta avconv [65], e o comando utilizado para tal poderá ser

$$avconv\ –i\ video.264\ –pix\_fmt\ nv12\ video.yuv\ .$$

3.5 Algoritmos para processamento da imagem antes da compressão

Uma imagem pode ser modificada através de uma multiplicidade de algoritmos. Alguns deles procuram melhorar a imagem de modo a torná-la mais natural ou ajustá-la para uma aplicação específica, enquanto outras procuram adicionar efeitos especiais (como efeito sépia, efeito negativo, entre outros).

Foram selecionados algoritmos com poucos requisitos de computação, ou seja, cuja tempo de execução seja relativamente curto para ser possível a implementação em aplicações de tempo-real, e possam ser utilizados para melhorar as imagens. Tais algoritmos são descritos de seguida.

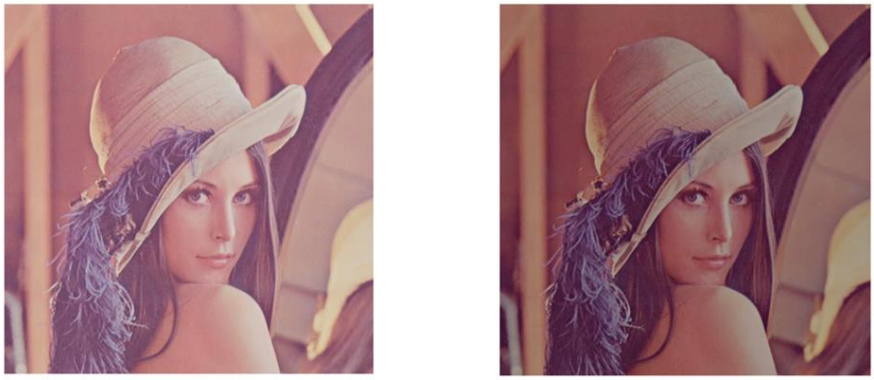
3.5.1 Algoritmos implementados

Foram analisados e implementados um total de quatro algoritmos. O algoritmo para alteração do contraste baseia-se na técnica de “constraste automático” referida na secção 2.1.4.2**,** contudo, ao invés de ser aplicada a gama de níveis de brilho do histograma original, é utilizada outra gama, calculada da seguinte forma:

- O histograma é calculado;
- O número de píxeis em cada nível de brilho é somado, desde a extremidade inferior do histograma;
- O “limite inferior” é determinado pelo nível onde o somatório referido atinge um dado valor determinado previamente;
- O “limite superior” é determinado da mesma forma, sendo que o somatório inicia-se na extremidade superior do histograma;
- A gama a ser utilizada como g na equação (2.10) é dada pela subtração do “limite inferior” ao “limite superior”.

A gama para o novo histograma é indicada explicitamente, sendo que o valor utilizado para g_n é 200. Foi utilizado um valor inferior ao máximo possível (255) por se ter verificado que, para algumas imagens, tal resultava num aumento excessivo

do contraste. A aplicação deste algoritmo à imagem da Figura 3.6 (a), resulta na imagem da Figura 3.6 (b).



(a) Original (b) Com alteração do contraste

Figura 3.6. Alteração do contraste de uma imagem.

O algoritmo mais simples de entre os quatro implementados é o que permite alterar a luminosidade, pois consiste apenas em adicionar um *offset* à componente luma da imagem [7]. No algoritmo implementado utilizou-se um *offset* constante de 50, que corresponde a aproximadamente 20% da maior gama dinâmica possível de imagens com profundidade de 8 *bits*. A título de exemplo, pode-se observar a Figura 3.7, em (a) é a imagem original, e (b) a imagem obtida após a aplicação deste algoritmo.

(a) Original (b) Com alteração da luminosidade

Figura 3.7. Alteração da luminosidade de uma imagem.

Também foi implementado um algoritmo para fazer a correção do balanço de brancos (tratado na secção 2.1.5), nomeadamente o algoritmo G*ray World,* descrito no Anexo A. Contudo, a imagem de entrada considerada nesta descrição encontra-se no espaço de cor RGB. Uma vez que esta aplicação recebe a imagem de entrada no formato UYVY, foi necessário adicionar o processo de conversão para o espaço de cor RGB antes da aplicação do algoritmo *Gray world,* e o processo de conversão

inverso, para o espaço de cor YUV, após a aplicação do algoritmo referido. Para tal, foram utilizadas as equações (2.5) e (2.6) apresentadas na secção 2.1.3.

Este procedimento, adicional ao algoritmo original descrito no Anexo A, contribui para o aumento no grau de complexidade do mesmo, de modo que é esperado que este seja o algoritmo mais pesado computacionalmente de entre os quatro analisados. Para ilustrar o tipo de alteração realizado nas imagens, observe-se o exemplo da Figura 3.8.

(a) Original (b) Com correção do balanço de brancos

Figura 3.8. Aplicação do algoritmo *Gray World* numa imagem.

O outro algoritmo implementado é adequado para fazer a equalização do histograma, ou seja, procura uniformizar a distribuição de intensidades de imagens, que tenham como finalidade, por exemplo, serem impressas ou para efeitos de comparação entre elas [7]. A imagem é modificada de forma a que o histograma original, $h(i)$ na Figura 3.9 (a), se torne aproximadamente linear, como $h_{eq}(i)$, na Figura 3.9 (b) [7]. Para tal, primeiramente é necessário obter o histograma cumulativo, $H(i)$ na Figura 3.9 (c), e de seguida obter os novos valores, $f_{eq}(i)$, a partir da equação

$$f_{eq}(i) = \left\lfloor H(i) \cdot \frac{K-1}{MN} \right\rfloor, \tag{3.1}$$

onde $M \times N$ é o tamanho da imagem em píxeis, os quais variam na gama $[0, K-1]$.

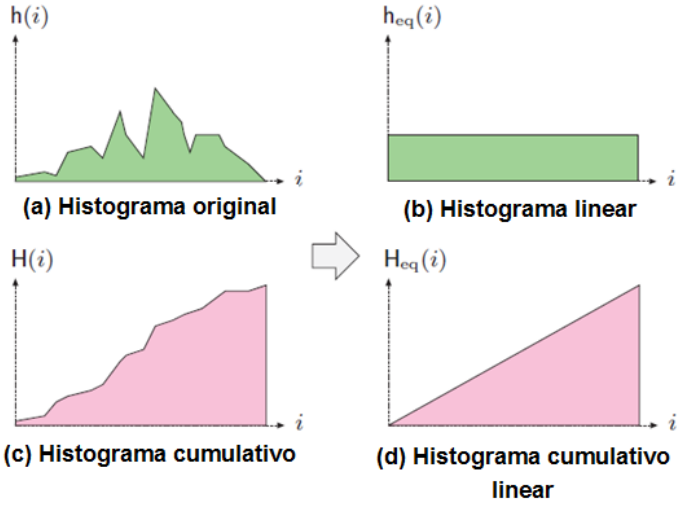


Figura 3.9. Equalização de histograma [7].

O histograma cumulativo da imagem resultante aproxima-se a $H_{eq}(i)$ na Figura 3.9 (d). A imagem da Figura 3.10 (b) ilustra o resultado após a equalização do histograma da imagem da Figura 3.10 (a).

(a) Original (b) Com equalização do histograma

Figura 3.10. Aplicação do algoritmo para equalização do histograma numa imagem.

Estes algoritmos foram integrados numa aplicação que permite utilizá-los para processar uma dada imagem antes de comprimi-la com o codificador JPEG, a qual é seguidamente apresentada.

3.5.2 Aplicação desenvolvida para analisar o desempenho dos algoritmos de processamento de imagem

A aplicação utilizada para analisar o desempenho dos algoritmos de processamento de imagem encontra-se descrita no fluxograma da Figura 3.11 enquanto o respetivo código pode ser analisado no Anexo K. Esta aplicação, desenvolvida em linguagem C, que é para ser executada no controlador DM368, obtém uma imagem a partir de ficheiro, permite a modificação da mesma através dos diferentes algoritmos quantas vezes se pretender, comprime a imagem

resultante com o codificador JPEG e guarda-a num ficheiro. Uma vez que se pretende não só analisar o desempenho dos referidos algoritmos de processamento de imagem, mas também verificar se a sua implementação afeta o processo de compressão JPEG, a imagem volta a ser descomprimida, e o EQM em relação à imagem de entrada é calculado e guardado num ficheiro de texto. Para além do EQM, são recolhidos e registados os seguintes dados: algoritmos implementados, tempo de execução de cada algoritmo implementado e rácio de compressão da imagem.

A análise foi realizada com imagens porque a análise e os testes realizados com imagens são menos morosos do que com vídeos. Contudo, estes algoritmos também podem ser utilizados para processamento de *frames* numa sequência de vídeo.

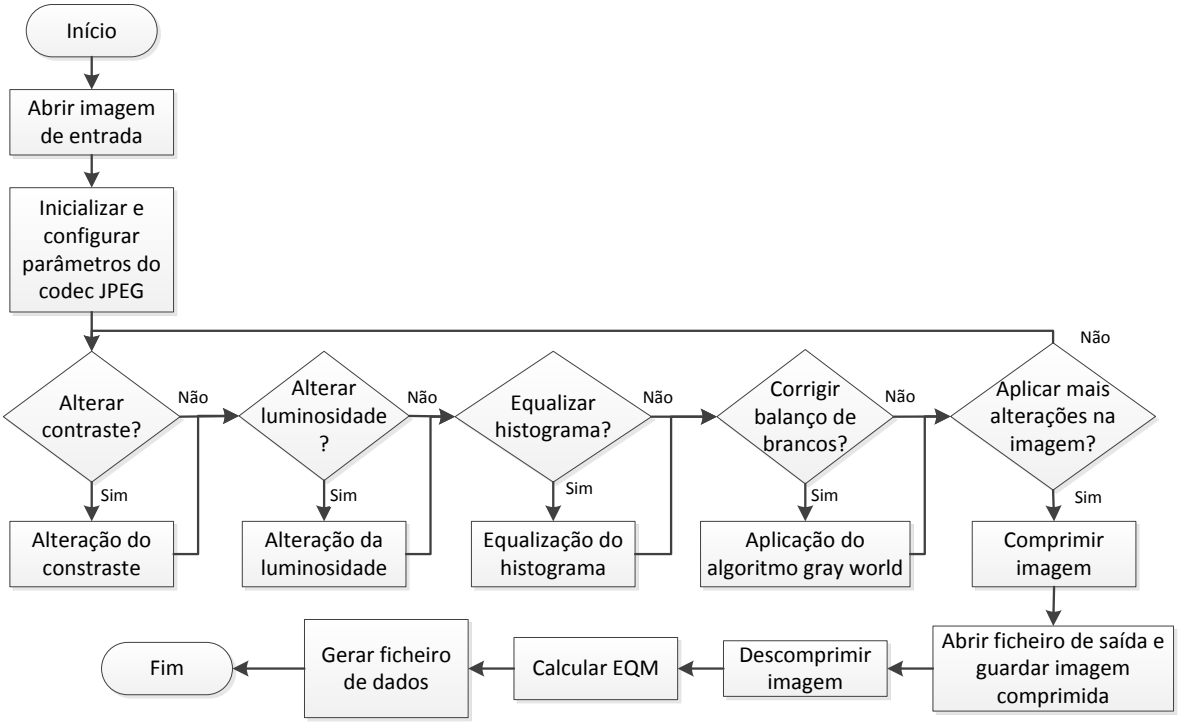


Figura 3.11. Fluxograma representativo da aplicação desenvolvida para análise do desempenho dos algoritmos de processamento de imagem.

As imagens utilizadas como entrada são imagens tipicamente utilizadas nas áreas de análise e processamento de imagem, obtidas em [66], e posteriormente modificadas para irem de encontro às alterações implementadas pelos algoritmos. Tais imagens, de resolução CIF (à exceção da imagem (b), com resolução 256x256), apresentam-se no Anexo M.

Neste capítulo foram apresentadas as aplicações desenvolvidas para a análise do desempenho de compressão JPEG e H.264. Os dados recolhidos durante a execução das mesmas são apresentados e analisados no capítulo 4.

4. Avaliação do desempenho da compressão de imagem e vídeo num controlador ARM

O desempenho de compressão com os codificadores JPEG e H.264 foi analisado com base nas imagens e vídeos obtidos, mas principalmente com base nos dados registados nos ficheiros de texto gerados pelas aplicações. Tais dados foram tratados, e nesta secção é feita a análise dos mesmos.

4.1. Desempenho da compressão JPEG

Os dados recolhidos ao longo da execução da aplicação referentes a cada uma das imagens (tempo de compressão, rácio de compressão, EQM e N_b) podem ser analisados na Tabela N.1 no Anexo N. Nesta secção tais dados são apresentados de forma gráfica no sentido de permitir uma melhor interpretação dos mesmos. Quanto às imagens resultantes dos diversos testes encontram-se no Anexo N.

O desempenho do codificador JPEG foi avaliado segundo os seguintes parâmetros: taxa de *frames*, EQM, N_b, rácio de compressão e qualidade visual. Para se calcular a taxa de *frames* registou-se o tempo tomado pelo controlador para fazer a operação de compressão. Já o EQM, o N_b, o rácio de compressão e a qualidade visual foram medidos pelas respetivas formas descritas na secção 2.2.4. Para se medir a qualidade visual, as imagens comprimidas foram apresentadas a um conjunto de 21 pessoas, as quais foram solicitadas a fazer uma avaliação de acordo com a Tabela 2.2 na secção 2.2.4**.** Nestes testes os utilizadores nunca foram confrontados com a imagem original, nem lhes foi indicado qual a imagem comprimida com menor nível de quantização. Começou-se por apresentar as duas imagens com menor nível de quantização, seguindo-se pelas imagens comprimidas com maiores níveis de quantização de forma aleatória, terminando sempre com a imagem comprimida com maior nível de quantização. Antes de se iniciar o teste foi ainda apresentado a cada utilizador os tipos de distorção comumente encontrados em imagens comprimidas com o JPEG, e que portanto, poderiam ser encontrados nas imagens analisadas, tais como blocagem (Figura 2.3), *ringing* e cor esbatida (ver Anexo C).

Os dados recolhidos foram tratados calculando o MOS, de acordo com a equação (2.13) na secção 2.2.4, tendo sido realizado também o tratamento dos mesmos através de uma regressão linear, cujos resultados podem ser analisados no Anexo N. Na Figura 4.1 pode-se observar o diagrama de dispersão com os valores MOS calculados a partir das classificações atribuídas pelos utilizadores às imagens

comprimidas com diferentes níveis de quantização, relativamente à qualidade das mesmas. Já os valores MOS na Figura 4.2 também foram calculados a partir das classificações atribuídas pelos utilizadores às imagens comprimidas com diferentes níveis de quantização, mas em relação à distorção das imagens.

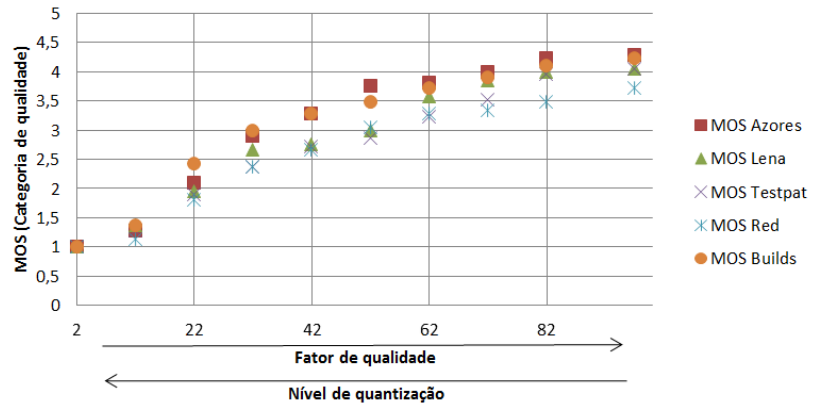


Figura 4.1. Classificação da qualidade das imagens pelo método MOS, em função do nível de quantização.

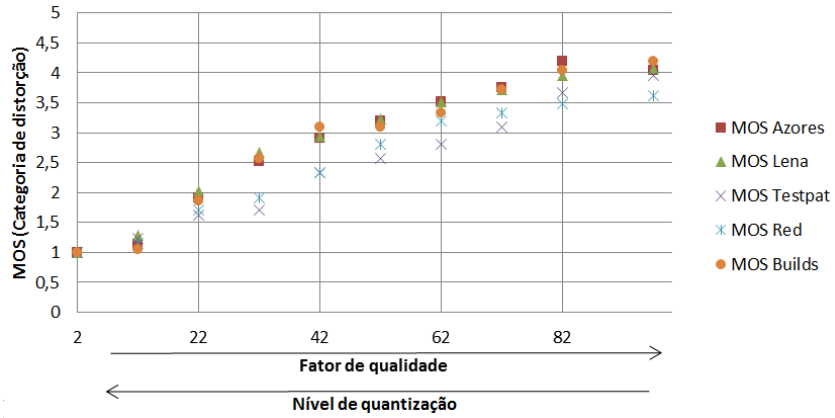


Figura 4.2. Classificação da distorção das imagens pelo método MOS, em função do nível de quantização.

É possível verificar que os gráficos da Figura 4.1 e da Figura 4.2 estão de acordo com o indicado na documentação do codificador [52], uma vez que a qualidade média percebida pelos utilizadores aumenta de acordo com o aumento no fator de qualidade. Através do método de predição inversa (Anexo N) procurou-se fazer um mapeamento entre o nível de qualidade das imagens, e o nível de quantização com que foram comprimidas. Uma vez que no codificador não se tem acesso direto ao nível de quantização, este foi alterado através do parâmetro fator de qualidade. A Tabela 4.1 apresenta o mapeamento obtido através do referido método de predição inversa.

Tabela 4.1. Classificação da qualidade e distorção das imagens analisadas de acordo com o método MOS, em função do fator de qualidade utilizado na compressão.

Fator de qualidade	Qualidade
[2,14[	Má
]2,44[	Pobre
]30,74[	Razoável
]60,97]	Boa
]90,97]	Excelente

Fator de qualidade	Distorção
[2,17[	Muito perturbante
]12,47[	Perturbante
]42,76[	Ligeiramente perturbante
]71,97]	Percetível mas pouco perturbante

Verifica-se que existe sobreposição de intervalos de categorias consecutivas. Isso significa que para os fatores de qualidade que estão incluídos em categorias MOS diferentes é difícil prever qual a categoria das imagens que são codificadas com esses fatores de qualidade. Tal é mais saliente para os fatores extremos, nomeadamente para a categoria MOS para a qualidade, que revela a dificuldade em distinguir imagens de qualidade Má das imagens de qualidade Pobre, e em distinguir imagens de qualidade Excelente das imagens de qualidade Boa.

Na Figura 4.3 são apresentadas 4 imagens resultantes da compressão da imagem “Lena” com diferentes níveis de quantização, cada uma referente a uma diferente categoria a nível de distorção e qualidade segundo as classificações na Tabela 4.1.

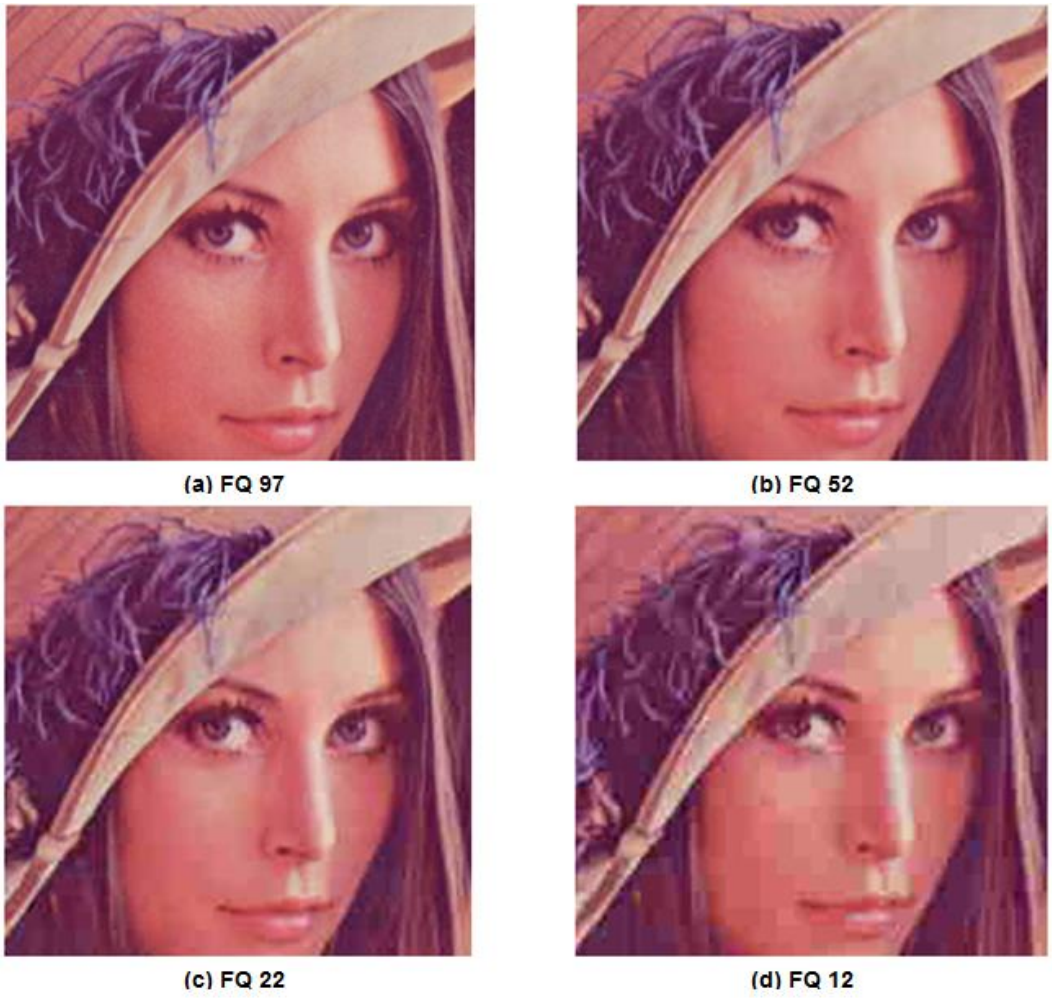


Figura 4.3. Parte da imagem "Lena" comprimida com diferentes fatores de qualidade.

Procurou-se também fazer uma avaliação com base no número de *bits*/píxel (N_b), tal como é indicado na secção 2.2.4. No sentido de se poder comparar a avaliação sugerida na Tabela 2.3 através do valor de N_b, com a avaliação obtida através do método MOS, acrescentou-se a classificação "Má qualidade" à Tabela 2.3 para se referir às imagens com menos de 0,25 *bits*/píxel, e numerou-se cada uma das categorias desta tabela desde 1 até 5, tal como apresenta a Tabela 4.2. Após atribuir as categorias às imagens de acordo com o N_b das mesmas, foi possível obter a relação entre a classificação de acordo com o valor de N_b e a classificação de acordo com o método MOS. Tal relação encontra-se representada graficamente na Figura 4.4.

Tabela 4.2. Nova classificação da qualidade de imagem em função da taxa.

Categoria	N_b (*bits*/píxel)	Qualidade da imagem
1	< 0,25	Má qualidade
2	0,25 – 0,5	Moderada a boa qualidade
3	0,5 – 0,75	Boa a muito boa qualidade
4	0,75 – 1,5	Excelente qualidade
5	1,5 – 2,0	Normalmente indistinguível da original

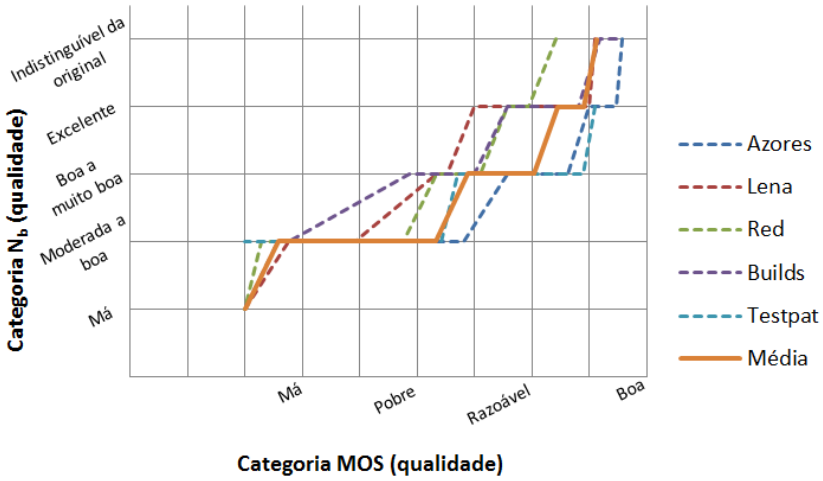


Figura 4.4. Qualidade das imagens segundo a classificação MOS em confronto com a classificação N_b.

Como se pode ver na Figura 4.4, a qualidade das imagens é sobrevalorizada pela classificação com base no valor de N_b, relativamente à classificação com base no método MOS, o que significa que, para grande parte dos utilizadores questionados, a classificação com base no valor de N_b não reflete adequadamente a qualidade da imagem. De qualquer forma, este gráfico permite constatar o facto de imagens de melhor qualidade estão associadas a um maior valor de N_b, de modo que este método de avaliação objetivo deve efetivamente ser utilizado para classificação da qualidade relativa entre imagens.

Na Figura 4.5 encontra-se a representação gráfica da relação entre o rácio de compressão e a qualidade medida através do método MOS. Este gráfico mostra que, em média, é possível alcançarem-se rácios de até cerca de 95:1, contudo a qualidade das imagens com este rácio é má. Se reduzir o rácio para cerca 45:1, a qualidade das imagens obtidas melhora ligeiramente, situando-se na categoria de qualidade pobre. Já imagens com qualidade razoável implicam reduzir o rácio para cerca de 34:1 ou menos, enquanto as imagens de boa qualidade estão associadas a rácios cujo valor máximo é de cerca 22:1. Tais resultados podem ser equiparados aos apresentados em [35] e [67].

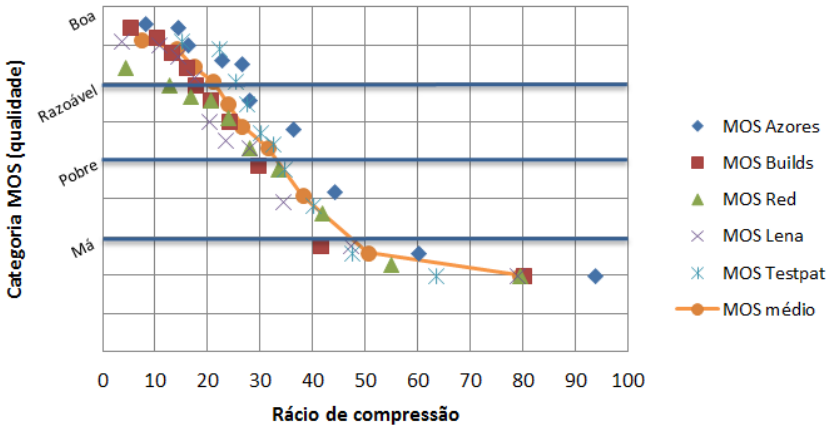


Figura 4.5. Qualidade das imagens segundo o método MOS, em função do rácio de compressão.

Pode-se verificar também que, para um dado rácio de compressão, geralmente as imagens apresentam diferentes valores MOS, e nalguns casos até se situam em categorias diferentes. A imagem "Azores", por exemplo, identifica-se com a imagem na Figura 4.6 (b), a qual não é o caso ideal em que não há qualquer variação de cor, como na imagem da Figura 4.6 (a), mas também não é caso da imagem na Figura 4.6 (c), que é maioritariamente constituído por elevadas componentes de alta frequência. O processo de compressão e descompressão JPEG dos blocos 8x8 na Figura 4.6 foi realizado através do Matlab, e é apresentado passo a passo no Anexo O.

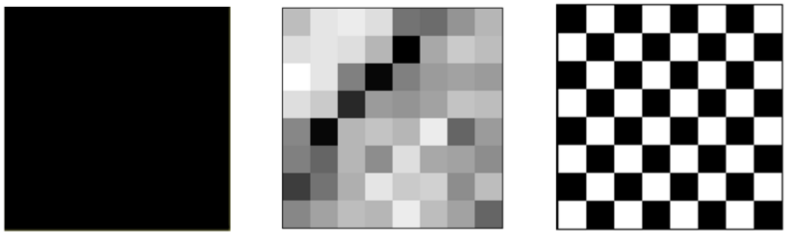


Figura 4.6. Blocos 8x8 provenientes de diferentes zonas da imagem "Testpat".

Apesar da imagem "Azores" não apresentar os maiores valores de PSNR para todos os diferentes rácios de compressão (como se pode ver na Figura 4.7), o facto de toda ela ser constituída por variações suaves de cor (ou seja, reduzidas componentes de alta frequência), as alterações provocadas na cor não são tão intensas como acontece em imagens com muitas componentes de alta frequência (como acontece no exemplo da imagem na Figura 4.6 (c), em que, como se pode ver no Anexo O, são menos os pixéis afetados, contudo a diferença relativamente aos pixéis originais é maior), de modo que a qualidade percebida pelo sistema visual humano é maior.

A imagem “Testpat”, apresenta qualquer um dos padrões tidos como referência na Figura 4.6, uma vez que tais exemplos foram desenvolvidos justamente a partir de diferentes zonas desta imagem. Como tal, as razões que levam a que esta imagem apresente rácios de compressão superiores à média para as duas melhores categorias MOS prendem-se com o facto de grande parte da imagem ser constituída por zonas de cor constante (como na imagem da Figura 4.6 (a)) e por zonas com reduzidas componentes de alta frequência (como na imagem da Figura 4.6 (b)). O nível de qualidade é afetado, porém, pelas componentes de alta frequência nos padrões em torno da imagem “Lena” no centro, nos quais a distorção rapidamente se torna perturbante (Figura 4.3).

No caso da imagem “Builds”, esta apresenta qualidade MOS superior à média (Figura 4.1), mas em contrapartida, o facto de ser maioritariamente constituída por componentes de alta frequência, provoca uma redução no rácio de compressão, tal como ilustra o exemplo da imagem na Figura 4.6 (c), razão pela qual apresenta menores rácios para a grande parte dos valores MOS e para grande parte dos valores de PSNR, como se pode ver pela Figura 4.5 e pela Figura 4.8.

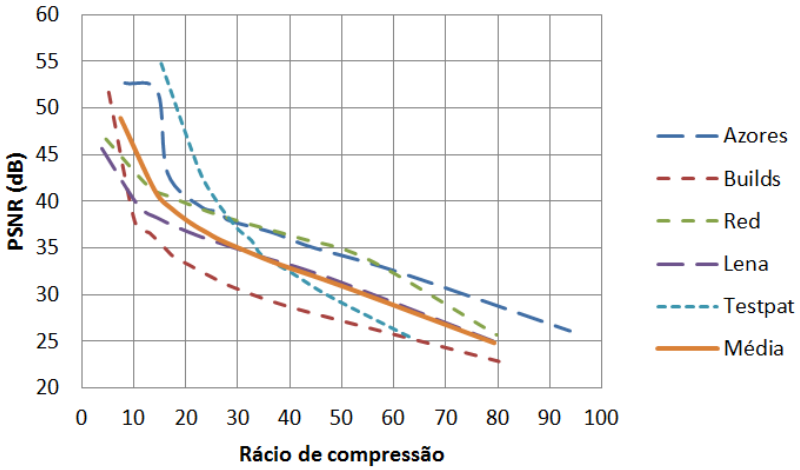


Figura 4.7. PSNR das imagens analisadas em função do rácio de compressão.

Na Figura 4.8 é possível observar como varia o PSNR para as diferentes categorias MOS para a qualidade. Pela observação da Figura 4.8 verifica-se que imagens classificadas na categoria “Má” estão associadas a valores PSNR abaixo de cerca 32 dB, imagens classificadas na categoria “Pobre” estão associadas a valores de PSNR abaixo de cerca 34 dB, imagens classificadas na categoria “Razoável” estão associadas a valores de PSNR abaixo de cerca 37 dB, e valores de PSNR superiores a cerca 37 dB estão associados às imagens cuja qualidade foi classificada como “Boa”.

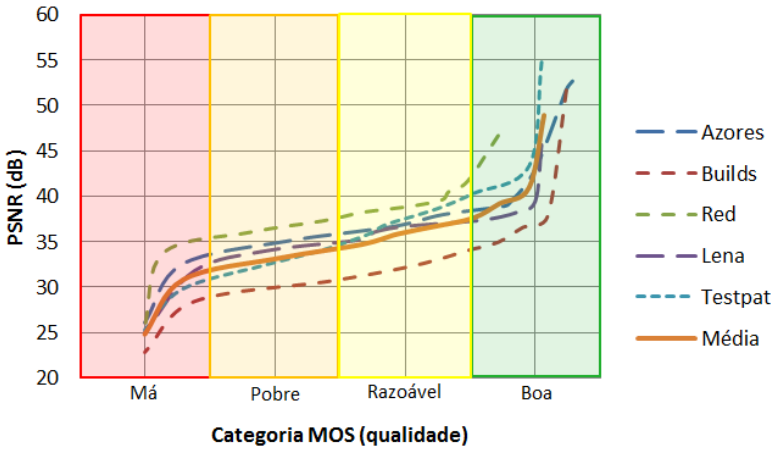


Figura 4.8. PSNR das imagens analisadas em função da qualidade das mesmas.

Relativamente às medições dos tempos de compressão, vieram mostrar que o tempo tomado apenas pelo processo de compressão é aproximadamente constante independentemente do nível de quantização utilizado, como se pode constatar pela observação do gráfico na Figura 4.9. No entanto varia significativamente de acordo com a resolução das imagens comprimidas. Na Tabela 4.3 são apresentados os tempos médios e as velocidades médias de compressão em fps e MP/segundo medidos para as diferentes resoluções correspondentes às imagens utilizadas.

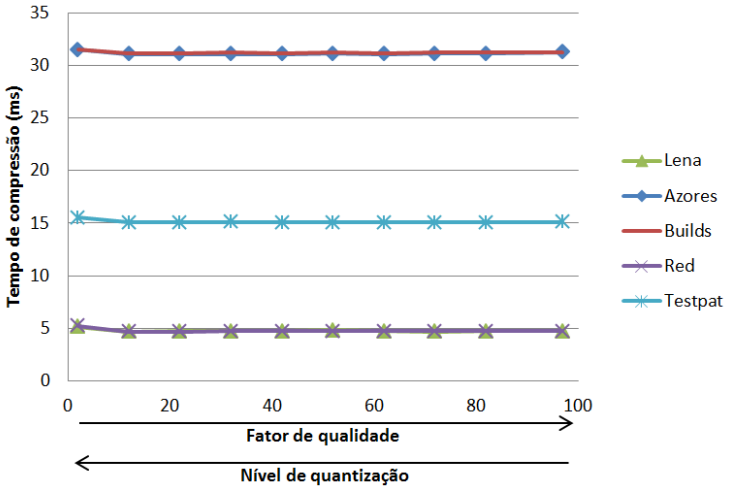


Figura 4.9. Tempo de compressão das imagens analisadas em função do nível de quantização.

Tabela 4.3. Velocidade de compressão média para diferentes resoluções.

Resolução	Tempo médio de compressão (ms)	Taxa de *frames* (fps)	Taxa de píxeis (MP/segundo)
1280x720	31,18	32,07	29,56
1024x1024	15,13	66,08	69,29
512x512	4,77	209,78	54,99

Quanto ao tempo de compressão, obteve-se um valor médio de cerca 31 ms para a resolução de 1280x720, o qual é ligeiramente inferior ao valor típico para um controlador DM365 de 300 MHz (ver secção 3.2.1), tal como se esperava. Ainda para esta resolução, a taxa de *frames* registada é de 32 fps, ligeiramente superior à de 30 fps indicada como típica (ver secção 3.2.1). Bons resultados também foram obtidos para a taxa de píxeis, uma vez que foi atingido um valor de 69 MP/segundo, também superior à taxa de 66 MP/segundo indicada em [54] como sendo o melhor desempenho para um controlador DM365 de 300 MHz.

4.2. Desempenho da compressão H.264

O desempenho da compressão de vídeo H.264 foi avaliado recorrendo a algumas das métricas mencionadas na secção 2.2.4, nomeadamente o PSNR e o rácio de compressão, como também à velocidade de compressão em *frames* por segundo (*fps*), calculada pela equação

$$Taxa\ de\ frames\ (fps) = \frac{N\acute{u}mero\ de\ frames\ codificadas}{Tempo\ de\ codifica\c{c}\tilde{a}o\ (segundos)}, \tag{4.1}$$

e à taxa de *bits* (em *bits* por segundo) para uma dada velocidade de reprodução, calculada pela equação

$$Taxa\ de\ bits\ (bps) = \\ = \frac{Tamanho\ do\ ficheiro\ codificado\ (bits) \times Velocidade\ de\ reprodução\ (fps)}{N\acute{u}mero\ de\ frames}. \tag{4.2}$$

De entre os modos recomendados pela TI para aplicações de *broadcasting, streaming,* videoconferência, videovigilância e armazenamento, a nível do rácio de compressão, os maiores valores foram obtidos para os vídeos comprimidos com os modos Videoconferencia, Videovigilancia_dp e Armazenamento_lE, correspondentes aos recomendados para aplicações de videoconferência, videovigilância para reprodução em tempo-real, e armazenamento em dispositivos *low-end,* assim como se pode constatar pela Tabela 4.4. Tais rácios de compressão resultaram, por um lado, da utilização de um reduzido número de *frames* do tipo I (são completamente intracodificadas, envolvendo um maior número de *bits*), e por outro, da imposição de uma taxa de transmissão de *bits* de canal reduzida, tal como é indicado na descrição do controlo da taxa de *bits* no Anexo F, e como se pode constatar pela análise da Tabela 4.5, a qual apresenta os resultados obtidos para os vídeos comprimidos nos modos intraF0, intraF15 e intraF30.

Tabela 4.4. Rácios de compressão dos vídeos analisados resultantes da codificação em diferentes modos.

Modo	Hall	Stefan	Davinci	Big Buck Bunny	Solar Coronal Holes
Videoconferencia	71:1	71:1	142:1	93:1	243:1
Broadcast	36:1	36:1	149:1	46:1	105:1
Armazenamento_hE	36:1	36:1	149:1	27:1	51:1
Armazenamento_lE	70:1	70:1	83:1	72:1	177:1
Videovigilancia _dp	71:1	71:1	153:1	92:1	183:1
Videovigilancia _st	36:1	36:1	169:1	46:1	105:1

Tabela 4.5. Resultados obtidos para os vídeos comprimidos com diferentes periodicidades de *frames* do tipo I.

Vídeo	Modo	Rácio de compressão	Velocidade de compressão (fps)	PSNR médio (dB)
Stefan	intraF30	17,97:1	349,71	35,26
	intraF15	17,79:1	354,52	35,04
	intraF0	17,01:1	342,59	28,96
Hall	intraF30	18,09:1	356,94	40,47
	intraF15	17,90:1	356,02	40,41
	intraF0	17,25:1	342,19	37,56
Davinci	intraF30	154,89:1	68,36	60,24
	intraF15	51,25:1	68,11	60,19
	intraF0	53,33:1	60,22	57,34

Foi calculada a diferença entre a taxa de transmissão de *bits* de canal definida e a obtida para cada vídeo comprimido nos diferentes modos. Os valores obtidos apresentam-se de forma percentual na Tabela 4.6, bem como é apresentada a diferença máxima (também de forma percentual) permitida, a qual é especificada pelo tamanho do *buffer* VBV (*Video Buffering Verifier*).

Pela observação da Tabela 4.6 verifica-se que os vídeos comprimidos nos modos Armazenamento_lE e Armazenamento_hE, associados ao algoritmo de controlo da taxa de *bits* VBR (*Variable Bit Rate),* são os que apresentam os maiores desvios. Pode-se constatar, portanto, que a utilização do algoritmo CBR (*Constant Bit Rate)* permite um controlo da taxa de *bits* efetivamente mais rigoroso que o algoritmo VBR, tal como indica a análise teórica no Anexo F, pois conduziu às taxas de *bits* médias mais próximas das de transmissão de *bits* de canal definidas, e dentro do limite especificado pelo tamanho do *buffer* VBV, tal como é requerido para aplicações com limitações na largura de banda.

Tabela 4.6. Desvio entre as taxas de *bits* obtidas relativamente às máximas definidas.

Modo	Desvio em relação à taxa de transmissão de *bits* de canal (%)					
	Máximo	Vídeo				
		Hall	Stefan	Davinci	Big Buck Bunny	Solar Coronal Holes
Videoconferencia	50	0,57	0,61	-22,39	1,21	2,27
Broadcast	600	0,55	0,66	-68,12	2,37	1,32
Armazenamento_hE	600	1,32	1,66	-84,11	26,18	4,53
Armazenamento_lE	600	2,12	1,95	-0,57	30	5,63
Videovigilancia _dp	50	0,59	0,59	-45,93	1,77	1,75
Videovigilancia_st	500	0,65	0,61	-72,03	2,59	1,39

A taxa de *bits* máxima permitida e o tamanho do *buffer* VBV devem portanto ser ajustados de acordo com a largura de banda disponível, tendo sempre em conta que isso afeta diretamente o rácio de compressão como também o PSNR, pois menores taxas conduzem à utilização de maiores parâmetros de quantização, e consequentemente, a maiores rácios de compressão e a maior distorção. Tal pode ser constatado, por exemplo, através dos resultados na Tabela 4.4 e na Tabela 4.7 para os modos Armazenamento_lE e Armazenamento_hE. Note-se ainda que uma taxa de *bits* máxima muito reduzida combinada com o algoritmo de controlo de *bits* CBR pode resultar na perda de algumas *frames* [61].

Tabela 4.7. Valores médios para o PSNR dos vídeos codificados nos diferentes modos.

Modo	PSNR (dB)				
	Hall	Stefan	Davinci	Big Buck Buney	Solar Coronal Holes
Videoconferencia	30,94	28,23	29,77	29,7	30,84
Broadcast	31,08	29,87	29,79	30,84	31,15
Armazenamento_hE	31,12	30,22	29,79	31,31	31,19
Armazenamento_lE	30,96	28,19	29,79	30,32	31,03
Videovigilancia _dp	30,8	27,81	29,79	30,03	31,08
Videovigilancia_st	30,78	29,76	29,79	30,78	31,11

Uma outra forma de controlar a taxa de *bits* consiste em limitar a gama de valores que os algoritmos de controlo da taxa de *bits* podem tomar para o passo de quantização [61]. Maiores passos de quantização conduzem a maiores rácios de compressão, e assim, a menores taxas de *bits*, tal como se pode verificar pelos resultados na Tabela 4.8, a qual apresenta os resultados obtidos para os vídeos comprimidos com PQ fixo.

Tabela 4.8. Resultados obtidos para os vídeos comprimidos com PQ fixo.

Vídeo	Modo	Rácio de compressão	Velocidade de compressão (fps)	PSNR médio (dB)
Stefan	PQ0	2:1	216,03	53,45
	PQ8	4:1	277,87	47,08
	PQ18	9:1	348,79	39,88
	PQ28	28:1	370,12	33,19
	PQ38	98:1	371,59	26,96
	PQ51	295:1	372,25	20,06
Hall	PQ0	3:1	233,32	53,55
	PQ8	5:1	316,56	47,10
	PQ18	18:1	360,45	40,72
	PQ28	108:1	372,63	36,14
	PQ38	446:1	363,94	30,58
	PQ51	1640:1	363,85	23,46
Davinci	PQ0	70:1	68,38	65,51
	PQ8	149:1	68,69	60,75
	PQ18	327:1	68,64	55,38
	PQ28	634:1	68,72	47,76
	PQ38	961:1	68,66	41,76
	PQ51	919:1	68,62	32,80

As menores velocidades de compressão foram obtidas para os vídeos comprimidos com os modos Videovigilancia_dp e Videovigilância_st, correspondentes aos recomendados para aplicações de videovigilância, tal como mostra a Tabela 4.9. A razão para tal deve-se ao facto de, para estes modos de compressão, o codificador estar configurado no modo compatível com a versão 1.1 (ver secção 3.2.2). Já as velocidades obtidas para os outros modos apresentados na Tabela 4.9 aproximam-se dos valores típicos referidos na secção 3.2.2 (1080p@30fps para o controlador DM368, e 720p@30 fps para o DM365 a operar a 300 MHz), o que foi possível, principalmente, devido ao uso do codificador no modo Platinum.

Tabela 4.9. Velocidade de compressão dos vídeos para diferentes modos.

Modo	Velocidade de compressão (fps)				
	Hall	Stefan	Davinci	Big Buck Bunny	Solar Coronal Holes
Videoconferencia	326,67	329,51	60,91	25,95	29,26
Broadcast	357,38	359,43	68,44	30,29	30,42
Armazenamento_hE	373,36	363,64	66,57	29,26	30,76
Armazenamento_lE	374,42	376,20	67,56	30,97	31,03
Videovigilancia _dp	306,04	301,64	51,64	23,14	22,16

Videovigilancia_st	312,53	300,54	50,74	22,97	21,57

Como se pode verificar pela Tabela 4.7, os vídeos obtidos com o codificador configurado no modo Armazenamento_hE são os que apresentam maiores PSNR, o que se deve a vários fatores. Um deles é a elevada taxa de *bits* máxima configurada neste modo, que permite a aplicação de menores parâmetros de quantização.

Outro fator importante é a utilização do algoritmo VBR para o controlo da taxa de *bits*, pois, como se pode ver na Figura 4.10, em relação ao CBR, permite que maiores taxas de *bits* instantâneas sejam alcançadas para que um dado nível de qualidade se mantenha, tal como é indicado no Anexo F. Note-se que os picos registados neste gráfico referem-se às *frames* do tipo I, que são codificadas com maior número de *bits* que as do tipo P. O codificador se encontrar no modo Platinum optimizado para valorizar a qualidade também contribuiu para tal.

Os vídeos codificados no modo Broadcast, apesar de utilizarem o algoritmo CBR e menores taxas de *bits*, a utilização do codificador no modo Platinum optimizado para valorizar a qualidade permitiu que elevados PSNR fossem alcançados também.

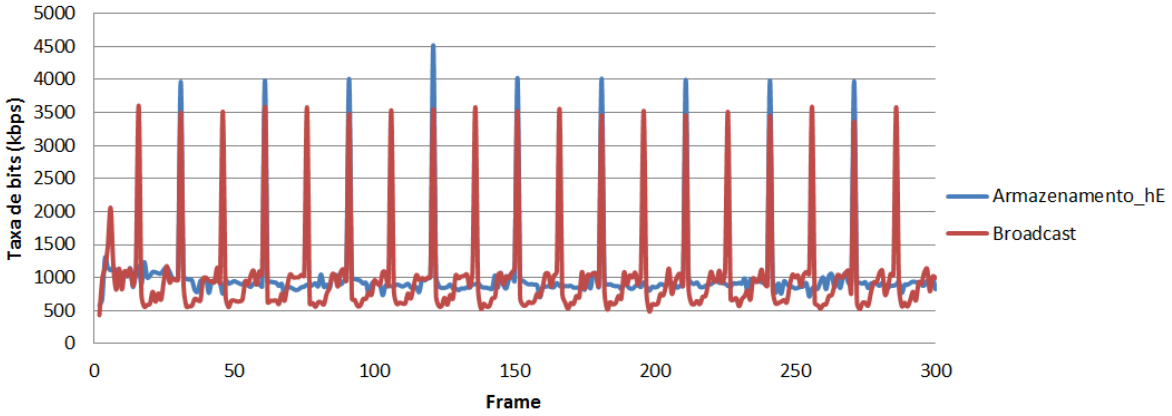


Figura 4.10. Taxa de *bits* registada para cada *frame* do vídeo Hall codificados nos modos Armazenamento_hE e Broadcast.

Os vídeos codificados no modo Armazenamento_lE**,** correspondente à aplicação de armazenamento para dispositivos *low-end,* não alcançaram PSNR tão elevados como nos modos referidos anteriormente porque, apesar de ter sido codificado com o algoritmo VBR, utilizou o codificador no modo Platinum optimizado para valorizar a velocidade, daí ter alcançado as maiores velocidades em *fps* para a maior parte dos casos, tal como se pode verificar na Tabela 4.9.

O modo de compressão NoFilter encontra-se configurado de forma semelhante ao modo Videovigilancia_dp, contudo o filtro de remoção do efeito de blocagem foi desativado com o intuito de se analisar o seu impacto no processo de

compressão. Como se pode ver na Tabela 4.10, as diversas medições realizadas apresentam valores semelhantes para os vídeos comprimidos nestes modos, de modo que, aparentemente, a aplicação deste filtro não influencia o processo de compressão. Contudo, ao se observarem os vídeos, o efeito de blocagem é bem mais visível no vídeo comprimido no modo NoFilter do que no vídeo comprimido no Videovigilancia_dp**.** Tal é notório nas imagens da Figura 4.11, as quais mostram uma parte de uma *frame* do vídeo Stefan comprimido nestes modos.

Tabela 4.10. Parâmetros medidos relativos aos vídeos comprimidos nos modos Videovigilancia_dp e NoFilter.

Vídeo	Modo	Rácio de compressão	Velocidade de compressão média (fps)	Taxa de *bits* média (kbps)	PSNR médio (dB)
Hall	Videovilancia_dp	70,86	306,04	515,02	30,8
	NoFilter	70,83	303,28	515,24	30,77
Stefan	Videovilancia_dp	70,86	301,64	515,03	27,81
	NoFilter	70,83	299,74	515,23	27,78
Davinci	Videovilancia_dp	153,39	51,64	2162,91	29,79
	NoFilter	153,4	51,88	2162,86	29,79
Big Buck Bunny	Videovilancia_dp	92,37	23,14	8141,24	30,03
	NoFilter	92,29	23,28	8148,22	29,98
Solar Coronal Holes	Videovilancia_dp	183,41	22,16	4070,13	31,08
	NoFilter	183,51	23,63	4067,97	31,05

(a) NoFilter (b) Videovigilancia_dp

Figura 4.11. *Frame* do vídeo Stefan comprimido nos modos NoFilter e Videovigilancia_dp.

Uma vez que a utilização do filtro de remoção de artefatos não tem influência em parâmetros como o rácio de compressão, velocidade de codificação e taxa

média de *bits*, mas melhora significativamente a qualidade de imagem, conclui-se que este deve ser sempre utilizado no processo de compressão.

Na Figura 4.12 apresenta-se um diagrama que mostra como devem ser ajustados alguns parâmetros do codificador de modo a se satisfazerem determinados requisitos de compressão. Tal diagrama foi realizado com base na análise ao codificador H.264, tanto a teórica (no Anexo F e no Anexo J) como a experimental, cujos resultados estão expostos nesta secção.

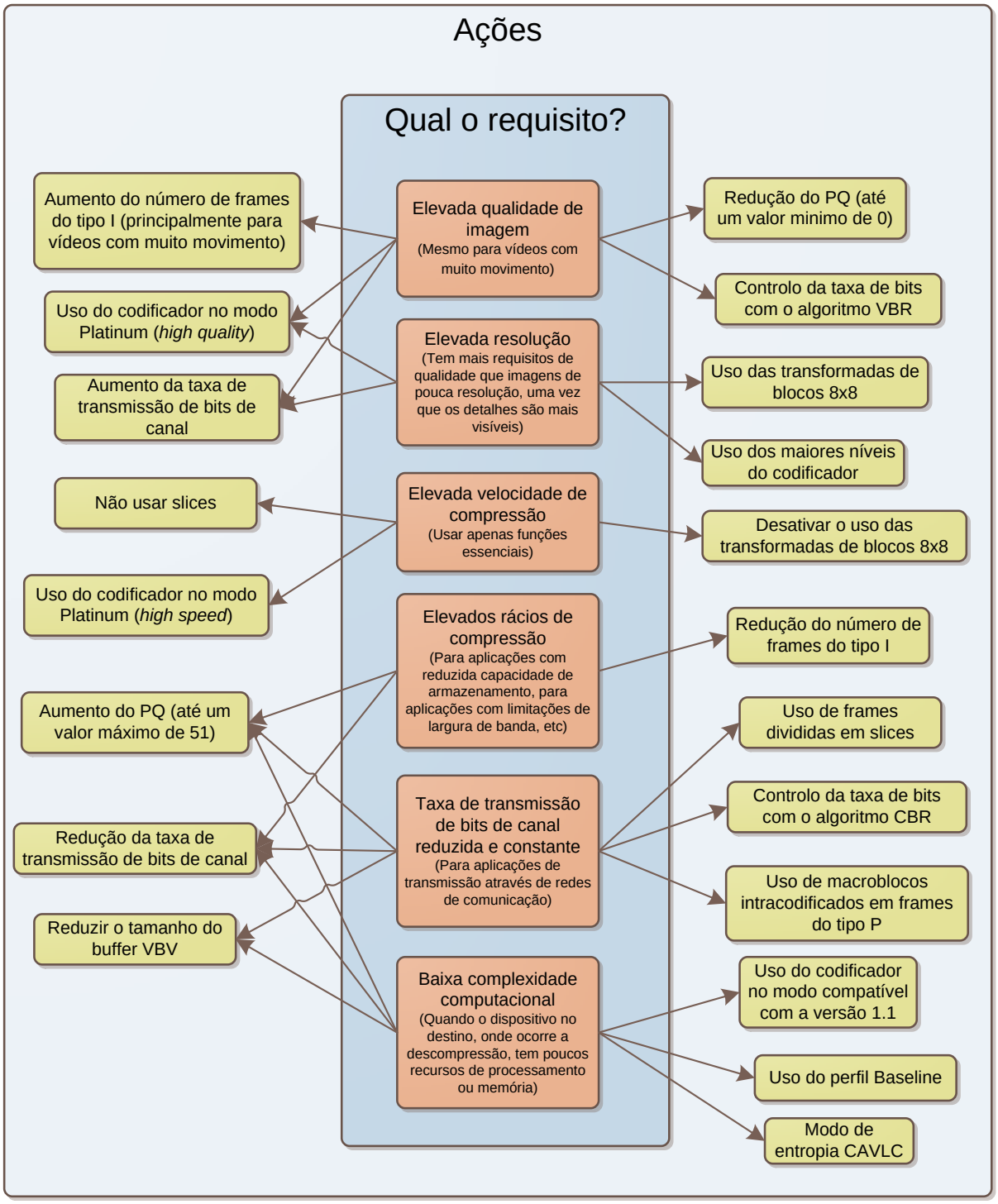


Figura 4.12. Ajuste de parâmetros no codificador H.264 para diferentes requisitos de compressão.

Como se pode observar, este diagrama não faz referência a todos os parâmetros ajustáveis no codificador H.264, uma vez alguns deles apenas são interessantes para aplicações específicas, ou a influência exercida no processo de codificação não é significativa para os requisitos em causa. Ainda assim, verifica-se que são vários os parâmetros ajustáveis para se ir de encontro a um dado requisito.

4.3. Desempenho dos algoritmos de processamento de imagem

O desempenho dos algoritmos de processamento de imagem implementados foi analisado em relação à complexidade computacional associada a cada um deles. Para tal, foi medido o tempo médio de processamento tomado por cada um deles, o que pode ser analisado na Tabela 4.11. No cálculo destes tempos médios não foram incluídos os tempos relativos à imagem "Couple", uma vez que apresenta tempos significativamente menores devido à sua menor resolução.

Tabela 4.11. Tempo médio de processamento tomado por cada algoritmo.

Algoritmo	Tempo médio de processamento (ms)
Luminosidade	51,21
Contraste	93,55
Equalização do histograma	104,71
Balanço de brancos	356,74

Se se medir a complexidade de cada algoritmo através do tempo médio que demora a ser executado, verifica-se que o mais complexo é o que implementa a correção do balanço de brancos, enquanto o mais simples é o que permite alterar a luminosidade, o que já se tinha previsto na secção 3.5.1. Já pela análise da Tabela 4.12, o algoritmo que se destaca é o que implementa a equalização do histograma, pois reduz o rácio de compressão em cerca de 50% em relação aos outros algoritmos e em relação à imagem sem qualquer processamento. Também se destaca por ser o que reduz mais significativamente o PSNR, o que não significa que tenha inserido distorção (como acontece quando se está a analisar imagens ou vídeos comprimidos com diferentes parâmetros). Tal significa é que este é o algoritmo que implementa uma maior modificação na imagem, modificação a qual pode ser no sentido de melhorar ou de piorar a imagem.

Tabela 4.12. Rácio de compressão e PSNR das imagens alteradas com um algoritmo de processamento, e das imagens que não foram alteradas.

	Algoritmos	Imagem						
		Couple	View	F16	Baboo light	Baboo dark	Lena	Tiffany
Rácio de compressão	Nenhum	15,3:1	9,5:1	12,8:1	17,6:1	8,4:1	18,8:1	13,8:1
	Contraste	15,3:1	9,5:1	12,8:1	17,6:1	8,4:1	18,8:1	13,8:1
	Luminosidade	13,7:1	9,7:1	11,5:1	16,1:1	7,4:1	18,8:1	16,3:1
	Eq. do histograma	7,7:1	5,8:1	7,9:1	10,3:1	4,9:1	10,1:1	7,4:1
	Balanço de brancos	15,7:1	9,5:1	13,1:1	17,4:1	7,6:1	19,4:1	13,0:1
PSNR (dB)	Nenhum	40,57	34,74	39,00	46,63	33,46	41,38	38,06
	Contraste	40,57	34,79	39,03	46,64	33,48	41,38	38,07
	Luminosidade	39,13	35,01	37,91	45,75	32,34	41,38	38,91
	Eq. do histograma	33,24	30,95	33,18	40,75	29,79	34,51	32,09
	Balanço de brancos	40,66	34,74	39,38	46,30	32,79	41,68	37,81

Quanto ao algoritmo para alteração do contraste, apesar de apresentar um tempo de processamento médio próximo daquele do algoritmo para equalização do histograma, praticamente não afetou o rácio de compressão o que o torna mais adequado para integração numa aplicação de compressão de imagem ou vídeo.

5. Captura, compressão, armazenamento e transmissão de imagem e vídeo

As aplicações descritas anteriormente foram desenvolvidas especificamente para analisar o desempenho de codificação JPEG e H.264, bem como alguns algoritmos de processamento de imagem. Nesta aplicação pretende-se reunir as principais funcionalidades das aplicações anteriores, bem como adicionar o processo de captura através de um sensor, e a transmissão dos dados através de uma saída DVI.

5.1. Descrição da aplicação

A captura de imagens é realizada através do módulo VPFE descrito na secção 3.1.1**.** Mais precisamente, recorre-se ao IPIPE para aceder aos dados nos formatos UYVY e NV12 para que possam ser comprimidos com os codificadores JPEG e H.264, respetivamente. Diferentes operações podem ser aplicadas em *hardware* aos dados em formato RGB, antes da conversão para o espaço de cor YCbCr. Uma delas é o balanço de brancos, cuja implementação se baseia na soma de um *offset* a cada um dos componentes de cor (de 12 *bits*), seguido pela multiplicação por um ganho [51]. O *offset* pode variar numa gama de -2048 a +2047, enquanto o ganho pode tomar valores desde 0 a 15,998, em passos de 1/512 [51].

Outra operação que este módulo permite realizar, mas já sobre os dados no espaço de cor YCbCr, é o ajuste do contraste e o ajuste da luminosidade. O processo é representado pela equação

$$Y_{out} = clip8\big(clip8\big((Y_{in} \times CTR) \gg 4\big) + BRT\big), \tag{5.1}$$

em que CTR é um fator de melhoramento de contraste que pode tomar valores desde 0 a 15,15, BRT é um fator de melhoramento do brilho que varia desde 0 a 255, Y_{in} é a componente luma dos dados de entrada (8 *bits*), e Y_{out} é a nova componente luma (8 *bits*). A função $clip8$ é dada por

$$clip8(x) = \begin{cases} x, & se\ x \leq 255 \\ 255, & se\ x > 255. \end{cases} \tag{5.2}$$

Uma vez que estas operações já se encontram implementadas em *hardware*, estão otimizadas quanto ao tempo de processamento relativamente aos algoritmos desenvolvidos em *software* para desempenho de funções semelhantes (algoritmos para balanço de brancos, alteração do contraste e alteração da luminosidade tratados na secção 3.5.1), de modo que se optou por integrá-las nesta aplicação. Foram definidas 6 combinações para os parâmetros *offset* e ganho, sendo que o

utilizador deve escolher a combinação que melhor se ajusta às condições de luz do ambiente da captura. Já para a os parâmetros CTR e BRG foram definidas 3 combinações, uma para alterar apenas o contraste, uma para alterar apenas a luminosidade e uma para alterar ambos. Naturalmente que estes valores podem ser alterados no código.

Apesar de estas operações estarem optimizadas em *hardware,* elas têm de ser ativadas durante a configuração do IPIPE, antes de se iniciar a captura, sendo que os ganhos e *offsets* são definidos nesta fase, e não podem ser alterados durante a captura de imagens. Assim sendo, integrou-se também na aplicação os algoritmos em *software* para alteração do contraste e da luminosidade, de modo a se poder ajustar estas caraterísticas durante o processo de captura, sem que seja necessário interrompe-la. Para tal, fizeram-se algumas alterações a estes algoritmos relativamente ao que está descrito na secção 3.5.1**.**

No caso do algoritmo para alteração da luminosidade, foi definido o valor 0 para o *offset* inicial (alteração nula), o qual pode ser aumentado ou reduzido em passos de 20 (cerca de 8% da gama de níveis de intensidade), conforme o utilizador desejar. Este expressa o desejo de alterar a luminosidade digitando o carater “b”, o desejo em aumentar a luminosidade digitando os carateres “bp”, e em reduzir a luminosidade digitando “bm”. Para desativar o algoritmo da alteração da luminosidade, o carater “b” deve ser novamente digitado.

Quanto ao algoritmo para alteração do contraste, continua a ser baseado no método do “contraste automático”, contudo o fator $\frac{g_n}{g}$ na equação (2.10) é calculado de forma diferente daquela descrita na secção 3.5.1. Ao se digitar o carater “c” o algoritmo começa a ser executado, tendo em conta um fator inicial igual a 1. Após este passo, ao se digitar “cp” o fator é somado a 0,25, enquanto se se digitar “cm” é subtraído 0,25 ao valor do fator. Quando se torna a digitar “c”, o algoritmo deixa de ser executado, até que este carater volte a ser digitado.

O algoritmo para equalização do histograma também foi integrado para possível utilização, o que pode ser ativado digitando “h”, contudo nenhuma alteração foi realizada ao mesmo relativamente à descrição na secção 3.5.1. Já o algoritmo para ajuste do balanço de brancos não foi integrado devido à complexidade computacional a que está associado.

O utilizador indica que deseja comprimir e guardar a imagem atual digitando “r”, enquanto no caso da captura de vídeo, ao se digitar este comando um número de *frames* indicado pelo utilizador são comprimidas e guardadas num ficheiro, terminando automaticamente a captura no fim deste processo. Quando se captura

imagens, é necessário indicar o término da captura digitando “s”. O fluxograma representativo desta aplicação apresenta-se na Figura 5.1.

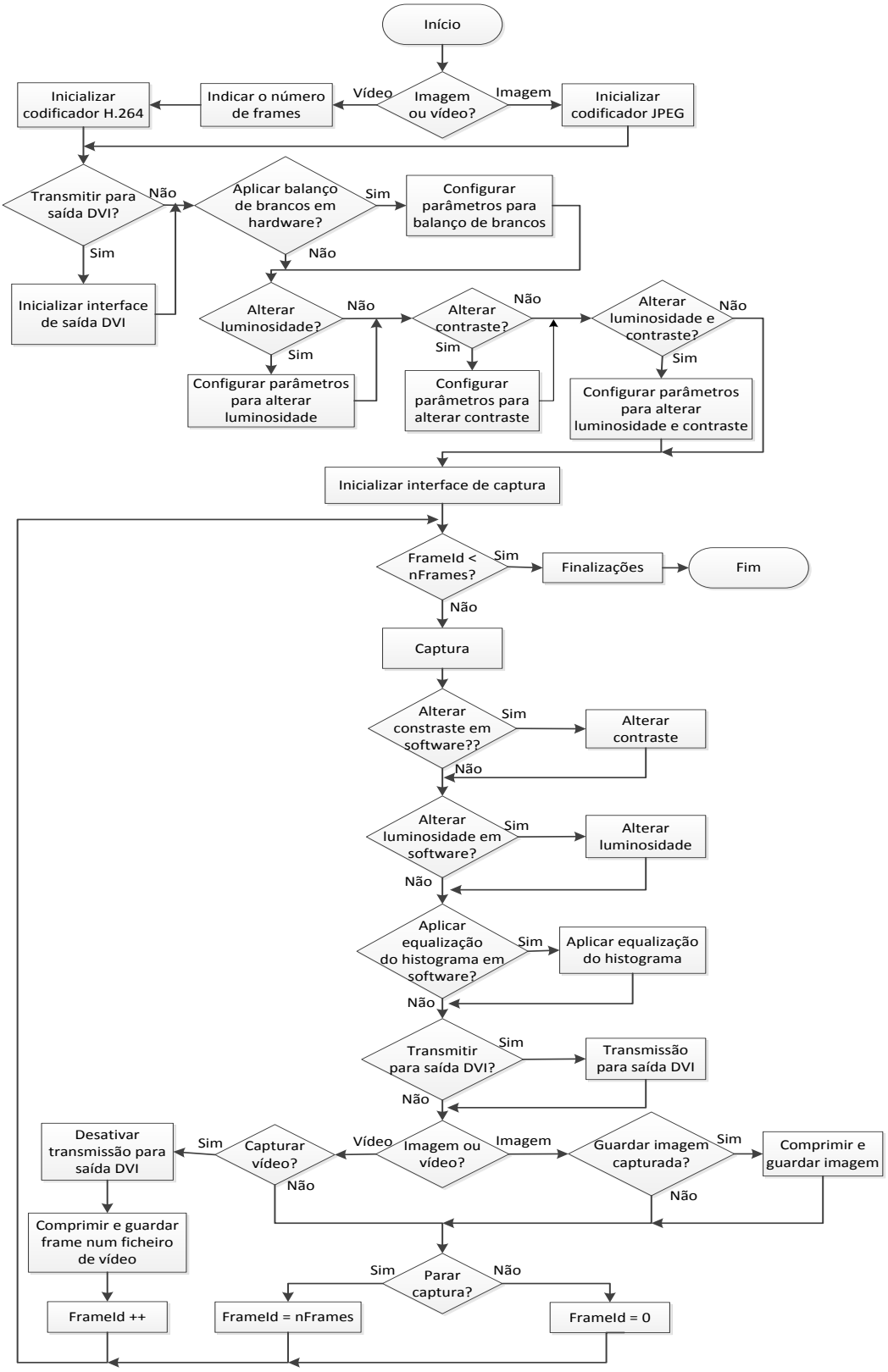


Figura 5.1. Fluxograma descritivo da aplicação para captura, processamento, compressão, transmissão ou armazenamento de imagens ou vídeos.

A aplicação é executada em linha de comandos, no sistema operativo Ubuntu. Na Figura 5.2 pode-se observar o aspeto da mesma para um caso em que foi executada com o intuito de capturar imagens.

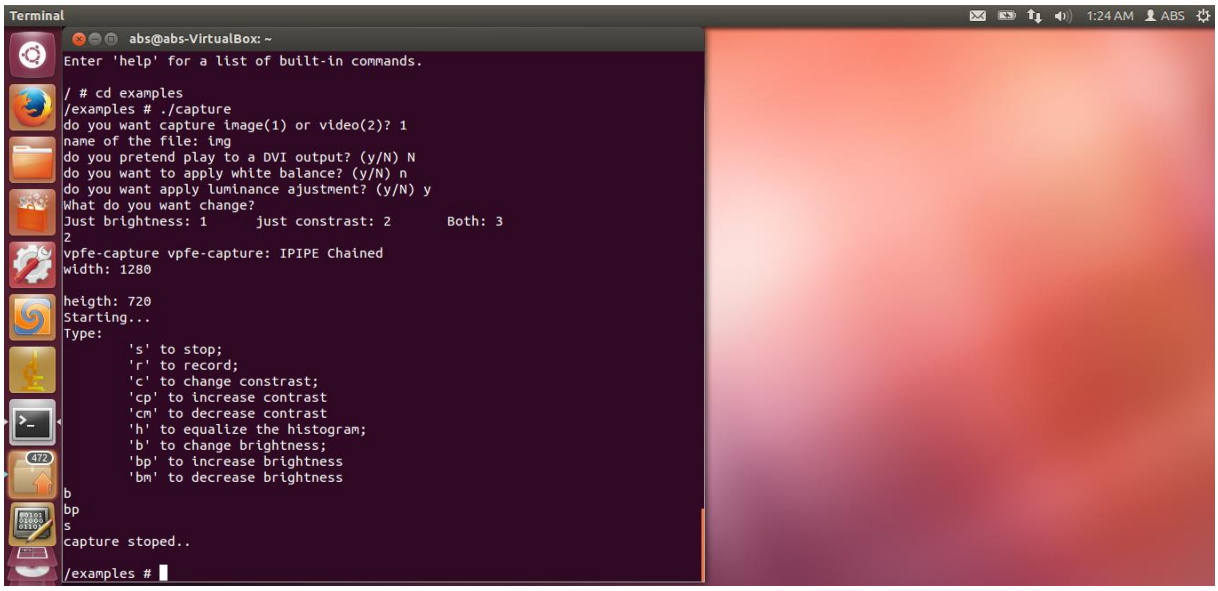


Figura 5.2. Execução da aplicação para capturar imagens.

Relativamente às configurações dos codificadores, para o JPEG foi utilizado o fator de qualidade 75, pois verificou-se que este permite obter imagens de “Boa” qualidade, com distorção pouco perturbante. O codificador H.264 foi configurado um modo direcionado para o armazenamento, nomeadamente o modo Armazenamento_lE, correspondente à aplicação de armazenamento em dispositivos *low-end.* A escolha deste modo para a codificação H.264 foi tomada tendo em consideração que esta aplicação não permite enviar os dados comprimidos através de uma rede, logo não tem limitações de largura de banda, e torna-a adequada a aplicações com limitações de armazenamento.

Num ficheiro de texto, são escritos ainda os *timestamps* registados após a compressão de cada *frame*. Através da diferença entre os tempos de *frames* consecutivas, foi calculada a taxa de *frames* média de captura e compressão dos diferentes vídeos. O mesmo foi realizado para calcular a taxa de *frames* média de captura e transmissão para a saída DVI, com a diferença de que os *timestamps* são registados após a transmissão de uma *frame* completa para a saída DVI.

5.2. Desempenho da aplicação

Para se analisar do desempenho desta aplicação foram capturadas imagens e vídeos em diferentes resoluções, bem como foram aplicados os diferentes métodos de processamento integrados na aplicação. Na Figura 5.3 apresentam-se

algumas das imagens capturadas, onde se inclui uma imagem na qual não foi aplicada qualquer alteração. Estas e outras imagens capturadas podem ser observadas também no Anexo P.

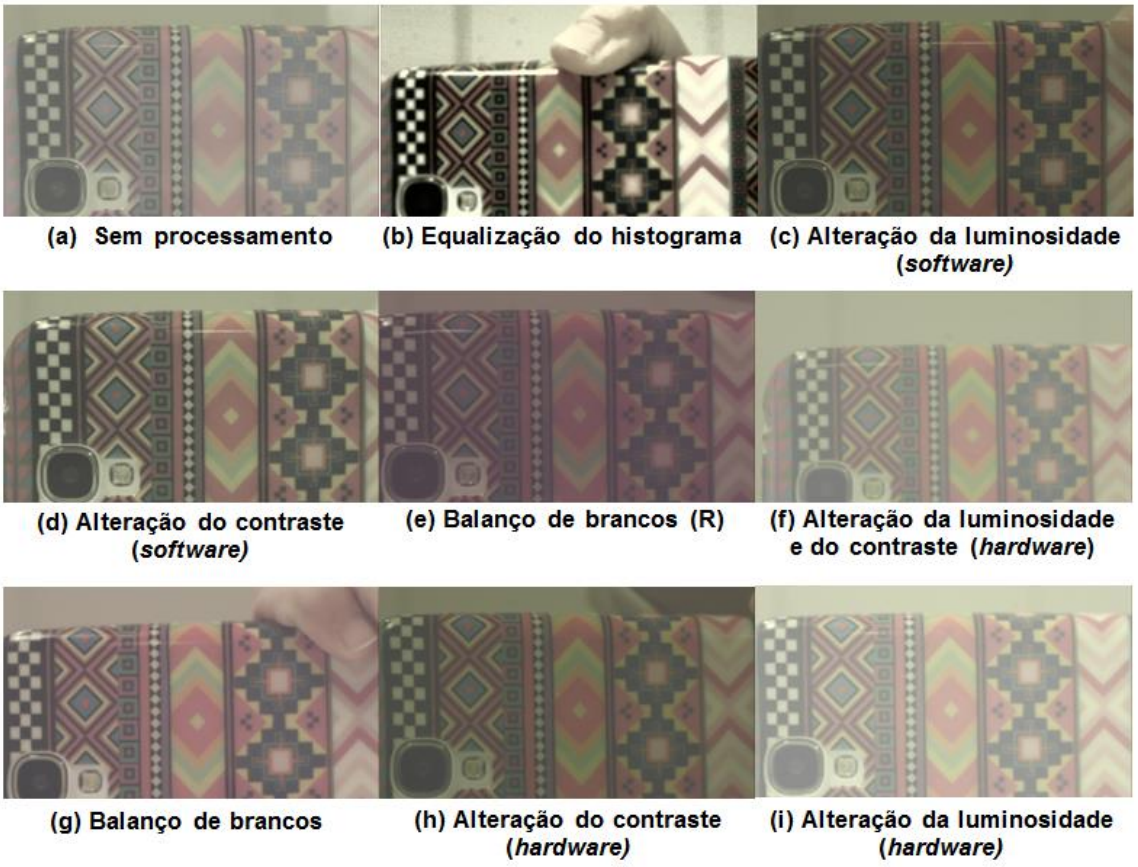


Figura 5.3. Imagens capturadas e comprimidas com codificador JPEG.

As imagens na Figura 5.3 (e) e na Figura 5.3 (g) resultaram da alteração do balanço de brancos em *hardware*, usando diferentes ganhos e *offsets.* Ambas têm pouca luminosidade, contudo os ganhos e *offsets* que deram origem à imagem (e), estão associados a uma componente R (vermelha) mais forte, fazendo-os adequados a ambientes em que esta componente esteja em falta. Para ambientes em que uma ou ambas as outras componentes estejam em falta, os ganhos e *offsets* associados às mesmas devem ser aumentados.

As imagens apresentadas na Figura 5.3 foram capturadas com resolução 1280x720. Para esta resolução, registaram-se velocidades de compressão até cerca de 73 fps, valor o qual é superior ao típico indicado na secção 3.2.1, de 60 fps. Contudo outras imagens foram obtidas com resoluções diferentes, nomeadamente, 1920x1080, 1280x960, 640x480 e 768x512. Estas mesmas resoluções também foram utilizadas na captura de vídeo (com exceção da resolução 1920x1080, tendo

sido utilizada 1920x1088 porque o codificador de vídeo apenas suporta valores múltiplos de 16 [57]).

A Tabela 5.1 apresenta a taxa de *frames* de captura e transmissão e a taxa de *frames* de captura e compressão médias para vídeos capturados com diferentes resoluções, e sem qualquer modificação. Como se pode verificar, a taxa de *frames* de captura e transmissão para saída DVI reduz com o aumento da resolução, tal como se esperava. Já em relação à taxa de *frames* média de captura e compressão tal não acontece. Estes resultados não se devem ao processo de compressão, uma vez que na secção 4.2 verificou-se que a taxa de *frames* de compressão é maior para as menores resoluções, de modo que o processo de captura é o responsável por tal. Verifica-se ainda que a taxa média registada para a resolução 768x512 é significativamente elevada em relação aos valores obtidos para as outras resoluções. Tal aspeto faz desta resolução a mais adequada para aplicações em que uma elevada taxa de *frames* seja um requisito.

Tabela 5.1. Taxas de *frames* medidas em vídeos capturados com diferentes resoluções.

Resolução	**1920x1088**	**1280x960**	**1280x720**	**768x512**	**640x480**
Taxa de *frames* de captura e compressão média (fps)	28,42	34,41	23,00	107,44	29,94
Taxa de *frames* de captura e transmissão média (fps)	(não foram transmitidas)		13,01	25,03	25,89

Procurou-se saber também qual o impacto da aplicação dos vários métodos utilizados para alterar as imagens na taxa de *frames* de captura e compressão, tendo sido utilizada para tal apenas a resolução 1280x720. Verificou-se que a referida taxa não foi afetada quando se realizaram as alterações através dos métodos em *hardware*, contudo, esta sofre uma redução significativa com a aplicação dos algoritmos em *software* (alteração da luminosidade, alteração do contraste e equalização do histograma).

O algoritmo responsável pela maior redução é o que trata da equalização do histograma, provocando uma redução para cerca de 2,12 fps, seguido pelo algoritmo de alteração do contraste, associado a uma redução para cerca de 2,70 fps, e pelo algoritmo para alteração da luminosidade, cuja redução é para 5,21 fps. Tais resultados já eram esperados, pois os algoritmos em *software* são aplicados em tempo real nas imagens capturadas, logo antes da compressão das mesmas, interferindo diretamente no processo. É natural que seja a equalização do histograma que maior redução provoca, pois verificou-se na secção 4.3 que este é o

algoritmo que toma maior tempo de processamento de entre os três integrados nesta aplicação.

As alterações implementadas tanto nos vídeos como nas imagens foram manipuladas através da configuração de alguns parâmetros. Isto quer dizer, portanto, que a aplicação é flexível o suficiente para que o utilizador tenha liberdade para implementar as alterações que lhe forem mais convenientes. Para além da possibilidade de alterar os parâmetros relativos a cada método de processamento, diferentes métodos podem ser utilizados simultaneamente, de modo que se pode obter uma multiplicidade de imagens através de diferentes combinações de métodos, e diferentes combinações nas configurações dos parâmetros de cada método.

O método mais adequado para implementar as alterações, isto é, se através do *hardware* ou através do *software*, varia conforme a utilidade a ser dada a esta aplicação. A implementação das alterações através do *hardware*, durante a captura, implica que os parâmetros apenas possam ser configurados antes da captura, e não durante. Deste modo, adequa-se a utilizações em que o sensor seja utilizado num ambiente em que as condições de luz variem pouco ou nada, de forma que os parâmetros tenham de ser configurados apenas uma vez, antes de se iniciar a captura, pois não é possível fazer o ajuste dinâmico. Como o processamento da imagem realizado desta forma não afeta a taxa de *frames* de captura e compressão, então este é o mais adequado para a captura de vídeo.

Já o processamento da imagem através dos algoritmos em *software* afeta significativamente a taxa de *frames* de captura e compressão, de modo que não devem ser utilizados em aplicações de captura de vídeo, em que a taxa de *frames* é normalmente um requisito importante. Por outro lado, permite alterar a imagem em tempo real, de modo a ajustá-la às condições da luz ambiente. Assim sendo, o processamento da imagem através dos algoritmos deve ser utilizado em aplicações como a captura e compressão JPEG de imagens individuais, ou em aplicações de vídeo em que os requisitos em termos de taxa de *frames* sejam reduzidos.

6. Conclusões e trabalhos futuros

Nesta secção são apresentadas as principais conclusões retidas ao longo do desenvolvimento deste trabalho, bem como são indicadas algumas propostas de trabalhos futuros.

6.1. Conclusões

Na revisão do estado da arte verificou-se que são diversas as técnicas utilizadas para reduzir as redundâncias num sinal de imagem ou vídeo. Tais técnicas são adotadas pelos padrões de compressão no sentido de capacitá-los para responder às necessidades das aplicações de compressão, as quais se referem principalmente ao rácio de compressão e à qualidade de imagem. Não é possível, contudo, satisfazer ambos os requisitos simultaneamente, existindo uma relação de compromisso entre eles: elevados rácios de compressão estão associados a reduzida qualidade de imagem, e vice-versa. Para além da otimização desta relação, ou seja, elevados rácios de compressão com a melhor qualidade de imagem possível, os padrões de compressão têm sido desenvolvidos também no sentido de satisfazer os requisitos de largura de banda das aplicações de compressão. Verifica-se que melhorias nestes campos têm sido alcançadas, contudo à custa de aumentos significativos na complexidade computacional. Tal tornou-se mais saliente por terem sido encontradas implementações do recente padrão de compressão de vídeo H.265 apenas em arquiteturas com pelo menos 8 núcleos.

Verificou-se também que uma solução frequente para a implementação de sistemas de compressão de imagem e vídeo em SoC passa pela utilização de núcleos ARM. O desempenho de cada SoC, em termos de padrões de compressão implementados, velocidade de compressão e resoluções suportadas, varia, contudo, em função de parâmetros como o número de processadores e a frequência de processamento dos mesmos. Verificou-se ainda que os núcleos ARM estão presentes nos SoC com maiores desempenhos relativamente aos parâmetros referidos.

Na análise ao desempenho da compressão de imagem e vídeo no processador ARM utilizado, tendo em conta os valores típicos e as indicações na documentação, a taxa de *frames* para uma dada resolução foi efetivamente alcançada. Mais precisamente, para a compressão JPEG foi registado um desempenho de até cerca de 73 fps para a resolução 1280x720 com apenas o codificador em execução, e para a compressão H.264 constatou-se o desempenho de 60 fps para 1280x720, e 30 fps para 1920x1088. A documentação destes codecs,

contudo, é escassa, o que dificultou o processo de integração dos codecs nas aplicações, a compilação das mesmas, e a análise dos processos de compressão no âmbito deste trabalho.

Os resultados obtidos da análise à compressão JPEG, nomeadamente do tratamento dos dados recolhidos nos inquéritos realizados, mostram que imagens de "Boa" qualidade (de entre as categorias utilizadas nos inquéritos) podem ser obtidas com um fator de qualidade mínimo de cerca 60, e com rácios de compressão que podem ir até cerca de 22:1. Quanto à distorção, começa a ser ligeiramente perturbante para fatores de qualidade inferiores a cerca de 76. Quanto à velocidade de compressão, é praticamente independente do fator de qualidade utilizado, sendo afetada essencialmente pela resolução das imagens. Verificou-se ainda que o desempenho de compressão em termos de qualidade da imagem, distorção visível, rácio de compressão, e PSNR também varia conforme a imagem seja constituída por poucas ou muitas componentes de alta frequência. Os melhores desempenhos, ao nível do conjunto de parâmetros referidos, foram obtidos para imagens com menos componentes em alta frequência, o que era esperado tendo em conta as caraterísticas deste padrão.

Quanto ao codificador H.264 analisado neste trabalho, dispõe de uma larga gama de opções de codificação definidas no padrão. Esta multiplicidade de parâmetros ajustáveis proporcionam a este codificador de vídeo uma elevada adaptabilidade às mais diversas aplicações de compressão, uma vez que permitem ajustar os diversos parâmetros de acordo com os requisitos de cada aplicação, tal como se constatou pela análise dos resultados obtidos.

Relativamente ao rácio de compressão, pode ser aumentado principalmente através de elevados parâmetros de quantização, de reduzidas taxas máximas de *bits*, e de um reduzido número de *frames* do tipo I. Estes parâmetros também afetam a qualidade de imagem (medida objetivamente pelo PSNR), mas de forma inversa, ou seja, reduzem a qualidade. Note-se que no caso do número de *frames* do tipo I, um elevado número apenas melhora a qualidade da imagem se a taxa de *bits* máxima for suficientemente elevada para permitir que tais *frames* sejam codificadas com reduzidos parâmetros de quantização. Caso contrário, elevados parâmetros de quantização serão utilizados para tais *frames*, não causando melhoria na qualidade, podendo até piorar. Outro parâmetro que melhora a qualidade de imagem é o uso do filtro de remoção de artefatos. Para além disso, a sua utilização não provocou qualquer alteração a nível de rácio de compressão ou velocidade de compressão comparativamente aos vídeos onde não foi utilizado.

Para aplicações onde seja requerido um rigoroso controlo da taxa de *bits*, o algoritmo CBR deve ser utilizado, caso contrário pode ser utilizado o VBR, ou, se não for necessário qualquer controlo da taxa de *bits*, pode ser definido um parâmetro de quantização fixo para todas as *frames*, valor do qual deve ser ajustado de acordo com os requisitos de qualidade e de rácio de compressão pretendidos.

A codificação de imagem JPEG e a codificação de vídeo H.264 foram reunidos numa única aplicação. A esta, adicionou-se ainda a captura de imagem e vídeo com diferentes resoluções, a transmissão dos dados capturados para uma saída DVI, e o processamento das imagens capturadas através de implementações em *hardware* e em *software*. Para a captura, compressão H.264 e armazenamento de vídeo, sem a aplicação de qualquer processamento, foi obtida uma taxa média de *frames* de 23 fps para a resolução de 1280x720. Já para a resolução 1920x1088, a taxa média de *frames* obtida foi de 28 fps, o qual é muito próximo dos 30 fps indicados para a captura e para a compressão H.264 desta resolução. A taxa obtida para a resolução de 768x512 destacou-se por ser significativamente mais elevada que as outras, cerca de 107 fps.

Estas taxas mantêm-se quando se aplicam alterações através do *hardware*, de modo que é possível aplicar processamento efetivo nas imagens capturadas, sem que o desempenho a este nível seja afetado. Contudo, o processamento que é aplicado às imagens obtidas depende da configuração de alguns parâmetros, a qual apenas pode ser realizada antes da captura, o que significa que tais parâmetros não podem ser configurados dinamicamente em função das condições de luz ambiente onde ocorre a captura. Nesse sentido, os algoritmos implementados em *software* são vantajosos, uma vez que permitem esse ajuste dinâmico. Por outro lado, a utilização de tais algoritmos afeta significativamente as taxa médias de *frames* referidas. No caso da resolução 1280x720, a redução são para 5,21 fps, 2,70 fps e 2,12 fps quando são utilizados os algoritmos para alteração da luminosidade, para alteração do contraste e para a equalização do histograma, respetivamente.

O desempenho obtido para esta aplicação faz dela adequada tanto para a captura de imagens individuais, como para a captura de vídeo. Uma vez que o processamento de imagem implementado em *hardware* não afeta a taxa de captura e compressão, este pode ser utilizado para ambos os casos. Já os algoritmos implementados em *software*, devido aos atrasos introduzidos, apenas devem ser utilizados para a captura de vídeo em que os requisitos em termos de taxa de *frames* sejam reduzidos, ou para a captura de imagens. O vídeo capturado pode ser armazenado ou transmitido para uma saída DVI, de modo que esta aplicação está

dirigida para o armazenamento, não estando preparada para a transmissão de dados através de uma rede.

6.2. Trabalhos futuros

Neste trabalho foi realizada uma análise aos padrões de compressão de imagem e vídeo, nomeadamente o JPEG e o H.264. Para se estudar o desempenho destes padrões num processador ARM, foi utilizado o controlador DM368, de modo que futuramente seria interessante poder comparar o desempenho obtido neste controlador com o obtido noutro controlador que também integre um processador ARM.

Também foi desenvolvida uma aplicação que possui potencial para melhoramentos em vários sentidos. Em trabalhos futuros, eis alguns aspetos a ter em conta:

- Melhoria do desempenho ao nível da taxa de *frames* de captura, compressão H.264 e armazenamento de vídeo com resolução 1280x720;
- Otimização do tempo de processamento dos algoritmos implementados em *software*;
- Melhoria da qualidade das imagens e vídeos capturados pela introdução de mais métodos de processamento de imagem;
- Adicionar funcionalidade no sentido de permitir o *streaming* de imagem e vídeo comprimido através de uma rede;
- Desenvolver uma interface gráfica para melhor interação com o utilizador;
- Implementação de uma aplicação semelhante, mas com outros padrões como o JPEG2000 e o H.265, naturalmente utilizando uma plataforma com maior capacidade de processamento.

7. Referências

[1] D. Austerberry, *The Technology of Video and Audio Streaming*, 2nd ed. Focal Press, 2005.

[2] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*. Chichester, UK: John Wiley & Sons, Ltd, 2003.

[3] T. Acharya and A. Ray, *Image processing: principles and applications*, 1º ed. Hoboken: John Wiley & Sons, 2005.

[4] M. Martina, M. Shafique, and A. Norkin, "VLSI Circuits, Systems, and Architectures for Advanced Image and Video Compression Standards," *VLSI Des.*, vol. 2012, no. February 2012, pp. 1–3, 2012.

[5] L. G. Shapiro and G. C. Stockman, *Computer Vision*. Upper Saddle River: Prentice Hall, 2001.

[6] R. C. Gonzalez and R. E. Woods, "Digital Image Fundamentals," in *Digital Image Processing*, 2nd ed., Upper Saddle River: Prentice Hall, 2001, pp. 34–74.

[7] W. Burger and M. J. Burge, *Digital Image Processing - An Algorithmic Introduction using Java*. 2008.

[8] C. Poynton, *Digital video and HD: Algorithms and Interfaces*. San Francisco: Morgan Kaufmann Publishers, 2003.

[9] M. Perkins, *The practical guide to digital imaging: Mastering the terms, technologies, and techniques*. Buffalo, NY: Amherst Media, Inc., 2005.

[10] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed., vol. 2011. Pearson Education, 2011.

[11] J. Burg, *The science of digital media*, no. 0340969. Upper Saddle River: Prentice Hall, 2007.

[12] F. C. V. Grilo, M. E. S. C. António, J. A. C. Lopes, and J. A. R. Azevedo, "Transformada de Fourier," in *Teoria do Sinal e suas aplicações*, Escolar Editora, 2010.

[13] *The free dictionary by farlex*. [Online]. URL: http://encyclopedia2.thefreedictionary.com/Spatial+aliasing.

[14] K. Jack, "Chapter 3: Color spaces," in *Video Demystified: A Handbook for the Digital Engineer*, 5th ed., vol. 5, no. 3, 2007, pp. 15 – 34.

[15] E. T. M. Manoel, “Codificação de Vídeo H.264 - Estudo de Codificação Mista de Macroblocos,” Universidade Federal de Santa Catarina, 2007.

[16] I. Bocharova, *Compression For Multimedia*. New York: Cambridge University Press, 2010.

[17] *Recommendation ITU-T H.265, Series H: Audiovisual and Multimedia Systems, Infrastructure of audiovisual services – Coding of Moving Video, High Efficiency Video Coding*, ITU-T, 2013

[18] U. Mitra, *Introduction to multimedia systems*. Academic Press, 2002.

[19] D. Vernon, “Fundamentals of digital image processing,” in *Machine Vision: Automated Visual Inspection and Robot Vision*, Prentice Hall, 1991, pp. 44–84.

[20] E. Y. Lam, “Combining gray world and retinex theory for automatic white balance in digital photography,” *Proc. Ninth Int. Symp. Consum. Electron. 2005. (ISCE 2005).*, pp. 134–139.

[21] E. Lam and G. Fung, “Automatic white balancing in digital photography,” in *Single-sensor imaging : Methods and applications for digital cameras*, CRC Press, 2008, pp. 267–294.

[22] S. Bianco, F. Gasparini, and R. Schettini, “Combining strategies for white balance,” in *Digital Photography III*, 2007, vol. 39.

[23] E. H. Land and J. J. McCann, “Lightness and Retinex Theory,” *J. Opt. Soc. Am.*, vol. 61, 1971.

[24] *H.264 video compression standard - New possibilities within video surveillance*, white paper, AXIS Communications, 2008

[25] K. Sayood, *Introduction to Data Compression*, 3rd ed. San Francisco: Morgan Kaufmann Publishers, 2006.

[26] B. Furht, S. Smoliar, and H. Zhang, *Video and Image Processing in Multimedia Systems*. Kluwer Academic Publishers, 1995.

[27] *ITU-R Recommendation BT.500-13, "Methodology for the subjective assessment of the quality of television pictures*," 2012

[28] *ITU-T Recommendation P.910, "Subjective video quality assessment methods for multimedia applications,"* Abril. 2008

[29] I*TU-R Recommendation BT.710, “Subjective Assessment Methods for Image Quality in High-definition Television,”* Nov. 1998

[30] A. N. Netravali and B. G. Haskell, *Digital Pictures: Representation, Compression and Standards*, 2nd ed., no. 1. Plenum, 1995.

[31] D. M. Rouse, R. Pépion, P. Le Callet, and S. S. Hemami, “Tradeoffs in subjective testing methods for image and video quality assessment,” *Proc. SPIE*, vol. 7527, 2010.

[32] G. Wallace, “The JPEG still picture compression standard,” *Commun. ACM*, vol. 34, pp. 30–44.

[33] D. Salomon, *Data Compression: The complete reference*, 4th ed. Springer, 2007.

[34] D. Grois, D. Marpe, A. Mulayoff, and O. Hadar, “Performance Comparison of H.265 / MPEG-HEVC , VP9 , and H.264/MPEG-AVC Encoders,” vol. 2013. 2013.

[35] F. Ebrahimi, M. Chamik, S. Winkler, "JPEG vs. JPEG2000: An objective comparison of image encoding quality", Proc. SPIE Applications of Digital Image Processing, vol. 5558, 300-308, 2004.

[36] *ITU-T Recommendation H.265, Series H: Audiovisual and Multimedia Systems, Infrastructure of audiovisual services – Coding of Moving Video, High Efficiency Video Coding*, ITU-T, 2013.

[37] K. R. Rao, D. N. Kim, and J. J. Hwang, *Video coding standards: AVS China, H.264/MPEG-4 PART 10, HEVC, VP6, DIRAC and VC-1*. Springer, 2013, p. 463.

[38] *Emerging Markets for H.264 Video Encoding Leveraging High Definition and Efficient IP Networking*, white paper, Fujitsu Microelectronics America, Inc., Maio 2010

[39] P. Tseng, Y. Chang, Y. Huang, and H. Fang, “Advances in Hardware Architectures for Image and Video Coding - A Survey,” *Proc. IEEE*, vol. 93, no. 1, pp. 184–197, Janeiro 2005.

[40] S. Chien, Y. Huang, C. Chen, H. H. Chen, and L. Chen, “Hardware Architecture Design of Video Compression for Multimedia Communication Systems,” *IEEE Commun. Mag.*, vol. 43, no. August, pp. 123–131, 2005.

[41] Texas Instruments Incorporated, *Digital Signal Processors* [Online], URL:, http://www.ti.com/lsds/ti/dsp/video_processors/overview.page.”, Consultado em: Setembro de 2013.

[42] Ambarella, *iOne - Smart Camera Solution,* [PDF] URL: http://www.ambarella.com/uploads/docs/iOne%20Brief%20121113.pdf, Consultado em: Maio de 2014.

[43] MediaTek, *MT6595: Octa-core LTE platform,* [Online], URL:, http://goo.gl/3ighz4, Consultado em: Janeiro de 2015.

[44] Zopo, *zp999,* [Online], URL: http://www.zopomobile.com/?p=26&a=view&r=16,” Consultado em: Janeiro de 2015.

[45] Qualcomm, *Snapdragon 805 devices,* [Online], URL: https://www.qualcomm.com/products/snapdragon/devices/all?processor=805, Consultado em: Janeiro de 2015.

[46] Texas Instruments Incorporated, Codec Engine Server Integrator User's Guide, User datasheet SPRUED5B, 2007

[47] Texas Instruments Incorporated, xDAIS-DM (Digital Media) User Guide, User datasheet SPRUEC8B, 2007.

[48] Texas Instruments Incorporated, TMS320DM368 Digital Media System-on-Chip (DMSoC), User datasheet SPRS668C, 2011.

[49] RidgeRun, *RidgeRun Embedded Solutions*, [Online], URL: https://www.ridgerun.com/www/index.php.”, Consultado em: Setembro de 2013.

[50] RidgeRun, *RidgeRun SDK Turrialba User Guide* [Online], URL: https://developer.ridgerun.com/wiki/index.php/RidgeRun_Turrialba_SDK_User_Guide, Consultado em: Setembro de 2013.

[51] Texas Instruments Incorporated, TMS320DM36x Digital Media System-on-Chip (DMSoC), Video Processing Front End (VPFE) User ’ s Guide, User datasheet SPRUFG8C, 2010.

[52] Texas Instruments Incorporated, JPEG Sequential Encoder on DM365 User ’s Guide, User datasheet SPRUEV4B, 2010.

[53] Texas Instruments, *DM365 Codecs FAQ,* [Online], URL: http://processors.wiki.ti.com/index.php/DM365_Codecs_FAQ, Consultado em: Janeiro de 2014.

[54] Texas Instruments, *DM365 Codec Availability Schedule,* [Online], URL: http://processors.wiki.ti.com/index.php/DM365_Codec_Availability_Schedule, Consultado em: Janeiro de 2014.

[55] TI E2E Community, D*aVinci Video Processors*, [Online], URL: http://e2e.ti.com/support/dsp/davinci_digital_media_processors/f/100/p/273870/9, Consultado em: Janeiro de 2014.

[56] Texas Instruments Incorporated, JPEG Sequential Decoder on DM365 User ’s Guide, User datasheet SPRUEV3B, 2010.

[57] Texas Instruments Incorporated, H . 264 Base / Main / High Profile Encoder on DM365 / DM368 User ’ s Guide, User datasheet SPRUEU9B, 2011.

[58] USC Viterbi, *The USC-SIPI Image Database*, [Online] URL: http://sipi.usc.edu/database/database.php, Consultado em: Setembro de 2013.

[59] Blelloch, G., "Introduction to Data Compression". Carnegie Mellon University, URL: http://www.cs.cmu.edu/~guyb/realworld/compression.pdf, Janeiro 2013

[60] G. K. Wallace, “The JPEG still picture compression standard,” *IEEE Trans. Consum. Electron.*, vol. 38, pp. 1–17, 1992.

[61] Texas Instruments Incorporated, Application Parameter Settings for TMS320DM365 H.264 Encoder, Application Report, SPRABA9, 2010.

[62] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, “Video coding with H.264/AVC: tools, performance, and complexity,” *IEEE Circuits Syst. Mag.*, vol. 4, no. 1, pp. 7–28, 2004.

[63] O. Kamariotis, “Bridging the Gap Between CBR and VBR for H264 Standard,” *Int. J. Electr. Comput. Electron. Commun. Eng.*, vol. 1, no. 8, pp. 339–344, 2007.

[64] Matlab, *MATLAB - The Language Of Technical Computing* [Online], URL: http://www.mathworks.com/products/matlab/, Consultado em: Setembro de 2013.

[65] avconv, *avconv Documentation*, [Online] URL: https://libav.org/avconv.html, Consultado em: Maio de 2014

[66] USC Viterbi, Volume 3: Miscellaneous, [Online] URL: http://sipi.usc.edu/database/database.php?volume=misc, Consultado em: Setembro de 2013.

[67] S. V. Viraktamath and G. V. Attimarad, “Performance analysis of JPEG algorithm,” in *2011 - International Conference on Signal Processing, Communication, Computing and Networking Technologies, ICSCCN-2011*, 2011, no. Icsccn, pp. 629–633.

[68] G. Zapryanov, D. Ivanova, and I. Nikolova, “Automatic White Balance Algorithms for Digital Still Cameras–a Comparative Study,” *acad.bg*, pp. 16–22, 2012.

[69] C. Weng, H. Chen, and . C. F., “A Novel Automatic White Balance Method for Digital Still Cameras,” *Proc. IEEE Int. Symp. Circuits Syst.*, p. 3804, 2005.

[70] M. K. Jain, “Numerical Methods for Scientific and Engineering Computation,” *SIAM Review*, vol. 29. New Age International, p. 117, 2003.

[71] B. Oztan, A. Malik, Z. Fan, and R. Eschbach, “Removal of artifacts from JPEG compressed document images,” *Proc. SPIE*, 2007.

[72] Y. J. Butler, *The Advanced Digital Photographer's Workbook: Professionals Creating And Outputting World-Class Images*, Taylor & F. 2005.

[73] T. LoCascio, *Mastering Photoshop CS3 for Print Design and Production*, John Wiley. 2007.

[74] Wikipedia, *Posterization*. [Online]. URL: http://en.wikipedia.org/wiki/Posterization, Consultado em: Setembro de 2014.

[75] CCITT, *Terminal Equipment and Protocols for Telematic Services, Information Technology - Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines,* Recommendation T.81, Setembro 1992.

[76] M. P. Ribeiro dos Santos, "Codificação de Vídeo MPEG-4 em FPGA," Universidade de Aveiro, 2007.

[77] Chen, Z., Reznik, Y., "Analysis of video Codec Buffer and Delay under Time-varying Channel," IEEE Visual Communications and Image Processing Conference (VCIP), San Diego, CA, USA, November 27 - 30, 2012.

[78] Grois, D., Hadar, O., "Recent Advances in Region-of-interest Video Coding, Recent Advances on Video Coding", Dr. Javier Del Ser Lorente (Ed.), ISBN: 978-953-307-181-7, InTech, DOI: 10.5772/17789. URL: http://goo.gl/e8sGhQ, 2011

[79] Texas Instruments Incorporated, DM3730, DM3725 Digital Media Processors, User datasheet SPRS685D, 2011.

[80] Texas Instruments Incorporated, TMS320DM816x DaVinci Video Processors, User datasheet SPRS614E, 2014.

[81] Texas Instruments Incorporated, TMS320DM6467T Digital Media System-on-Chip, User datasheet SPRS605C, 2012.

[82] Texas Instruments Incorporated, DM385, DM388 Digital Media Processor, User datasheet SPRS821D, 2013.

[83] Intel, *Intel Atom Processor E640,* [Online] URL: http://goo.gl/MwIH74, Consultado em: Maio de 2014.

[84] Allwinner, *Allwinner Technology,* [Online] URL: www.allwinnertech.com/en/clq/processora/A20.html, Consultado em: Maio de 2014.

[85] freescale, *i.MX6Q: i.MX 6Quad Processors – Quad Core, High Performance, Advanced 3D Graphics, HD Video, Advanced Multimedia, ARM Cortex-A9 Core,* [Online] URL: http://goo.gl/DzXLZG, Consultado em: Maio de 2014.

[86] Broadcom, *High Definition 1080p Embedded Multimedia Applications Processor,* [Online] URL: http://www.broadcom.com/products/BCM2835, Consultado em: Maio de 2014.

[87] Ambarella, *A9 Ultra HD 4K Camera SoC,* [PDF] URL: http://goo.gl/hQUs1f, Consultado em: Maio de 2014.

[88] CodeSourcery, Code Sourcery ARM toolchain 2009q1-203, [Online] URL: https://developer.ridgerun.com/wiki/index.php/Code_Sourcery_ARM_toolchain_2009q1-203, Consultado em: Setembro de 2013.

[89] Digital Video Software Development Kit (DVSDK), *DVSDK or EZSDK Installation*, [Online] URL: http://goo.gl/cwuxoD, Consultado em: Setembro de 2013.

[90] RidgeRun, *RidgeRun Download Center*, [Online] URL: http://ridgerun.com/www/index.php/download-center.html, Consultado em: Setembro de 2013.

[91] RidgeRun, *How to boot a board from a SD card,* [Online] URL: https://developer.ridgerun.com/wiki/index.php/How_to_boot_a_board_from_a_SD_card, Consultado em: Setembro de 2013.

[92] RidgeRun, *Setting up Picocom - Ubuntu*, [Online] URL: https://developer.ridgerun.com/wiki/index.php/Setting_up_Picocom_-_Ubuntu, Consultado em: Setembro de 2013.”

[93] J. H. Zar, *Biostatistical analysis*, 4th ed. Londres: Prentice Hall International, 1999, pp. 342–343.

Anexos

Anexo A Algoritmos AWB

A.1 *Gray World*

Este método baseia-se na suposição do mundo cinzento, no qual a intensidade média de uma cena é cinza. Dito de outra forma, assume que a intensidade média dos componentes R, G e B deve ser a mesma, e portanto, a correção das imagens é realizada neste sentido. Este método tem a sua origem no filme fotográfico, onde, para os negativos, a média aproxima-se das regiões escuras da cena, e portanto, tende a ser nula [23]. A sua implementação, analisada mais pormenorizadamente de seguida, consiste basicamente em calcular as intensidades médias de cada canal R, G e B, e aplicar a dois deles um fator de ganho de modo às suas médias tornarem-se iguais à do outro canal, considerado de referência, e que normalmente é o verde.

Considere-se uma imagem a cores com resolução dada por *n* x *n,* e cujos pixéis sejam representados por *RGB* (*x,y*), onde x e y denotam os índices da posição do píxel. Primeiramente, a média de cada um dos canais, R_{avg}, G_{avg}e B_{avg}, deve ser calculada através de

$$\begin{cases} R_{avg} = \frac{1}{n^2}\sum_{x=1}^{n}\sum_{y=1}^{n} R(x,y); \\ G_{avg} = \frac{1}{n^2}\sum_{x=1}^{n}\sum_{y=1}^{n} G(x,y)\,; \\ B_{avg} = \frac{1}{n^2}\sum_{x=1}^{n}\sum_{y=1}^{n} B(x,y)\,. \end{cases} \tag{A.1}$$

Naturalmente que, se os três valores calculados forem iguais, a imagem já satisfaz a suposição *Gray World,* e portanto não são necessárias correções, o que não é comum acontecer.

De seguida, considerando o canal G como referência, é necessário calcular o ganho a aplicar aos canais *R* e *B*, chamemos C_R e C_B, respetivamente, o que pode ser realizado pelas equações

$$\begin{cases} C_R = \dfrac{G_{avg}}{R_{avg}}; \\ C_B = \dfrac{G_{avg}}{B_{avg}}. \end{cases} \tag{A.2}$$

Estes ganhos, aplicados aos devidos canais, resultam em novos valores para as intensidades dos canais R, G e B, gerando assim uma nova imagem. Os novos canais R, G e B, chamemos R_n, G_ne B_n, são portanto dados por

$$\begin{cases} R_n(x,y) = C_R \times R(x,y); \\ G_n(x,y) = G(x,y); \\ B_n(x,y) = C_B \times B(x,y). \end{cases} \tag{A.3}$$

Normalmente, os valores resultantes encontram-se dentro da gama definida para este espaço de cor (na maior parte dos casos é de 0 a 255), no entanto, se se detetar que estes limites não estão sendo respeitados, todos os três canais devem ser escalados pelo mesmo fator de modo a que as intensidades médias se mantenham iguais [21].

Tal como se referiu anteriormente, este método é muito eficiente na prática, exceto para situações em que uma determinada cor domina a imagem. Existem algumas variantes deste método que permitem lidar melhor com estas situações. Na variante sugerida em [10], por exemplo, é definida uma região no plano formado por $(R_{avg} - G_{avg}, B_{avg} - G_{avg})$, em que, se o conjunto $\{R_{avg}, B_{avg}, G_{avg}\}$ estiver incluído, indica que a imagem é suficientemente boa e não necessita das alterações provocadas por este método.

A.2 *White patch*

Este método é baseado na teoria *Retinex* da constância da cor visual [23], a qual argumenta que o branco percebido está associado aos sinais de cone máximos do sistema visual humano [23], e gera o sinal máximo na câmara para todos os três canais de cor [68]. Este também é conhecido por suposição *white world,* isto porque normalmente o ponto mais brilhante na imagem refere-se ao reflexo de uma superfície brilhante, a qual tende a refletir a cor atual da fonte de luz.

Como tal, o método de balanceamento de brancos procura equalizar os valores máximos dos canais vermelho, verde e azul para produzir uma mancha branca, tendo como referência a intensidade máxima do canal verde [21] [68]. Para evitar distúrbios no cálculo, causados por alguns pixéis brilhantes, uma possibilidade é fazer o tratamento por clusters de pixéis ou passar a imagem por um filtro passa

baixo [23]. O algoritmo começa, portanto, por encontrar o conjunto $\{R_{max}, G_{max}, B_{max}\}$, através de

$$\begin{cases} R_{max} = \max\limits_{x,y}\{R(x,y)\}; \\ G_{max} = \max\limits_{x,y}\{G(x,y)\}; \\ B_{max} = \max\limits_{x,y}\{B(x,y)\}. \end{cases} \tag{A.4}$$

Estes valores são utilizados para calcular os coeficientes a aplicar aos canais vermelho e azul, C_R e C_B, respetivamente, enquanto o canal verde permanece inalterado, uma vez que é a referência. Tais coeficientes são dados por [20]

$$\begin{cases} C_R = \dfrac{G_{max}}{R_{max}}; \\ C_G = 1; \\ C_B = \dfrac{G_{max}}{B_{max}}. \end{cases} \tag{A.5}$$

A imagem corrigida resulta da aplicação destes coeficientes a cada um dos canais de cor, ou seja

$$\begin{cases} R_n = C_R \times R; \\ G_n = C_G \times G; \\ B_n = C_B \times B. \end{cases} \tag{A.6}$$

Tal como acontece para o método Gray world, existem diferentes formas de implementação do método White Patch. Em [69] é ainda apresentada uma forma de aplicação desta técnica no espaço de cor YCbCr, denominada por WPYCC (White patches in YCbCr). Nesta, a imagem é convertida para o espaço de cor YCbCr, e os fatores de correção são obtidos através de

$$\begin{cases} C_R = \dfrac{Y_{max}}{R_{avg}}; \\ C_G = \dfrac{Y_{max}}{G_{avg}}; \\ C_B = Y_{max}/B_{avg}, \end{cases} \tag{A.7}$$

onde Y_{max} é o valor máximo da intensidade no espaço de cor YCbCr.

A.3 *Gray world* e *White patch* combinados

Como já se referiu, ambos os métodos *gray world* e *white patch* são simples e eficientes, no entanto cada um deles é mais adequado para determinadas correções que o outro, e vice-versa. Assim sendo, é admissível acreditar-se que, satisfazendo as condições de ambos os métodos, poder-se-á aplicar nas imagens as correções

introduzidas por cada um deles, e portanto, obter-se melhores imagens que aquelas obtidas pela aplicação de apenas um dos métodos. Por outro lado, é necessário ter-se em conta que para muitas imagens, os dois métodos produzem resultados diferentes, pois é difícil a imagem corrigida satisfazer ambas as suposições. Note-se também que em ambos os métodos as imagens corrigidas são resultantes da aplicação de ajustes lineares às intensidades dos pixéis. Para além disso existe um ponto fixo nos mapeamentos, o qual permanece inalterado perante a aplicação de qualquer um dos dois métodos: os pixéis com intensidade nula. Evidentemente, é difícil respeitar os requisitos de ambos os métodos com uma técnica linear [20].

Como tal, em [21] é apresentada uma implementação em que as intensidades são ajustadas através de um método quadrático. Neste, o ajuste no canal vermelho é calculado por

$$R_n(x,y) = \mu R^2(x,y) + vR(x,y), \tag{A.8}$$

onde μ e v são parâmetros por descobrir. O ajuste no canal azul é calculado analogamente. Para satisfazer a suposição do *gray world*, é requerido que

$$\sum_{x=1}^{n}\sum_{y=1}^{n} R(x,y) = n^2 G_{avg}, \tag{A.9}$$

e portanto,

$$\mu \sum_{x=1}^{n}\sum_{y=1}^{n} R^2(x,y) + v \sum_{x=1}^{n}\sum_{y=1}^{n} R(x,y) = n^2 G_{avg}. \tag{A.10}$$

Simultaneamente, para satisfazer a suposição *Retinex* para produzir um padrão branco, é necessário que

$$\max_{x,y} R(x,y) = R_{max} = G_{max}, \tag{A.11}$$

e portanto, se se assumir que $R(x,y)$ tem valores inteiros entre 0 e 255, e μ e v são números positivos,

$$\mu \max_{x,y} R^2(x,y) + v \max_{x,y} R(x,y) = G_{max}. \tag{A.12}$$

As equações (10) e (12) juntas formam duas equações com duas variáveis desconhecidas. É possível representá-las numa matriz:

$$\begin{bmatrix} \sum_{x=1}^{n}\sum_{y=1}^{n} R^2(x,y) & \sum_{x=1}^{n}\sum_{y=1}^{n} R(x,y) \\ \max_{x,y} R^2(x,y) & \max_{x,y} R(x,y) \end{bmatrix} \begin{bmatrix} \mu \\ v \end{bmatrix} = \begin{bmatrix} n^2 G_{avg} \\ G_{max} \end{bmatrix}. \tag{A.13}$$

Isto pode ser resolvido analiticamente para u e v usando a regra de *Cramer* [70].

Anexo B Predição com estimação e compensação de movimento baseada em blocos

Este é um método prático e muito usado que procura compensar o movimento de blocos retangulares de amostras entre *frames*. Para cada bloco de *M*x*N* amostras, envolve a estimação do movimento (EM), a compensação do movimento (CM) e um vetor do movimento (VM), respetivamente, da seguinte forma:

1. Para a estimação do movimento, começa por ser realizada uma busca na *frame* de referência por uma região de *M*x*N* amostras semelhante ao bloco de *M*x*N* amostras da *frame* corrente. Em muitos casos, esta semelhança é determinada pela energia do residual resultante da subtração da região da *frame* de referência ao bloco da *frame* corrente, de modo que a melhor correspondência refere-se à região que minimiza esta energia [2];
2. A região selecionada torna-se então o preditor para o bloco atual, e a compensação do movimento é realizada pela subtração desta região ao bloco atual, dando origem ao bloco residual *M*x*N*;
3. Este bloco residual é codificado e transmitido, juntamente com o vetor de movimento que descreve o deslocamento entre a região de predição e o bloco corrente [2].

Existem muitas variações no processo básico de estimação e compensação de movimento. Relativamente à *frame* de referência, pode ser uma *frame* anterior ou seguinte (na ordem temporal), desde que tenha sido codificada anteriormente, ou uma combinação de predições a partir de duas ou mais *frames* codificadas anteriormente. No entanto, quando a diferença entre a *frame* atual e a de referência é significativa (como acontece numa mudança de cena) pode ser mais eficiente codificar o bloco sem compensação de movimento, de modo que o codificador pode escolher entre o modo intra (sem compensação de movimento) e o modo inter (com compensação de movimento) para cada bloco [2].

As dimensões destes blocos também podem variar. A utilização de blocos com menores dimensões pode resultar em residuais também com menor energia, contudo implicam uma maior complexidade de computação (mais operações de procura têm de ser realizadas) e a um maior número de vetores de movimento que têm de ser transmitidos, de modo que uma carga excessiva de informação a ser transmitida pode não compensar a redução de energia residual alcançada [2]. Uma forma de equilibrar estes aspetos refere-se à adaptação do tamanho do bloco de acordo às caraterísticas da imagem, escolhendo, por exemplo, dimensões maiores para as zonas planas e homogéneas da *frame*, e dimensões menores para áreas com elevado detalhe e movimento complexo, o que já é possível no padrão H.264. Os macroblocos, como são conhecidos os blocos de 16x16 píxeis, são os mais

populares, representando a unidade básica para a predição com compensação de movimento numa variedade de padrões de codificação visual importantes [2].

A compensação de movimento baseada em blocos é usado por diversas razões, nomeadamente, pela sua simplicidade e capacidade de ser tratada computacionalmente, por se adaptar bem em *frames* de vídeo retangulares e com transformadas de imagem baseadas em blocos, e por proporcionar um modelo temporal razoavelmente efetivo para muitas sequências de vídeo [2]. Também apresenta alguns inconvenientes, os quais estão relacionados, por exemplo, com o facto de elementos “naturais” raramente terem limites nítidos que correspondam às medidas retangulares, os elementos normalmente moverem-se por um número fracionário de posições de píxeis entre *frames*, e alguns movimentos (como rotação, inclinação e deformação) são difíceis de compensar usando as medidas dos blocos. Apesar disto, a compensação de movimento baseada em blocos é a base do modelo temporal usado por todos os padrões de codificação de vídeo atuais [2].

Anexo C Codificação baseada em transformadas

A codificação baseada na transformada é implementada por alguns padrões de compressão de imagem e vídeo, e envolve a perda de dados. Este processo é realizado em três fases: transformação, quantização e reordenação.

C.1 Transformada

O objetivo da fase da transformada num codec de vídeo ou imagem é representar a imagem ou os dados residuais compensados de movimento no domínio da frequência [1]. Diversas transformadas têm sido propostas para aplicação na compressão de imagem, de modo que alguns critérios devem ser tidos em conta aquando da seleção de uma delas. A transformada selecionada deve garantir que os dados no novo domínio encontram-se descorrelacionados (separados em componentes com a mesma interdependência) e compactados (a maior parte da energia dos dados transformados deve estar concentrada num pequeno número de valores). A transformada também deve ser reversível, de modo a que os dados possam ser representados novamente no domínio espacial, e deve ser computacionalmente tratável, ou seja, deve ter em conta aspetos como baixos requisitos de memória, aritmética de precisão limitada e um pequeno número de operações aritméticas.

Normalmente são usadas transformadas baseadas em blocos ou baseadas em imagem. As baseadas em blocos atuam em blocos de *N*x*N* amostras de uma imagem ou residual, estando portanto, bem adaptadas à compressão de residuais de compensação de movimento baseados em blocos. Também têm baixos requisitos de memória, no entanto tendem a apresentar blocagem, um artefacto que surge nos limites dos blocos (ver Figura 2.3). Já as baseadas em imagem atuam numa *frame* ou imagem inteira (ou numa grande porção da imagem, conhecida por *tile*). Estas têm apresentado melhor desempenho que as baseadas em blocos na compressão de imagem e não estão associadas ao artefacto de blocagem, contudo tendem a ter maiores requisitos de memória uma vez que a unidade de tratamento é uma imagem inteira ou um *tile.*

De entre as transformadas baseadas em blocos mais populares encontram-se a transformada discreta de *Fourier* (DFT), a transformada Karhunen-Loeve (*Karhunen-Loeve Transform*, KLT), a transformada Walsh-Hadamard (*Walsh-Hadarmd Transform*, WHT) e a transformada discreta do cosseno (*Discrete Cosine Transform,* DCT). Do ponto de vista computacional, a WHT é das mais simples para implementar, enquanto a KLT é das mais complexas. A complexidade associada à KLT deve-se à dependência desta relativamente aos dados de entrada, fazendo com

que seja raramente usada para compressão de imagem. Por outro lado, a KLT é a transformada ótima relativamente ao empacotamento da informação, pois é a que minimiza o erro quadrático médio (ver secção 2.2.4) para qualquer imagem de entrada e qualquer número de coeficientes retidos [10]. Quanto à DCT, apesar da sua habilidade em empacotar dados não superar a da KLT, é superior àquela proporcionada tanto pela DFT como pela WHT [10]. Esta transformada apresenta um bom compromisso entre complexidade computacional e habilidade em empacotar dados, sendo também a mais utilizada para compressão de imagem. Uma outra caraterística que torna esta transformada mais adequada que as restantes é que permite minimizar o efeito de blocagem.

Relativamente às dimensões dos blocos, quanto maiores estas forem, maiores serão também o grau de compressão e a complexidade computacional. As dimensões mais populares são 8x8 e 16x16 amostras [10].

Quanto às transformadas baseadas em imagem, a mais popular é a DWT ou simplesmente *wavelet*. As *wavelets* são funções que permitem decompor os sinais e representá-los simultaneamente no domínio dos tempos e no das frequências [25]. Estas são equivalentes a conjuntos de filtros, e são as bases da transformada *wavelet*. A decomposição de sinais bi-dimensionais pode ser realizada de duas formas: usando filtros bi-dimensionais, ou transformadas individuais que podem ser implementadas usando filtros uni-dimensionais primeiro nas linhas e depois nas colunas (ou vice-versa). A maior parte das abordagens, como o padrão JPEG2000, usa a segunda implementação.

A DWT permite atingir a mesma qualidade de imagem como a DCT com rácios de compressão muito mais elevados [1]. A DCT e a DWT são ambas incorporadas no MPEG-4 Visual, e uma variante da DCT é incorporada no H.264 [2].

C.2 Quantização

A quantização refere-se ao processo de representar ou mapear um conjunto de valores num outro muito mais pequeno [25]. Esta é uma forma de reduzir a redundância percetiva, pela remoção dos coeficientes pouco significativos que transportam pouca informação, como aqueles resultantes da DCT ou *wavelet* próximos de zero [2]. É uma operação irreversível, resultando portanto, nalguma perda de informação. Evidentemente, este bloco não é incluído nos sistemas de compressão sem perdas.

A quantização pode ser escalar ou vetorial. A quantização escalar mapeia uma amostra do sinal de entrada num valor de saída quantizado [2]. Um exemplo

simples de quantização escalar consiste em arredondar um número fracionário para o número inteiro mais próximo, passando do conjunto $\mathbb{R}$ para o conjunto $\mathbb{Z}$. É fácil entender que este processo é irreversível, com perdas, uma vez que não é possível determinar qual o valor fracionário que deu origem ao inteiro arredondado [2]. Assim sendo, uma "quantização inversa" no descodificador, ao invés de recuperar os valores, faz uma espécie de reescalonamento dos mesmos.

Numa quantização uniforme, os níveis de saída quantizados estão espaçados em intervalos uniformes, os quais são chamados passos de quantização [2]. Um exemplo mais geral de uma quantização uniforme é

$$FQ = round\left(\frac{X}{QP}\right), \tag{C.1}$$

onde QP é um tamanho de passo de quantização, X é o valor a quantizar e FQ o valor quantizado. A quantização ilustrada na Figura C.1 não só é uniforme, como também é linear, uma vez que tem um mapeamento linear entre os valores de entrada e de saída.

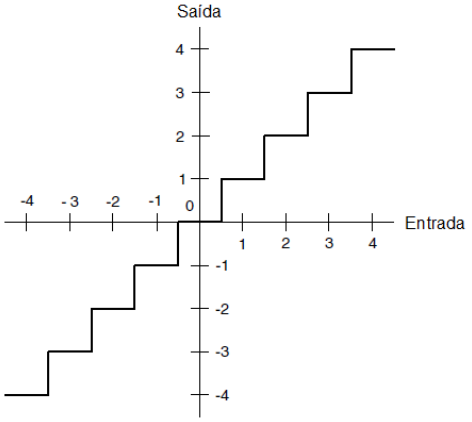
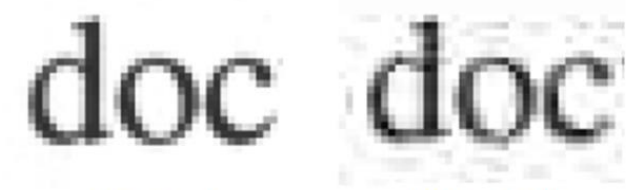


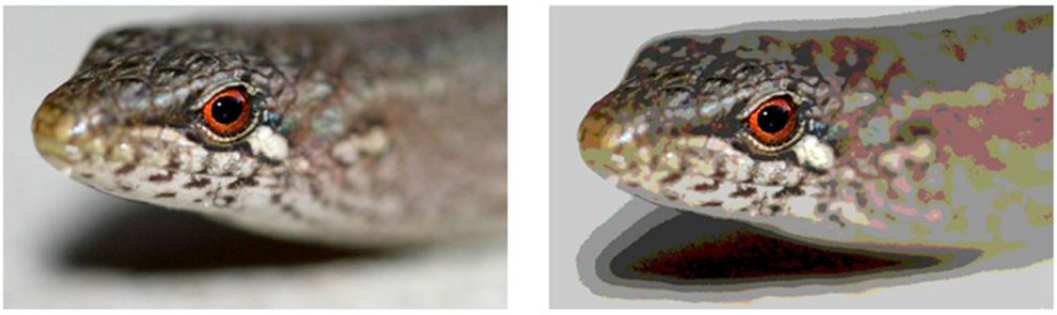
Figura C.1. Quantização uniforme e linear.

O tamanho do passo de quantização é um parâmetro crítico que tem grande influência no processo de quantização. Um passo de quantização largo dá origem a uma pequena gama de valores quantizados uma vez que os coeficientes DCT tornam-se mais próximos uns dos outros, e portanto, a uma eficiente compressão [71]. Contudo os valores reescalonados são uma aproximação grosseira ao sinal original, podendo introduzir artefactos na imagem reconstruída. Para além do artefacto de blocagem referido anteriormente, outros típicos artefactos são *ringing* (Figura C.2) e posterização (Figura C.3) [72], [73].



(a) Original (b) Com *ringing*

Figura C.2. Artefato *ringing* [71].



(a) Original (b) Com posterização

Figura C.3. Artefato posterização [74].

Já um passo de quantização pequeno dá origem a valores reescalonados mais próximos dos do sinal original, contudo a maior gama de valores quantizados reduz a eficiência de compressão [2].

Uma quantização vetorial mapeia um grupo de amostras de entrada (como um bloco de amostras de imagem) num único valor, ou grupo de valores quantizados (um vetor) [2].

C.3 Reordenação

A reordenação é realizada para agrupar os coeficientes quantizados diferentes de zero e representar os coeficientes de valor nulo eficientemente, permitindo uma maior compactação aquando da codificação da entropia. O caminho de reordenação ótimo depende essencialmente da distribuição dos coeficientes de valor zero.

No caso de um típico bloco de coeficientes provenientes da aplicação da DCT, uma adequada varredura é aquela conhecida por *zigzag.* Neste tipo de varredura os coeficientes são copiados para um *array* pela ordem ilustrada na Figura C.4, começando sempre pelo coeficiente DC. Desta forma, os coeficientes diferentes de zero tendem a ficar agrupados no início do *array* reordenado, seguidos por longas sequências de zeros [2].

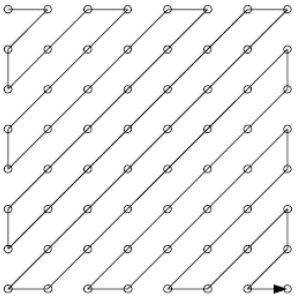


Figura C.4. Ordenação pela varredura *zigzag*.

Estas longas sequências de zeros podem ser codificadas de forma a alcançar uma representação mais compacta, o que pode ser realizado através da codificação *run-level*. Esta codificação permite representar o *array* de valores numa série de pares *(run, level)*, em que *run* indica o número de zeros precedendo um coeficiente diferente de zero e *level* indica a magnitude do coeficiente diferente de zero [2]. É necessária também uma forma de indicar qual o ultimo coeficiente diferente de zero. No caso de ser utilizada uma codificação *run-level* "bi-dimensional", tal é realizado usando um símbolo de código separado, "*last".* Se for utilizada codificação *run-level* "tri-dimensional", cada símbolo codifica três quantidades, *(run, level, last)*. No exemplo que se segue, em que é utilizada codificação "tri-dimensional", o valor 1 em "*last*" do código final indica que esse é o último coeficiente diferente de zero no bloco.

Sequência de entrada: 16,0,0,−3,5,6,0,0,0,0,−7;

Símbolos de saída: (0, 16, 0), (2,−3, 0), (0, 5, 0), (0, 6, 0), (4,−7, 1).

Num sinal decomposto com a DWT, é possível considerar uma "árvore" de coeficientes quantizados diferentes de zero, começando por uma "raiz" na sub-banda de mais baixa-frequência, como indica a Figura C.5 [2]. Um coeficiente individual na banda LL da camada 1 tem um coeficiente correspondente em cada uma das outras bandas da camada 1 (isto é, estes 4 coeficientes correspondem à mesma região na imagem original). Por sua vez, cada um destes três coeficientes da camada 1, mapeia 4 posições de coeficientes "filho" correspondentes em cada sub-banda na camada 2 (note-se que as sub-bandas da camada 2 têm o dobro da resolução horizontal e vertical que as sub-bandas da camada 1).

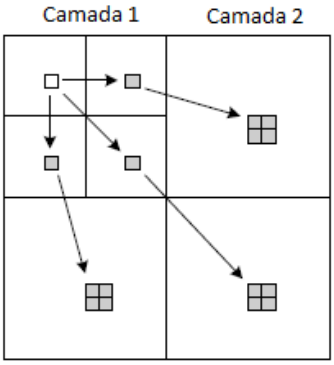


Figura C.5 Coeficientes wavelet e respetivos "filhos".

Uma forma eficiente de codificar estes coeficientes é codificando cada árvore de coeficientes diferentes de zero começando pelo mais baixo nível de decomposição (raiz). Um coeficiente na camada mais baixa é codificado, seguido pelos seus coeficientes filho na próxima camada superior, e assim sucessivamente. O processo de codificação continua até a árvore alcançar um coeficiente de valor zero. Como os coeficientes filho de um coeficiente 0 tendem a ser também 0, os coeficientes com este valor tendem a dar origem a sequências de 0s que podem ser codificados como uma árvore de zeros (*zerotree*). O descodificador reconstrói o mapa de coeficientes começando na raiz de cada árvore, os coeficientes diferentes de zero são descodificados e reconstruidos, e quando uma *zerotree* é alcançada, todos os "filhos" restantes são colocados a zero. Esta é a base do método árvore de zeros embebida (*embedded zerotree wavelet*, EZW) de codificação de coeficientes *wavelet* [2].

Anexo D Técnicas de codificação da entropia

Neste anexo são descritas algumas técnicas utilizadas pelos padrões de compressão de imagem e vídeo para a codificação da entropia.

D.1 Codificação preditiva

Nesta fase do processo de compressão, a predição pode ser usada para explorar a possível correlação, por exemplo, entre o valor DC de blocos vizinhos, e entre vetores de movimento vizinhos.

No caso dos vetores de movimento, os associados a blocos vizinhos estão frequentemente correlacionados porque o movimento de objetos pode se estender ao longo de largas regiões de uma *frame*. Isto é especialmente aplicado a tamanhos de blocos pequenos (ex., vetores de blocos 4x4) e/ou objetos com muito movimento. Uma predição simples para o vetor do bloco atual consiste em usar como preditor o bloco horizontalmente adjacente, ou mesmo outros vetores previamente codificados. A diferença entre o vetor de movimento predito e o atual, conhecida como diferença entre vetores de movimento (*motion vector difference*, MVD), é codificada e transmitida [2].

O parâmetro de quantização também pode ser codificado desta forma. Como as possíveis alterações no valor do parâmetro de quantização aplicado numa sequência de vídeo normalmente são reduzidas, ao invés de transmitir o novo valor, é preferível enviar a diferença indicando a mudança necessária [2].

D.2 Codificação de comprimento variável

Num codificador de comprimento variável os símbolos são codificados em palavras de código (*codewords*) cujo comprimento é variável. Neste tipo de codificação os símbolos com menor probabilidade de ocorrência são mapeados em palavras de código de maior comprimento, enquanto aqueles que ocorrem com maior probabilidade são mapeados em palavras código mais curtas [2]. A codificação *Huffman* e a codificação aritmética são duas técnicas bem conhecidas para a construção de tais códigos [3].

D.2.1 Codificação Huffman

A técnica de codificação *Huffman* permite criar códigos com o mais pequeno comprimento médio possível, que se aproxima da entropia da fonte, dado um conjunto de símbolos e a probabilidade de ocorrência de cada um deles [3]. Note que cada símbolo tem de ser codificado individualmente, ou seja, a codificação ocorre para um símbolo de cada vez, sendo que os dois símbolos com menor

probabilidade de ocorrência têm palavras de código do mesmo comprimento, diferindo apenas no *bit* menos significativo. Os códigos *Huffman* podem ser mapeados numa árvore binária, popularmente conhecida como árvore *Huffman*. Contudo, quando a codificação tem de ser aplicada a um elevado número de símbolos, a construção de um código *Huffman* ótimo pode tornar-se uma tarefa relativamente complexa, sendo que, nalguns casos, é preferível sacrificar alguma eficiência de codificação de modo a reduzir a complexidade computacional. Existem outros códigos de comprimento variável que têm em conta este compromisso, tais como o código binário natural e o código de *Huffman* truncado [10].

Do ponto de vista de um codec de vídeo, a codificação de *Huffman* também apresenta algumas desvantagens. Uma delas tem a ver com o facto de que o descodificador tem de usar o mesmo conjunto de palavras de código que o codificador, o que implica que a informação contida na tabela de probabilidades tenha de ser transmitida também. Tal provoca o aumento da carga a ser transmitida, reduzindo assim a eficiência de compressão, principalmente para sequências de vídeo pequenas. Um outro problema, associado principalmente a sequências de vídeo, é que os dados não podem ser codificados sem que a tabela de probabilidades tenha sido calculada, uma vez que tal pode introduzir um atraso inaceitável no processo de codificação [2]. Por estas razões, alguns padrões de codificação de imagem e vídeo têm optado pela codificação baseada em *Huffman* pré-calculada, em que definem conjuntos de palavras de código baseados nas distribuições de probabilidades de material de vídeo “genérico”.

Os códigos baseados em *Huffman* apresentam mais uma desvantagem, que se reflete aquando da transmissão dos dados codificados. Estes códigos são sensíveis a erros de transmissão, de modo que um erro duma sequência de códigos pode causar a dessincronização no descodificador, e portanto, outros erros na descodificação dos códigos subsequentes. Os códigos de comprimento variável reversíveis foram desenvolvidos no sentido de minimizar estes erros, ao permitirem a descodificação em ambas as direções direta e inversa [2]. Uma abordagem alternativa é usar códigos que podem ser gerados automaticamente (‘*on the fly*’) se o símbolo de entrada for conhecido. Os códigos *Exponential Golomb* (*Exp-Golomb*) estão nesta categoria.

D.2.2 Codificação aritmética

Os códigos de comprimento variável referidos anteriormente atribuem, a cada símbolo, uma palavra de código que contém um número inteiro de *bits*. Esta é uma caraterística que os torna sub-ótimos uma vez que o número ótimo de *bits* para um

símbolo depende do conteúdo da informação e é normalmente um número fracionário [2]. A codificação aritmética, por outro lado, converte uma sequência de símbolos de dados num único número fracionário e permite a aproximação ao número de *bits* fracionário ótimo requerido para representar cada símbolo, superando assim o desempenho da codificação *Huffman* [2].

Neste tipo de codificação uma sequência de símbolos de entrada é representada por um intervalo de números reais entre 0,0 e 1,0. Quanto maior a mensagem, menor se torna o intervalo para representar a mesma, sendo necessário um maior número de *bits* para a sua codificação [3]. A probabilidade de ocorrência dos símbolos também afeta o número de *bits* para a codificação, sendo que os símbolos mais prováveis reduzem menos o intervalo que os símbolos menos prováveis.

Existe uma variante desta codificação, a codificação aritmética baseada em contexto (*context based arithmetic encoding*, CAE), que procura aumentar a eficiência de codificação através da formulação de modelos precisos de probabilidades dos símbolos. Mais precisamente, esta técnica usa as caraterísticas espaciais e/ou temporais locais para estimar a probabilidade de um símbolo ser codificado [2]. A CAE é usada em padrões como o JBIG, o MPEG-4 Visual e o H.264.

São essencialmente dois os fatores que afetam o desempenho desta codificação: a adição de indicadores de fim-de-mensagem que são necessários para separar uma mensagem de outra, e o uso de aritmética de precisão finita [10]. Algumas implementações têm sido desenvolvidas no sentido de contornar o obstáculo relativo à precisão da aritmética.

Os inconvenientes desta abordagem estão relacionados com a maior complexidade computacional e a menor imunidade a erros relativamente à codificação *Huffman* [3].

D.3 Codificação *Lempel-Ziv-Welch* (LZW)

Neste tipo de codificação são atribuídas palavras de código de comprimento fixo a sequências de símbolos de comprimento variável, no entanto não requer um conhecimento à priori das probabilidades de ocorrência dos símbolos [10]. Para tal, faz uso de um livro de códigos ou “dicionário” com os símbolos a serem codificados, que constrói no início do processo de codificação, possivelmente ao mesmo tempo que realiza a codificação dos dados. O descodificador funciona de forma

semelhante, no sentido de que constrói o dicionário ao mesmo tempo de descodifica o fluxo de dados codificado.

O tamanho do dicionário é um importante parâmetro do sistema, pois se for muito pequeno, a deteção de sequências correspondentes será mais difícil, e se for muito grande, o tamanho das palavras de código irá adversamente afetar o desempenho da compressão. Diversas abordagens têm sido desenvolvidas para controlar este parâmetro [10].

A simplicidade conceptual associada à codificação LZW também contribui para a sua integração numa variedade de formatos de ficheiros de imagem convencionais tais como *graphic interchange format* (GIF), *tagged image file format* (TIFF) e *portable document format* (PDF) [10].

D.4 Codificação *run-length* (RLE)

A codificação *run-length* é uma abordagem simples e adequada quando existe uma longa sequência dos mesmos dados, de uma forma consecutiva, num conjunto de dados [3]. Ao invés de codificar cada amostra na sequência individualmente, os dados podem ser representados compactamente simplesmente pela indicação do valor da amostra e do comprimento da sua sequência. A codificação *run-level* referida no Anexo C é uma variante desta técnica.

Nalguns casos poderá ser necessário reordenar os dados (por exemplo, pela ordem *zigzag* ou através de uma *zerotree*) de modo a facilitar a codificação *run-length*.

Anexo E Padrão JPEG

O JPEG é um dos padrões mais utilizados atualmente para compressão de imagens de qualidade fotográfica. Este define quatro modos de operação [3]:

- Modo sequencial sem perdas: comprime a imagem numa única varredura, e a imagem descodificada é uma réplica exata da imagem original;
- Modo sequencial baseado na DCT: comprime a imagem numa única varredura usando a técnica de compressão com perdas baseada na DCT. Como resultado, a imagem descodificada não é uma réplica exata, mas uma aproximação à imagem original;
- Modo progressivo baseado na DCT: comprime a imagem em múltiplas varreduras e também descomprime a imagem em múltiplas varreduras com cada varredura sucessiva produzindo uma qualidade de imagem melhorada.
- Modo hierárquico: comprime a imagem em múltiplas resoluções para mostrar em dispositivos diferentes.

Os três modos JPEG baseados na DCT proporcionam compressão com perdas devido à limitação de precisão para calcular digitalmente a DCT (e a sua inversa), e ao processo de quantização que introduz distorção na imagem reconstruída. Para o modo de compressão sequencial sem perdas, a codificação preditiva é usada ao invés da transformada baseada na DCT, e também não existe quantização envolvida neste modo [3]. O modo hierárquico usa extensões da técnica de codificação baseada na DCT como também da técnica de codificação preditiva. A forma mais simples do algoritmo JPEG sequencial baseado na DCT é chamada de algoritmo *JPEG baseline,* que é baseada na codificação *Huffman* para a codificação de entropia. A outra forma do algoritmo JPEG baseado na DCT sequencial faz uso da codificação aritmética para a codificação da entropia. O algoritmo JPEG *baseline* é largamente usado na prática [3], razão pela qual é descrito com mais detalhe.

Caso a imagem alvo da compressão esteja no espaço de cor RGB, o processo de compressão JPEG *baseline* começa por reduzir a correlação espacial entre as componentes de cor, através da conversão para um espaço de cor descorrelacionado, como o YUV ou o YCbCr [3], [59]. O método tipicamente utilizado para tal é pela aplicação da equação (2.7) [3]. Nestes espaços de cor é possível reduzir ainda mais a quantidade de dados devido aos canais de crominância conterem muita informação redundante que pode ser facilmente subamostrada sem uma perda significativa de qualidade visual na imagem reconstruida (ver secção 2.1.3.3).

O algoritmo de compressão JPEG *baseline* segue os princípios de codificação de transformada baseada em blocos. Quando se trata de uma imagem em escala de

cinza, ou seja, com apenas uma componente, os procedimentos aplicados são os ilustrados no diagrama de blocos da Figura E.1, em que cada bloco de 8x8 amostras é codificado separadamente. Este diagrama de blocos também aplica-se a imagens de cor, com todas as componentes divididas em blocos de 8x8, sendo os blocos das diferentes componentes tratados segundo uma determinada ordem. Os blocos das diferentes componentes podem ser tratados segundo uma ordem intercalada, na qual a componente a que pertence cada bloco é variável, ou segundo uma ordem não intercalada, na qual a componente a que pertence cada bloco não muda enquanto todos os blocos dessa componente não forem todos tratados, ou seja, cada componente é tratada separadamente das outras [3].

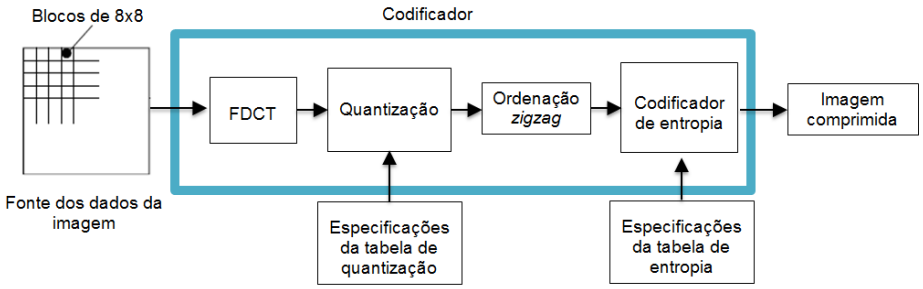


Figura E.1 Codificação JPEG no modo *baseline.*

Em cada bloco, o valor de cada amostra começa por ser deslocado de modo a convertê-lo num inteiro com sinal, o que é realizado subtraindo 128 ao valor de cada amostra [3], [75].

O processo realizado, passo a passo, durante a compressão JPEG *baseline* de três diferentes blocos de 8×8 é apresentado no Anexo O**.**

E.1 Compressão JPEG *baseline*

E.1.1 Transformada

Os valores deslocados são então convertidos para o domínio da frequência ao ser aplicada a DCT direta, ou *Forward DCT* (FDCT). A FDCT é aplicada a um bloco X de $N \times N$ amostras e transforma-o num bloco Y de $N \times N$ coeficientes através da equação [2]

$$Y_{uv} = C_u C_v \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)v\pi}{2N} \cos\frac{(2i+1)u\pi}{2N}, \qquad \text{(E.1)}$$

na qual

$$C_i = \sqrt{\frac{1}{N}}(i = 0), \quad C_i = \sqrt{\frac{2}{N}}(i > 0), \tag{E.2}$$

com $N = 8$. O primeiro coeficiente, na primeira linha e primeira coluna de um bloco $N \times N$, é chamado de coeficiente DC, enquanto os restantes são coeficientes AC [75].

E.1.2 Quantização

O próximo passo consiste na quantização dos coeficientes. Tal é realizado recorrendo a tabelas de quantização de 8x8 elementos, os quais são números inteiros que variam entre 1 e 255, e representam os tamanhos dos passos de quantização [75]. O padrão JPEG não define qualquer tabela de quantização como sendo fixa, contudo, como referência, apresenta em [75] a Tabela K.1 para a quantização dos coeficientes de luminância, e a Tabela K.2 para quantização dos coeficientes de crominância. É indicado ainda que se os valores nestas tabelas forem divididos por 2, a imagem reconstruida resultante é normalmente quase indistinguível da original [75].

Os coeficientes quantizados, $Y_q(u,v)$, são obtidos dividindo cada coeficiente DCT, $Y(u,v)$, pelo parâmetro de quantização correspondente, $Q(u,v)$, e arredondando o resultado para o inteiro mais próximo [60], ou seja

$$Y_q(u,v) = round\left(\frac{Y(u,v)}{Q(u,v)}\right). \tag{E.3}$$

E.1.3 Ordenação zigzag e codificação run-level

Após a quantização, os coeficientes DC quantizados são codificados com codificação diferencial. Ou seja, o coeficiente DC_i do bloco corrente é subtraído pelo coeficiente DC_{i-1} do bloco anterior, e a diferença

$$DIFF_i = DC_i - DC_{i-1}. \tag{E.4}$$

é codificada. Quanto aos coeficientes AC, como num bloco de 8x8 grande parte deles encontra-se a zero devido à quantização, estes são ordenados conforme a ordem *zigzag,* de forma a agrupar os coeficientes de baixa frequência, e originar longas sequências de zeros correspondentes aos coeficientes quantizados de maiores frequências [3].

De forma a representar os coeficientes DC diferenciais e as sequências de coeficientes AC de uma forma mais compacta, antes da codificação da entropia é realizada ainda codificação *run-level.* Esta codificação é ligeiramente diferente para

os coeficientes DC e para os AC. No caso dos coeficientes AC, na sequência ordenada segundo o esquema *zigzag*, cada coeficiente AC diferente de zero é representado numa combinação com o "*runlength"* (número consecutivo) de coeficientes AC de valor zero que o precedem na sequência. Cada combinação *runlength*/coeficiente diferente de zero é normalmente representada por um par de símbolos:

- Símbolo1: (*RUNLENGTH*,COMPRIMENTO);
- Símbolo2: (AMPLITUDE).

No Símbolo1, em *RUNLENGTH* é indicado o comprimento da sequência de zeros precedentes ao coeficiente AC diferente de zero*,* e em COMPRIMENTO é indicado o número de *bits* usados para codificar o valor de AMPLITUDE, ou seja, para codificar o Símbolo2 através de um método de codificação *Huffman* particular [32]. Por sua vez, AMPLITUDE, no Símbolo2, é a amplitude do coeficiente AC diferente de zero após a sequência de zeros. *RUNLENGHT* representa sequências de zeros de comprimento desde 0 a 15, contudo, como uma sequência de zeros pode ser maior que 15, o Símbolo1 de valor (15,0) é usado para representar uma sequência de 16 zeros. Cada Símbolo1 pode ter até três extensões consecutivas (15,0), o qual é sempre seguido por um único Símbolo2, com exceção para o caso em que a última sequência de zeros inclui o último coeficiente AC do bloco. Neste caso, o Símbolo1 de valor específico (0,0) é usado para indicar o fim do bloco (*end of block,* EOB) [32].

Assim, para cada bloco de amostras 8x8, a sequência *zigzag* dos 63 coeficientes AC quantizados é representada como uma sequência de pares de símbolos Símbolo1, Símbolo2, sendo que cada "par" pode ter repetições do Símbolo1 no caso de uma longa sequência, ou apenas um Símbolo1 no caso de um EOB. A codificação de AMPLITUDE do Símbolo2 é realizada com 1 a 10 *bits*, de modo que COMPRIMENTO varia entre 1 e 10, e *RUNLENGTH* apresenta um valor entre 0 e 15 [32].

Já o coeficiente DC de um bloco de 8x8 amostras é codificado com um Símbolo1 que apenas representa a informação COMPRIMENTO, e um Símbolo2 apresentando a informação AMPLITUDE:

- Símbolo1: (COMPRIMENTO);
- Símbolo2: (AMPLITUDE).

O valor diferencial usado para a codificação do coeficiente DC encontra-se na gama de $[-2^{11}, 2^{11-1}]$, de modo que o Símbolo1 para os coeficientes DC apresenta um valor entre 1 e 11 [32].

E.1.4 Codificação da entropia

O padrão JPEG indica duas opções para a codificação da entropia. No modo sequencial *baseline* apenas é possível utilizar a codificação *Huffman*, enquanto no modo progressivo baseado na DCT e no modo sequencial sem perdas é possível utilizar codificação *Huffman* ou codificação aritmética [75]. No modo *baseline* é possível utilizar até duas tabelas *Huffman* para a codificação dos coeficientes DC e duas tabelas *Huffman* para a codificação dos coeficientes AC [75]. O padrão indica possíveis tabelas como referência – Tabelas K.3 e K.4 apresentadas em [75] para os coeficientes DC, e Tabelas K.5 e K.6 do mesmo documento para os coeficientes AC –, contudo não define nenhuma como fixa, podendo a implementação do codificador ser realizada com outras tabelas, sendo que as mesmas têm de ser especificadas no cabeçalho do ficheiro comprimido [75].

E.2 Descompressão

A descompressão é o processo inverso para descodificar o fluxo de *bits* comprimidos de modo a reconstruir apropriadamente a imagem. A Figura E.2 mostra os principais procedimentos no processo de descodificação JPEG no modo *baseline.* Cada passo apresentado mostra essencialmente o inverso do seu principal procedimento correspondente no codificador. O descodificador de entropia descodifica a sequência *zigzag* de coeficientes DCT quantizados. Depois da "desquantização" os coeficientes DCT são transformados num bloco de 8x8 amostras através da DCT inversa, ou IDCT (*Inverse DCT*) [75]. A IDCT é dada por [2]

$$X_{ij} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C_u C_v Y_{uv} \cos\frac{(2j+1)v\pi}{2N} \cos\frac{(2i+1)u\pi}{2N}. \quad \text{(E.5)}$$

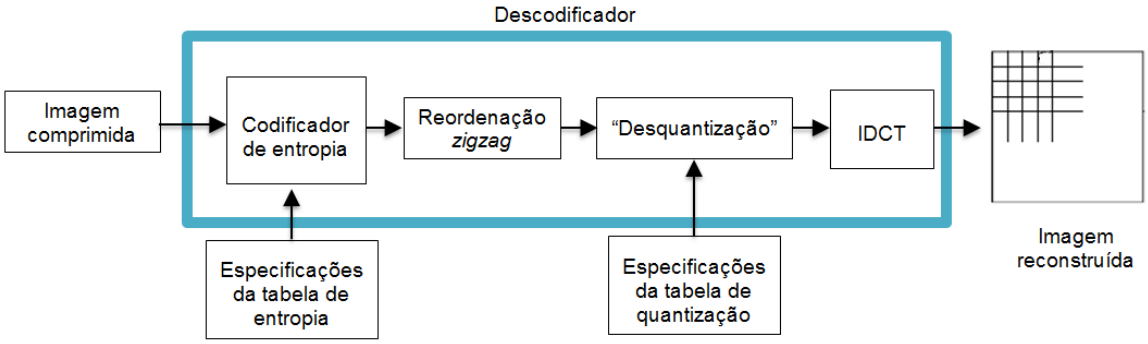


Figura E.2 Descodificação JPEG no modo *baseline.*

Anexo F Compressão H.264/MPEG-AVC

O H.264/MPEG-AVC possui flexibilidade para suportar uma larga gama de aplicações com requisitos de taxas de *bits* bastante diferentes. Este padrão inclui elementos funcionais que são comuns a outros padrões de compressão de vídeo, sendo os detalhes associados a cada elemento que marcam a diferença no H.264. Alguns desses detalhes são apresentados na descrição que se segue.

F.1 Perfis e níveis

As capacidades de compressão do padrão H.264 estão associadas a um total de três perfis: *Baseline, Main* e *Extended*. Cada perfil define um conjunto particular de algoritmos que devem ser usados na compressão, os quais determinam a complexidade de implementação do descodificador. Tal complexidade, bem como as caraterísticas associadas aos algoritmos utilizados, tornam um dado perfil mais adequado a um dado tipo de aplicações do que a outros. O perfil *Baseline,* por exemplo, é mais adequado para aplicações com recursos de computação limitados e aplicações em tempo-real, como videochamadas e videoconferência [24]. O perfil *Extended* apresenta-se como uma extensão do *Baseline* mas é mais resistente a erros, sendo adequado a aplicações de *streaming*. Já o perfil *Main* está direcionado para aplicações como *broadcast* de tv e armazenamento de vídeo [57].

Os limites de desempenho para os codecs são definidos por um conjunto de 11 níveis, cada um colocando limites nos parâmetros como taxa de processamento de amostras, tamanho da imagem, taxa de *bits* codificados e requisitos de memória [24]. Quanto maior a resolução das imagens, maior o nível necessário.

F.2 Organização dos dados

No padrão H.264/MPEG-AVC o vídeo é organizado hierarquicamente em objetos encapsulados, como mostra a Figura F.1.

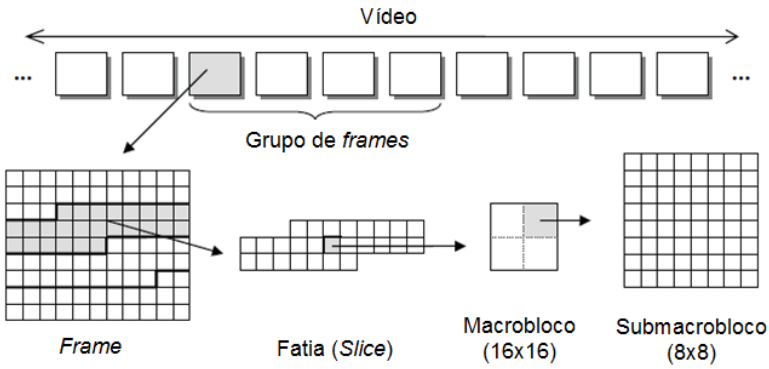


Figura F.1 Organização dos dados de um vídeo no padrão H.264/MPEG-AVC.

Tal como outros padrões de compressão de vídeo, o padrão H.264/MPEG-AVC utiliza o conceito de macroblocos para a organização dos dados. Um macrobloco apresenta a mesma estrutura que usada por outros padrões, sendo composto por 4 blocos 8x8 de componentes luminância e 2 blocos 8x8 de componentes crominância [25]. Contudo, ao contrário de outros padrões, o H.264 permite ainda dividir um macrobloco em partições macrobloco, ou seja, em blocos de 16x16, 16x8, 8x16 e 8x8 amostras luma (e correspondentes amostras croma) [2]. No caso da partição 8x8, cada sub-macrobloco 8x8 pode ainda ser dividido em partições sub-macrobloco de tamanho 8x8, 8x4, 4x8 e 4x4 amostras luma (e correspondentes amostra croma) [2] , tal como mostra a Figura F.2. Quanto às componentes croma de cada macrobloco, têm metade da resolução horizontal e vertical da componente luma. Note-se no entanto que estas partições do macrobloco apenas podem ser utilizadas para fins de predição ou transformação, pois a unidade com que é realizada a codificação é sempre a de macrobloco [15].

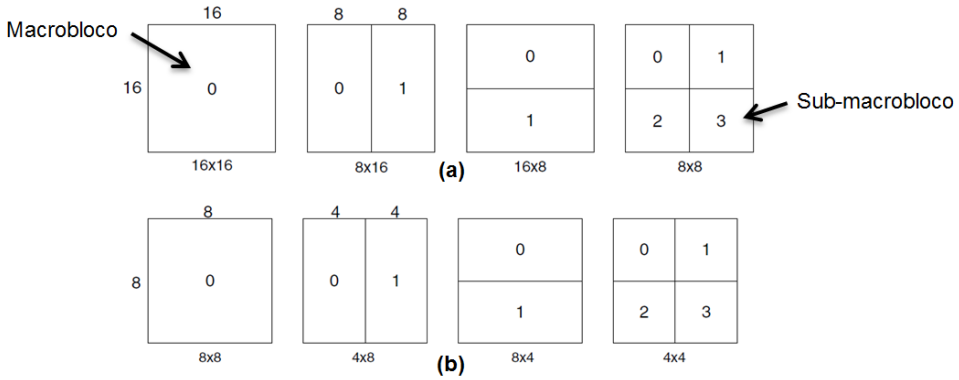


Figura F.2 Partições de macrobloco (a) e partições de sub-macrobloco (b).

Existem 3 tipos diferentes de macroblocos: *intra* (I), *preditive* (P) e *bi-preditive* (B):

- Tipo I: resultam de predição *intra* a partir de amostras descodificadas na mesma *frame*, de modo que a primeira *frame* de qualquer sequência de vídeo é codificada apenas com estes macroblocos [24]. Uma predição pode ser formada tanto para um macrobloco completo, ou para cada bloco 4x4 de amostras luma (e correspondentes amostras croma) de um macrobloco.
- Tipo P: são preditos usando predição *inter*, com cada partição macrobloco (e correspondentes partições sub-macrobloco) sendo predita a partir de uma *frame* de referência [2];
- Tipo B: são preditos usando predição *inter*, com cada partição macrobloco (e correspondentes partições sub-macrobloco) sendo predita a partir de uma ou duas *frames* de referência [2];

As *frames* também podem ser classificadas como sendo do tipo I, P ou B (ver Figura F.3) conforme os tipos de macroblocos que as constituam:

- *Frames* I: contêm apenas macroblocos do tipo I;
- *Frames* P: contêm macroblocos do tipo P, e também podem incluir o tipo I;
- *Frames* B*:* Contêm macroblocos do tipo B, e também podem incluir os tipos I e P.

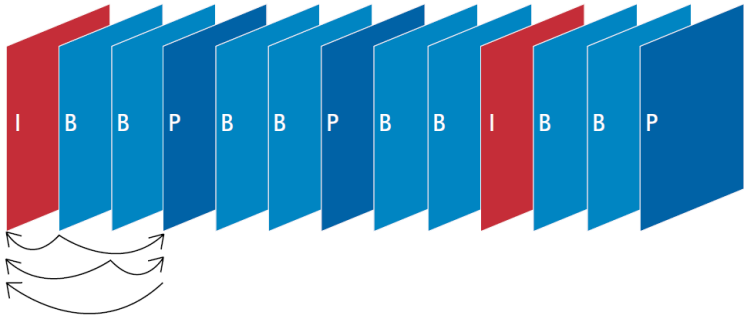


Figura F.3 Uma sequência de vídeo típica, com *frames* dos tipos I, P e B [24].

Este padrão permite ainda que cada *frame* de uma sequência de vídeo seja codificada como uma ou mais fatias (ou *slices*), cada uma contendo um número inteiro de macroblocos sequenciais desde 1 (1 macrobloco por fatia) até o número total de macroblocos da imagem (1 fatia por imagem) [2]. O número de macroblocos por fatia não precisa de ser constante dentro de uma imagem. Existem 5 tipos de fatias, apresentadas na Tabela F.1, podendo uma imagem ser codificada com mais do que 1 deles. Também é possível enviar e descodificar as fatias de uma *frame* segundo uma ordem arbitrária, independentemente da ordem sequencial das mesmas [2]. Esta caraterística é conhecida por ASO (*Arbitrary Slice Order*).

Tabela F.1 Tipos de fatias definidas no H.264.

Tipo de fatia	Descrição	Perfis de utilização
I	Contém apenas macroblocos do tipo I.	Todos
P	Contém macroblocos do tipo P e também pode conter do tipo I.	Todos
B	Contém macroblocos do tipo B e também pode conter do tipo I.	*Extended* e *Main*
SP	Facilita a mudança entre fluxos codificados, e contém macroblocos do tipo P e/ou tipo I.	*Extended*
SI	Facilita a mudança entre fluxos codificados, e contém macroblocos do tipo SI (um tipo especial de macroblocos intra-codificados).	*Extended*

Uma *frame* pode ainda ser dividida em grupos de fatias, sendo que cada grupo de fatias pode ser composto por uma ou mais fatias [2]. Este particionamento

pode ser realizado de diversas formas, existindo já algumas predefinidas. A utilização de grupos de fatias proporciona maior robustez à perda de dados, bem como uma forma de codificação de regiões de interesse com parâmetros distintos [15].

F.3 Processos de codificação e descodificação

Tal como outros padrões de codificação recentes, o H.264/MPEG-AVC não define explicitamente o codificador. Ao invés disso, define a sintaxe de um fluxo de *bits* de vídeo codificado, bom como o método de descodificação desse fluxo de *bits* [2]. Na prática, um codificador e descodificador compatíveis devem incluir os elementos funcionais apresentados na Figura F.4 e na Figura F.5, respetivamente. Estes diagramas são semelhantes aos utilizados por outros padrões de compressão de vídeo, com a exceção de que nestes é incluído um filtro para reduzir o efeito de blocagem. O fluxo seguido pelos dados nos processos de codificação e descodificação é descrito de seguida. Para efeitos de simplicidade de compreensão, por "bloco" entenda-se uma partição macrobloco, ou uma partição sub-macrobloco.

No codificador, cada *frame* F_n é processada em unidades de macrobloco. Cada macrobloco é codificado no modo intra ou inter e, para cada bloco num macrobloco, é formada uma predição *P*. No modo Intra, *P* é formada a partir de amostras da fatia atual, uF'_n, que já tenham sido codificadas, descodificadas e reconstruidas (sem filtragem), enquanto no modo Inter, *P* é formada através da predição pela estimação de movimento (EM) e compensação de movimento (CM) a partir de uma ou duas *frames* de referência, F'_{n-1}. A predição *P* é subtraída ao bloco atual para produzir um bloco D_n residual (de diferença), o qual é transformado (usando um transformada de blocos) em *T* e quantizado em *Q,* resultando em *X*, um conjunto de coeficientes da transformada quantizados. Estes coeficientes são posteriormente reordenados conforme um esquema *zigzag* e a sua entropia é codificada. Os dados na saída do codificador de entropia, juntamente com informação necessária à descodificação de cada bloco (modos de predição, parâmetro de quantização, informação do vetor de movimento, etc.) representam dados da camada de codificação de vídeo (*Video Coding Layer,* VCL), e são mapeados em unidades NAL (*Network Abstraction Layer)* antes da transmissão ou armazenamento.

O processo de reconstrução de uma *frame* para referência no codificador começa pelo quantização inversa (Q^{-1}) dos coeficientes *X*, seguida pela aplicação da transformada inversa (T^{-1}) de modo a produzir um bloco de diferença, D'_n. Este é de seguida adicionado a *P,* resultando numa versão descodificada do bloco atual, uF'_n,

que por sua vez é submetido a um filtro que reduz o efeito de blocagem. A *frame* reconstruida é criada, portanto, a partir de uma série de blocos F'_n.

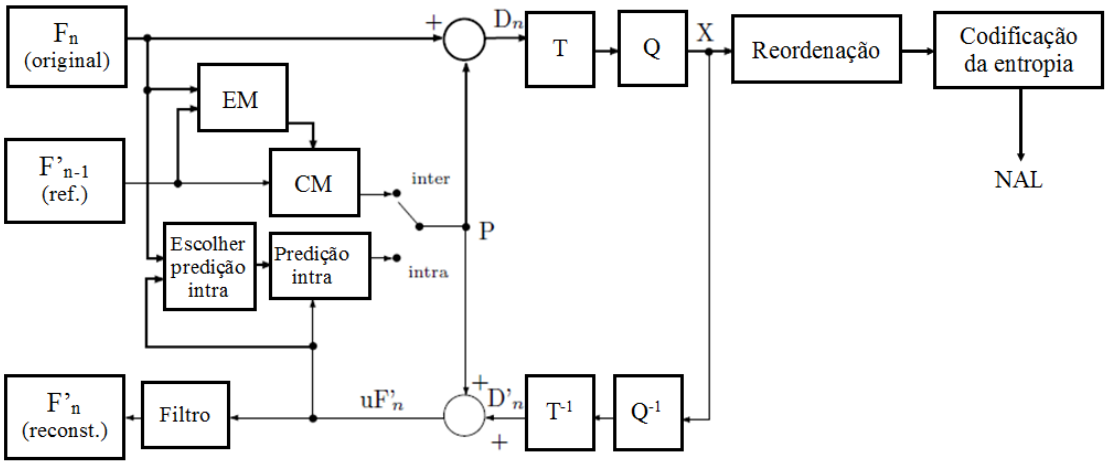


Figura F.4 Diagrama de blocos do codificador H.264/MPEG-AVC.

No descodificador, o fluxo de *bits* comprimidos são recebidos através das unidades NAL, e o conjunto de coeficientes quantizados *X* são obtidos após a descodificação de entropia e da reordenação das unidades NAL. Após a quantização inversa e transformada inversa, é obtido conjunto D'_n (idêntico ao D'_n do codificador). Usando a informação de cabeçalho descodificada do fluxo de *bits*, o descodificador cria um bloco de predição *P*, também da mesma forma que ocorre no codificador, o qual é adicionado a D'_n para produzir uF'_n, que por sua vez é filtrado para criar cada bloco descodificado F'_n.

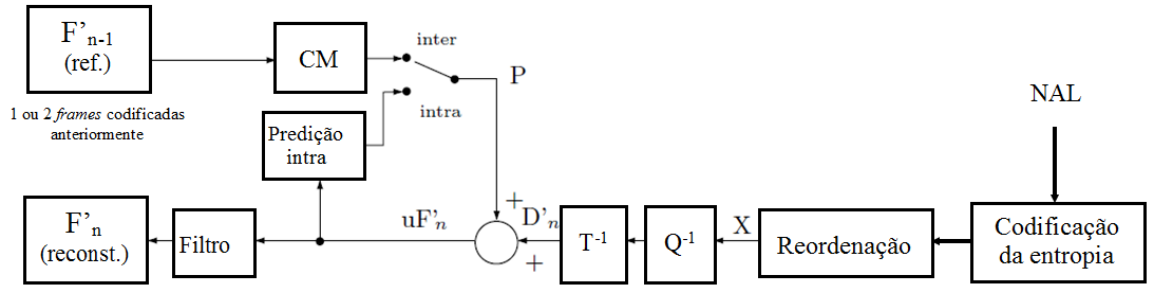


Figura F.5 Diagrama de blocos do descodificador H.264/MPEG-AVC.

Um outro aspeto que é novo no padrão H.264/MPEG-AVC é o procedimento seguido para a reconstrução de *frames* no codificador. Este desempenha um papel importante, pois garante que tanto o codificador como o descodificador usam *frames* de referência idênticas para criar a predição *P.* É importante que as predições *P* no codificador e descodificador sejam idênticas uma vez que quaisquer diferenças entre elas tendem a se acumular e a levar a um erro crescente entre o codificador e descodificador [33].

F.4 Predição *intra*

O padrão H.264 contém diversos de modos de predição espacial. Para blocos 4x4 existem 9 modos de predição, estando 8 deles sumarizados na Figura F.6. Os 16 pixéis de *a* a *p* no bloco são preditos usando os 13 píxeis no contorno [25]. As setas correspondentes aos números modo mostram a direção da predição. Na Tabela F.2 mostra alguns exemplos de quais os píxeis que são preditos a partir de um outro, em função do modo usado.

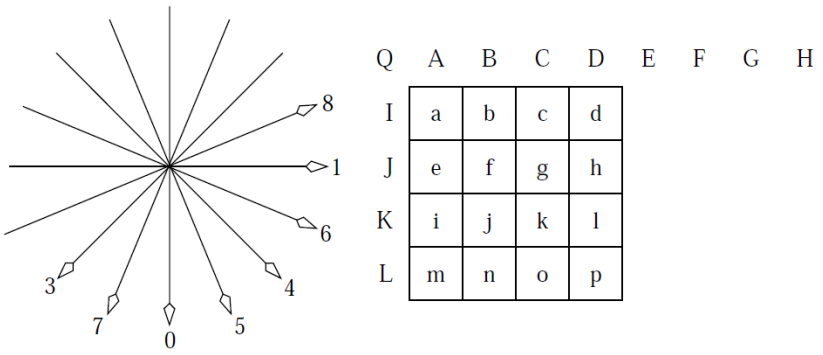


Figura F.6 Modos de predição usados na codificação intra de blocos 4x4 [25].

Tabela F.2 Píxeis do bloco 4x4 da Figura F.6 *intra preditos* nos modos 0 e 3.

Píxel de referência	A	B	C	D	B	C	D	E	F	G	H
Modo	0				3						
Píxeis preditos	a, e, i, m	b, f, j, n	c, g, k, o	d, h, l, p	a	b,e	c, f, i	d, g, j, m	h, k, n	l, o	p

Note-se que nenhuma direção está disponível para o modo 2. Este é chamado modo DC, no qual a média dos píxeis do contorno esquerdo e em cima é usada como predição para todos 16 píxeis num bloco 4x4 [25] [33]. Em muitos casos os modos de predição usados para todos os blocos 4x4 num macrobloco estão altamente correlacionados. O padrão usa esta correlação para codificar eficientemente a informação do modo.

Nas regiões suaves da imagem é mais conveniente usar predição na base do macrobloco. Para este caso, existem 4 modos de predição. Três deles correspondem aos modos 0, 1 e 2 (vertical, horizontal e DC), enquanto o quarto modo de predição é chamado modo de predição planar [25], [33].

F.5 Predição *Inter*

A predição *inter* implementa o modelo temporal, o qual tem como base as *frames* codificadas previamente à corrente. A compensação de movimento é realizada de forma muito precisa e eficiente no H.264. As principais causas que conduzem a tal referem-se à capacidade de blocos de diferentes dimensões – tal como ilustra a Figura F.2 –, e ao uso de vetores de movimento de fina resolução [33].

O algoritmo de compensação de movimento começa por dividir a componente luma de um macrobloco de 16x16 em partições como as da Figura F.2, e calcular um vetor de movimento individual para cada partição. O mesmo acontece com as componentes croma, sendo cada partição compensada de movimento individualmente [33].

Naturalmente que o esquema de partição e os vetores de movimento individuais têm de ser enviados ao descodificador como informação paralela. É importante, portanto, manter um equilíbrio entre a melhor compressão que resulta de pequenas partições, e a maior informação paralela que as pequenas partições requerem. Tal é possível pelo ajuste do tamanho da partição. A regra geral consiste em usar partições maiores para codificar as áreas homogéneas de uma *frame*, e partições mais pequenas para as áreas com mais detalhes [2].

Cada partição ou partição sub-macrobloco num macrobloco *inter*-codificado é predita a partir de uma área com a mesma dimensão na imagem de referência. O vetor de movimento, que reflete o deslocamento entre as duas áreas, tem resolução de um quarto de amostra para a componente luma e resolução de uma oitava de amostra para as componentes croma [2]. A Figura F.7 ilustra dois exemplos de predição de um bloco de 4x4 amostras, a partir de uma região da imagem de referência, na vizinhança da posição do bloco corrente. Na Figura F.7 (b), o vetor de movimento associado ao movimento entre a *frame* corrente, na Figura F.7 (a), e a *frame* de predição, possui componentes verticais e horizontais inteiras, de modo que as amostras relevantes no bloco de referência realmente existem (pontos cinzentos). Já na Figura F.7 (c), ambas as componentes vertical e horizontal são valores fracionários, e as amostras relevantes no bloco de referência (pontos cinzentos) não existem, sendo, portanto, geradas pela interpolação entre amostras adjacentes na *frame* de referência [2].

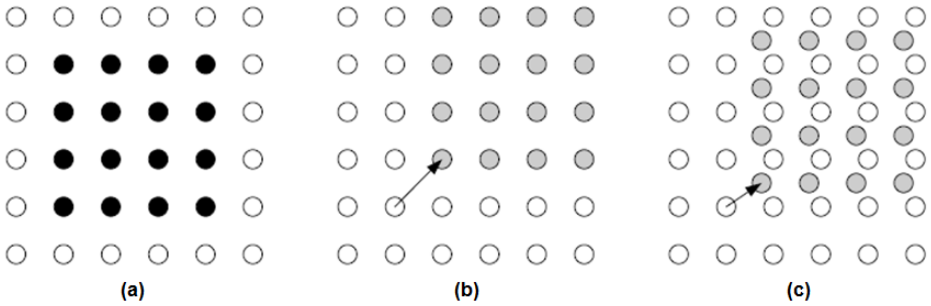


Figura F.7 Predição *inter* de um bloco 4x4 [2].

A codificação de um vetor de movimento para cada partição pode custar um número significativo de *bits*, especialmente se forem utilizadas partições de reduzidas dimensões. Para além disso, os vetores de movimento para partições vizinhas estão normalmente altamente correlacionados, razões pelas quais cada vetor de movimento é predito a partir de vetores de partições próximas codificadas anteriormente. Um vetor predito é formado com base nos vetores de movimento anteriormente calculados e o MVD (*Motion Vector Difference*), a diferença entre o vetor atual e o vetor predito, é codificada e transmitida.

F.6 Filtro de remoção de artefactos – *Deblocking Filter*

O filtro suaviza os limites dos blocos, conduzindo a uma redução do efeito de blocagem. Tal é bem visível nas imagens da Figura F.8.

(a) *Frame* com filtragem (b) *Frame* sem filtragem

Figura F.8 Efeito da aplicação do filtro de remoção de artefatos numa *frame* [76].

A aplicação do filtro permite aumentar o desempenho da compressão, uma vez que a imagem filtrada é, normalmente, uma reprodução mais fiel da *frame* original, do que uma imagem não filtrada (com blocagem) [2]. A filtragem é aplicada às arestas verticais e horizontais de blocos de 4x4 (com exceção das arestas nos

limites dos *slices*), nomeadamente aquelas a tracejado na Figura F.9, pela seguinte ordem:

1. Filtragem dos 4 limites verticais da componente luma (pela ordem a, b, c e d, na Figura F.9);
2. Filtragem dos 4 limites horizontais da componente luma (pela ordem e, f, g e h, na Figura F.9);
3. Filtragem dos 2 limites verticais da componente croma (i, j);
4. Filtragem dos 2 limites horizontais da componente croma (k, l);

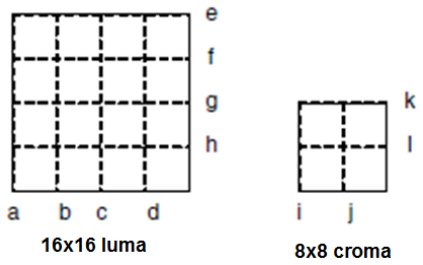


Figura F.9 Ordem com que são filtradas as arestas num macrobloco [2].

Cada operação de filtragem afeta até três amostras de cada lado do limite que separa dois blocos adjacentes, p e q, da forma apresentada na Figura F.10.

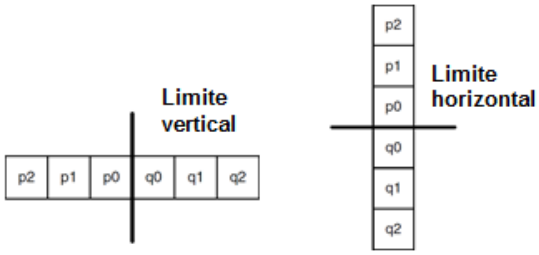


Figura F.10 Amostras em torno dos limites vertical e horizontal de dois blocos adjacentes.

O parâmetro *bS* (*boundary strength*) determina a intensidade da filtragem, e depende do parâmetro de quantização (PQ) associado aos blocos, dos modos de codificação dos blocos vizinhos e do gradiente (mudança) das amostras da imagem em torno do limite [2], nomeadamente segundo as seguintes regras:

- p e/ou q é intra codificado, e o limite entre eles é um limite de macrobloco: *bS*=4 (filtragem mais forte);
- p e q são intra codificados, e limite entre eles não é um limite de macrobloco: *bS*=3;
- p e q não são intra codificados, mas contêm coeficientes codificados: *bS*=2;
- p e q não são intra codificados, nem contêm coeficientes codificados; p e q usam diferentes imagens de referência, ou um número diferente de imagens referência, ou os seus vetores de movimento têm valores que diferem por uma amostra luma ou mais: *bS*=1.

Se nenhuma destas situações ocorrer, então *bS*=0 e não é realizada qualquer filtragem. A aplicação destas regras resulta numa filtragem mais intensa nas zonas onde a blocagem é mais significativa, como nos limites de um macrobloco codificado ou num limite entre blocos que contêm coeficientes codificados [2]. Para além de *bS* ter de ser diferente de 0, a filtragem apenas ocorre se

- $|p0 - q0| < \alpha(PQ)$,
- $|p1 - p0| < \beta(PQ)$,
- $|q1 - q0| < \beta(PQ)$,

onde α e β são *thresholds* definidos no padrão, cujos valores aumentam com o valor médio do PQ dos dois blocos p e q. Quando o PQ é pequeno, um gradiente significativo no limite entre p e q, normalmente deve-se às caraterísticas da imagem (e não ao efeito de blocagem), de modo que tal deve ser preservado, e portanto, α e β são baixos e não ocorre filtragem. Já quando o PQ é grande, é mais provável que a imagem contenha efeitos de blocagem, de modo que α e β são maiores para que os limites das amostras sejam filtrados [2].

O fluxograma na Figura F.11 descreve o modo como este filtro é implementado [2].

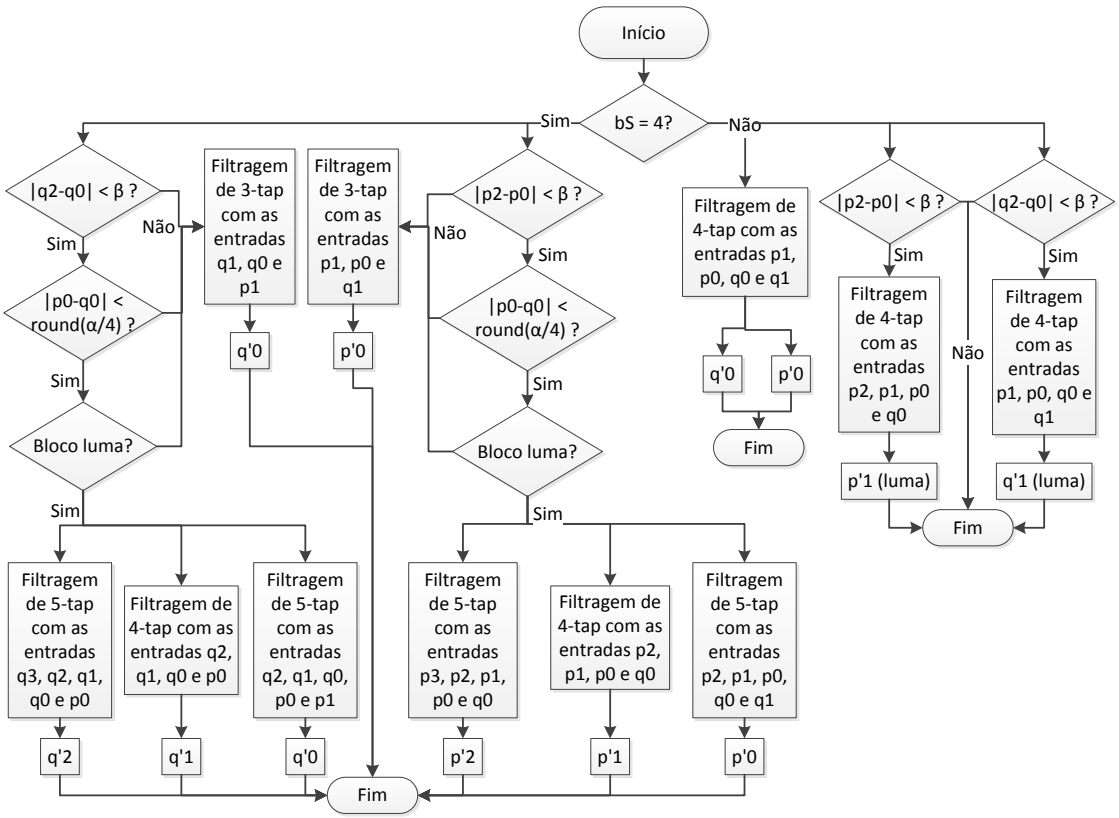


Figura F.11 Implementação do filtro de remoção de artefatos.

F.7 Transformada

O padrão H.264 usa três transformadas na codificação de diferentes tipos de dados residuais: os coeficientes luma DC provenientes de macroblocos 16x16 *intra* preditos são codificados em blocos de 4x4 através de uma transformada *Hadamard*; os coeficientes croma DC também são codificados com uma transformada *Hadamard*, em blocos de 2x2; já os restantes coeficientes são codificados em blocos de 4x4 através de uma transformada baseada na DCT [25].

A Figura F.12 mostra a ordem com que são transmitidos os dados dentro de um macrobloco. Se este for codificado no modo intra 16x16, então o bloco numerado com -1, que contém o coeficiente DC de cada bloco luma 4x4 transformado, é transmitido primeiro. De seguida são transmitidos os blocos luma residuais numerados de 0 a 15 (sem respetivos coeficientes DC) pela ordem apresentada, seguidos pelos blocos 16 e 17, cada um de 2x2 amostras relativas aos coeficientes DC das componentes croma Cb e Cr, respetivamente. Finalmente, são enviados os blocos residuais de componentes croma numerados de 18 a 25 (sem os respetivos coeficientes DC).

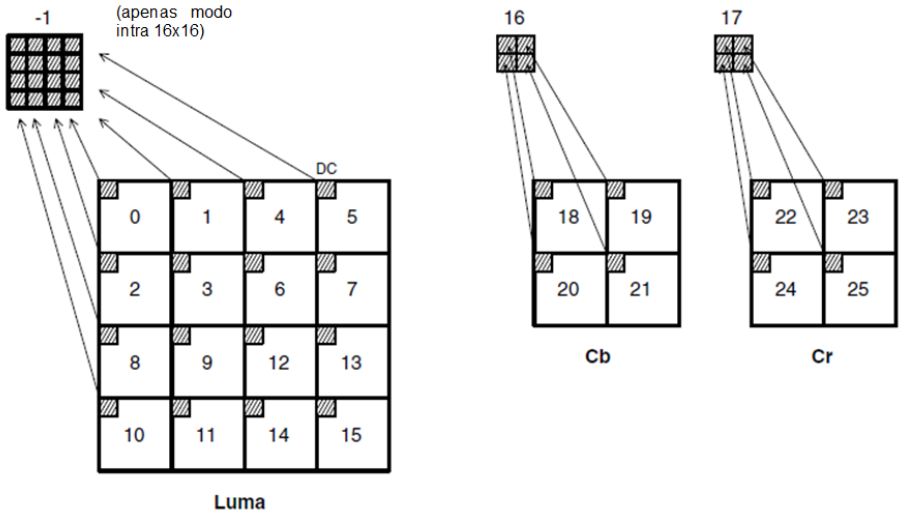


Figura F.12 Ordem de varredura dos blocos residuais dentro de um macrobloco [2].

Uma breve descrição das referidas transformadas é apresentada de seguida.

F.7.1 Transformada de blocos 4x4 baseada na DCT e quantização

Esta é a transformada utilizada na codificação dos blocos 4x4 de dados residuais numerados de 0 a 15 e de 18 a 25 da Figura F.12. Como já se referiu, é parecida à DCT, mas faz uso de uma matriz inteira 4x4 [2], o que proporciona algumas vantagens. A sua natureza inteira permite que todas as operações possam ser executadas usando aritmética inteira, sem perda de precisão, e evitando a

acumulação de erros no processo de transformação. Esta caraterística também torna a implementação simples, através do uso de adições e deslocamentos, e garante total compatibilidade entre as transformadas inversas no codificador e descodificador. Para além disso, uma multiplicação escalar pertencente à transformada é integrada no quantizador, reduzindo o número total de multiplicações [2].

A transformada DCT 4x4 é dada por

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}, \tag{F.1}$$

onde

$$a = \frac{1}{2}, \;\; b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \;\; e \;\; c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right). \tag{F.2}$$

A fatorização desta multiplicação de matrizes resulta na forma equivalente

$$Y = (CXC^T) \otimes E$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}, \tag{F.3}$$

em que CXC^T é o "núcleo" de uma transformada 2D, E é uma matriz de fatores escalares, e o símbolo ⊗ indica que cada elemento de CXC^T é multiplicado pelo fator escalar na mesma posição na matriz E (multiplicação escalar ao invés de multiplicação de matrizes). A constante d é dada por c/b (aproximadamente 0,414).

No sentido de simplificar a transformada, d é aproximado para 0,5. Assim sendo, de modo a assegurar que a transformada mantém-se ortogonal, b precisa de ser modificado para $\sqrt{2/5}$. A transformada resultante após estas modificações é a seguinte [2]:

$$Y = C_f X C_f^T \otimes E_f \tag{F.4}$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}.$$

A quantização destes coeficientes é realizada de forma escalar, e ambas as operações direta e inversa são dificultadas pelos requisitos de (a) evitar divisões e/ou aritmética de ponto flutuante, e (b) incorporar as matrizes escalares E e E_f [2]. A quantização direta é realizada através de

$$Z_{ij} = round\left(\frac{Y_{ij}}{Qstep}\right), \tag{F.5}$$

em que Y_{ij} é um coeficiente da transformada, $Qstep$ é um tamanho do passo de quantização e Z_{ij} um coeficiente quantizado. O padrão suporta um total de 52 valores para $Qstep$ indexados por um parâmetro de quantização (PQ), proporcionando flexibilidade ao codificador para controlar o equilíbrio entre taxa de *bits* e a qualidade. Os valores de PQ podem ser diferentes para as componentes luma e croma, contudo ambos variam na gama de 0 a 51. A equação (F.5) também pode ser representada na forma

$$Z_{ij} = round\left(W_{ij}.\frac{PF}{Qstep}\right), \tag{F.6}$$

se se considerar $W_{ij} = CXC^T$, e PF o fator da matriz E na posição correspondente a (i, j). É possível ainda simplificar a aritmética se o quociente $\frac{PF}{Qstep}$ for implementado como uma multiplicação por um fator de multiplicação MF (*Multiplication factor*), dependente da posição (i, j) como indicado na Tabela F.3, e um deslocamento à direita, evitando quaisquer operações de divisão, ou seja,

$$Z_{ij} = round\left(W_{ij}.\frac{MF}{2^{qbits}}\right), \tag{F.7}$$

onde

$$\frac{MF}{2^{qbits}} = \frac{PF}{Qstep} \tag{F.8}$$

e

$$qbits = 15 + floor\left(\frac{PQ}{6}\right). \tag{F.9}$$

Tabela F.3 Fator de multiplicação MF [2].

PQ	Posições (0,0), (2,0), (2,2), (0,2)	Posições (1,1), (1,3), (3,1), (3,3)	Outras posições
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

A equação para realizar a quantização pode ser reduzida para a forma

$$\left|Z_{ij}\right| = \left(\left|W_{ij}\right|.MF + f\right) \gg qbits, \tag{F.10}$$

$$sign\left(Z_{ij}\right) = sign\left(W_{ij}\right).$$

Quanto à transformada inversa, é dada por

$$Y = C_i^T (Y \otimes E_i) C_i$$

$$\leftrightarrow Y = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} X \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}. \tag{F.11}$$

Note-se que as transformadas direta e inversa são ortogonais, isto é, $T^{-1}\left(T(X)\right) = X$. Quanto à operação de quantização inversa básica, é dada por

$$Y'_{ij} = Z_{ij} Qstep. \tag{F.12}$$

De forma análoga ao processo de quantização direta, esta equação básica pode ser representada na forma

$$W'_{ij} = Z_{ij} V_{ij} 2^{floor(\frac{PQ}{6})}, \tag{F.13}$$

em que W'_{ij}é um coeficiente resultante da transformada inversa, $C_i^T W C_i$, e

$$V = Qstep.PF.64 \tag{F.14}$$

é um parâmetro definido para $0 \leq PQ \leq 5$ e para cada coeficiente na posição (i,j), cujos valores apresentam-se na Tabela F.4.

Tabela F.4 Fator de escala V [2].

PQ	Posições (0,0), (2,0), (2,2), (0,2)	Posições (1,1), (1,3), (3,1), (3,3)	Outras posições
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

A multiplicação por 64 representa uma operação escalar por um valor constante com o propósito de evitar erros de arredondamento [2].

F.7.2 Transformada de blocos 4x4 de coeficientes luma DC

Se o macrobloco for codificado no modo de predição 16x16 (ou seja, se toda a componente luma 16x16 for predita a partir de amostras vizinhas), cada bloco residual 4x4 começa por ser transformado usando o "núcleo" de transformada descrita acima $(C_f X C_f^T)$ [2]. No caso do coeficiente DC de cada bloco 4x4, é ainda transformado usando a transformada *Hadamard* 4x4

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} W_D \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) /2, \tag{F.15}$$

em que W_Dé o bloco de 4x4 coeficientes DC, e Y_Dé o bloco após a transformação. A quantização é então realizada através de

$$\begin{gathered} |Z_{D(i,j)}| = \left(|Y_{D(i,j)}| MF_{(0,0)} + 2f\right) \gg (qbits + 1), \\ sign\left(Z_{D(i,j)}\right) = sign\left(Y_{D(i,j)}\right). \end{gathered} \tag{F.16}$$

A ordem com que estas operações são implementadas mantém-se aquando da descodificação, ou seja, a transformada *Hadamard* inversa na equação (F.17),

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} Z_D \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right), \tag{F.17}$$

tem de ser aplicada antes da quantização inversa, na equação (F.18),

$$W'_{D(i,j)} = \begin{cases} W_{QD(i,j)} V_{(0,0)} 2^{floor\left(\frac{PQ}{6}\right) - 2} & (PQ \geq 12), \\ \left[W_{QD(i,j)} V_{(0,0)} + 2^{1 - floor\left(\frac{PQ}{6}\right)} \right] >> (2 - floor\left(\frac{PQ}{6}\right) & (PQ < 12), \end{cases} \tag{F.18}$$

na qual $V_{(0,0)}$ é o fator de escala V para a posição (0,0) na Tabela F.4. Os coeficientes DC W'_D obtidos após estas operações são então inseridos nos seus respetivos blocos 4x4, e cada bloco 4x4 é transformado inversamente através do "núcleo" da transformada inversa baseada na DCT ($C_i^T W' C_i$) [2].

F.7.3 Transformada e quantização dos coeficientes DC croma 2x2

A transformada aplicada aos blocos 2x2 de coeficientes DC das componentes croma, W_D, é

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} W_D \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{F.19}$$

Seguidamente, os coeficientes resultantes são quantizados através de

$$\begin{aligned} |Z_{D(i,j)}| &= \left(|Y_{D(i,j)}| MF_{(0,0)} + 2f\right) \gg (qbits + 1), \\ sign(Z_{D(i,j)}) &= sign(Y_{D(i,j)}). \end{aligned} \tag{F.20}$$

Quanto à descodificação, a transformada inversa aplicada é

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} Z_D \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{F.21}$$

enquanto a quantização inversa é dada por

$$W'_{D(i,j)} = \begin{cases} W_{QD(i,j)} V_{(0,0)} 2^{floor}\left(\frac{PQ}{6}\right) - 1 & (PQ \geq 6), \\ \left[W_{QD(i,j)} V_{(0,0)}\right] >> 1 & (PQ < 6). \end{cases} \tag{F.22}$$

Os coeficientes resultantes são substituídos nos seus respetivos blocos 4x4 de coeficientes croma que são então transformados como acima ($C_i^T W' C_i$) [2].

F.8 Codificação da entropia

O H.264/AVC especifica dois métodos alternativos para a codificação da entropia. Num deles, o usado no modo *baseline,* são utilizados códigos CAVLC (*Context Adaptive Variable length Coding*) para codificar os dados da transformada quantizados, e códigos *Exp-Golomb* para codificar os restantes elementos da sintaxe [62]. No outro, com maiores requisitos computacionais, os dados são transformados em *strings* binárias e codificados com um código CABAC (*Context Adaptive Binary Arithmetic Coding*) [25].

Tipicamente, após a reordenação, cada bloco contém apenas alguns coeficientes significativos, ou seja, diferentes de zero, onde se incluem sequências de coeficientes de magnitude 1, chamadas de *trailing 1's* (T1), nomeadamente no final da varredura. Quando é o usado o método com códigos CAVLC, o número de

coeficientes diferentes de zero e o número de T1's são os primeiros a ser transmitidos usando uma palavra de código combinada onde uma de entre quatro tabelas VLC é usada com base no número de coeficientes significativos dos blocos vizinhos [62]. De seguida, o valor e o sinal de cada coeficiente significativo são codificados segundo uma ordem inversa, ou seja, os coeficientes de maiores frequências são codificadas primeiro, seguindo-se a codificação do número total de 1's desde o início da varredura até o último coeficiente diferente de zero [25]. Por fim, os coeficientes quantizados para 0 são codificados pela transmissão do número total de 0's antes do último coeficiente significativo, e adicionalmente, para cada coeficiente significativo, é indicado o número consecutivo de 0's que o precedem [62]. A escolha de um VLC adequado para a codificação do valor de cada sequência de 0's é realizada em função da monitorização do número máximo de 0's possível em cada passo da codificação. Este método é capaz de reduzir a taxa de *bits* em cerca de 2% a 7% em relação à obtida com um esquema *run-length* tradicional baseado num único código *Exp-Golomb* [62].

Quanto ao outro método, com recurso a códigos CABAC, apesar de ser computacionalmente mais complexo, apresenta maior eficiência de compressão ao proporcionar reduções na taxa de *bits* de 5% a 15% comparativamente com o CAVLC [2], [25], [62]. Este método é baseado em três elementos chave: binarização, modelação do contexto, e codificação aritmética binaria. A binarização faz o mapeamento dos elementos da sintaxe não-binários para uma sequência de *bits*, chamada *bin string,* o que permite codificação aritmética binaria eficiente. Cada elemento desta *bin string* pode ser processado no modo de codificação regular ou no modo *bypass*. O último é utilizado para *bin strings* selecionadas, tais como informação do sinal ou *bin strings* menos significativas, de modo a acelerar todo o processo de codificação (e descodificação) através de um mecanismo de codificação simplificado *bypass*. Já no modo de codificação regular uma *bin string* pode ser modelada ao contexto e subsequentemente codificada aritmeticamente. Em geral, apenas a *bin string* mais provável de um elemento da sintaxe é fornecido com um modelo de contexto usando bin *strings* previamente codificadas [62]. Para além disso, todas as *bin strings* codificadas regularmente são adaptados pela estimação da sua distribuição de probabilidade real. A estimação da probabilidade e a codificação aritmética binaria real é conduzida usando um método livre de multiplicações, que permite implementações eficientes em *hardware* e *software* [62].

F.9 Controlo da taxa de *bits*

Quando se pretende manter uma qualidade visual consistente entre *frames* numa sequência de vídeo, o número de *bits* requeridos para codificar cada *frame* pode ser muito variável, uma vez que depende da complexidade de cada *frame*. Embora tal não represente qualquer problema para algumas aplicações, para outras, especialmente para comunicações em tempo-real como *streaming,* videoconferência e videovigilância, é fundamental que a taxa de *bits* se mantenha constante. Para este tipo de aplicações, é necessário um esquema de controlo de taxa que vá de encontro a uma taxa de canal restrita através do controlo do número de *bits* gerados. Contudo é muito difícil ajustar os parâmetros de codificação diretamente para obter um número *bits* fixo para cada *frame* codificada num canal de taxa de *bits* constante. É necessário, portanto, a utilização de um *buffer* VBV (*Video Buffering Verifier*) [77] que regule o fluxo de *bits* antes da transmissão, e uma técnica de controlo de taxa que impeça que a taxa de saída cause *overflow* ou *underflow* do *buffer*. Estas técnicas recorrem essencialmente à alocação de *bits* a cada *frame* de acordo com a sua complexidade, e ao parâmetro de quantização para codificar a mesma de modo a alcançar o número de *bits* pretendido [78].

Duas técnicas de controlo de *bits* comumente utilizadas são o controlo de taxa de *bits* constante (*Constant Bit Rate* – CBR) e o controlo de taxa de *bits* variável (*Variable Bit Rate* – VBR). O controlo de acordo com um algoritmo CBR é mais rigoroso que aquele realizado com um algoritmo VBR, permitindo manter atrasos *end-to-end* dentro de um limite especificado pelo codificador [61]. Consequentemente, pode dar origem a vídeos com qualidade visual inconsistente, uma vez que a taxa de *bits* é aproximadamente constante entre as diversas *frames*, independentemente da complexidade das mesmas [63]. O algoritmo CBR implementado pelo codificador H.264 da TI nos controladores DM365 e DM368 [57] baseia-se no seguinte:

1- Os *bits* são alocados às *frames* para assegurar que a taxa de *bits* máxima pretendida não é excedida;

2- No início de uma *frame*, o parâmetro de quantização é ajustado conforme o número de *bits* da *frame* anterior: se esse valor estiver acima do valor pretendido, o parâmetro de quantização é aumentado; já se estiver abaixo, o parâmetro de quantização é reduzido;

3- Se o número de *bits* consumidos por uma dada *frame* for maior que o valor pretendido, já tendo em conta o atraso permitido, essa *frame* é ignorada, isto é, não é codificada;

4- O controlo também é feito ao nível da linha, ou seja, para cada linha de macroblocos, um ajuste adicional é feito com base se a *frame* atual está sendo codificada a uma taxa acima ou abaixo do valor pretendido. No primeiro caso o parâmetro de quantização é aumentado, enquanto no segundo caso o parâmetro de quantização é reduzido;

5- No final da *frame*, o parâmetro de quantização médio da *frame* é calculado para usar como base para a próxima *frame*.

Já um algoritmo VBR implementa um controlo que tem como foco manter a qualidade visual da sequência, ao invés de procurar manter a taxa de *bits* constante a curto prazo, como é o caso do CBR [63]. O VBR tem flexibilidade para alcançar a média da taxa de *bits* após um intervalo de tempo maior quando comparado com o CBR, tolerando, portanto, maiores atrasos *end-to-end* e maiores taxas de *bits* instantâneas se necessário. Enquanto com o CBR é possível que algumas *frames* não sejam codificadas para que a taxa de *bits* pretendida não seja ultrapassada, no VBR tal não acontece, de modo que todas as *frames* são codificadas [61]. A título de exemplo apresenta-se de seguida a descrição do algoritmo VBR implementado no codificador H.264 da TI nos controladores DM365 e DM368 [57].

1- Os *bits* são alocados às *frames* para assegurar que a taxa de *bits* máxima pretendida não é excedida;

2- No início de uma *frame*, o parâmetro de quantização é ajustado conforme o número de *bits* da *frame* anterior: se esse valor estiver acima do valor pretendido, o parâmetro de quantização é aumentado; já se estiver abaixo, o parâmetro de quantização é reduzido;

3- O controlo também é feito ao nível da linha, ou seja, para cada linha de macroblocos, um ajuste adicional é feito com base se a *frame* atual está sendo codificada a uma taxa acima ou abaixo do valor pretendido. No primeiro caso o parâmetro de quantização é aumentado, enquanto no segundo caso o parâmetro de quantização é reduzido;

4- No final da *frame*, o parâmetro de quantização médio da *frame* é calculado para usar como base para a próxima *frame*. O parâmetro de quantização base é calculado tendo em conta a ocupação do *buffer* virtual e na diferença entre a taxa de *bits* pretendida e a atual.

Anexo G SoC para codificação de vídeo

A Tabela G.1 apresenta as caraterísticas principais de alguns SoC direcionados para a codificação de vídeo disponíveis no mercado.

Tabela G.1 Descrição de alguns SoC para codificação de vídeo disponíveis no mercado.

SoC	Processadores	Entrada de vídeo	Processamento de vídeo	Saída de vídeo	Comunicação	Memória	Placas de avaliação
Texas instruments DM368 [48]	- ARM926EJ-S, até 432 Mhz	- Interface paralela para sensor de 120MHz, 16 *bit*	- Inclui 2 co-processadores de vídeo (HDVICP, MJCP) - Mecanismo de deteção facial - Mecanismo de redimensionamento - módulo para correção de distorção das lentes - Módulo para histograma - IPIPE em *hardware* para processamento de imagem em tempo real - Suporte em *hardware* para codecs H264,MPEG4,MPEG2, MJPEG, JPEG,WMV9/VC1 - Codificação de vídeo H.264 1080p@30fps, ou 720p@60 fps - Codificação de imagem JPEG 720p@60 fps	- 3 DACs para saída de vídeo HD - *Hardware* OSD (On-Screen Display) - 8/16*bit* YCC e até 24*bit* RGB888 - 2 caminhos para saída simultaneos - Controlador LCD - Codificador de vídeo composto NTSC/PAL - Interface digital para padrões BT.601/BT.656 YCbCr 4:2:2	- Ethernet MAC de 10/100 Mb/s -USB 2.0 (Host,device) - 2 UART - 5 SPI	- DDR2 e mDDR - Interface para memória assíncrona de 16/8*bit* - Interface para cartão SD	- Leopard Board DM368 (Mouser electronics: 119,20 €) - Z3-DM368-RPS
Texas instruments DM3730 [79]	- ARM cortex - A8 com NEON, até 1GHz - DSP C64x, até 800 MHz	- Interface paralela para sensor de 148MHz, 12 *bit*	- Anti-aliasing de imagem de alta qualidade programável -Mecanismo de redimensionamento - Inclui acelerador gráfico POWERRVR SGX - Suporte em *hardware* para codecs H264, MPEG2, MPEG4,VC1, JPEG - Codificação de vídeo H.264 BP 720p@30fps	- Controlador LCD - Codificador de vídeo composto NTSC/PAL - Interface digital para padrões BT.601/BT.656 YCbCr 4:2:2	- 4 UART (1 com IrDA/CIR) - 4 SPI - USB 2.0 (host,device) - 3 I2C	- mDDR - 3 interfaces para MMC/SD/SDIO	- BeagleBoard-xM (Digi-key: 120,05 €) - Zoom DM3730 Torpedo Development Kit (Digi-key: 853,02 €) - TMDSEVM3730 (Mouser-electronics: 1233,09€)

SoC	Processadores	Entrada de vídeo	Processamento de vídeo	Saída de vídeo	Comunicação	Memória	Placas de avaliação
Texas instruments DM8168 [80]	- ARM cortex - A8 com NEON, até 1,35GHz - DSP C64x VLIW, até 1,125 GHz	- Dois canais de captura de 165 MHz (um de 16 *bits* ou 24 *bits*, e um de 16 *bits*)	- Inclui até 3 co-processadores de imagem e vídeo HD (HDVICP2) - Mecanismo de estamção de movimento - Inclui acelerador gráfico POWERRVR SGX - Filtragem de ruído - Suporte em *hardware* para codecs H264, MPEG2, MPEG4,MJPEG,JPEG - Desempenho de codificação e descodificação H.264 1080p@60 fps - Suporte em *hardware* para operações de transcodificação - Anti-aliasing de imagem programável	- Dois canais a 165 MHz para display HD - Saída analógica simultanea para SD e HD - HDMI digital 1.3	- USB 2.0 (host,device) - 3 UART - 1 SPI - 2 I2C - 1 SATA 3.0 - 1 PCIe 2.0 - 2 Ethernet MAC de 10/100/1000 Mb/s	- Duas interfaces de 32 *bit* para DDR2 e DDR3 - *Bus* assíncrono de 16*bit*	- Z3-DM8168-RPS - TMDXEVM8168C (Mouser electronics: 1999 €)
Texas Instruments DM6467T [81]	- ARM926EJ-S, de 500 MHz - DSP C64x, de 1GHz	- Dois canais de 8 *bits* SD (BT.656), um canal de 16 *bits* HD (BT.1120) ou um canal RAW de 8,10 ou 12 *bits*, a 150 MHz	- 2 co-processadores HDVICP - Suporte em *hardware* para codecs H264, MPEG2, MPEG4,VC1,JPEG - Desempenho de codificação e descodificação H.264 1080p@30 fps - Mecanismo para conversão de dados de vídeo (redução vertical e horizontal; conversão entre 4:2:2 e 4:2:0)	- Dois canais de 8 *bits* SD (BT.656) ou um canal de 16 *bits* HD (BT.1120), a 150 MHz	- Ethernet MAC de 10/100/1000 Mb/s - USB 2.0 - PCI - SPI - 3 UART/IrDA/CIR	- Memória assincrona de 16 *bits* (NAND/NOR) - DDR2 de 32 *bits*, 400 MHz	- TMDXEVM6467T (farnell: 1556,10€)
Texas Instruments DM385 [82]	- ARM Cortex-A8, 1 GHz	- Ligação paralela para RAW (até 16 *bits*) e para a norma BT.656 /BT.1120 (8 ou 16 *bits*) - Ligação série CSI2	- 1 co-processador HDVICP2 - Suporte em *hardware* para codecs H264, MPEG2, MPEG4,VC1,JPEG - Desempenho de codificação e descodificação H.264 BP/MP/HP 1080p@60 fps - Mecanismo de redimensionamento - Pipe de imagem (IPIPE) para processamento de imagem e video em tempo-real - Mecanismo de detecção facial - *Hardware* 3A (H3A) para geração de estatisticas chave para o controlo 3A (AE, AWB,AF)	- Analógica composta - Analógica componente HD - HDMI 1.3 - 1 DAC SD - 3 DACs HD	- Ethernet MAC de 10/100/1000 Mb/s - 2 USB 2.0 - PCIe - 3 UART/IrDA/CIR	- 3 interfaces MMC/SD/SDIO - Interface de 32 *bits* para DDR2, DDR3 e DDR3L	- FS-DM385-SOM ($US99-179, alibaba.com)
Intel Atom E640 [83]	- Intel E640, até 1 GHz	- Entrada de vídeo através de USB	- Inclui um acelerador gráfico em *hardware,* permite desempenhar *pixel shading* e *vertex shading* - Posui suporte em *hardware* para codificação MPEG4 e H.264 BP - Desempenho de codificação H.264 BP 720p@30 fps - Posui suporte em *hardware* para descodificação MPEG2, VC1, WMV9, MPEG4 e H.264	- LVDS - *Serial DVO*	- 4 PCIe 1.0a - SPI - Ethernet MAC de 10/100/1000 Mb/s - UART - USB - Interface SATA II de 3Gbps - I2C	- Interface para DDR2-800 - SDIO	- Minnowboard (Mouser electronics: 151,20 €)

SoC	Processadores	Entrada de vídeo	Processamento de vídeo	Saída de vídeo	Comunicação	Memória	Placas de avaliação
Allwinner A20 [84]	- 2 ARM Cortex-A7, com NEON	-2 Interfaces paralelas para sensor YUV (uma de 8 *bits* e outra de 24 *bits*) - Suporta sensor CMOS 5M - Suporta sensores duplos	- Inclui um GPU Mali400MP2 - Descodificação de vídeo 2160p H.264 HD - Descodificação de vídeo FHD de multiplos formatos, tais como MPEG 1/2/4, H.263, H.264, JPEG/MJPEG - Codificação de vídeo H.264 1080p@30fps, ou 720p@60 fps	- Porta HDMI 1.4 - Interface LCD	- Porta *Keypad*/RTP - IR/LRADC - Ethernet MAC de 10/100/1000 Mb/s - 3 USB - 8 UARTS - 4 SPI	- Controlador de memória DDR2/DDR3/DDR3L - Controlador NAND Flash com ECC de 64 *bits* - SD/MMC	- CupeBoard 2 (arduino solutions: 77,67 €) - Cupietruck (dfrobot: 71,13 €) - Banana Pi (the pi dish: £49,99) - pcDuino3 (exp-tech: 67,76 €)
Freescale i.MX6 Quad [85]	- 4 ARM Cortex-A9 com acelerador média NEON SIMD, até 1,2 GHz	- Porta paralela para câmera (até 20 *bits* e até 240 MHz) - Porta série MIPI CSI-2 para câmera (suporta até 1 Gbps se forem usadas 3 pistas, e até 800 MHz se usadas todas as 4 pistas)	- Possui 5 aceleradores de *hardware:* VPU, IPUv3H, GPU3Dv4, GPU2Dv2, GPUVG e ASRC - Posui suporte em *hardware* para codecs H.264 BP/CBP, MPEG-4 SP, H.263 P0/P3 e MJPEG BP. - Desempenho de codificação H.264 BP até 1920x1088@30fps - Conversão de espaços de cor - Redimensionamento de imagem - Rotação de imagem	- Porta paralela de 24 *bits* para visualizador; - Portas série LVDS (uma porta de até 255 Mpixeis/segundo ou duas de até 85 Mpixeis/segundo); - Porta HDMI 1.4 - MIPI/DSI, duas pistas a 1 Gbps	- 4 portas USB 2.0 - PCIe 2.0 - SSI - ESAI - 5 portas UART - 5 eCSPI - 3portas I2C - Ethernet MAC de 10/100/1000 Mb/s - Porta para *Keypad* 8x8 - SPDIF - Interface SATA II de 3Gbps para discos rígidos	- ROM de inicialização - RAM de 256 kB - Interface de 16, 32 e 64 *bits* para DDR2 e DDR3 - Interface de 8 *bits* para NAND-Flash - Interface de 16 ou 32 *bits* para NOR Flash - Interface de 16 ou 32 *bits* para PSRAM - 4 portas MMC/SD/SDIO	- Wandboard Quad (Mouser electronics: 111,20 €) - CompuLab CM-FX6 (a partir de 49$) + SBC-FX6 (a partir de 75$)
Broadcom BCM2835 [86]	- ARM1176JZF-S, até 700 MHz	- ISP (*image sensor pipeline*) avançado para câmeras	- Inclui 1 co-processador multimédia VideoCore IV de dois núcleos - Inclui um GPU VideoCore - Suporte em *hardware* para o codec de video H.264 HP - Desempenho de codificação e descodificação de resolução Full HD (1080p@30fps)	- Vídeo/RCA composto - Porta HDMI	- 2 portas USB 2.0 - I2C - 3 SPI - 2 UART	- Ethernet MAC de 10/100 Mb/s - Interface para certão SD/MMC/SDIO - SDRAM	- RasperryPi model B (mixtronica: 34,75 €)

SoC	Processadores	Entrada de vídeo	Processamento de vídeo	Saída de vídeo	Comunicação	Memória	Placas de avaliação
Ambarella A9 4K Ultra HD [87]	- 2 ARM cortex - A9 com NEON, até 1 GHz - Ambarela Video DSP - Ambarela Image DSP	- 12 pistas SLVDS/HISPi/sub LVDS, 4 pistas MIPI, ou 16 *bit* - 1pista SLVDS/MIPI - BT.601/656/1120 - Entrada do sensor suporta processamente de 700 MP/s	- Suporte em *hardware* para codecs de vídeo H.264 e MJPEG - Desempenho de codificação de resoluções 4K ultra HD (1080p@120fps, 720p@240fps) - Estabilização de imagem electronica - AE/AWB/AF ajustáveis	- Saída RGB até 24*bit* - HDMI 1.4 - Padrões SD PAL/NTSC composto - BT.656/1120	- Ethernet MAC de 10/100/1000 Mb/s - USB 2.0 (host,device) - SPI - UART	- DDR3/DDR3L - Interface para 2 cartões SD	- A9 Camera Development Platform

Anexo H Compilação do SDK e inicialização da LeopardBoard DM368

Neste anexo são indicados os principais procedimentos que devem ser tomados no sentido de proceder à instalação e compilação do SDK da *RidgeRun,* bem como da inicialização da LeopardBoard.

H.1 Instalação do SDK

A instalação do SDK foi realizada de acordo com o guião em [50]. Este procedimento inclui essencialmente 3 etapas: (1) instalação do ARM *toolchain,* (2) instalação do DVSDK, e (3) instalação do SDK RidgeRun.

H.1.1 Instalação do ARM toolchain

O *toolchain* utilizado foi o CodeSourcery GCC, o qual pode ser obtido diretamente a partir do *link* referenciado em [88]. Após concluir-se a transferência do ficheiro de instalação, este deve ser executado em linha de comandos, pois a pasta de instalação dos ficheiros deve ficar numa localização que, através da interface gráfica, não é possível aceder. Assim sendo, deve-se digitar

sudo ./arm-2009q1-203-arm-none-linux-gnueabi.bin -i console

para que a instalação seja executada em linha de comandos. A dada altura deve ser digitado "1" de modo a ser realizada a *typical instalation*. Tenha-se especial atenção aquando da seleção da diretoria de instalação dos ficheiros, que deverá ser:

/opt/codesourcery/arm-2009q1.

Em *Choose link location* selecionar a diretoria *Default*.

H.1.2 Instalação do DVSDK (Digital Video Software Development Kit)

De seguida é necessário instalar o DVSDK, cujo ficheiro de instalação pode ser obtido a partir de [89]. Após o download do ficheiro, deve-se executá-lo em linha de comandos digitando

chmod +x dvsdk_myplatform-evm_4_02_00_06_setuplinux
./dvsdk_myplatform-evm_4_02_xx_xx_setuplinux.

É possível que seja requerida a instalação de alguns pacotes em falta no sistema operativo, sendo que o modo de fazê-lo é indicado na linha de comandos, mais especificamente o comando utilizado é:

sudo apt-get install nome_pacote.

Seguidamente, será apresentada uma interface gráfica a partir da qual é realizada a instalação. A determinada altura, é necessário indicar-se a diretoria onde se encontram instalados os ficheiros da *toolchain.* Após este passo, prossegue-se a instalação de acordo com os passos indicados na interface gráfica, não sendo necessário realizar qualquer alteração ao que já está definido.

H.1.3 Instalação do SDK RidgeRun

Para se obter o ficheiro de instalação, é necessário aceder à página de *Downloads* da RidgeRun, em [90], selecionar o SDK adequado para o processador DM368 e placa *Leopardboard.* Tendo-se realizado o *download* do ficheiro de instalação, deve-se executá-lo digitando, na linha de comandos, o seguinte:

sudo chmod a+x RidgeRun-SDK-DM36x-Turrialba-Eval-Linux-x86-Install.bin
./RidgeRun-SDK-DM36x-Turrialba-Eval-Linux-x86-Install.bin.

Tal fará abrir uma interface gráfica, a partir da qual, seguindo os passos indicados, facilmente se realiza a referida instalação. É aconselhável que se instalem os ficheiros na diretoria pessoal de trabalho, *$HOME/work/.*

H.2 Compilação do SDK

O SDK inclui algumas aplicações de exemplo que podem ser compiladas e executadas na placa. Na integração de novas aplicações deve-se ter em atenção alguns procedimentos importantes, cuja descrição é apresentada com mais detalhe no *link* referenciado em [50].

Primeiramente é necessário criar uma diretoria para a nova aplicação, a qual deve estar localizada na pasta *myapps,* que por sua vez se encontra na diretoria onde foram instalados os ficheiros do SDK. Nesta diretoria, para além do ficheiro com o programa da aplicação, deve conter um ficheiro de configuração, *Config,* o qual contém a lógica para posterior seleção se a aplicação deve ou não ser compilada com o SDK, e um ficheiro *Makefile,* o qual deve respeitar determinadas regras para que possa ser integrado com o SDK. A descrição destes ficheiros também se encontra em [50].

Para selecionar quais as aplicações que devem ser compiladas com o SDK, é necessário, através da linha de comandos, aceder à diretoria onde estão instalados os ficheiros do SDK, e escrever o comando

make config.

Será apresentada uma interface gráfica, a partir da qual é possível alterar diversas configurações do SDK. A seleção das aplicações a compilar é realizada em

"*User Applications".* Após a seleção das aplicações e a possível alteração de configurações deve-se sair de tal interface e guardar as alterações.

Antes de fazer a compilação é necessário ainda definir uma variável de sistema denominada *DEVDIR* e exportá-la para a diretoria de desenvolvimento (esta variável refere-se portanto à diretoria onde está instalado o SDK). Uma forma de fazê-lo é executando o comando

$(make env)

ainda na diretoria onde estão instalados os ficheiros do SDK. Procede-se então a compilação do SDK através do comando

make.

Seguidamente, a nova imagem para o sistema de ficheiros da placa é criada através do comando

make install.

H.3 Inicialização da placa

A inicialização da placa pode ser realizada de 7 modos diferentes: NAND, MMC/SD, UART, USB, SPI, EMAC e HPI. O modo SD é simples e prático, tendo-se optado por esse. A seleção deste modo é realizada através da interface gráfica apresentada ao digitar o comando

make config,

ou seja, tem de ser realizada antes da compilação. Mais precisamente, a seleção é realizada no menu "*Installer Configuration",* e em "*Firmware deployment mode*" escolher a opção "*Deploy all the firmware to an SD card*". Após guardar esta configuração e terminar o processo de compilação, o cartão SD deve ser inserido no PC, e o procedimento em [91] deve ser seguido de modo a efetuar a instalação do sistema de ficheiros no cartão.

Já na placa, a seleção deste modo na placa é feita colocando a combinação 010 no *dip switch* de controlo sinalizado com o retângulo amarelo na Figura 3.2. Após a placa estar conectada ao PC (utilizou-se a conexão UART) e devidamente alimentada (utilizou-se a conexão USB), o cartão SD deve ser inserido na respetiva interface. Para se poder aceder à porta série, e assim interagir com a placa através do PC, em linha de comandos deve ser executando o comando indicado em [92], ou seja

picocom –b 115200 –r –l /dev/ttyUSB0.

Anexo I Integração de um codec numa aplicação

Para se poder integrar um codec numa dada aplicação alguns procedimentos específicos devem ser tidos em conta. Nesta secção são descritos os procedimentos necessários à integração do codec de compressão de imagem JPEG, contudo a integração de outros codecs pode ser realizada seguindo procedimentos semelhantes.

Um passo fundamental consiste na declaração de determinadas bibliotecas na aplicação em C, de modo que as seguintes linhas de código devem ser incluídas nesse sentido:

```
#include <xdc/std.h>;
#include <ti/sdo/ce/Engine.h>;
#include <ti/sdo/ce/CERuntime.h>;
#include <ti/sdo/ce/osal/Memory.h>.
```

No caso de um codec para imagem é necessário incluir ainda

```
#include <ti/sdo/ce/image1/imgenc1.h>;
#include <ti/sdo/ce/image1/imgdec1.h>.
```

Para além da declaração de bibliotecas, uma série de outros elementos têm de ser incluídos. O esquema da Figura I.1 apresenta a ordem com que tal tem de ser realizado.

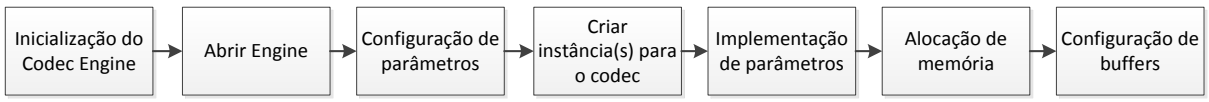


Figura I.1 Parte do procedimento de integração do codec JPEG na aplicação em linguagem C.

A inicialização da execução do Codec Engine é realizada através de

```
CERuntime_init().
```

De seguida é necessário abrir o *Engine* associado a determinadas configurações relativas aos codecs utilizados. Estas configurações incluem as diretorias para os ficheiros com tais codecs, bem como os nomes identificadores dos codificadores e descodificadores utilizados. Tal pode ser realizado com um comando semelhante a

```
Engine_Handle hEngine = NULL;
hEngine = Engine_open(engineName, NULL, NULL);
```

em que *Engine_Handle* é um tipo de dados para suporte da instância para o *Engine*, e *engineName* é uma *string* que contém o nome para o *Engine*. O próximo passo

consiste na configuração de alguns parâmetros necessários para criar as instâncias do codec. No caso do codificador, tais parâmetros pertencem às estruturas de dados IMGENC1_Params e IMGENC1_DynamicParams, enquanto no caso do descodificador pertencem às estruturas de dados IMGDEC1_Params e IMGDEC1_DynamicParams. É através destas estruturas que se definem parâmetros como o fator de qualidade e a resolução da imagem. As instâncias para o codificador e descodificador podem ser criadas a partir dos comandos

```
IMGENC1_Handle enc;
IMGDEC1_Handle dec;
enc = IMGENC1_create(hEngine, encoderName, &params);
dec = IMGDEC1_create(hEngine,decoderName,&decParams);
```

em que *IMGENC1_Handle* é uma estrutura para suporte de uma instância para codificação, IMGDEC1_Handle é uma estrutura para suporte de uma instância para descodificação, *encoderName* e *decoderName* são *strings* identificadoras dos nomes do codificador e descodificador associados ao *hEngine*, e *params* e *decParams* são variáveis dos tipos IMGENC1_Params e IMGDEC1_Params, respetivamente. Já os comandos

```
status1 = IMGENC1_control(enc, XDM_SETPARAMS, &dynParams, &encStatus);
status2 = IMGDEC1_control(dec, XDM_SETPARAMS, &decDynParams, &decStatus);
```

são utilizados para a aplicação dos parâmetros relativos às estruturas IMGENC1_DynamicParams (identificado por *dynParams*) e IMGDEC1_DynamicParams (identificado por *decDynParams*). Nestes comandos, *status1* e *status2* são variáveis do tipo inteiro, e *encStatus* e *decStatus* são variáveis dos tipos IMGENC1_Status e IMGDEC1_Status, respetivamente.

A configuração dos *buffers* para guardar os dados originais, comprimidos e descomprimidos inclui a associação dos mesmos a determinadas porções de memória, de modo que a alocação da mesma tem de ser realizada antes. Os descritores dos *buffers* têm de ser configurados não só com a localização da respetiva memória alocada, como também com o número de *buffers* e com o tamanho de cada um deles. Os descritores representam argumentos na função responsável pelo processo de compressão, tal como

```
status1 = IMGENC1_process(enc, &inBufDesc, &encBufDesc, &encInArgs, &encOutArgs);
```

em que *inBufDesc* e *encBufDesc* são variáveis do tipo XDM1_BufDesc, descritoras dos *buffers* com os dados originais e os dados comprimidos, respetivamente. Já *encInArgs* e *encOutArgs* são variáveis dos tipos IMGENC1_InArgs e IMGENC1_OutArgs, respetivamente.

Após os processos de compressão e descompressão, e de todas as operações incluídas na aplicação, é necessário reverter alguns passos, tais como eliminar as instâncias criadas, fechar o *Engine,* desalocar a memória, entre outras.

Para além destes procedimentos, são ainda necessários outros no sentido de tornar possível a correta compilação da aplicação, com o codec funcionando bem. Estes procedimentos tiveram como base as aplicações dadas como exemplos na diretoria de instalação dos ficheiros do Codec Engine.

Alguns ficheiros começaram por ser copiados para a diretoria de trabalho, mais precisamente para a pasta onde se encontra o programa em C. Esses ficheiros foram:

- xdcrules.mak: Indica o comando para executar a "construção"; copiado de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples/buildutils
- config.bld: Ficheiro de configuração; copiado a partir de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples
- local.cfg: Ficheiro de configuração do executável da aplicação; copiado de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples/ti/sdo/ce/examples/apps/image1_copy
- package.bld: Ficheiro com os scripts de construção; copiado de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples/ti/sdo/ce/examples/apps/image1_copy
- package.xdc: Ficheiro de definição do nome do pacote de ficheiros de suporte para a "construção" (*build*) da aplicação com os codecs; copiado de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples/ti/sdo/ce/examples/apps/image1_copy
- Rules.make: Ficheiro com alguns caminhos para diretorias importantes; foi copiado a partir de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk
- xdcpaths.mak: Ficheiro com alguns comandos e com caminhos para diretorias importantes; foi copiado a partir de $(DEVDIR)/proprietary/dvsdk-4_02_00_06/dvsdk/codec-engine_2_26_02_11/examples

Tendo em conta as caraterísticas de cada um dos ficheiros enumerados, é evidente que alguns tiveram de sofrer alterações. As configurações no ficheiro local.cfg são essenciais. Neste ficheiro, a variável na linha 44 define o caminho para a diretoria onde se encontra o codificador e/ou descodificador que se pretende utilizar. No caso da aplicação com o padrão JPEG, em que se implementou o codificador e o descodificador, as linhas 44 e 45 foram substituídas pelo seguinte:

```
var encoder = xdc.useModule('ti.sdo.codecs.jpegenc.ce.JPEGENC');
```

var decoder = xdc.useModule('ti.sdo.codecs.jpegdec.ce.JPEGDEC');

Nas últimas linhas deste ficheiro, tenha-se ainda atenção ao seguinte:

- O nome entre aspas seguido à variável *myEngine*, por exemplo *image_encode* na linha

  var myEngine = Engine.create("image_encode",

  é utilizado para a identificação do *Engine* no programa em C, mais precisamente é o nome associado à variável *engineName* do tipo *string* referida anteriormente;
- O nome entre aspas a seguir a *mod* corresponde à variável definida na linha 44 (neste caso é *encoder*);
- O nome entre aspas a seguir a *name* corresponde ao nome da diretoria que contém os ficheiros do codificador/descodificador referenciado pela variável declarada na linha 44 (neste caso, a variável *encoder* faz referência à diretoria *jpegenc*). Este é o nome que identifica o codificador/descodificador no programa em C, e é o mesmo que tem de estar associado à variável *encoderName* do tipo *string* referida anteriormente.

No caso do ficheiro local.cfg utilizado para a aplicação referente ao padrão JPEG, estas linhas foram substituídas por

```
var myEngine = Engine.create("image_encode", [
  {
    name: "jpegenc",
    mod  : encoder,
    local: true,
        groupId: 0
  },
  {
    name: "jpegdec",
    mod  : decoder,
    local: true,
        groupId: 0
  }
]);
```

No ficheiro package.bld a alteração tem de ser realizada na linha 126, em que o nome entre aspas associado á variável *srcs* tem de ser o mesmo dado ao ficheiro que contém o programa em C. No ficheiro package.xdc basta especificar o caminho para a diretoria onde se encontra o ficheiro com o programa em C, a partir de *DEVDIR.* Em Rules.make, na última linha, *EXEC_DIR* deve ser alterado para a diretoria onde se pretende que o executável seja copiado. Já no ficheiro

xdcpaths.mak é necessário especificar uma série de diretorias (o ficheiro descreve que diretorias são estas, ou seja, a que se referem), nomeadamente:

- *CE_INSTALL_DIR;*
- *XDC_INSTALL_DIR;*
- *XDAIS_INSTALL_DIR;*
- *FC_INSTALL_DIR;*
- *CMEM_INSTALL_DIR;*
- *CGTOOLS_V5T;*
- *RULES_MAKE.*

Existem outras alterações que podem ser realizadas neste ficheiro no sentido de simplificar o processo de compilação, contudo não são aqui tratadas por serem opcionais.

Naturalmente que o Makefile também tem de ser alterado, nomeadamente adicionaram-se algumas diretorias e algumas *flags*. O conteúdo do Makefile utilizado para o codec JPEG é o seguinte:

```
include xdcpaths.mak
include $(DEVDIR)/myapps/xdcrules.mak

.PHONY: build install clean

KERNEL_DIR    = $(DEVDIR)/kernel/src

LINUX_KERNEL_INSTALL_DIR=$(PSP_INSTALL_DIR)/linux-2.6.32.17-psp03.01.01.39
PACKAGE_DIR = $(CURDIR)/package
BIN_DIR = $(CURDIR)/bin/ti_platforms_evmDM365
XDCBUILDCFG=$(CURDIR)/config.bld

XDC_PATH                                                    :=
$(DEVDIR);$(XDC_PATH);$(CODEC_INSTALL_DIR)/packages;$(KERNEL_DIR)/include

BIN                 = app
SRCS                = $(BIN).c
OBJS                = $(SRCS:.c=.o)

CFLAGS += $(EXTRA_CFLAGS)
CFLAGS += $(EXTRA_CFLAGS) -I$(KERNEL_DIR)/include -I$(CMEM_INSTALL_DIR) -
I$(CE_INSTALL_DIR)/packages -I$(LINUX_KERNEL_INSTALL_DIR)/include -
I$(LINUXLIBS_INSTALL_DIR) -I$(FC_INSTALL_DIR)/packages -I$(XDC_INSTALL_DIR)/packages -
```

```
I$(XDAIS_INSTALL_DIR)/packages -Dxdc_target_name__=arm/GCArmv5T -
Dxdc_target__='<gnu/targets/std.h>'

    build:
        $(V)$(CC) -c $(CFLAGS) $(SRCS) -o $(OBJS) $(QOUT)
        $(V)$(CC) $(LDFLAGS) -o $(BIN) $(OBJS) $(QOUT)

    install:
        $(V)install -D -m 755 $(BIN_DIR)/app_local.xv5T $(FSROOT)/examples/$(BIN)
$(QOUT)

    install_sd:
        $(V)install -D -m 755 $(BIN_DIR)/app_local.xv5T /media/rootfs/examples/$(BIN)
$(QOUT)

    clean:
        $(V) -$(RM) -rf package/* bin/* *~ package.mak

    include ../../bsp/classes/rrsdk.class
```

Tendo-se realizado tais alterações, é possível compilar a aplicação a partir dos procedimentos indicados no Anexo H.

Anexo J Parâmetros de configuração do codificador H.264

A Tabela J.1 apresenta os principais parâmetros que podem ser configurados no codificador H.264.

Tabela J.1 Estruturas de dados associadas ao codificador H.264 da TI.

Estrutura de dados	Parâmetro	Descrição	Opções
H264VENC_Params	videncParams	Estrutura de dados, do mesmo tipo que IVIDENC1_Params	
	profileIdc	- Perfil do codificador H.264	(66) *Baseline;* (77) *Main;* (100) *High.*
	levelIdc	- Nível do codificador H.264; - Opções são definidas por IH264VENC_Level	(10) IH264VENC_LEVEL_10: nível 1.0; (até) (50) IH264VENC_LEVEL_50: nível 5.0.
	entropyMode	- Modo de codificação da entropia - Apenas suportado nos perfis *Main* e *High*	(0) CAVLC; (1) CABAC.
	encQuality	- Configuração no codificador	(0) Modo de compatibilidade com versão 1.1; (2) Modo Platinum, high quality (mesmo que encodingPreset = XDM_HIGH_QUALITY); (3) Modo Platinum, high speed (mesmo que encodingPreset = XDM_HIGH_SPEED).
IVIDENC1_Params	encodingPreset	- Algoritmo de criação de parâmetros de tempo específicos; - Opções são definidas por XDM_EncodingPreset.	(0) XDM_DEFAULT: equivalente a XDM_HIGH_QUALITY. (1) XDM_HIGH_QUALITY: (2) XDM_HIGH_SPEED; (3) XDM_USER_DEFINED: configuração definida pelo utilizador usando parâmetros avançados.

Estrutura de dados	Parâmetro	Descrição	Opções
IVIDENC1_Params	rateControlPreset	- Controlo da taxa de *bits* de codificação; - Opções são definidas por IVIDEO_RateControlPreset.	(0) IVIDEO_NONE: nenhum controlo é utilizado; (1) IVIDEO_LOW_DELAY: é usada uma taxa de *bits* constante (*Constant Bit Rate*, CBR); (2) IVIDEO_STORAGE: é usada uma taxa de *bits* variável (*Variable Bit Rate*, VBR); (3) IVIDEO_USER_DEFINED: configuração definida pelo utilizador usando parâmetros avançados; (5) IVIDEO_RATECONTROLPRESET_DEFAULT: equivalente ao IVIDEO_LOW_DELAY.
	maxWidth	- Comprimento máximo de cada *frame* em píxeis; - Os valores mínimo e máximo para este parâmetro dependem do valores definidos em *levelIdc* e em *encodingPreset.*	
	maxHeight	- Altura máxima de cada *frame* em píxeis; - Os valores mínimo e máximo para este parâmetro dependem do valores definidos em *levelIdc* e em *encodingPreset.*	
	max*Frame*Rate	- Valor máximo para a taxa de *frames*, em *frames* por segundo (fps), multiplicado por 1000; - Valor máximo é limitado pelo *levelId.*	
	inputChromaFormat	- Formato dos dados de entrada; - Opções são definidas por XDM_ChromaFormat; - Apenas suporta 1 formato.	(9) XDM_YUV_420SP: formato NV12.

Estrutura de dados	Parâmetro	Descrição	Opções
IVIDENC1_Params	inputContentType	- Tipo do conteúdo do vídeo de entrada.	(1) IVIDEO_PROGRESSIVE; (2) IVIDEO_INTERLACED.
	reconChromaFormat	- Formato dos dados nos *buffers* de reconstrução; - Opções são definidas por XDM_ChromaFormat; - Apenas suporta 1 formato.	(9) XDM_YUV_420SP: formato NV12.
IH264VENC_DynamicParams	videncDynamicParams	- É uma estrutura de dados do mesmo tipo que IVIDENC_DynamicParams.	
	rcAlgo	- Tipo de algoritmo para o controlo da taxa de *bits*; - Apenas usado se rateControlPreset = IVIDEO_USER_DEFINED.	(0) CBR; (1) VBR; (2) Parâmetro de quantização (PQ) fixo (*Fixed QP*); (3) CVBR; (4) *Custom RC1;* (5) *Custom CBR1;* (6) *Custom VBR1.*
	intra*Frame*QP	- Parâmetro de quantização das *frames* I no modo *Fixed QP;* - Apenas útil quando: rateControlPreset = IVIDEO_NONE, rcAlgo = 2, target*Bit*Rate = 0.	Valores inteiros de 0 a 51.
	interP*Frame*QP	- Parâmetro de quantização das *frames* P no modo *Fixed QP;* - Apenas útil quando: rateControlPreset = IVIDEO_NONE; rcAlgo = 2; target*Bit*Rate = 0.	Valores inteiros de 0 a 51.
IH264VENC_DynamicParams	initQ	- Parâmetro de quantização para a primeira *frame*; - Apenas útil quando rateControlPreset não é IVIDEO_NONE, e rcAlgo não é 2.	PQ's desde 0 a 51; (-1) O codificador calcula o PQ com base na taxa de *bits*, na taxa de *frames*, e na resolução da primeira *frame*.

Estrutura de dados	Parâmetro	Descrição	Opções
IH264VENC_DynamicParams	rcQMax	- Valor máximo do PQ que pode ser usado na codificação; - Apenas útil quando rateControlPreset não é IVIDEO_NONE, e rcAlgo não é 2.	PQ's desde rcQMin a 51.
	rcQMin	- Valor mínimo do PQ que pode ser usado na codificação; - Apenas útil quando rateControlPreset não é IVIDEO_NONE, e rcAlgo não é 2.	PQ's desde 0 a rcQMax.
	rcQMaxI	- Valor máximo do PQ que pode ser usado na codificação de *frames* I; - Apenas útil quando rateControlPreset não é IVIDEO_NONE, e rcAlgo não é 2.	PQ's desde rcQMinI a 51.
	rcQMinI	- Valor mínimo do PQ que pode ser usado na codificação de *frames* I; - Apenas útil quando rateControlPreset não é IVIDEO_NONE, e rcAlgo não é 2.	PQ's desde 0 a rcQMaxI.
IVIDENC_DynamicParams	size	- Tamanho desta estrutura de dados em *bytes.*	IVIDENC1_DynamicParams: Deve ser usado se não se pretender definir parâmetros avançados relativos ao padrão H.264; IH264VENC_DynamicParams: Deve ser usado se se pretender definir parâmetros avançados relativos ao padrão H.264.
	inputWidth	- Comprimento da *frame* atual em píxeis; - Valor mínimo depende do valor definido em *encodingPreset;* - Valor máximo não ultrapassar o valor maxWidth.	Tem de ser múltiplo de 16.

Estrutura de dados	Parâmetro	Descrição	Opções
IVIDENC_DynamicParams	inputHeight	- Altura da *frame* atual em píxeisM; - Valor mínimo depende do valor definido em *encodingPresetM;* - Valor máximo não ultrapassar o valor maxHeight.	Se inputContentType = 1: tem de ser múltiplo de 16; Se inputContentType = 2: tem de ser múltiplo de 32;
	target*Frame*Rate	- Valor da taxa de *frames* do ficheiro de entrada que se pretende que seja suportada, em fps, multiplicado por 1000; - Valor máximo é limitado pelo *maxFrameRate.*	Tem de ser múltiplo de 0,5.
	intra*Frame*Interval	- Intervalo (em *frames*) entre duas *frames* tipo I consecutivas.	(0) Apenas a primeira *frame* é do tipo I; (1) Apenas são usadas *frames* do tipo I; (2) É usada uma *frame* do tipo P entre cada duas *frames* I (IPIPIP…); (3) São usadas duas *frames* do tipo P entre cada duas *frames* I (IPPIPPIPP…); E assim sucessivamente…

Anexo K Códigos das aplicações desenvolvidas em linguagem C

Este anexo é constituído pelos códigos das quatro aplicações desenvolvidas em linguagem C Todos eles encontram-se em formato digital na pasta Anexo K.

- O ficheiro com o código relativo à aplicação desenvolvida para análise da compressão JPEG denomina-se JPEGcodingApp.c
- O ficheiro com o código relativo à aplicação desenvolvida para análise da compressão H.264 denomina-se H264codingApp.c
- O ficheiro com o código relativo à aplicação desenvolvida para análise dos algoritmos de processamento de imagem denomina-se ImgProcAlgo.c.
- O ficheiro com o código relativo à aplicação para captura de imagens ou vídeos, processamento de imagem, compressão, e armazenamento ou transmissão para saída DVI denomina-se App.c

Os restantes ficheiros são os necessários à integração dos codecs nas aplicações (referidos no Anexo I), e à compilação dos códigos.

Anexo L Dados utilizados e obtidos nos testes à compressão H.264

Os vídeos utilizados e obtidos nos testes ao codificador H.264 estão em formato digital na pasta Anexo L, e estão organizados em cinco pastas, cada uma relativa às diferentes versões de “Stefan”, “Hall”, “Davinci”, “Big Buck Bunny” e “Solar Coronal Holes”. Em cada uma delas é possível encontrar um arquivo zip que contém a imagem original no formato yuv e as versões descomprimidas com as quais foi calculado o EQM, e as versões comprimidas de acordo com os diferentes modos de compressão definidos. Os ficheiros denominados por *nomeOriginal_modo.264* referem-se a versões comprimidas do vídeo original, *nomeOriginal.yuv,* com um modo de compressão *modo,* enquanto as versões descomprimidas são denominadas por *nomeOriginal_modo.yuv.* Na pasta Anexo L também pode ser encontrado o código utilizado para calcular o EQM de cada vídeo através do Matlab, mais precisamente no ficheiro “videoEQM.m”.

O campo *nomeOriginal* para cada vídeo utilizado é:

- “bbb_1088p_NV12” para “Big Buck Bunny;
- “solar103” para “Solar Coronal Holes”;
- “davinci_720p_NV12” para “Davinci”;
- “stefan_cif_nv12” para “Stefan”;
- “hall_cif_nv12” para “Hall”.

Os vídeos *raw* (.yuv) podem ser lidas usando *softwares* como o ImageJ e o 7yuv, enquanto ao vídeo comprimidos podem ser lidos com o *software* MPC-HC. Juntamente com os vídeos, cada pasta contém também o ficheiro de texto criado pela mesma aplicação, que contém diversos dados relativos a cada um dos processos de compressão. Esse ficheiro está denominado com o formato *Resultados_nomeOriginal.txt.*

Em cada uma das referidas pastas também é possível encontrar uma pasta denominada “EQM”. Esta pasta contém os ficheiros de texto resultantes do cálculo do EQM através do Matlab, para cada um dos vídeos. Em cada ficheiro apresentam-se os valores do EQM para cada *frame* do vídeo a que se refere, por ordem de reprodução. O nome dos ficheiros tem o formato *mse_nomeOriginal_modo.txt,* com *mse* sendo comum a todos, e *nomeOriginal* e *modo* com o significado referido anteriormente.

Quanto aos modos de compressão utilizados na codificação dos vídeos, são 16 no total. Os parâmetros com que o codificador foi configurado para cada um desses modos de compressão apresentam-se na Tabela L.1.

Tabela L.1. Modos de compressão utilizados nos testes ao codificador H.264.

Modo de compressão		Videoconferência	Videovigilância_st	Videovigilância_dp	Broadcast	Armazenamento_lE	Armazenamento_hE	NoFilter	intraF0	intraF15	intraF30	PQ0	PQ8	PQ18	PQ28	PQ38	PQ51
Perfil		Baseline	High														
Nível		5.0															
Taxa de *frames* (fps)		30															
Taxa de *bits* máxima (kbps)	CIF	512	1000	512	1000	512	1000	512	2000			Não aplicável (0)					
	720p	3000	7000	4000	7000	4000	14000	4000									
	1440x1440																
	1088p	8000	16000	8000	16000	8000	22000	8000									
Filtro de remoção de artefatos		ON						OFF	ON								
Slices (*bits*)		0			10000	0											
Periodicidade *frames* tipo I		0 (apenas a primeira *frame*)	30	15	15	30		15	1	15	30	30					
Algoritmo de controlo da taxa de *bits*		CBR				VBR		CBR	VBR			VBR					
PQ	Mínimo	8										Não aplicável					
	Máximo	44															
	Fixo	Não Aplicável										0	8	18	28	38	51

Modo de compressão		Videoconferência	Videovigilância_st	Videovigilância_dp	Broadcast	Armazenamento_lE	Armazenamento_hE	NoFilter	intraF0	intraF15	intraF30	PQ0	PQ8	PQ18	PQ28	PQ38	PQ51
Modo do codificador		Platinum (elevada qualidade)	Versão 1.1		Platinum (elevada qualidade)			Versão 1.1	Platinum (elevada qualidade)								
Capacidade do buffer VBV relativamente à taxa de *bits* (%)		50	500	50	600			50	200								
Método de entropia		CAVLC	CABAC			CAVLC	CABAC										
Transformada 8x8 em *frames* I		OFF	ON			OFF	ON										
Nº de macroblocos codificados no modo intra	CIF	4	0														
	720p	30															
	1440x1440																
	1088p	90															

Os modos denominados por *PQx* referem-se áqueles em que não foi utilizada qualquer controlo da taxa de *bits*, tendo sido utilizado um parâmetros de quantização fixo indicado pelo valor *x.* Quanto aos modos denominados por *intraFy,* referem-se a modos de compressão em que a periodicidade de *frames* do tipo I foi variada. No modo em que *y* está a 0 o vídeo foi codificado apenas com *frames* do tipo I, no modo com *y* a 15 o vídeo contém 1 *frame* do tipo I após 14 *frames* do tipo P, e no modo com *y* a 30 o vídeo contém 1 *frame* do tipo I após 29 *frames* do tipo P.

Anexo M Imagens utilizadas e resultantes dos testes aos algoritmos de processamento de imagem

As imagens utilizadas nos testes ao desempenho dos algoritmos de processamento de imagem encontram-se em formato digital na pasta Anexo M. Esta pasta contem outras sete pastas, uma para os resultados de cada imagem utilizada, denominadas por: Tiffany, View, F16, Couple, Baboo_dark, Baboo_light, Lena.

Em cada uma delas é possível encontrar o ficheiro original, cujo nome tem o formato *nomeOriginal.yuv,* uma versão comprimida do mesmo sem a aplicação de qualquer algoritmo, designado com o formato *nomeOriginal_0_F2.jpg,* e as versões da imagem original alteradas com diferentes algoritmos e posteriormente comprimidas. Para o último caso, o formato do nome do ficheiro que contém cada imagem é *nomeOriginal_algoritmos_numero_F2.jpg.* Cada uma destas pastas contém também o ficheiro de texto gerado durante os vários processos de compressão, denominado por *Resultados_fromFile2_nomeOriginal.txt,* que contém alguns dados úteis para a análise realizada.

O significado de cada campo é o seguinte:

- *nomeOriginal* é o nome da imagem original utilizada para os diversos testes;
- *algoritmos* indica os algoritmos que foram aplicadas à imagem, por ordem de aplicação;
- *numero* é um valor indentificador dos algoritmos aplicados.

O campo *nomeOriginal,* para cada imagem, é o seguinte:

- “bGirl” para “Tiffany”;
- “couple_v1” para “Couple”;
- “plane” para “F16”;
- “lena” para “Lena”;
- “monkey_dark” para “Baboo_dark”;
- “monkey_light” para “Baboo_light”
- “view” para “View”.

O campo *algoritmos* é definido da seguinte forma:

- Contém “ct” se o algoritmo de alteração de contraste tiver sido aplicado;
- Contém “lm” se o algoritmo de alteração da luminosidade tiver sido aplicado;
- Contém “ht” se o algoritmo de equalização do histograma tiver sido aplicado;
- Contém “gr” se o algoritmo de balanço de brancos tiver sido aplicado;

Anexo N Dados usados e obtidos nos testes à compressão JPEG

Neste anexo apresentam-se os dados obtidos nos testes à compressão JPEG, bem como é feito um tratamento estatístico dos mesmos.

N.1 Imagens utilizadas e obtidas

As imagens utilizadas e obtidas nos testes ao codificador JPEG encontram-se em formato digital na pasta denominada Anexo N. Esta pasta contém outras cinco pastas, cada uma relativa às diferentes versões das imagens “Lena”, “Builds”, “Red”, “Azores” e “Testpat”, e um ficheiro em formato PDF denominado por “Inquéritos” que contém as respostas dadas nos inquéritos realizados.

Em cada uma das pastas é possível encontrar o ficheiro original no formato yuv, as versões comprimidas com fatores de quantização diferentes, e as versões descomprimidas com as quais foi calculado o EQM. Os ficheiros denominados por *nomeOriginal_fator.jpg* referem-se a versões comprimidas da imagem original, *nomeOriginal.yuv,* com um fator de qualidade *fator,* enquanto as versões descomprimidas são denominadas por *nomeOriginal_fator_o.yuv.*

O campo *nomeOriginal* para cada imagem utilizada é:

- “azores_1920by1200” para “Azores;
- “builds” para “Builds”;
- “lenna512” para “Lena”;
- “red” para “Red”;
- “testpat1024” para “Testpat”.

As imagens *raw* (.yuv) podem ser lidas usando *software* como o ImageJ e o 7yuv.

Em cada uma das referidas pastas também se encontra um ficheiro denominado *Resultados_nomeOriginal.txt,* que se refere ao ficheiro de texto criado pela aplicação de testes durante a compressão das imagens com os diferentes fatores de qualidade. Os dados recolhidos nestes ficheiros estão organizados na Tabela N.1.

Tabela N.1. Rácio de compressão, EQM e N_b relativos às versões comprimidas das várias imagens com diferentes fatores.

Imagem	Fator de qualidade	Rácio de compressão	EQM	N_b (*bits*/píxel)	Tempo de compressão (µs)
Builds	97	5,18:1	0,441	3,018	31260
	82	10,19:1	10,710	1,568	31192
	72	13,09:1	14,266	1,222	31194
	62	16,05:1	20,795	0,996	31139
	52	17,74:1	25,674	0,901	31281
	42	20,59:1	31,127	0,776	31121
	32	24,01:1	39,450	0,666	31248
	22	29,59:1	55,535	0,540	31114
	12	41,59:1	93,227	0,384	31100
	2	80,23:1	337,530	0,199	31486
Azores	97	8,29:1	0,351	1,928	31236
	82	14,57:1	0,435	1,098	31138
	72	16,50:1	3,251	0,969	31128
	62	23,00:1	7,480	0,695	31088
	52	26,66:1	8,341	0,600	31104
	42	28,21:1	10,356	0,566	31095
	32	36,60:1	13,996	0,437	31077
	22	44,31:1	20,191	0,361	31085
	12	60,39:1	36,497	0,264	31058
	2	93,91:1	159,185	0,170	31495
Lena	97	3,85:1	1,763	4,146	4743
	82	10,93:1	7,308	1,462	4711
	72	14,54:1	9,772	1,100	4704
	62	17,77:1	11,938	0,900	4713
	52	20,60:1	13,919	0,776	4810
	42	23,60:1	16,101	0,677	4709
	32	28,09:1	19,442	0,569	4722
	22	34,63:1	25,565	0,462	4713
	12	47,50:1	42,904	0,336	4686
	2	79,11:1	205,121	0,202	5139

Imagem	Fator de qualidade	Rácio de compressão	EQM	N_b (*bits*/píxel)	Tempo de compressão (µs)
Red	97	4,66:1	1,398	3,431	4756
	82	12,82:1	4,487	1,247	4747
	72	16,91:1	5,790	0,946	4713
	62	20,84:1	7,085	0,767	4715
	52	24,19:1	8,334	0,661	4709
	42	28,06:1	9,784	0,570	4707
	32	33,70:1	12,020	0,474	4718
	22	41,92:1	16,143	0,381	4703
	12	55,08:1	27,253	0,290	4696
	2	79,78:1	175,477	0,200	5257
Testpat	97	15,27	0,217	1,047	15127
	82	22,55	2,997	0,709	15085
	72	25,6	6,044	0,624	15086
	62	27,75	8,983	0,576	15079
	52	30,23	13,251	0,529	15081
	42	32,62	17,027	0,49	15083
	32	34,75	24,95	0,46	15112
	22	40,24	37,604	0,397	15075
	12	47,81	68,51	0,334	15075
	2	63,6	191,391	0,251	15538

Os dados dos inquéritos foram tratados através de uma regressão linear realizada no *software* Microsoft Excel. O procedimento realizado e a análise dos resultados encontram-se descritos de seguida.

N.2 Tratamento estatístico dos dados dos inquéritos

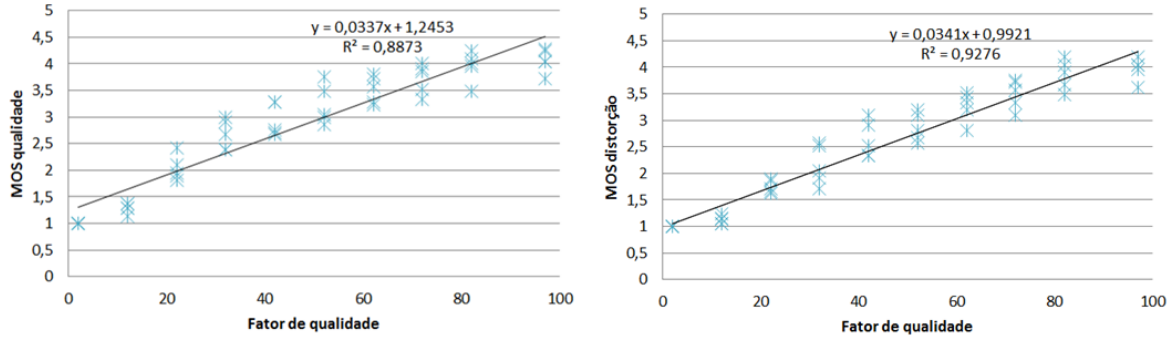
Os dados dos inquéritos foram utilizados para calcular o MOS, dado pela equação (2.13) na secção 2.2.4**.** A Tabela N.2 apresenta a classificação MOS para cada uma das imagens comprimidas com diferentes fatores de qualidade. Já na Figura N.1 os mesmos dados encontram-se representados através de diagramas de dispersão, mas sem qualquer referência às respetivas imagens. Através dos mesmos computou-se, no *software* Microsoft Office Excel, uma linha de tendência linear no sentido de procurar uma aproximação linear à relação entre os parâmetros em análise, a qual também pode ser observada na Figura N.1. Através da ferramenta de análise "Regressão" disponível neste *software* foram obtidas mais

algumas medidas que permitem interpretar os dados recolhidos. Neste processo foi utilizado um nível de confiança de 0,90.

Tabela N.2. Classificação de cada imagem comprimida em termos de qualidade e distorção, obtidada através do método MOS.

Imagem	Fator de qualidade	MOS qualidade	MOS distorção
Lena	97	4,05	4,05
	82	4,00	3,90
	72	3,86	3,57
	62	3,57	3,43
	52	3,00	2,67
	42	2,76	2,52
	32	2,67	2,05
	22	1,95	1,67
	12	1,38	1,05
	2	1,00	1,00
Red	97	3,71	3,62
	82	3,48	3,48
	72	3,33	3,33
	62	3,29	3,19
	52	3,05	2,81
	42	2,67	2,33
	32	2,38	1,91
	22	1,81	1,71
	12	1,14	1,15
	2	1,00	1,00
Testpat	97	4,05	3,95
	82	3,95	3,67
	72	3,52	3,10
	62	3,24	2,81
	52	2,86	2,57
	42	2,71	2,33
	32	2,38	1,71
	22	1,90	1,62
	12	1,29	1,24
	2	1,00	1,00
Azores	97	4,29	4,05
	82	4,24	4,19
	72	4,00	3,76
	62	3,81	3,52
	52	3,76	3,19

Imagem	Fator de qualidade	MOS qualidade	MOS distorção
Azores	42	3,29	2,90
	32	2,90	2,52
	22	2,10	1,90
	12	1,29	1,14
	2	1,00	1,00
Builds	97	4,24	4,19
	82	4,10	4,05
	72	3,90	3,71
	62	3,71	3,33
	52	3,48	3,10
	42	3,29	3,10
	32	3,00	2,57
	22	2,43	1,86
	12	1,38	1,05
	2	1,00	1,00



(a) MOS qualidade (b) MOS distorção

Figura N.1 Diagrama de dispersão do MOS para a qualidade e do MOS para a distorção, em função do fator de qualidade.

Relativamente à qualidade, os resultados obtidos através da regressão mostram que a correlação de Pearson entre o fator de qualidade e a classificação MOS é de 0,94, ou seja, 94%. Elevando este valor ao quadrado obtém-se 0,89, o que significa que 89% da classificação MOS pode ser estimada linearmente a partir do fator de qualidade. Para além disso, o valor obtido para a significância encontra-se na ordem de 10^{-24}. Tais dados indicam que o modelo linear parece representar adequadamente a relação entre as variáveis, embora o diagrama de dispersão tenha alguma sugestão de curvatura, com um crescimento rápido do MOS para valores mais baixos do fator de qualidade e alguma saturação para valores mais altos do fator de qualidade. A mesma conclusão se aplica ao modelo linear relativo à classificação MOS para a distorção, uma vez que está associado a uma correlação

de Pearson igualmente elevada, de 96%, e a uma significância bastante reduzida, na ordem de 10^{-29}.

A categoria MOS pode, portanto, ser estimada a partir do valor do fator de qualidade tendo em conta um desvio padrão de 0,36, para o caso das categorias de qualidade, e um desvio padrão de 0,29, para o caso das categorias de distorção. Tais valores são satisfatórios, uma vez que se tem um total de 5 categorias distanciadas de 1 unidade, tanto para a qualidade como para a distorção. Em função do declive da reta que descreve a aproximação linear relativa à classificação MOS para a qualidade, é esperado que uma variação de $0{,}0337^{-1} \cong 30$ unidades no fator de qualidade cause a alteração da categoria de qualidade MOS. Para a distorção, a respetiva variação espera-se que seja ligeiramente menor, de cerca $0{,}0341^{-1} \cong 29$ unidades no fator de qualidade.

Pretendeu-se obter uma previsão do intervalo de fatores de qualidade que devem ser utilizados quando se pretende uma imagem que se encaixe numa dada categoria MOS. Para tal, recorreu-se ao método de predição inversa descrito em [93]**,** no qual os intervalos inferior e superior são obtidos pela equação

$$lim = \bar{x} + b\frac{y_i - \bar{y}}{K} \mp \frac{t_{\frac{\alpha}{2},n-2}}{K}\sqrt{s_{y,x}^2\left[\frac{(y_i-\bar{y})^2}{\sum_{i=1}^{n}(x_i-\bar{x})^2} + K\left(1+\frac{1}{n}\right)\right]}, \quad \text{(N.1)}$$

com

$$K = b^2 - s_b^2\left(t_{\frac{\alpha}{2},n-2}\right)^2 \quad \text{(N.2)}$$

em que $\bar{x}$ e $\bar{y}$ são os valores médios do fator de qualidade e da categoria MOS, respetivamente, b é o declive da reta resultante da aproximação linear, n é o número total de amostras, s_b é o erro padrão da estimativa do declive (cujos valores foram referidos anteriormente), $s_{y,x}$ é o erro padrão da estimativa (obtido através da regressão), e $t_{\frac{\alpha}{2},n-2}$ é um valor que se pode obter no Excel através da função “INVT”, cujos parâmetros de entrada são $\frac{\alpha}{2}$, sendo α o valor resultante da subtração do nível de confiança utilizado ao valor 1, ou seja, $\alpha = 1 - 0{,}90 = 0{,}10$, e o número total de amostras, n, subtraído por 2. Os limites obtidos a partir da aplicação deste método aos resultados das regressões relativas ao MOS para a qualidade e ao MOS para a distorção apresentam-se na Tabela N.3.

Tabela N.3. Intervalo de valores para o fator de qualidade obtidos para cada categoria MOS através do método de previsão inversa.

Número da categoria MOS	Intervalo de fatores de qualidade			
	Qualidade		Distorção	
	Limite Inferior	Limite Superior	Limite Inferior	Limite Superior
1	-30,13	14,37	-17,43	17,27
2	-0,10	44,41	12,09	46,79
3	29,94	74,44	41,62	76,31
4	59,97	104,48	71,14	105,84
5	90,01	134,52	100,66	135,36

Pela observação dos resultados na Tabela N.3, verifica-se que a categoria MOS de valor 5 para a distorção tem um limite inferior que excede o valor máximo permitido para o fator de qualidade (que é 97, como indicado na secção 3.2.1), o que significa que em nenhuma das imagens analisadas este nível de distorção foi alcançado, nem mesmo com o maior fator de qualidade máximo possível.

Entre as possíveis razões para se ter obtido este resultado estão o facto de esta análise ter sido realizada com um reduzido número de amostras, e ter sido utilizada uma abordagem linear para procurar a relação entre os dados de entrada quando, na verdade, a relação entre estes apresenta-se mais curvilínea. Uma vez que os valores MOS medidos tendem a manter-se constantes para os fatores de qualidade nos extremos enquanto e os preditos através da reta continuam a aumentar ou a diminuir, a diferença entre o MOS estimado e o MOS real tende a aumentar nestas zonas.

Tendo em conta que fator de qualidade varia desde 2 a 97 (ver secção 3.2.1), torna-se necessário truncar os limites obtidos, uma vez que estes, nalguns casos, ultrapassam o intervalo possível para o fator de qualidade. Considere-se os resultados da Tabela N.3 após a realização da truncagem. Verifica-se que os limites entre categorias consecutivas apresentam alguma sobreposição, o que significa que nestas zonas de sobreposição é difícil prever se a categoria MOS da imagem será a maior ou a menor associada a esse intervalo de fatores de qualidade.

Anexo O Compressão e descompressão JPEG de um bloco de 8x8 em Matlab

No sentido de ilustrar como são transformados os dados ao longo de um processo de compressão JPEG, assim como para se entender como varia o rácio de compressão para diferentes cenários, desenvolveu-se um algoritmo em Matlab para realizar a compressão JPEG de blocos 8x8 de componentes luma. Fez-se uso da tabela de quantização referenciada como Tabela K.1 em [75], da tabela de *Huffman* para os coeficientes AC referenciada como Tabela K.5 em [75], e da tabela de *Huffman* para os coeficientes DC referenciada como Tabela K.3 em [75]. O código pode ser analisado no ficheiro “JPEGencoding.m” em formato digital na pasta Anexo O.

As superfícies alvo da compressão encontram-se no espaço de cor YCbCr, e apresentam-se na Figura O.1. Como se pode observar, a imagem em (a) é apenas de uma cor (preta), e serve para ilustrar como ocorre a compressão nestas superfícies cuja frequência é nula, e o que acontece na compressão de imagens em escala de cinza cujas componentes crominância têm valor constante em toda a imagem. Na imagem em (b) o valor da luminância já apresenta alguma variação, sendo caraterizada por componentes de baixa frequência. Já a imagem em (c) apresenta mudanças rápidas no valor da luminância, sendo caraterizada por uma componente de alta frequência.

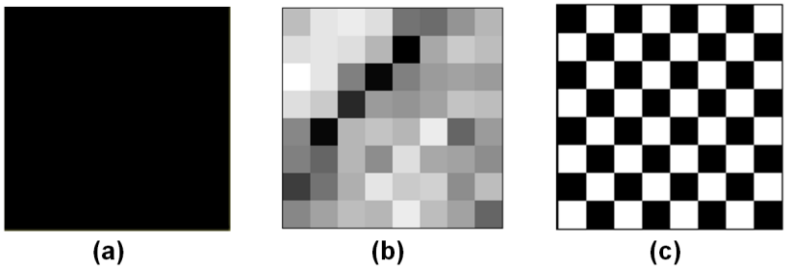


Figura O.1 Superfícies alvo de compressão JPEG no algoritmo em Matlab.

As matrizes na Figura O.2 apresentam os valores das componentes luma das imagens da Figura O.1.

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

(a)

16	235	16	235	16	235	16	235
235	16	235	16	235	16	235	16
16	235	16	235	16	235	16	235
235	16	235	16	235	16	235	16
16	235	16	235	16	235	16	235
235	16	235	16	235	16	235	16
16	235	16	235	16	235	16	235
235	16	235	16	235	16	235	16

(b)

99	105	106	104	88	87	93	98
102	108	104	105	104	98	71	96
94	96	104	102	109	105	90	72
69	91	95	101	110	107	104	101
100	69	71	97	102	92	95	105
144	153	135	68	79	113	98	103
157	147	156	170	122	76	101	104
184	165	174	158	163	150	73	104

(c)

Figura O.2. Valores da componente luma, (a), (b) e (c), correspondentes às imagens (a), (b) e (c) na Figura O.1, respetivamente.

O.1 Processo de compressão

Após aplicar-se o deslocamento de -128, e aplicar-se a DCT, os valores resultantes são os da Figura O.3.

-128	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a)

-128	72	-16	-6	10	-12	15	-2
-114	-67	-4	-1	-10	1	-13	-3
83	71	-13	-15	12	-5	12	-6
-33	-6	56	-14	-12	9	-7	2
-3	-36	-16	11	16	-2	5	-3
7	0	-8	-29	-7	20	-24	5
4	12	17	23	-37	-14	18	4
-15	-5	-16	-1	32	27	-13	3

(b)

-20	0	0	0	0	0	0	0
0	-28	0	-34	0	-50	0	-128
0	0	0	0	0	0	0	0
0	-34	0	-40	0	-59	0	-128
0	0	0	0	0	0	0	0
0	-50	0	-59	0	-89	0	-128
0	0	0	0	0	0	0	0
0	-128	0	-128	0	-128	0	-128

(c)

Figura O.3. Coeficientes em (a), (b) e (c) resultantes da aplicação do deslocamento de -128 e da DCT às componentes luma da Figura O.1 (a), (b) e (c), respetivamente.

Quanto às matrizes da Figura O.4 representam as matrizes da Figura O.3 após o processo de quantização. Como se pode verificar, para qualquer um dos casos, a gama de valores nas matrizes sofreram uma redução significativa, levando também à redução do número de *bits* necessários para representar tais valores. Pode-se verificar também que, na matriz da Figura O.3 (b), os coeficientes mais pequenos associados às maiores frequências foram quantizados para 0, perdendo-se assim informação relativa aos mesmos.

-56	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(a)

-10	7	-2	0	0	0	0	0
-10	-6	0	0	0	0	0	0
6	5	-1	-1	0	0	0	0
-2	0	3	0	0	0	0	0
0	-2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

-1	0	0	0	0	0	0	0
0	-2	0	-2	0	-1	0	-3
0	0	0	0	0	0	0	0
0	-2	0	-1	0	-1	0	-3
0	0	0	0	0	0	0	0
0	-1	0	-1	0	-1	0	-3
0	0	0	0	0	0	0	0
0	-2	0	-2	0	-3	0	-7

(c)

Figura O.4. Coeficientes em (a), (b) e (c) resultantes da quantização dos coeficientes em (a), (b) e (c) da Figura O.3, respetivamente.

Após os processos de reordenação *zig-zag* e codificação de entropia, os fluxos resultantes são os que se apresentam na Tabela O.1.

Tabela O.1 Fluxos de *bits* resultantes após a compressão.

Imagem	Fluxo de *bits* resultante	Tamanho (*bits*)	Rácio de compressão
Figura O.1 (a)	11100001111010	14	37:1
Figura O.1 (b)	1010101100111101101011001101000010101111111011101010111100011101100111010110100	80	6:1
Figura O.1 (c)	0100111110111011111111101100111011011111110000110001100011111111101111100111000110001100011011001111111101100111000110110011111110000011011001111110111000	149	3:1

Como se pode observar, o maior rácio de compressão foi alcançado para a imagem sem qualquer variação na luminância, seguido da imagem variações suaves na luminância, e por último, pela imagem com mais componentes de alta frequência.

O.2 Processo de descompressão

O processo de descompressão também foi realizado, de forma a reconstruir os dados comprimidos e se obterem as imagens resultantes. O código desenvolvido para tal encontra-se em formato digital no ficheiro “JPEGdecoding” na pasta Anexo F. Os dados reconstruidos são apresentados nas matrizes da Figura O.5, enquanto as respetivas imagens podem ser observadas na Figura O.6.

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

(a)

101	103	102	95	87	87	96	104
100	105	109	108	101	91	83	79
86	92	103	112	112	102	85	72
77	78	84	96	107	109	101	92
103	93	84	83	91	101	105	105
149	135	116	100	93	92	94	96
168	165	157	142	122	105	94	89
160	172	181	177	155	128	106	94

(b)

8	255	7	237	15	245	0	244
247	9	234	25	227	18	243	5
0	233	25	230	22	227	19	252
255	21	233	37	215	19	231	0
0	231	19	215	37	233	21	255
252	19	227	22	230	25	233	0
5	243	18	227	25	234	9	247
244	0	245	15	237	7	255	8

(c)

Figura O.5. Dados reconstruidos após descompressão dos fluxos de *bits* na Tabela O.1.

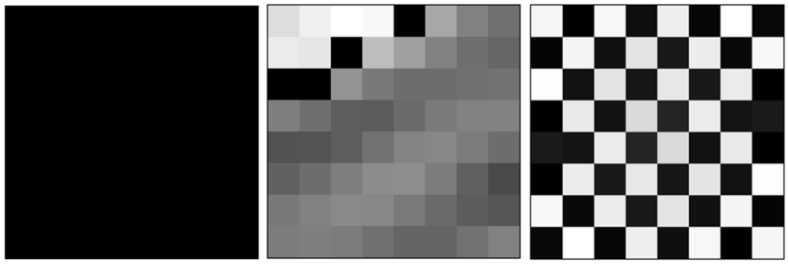


Figura O.6. Imagens reconstruídas após descompressão dos fluxos de *bits* na Tabela O.1.

Como se pode verificar, a imagem em que a luminância é constante é exatamente igual à original, não tendo apresentado qualquer perda devido à compressão. Já nas outras duas imagens, as diferenças em relação às originais são salientes. Tais diferenças devem-se à perda de informação durante o processo de quantização. Calculou-se ainda o EQM para cada uma destas imagens. Naturalmente que o valor obtido para a imagem na Figura O.6 (a) foi 0, enquanto para as imagens (b) e (c) na Figura O.6 obtiveram-se valores para o EQM de aproximadamente 159 (equivalente a PSNR de 26 dB) e 120 (equivalente a PSNR de 27 dB), respetivamente.

Anexo P Imagens e vídeos capturados através da aplicação para captura, processamento, compressão e armazenamento ou transmissão

O conteúdo deste anexo encontra-se em formato digital na pasta Anexo P. Esta contém duas pastas, uma com os vídeos e outra com as imagens capturadas, e o ficheiro de texto gerado pela aplicação com diversos dados a partir dos quais foi analisado o desempenho da aplicação.

As imagens encontram-se na pasta Imagens, e são denominadas com o formato *img_res_algo_0.jpg,* enquanto os vídeos estão na pasta Vídeos, e são denominados com o formato *video_res_algo.264*

O campo *res* indica a resolução da seguinte forma:

- “720p” respresenta resolução 1280x720;
- “512p” representa resolução 768x512;
- “480p” representa resolução 640x480;
- “960p” representa 1280x960;
- “1080p” representa 1920x1080.

O campo *algo* indica as alterações realizadas da seguinte forma:

- “dft” significa que nenhuma alteração foi aplicada à imagem ou vídeo capturado;
- “Brg” significa que a luminosidade foi alterada através do IPIPE;
- “Cont” significa que o contraste foi alterado através do IPIPE;
- “contBrg” significa que a luminosidade e contraste foram alterados através do IPIPE;
- “wb*k*” significa que o balanço de brancos foi alterado através do IPIPE, com os parâmetros relativos à combinação *k*;
- “sBrg” significa que a luminosidade foi alterada através do algoritmo em *software*;
- “sCont” significa que o contraste foi alterado através do algoritmo em *software*;
- “hist” significa que foi realizada a equalização do histograma.