

# **Universidade da Beira Interior**

## **Departamento de Informática**



**Departamento de  
Informática**

### **Nº 112 - 2019: *Codificação da Cor de Hologramas Digitais Usando Multivistas***

Elaborado por:

**Raquel Sofia Brás Guerra**

Orientador:

**Professora Doutora Maria Manuela Areias da Costa Pereira de  
Sousa**

2 de Setembro de 2020



# ***Agradecimentos***



# Conteúdo

<b>Conteúdo</b>	<b>iii</b>
<b>Lista de Figuras</b>	<b>v</b>
<b>Lista de Tabelas</b>	<b>vii</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Enquadramento . . . . .	1
1.2 Motivação . . . . .	1
1.3 Objetivos . . . . .	2
1.4 Organização do Documento . . . . .	2
<b>2 Estado da Arte</b>	<b>3</b>
2.1 Introdução . . . . .	3
2.2 Perspetiva Histórica . . . . .	3
2.3 Conceitos Base . . . . .	3
2.3.1 Holografia . . . . .	3
2.3.1.1 Princípios de Holografia . . . . .	3
2.3.1.2 Representação de Dados Holográficos . . . . .	4
2.3.1.3 Reconstrução de Holograma . . . . .	4
2.3.2 Compressão . . . . .	4
2.3.3 JPEG2000 . . . . .	4
2.3.4 Cor . . . . .	4
2.3.4.1 <i>Red, Green &amp; Blue</i> (RGB) . . . . .	4
2.3.4.2 YCbCr . . . . .	5
2.4 Estado da Arte . . . . .	5
2.5 Conclusões . . . . .	6
<b>3 Tecnologias e Ferramentas Utilizadas</b>	<b>7</b>
3.1 Introdução . . . . .	7
3.2 Tecnologias e Ferramentas . . . . .	7
3.2.1 Software de Reconstrução de Hologramas e sua Trans- crição para Python . . . . .	8

3.2.2	<i>Kakadu Software</i> . . . . .	8
3.3	Materiais Utilizados . . . . .	9
3.3.1	<i>Hardware</i> . . . . .	9
3.3.2	Hologramas . . . . .	10
3.4	Conclusões . . . . .	10
<b>4</b>	<b>Etapas de Desenvolvimento e Implementação</b>	<b>13</b>
4.1	Introdução . . . . .	13
4.2	Reconstrução dos Hologramas . . . . .	14
4.3	Compressão dos Hologramas Reconstruídos . . . . .	17
4.3.1	Breve Estudo do <i>Kakadu Software</i> . . . . .	17
4.3.2	Implementação e Execução do <i>Script</i> de Compressão . . . . .	17
4.3.3	Automatização dos <i>Scripts</i> . . . . .	18
4.4	Determinação das Métricas de Compressão . . . . .	19
4.4.1	Criação de gráficos . . . . .	20
4.5	Conclusões . . . . .	20
<b>5</b>	<b>Testes e Resultados</b>	<b>21</b>
5.1	Introdução . . . . .	21
5.2	Testes . . . . .	21
5.3	Resultados . . . . .	22
5.4	Discussão . . . . .	22
5.5	Conclusões . . . . .	26
<b>6</b>	<b>Conclusões e Trabalho Futuro</b>	<b>29</b>
6.1	Conclusões Principais . . . . .	29
6.2	Trabalho Futuro . . . . .	29
<b>A</b>	<b>Documentação de Funções</b>	<b>31</b>
A.1	Função <code>load_hologram</code> . . . . .	31
A.2	Função <code>propagate_asm</code> . . . . .	32
A.3	Função <code>reconst_hologram</code> . . . . .	33
A.4	Função <code>reconst_16views</code> . . . . .	34
A.5	Função <code>compress_views</code> . . . . .	35
A.6	Função <code>cod_jpeg2000</code> . . . . .	36
A.7	Função <code>dec_jpeg2000</code> . . . . .	37
A.8	Função <code>psnr</code> . . . . .	38

## ***Lista de Figuras***

3.1	Holograma Dices4k (imagem original). . . . .	11
3.2	Holograma DiffuseCar4k (imagem original). . . . .	11
3.3	Holograma Piano4k (imagem original). . . . .	12
4.1	Diagrama do processo efetuado sobre um holograma para obten- ção dos resultados. . . . .	15
4.2	Diagrama de dependências da função reconst_16views. . . . .	16
4.3	Diagrama de dependências da função compress_views. . . . .	18
5.1	Variação da média da métrica <i>Peak signal-to-noise ratio</i> (PSNR) re- lativo ao holograma dices4k. <i>Valor maior significa melhor quali- dade.</i> . . . . .	24
5.2	Variação da média da métrica PSNR relativo ao holograma diffuseCar4k. <i>Valor maior significa melhor qualidade.</i> . . . . .	26
5.3	Variação da média da métrica PSNR relativo ao holograma piano4k. <i>Valor maior significa melhor qualidade.</i> . . . . .	28





## ***Lista de Tabelas***

3.1	Especificações dos hologramas. . . . .	10
5.1	Parâmetros variáveis utilizados na reconstrução dos hologramas. .	22
5.2	Resultados da métrica PSNR para o holograma dices4k. . . . .	23
5.3	Resultados da métrica PSNR para o holograma diffuseCar4k. . .	25
5.4	Resultados da métrica PSNR para o holograma piano4k. . . . .	27
A.1	Documentação da função load_hologram. . . . .	31
A.2	Documentação da função propagate_asm. . . . .	32
A.3	Documentação da função reconst_hologram. . . . .	33
A.4	Documentação da função reconst_16views. . . . .	34
A.5	Documentação da função compress_views. . . . .	35
A.6	Documentação da função cod_jpeg2000. . . . .	36
A.7	Documentação da função dec_jpeg2000. . . . .	37
A.8	Documentação da função psnr. . . . .	38



# ***Acrónimos***

**ASM** *Angular Spectrum Method*

**dB** *Decibel*

**JPEG** *Joint Photographic Experts Group*

**JSON** *JavaScript Object Notation*

**kdu** *Kakadu Software*

**PSNR** *Peak signal-to-noise ratio*

**RGB** *Red, Green & Blue*

**SDK** *Software Development Kit*



## Capítulo

# 1

## Introdução

### 1.1 Enquadramento

A história da captura, armazenamento e visualização de imagens é extremamente rica e milenar. Marcos importantes destacam-se, sendo do particular interesse no Século XXI os grandes passos dados na imagem digital.

Contudo, a vasta maioria da fotografia tem-se centrado na captura de imagens estáticas em duas dimensões. O interesse na captura e representação de objetos e momentos em três dimensões tem ganho um interesse crescente nas últimas décadas.

A área dedicada ao estudo deste modelo, a **holografia**, carece de vários marcos que já fazem parte do quotidiano da fotografia clássica, nomeadamente padrões *standardizados* para a codificação e compressão de **hologramas** em formato digital.

### 1.2 Motivação

Dada a referida ausência de *standards* no armazenamento e representação da informação, reconstrução e codificação de um holograma, é do interesse da comunidade do JPEG Pleno estudar os codificadores existentes para melhor perceber qual a sua adaptabilidade aos hologramas e quais as modificações necessárias para resolver a falta de padrões nos pontos mencionados.

## 1.3 Objetivos

Tendo em mente a motivação apresentada na secção 1.2, o presente projeto tem por objetivo principal investigar o desempenho do codec JPEG2000 na codificação de hologramas a cores em multivistas.

Por seu turno, os objetivos secundários — os quais refletem as diferentes fases da investigação — são os seguintes:

1. Implementar um reconstrutor para hologramas com cor;
2. Reconstruir hologramas em vários pontos de vista;
3. Comprimir hologramas reconstruídos com recurso ao codificador JPEG2000;
4. Avaliar a qualidade das imagens comprimidas face à reconstrução original.

Os objetivos supra-mencionados refletem o objetivo geral de estudar holografia e, assim, expandir o conhecimento na área das tecnologias multimédia.

## 1.4 Organização do Documento

TODO

## **Capítulo**

# 2

## ***Estado da Arte***

### **2.1 Introdução**

TODO

### **2.2 Perspetiva Histórica**

TODO

### **2.3 Conceitos Base**

TODO

#### **2.3.1 Holografia**

Quando um objeto é iluminado, a luz é dispersa pela superfície desse objeto, criando uma onda. Esta onda contém toda a informação sobre a luz: a amplitude define o brilho e a fase representa a forma do objeto. Enquanto as fotografias clássicas gravam apenas a intensidade da luz, um holograma preserva a fase do objeto através das características de interferência e difração da luz, guardando assim toda a informação necessária à reconstrução 3D do objeto original.

##### **2.3.1.1 Princípios de Holografia**

O princípio de holografia foi descoberto em 1948 pelo físico Dennis Gabor enquanto investigava microscopia de elétrons.

Ao contrário da fotografia convencional, que permite a captura da intensidade da luz, holografia permite guardar a amplitude e a fase da onda de luz dispersa por um objeto. Com a iluminação correta, o holograma produz a onda de luz original, permitindo ao utilizador observar o objeto tal como se estivesse fisicamente presente.

#### **2.3.1.2 Representação de Dados Holográficos**

Os dados holográficos podem ser representados de várias formas. Embora sejam todas equivalentes no sentido em que representam o mesmo objeto, algumas tornam a compressão mais eficiente.

No âmbito deste projeto, apenas é relevante a representação no campo de onda complexo.

- Dados reais e imaginários — Utiliza um sistema de coordenadas cartesianas para representar amplitudes complexas;
- Dados da amplitude e fase — Os valores complexos são expressos num sistema de coordenadas polares.

Os hologramas utilizados neste projeto são representados pelo formato de amplitude-fase.

#### **2.3.1.3 Reconstrução de Holograma**

TODO

### **2.3.2 Compressão**

TODO

### **2.3.3 JPEG2000**

TODO

### **2.3.4 Cor**

TODO

#### **2.3.4.1 *Red, Green & Blue* (RGB)**

TODO



### 2.3.4.2 YCbCr

TODO

## 2.4 Estado da Arte

Primeira proposta para codificação digital de hologramas data 1991, Sato et al. captura franjas holográficas usando uma câmara que foram por sua vez modulados em sinal TV e transmitidos para um recetor [1]. (captured the holographic fringes using a camera, which was then modulated into a TV signal and transmitted to the receiver.);

Em 1993, Yoshikawa notou que não era prática a aplicação da compressão de imagem 2d diretamente no holograma. Propôs a compressão do holograma em segmentos que correspondem a diferentes perspectivas de reconstrução. Segmentos foram comprimidos com MPEG-1 e MPEG-2 [2,3]. (ver resultados)

Em 2002, Naughton et al. estudou a compressibilidade da holografia digital de mudança de fase usando vários algoritmos de compressão sem perdas [4]. Concluíram que são esperadas melhores taxas de compressão quando o holograma digital é codificado em componentes reais e imaginárias independentemente.

Em [4] foram também estudadas outras técnicas de compressão com perdas tais como subamostragem e quantificação, sendo a última muito eficaz. A eficácia da quantização tanto na simulação numérica como na ótica foi confirmada por Mills e Yamaguchi [5].

A quantização no domínio da reconstrução (não sei o que isto quer dizer) de hologramas de mudança de fase de foram analisados por Darakis and Soraghan [6].

Naughton et al. em 2003 e Darakis et al. em 2006 demonstraram que a aplicação direta de wavelets standard em hologramas não é muito eficiente, visto que as wavelets standard são tipicamente usadas no processamento de sinais com poucas variações (smooth signals). Propuseram a utilização de uma outra família de wavelets Fresnelets. Fresnelets foram também aplicadas em 2003 por Livelin et al. [8]

Em 2006, Seo et al. propôs comprimir segmentos do holograma usando multi-vistas e temporal prediction dentro de MPEG-2 modificado.

Em 2010 Darkis et al. Determinaram a taxa de compressão mais elevada que pode ser obtida em hologramas mantendo uma qualidade de reconstrução visualmente sem perdas. Nos seus ensaios foram usados MPEG-4 e Dirac. Na informação amplitude-fase foi aplicado um método multiple description

coding utilizando máximo à posterior. Mostrou-se um mecanismo poderoso para mitigar erros no canais em hologramas digitais.

Em 2013 Blinder investigou a decomposição alternativa em hologramas off-axis. Em 2014 Still, Xing e Dufaux estudaram codificação sem perdas baseada em quantização vetorial.

Recentemente Peixeiro et al. [9] realizou um benchmark dos codificadores standard de imagens aplicados em hologramas digitais, em conjunto com os formatos de representação principais. Foram comparados os seguintes codificadores de imagem padrão JPEG; JPEG 2000; H.264/AVC intra; HEVC intra. Os autores concluíram que os melhores formatos de representação são phase-shifted e real-imaginário

Em 2016, Dufaux review o estado da arte da compressão de hologramas digitais

## 2.5 Conclusões

TODO

## Capítulo

# 3

## ***Tecnologias e Ferramentas Utilizadas***

### **3.1 Introdução**

O projeto apresentado só foi possível ser concluído devido à existência de tecnologias e ferramentas, assim como de materiais, os quais se revelaram essenciais.

Em particular, este Capítulo aborda:

- O *software* de reconstrução de hologramas;
- A escolha da linguagem de programação para o projeto;
- O *Software Development Kit* (SDK) de codificação de imagens no formato JPEG2000;
- O *hardware* utilizado;
- Os hologramas testados.

### **3.2 Tecnologias e Ferramentas**

O desenvolvimento do projeto envolveu o trabalho conjunto de diversas ferramentas, nomeadamente Python 3, *Kakadu Software* e *software* escrito no âmbito do projeto JPEG Pleno.

### 3.2.1 Software de Reconstrução de Hologramas e sua Transcrição para Python

Do 80º Encontro do Grupo JPEG, realizado em Berlim entre 7 e 13 de julho de 2018, resultou um software desenvolvido por Antonin Gilles e Patrick Gioia, do *Institute of Research & Technology b<>com*. O respetivo código, fornecido pela professora orientadora, encontra-se implementado em MATLAB.

Dada a ausência de uma licença do MATLAB para utilizar este *software* desenvolvido no âmbito do JPEG Pleno, foi necessário transcrever o código para uma nova linguagem de programação. Neste sentido, optou-se pelo Python 3.

O recurso a Python apresenta uma miríade de vantagens, entre elas:

- Linguagem *open source*;
- Utilização e distribuição gratuita da linguagem;
- Maior eficiência face ao MATLAB;
- Utilização comum no contexto de processamento multimédia e respetivos projetos;
- Vasto leque de bibliotecas, facilitando a implementação de *software* específico;
- Abundância de documentação;
- Forte comunidade *online* de suporte.

A escolha do Python foi, portanto, natural no âmbito do presente projeto.

Durante a fase de transcrição decorreu uma atualização do Python, tendo sido a última versão do código executada e testada na versão 3.8.2 em três distribuições GNU/Linux de 64 bits: Ubuntu 18.04 LTS, Fedora 31 e Linux Mint 20.

### 3.2.2 Kakadu Software

Uma vez que o JPEG2000 é o formato alvo deste projeto, e tendo em conta a dificuldade encontrada em projetos anteriores no contexto da holografia em utilizar a ferramenta *FFmpeg*, foi recomendado pela professora orientadora a utilização do *Kakadu Software*.

Este é um SDK para codificação e decodificação de imagens com recurso ao formato JPEG2000, segundo o *standard* pela *Joint Photographic Experts Group* (JPEG).

Os comandos deste SDK de relevo para o projeto são os seguintes:

1. `kdu_compress`: codifica uma imagem para o formato JPEG2000.

Sintaxe:

---

```
kdu_compress -i input_file -o output_file -rate n  
↳ Cycc=<yes|no> -precise -quiet
```

---

onde:

- `-i input_file`: imagem de *input*;
- `-o output_file`: ficheiro de *output*;
- `-rate n`: número de *bits* por amostra (*n* pode ser um número flutuante);
- `Cycc=<yes|no>`: yes caso seja usada a codificação com transformação de cor de RGB para YCbCr, no em caso contrário;
- `-precise`: força o uso de representações de 32 *bits*;
- `-quiet`: suprime o *output* do programa.

2. `kdu_expand`: descodifica uma imagem no formato JPEG2000.

Sintaxe:

---

```
kdu_expand -i input_file -o output_file -rate n -quiet
```

---

onde:

- `-i input_file`: imagem de *input*;
- `-o output_file`: ficheiro de *output*;
- `-rate n`: número de *bits* por amostra (*n* pode ser um número flutuante);
- `-quiet`: suprime o *output* do programa.

## 3.3 Materiais Utilizados

### 3.3.1 *Hardware*

De notar que esta implementação do *software* de reconstrução de hologramas requer computadores com especificações mais generosas.

Para este projeto, dois computadores em particular executaram as várias iterações de desenvolvimento do *software* transcrito em Python:

1. *Desktop*: processador Ryzen™ 7 2700X 3.7–4.3GHz, placa gráfica NVidia® Quadro K5000 (4GB), memória RAM de 32GB e *swap* de 96GB, armazenamento SSD de 1TB;
2. Portátil: Intel® Core™ i5-10210U 1.6–4.2GHz, memória RAM de 16GB e *swap* de 8GB, armazenamento SSD de 512GB.

### 3.3.2 Hologramas

Os hologramas reconstruídos com o *software* desenvolvido no âmbito deste projeto são fornecidos pelo *Institute of Research & Technology b<>com*. De entre os disponíveis, foram utilizados os seguintes hologramas com respectivas características resumidas na Tabela 3.1:

1. Dices4k (Figura 3.1);
2. DiffuseCar4k (Figura 3.2);
3. Piano4k (Figura 3.3).

Tabela 3.1: Especificações dos hologramas.

Holograma	size	pitch	r	g	b	z
Dices4k	4096×4096	0.4	640	532	473	0.164 – 0.328
DiffuseCar4k	4096×4096	0.4	640	532	473	0.110 – 0.250
Piano4k	4096×4096	0.4	640	532	473	0.170 – 0.313

size:	Resolução	(em pixels)
pitch:	<i>Pixel pitch</i>	(em $\mu\text{m}$ )
r:	Comprimento de onda vermelho	(em nm)
g:	Comprimento de onda verde	(em nm)
b:	Comprimento de onda azul	(em nm)
z:	Intervalo de localização da cena	(em cm)

## 3.4 Conclusões

Após a seleção das tecnologias e materiais, conforme supra-mencionados, ir-se-á proceder no Capítulo 4 ao delineamento da estratégia de investigação



Figura 3.1: Holograma Dices4k (imagem original).



Figura 3.2: Holograma DiffuseCar4k (imagem original).



Figura 3.3: Holograma Piano4k (imagem original).

do projeto, a qual está intimamente ligada às escolhas apresentadas no presente Capítulo, entre elas a escolha da linguagem Python para transcrição do *software* original de reconstrução de hologramas, o SDK para realizar a codificação no formato JPEG2000 e os hologramas testados.



## ***Etapas de Desenvolvimento e Implementação***

### **4.1 Introdução**

Para o sucesso deste projeto foi essencial delinear uma estratégia de investigação. Esta divide-se em 3 partes: reconstrução dos hologramas, compressão dos hologramas reconstruídos, e determinação das métricas de compressão. A cada uma destas fases atribuíram-se objetivos para a sua profícua execução:

1. *Reconstrução dos hologramas* (2º objetivo secundário do projeto):
  - a) Estudar a holografia;
  - b) Estudar as funções originais em MATLAB do Projecto JPEG Pleno;
  - c) Transcrever estas funções para a linguagem de programação Python;
  - d) Comparar o *output* entre as funções originais em MATLAB e as respetivas transcrições em Python;
  - e) Desenvolver um *script* para reconstruir cada holograma em 16 vistas distintas.
2. *Compressão dos hologramas reconstruídos* (3º objetivo secundário):
  - a) Estudar o uso do *Kakadu Software* (kdu);
  - b) Automatizar a execução dos *scripts* e *softwares* envolvidos;
  - c) Implementar um *script* para compressão (com e sem transformada de cor em diferentes *bitrates*) e descompressão dos hologramas reconstruídos.

3. *Determinação das métricas de compressão* (a fim de alcançar o 4º objetivo secundário, objeto de estudo do Capítulo 5):
  - a) Calcular o débito com a métrica *Peak signal-to-noise ratio* (PSNR) entre os hologramas originais e as imagens comprimidas;
  - b) Determinar o melhor método de armazenamento dos débitos calculados;
  - c) Implementar o *output* dos resultados no método selecionado;
  - d) Gerar gráficos dos resultados.

As Secções subsequentes expõem o trabalho desenvolvido neste sentido e a Figura 4.1 esquematiza o processo levado a cabo para a execução do projeto.

## 4.2 Reconstrução dos Hogramas

O projeto iniciou-se com uma pesquisa exaustiva sobre a ciência da holografia, a qual resultou no Estado da Arte resumido no Capítulo 2.

Simultaneamente, foi efetuado um estudo das funções do *software* desenvolvido no âmbito do projeto JPEG Pleno a fim de se poder fazer a respetiva transcrição para Python. O resultado deste estudo e transcrição encontra-se exposto nas tabelas A.1, A.2 e A.3.

Estas funções foram compiladas num único *script* Python. A fim de cumprir o 2º objetivo secundário, uma nova função, *reconst\_16views*, foi implementada (Tabela A.4). Esta função está, portanto, encarregue de abrir um holograma, as respetivas especificações e reconstruí-lo em exatamente 16 vistas distintas. De notar que as especificações são fornecidas por um ficheiro *JavaScript Object Notation* (JSON) cujo conteúdo inclui:

- *holo\_size*: Resolução do holograma (em pixels);
- *pitch*: Distância entre pixels (em metros);
- *wavelengths*: Lista com os comprimentos de onda respetivos aos canais RGB (em metros);
- *z*: Distância de reconstrução (em metros);
- *pupil\_size*: Lista que contém o tamanho da janela de reconstrução (em pixels).

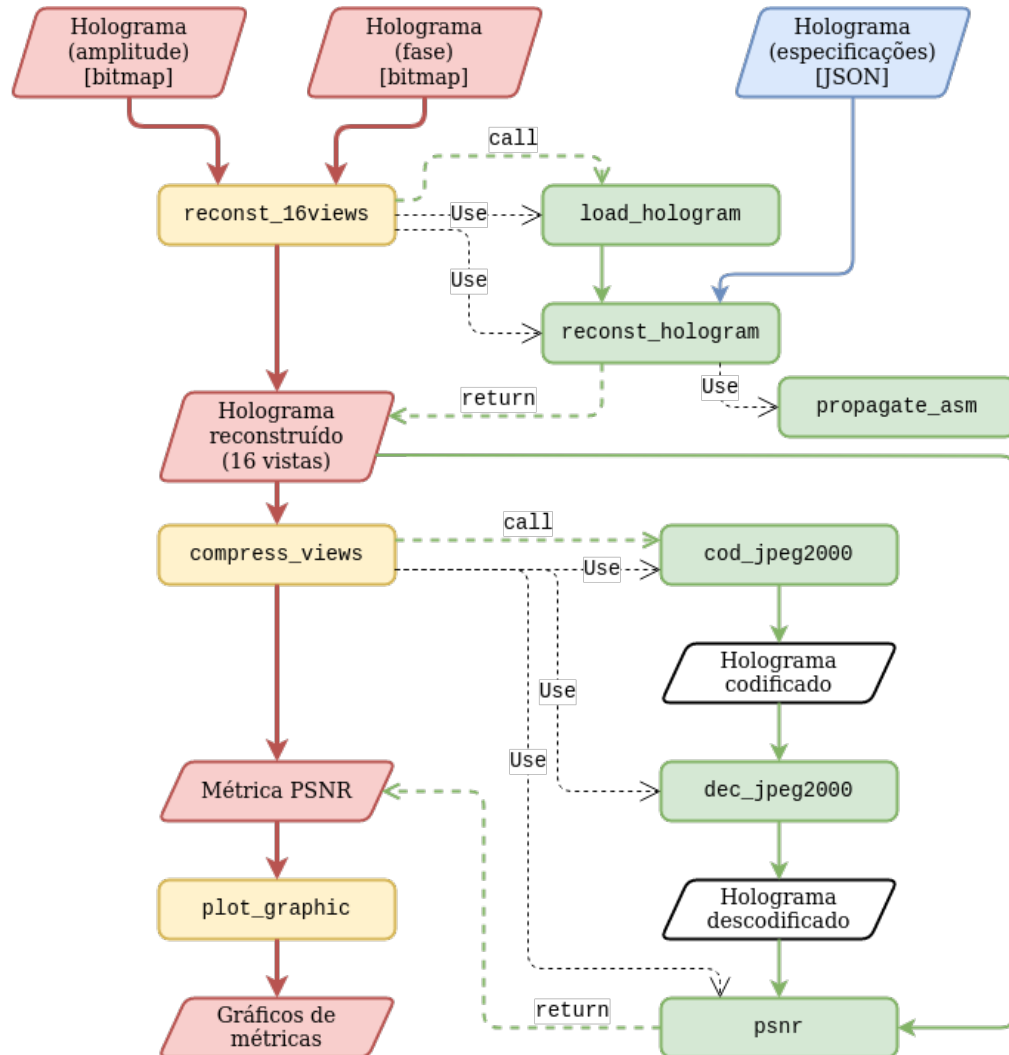


Figura 4.1: Diagrama do processo efetuado sobre um holograma para obtenção dos resultados.

O caminho vermelho/amarelo representa o trajeto principal pelo qual o holograma sofre processamento. Os caminhos verdes detalham os processos intermédios correspondentes a um determinado processo do trajeto principal (a entrada é dada por “call” e o retorno é dado por “return”). As dependências entre as principais funções são indicadas por “Use”. A azul estão os ficheiros externos auxiliares.

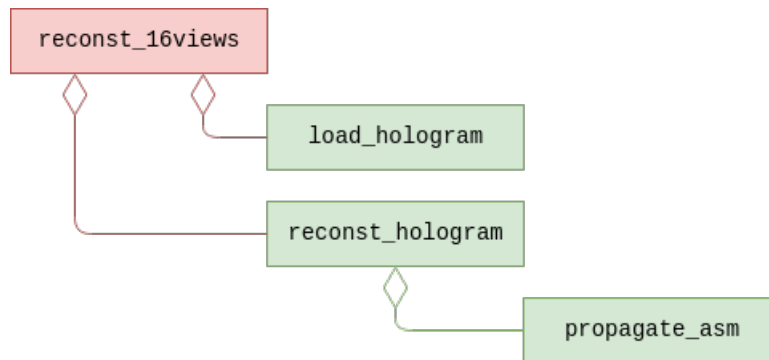


Figura 4.2: Diagrama de dependências da função `reconst_16views`.

Todavia, havendo uma transcrição de funções entre linguagens de programação, exigiu-se uma forma de comprovar que as funções transcritas em Python são equivalentes às originais em MATLAB. Para este fim, delineou-se a seguinte estratégia:

1. Etapa de implementação:
  - a) Transcrever as funções para Python (forme apresentado).
2. Etapa de reconstrução:
  - a) Reconstruir os hologramas numa vista com as funções originais em MATLAB;
  - b) Reconstruir os hologramas numa vista com as funções transcritas em Python.
3. Etapa de comparação e resultados:
  - a) Comparar os hologramas produzidos pelos *scripts* das duas linguagens através de uma análise visual a olho nu;
  - b) Comparar os hologramas produzidos pelos *scripts* das duas linguagens através da métrica PSNR.

Com o *script* devidamente implementado e testado, prosseguiu-se com a seguinte fase do projeto relativo à compressão dos hologramas agora reconstruídos.

## 4.3 Compressão dos Hogramas Reconstruídos

Após a reconstrução dos hogramas em multivista, segue-se o teste ao formato JPEG2000 através de compressão e descompressão de forma a comparar com a reconstrução do holograma original (cuja métrica será abordada na Secção 4.4).

Conforme a estratégia delineada na Secção 4.1, esta fase do projeto envolveu o estudo da ferramenta kdu, a implementação dos *scripts* e o cálculo das métricas.

### 4.3.1 Breve Estudo do *Kakadu Software*

A fase de estudo do kdu resultou no conhecimento exposto na Secção 3.2.2, onde se encontram descritos os dois comandos utilizados no âmbito do projeto.

### 4.3.2 Implementação e Execução do *Script* de Compressão

De forma a se poder comparar o holograma original com o holograma comprimido no formato JPEG2000, revela-se necessário realizar uma sequência de compressão e descompressão do primeiro. Tal irá dar origem a um holograma reconstruído após compressão, o qual poderá ser comparado com o original.

Conforme visto anteriormente, a ferramenta kdu é a ferramenta eleita para este processo. Contudo, não é prático executar manualmente a sequência de comandos na *shell*. Por conseguinte, um *script* escrito em Python foi implementado. Este inclui a função `compress_views` (Tabela A.5 e Figura 4.3), a qual executa a seguinte sequência de operações:

1. *Compressão do holograma original, previamente reconstruído.*  
É feita uma chamada ao sistema do comando `kdu_compress`.  
Esta encontra-se encapsulada na função auxiliar `cod_jpeg2000` (Tabela A.6).
2. *Descompressão do holograma agora comprimido.*  
É feita uma chamada ao sistema do comando `kdu_expand`.  
Esta encontra-se encapsulada na função auxiliar `dec_jpeg2000` (Tabela A.7).
3. *Cálculo do débito através de uma métrica de compressão.*  
O holograma original e o holograma agora descomprimido são compa-

rados com recurso à métrica PSNR (Secção 4.4).

Esta operação é realizada pela função auxiliar `psnr` (Tabela A.8).

De notar os *bitrates* testados: 0.1, 0.3, 0.6, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5 e 5.0. Há ainda a referir a realização dos testes com e sem transformada de cor.

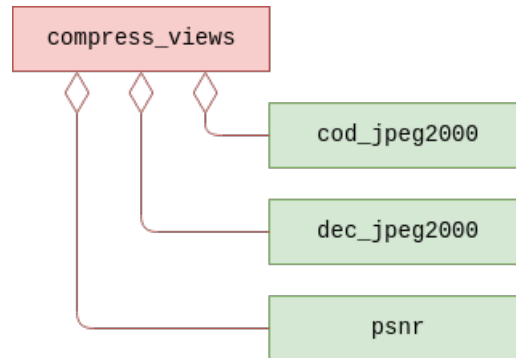


Figura 4.3: Diagrama de dependências da função `compress_views`.

Por fim, de forma a facilitar a extração dos resultados pretendidos segundo o objetivo primário do projeto (Secção 1.3), é gerado um ficheiro JSON por cada holograma. Este tem a seguinte estrutura:

```

<nome_holograma>.json
├── no_ycbcr
│   ├── <rate> ..... Número de bits por amostra
│   │   ├── <vista>.ppm: valor ..... Posição da janela de reconstrução
│   │   ├── avg: valor ..... PSNR médio das vistas para este rate
│   │   └── std: valor ..... Desvio padrão
│   └── ycbcr
│       ├── <rate>
│       │   ├── <vista>.ppm: valor
│       │   ├── avg: valor
│       │   └── std: valor
  
```

O ficheiro JSON supra-mencionado será portanto utilizado no processo descrito na Secção 4.4.

### 4.3.3 Automatização dos *Scripts*

Para a otimização do processo de investigação neste projeto, foram criados dois *scripts* adicionais, os quais trabalham em conjunto a fim de automatizar

todo este processo. São eles os seguintes:

1. *Makefile*: Introduz as 8 opções infra-apresentadas, as quais invocam o *script* `main.py`:
  - a) `reconst`: reconstrói o holograma fornecido por argumento em 16 vistas (Capítulo 4.2 e Tabela A.4);
  - b) `compress`: executa o *script* apresentado na Secção 4.3.2;
  - c) `plot`: calcula os gráficos com os resultados de comparação da compressão com e sem transformada de cor (Secção 4.4);
  - d) `all`: executa os 3 itens anteriores;
  - e) `test`: opção utilizada na fase de *debugging*;
  - f) `install`: instala pacotes no sistema operativo essenciais à execução dos *scripts*;
  - g) `clean`: remove os diretórios relativos a: hologramas reconstruídos, *output* do kdu, ficheiros JSON dos resultados do PSNR, e ficheiros de *cache* do Python;
  - h) `clean-compressed`: remove os diretórios relativos a: *output* do kdu, ficheiros JSON dos resultados do PSNR, e ficheiros de *cache* do Python.
2. `main.py`: *script* Python que executa as primeiras 5 opções supra-mencionadas conforme o argumento passado pelo *Makefile*. Faz uso das funções `reconst_16views` (Tabela A.4) e `compress_views` (Tabela A.5).

## 4.4 Determinação das Métricas de Compressão

Na sequência do processo de compressão e descompressão no formato JPEG2000, e conforme previamente introduzido na Secção 4.3.2, a determinação das métricas de compressão constituem a última fase da investigação a fim destes resultados poderem ser escrutinados no Capítulo 5.

Variadas métricas estão disponíveis para se poder comparar amostras e, no caso do presente projeto, avaliar a qualidade do holograma após compressão no formato JPEG2000. De entre as métricas existentes, foi eleita a *Peak signal-to-noise ratio* (PSNR) por indicação da professora orientadora, a qual está envolvida no Projeto JPEG Pleno.

Conforme introduzido na Secção 4.3.2, foi utilizada uma função auxiliar `psnr` (Tabela A.8), invocada pela função principal `compress_views` (Tabela

A.5). Este método de implementação foi escolhido para fins de eficiência de armazenamento.

Uma vez que não é mandatório armazenar os hologramas rescontruídos após compressão, sendo apenas do nosso interesse as métricas finais a si referentes, a função `dec_jpeg2000` exporta o holograma descomprimido num ficheiro temporário `temp.ppm`. Este é utilizado de seguida pela função `psnr` para cálculo do PSNR (sendo os resultados armazenados num ficheiro JSON), sendo de seguida eliminado.

#### 4.4.1 Criação de gráficos

O ficheiro JSON gerado (descrito na Secção 4.3.2) é posteriormente carregado num novo *script* Python, especificamente implementado para gerar gráficos em si baseados para mais fácil visualização, análise e discussão (Capítulo 5). Este *script* recorre à biblioteca `matplotlib.pyplot`.

A geração de um gráfico por holograma por parte deste *script* marca o fim do processo de investigação relativo à manipulação dos hologramas e extração de métricas.

### 4.5 Conclusões

Após a conclusão das 3 fases de investigação segundo a estratégia delineada inicialmente e ilustrada pela Figura 4.1, resta a exposição e análise dos resultados obtidos no Capítulo 5.

A estratégia desenhada revelou-se eficaz e permitiu a obtenção de *scripts* facilmente adaptáveis para trabalhos futuros no âmbito do Projeto JPEG Pleno, assim como os próprios resultados para análise por parte de outros interessados. De igual forma, a escolha de *standards* gratuitos e *open-source* (justificada na Secção 3.2) revelou-se acertada pelos mesmos motivos.



## Capítulo

# 5

## ***Testes e Resultados***

### **5.1 Introdução**

A implementação das funções e dos *scripts* do projeto e respetiva execução com os hologramas selecionados para este culmina no cumprimento do 4º e último objetivo secundário do projeto, o qual permitirá retirar conclusões para o objetivo primário identificado na Secção 1.3.

Para tal, ir-se-ão resumir os testes efetuados (Secção 5.2) e os resultados de si obtidos (Secção 5.3), assim como discuti-los na perspetiva do objetivo do projeto (Secção 5.4).

### **5.2 Testes**

A sequência de fases de investigação apresentada no Capítulo 4 e resumida na *pipeline* ilustrada pela Figura 4.1 foi exaustivamente testada com os 3 hologramas enumerados na Secção 3.3.2.

Os parâmetros apresentados na Tabela 3.1 são fixos (a fim de garantir a reconstrução do holograma), sendo a distância de reconstrução (parâmetro  $z$ ) em particular um intervalo. Qualquer valor dentro deste intervalo pode ser tomado. A par deste parâmetro, também o tamanho da vista (valor arbitrário menor do que a resolução do holograma) é necessário para a fase de testes, conforme as funções transcritas e implementadas ao longo do Capítulo 4. Os valores selecionados para estes dois parâmetros são apresentados na Tabela 5.1.

Os ficheiros JSON gerados pelos *scripts* previamente implementados armazenam, conforme descrito na Secção 4.3.2, os resultados doravante explorados na Secção 5.3.

Tabela 5.1: Parâmetros variáveis utilizados na reconstrução dos hologramas.

Holograma	z	pupil_size
Dices4k	0.0020	2048×2048
DiffuseCar4k	0.0012	2048×2048
Piano4k	0.0019	2048×2048

z : Distância de reconstrução (em metros)

pupil\_size : Tamanho da vista (em pixels)

### 5.3 Resultados

Os resultados dos testes efetuados são os valores da métrica PSNR, em particular um por holograma, por *bitrate* (débito), por vista, e com e sem transformada de cor. No caso deste projeto, tal traduz-se em  $3 \times 12 \times 16 \times 2 = 1152$  valores. Ora, esta quantidade de valores é impraticável de ser exposta e devidamente explorada.

Neste sentido, optou-se por se calcular a média dos valores de PSNR e o respetivo desvio padrão por cada *bitrate*, o que reduz para 72 valores, tornando-se alcançável a devida discussão dos resultados do projeto.

Destacam-se as tabelas e figuras que expõem os resultados da investigação. Porquanto as tabelas dispõem os valores das médias da métrica PSNR para as 16 vistas reconstruídas por cada *bitrate* e por opção de com ou sem de transformada de cor, as figuras apresentam-nos graficamente. Em particular:

- Holograma dices4k: Tabela 5.2 e Figura 5.1;
- Holograma diffuseCar4k: Tabela 5.3 e Figura 5.2;
- Holograma piano4k: Tabela 5.4 e Figura 5.3.

### 5.4 Discussão

Tendo em conta que quanto maior o valor do PSNR melhor será a qualidade da imagem após a compressão, uma análise visual aos gráficos das figuras 5.1, 5.2 e 5.3 permite facilmente observar a diminuição da qualidade de todos os três hologramas testados ao ser utilizada uma transformada de cor de RGB para YCbCr.

Tabela 5.2: Resultados da métrica PSNR para o holograma *dices4k*.

<i>Bitrate</i>	Sem transformada de cor		Com transformada de cor	
	avg	std	avg	std
0.1	25.432	3.852	24.768	3.684
0.3	28.513	4.772	27.071	4.662
0.6	31.221	5.427	29.311	5.160
1.0	33.723	5.919	31.176	5.591
1.5	36.210	6.160	33.000	5.870
2.0	38.212	6.270	34.743	6.369
2.5	39.942	6.337	35.930	6.410
3.0	41.418	6.116	37.445	6.658
3.5	42.824	6.114	38.966	6.676
4.0	44.097	5.957	39.966	6.521
4.5	45.310	5.704	40.962	6.645
5.0	46.209	5.411	42.084	6.780

avg : Média do PSNR (em dB)

std : Desvio padrão (em dB)

Há ainda a referir, pela observação das mesmas figuras, que a qualidade dos hologramas comprimidos aumenta com a subida do *bitrate*.

Tal é conforme o esperado uma vez que o aumento do débito implica a possibilidade de se armazenar uma maior quantidade de informação acerca da imagem original.

Por seu turno, de uma análise mais detalhada às Tabelas 5.2, 5.3 e 5.4 constata-se a ocorrência de um fenómeno igualmente antecipado nos valores do desvio padrão: este tende a diminuir para *bitrates* mais elevados.

Do exposto, 3 questões colocam-se e que são meritórias de discussão no âmbito da investigação:

1. Qual o motivo para a transformada de cor levar à significativa redução de qualidade após compressão?

A transformada de cor de RGB para YUV 4:2:0 implica perda de informação nos canais G (*Green*) e B (*Blue*) a fim de tirar partido das características da visão humana. Por si só, esta transformada de cor torna a perda de informação impercetível ao olho humano, sendo esta de uma aparente qualidade equivalente.

Todavia, a compressão no formato JPEG2000 implica uma nova perda

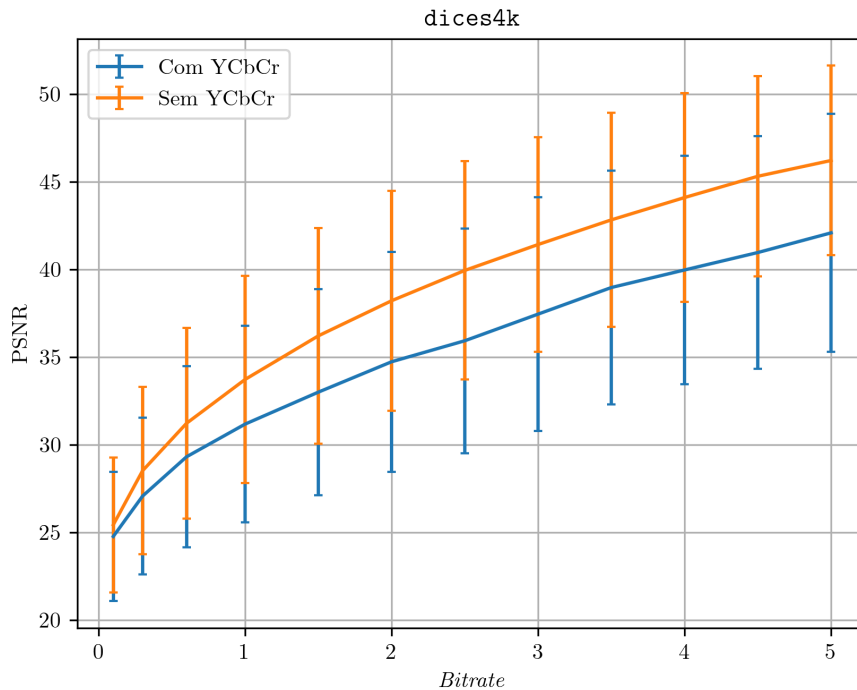


Figura 5.1: Variação da média da métrica PSNR relativo ao holograma dices4k. Valor maior significa melhor qualidade.

de informação uma vez que este é um formato de compressão com perdas.

Teoriza-se que a aplicação consecutiva de dois processos que implicam perdas de informação levem a uma significativa redução da qualidade final do holograma comprimido, resultando em valores de PSNR claramente inferiores.

## 2. Por que razão o desvio padrão se revela menor em bitrates maiores?

O aumento da qualidade dos hologramas comprimidos com *bitrates* maiores (conforme observado anteriormente) dever-se-á explicar pela maior quantidade de informação armazenada por consequência. Tal leva à existência de menos ruído nas imagens comprimidas, o que implica uma maior aproximação face à imagem original.

A par do aumento do PSNR, é de denotar que o aumento da informação armazenada deverá implicar igualmente uma maior consistência na qualidade dos hologramas comprimidos, independentemente da vista tomada.

Tabela 5.3: Resultados da métrica PSNR para o holograma diffuseCar4k.

<i>Bitrate</i>	Sem transformada de cor		Com transformada de cor	
	avg	std	avg	std
0.1	29.441	4.763	27.438	3.795
0.3	33.938	6.170	29.762	5.089
0.6	38.299	6.820	32.336	6.019
1.0	42.364	6.864	35.398	7.046
1.5	46.053	6.540	38.747	7.682
2.0	48.611	5.938	41.496	7.635
2.5	50.706	5.339	44.037	7.577
3.0	52.038	4.599	45.643	6.937
3.5	53.174	3.899	47.263	6.262
4.0	54.096	3.318	48.878	5.400
4.5	54.819	2.812	50.110	4.567
5.0	55.280	2.310	50.851	3.706

avg : Média do PSNR (em dB)

std : Desvio padrão (em dB)

Ora, estatisticamente, uma maior consistência dos dados traduz-se na diminuição do desvio padrão.

3. *Por que razão o desvio padrão se revela menor em bitrates menores?*

Paralelamente ao ponto anterior, a diminuição do débito introduz um considerável aumento do ruído nos hologramas comprimidos, o que leva a um decaimento rápido do valor de PSNR à medida que se diminui o *bitrate* utilizado.

É, pois, expectável que não existam grandes variações no PSNR independentemente da vista tomada na reconstrução do holograma. Por outras palavras, é de esperar consistência na falta de qualidade das imagens (*e.g.* é altamente improvável obter um PSNR de 50 dB quando se obtém consistentemente valores a rondar os 20 dB para um *bitrate* de 0.1).

Tal deverá justificar o fenómeno da notória diminuição do desvio padrão para *bitrates* mais baixos.

De referir que a reconstrução em multivista permite verificar que a qualidade de compressão pode igualmente depender da vista a partir da qual é reconstruída.

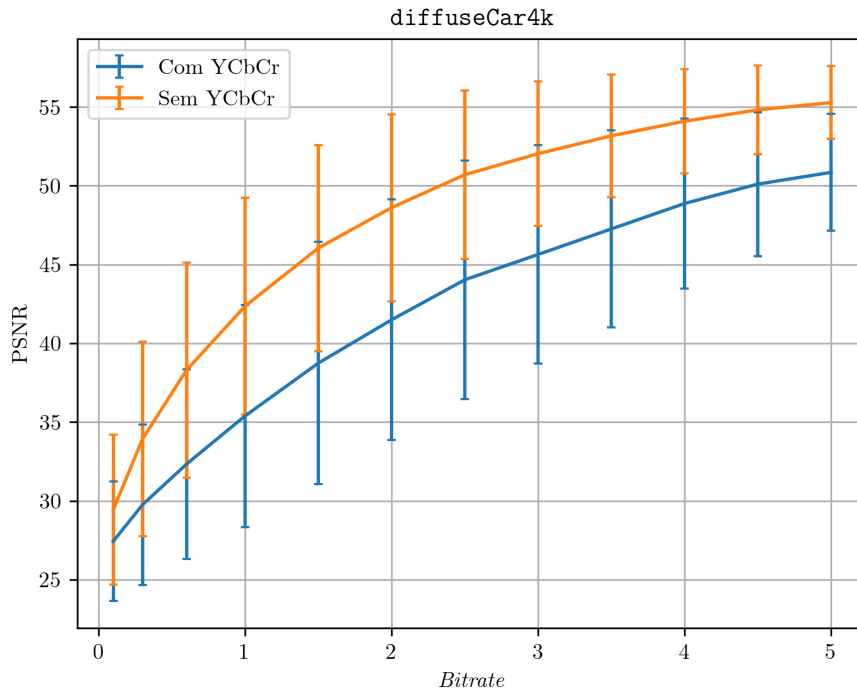


Figura 5.2: Variação da média da métrica PSNR relativo ao holograma diffuseCar4k. *Valor maior significa melhor qualidade.*

## 5.5 Conclusões

A análise de três grandes variáveis foi levada em consideração durante a investigação e que neste Capítulo foram abordadas:

1. Reconstrução em 16 vistas distintas;
2. Uso de uma transformada de cor de RGB para YCbCr;
3. Uso de 12 *bitrates* distintos.

Tendo sido assim cumpridos os objetivos secundários propostos para o projeto e após a discussão dos resultados obtidos sob a perspectiva de três questões, a retirada de conclusões para responder ao objetivo primário torna-se, portanto, exequível.

Tabela 5.4: Resultados da métrica PSNR para o holograma piano4k.

<i>Bitrate</i>	Sem transformada de cor		Com transformada de cor	
	avg	std	avg	std
0.1	27.352	3.958	26.156	3.579
0.3	31.336	4.660	29.166	4.538
0.6	34.930	4.847	31.711	4.558
1.0	38.291	4.827	33.782	4.631
1.5	41.393	4.697	36.231	4.775
2.0	43.914	4.291	38.247	4.858
2.5	45.947	4.019	40.340	4.911
3.0	47.654	3.632	41.937	4.582
3.5	48.987	3.236	43.430	4.722
4.0	50.300	2.873	45.279	4.550
4.5	51.290	2.715	46.741	4.316
5.0	52.210	2.471	47.638	3.642

avg : Média do PSNR (em dB)

std : Desvio padrão (em dB)

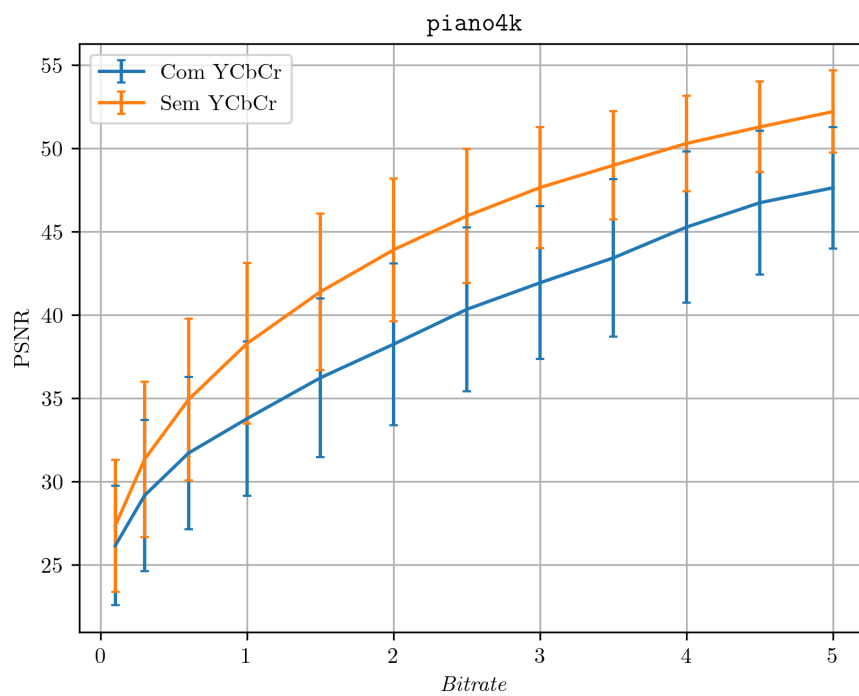


Figura 5.3: Variação da média da métrica PSNR relativo ao holograma piano4k. Valor maior significa melhor qualidade.



## ***Conclusões e Trabalho Futuro***

### **6.1 Conclusões Principais**

TODO

Após os resultados expostos na Secção 5.3 e a sua respetiva discussão na Secção 5.4, podem-se enumerar **X** conclusões:

1. O recurso à transformada de cor não é desejável;
2. O uso de maiores *bitrates* produz hologramas comprimidos de melhor qualidade;
3. O uso de maiores *bitrates* produz hologramas comprimidos de qualidade mais consistentes para diferentes vistas.

### **6.2 Trabalho Futuro**

TODO

A par do concluído na Secção 6.1, propõe-se o estudo alargado a mais hologramas para se poder determinar se existe um *bitrate* mínimo comum aceitável ou se este deve sempre depender do holograma original.

No mesmo sentido, caso o débito dependa do holograma, é plausível a pertinência de um estudo sobre métodos e métricas para determinação ótima do débito aquando da compressão.



## Apêndice

# A

## Documentação de Funções

### A.1 Função load\_hologram

Tabela A.1: Documentação da função load\_hologram.

<b>Nome da função</b>	load_hologram
<b>Protótipo original em MATLAB</b>	<code>function [hologram] = load_hologram(ampli_path, phase_path)</code>
<b>Protótipo transcrito em Python</b>	<code>def load_hologram(ampli_path, phase_path)</code>
<b>Descrição</b>	Esta função carrega um holograma da base de dados b<>com a partir dos seus ficheiros de amplitude e fase.
<b>Inputs</b>	ampli_path: Diretório do ficheiro da imagem da amplitude (caminho relativo ou absoluto). phase_path: Diretório do ficheiro da imagem da fase (caminho relativo ou absoluto).
<b>Output</b>	Modulação complexa do holograma (3 canais: RGB).
<b>Efeitos colaterais</b>	Não aplicável.
<b>Dependências</b>	Não aplicável.

## A.2 Função `propagate_asm`

Tabela A.2: Documentação da função `propagate_asm`.

<b>Nome da função</b>	<code>propagate_asm</code>
<b>Protótipo original em MATLAB</b>	<code>function [v] = propagate_asm(u, pitch, wavelength, z)</code>
<b>Protótipo transcrito em Python</b>	<code>def propagate_asm(u, pitch, wavelength, z)</code>
<b>Descrição</b>	Esta função simula a propagação no plano complexo <i>u</i> sobre a distância <i>z</i> utilizando o <i>Angular Spectrum Method</i> (ASM).
<b>Inputs</b>	<p><i>u</i>: Campo de onda de luz do plano de <i>input</i> (um canal).</p> <p><i>pitch</i>: Distância entre pixels (em metros).</p> <p><i>wavelength</i>: Comprimento de onda do canal de cor a propagar (em metros).</p> <p><i>z</i>: Distância de propagação ao longo do eixo ótico (em metros).</p>
<b>Output</b>	Campo de onda de luz no plano de destino (um canal).
<b>Efeitos colaterais</b>	Não aplicável.
<b>Dependências</b>	Não aplicável.

## A.3 Função reconst\_hologram

Tabela A.3: Documentação da função reconst\_hologram.

<b>Nome da função</b>	reconst_hologram
<b>Protótipo original em MATLAB</b>	<code>function [recons] = reconsHologram(hologram, pitch, wavelengths, z, pupilPos, pupilSize)</code>
<b>Protótipo transcrito em Python</b>	<code>def reconst_hologram(hologram, pitch, wavelengths, z, pupil_pos, pupil_size)</code>
<b>Descrição</b>	Esta função reconstrói o holograma a uma distância $z$ , utilizando o ASM. Permite o uso de uma janela para obter reconstruções de diferentes pontos de vista.
<b>Inputs</b>	<p>hologram: Holograma de modulação complexa (3 canais: RGB).</p> <p>pitch: Distância entre pixels (em metros).</p> <p>wavelengths: Comprimentos de onda de luz (em metros, 3 canais: RGB).</p> <p>z: Distância de reconstrução (em metros).</p> <p>pupilPos: Posição da janela (em pixels, canto superior direito).  <i>Valor por defeito.</i> [0, 0].</p> <p>pupilSize: Tamanho da janela (em pixels, altura <math>\times</math> largura).  <i>Valor por defeito.</i> None.</p>
<b>Output</b>	Reconstrução numérica do holograma (3 canais: RGB).
<b>Efeitos colaterais</b>	Não aplicável.
<b>Dependências</b>	Função propagate_asm (Tabela A.2).

## A.4 Função `reconst_16views`

Tabela A.4: Documentação da função `reconst_16views`.

<b>Nome da função</b>	<code>reconst_16views</code>
<b>Protótipo em Python</b>	<code>def reconst_16views(hologram)</code>
<b>Descrição</b>	<p>Reconstrói o holograma fornecido por argumento em 16 vistas.</p> <p><i>Processo.</i> Carrega o holograma (dois ficheiros <i>bitmap</i>: amplitude e fase) e o respetivo ficheiro de especificações (formato JSON). Calcula as posições das 16 vistas tendo em conta o tamanho do holograma.</p>
<b>Inputs</b>	<p><code>hologram</code>: Nome do holograma a ser reconstruído.</p> <p><i>Valor por defeito.</i> “dices4k”.</p>
<b>Output</b>	Não aplicável.
<b>Efeitos colaterais</b>	<p>Produz, dentro da pasta <code>./reconst/</code>, uma nova pasta com o nome do holograma. O respetivo conteúdo incluirá as 16 vistas (ficheiros <code>*.ppm</code>) correspondentes às reconstruções produzidas pela função.</p>
<b>Dependências</b>	<p>Função <code>load_hologram</code> (Tabela A.1);</p> <p>Função <code>reconst_hologram</code> (Tabela A.3).</p> <p><i>Ver Figura 4.2.</i></p>

## A.5 Função compress\_views

Tabela A.5: Documentação da função compress\_views.

<b>Nome da função</b>	compress_views
<b>Protótipo em Python</b>	<code>def compress_views(hologram, ycbcr, rate)</code>
<b>Descrição</b>	<p>Comprimir os hologramas e calcular o PSNR.</p> <p><i>Processo.</i> Para cada vista, o holograma é codificado, decodificado e analisado em termos do PSNR. São criados 16 ficheiros *.jp2 por pasta rate_n, correspondentes às 16 vistas reconstruídas, assim como ficheiros JSON (um por holograma) na pasta kduOutput com as métricas de compressão (PSNR).</p>
<b>Inputs</b>	<p>hologram: Nome do holograma a comprimir.  <i>Valor por defeito.</i> “dices4k”.</p> <p>ycbcr: <i>Flag</i> que indica ao kdu se deve ser efetuada uma transformação de cor.  <i>Valor por defeito.</i> False.</p> <p>rate: número de <i>bits</i> por amostra.  <i>Valor por defeito.</i> 1.0.</p>
<b>Output</b>	Não aplicável.
<b>Efeitos colaterais</b>	<p>Armazena as imagens no formato *.jp2 nas pastas rate_n, assim como os ficheiros JSON na pasta kduOutput, segundo a seguinte árvore de diretórios:</p> <pre> ./kduOutput ├── &lt;nome_holograma&gt; │   ├── no_ycbcr │   │   ├── rate_n..... n: número de bits │   │   └── ycbcr └──</pre>
<b>Dependências</b>	<p>Função cod_jpeg2000 (Tabela A.6);</p> <p>Função dec_jpeg2000 (Tabela A.7);</p> <p>Função psnr (Tabela A.8).</p> <p>Ver Figura 4.3.</p>

## A.6 Função `cod_jpeg2000`

Tabela A.6: Documentação da função `cod_jpeg2000`.

<b>Nome da função</b>	<code>cod_jpeg2000</code>
<b>Protótipo em Python</b>	<code>def cod_jpeg2000(in_path, out_path, ycbcr, rate)</code>
<b>Descrição</b>	Invoca ao sistema a execução do comando <code>kdu_compress</code> , conforme descrito na Secção 3.2.2.
<b>Inputs</b>	<p><code>in_path</code>: caminho do ficheiro de <i>input</i>.</p> <p><code>out_path</code>: caminho do ficheiro de <i>output</i>.  <i>Valor por defeito.</i> “out.jp2”.</p> <p><code>ycbcr</code>: <i>flag</i> que indica ao <code>kdu</code> se deve ser efetuada uma transformação de cor.  <i>Valor por defeito.</i> <code>False</code>.</p> <p><code>rate</code>: número de <i>bits</i> por amostra.  <i>Valor por defeito.</i> 1.0.</p>
<b>Output</b>	Não aplicável.
<b>Efeitos colaterais</b>	Gera uma imagem no formato *.jp2 no diretório definido por <code>out_path</code> .
<b>Dependências</b>	Não aplicável.



## A.7 Função dec\_jpeg2000

Tabela A.7: Documentação da função dec\_jpeg2000.

<b>Nome da função</b>	dec_jpeg2000
<b>Protótipo em Python</b>	<code>def dec_jpeg2000(in_path, out_path, ycbcr, rate)</code>
<b>Descrição</b>	Invoca ao sistema a execução do comando kdu_expand, conforme descrito na Secção 3.2.2.
<b>Inputs</b>	<p>in_path: caminho do ficheiro de <i>input</i>.</p> <p>out_path: caminho do ficheiro de <i>output</i>.  <i>Valor por defeito.</i> "out.jp2".</p> <p>ycbcr: <i>flag</i> que indica ao kdu se deve ser efetuada uma transformação de cor.  <i>Valor por defeito.</i> False.</p> <p>rate: número de <i>bits</i> por amostra.  <i>Valor por defeito.</i> 1.0.</p>
<b>Output</b>	Não aplicável.
<b>Efeitos colaterais</b>	Gera uma imagem no formato *.tmp no diretório definido por out_path.
<b>Dependências</b>	Não aplicável.

## A.8 Função psnr

Tabela A.8: Documentação da função psnr.

<b>Nome da função</b>	psnr
<b>Protótipo em Python</b>	<code>def psnr(p1, p2)</code>
<b>Descrição</b>	Calcula a métrica de compressão PSNR entre duas imagens.
<b>Inputs</b>	p1: caminho para a primeira imagem. p2: caminho para a segunda imagem.
<b>Output</b>	Valor calculado do PSNR.
<b>Efeitos colaterais</b>	Não aplicável.
<b>Dependências</b>	Não aplicável.