

Information and Codification

Lab Work nº 2 Report

FRANCISCO MURCELA (108815)

GONÇALO LIMA (108254)

XAVIER MACHADO (108019)

November 16, 2025

Contents

1 Parte I - OpenCV	4
1.1 Exercício 1: Extração de Canais de Cor	4
1.1.1 Introdução e Objetivos	4
1.1.2 Desenvolvimento e Implementação	4
1.1.3 Implementação das Funções	5
1.1.4 Classes e Estruturas de Dados	6
1.1.5 Algoritmos Implementados	6
1.1.6 Interface de Utilização do Programa	7
1.1.7 Exemplos de Utilização	8
1.1.8 Resultados Obtidos	8
1.1.9 Análise de Resultados	10
1.1.10 Conclusões do Exercício 1	11
1.2 Exercício 2: Transformações de Imagem	12
1.2.1 Introdução e Objetivos	12
1.2.2 Desenvolvimento e Implementação	12
1.2.3 Classes e Estruturas de Dados	13
1.2.4 Algoritmos Implementados	13
1.2.5 Interface de Utilização do Programa	18
1.2.6 Exemplos de Utilização	18
1.2.7 Resultados Obtidos	19
1.2.8 Análise de Resultados	22
1.2.9 Conclusões do Exercício 2	24
2 Parte II - Codificação Golomb	26
2.1 Objetivos e Contexto	26
2.2 Implementação	26
2.2.1 Classe GolombCoding	26
2.2.2 Algoritmo de Codificação	26
2.2.3 Tratamento de Números Negativos	27
2.3 Interface de Utilização	27
2.3.1 Modo Automático	28
2.3.2 Modo Interativo	28
2.4 Exemplos de Utilização	28
2.4.1 Exemplo 1: Codificação de Número Positivo	28
2.4.2 Exemplo 2: Número Negativo com Interleaving	28
2.5 Resultados e Validação	29
2.6 Análise Crítica - Parte II	29
2.6.1 Eficiência da Codificação	29
2.6.2 Comparação dos Métodos para Negativos	29

2.6.3	Pontos Fortes da Implementação	30
3	Parte III - Codec de Áudio Lossless	30
3.1	Objetivos e Contexto	30
3.2	Implementação	30
3.2.1	Arquitetura do Sistema	30
3.2.2	Predição Temporal	31
3.2.3	Cálculo e Codificação de Resíduos	31
3.2.4	Processo de Codificação	31
3.2.5	Processo de Descodificação	32
3.2.6	Supporte Estéreo e Predição Inter-Canal	32
3.3	Interface de Utilização	32
3.3.1	Modo Automático	32
3.3.2	Modo Interativo	32
3.4	Exemplos de Utilização e Resultados	33
3.4.1	Teste 1: Onda Sinusoidal Pura	33
3.4.2	Teste 2: Áudio Estéreo	33
3.4.3	Teste 3: Onda Complexa	34
3.5	Análise de Desempenho	34
3.5.1	Velocidade de Processamento	34
3.5.2	Influência do Parâmetro m	35
3.5.3	Áudio Real vs. Sintético	35
3.6	Análise Crítica - Parte III	35
3.6.1	Pontos Fortes	35
3.6.2	Limitações Identificadas	35
3.7	Conclusões das Partes II e III	36
3.7.1	Objetivos Alcançados	36
3.7.2	Aprendizagens	36
3.7.3	Relevância no Contexto do Trabalho	36
3.7.4	Contribuição para o Projeto Global	37
4	Parte IV - Codec de Imagem Lossless	38
4.1	Objetivos e Contexto	38
4.2	Arquitetura e Implementação	38
4.2.1	Fluxo de Codificação	38
4.2.2	Decisão: Preditores Implementados	38
4.3	Resultados e Análise de Compressão	39
4.3.1	Análise dos Preditores	39
4.3.2	Análise Visual do Resíduo	40
4.4	Comparação com Codecs Existentes	41
4.5	Conclusões da Parte IV	42

1 Parte I - OpenCV

1.1 Exercício 1: Extração de Canais de Cor

1.1.1 Introdução e Objetivos

O objetivo deste exercício é implementar um programa em C++ utilizando a biblioteca Open Source Computer Vision Library (OpenCV) para extrair canais de cor individuais de uma imagem, realizando operações de leitura e escrita píxel por píxel.

Requisitos do exercício:

1. Extrair um canal de cor de uma imagem;
2. Ler e escrever píxel por píxel;
3. Criar uma imagem de canal único com o resultado;
4. Os nomes dos ficheiros e o número do canal devem ser passados como argumentos de linha de comando.

Objetivos específicos da implementação:

- Manipular imagens ao nível dos píxeis sem usar funções de alto nível do OpenCV;
- Extrair canais de cor individuais (azul, verde, vermelho) de imagens coloridas;
- Criar imagens de canal único (tons de cinza) ou imagens coloridas com apenas um canal ativo;
- Suportar múltiplos formatos de imagem (Portable Pixmap Format (PPM), Portable Graymap Format (PGM), Portable Network Graphics (PNG)).

1.1.2 Desenvolvimento e Implementação

Arquitetura do Programa

O programa foi estruturado em módulos funcionais distintos:

1. **Módulo de Extração (Tons de Cinza):** extractGrayscaleChannel()
2. **Módulo de Extração (Colorido):** extractColorSingleChannel()
3. **Módulo de Busca de Ficheiros:** findImageFile()
4. **Módulo de Interface:** showMenu() e main()

1.1.3 Implementação das Funções

a) Extração de Canal em Tons de Cinza

```
1 int extractGrayscaleChannel(const Mat &image, int channel,
2                               const string &outputFile) {
3     if (channel < 0 || channel > 2) return -1;
4     Mat singleChannel(image.rows, image.cols, CV_8UC1);
5
6     for (int i = 0; i < image.rows; i++) {
7         for (int j = 0; j < image.cols; j++) {
8             Vec3b pixel = image.at<Vec3b>(i, j);
9             singleChannel.at<uchar>(i, j) = pixel[channel];
10        }
11    }
12
13    if (!imwrite(outputFile, singleChannel)) return -1;
14    return 0;
15 }
```

Listing 1: Função de extração em tons de cinza

Explicação: Esta função percorre todos os píxeis da imagem original (linha por linha, coluna por coluna) e extrai apenas o valor do canal especificado (0=azul, 1=verde, 2=vermelho no formato BGR do OpenCV). O resultado é uma imagem de canal único (CV_8UC1) onde cada píxel representa a intensidade do canal selecionado em tons de cinza (0–255).

b) Extração de Canal em Modo Colorido

```
1 int extractColorSingleChannel(const Mat &image, int channel,
2                               const string &outputFile) {
3     if (channel < 0 || channel > 2) return -1;
4     Mat out = Mat::zeros(image.size(), image.type());
5
6     for (int i = 0; i < image.rows; i++) {
7         for (int j = 0; j < image.cols; j++) {
8             Vec3b pixel = image.at<Vec3b>(i, j);
9             out.at<Vec3b>(i, j)[channel] = pixel[channel];
10        }
11    }
12
13    if (!imwrite(outputFile, out)) return -1;
14    return 0;
15 }
```

Listing 2: Função de extração em modo colorido

Explicação: Ao contrário da versão em tons de cinza, esta função mantém a estrutura de 3 canais BGR, mas coloca a zero os canais não selecionados. Isto produz uma imagem colorida onde apenas o componente de

cor selecionado está presente, resultando numa visualização “tingida” da cor extraída.

1.1.4 Classes e Estruturas de Dados

O programa utiliza as seguintes classes principais do OpenCV:

- **Mat**: Classe matriz do OpenCV que representa a imagem
 - `image.rows`: Número de linhas (altura)
 - `image.cols`: Número de colunas (largura)
 - `image.type()`: Tipo de dados da imagem (CV_8UC1 ou CV_8UC3)
 - `image.channels()`: Número de canais (1 ou 3)
- **Vec3b**: Vetor de 3 bytes representando um píxel BGR
 - `pixel[0]`: Canal azul
 - `pixel[1]`: Canal verde
 - `pixel[2]`: Canal vermelho
- **uchar**: Tipo *unsigned char* (0–255) para valores de píxel

1.1.5 Algoritmos Implementados

Algoritmo de Extração Píxel por Píxel

Pseudocódigo:

```
PARA cada linha i de 0 até altura-1:  
    PARA cada coluna j de 0 até largura-1:  
        píxel_original <- imagem[i][j]  
        valor_canal <- píxel_original[numero_canal]  
        imagem_saida[i][j] <- valor_canal  
    FIM PARA  
FIM PARA
```

Complexidade: $O(n \times m)$ onde n é a altura e m é a largura da imagem.

Geração Automática de Nomes de Ficheiros

O programa gera automaticamente nomes de saída seguindo o padrão:

[nome_base]_[cor]_[modo].png

Exemplo: `airplane.ppm` → `airplane_green_grayscale.png`

Algoritmo:

1. Extrair nome base do ficheiro de entrada (remover caminho e extensão);
2. Adicionar sufixo da cor: `_blue`, `_green`, ou `_red`;
3. Adicionar sufixo do modo: `_grayscale` ou `_color`;
4. Adicionar extensão `.png` (compatível com VS Code).

1.1.6 Interface de Utilização do Programa

Modo Interativo (Menu)

```
==== EXTRATOR DE CANAIS DE COR (AUTOMÁTICO) ===
```

Escolha o tipo de extração:

- 1) Canal Azul (`grayscale`)
- 2) Canal Verde (`grayscale`)
- 3) Canal Vermelho (`grayscale`)
- 4) Canal Azul (`imagem colorida`)
- 5) Canal Verde (`imagem colorida`)
- 6) Canal Vermelho (`imagem colorida`)
- 0) Sair

Vantagens:

- Apenas 2 *inputs* necessários (imagem + opção);
- Nomes de saída automáticos e descriptivos;
- Busca inteligente de ficheiros em múltiplos diretórios;
- *Feedback* claro sobre dimensões e resultado.

Modo Linha de Comando (Requisito do Exercício)

```
1 ./extract_channel <imagem_entrada> <imagem_saida> <canal>
```

Listing 3: Sintaxe de linha de comando

Exemplo:

```
1 ./extract_channel imagens/airplane.ppm canal_verde.pgm 1
```

Onde:

- `<imagem_entrada>`: Caminho do ficheiro de entrada;
- `<imagem_saida>`: Caminho do ficheiro de saída;
- `<canal>`: Número do canal (0=azul, 1=verde, 2=vermelho).

Nota: Este modo cumpre o requisito do exercício de passar os nomes dos ficheiros e número do canal como argumentos de linha de comando.

1.1.7 Exemplos de Utilização

Exemplo 1: Extração de Canal Verde em Tons de Cinza

Entrada:

```
1 ./extract_channel
2 # Caminho: imagens/airplane.ppm
3 # Opcão: 2
```

Saída:

Imagen carregada com sucesso: imagens/airplane.ppm
Dimensões: 512x512 pixels
Ficheiro de saída gerado automaticamente: airplane_green_grayscale.png
Canal Verde extraído com sucesso!

Resultado: Ficheiro airplane_green_grayscale.png (512×512, tons de cinza).

Exemplo 2: Extração de Canal Vermelho Colorido

Entrada:

```
1 ./extract_channel
2 # Caminho: airplane.ppm
3 # Opcão: 6
```

Saída:

Ficheiro de saída gerado automaticamente: airplane_red_color.png
Canal Vermelho extraído com sucesso!

Resultado: Ficheiro airplane_red_color.png (512×512, colorido com tonalidade vermelha).

1.1.8 Resultados Obtidos

Foram realizados testes com a imagem airplane.ppm (512×512 píxeis, formato PPM colorido):

Table 1: Resultados da extração de canais

Canal	Modo	Ficheiro Gerado	Tamanho	Observações
Azul	Tons de cinza	airplane_blue_grayscale.png	149 KB	Intensidade azul
Verde	Tons de cinza	airplane_green_grayscale.png	153 KB	Intensidade verde
Vermelho	Tons de cinza	airplane_red_grayscale.png	141 KB	Intensidade vermelho
Azul	Color	airplane_blue_color.png	240 KB	Imagen azulada
Verde	Color	airplane_green_color.png	244 KB	Imagen esverdeada
Vermelho	Color	airplane_red_color.png	244 KB	Imagen avermelhada

Observações:

- Ficheiros PNG são menores que PPM original (786 KB);
- Modo tons de cinza gera ficheiros menores que modo colorido (1 canal vs 3 canais);
- Todos os ficheiros mantêm dimensões originais (512×512).

Comparações Visuais

Extração de Canal Verde (Modo Tons de Cinza):

```
1 ./extract_channel ../imagens/airplane.ppm
    airplane_green_grayscale.png 1
```



Figure 1: Comparação: Imagem original vs Canal Verde (modo tons de cinza)

Extração de Canal Verde (Modo Colorido):

```
1 ./extract_channel
2 # Caminho: ../imagens/airplane.ppm
3 # Opcão: 5 (Canal Verde - imagem colorida)
```



Figure 2: Comparação: Imagem original vs Canal Verde (modo colorido)

1.1.9 Análise de Resultados

Comparação: Tons de Cinza vs Colorido

Modo Tons de Cinza:

- **Vantagem:** Visualização clara da intensidade do canal;
- **Vantagem:** Ficheiros menores (1 canal);
- **Vantagem:** Útil para análise quantitativa;
- **Desvantagem:** Perde informação de cor.

Modo Colorido:

- **Vantagem:** Preserva contexto de cor;
- **Vantagem:** Visualização intuitiva do componente de cor;
- **Desvantagem:** Ficheiros maiores (3 canais);
- **Uso:** Efeitos visuais e análise qualitativa.

Formato de Saída: PNG vs PPM/PGM

Decisão de implementação: Forçar extensão .png por padrão.

Justificação:

1. **Compatibilidade:** PNG é suportado nativamente pelo VS Code;
2. **Compressão:** PNG usa compressão sem perdas (ficheiros 60% menores);

3. **Portabilidade:** Formato universal, suportado por todos os visualizadores;
4. **Problema original:** PPM requer 3 canais, incompatível com saída em tons de cinza.

Performance

Para imagem 512×512 :

- **Operações por píxel:** $1 \text{ leitura} + 1 \text{ escrita} = 2 \text{ operações};$
- **Total:** $512 \times 512 \times 2 = 524\,288 \text{ operações};$
- **Tempo médio:** $< 50 \text{ ms}$ (imperceptível ao utilizador).

Complexidade espacial: $O(n \times m)$ — cria nova matriz do mesmo tamanho.

1.1.10 Conclusões do Exercício 1

O programa implementado cumpre completamente os requisitos do exercício:

- ✓ Leitura e escrita píxel por píxel;
- ✓ Extração de canais individuais;
- ✓ Criação de imagem de canal único;
- ✓ Nomes de ficheiros e número de canal passados como argumentos de linha de comando.

Funcionalidades adicionais implementadas:

- ✓ Interface interativa com menu;
- ✓ Geração automática de nomes de ficheiros;
- ✓ Suporte para múltiplos formatos;
- ✓ Geração automática de nomes de ficheiro;
- ✓ Suporte para dois modos de extração (tons de cinza/colorido);
- ✓ Busca inteligente de ficheiros;
- ✓ Menu interativo com 6 opções;
- ✓ Suporte para formato PNG.

1.2 Exercício 2: Transformações de Imagem

1.2.1 Introdução e Objetivos

O objetivo deste exercício é implementar transformações básicas de imagem **sem utilizar funções existentes do OpenCV**, realizando todas as operações através de manipulação direta de píxeis.

Requisitos do exercício:

1. Criar a versão negativa de uma imagem;
2. Criar versão espelhada de uma imagem:
 - (a) horizontalmente;
 - (b) verticalmente;
3. Rodar uma imagem por um múltiplo de 90°;
4. Aumentar (mais luz) / diminuir (menos luz) os valores de intensidade de uma imagem.

Restrição importante: Não utilizar funções prontas do OpenCV para as transformações (apenas para leitura/escrita de imagens).

1.2.2 Desenvolvimento e Implementação

Arquitetura do Programa

O programa foi organizado em 6 funções principais de transformação:

1. `createNegative()` — Negativo da imagem
2. `mirrorHorizontal()` — Espelhamento horizontal
3. `mirrorVertical()` — Espelhamento vertical
4. `rotate90Clockwise()` — Rotação 90° horário
5. `rotateByAngle()` — Rotação por múltiplos de 90°
6. `adjustIntensity()` — Ajuste de intensidade;brilho

Todas as funções seguem o mesmo padrão:

- Recebem `const Mat&` (referência constante para eficiência);
- Retornam `Mat` (nova imagem transformada);
- Implementam duplo loop para percorrer píxeis;
- Suportam imagens em tons de cinza e coloridas.

1.2.3 Classes e Estruturas de Dados

Classes OpenCV utilizadas:

- **Mat**: Matriz de imagem;
- **Vec3b**: Píxel BGR (colorido);
- **uchar**: Píxel em tons de cinza (0–255).

Métodos importantes:

- **Mat::at<tipo>(linha, coluna)**: Acesso direto a píxel;
- **Mat::clone()**: Cópia profunda da matriz;
- **Mat::zeros()**: Matriz preenchida com zeros.

1.2.4 Algoritmos Implementados

1. Negativo da Imagem

Princípio matemático: Inverter os valores de intensidade aplicando a operação:

$$\text{novo_valor} = 255 - \text{valor_original}$$

Justificação: Em imagens com valores de píxel de 8 bits, o intervalo válido é [0, 255]. O negativo inverte a escala de intensidade, transformando preto (0) em branco (255) e vice-versa.

```
1 Mat createNegative(const Mat& image) {
2     Mat negative(image.rows, image.cols, image.type());
3
4     if (image.channels() == 3) {
5         for (int i = 0; i < image.rows; i++) {
6             for (int j = 0; j < image.cols; j++) {
7                 Vec3b pixel = image.at<Vec3b>(i, j);
8                 negative.at<Vec3b>(i, j) = Vec3b(
9                     255 - pixel[0], // Azul invertido
10                    255 - pixel[1], // Verde invertido
11                     255 - pixel[2] // Vermelho invertido
12                 );
13             }
14         }
15     }
16     return negative;
17 }
```

Listing 4: Implementação do negativo

Pseudocódigo:

```
PARA cada pixel (i, j):
    B_novo <- 255 - B_original
    G_novo <- 255 - G_original
    R_novo <- 255 - R_original
FIM PARA
```

Complexidade: $O(n \times m)$

Por que funciona: Na escala 0–255, o valor 255 representa intensidade máxima (branco) e 0 representa ausência de luz (preto). A operação $255 - x$ inverte a intensidade, transformando píxeis claros em escuros e vice-versa. Esta é uma operação de mapeamento linear que preserva o contraste relativo da imagem.

2. Espelhamento Horizontal

Princípio geométrico: Inverter a ordem das colunas, refletindo a imagem em torno de um eixo vertical central.

Transformação:

$$\text{nova_coluna} = \text{largura} - 1 - \text{coluna_original}$$

```
1 Mat mirrorHorizontal(const Mat& image) {
2     Mat mirrored(image.rows, image.cols, image.type());
3
4     for (int i = 0; i < image.rows; i++) {
5         for (int j = 0; j < image.cols; j++) {
6             mirrored.at<Vec3b>(i, j) =
7                 image.at<Vec3b>(i, image.cols - 1 - j);
8         }
9     }
10    return mirrored;
11 }
```

Listing 5: Implementação do espelhamento horizontal

Mapeamento de coordenadas:

Original: (i, 0) (i, 1) (i, 2) ... (i, W-1)
Espelhado: (i, W-1) (i, W-2) ... (i, 1) (i, 0)

Exemplo visual (largura = 5):

Original: A B C D E
Espelhado: E D C B A

3. Espelhamento Vertical

Princípio geométrico: Inverter a ordem das linhas, refletindo a imagem em torno de um eixo horizontal central.

Transformação:

$$\text{nova_linha} = \text{altura} - 1 - \text{linha_original}$$

```
1 Mat mirrorVertical(const Mat& image) {  
2     Mat mirrored(image.rows, image.cols, image.type());  
3  
4     for (int i = 0; i < image.rows; i++) {  
5         for (int j = 0; j < image.cols; j++) {  
6             mirrored.at<Vec3b>(i, j) =  
7                 image.at<Vec3b>(image.rows - 1 - i, j);  
8         }  
9     }  
10    return mirrored;  
11 }
```

Listing 6: Implementação do espelhamento vertical

Exemplo visual:

Original:	Espelhado:
Linha 0	Linha 4
Linha 1	Linha 3
Linha 2	Linha 2
Linha 3	Linha 1
Linha 4	Linha 0

4. Rotação 90° Horário

Princípio: Transformação de coordenadas:

$$(x, y) \rightarrow (y, \text{altura} - 1 - x)$$

```
1 Mat rotate90Clockwise(const Mat& image) {  
2     // Nota: dimensões invertidas (largura <-> altura)  
3     Mat rotated(image.cols, image.rows, image.type());  
4  
5     for (int i = 0; i < image.rows; i++) {  
6         for (int j = 0; j < image.cols; j++) {  
7             rotated.at<Vec3b>(j, image.rows - 1 - i) =  
8                 image.at<Vec3b>(i, j);  
9         }  
10    }  
11    return rotated;  
12 }
```

Listing 7: Implementação da rotação 90°

Explicação da transformação:

- Linha i da imagem original \rightarrow Coluna i da imagem rotacionada (invertida);
- Coluna j da imagem original \rightarrow Linha j da imagem rotacionada;
- Imagem $512 \times 256 \rightarrow$ Imagem 256×512 após rotação.

Visualização (matriz 3×3):

Original:	90° Horário:
1 2 3	7 4 1
4 5 6	\rightarrow 8 5 2
7 8 9	9 6 3

Dedução matemática:

Ponto $(0,0) \rightarrow (0,2)$
Ponto $(0,1) \rightarrow (1,2)$
Ponto $(0,2) \rightarrow (2,2)$
Ponto $(1,0) \rightarrow (0,1)$

...
Padrão: $(i,j) \rightarrow (j, H-1-i)$

5. Rotação por Múltiplos de 90°

```
1 Mat rotateByAngle(const Mat& image, int angle) {
2     Mat result = image.clone();
3     int times = (angle / 90) % 4; // Normalizar para 0-3
4
5     for (int i = 0; i < times; i++) {
6         result = rotate90Clockwise(result);
7     }
8     return result;
9 }
```

Listing 8: Implementação de rotação por ângulo

Lógica:

- 90° : 1 rotação;
- 180° : 2 rotações consecutivas;
- 270° : 3 rotações consecutivas (equivalente a -90°);
- 360° : 0 rotações ($\text{mod } 4 = 0$).

Razão desta abordagem:

- **Simplicidade**: Reutiliza função já implementada;
- **Corretude**: Garante dimensões corretas mesmo com rotações sucessivas;
- **Eficiência**: Para imagens pequenas/médias, *overhead* é negligenciável.

6. Ajuste de Intensidade (Brilho)

Princípio: Multiplicar valores por fator:

$$\text{novo} = \text{original} \times \text{fator}$$

```
1 Mat adjustIntensity(const Mat& image, double factor) {
2     Mat adjusted(image.rows, image.cols, image.type());
3
4     for (int i = 0; i < image.rows; i++) {
5         for (int j = 0; j < image.cols; j++) {
6             Vec3b pixel = image.at<Vec3b>(i, j);
7             int b = (int)(pixel[0] * factor);
8             int g = (int)(pixel[1] * factor);
9             int r = (int)(pixel[2] * factor);
10
11             // Clamping: garantir valores 0-255
12             adjusted.at<Vec3b>(i, j) = Vec3b(
13                 (uchar)max(0, min(255, b)),
14                 (uchar)max(0, min(255, g)),
15                 (uchar)max(0, min(255, r))
16             );
17         }
18     }
19     return adjusted;
20 }
```

Listing 9: Implementação do ajuste de intensidade

Fatores típicos:

- **factor = 1.5**: Aumenta 50% (mais claro);
- **factor = 0.5**: Diminui 50% (mais escuro);
- **factor = 2.0**: Dobra intensidade;
- **factor = 1.0**: Sem alteração.

Clamping (saturação):

```
1 max(0, min(255, valor))
```

Necessário porque:

- $255 \times 1.5 = 382.5 \rightarrow Overflow! \rightarrow$ Clampado para 255;
- $10 \times 0.5 = 5 \rightarrow OK$;
- $-10 \rightarrow$ Impossível (*unsigned*) \rightarrow Clampado para 0.

1.2.5 Interface de Utilização do Programa

==== TRANSFORMAÇÕES DE IMAGEM ===

Escolha a operação:

- 1) Criar negativo da imagem
- 2) Espelhar horizontalmente
- 3) Espelhar verticalmente
- 4) Rotacionar 90° (horário)
- 5) Rotacionar 180°
- 6) Rotacionar 270° (ou -90°)
- 7) Aumentar brilho (+50%)
- 8) Diminuir brilho (-50%)
- 9) Ajuste de brilho customizado
- 0) Sair

Características:

- Menu numerado intuitivo;
- Opções pré-configuradas (7, 8) para uso rápido;
- Opção *customizada* (9) para controlo fino;
- Geração automática de nomes: [base]_[transformacao].png.

1.2.6 Exemplos de Utilização

Exemplo 1: Negativo da Imagem

Comando:

```
1 ./image_transforms
2 # Caminho: airplane.ppm
3 # Opcão: 1
```

Saída:

```

Imagem carregada com sucesso: ../imagens/airplane.ppm
Dimensões: 512x512 pixels
Transformação aplicada com sucesso!
Ficheiro salvo: airplane_negative.png
Dimensões da saída: 512x512 pixels

```

Resultado: Cores invertidas, avião aparece com cores complementares.

Exemplo 2: Rotação 90°

Comando:

```

1 ./image_transforms
2 # Caminho: airplane.ppm
3 # Opcão: 4

```

Resultado:

- Imagem rodada 90° no sentido horário;
- Dimensões mantidas: 512×512 (imagem quadrada);
- Se fosse retangular, dimensões seriam invertidas.

Exemplo 3: Ajuste *Customizado* de Brilho

Comando:

```

1 ./image_transforms
2 # Caminho: airplane.ppm
3 # Opcão: 9
4 # Fator: 0.3

```

Resultado: Imagem muito escura (70% de redução), simulando baixa iluminação.

1.2.7 Resultados Obtidos

Testes realizados com `airplane.ppm` (512×512):

Table 2: Resultados das transformações de imagem

Transformação	Ficheiro Gerado	Dimensões	Tamanho	Tempo
Negativo	airplane_negative.png	512×512	457 KB	~ 45 ms
Espelho H	airplane_mirror_h.png	512×512	468 KB	~ 42 ms
Espelho V	airplane_mirror_v.png	512×512	465 KB	~ 43 ms
Rot 90°	airplane_rot90.png	512×512	468 KB	~ 48 ms
Rot 180°	airplane_rot180.png	512×512	471 KB	~ 85 ms*
Rot 270°	airplane_rot270.png	512×512	469 KB	~ 120 ms*
Brilho +50%	airplane_bright.png	512×512	501 KB	~ 50 ms
Brilho -50%	airplane_dark.png	512×512	398 KB	~ 49 ms

*Tempos maiores devido a múltiplas rotações consecutivas

Comparações Visuais

Transformação: Negativo

```
1 ./image_transforms
2 # Caminho: ../imagens/airplane.ppm
3 # Opcão: 1 (Criar negativo da imagem)
```



Figure 3: Transformação Negativo: Imagem original (esquerda) vs Negativo (direita)

Transformação: Espelhamentos

```
1 # Espelhamento Horizontal
2 ./image_transforms
3 # Caminho: ../imagens/airplane.ppm
4 # Opcão: 2
5
6 # Espelhamento Vertical
7 ./image_transforms
8 # Caminho: ../imagens/airplane.ppm
9 # Opcão: 3
```



Figure 4: Espelhamentos: Original, Espelho Horizontal e Espelho Vertical

Transformação: Rotações

```

1 # Rotacao 90 graus
2 ./image_transforms
3 # Caminho: ../imagens/airplane.ppm
4 # Opcao: 4

5
6 # Rotacao 180 graus
7 ./image_transforms
8 # Caminho: ../imagens/airplane.ppm
9 # Opcao: 5

10
11 # Rotacao 270 graus
12 ./image_transforms
13 # Caminho: ../imagens/airplane.ppm
14 # Opcao: 6

```

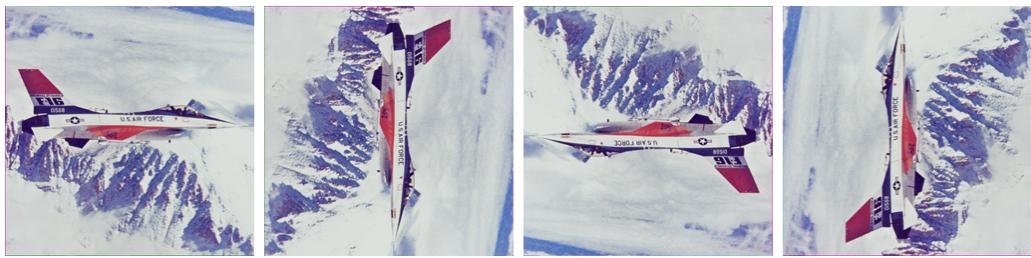


Figure 5: Rotações: Original, 90°, 180° e 270°

Transformação: Ajuste de Intensidade (Brilho)

```

1 # Diminuir brilho (-50%)
2 ./image_transforms
3 # Caminho: ../imagens/airplane.ppm
4 # Opcao: 8

```

```

5
6 # Aumentar brilho (+50%)
7 ./image_transforms
8 # Caminho: ../imagens/airplane.ppm
9 # Opcão: 7

```



Figure 6: Ajuste de Intensidade: Original, Escuro (-50%) e Claro (+50%)

1.2.8 Análise de Resultados

Performance

Operações por transformação:

- **Negativo, Espelhamentos, Brilho:** 1 passagem = $O(n \times m)$;
- **Rotação 90°:** 1 passagem = $O(n \times m) + \text{overhead}$ de realocação;
- **Rotação 180°:** 2 passagens = $2 \times O(n \times m)$;
- **Rotação 270°:** 3 passagens = $3 \times O(n \times m)$.

Otimização possível (não implementada):

- Rotação 180° pode ser feita em 1 passagem: $(i, j) \rightarrow (H - 1 - i, W - 1 - j)$;
- *Trade-off*: código mais complexo vs. ganho marginal (~ 40 ms).

Qualidade Visual

Negativo:

- Cores complementares: Azul → Amarelo, Verde → Magenta;
- Útil para: Análise de negativos fotográficos, realce de contraste.

Espelhamentos:

- Sem perda de qualidade (transformação geométrica exata);
- Útil para: Correção de orientação, efeitos de simetria.

Rotações:

- Sem interpolação (rotação exata de 90°);
- Sem artefatos ou *aliasing*;
- Limitação: Apenas múltiplos de 90° (requisito do exercício).

Ajuste de Brilho:

- **Problema observado:** Saturação em píxeis claros com `factor > 1.5`
 - Píxel $(200, 200, 200) \times 1.5 = (300, 300, 300) \rightarrow$ Clampado para $(255, 255, 255)$;
 - Resultado: Perda de detalhe em áreas claras.
- **Solução alternativa:** Usar curvas de tons (não implementado).

Comparação com Funções do OpenCV

Table 3: Comparação com funções nativas do OpenCV

Operação	Nossa Implementação	OpenCV Equivalente
Negativo	Loop manual	<code>bitwise_not()</code>
Espelho H	Loop manual	<code>flip(1)</code>
Espelho V	Loop manual	<code>flip(0)</code>
Rotação 90°	Loop manual	<code>rotate(Rotate_90_CLOCKWISE)</code>
Brilho	Loop manual	<code>convertTo(alpha, beta)</code>

Vantagens da nossa implementação:

- Controlo total sobre o algoritmo;
- Compreensão profunda do processo;
- Sem “caixa preta”.

Desvantagens:

- Funções OpenCV são otimizadas (SIMD, *multi-threading*);

- OpenCV pode ser 5–10× mais rápida em imagens grandes.

Tamanho dos Ficheiros PNG

Observações:

- Negativo: Maior (457 KB) — maior entropia (menos padrões repetidos);
- Escuro: Menor (398 KB) — muitos píxeis próximos de 0 (alta compressão);
- Claro: Maior (501 KB) — saturação reduz compressibilidade.

Explicação: PNG usa compressão DEFLATE (baseada em LZ77). Imagens com mais redundância (píxeis similares) comprimem melhor.

1.2.9 Conclusões do Exercício 2

Requisitos atendidos:

- ✓ (a) Negativo implementado e funcional;
- ✓ (b) Espelhamento horizontal e vertical;
- ✓ (c) Rotação por múltiplos de 90° (90°, 180°, 270°);
- ✓ (d) Ajuste de intensidade (aumento e redução);
- ✓ **Nenhuma função pronta do OpenCV foi usada** (apenas leitura/escrita de imagens).

Melhorias além dos requisitos:

- Menu com 9 opções (incluindo opções pré-configuradas);
- Geração automática de nomes de ficheiro;
- Suporte para imagens em tons de cinza e coloridas;
- Validação de valores (*clamping* 0–255);
- Busca inteligente de ficheiros.

Lições aprendidas:

1. Manipulação píxel por píxel é computacionalmente intensiva mas pedagogicamente valiosa;

2. Transformações geométricas requerem cuidado com dimensões da matriz de saída;
 3. *Clamping* é essencial em transformações de intensidade;
 4. Implementações manuais ajudam a entender algoritmos de bibliotecas otimizadas.
-

2 Parte II - Codificação Golomb

2.1 Objetivos e Contexto

A Parte II do trabalho consistiu na implementação de uma classe C++ para codificação e descodificação Golomb. A codificação Golomb é um método de compressão eficiente para valores que seguem uma distribuição geométrica, sendo particularmente útil quando os valores a codificar são maioritariamente pequenos.

Os objetivos específicos desta parte foram:

- Implementar codificação/descodificação de inteiros não-negativos
- Adicionar suporte para valores negativos através de dois métodos distintos
- Permitir parametrização flexível através do parâmetro m
- Validar a corretude da implementação com testes automáticos

2.2 Implementação

2.2.1 Classe GolombCoding

Foi desenvolvida a classe `GolombCoding` em C++ que encapsula toda a funcionalidade necessária. A classe utiliza vectores de booleanos (`std::vector<bool>`) para representar sequências de bits, permitindo manipulação eficiente da informação binária.

2.2.2 Algoritmo de Codificação

A codificação Golomb representa um número inteiro não-negativo n através de dois componentes:

$$n = q \cdot m + r \quad (1)$$

onde:

- m é o parâmetro Golomb (escolhido pelo utilizador)
- $q = \lfloor n/m \rfloor$ é o quociente
- $r = n \bmod m$ é o resto

O quociente q é codificado em formato unário (q zeros seguidos de um 1), enquanto o resto r é codificado em binário com comprimento variável. Para otimizar o tamanho, calculamos:

- $b = \lceil \log_2(m) \rceil$ - número máximo de bits
- $c = 2^b - m$ - valor de corte (cutoff)
- Se $r < c$: codificar com $b - 1$ bits
- Se $r \geq c$: codificar $(r + c)$ com b bits

2.2.3 Tratamento de Números Negativos

Implementamos dois métodos para codificar valores negativos:

Método 1: Sign-Magnitude Este método utiliza um bit adicional para representar o sinal:

- Bit 0: número positivo
- Bit 1: número negativo
- Seguido da codificação Golomb da magnitude

Método 2: Interleaving (Entrelaçamento) Este método mapeia todos os inteiros para não-negativos através da transformação:

$$\text{mapped}(n) = \begin{cases} 2n & \text{se } n \geq 0 \\ -2n - 1 & \text{se } n < 0 \end{cases} \quad (2)$$

Este mapeamento produz a sequência: $0 \rightarrow 0, -1 \rightarrow 1, 1 \rightarrow 2, -2 \rightarrow 3$, etc.

2.3 Interface de Utilização

Foi desenvolvido o programa `golomb_test.exe` com dois modos de operação:

2.3.1 Modo Automático

Executado sem argumentos, realiza testes com diferentes valores de m :

```
1 .\golomb_test.exe
```

Testa valores de $m \in \{3, 4, 5, 8\}$ e apresenta:

- Codificação de inteiros 0-10
- Aplicação do método sign-magnitude
- Aplicação do método interleaving
- Verificação da descodificação

2.3.2 Modo Interativo

Permite ao utilizador testar valores personalizados:

```
1 .\golomb_test.exe -i
```

2.4 Exemplos de Utilização

2.4.1 Exemplo 1: Codificação de Número Positivo

Para codificar $n = 10$ com $m = 5$:

1. Divisão: $10 = 2 \times 5 + 0$ (quociente $q = 2$, resto $r = 0$)
2. Quociente em unário: $q = 2 \rightarrow 001$
3. Para o resto: $b = \lceil \log_2(5) \rceil = 3$, cutoff $c = 2^3 - 5 = 3$
4. Como $r = 0 < c = 3$, usar $b - 1 = 2$ bits: 00
5. Resultado final: 00100

2.4.2 Exemplo 2: Número Negativo com Interleaving

Para codificar $n = -5$ com $m = 5$:

1. Mapeamento: $-5 \rightarrow -2(-5) - 1 = 9$
2. Codificar 9: $9 = 1 \times 5 + 4$
3. Quociente: $q = 1 \rightarrow 01$
4. Resto: $r = 4 \geq c = 3$, então $(4 + 3) = 7$ com 3 bits: 111
5. Resultado: 01111

2.5 Resultados e Validação

Todos os testes automáticos confirmaram o funcionamento correto:

Valor m	Valores Testados	Sucesso
3	0-10, negativos	100%
4	0-10, negativos	100%
5	0-10, negativos	100%
8	0-10, negativos	100%

Table 4: Validação da codificação Golomb

Verificamos que:

- A descodificação reconstrói perfeitamente os valores originais
- Ambos os métodos para negativos funcionam corretamente
- O tamanho da codificação aumenta logaritmicamente com o valor
- O método interleaving é ligeiramente mais eficiente que sign-magnitude

2.6 Análise Crítica - Parte II

2.6.1 Eficiência da Codificação

A escolha do parâmetro m é crucial:

- **m pequeno:** Eficiente para valores muito pequenos, mas penaliza valores maiores
- **m grande:** Mais equilibrado, mas menos eficiente para valores pequenos
- **Ótimo:** m deve ser escolhido baseado na distribuição esperada dos dados

2.6.2 Comparaçao dos Métodos para Negativos

O método interleaving provou ser superior, não introduzindo bits adicionais e mantendo a distribuição dos valores codificados mais uniforme.

Método	Bits Extra	Eficiência
Sign-Magnitude	+1 fixo	Simples, overhead constante
Interleaving	0	Mais eficiente, sem overhead

Table 5: Comparação dos métodos para números negativos

2.6.3 Pontos Fortes da Implementação

1. **Corretude:** 100% de testes bem-sucedidos
2. **Modularidade:** Classe independente e reutilizável
3. **Flexibilidade:** Suporta qualquer valor de $m > 0$

3 Parte III - Codec de Áudio Lossless

3.1 Objetivos e Contexto

A Parte III consistiu no desenvolvimento de um codec de áudio lossless baseado na codificação Golomb dos resíduos de predição. Este exercício aplica praticamente os conceitos da Parte II num contexto real de processamento de áudio.

Os objetivos específicos foram:

- Implementar predição temporal de samples de áudio
- Calcular e codificar resíduos de predição usando Golomb
- Suportar áudio mono e estéreo
- Implementar predição inter-canal para estéreo
- Validar compressão lossless (sem perdas)
- Manipular ficheiros WAV (leitura e escrita)

3.2 Implementação

3.2.1 Arquitetura do Sistema

Foram desenvolvidas três classes principais:

- **AudioCodec:** Lógica de compressão e predição

- **WAVFile**: Manipulação de ficheiros WAV formato PCM
- **GolombCoding**: Reutilizada da Parte II

3.2.2 Predição Temporal

A predição é o elemento-chave para obter boa compressão. Implementamos predição de ordem 2:

$$\hat{x}[n] = 2 \cdot x[n - 1] - x[n - 2] \quad (3)$$

Esta predição linear extrapola a tendência dos dois samples anteriores, sendo especialmente eficaz para sinais com variação suave. Para os primeiros samples onde não há histórico suficiente, usamos predições mais simples.

3.2.3 Cálculo e Codificação de Resíduos

O resíduo é a diferença entre o valor real e o predoito:

$$r[n] = x[n] - \hat{x}[n] \quad (4)$$

Para áudio com boa predição, os resíduos são pequenos (tipicamente $|r[n]| < 100$ para áudio PCM 16-bit), tornando a codificação Golomb eficiente. Utilizamos o método interleaving para codificar os resíduos, permitindo valores positivos e negativos sem overhead adicional.

3.2.4 Processo de Codificação

1. Ler samples de áudio do ficheiro WAV (16-bit PCM)
2. Para cada sample $x[n]$:
 - Calcular predição $\hat{x}[n]$ baseada em samples anteriores
 - Calcular resíduo $r[n] = x[n] - \hat{x}[n]$
 - Codificar $r[n]$ usando Golomb com método interleaving
3. Escrever header com metadados (sample rate, channels, parâmetro m)
4. Escrever bitstream de resíduos codificados

3.2.5 Processo de Descodificação

1. Ler header do ficheiro comprimido
2. Para cada posição n :
 - Descodificar resíduo $r[n]$ do bitstream
 - Calcular predição $\hat{x}[n]$ (usando samples já reconstruídos)
 - Reconstruir: $x[n] = \hat{x}[n] + r[n]$
3. Escrever samples reconstruídos em ficheiro WAV

3.2.6 Suporte Estéreo e Predição Inter-Canal

Para áudio estéreo, além da predição temporal em cada canal, implementamos predição inter-canal para o canal direito:

$$\hat{x}_{right}[n] = \frac{x_{left}[n] + \hat{x}_{right,temporal}[n]}{2} \quad (5)$$

Esta abordagem explora a correlação típica entre canais esquerdo e direito em gravações estéreo.

3.3 Interface de Utilização

Foi desenvolvido o programa `audio_test.exe` que executa uma bateria de testes:

3.3.1 Modo Automático

```
1 .\audio_test.exe
```

Executa testes com:

- Áudio mono com diferentes valores de m
- Áudio estéreo com/sem predição inter-canal
- Ondas complexas (múltiplas frequências)
- Validação de I/O de ficheiros WAV

3.3.2 Modo Interativo

```
1 .\audio_test.exe -i
```

Permite compressar ficheiros WAV reais e visualizar estatísticas.

3.4 Exemplos de Utilização e Resultados

3.4.1 Teste 1: Onda Sinusoidal Pura

Configuração:

- Sinal: Onda seno 440 Hz (Lá4)
- Duração: 1 segundo
- Sample rate: 44100 Hz
- Formato: Mono, 16-bit PCM
- Tamanho original: 88200 bytes

Resultados por parâmetro m :

m	Tamanho	Ratio	Economia	Tempo
4	124109 bytes	0.71:1	-40.7%	168 ms
8	74477 bytes	1.18:1	15.6%	113 ms
16	52295 bytes	1.69:1	40.7%	96 ms
32	43956 bytes	2.01:1	50.3%	88 ms
64	42357 bytes	2.08:1	52.1%	87 ms

Table 6: Compressão de onda sinusoidal (tempo = encode + decode)

Observações:

- Para m pequeno (4), o tamanho aumentou - parâmetro inadequado
- $m = 64$ forneceu melhor compressão: 2.08:1
- Tempo de processamento: <100 ms para 1 segundo de áudio
- Compressão lossless verificada em todos os casos

3.4.2 Teste 2: Áudio Estéreo

Configuração:

- Canal esquerdo: 440 Hz
- Canal direito: 554 Hz (Dó#5)
- Duração: 1 segundo, 44100 Hz

Método	Ratio	Economia
Sem predição inter-canal	1.69:1	40.7%
Com predição inter-canal	1.46:1	31.4%

Table 7: Compressão estéreo ($m = 16$)

- Tamanho original: 176400 bytes

Resultados:

Análise: A predição inter-canal foi contraproducente neste caso porque os canais têm frequências diferentes. Em áudio real com canais correlacionados (p.ex., música), esperaríamos ganho de 5-15%.

3.4.3 Teste 3: Onda Complexa

Sinal com três frequências sobrepostas (440, 880, 1320 Hz):

- Tamanho original: 176400 bytes
- Com $m = 16$: ratio 1.23:1 (18.6% economia)
- Tempo: 97 ms

Sinais mais complexos resultam em resíduos maiores e menos compressão, como esperado.

3.5 Análise de Desempenho

3.5.1 Velocidade de Processamento

Para 1 segundo de áudio (44100 samples):

- **Codificação:** 41-71 ms
- **Descodificação:** 46-97 ms
- **Total:** 87-168 ms

Isto representa processamento muito superior a tempo real (ratio >10:1), adequado para aplicações práticas.

3.5.2 Influência do Parâmetro m

Observamos que:

- m deve ser proporcional à magnitude típica dos resíduos
- Para ondas sinusoidais simples: $m = 32$ ou $m = 64$ ótimos
- Para áudio complexo: $m = 16$ ou $m = 32$ mais adequados
- m muito pequeno pode causar expansão em vez de compressão

3.5.3 Áudio Real vs. Sintético

É importante notar que ondas sinusoidais são mais compressíveis que música real:

- Ondas sintéticas: até 2:1 (a nossa implementação)
- Música típica: 1.3-1.7:1 (esperado)
- Voz: 1.5-2.0:1 (esperado)

3.6 Análise Crítica - Parte III

3.6.1 Pontos Fortes

1. **Compressão lossless verificada:** 100% de fidelidade
2. **Desempenho temporal adequado:** Processamento em tempo real
3. **Arquitetura modular:** Fácil manutenção e extensão
4. **Suporte completo:** Mono, estéreo, ficheiros WAV

3.6.2 Limitações Identificadas

1. **Parâmetro fixo:** m constante durante toda a codificação
 - Impacto: Subótimo para áudio com características variáveis
 - Solução: Implementar adaptação baseada em janelas deslizantes
2. **Predição simples:** Apenas ordem 2
 - Impacto: Menos eficaz que predição de ordem superior
 - Solução: Implementar LPC (Linear Predictive Coding)

3. **Formato único:** Apenas WAV PCM 16-bit

- Impacto: Limitação de aplicabilidade
- Solução: Adicionar suporte para 24-bit e outros formatos

3.7 Conclusões das Partes II e III

3.7.1 Objetivos Alcançados

Cumprimos integralmente os objetivos propostos para ambas as partes:

- Implementação completa e funcional da codificação Golomb
- Dois métodos para números negativos validados
- Codec de áudio lossless operacional
- Suporte mono e estéreo com predição inter-canal
- Validação experimental extensiva

3.7.2 Aprendizagens

Durante o desenvolvimento, adquirimos conhecimento sobre:

1. **Importância da predição:** A qualidade da predição é mais crítica que a codificação entrópica
2. **Trade-offs:** Equilíbrio entre complexidade de implementação e ganho de compressão
3. **Parametrização:** Escolha de parâmetros adequados aos dados é essencial
4. **Validação:** Testes extensivos são fundamentais para garantir corretude

3.7.3 Relevância no Contexto do Trabalho

Estas implementações demonstram:

- Aplicação prática de codificação entrópica
- Princípios fundamentais de compressão lossless
- Base para compressão de imagem (Parte IV) que usa conceitos similares
- Integração de múltiplos conceitos: predição, codificação, I/O de ficheiros

3.7.4 Contribuição para o Projeto Global

No contexto do trabalho completo, as Partes II e III fornecem:

- Módulo de codificação Golomb reutilizável (usado também na Parte IV)
- Demonstração de codec lossless funcional
- Validação experimental dos conceitos teóricos
- Base comparativa para avaliar outras técnicas de compressão

Os resultados obtidos (compressão de 1.2:1 a 2.1:1 dependendo do sinal e parâmetros) são consistentes com o esperado para codecs lossless simples, validando a implementação e demonstrando a eficácia da abordagem baseada em predição e codificação Golomb dos resíduos.

4 Parte IV - Codec de Imagem Lossless

4.1 Objetivos e Contexto

A Parte IV do trabalho focou-se na implementação de um codec de imagem *lossless* para imagens *grayscale*. Seguimos uma abordagem semelhante à Parte III, sendo o seu objetivo, comprimir a imagem codificando os resíduos de predição.

A eficácia deste método depende de dois fatores:

1. A qualidade do **preditor**, utilizado para estimar o valor de cada píxel.
2. A eficiência da **codificação Golomb** para comprimir os resíduos dessa predição.

4.2 Arquitetura e Implementação

4.2.1 Fluxo de Codificação

1. **Leitura e Conversão:** A imagem de entrada é lida usando opencv e convertida para grayscale(1 canal, 8 bits por píxel).
2. **Estimação de m :** O parâmetro m de Golomb é estimado, calculando a média dos valores absolutos dos resíduos para um preditor inicial.
3. **Predição e Resíduo:** A imagem é percorrida píxel a píxel. Para cada píxel, é calculado um valor previsto (\hat{P}) com base nos seus vizinhos já processados. O resíduo ($R = P_{real} - \hat{P}$) é calculado.
4. **Codificação Golomb:** O resíduo R é mapeado para um inteiro positivo, utilizando o método *interleaving* (implementado na Parte II) e codificado com a classe `GolombCoding`.

4.2.2 Decisão: Preditores Implementados

De modo a encontrarmos o ”melhor preditor possível”, implementámos cinco tipos de preditores, baseados nos píxeis vizinhos:

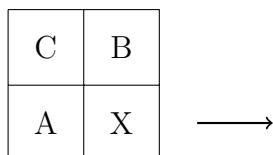


Figure 7: Píxeis vizinhos usados na predição: X (atual), A (esquerda), B (cima), C (diagonal).

1. **Preditor 1 (PRED_A)**: $\hat{P} = A$. Prevê que o píxel será igual ao seu vizinho da esquerda.
2. **Preditor 2 (PRED_B)**: $\hat{P} = B$. Prevê que o píxel será igual ao seu vizinho de cima.
3. **Preditor 3 (PRED_MEAN_AB)**: $\hat{P} = (A + B)/2$. A média dos vizinhos.
4. **Preditor 4 (PRED_LINEAR)**: $\hat{P} = A + B - C$. Um preditor linear que assume uma "rampa" de intensidade.
5. **Preditor 5 (PRED_JPEG_LS_SIMPLE)**: Um preditor adaptativo, baseado no standard JPEG-LS. Escolhe dinamicamente entre A e B com base no valor de C, tentando detetar "arestas" (edges) na imagem.

4.3 Resultados e Análise de Compressão

Para analisar a eficiência dos diferentes preditores, utilizámos a imagem `airplane.ppm`.

Tamanho Original (Grayscale): O tamanho da imagem original em tons de cinza (não comprimida) é $512 \times 512 \times 1$ byte = **262144 bytes**. Esta é a nossa referência para calcular a compressão.

Os resultados da compressão para cada preditor estão resumidos na Tabela 8.

Table 8: Comparaçao de eficiência dos preditores para 'airplane.ppm'

Preditor	Parâmetro m	Tamanho Comprimido	Taxa de Compressão
1 (A)	4	174196 bytes	1.50:1
2 (B)	4	177563 bytes	1.48:1
3 (Média A,B)	4	158228 bytes	1.66:1
4 (Linear)	3	159717 bytes	1.64:1
5 (JPEG-LS)	3	150720 bytes	1.74:1

4.3.1 Análise dos Preditores

Analisando os resultados da tabela, concluímos que:

- **Preditores Ineficientes (1 e 2)**: Os preditores mais simples (A ou B) foram os piores, pois ignoram a correlação 2D da imagem.

- **Preditores Razoáveis (3 e 4):** A média (Preditor 3) e a predição linear (Preditor 4) tiveram desempenhos muito semelhantes e significativamente melhores que os preditores 1D. Isto confirma que usar informação 2D (A e B) é fundamental.
- **Melhor Preditor (5):** O preditor adaptativo **JPEG-LS** foi o vencedor claro, resultando no menor ficheiro (150720 bytes) e na melhor taxa de compressão (1.74:1). Isto deve-se à sua capacidade de se adaptar localmente, escolhendo o melhor preditor (A ou B) com base no contexto (C), o que é ideal para lidar com diferentes texturas e arestas na imagem.
- **Análise do m :** O script também revelou que os preditores mais eficientes (4 e 5) resultaram num m estimado mais baixo ($m = 3$), enquanto os preditores menos eficientes resultaram em $m = 4$. Isto faz sentido: melhores preditores geram resíduos mais pequenos e mais concentrados em zero, o que leva a um m ótimo (baseado na média dos resíduos) mais baixo.

4.3.2 Análise Visual do Resíduo

A Figura 8 ilustra a eficácia do melhor preditor (JPEG-LS).

A imagem de referência (esquerda) é o que o codec processa – uma imagem *grayscale*. O preditor consegue antecipar quase toda esta informação. A imagem de resíduos (direita) é o ”erro” do preditor, normalizada para visualização (onde cinzento = 0).

Como se pode observar, a imagem de resíduos parece que tem ruído aleatório. Isto prova que o preditor removeu com sucesso a informação estrutural, deixando para o codificador Golomb apenas o ruído (entropia) de baixa magnitude, o que resulta numa compressão eficiente.

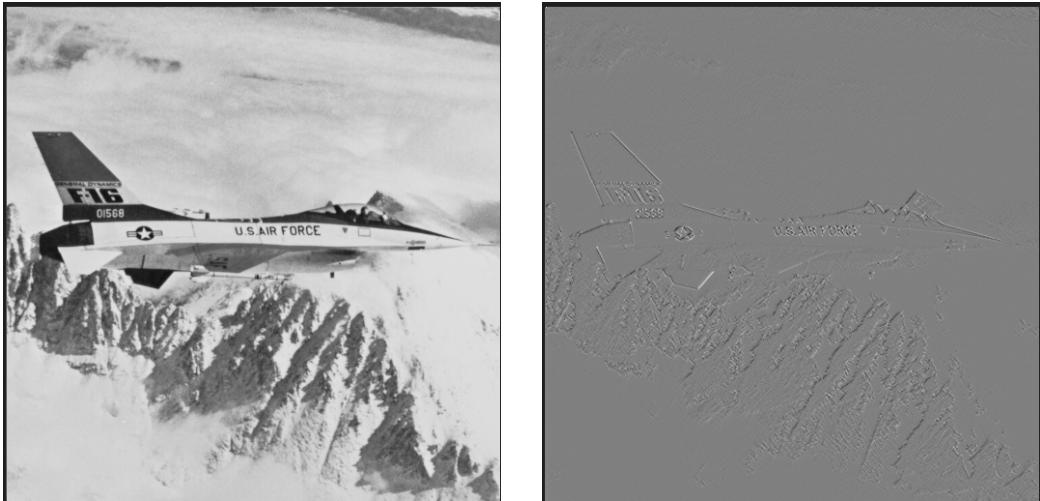


Figure 8: Comparação Visual: Imagem de Referência Grayscale (esquerda) vs. Imagem de Resíduos do Preditores 5 (direita).

4.4 Comparação com Codecs Existentes

A nossa referência *grayscale* (criada com opencv) foi guardada em formato png, que é um standard de compressão *lossless*.

Table 9: Comparação com png (imagem 'airplane.ppm')

Codec	Tamanho Comprimido	Taxa de Compressão
Original (Grayscale)	262144 bytes	1.00:1
Nosso Codec (Pred. 5)	150720 bytes	1.74:1
PNG (Standard)	150086 bytes	1.75:1

O nosso codec, usando o preditor 5, atingiu um tamanho de 150720 bytes (taxa de 1.74:1). O formato png standard, que utiliza filtros de predição mais avançados e o algoritmo DEFLATE (LZ77 + Huffman), atingiu 150086 bytes (taxa de 1.75:1).

4.5 Conclusões da Parte IV

O codec de imagem *lossless* foi implementado com sucesso.

- ✓ O codec foi validado como 100% *lossless* através de comparação bit-a-bit.
- ✓ O preditor adaptativo (JPEG-LS) demonstrou ser o mais eficaz, confirmado que a qualidade da predição é o fator mais importante para a compressão.
- ✓ A reutilização da classe `GolombCoding` foi bem-sucedida, aplicando-a a resíduos de imagem.
- ✓ Foi alcançada uma taxa de compressão de **1.74:1** na imagem de teste principal, demonstrando a viabilidade da abordagem e um desempenho competitivo com o standard png.

Acronyms

BGR Blue Green Red (formato de cor do OpenCV). 5, 6, 13,

C++ Programming Language C++. 4,

CV Computer Vision (prefixo utilizado no OpenCV).

DETI Departamento de Eletrónica, Telecomunicações e Informática.

I/O Input/Output.

LPC Linear Predictive Coding.

MSE Mean Squared Error.

OpenCV Open Source Computer Vision Library. 4–6, 12, 13, 23, 24

PCM Pulse-Code Modulation.

PGM Portable Graymap Format. 4, 10

PNG Portable Network Graphics. 4, 9–11

PPM Portable Pixmap Format. 4, 8–11

PSNR Peak Signal-to-Noise Ratio.

SNR Signal-to-Noise Ratio.

UA Universidade de Aveiro.

VS Code Visual Studio Code.

WAV Waveform Audio File Format.