Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Informação e Codificação

Projeto 2

Rafael Amorim (98197), rafael.amorim@ua.pt

Diogo Fontes (98403), diogo.fontes@ua.pt

4 de dezembro de 2022

Conteúdo

Introdução

Este relatório tem como objetivo descrever a resolução do Projeto 2 no âmbito da unidade curricular de Informação e Codificação.

O código desenvolvido para o Projeto 2 encontra-se disponível em:

https://github.com/Raf4morim/IC

Para a realização deste projeto recorreu-se à biblioteca:

- opencv

As indicações para a compilação dos programas encontram-se disponíveis no README.md do repositório.

Parte 1

Exercício 1

Neste exercício, pede-se a **cópia do conteúdo** de uma imagem já existente (sendo para isso dada uma pasta com um conjunto de imagens), pixel a pixel, para outro ficheiro novo.

Com esse propósito foram usadas as ferramentas disponíveis pelo **OpenCV**, instalado previamente.

Sabendo que cada pixel está numa posição de uma matriz bidimensional, daí ter de se percorrer todas as linhas e colunas através de ciclos **for**, por sua vez, guarda-se o valor da cor em RGB (valores variam entre 0 e 255 em cada um dos 3 bytes de entrada) na respetiva posição e finalmente escreve-se no ficheiro de saída.

Deste raciocínio surgiu o seguinte código:

```
Mat copia(imgInput.rows, imgInput.cols, imgInput.type()); // Criar uma imagem com as mesmas dimensões da imagem de entrada

for(int i=0; i < imgInput.rows; i++){
    for(int j=0 ; j < imgInput.cols; j++)
        copia.ptr<Vec3b>(i)[j] = Vec3b(imgInput.ptr<Vec3b>(i)[j][0], imgInput.ptr<Vec3b>(i)[j][1], imgInput.ptr<Vec3b>(i)[j][2]);
}

imwrite(oFile ,copia);        You, 2 weeks ago • 1 feito
```

Figura 1: Excerto código exercício1

Resultado:

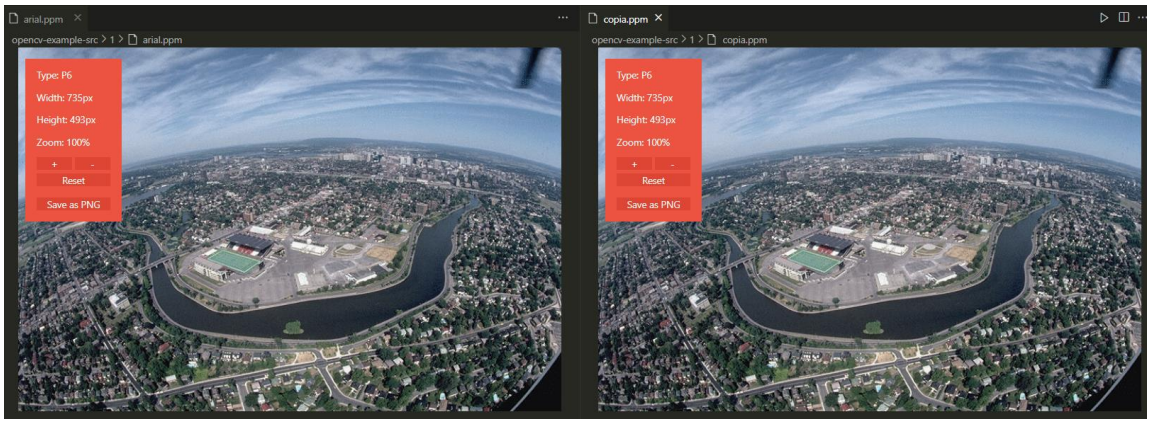

Figura 2: Resultado exercício 1

Exercício 2

Neste exercício, criou-se um **menu** com várias opções tendo o propósito de satisfazer todos os efeitos pedidos, sendo que cada opção corresponde a um efeito dando ainda a hipótese de ser mais pormenorizado tal como na rotação da imagem, escolher o sentido e no espelho sobre que eixo.

Neste programa apenas é necessário um ficheiro de entrada sendo que o de saída vai ser gerado automaticamente a partido do código e daí pode-se comparar.

Opção A

O Efeito **Negativo** de uma imagem é efetuado através da subtração entre uma matriz de zeros e uma matriz com todos os elementos iguais a 255 correspondendo ao máximo de intensidade possível a reduzir.

Código:

```
case 1:
    cout << "Negative Version\n";

    // inicializar a matriz de saída com zeros
    dst = Mat::zeros(src.size(), src.type());
    // cria uma matriz com todos os elementos iguais a 255 para subtrair
    sub_mat = Mat::ones(src.size(), src.type()) * 255;
    // subtraia a matriz original pela sub_mat para dar a saída negativa
    subtract(sub_mat, src, dst);
    // write the output image
    imwrite("./2/imgNegative.ppm", dst);
    break;
```

Figura 3: Excerto código exercício2a

Resultado:

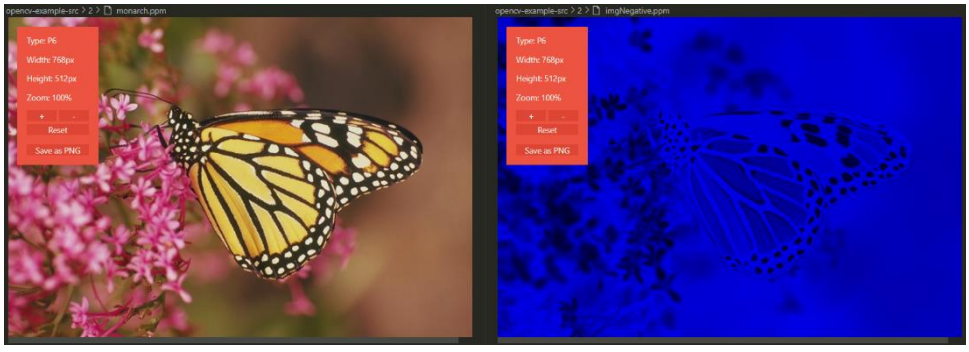

Figura 4: Resultado exercício 2a

Opção B

O efeito de **Espelho** onde se pode escolher o eixo horizontal ou vertical de uma imagem. Obteve-se através da função **flip** fornecida pelo OpenCV.

Esta função funciona da seguinte forma:

- flip(src, dst, 0); // eixo xx
- flip(src, dst, 1); // eixo yy

Código:

```
case 2:
    cout << "Mirrored Version\n";
    char direction;
    if (argc != 2)
        throw "Error: Usage sintax is ../opencv-example-bin/menu <input img>";
    cout << "Choose the diretion to do a mirrored version of image (horizontally or vertically): \n";
    cin >> direction;
    if (direction == 'h' || direction == 'H'){
        flip(src, dst, 0); // eixo xx
        imwrite("./2/imgHorizontal.ppm", dst);
    }else if (direction == 'v' || direction == 'V'){
        flip(src, dst, 1); // eixo yy
        imwrite("./2/imgVertical.ppm", dst);       You, 5 days ago • 2 completo
    }else{
        cout << "Invalid option!\n";
    }
    break;
```

Figura 5: Excerto código exercício 2b

Resultado:

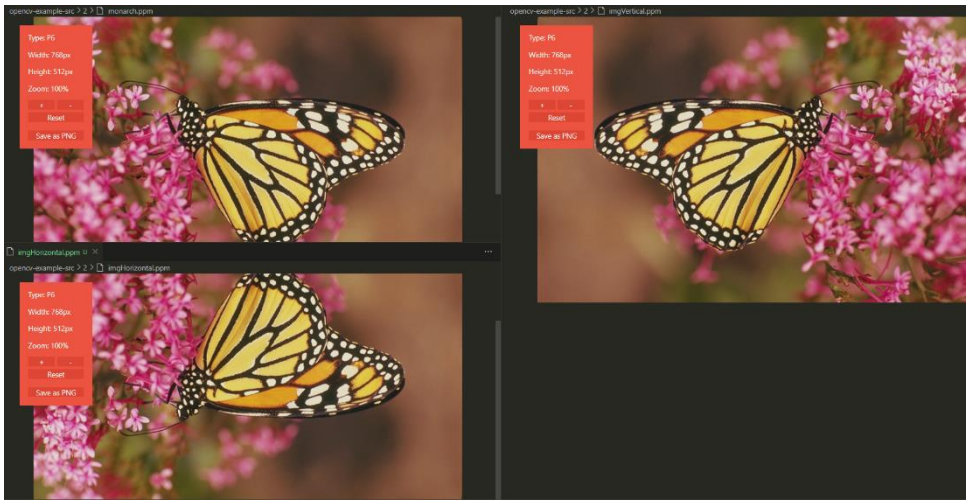

Figura 6: Resultado exercício 2b

Opção C

Para o efeito de **Rotação** foi feita uma pequena classe à parte chamada **rotate()** que usufrui das funções **getRotationMatrix2D()** para efetuar as rotações e da **warpAffine()** para aplicar a transformação.

Finalmente no código principal insere-se a imagem origem e o ângulo pretendido.

Código:

```
cout << "Choose the side to Rotate an image by a multiple of 90º (left or right): \n";
cin >> side;
if (side == 'l' || side == 'L'){
    dst = rotate(src, 90); // rotating image with 90 degree angle      You, 5 days ago
    imwrite("./2/rotateLeft.ppm", dst);
}else if (side == 'r' || side == 'R'){
    dst = rotate(src, -90); // rotating image with 30 degree angle
    imwrite("./2/rotateRight.ppm", dst);
}else{
    cout << "Invalid option!\n";
}
break;
```

Figura 7: Excerto código exercício 2c

Resultado:

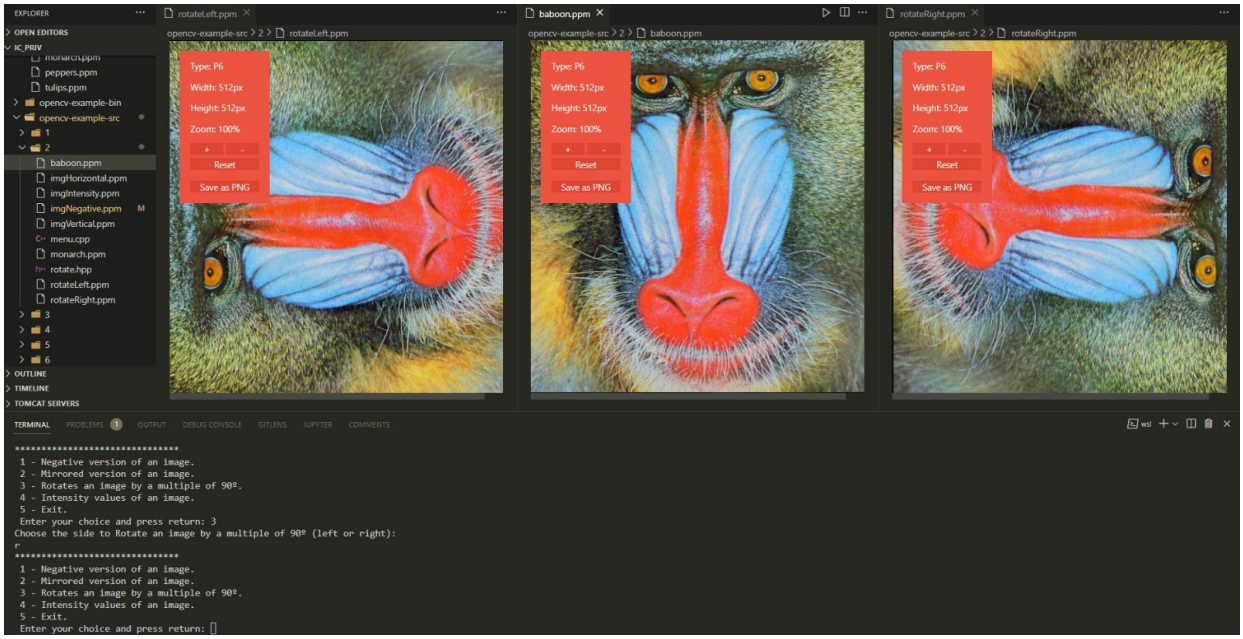

Figura 8: Resultado exercício 2c

Opção D

Para o efeito de **Intensidade** fez-se uso da função **saturate_cast()** usando um raciocínio idêntico ao exercício 1, onde se percorre uma matriz correspondentes às linhas e colunas e para cada canal, calculam-se as cores juntamente com os alcances inseridos pelo utilizador.

Código:

```
case 4:
    cout << "Intensity values of an image.\n";

    dst = Mat::zeros(src.size(), src.type());

    cout << "* Enter the alpha value [1.0-3.0] (more light) and [-3;-1]: (Less light): ";
    cin >> alpha;
    cout << "* Enter the beta value [0-100]: ";
    cin >> beta;
    for (int y = 0; y < src.rows; y++){
        for (int x = 0; x < src.cols; x++){
            for (int c = 0; c < src.channels(); c++){
                dst.at<Vec3b>(y, x)[c] = saturate_cast<uchar>(alpha * src.at<Vec3b>(y, x)[c] + beta);
            }
        }
    }
    imwrite("./2/imgIntensity.ppm", dst);
    break;
```

Figura 9: Excerto código exercício 2d

Resultados:

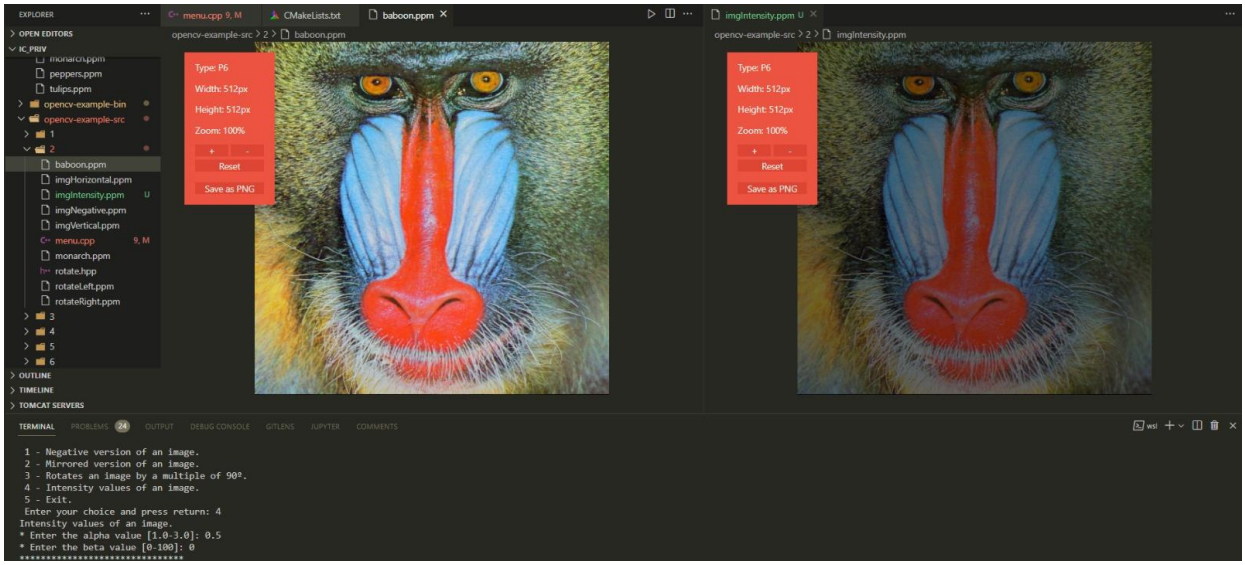

Figura 10: Resultado exercício 2d less light

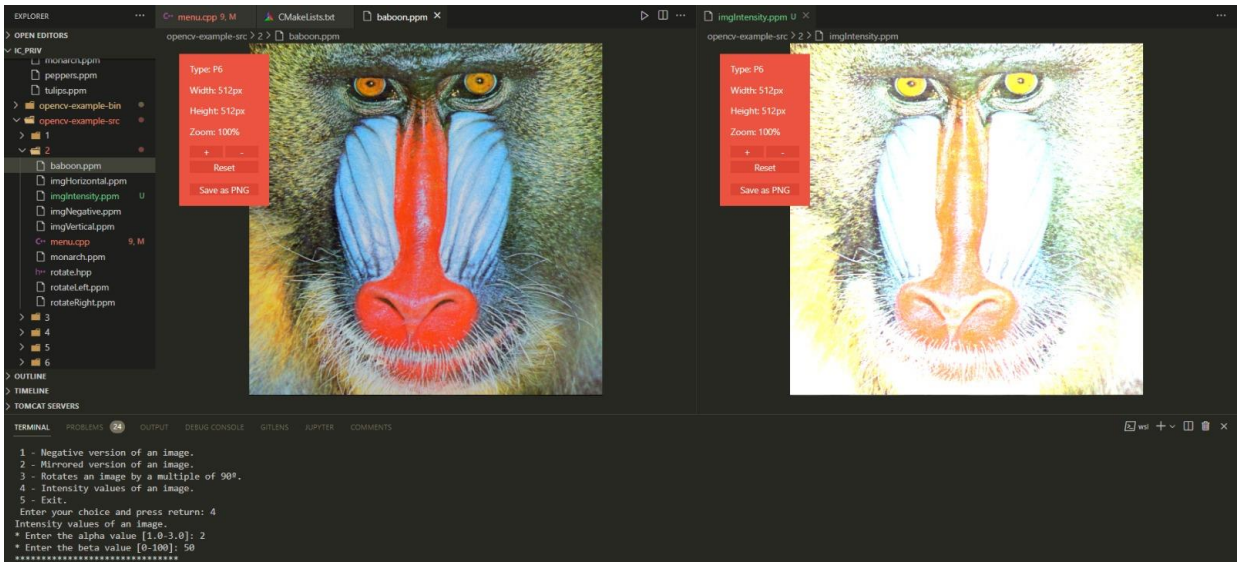

Figura 11: Resultado exercício 2d more light

Parte 2

Exercício 3

Para este exercício, foi necessário recorrer ao *BitStream* desenvolvido no último projeto onde são escritas e lidas codificações e descodificações, implementou-se a codificação de **Golomb** que permite **separar um inteiro 'n' em 2 partes**, **unária '*q*' e binária 'r'**, e também gera um conjunto de códigos de tamanho variável livres de prefixo.
A partir daqui conseguimos obter resultados ideais, onde existe a dependência do parâmetro *m>0*.

- $m = \left\lceil \frac{-1}{log(x)} \right\rceil$;
- $q = \left\lfloor \frac{n}{m} \right\rfloor$; $quociente\ pode\ ser\ 0,1,2,..$
- $r = n - qm$; $resto\ será\ códificado\ pelo\ código\ em\ binário\ que\ pertence\ a\ [0;\ m-1]$

Estas verificam-se para quando o valor de m é uma potência de 2. Caso contrário, o processo anterior não é válido e o procedesse com as seguintes expressões:

- $b = \lceil \log_2 m \rceil$;
- Caso $r < 2^b - m$ codifica-se r, em binário usando b-1 bits.
- Caso contrário, codifica-se com b bits $r + 2^b - m$

```
if (m!=0 && (m & (m-1)) == 0){      //Caso m seja potencia de 2
    value = r;
    numbits = b;
}
else{
    if (r < pow(2, b) - m){         //Caso m nao seja potencia de 2
        value =  r;
        numbits = b-1;
    }
    else{
        value = r + pow(2, b) - m;
        numbits = b;
    }
}
```

Figura 12: Excerto código exercício 3 m

A descodificação consiste na leitura dos valores escritos num ficheiro, onde **contava os 1** (U), para que desta forma, fosse possível determinar o tamanho do código:

- caso fosse uma potência de 2, que era $A + b + 1$ e, desta forma, o resto seria $c + 1$ bits. Ao converter este valor para decimal obtínhamos o resultado $mA + R$.

- caso não fosse potência de 2, calcula-se em decimal pelos $c - 1$ bits seguintes ao código unário. Para $r < 2^c - m$, temos $mA + R$, senão temos de considerar que os c bits seguintes ao código unário são R e o valor descodificado surge de $mA + R - (2^c - m)$.Para $r < 2^c - m$, temos $mA + R$, senão temos de considerar que os c bits seguintes ao código unário são R e o valor descodificado surge de $mA + R - (2^c - m)$.

Usamos as funções fold e unfold para representar valores negativos, ou seja, os números positivos são pares e números negativos são ímpares.

```
int Golomb::fold(int n){
    if (n >= 0)
        return n*2;
    else
        return abs(n)*2-1;
}

int Golomb::unfold(int n){
    if (n % 2 == 0)
        return n/2;
    return (-1)*ceil(n/2)-1;
}
```

Figura 13: Excerto código exercício 3 fold e unfold

```
if (m!=0 && (m & (m-1)) == 0){
    char binary[b];
    file.readNbits(binary,b);
    int tmp = 0;
    for( int i = b-1; i >= 0; i--){
        if(binary[i] != 0x0)
            R+= pow(2, tmp);
        tmp++;
    }

    return unfold(m*A + R);          //Calculo do valor decoded
}else{
    int tmp = 0;
    char binary[b];
    file.readNbits(binary,b-1);
    binary[b-1] = 0;

    for (int i = b-2; i >= 0; i--){ //Extrair b-1 MSBs e calcula R em decimal
        if(binary[i] != 0x00)
            R+= pow(2, tmp);
        tmp++;
    }

    if(R < pow(2, b) - m){
        return unfold(m*A + R);
    }else{                           //Extrair b MSBs e calcula R em decimal
        binary[b-1] = file.readBit();
        R=0, tmp=0;
        for (int i = b-1; i >= 0; i--){
            if(binary[i] != 0x0)
                R+= pow(2, tmp);
            tmp++;
        }
        return unfold(m*A + R - (pow(2, b) - m));
    }
}
```

Figura 14: Excerto código exercício 3 decode

Por fim, foi preciso complementar com umas funções auxiliares e pode-se testar a classe através do ***testGolomb.cpp***.

Parte 3

Exercício 4

Preditores

Preditor linear usado para o cálculo dos residuais:

- $(\hat{x}\_n)^0 = 0$
- $(\hat{x}\_n)^1 = x_{n-1}$
- $(\hat{x}\_n)^2 = 2x_{n-1} - x_{n-2}$
- $(\hat{x}\_n)^3 = 3x_{n-1} - 3x_{n-2} - x_{n-3}$

Calcular valor dos residuais:

- $r\_n = x\_n - \hat{x}\_n$

São necessárias operações inversas na reconstrução dos valores obtidos pelos preditores:

- $x\_n = r\_n - \hat{x}\_n$

Para obter uma entropia mais baixa, decidimos fazer o processamento canal a canal, ou seja, o preditor está dividido em dois, para fazer previsões só para o canal esquerdo, tendo em conta a informação deste canal e previsões para o canal direito, tendo em conta as informações do canal direito.

Todavia, as residuais resultantes da previsão dos canais são colocadas no mesmo vetor.

Lossless Codec

Para a implementação deste codec era necessário recolher os dados necessários para ser possível a reconstrução do ficheiro original no lado do descodificador. Neste sentido, foi criada uma função adicional na classe Golomb que permite a codificação do cabeçalho que vai possuir informação importante na reconstrução do ficheiro original.

Informação contida num cabeçalho:

Golomb m	32 bits
Nº de Samples	32 bits
Sample Rate	32 bits
Format	32 bits
Channels	4 bits

Tabela 1: Cabeçalho Golomb

O valor de cada sample é enviado para o preditor que vai calcular um valor usado para obter os residuais. Após receber os **residuais falta apenas calcular o m ideal** para codificar os residuais com códigos de Golomb.

- $mean = \frac{\sum fold(r_n)}{N}$

- $m = \frac{-1}{\log_2\left(\frac{mean}{mean|1.0|}\right)}$

Após o cálculo do 'm', os cabeçalhos são criados e escritos no ficheiro.

Para escrever os valores dos residuais codificados com códigos de Golomb no ficheiro destino são feitas chamadas sucessivas á função encode da classe Golomb.

Podemos, ainda, realizar o processo de descodificação, contrário ao explicado anteriormente. Este processo inicia com a leitura dos cabeçalhos com as funções desenvolvidas na classe Golomb. Ao ler o cabeçalho obtemos não só o m, mas também outros dados que nos permitem reconstruir o ficheiro original e saber quantos valores ainda faltam ler.

Para ler os valores dos residuais calculados são feitas chamadas sucessivas da função **decode**() da classe Golomb.

Exercício 5

Lossy Codec

Em relação ao exercício anterior só foi feita uma alteração que foi a adição de um processo de quantização aos residuais calculados com o preditor. Para isso foi também necessário alterar o preditor deste codec, uma vez que o cálculo dos residuais tem de refletir esta quantização.

A quantização é feita com um shift á direita do número de bits inserido pelo utilizador.

A descodificação é semelhante á efetuada no codec lossless, a diferença está presente na necessidade de reconstruir a quantização que foi efetuada. Essa reconstrução é feita efetuando um shift na direção oposta do mesmo número de vezes que foi realizada na codificação.

Parte 4

Exercício 6

Preditor

O preditor usado para este codec foi não linear, são usados 3 pixéis anteriores (a, b e c) para prever o pixel atual.

$$\min(a,b)\ if\ c\ \geq \max(a,b)$$

$$\max(a,b)\ if\ c\ \leq\ \min(a,b)$$

$$a+b-c\ otherwise$$

Cada preditor destes é usado 3 vezes por pixel, um por cada cor (RGB). Depois dos valores dos pixéis serem calculados, calculam-se os resíduos:

$$r_n = x_n - \hat{X}\_n$$

Decoder

O Decoder lê do ficheiro de imagem, descodifica os 3 primeiros valores, usando o decoder de Golomb, e cria uma matriz com os seguintes parâmetros:

- Número de linhas
- Número de colunas
- Tipo de imagem.

Para descodificar os valores de cada pixel, usam-se 2 ciclos **for** para percorrer as linhas e colunas da nova matriz. Para o **primeiro** pixel usa-se os valores dos resíduos. Para os pixéis da primeira **linha** usam-se os valores do pixel anterior (da mesma linha) somado aos resíduos. Para os pixéis da primeira **coluna** usam-se os valores do pixel anterior (da mesma coluna) somando aos resíduos. Para os restantes, usa-se um preditor similar ao do encoder.

Os valores finais dos pixéis são a soma dos valores previstos com os resíduos.

Resultados:

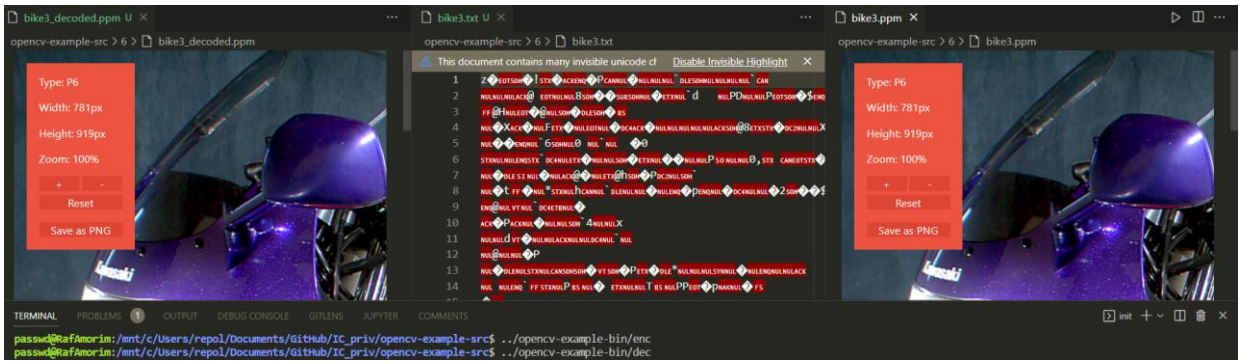

Figura 15: Resultado exercício 6

Contribuição dos Autores

Rafael Amorim – 50 %

Diogo Fontes – 50 %