

Processamento Digital de Imagens

Table of Contents

| | |
|---|---|
| 1 Manipulando pixels em uma imagem | 1 |
| 1.1 Exercício 1.1 - Filtro Negativo (regions.cpp) | 1 |
| 1.2 Exercício 1.2 - Troca Regiões (trocaregiones.cpp) | 3 |
| 2. Serialização de dados em ponto flutuante via FileStorage | 5 |
| 2.1. Exercício 2 - filestorage.cpp | 5 |
| 3. Decomposição de imagens em planos de bits | 7 |
| 3.1. Exercício 3 - esteg-encode.cpp | 7 |
| 4. Preenchendo regiões | 7 |
| 4.1. Exercício 4 - labeling.cpp | 7 |
| 5. Manipulação de histogramas | 7 |
| 5.1. Exercício 5 - histogram.cpp | 7 |
| 6. Filtragem no domínio espacial I | 8 |
| 6.1. Exercício 6 - filtroespacial.cpp | 8 |
| 7. Filtragem no domínio espacial II | 8 |
| 7.1. Exercício 7 - addweighted.cpp | 8 |
| 8. A Transformada Discreta de Fourier | 8 |
| 8.1. Exercício 8 - dftimage.cpp | 8 |

1 Manipulando pixels em uma imagem

Manipular pixels em uma imagem envolve a capacidade de alterar individualmente os elementos de cor que compõem a imagem. Cada pixel contém informações sobre sua cor específica, como vermelho, verde e azul (RGB), além de valores de transparência em alguns casos. Ao manipular os pixels, é possível realizar uma variedade de transformações na imagem, como ajustar o brilho, a saturação, o contraste, aplicar filtros, redimensionar ou recortar. Essas manipulações permitem corrigir imperfeições, realçar detalhes, criar efeitos especiais, entre outras possibilidades. A manipulação de pixels é uma técnica fundamental em áreas como processamento de imagem, design gráfico, edição de fotos e criação de arte digital. Com a ajuda de bibliotecas de processamento de imagem, é possível acessar e modificar os valores dos pixels em uma imagem, abrindo caminho para inúmeras possibilidades criativas e práticas.

1.1 Exercício 1.1 - Filtro Negativo (regions.cpp)

O filtro negativo é um mecanismo que busca eliminar conteúdos indesejados, inapropriados ou prejudiciais por meio da aplicação de técnicas de análise e restrição. Ele é amplamente utilizado em diversas áreas, como tecnologia da informação, mídias sociais e segurança, visando proteger os usuários e garantir a qualidade do conteúdo. No entanto, é necessário ter cautela para evitar restrições excessivas e preservar a liberdade de expressão.

Com isso, crie um programa chamado `regions` com base no código de exemplo "pixels.cpp". O programa deve pedir ao usuário as coordenadas de dois pontos, P1 e P2, que estão dentro dos limites do tamanho da imagem. Em seguida, ele deve exibir a imagem fornecida, mas com a região definida pelo retângulo formado pelos pontos P1 e P2 exibindo o negativo da imagem nessa área específica. Esse efeito produzirá uma região destacada com cores invertidas em relação ao restante da imagem.

1.1.1 Código e Resultado.

regions.cpp

```
// Renato Emanuel Medeiros de Lira
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main(int, char** argv) {

    Mat image;
    image=imread(argv[1], IMREAD_GRAYSCALE);

    if (image.empty()) {
        cout << "Imagem não foi carregada" << endl;
        return 1;
    }

    Point p1, p2;

    cout << "Tamanho da imagem: " << image.rows << "x" << image.cols << endl;

    cout << "Digite as coordenadas do ponto P1:" << endl;
    cin >> p1.y >> p1.x;

    do{
        cout << "Digite as coordenadas do ponto P2:" << endl;
        cin >> p2.y >> p2.x;
        if (p1.y >= p2.y || p1.x >= p2.x){
            cout << "Ambas coordenadas do ponto P2 tem que ser maiores que as coordenadas do ponto P1, escreva novamente!"<<endl;
        }
    }while(p1.y >= p2.y || p1.x >= p2.x);

    for (int i = p1.x; i < p2.x; i++) {
        for (int j = p1.y; j < p2.y; j++) {
            image.at<uchar>(i, j) = 255 - image.at<uchar>(i, j);
        }
    }
}
```

```
namedWindow("janela", WINDOW_AUTOSIZE);  
imshow("janela", image);  
waitKey(0);  
  
imwrite("Regions.png", image);  
  
return 0;  
}
```

Abaixo temos a imagem original [biel](#) e em seguida a imagem resultante do código [regions](#). As Coordenadas escolhidas foram P1 (90, 180) e P2 (90, 180).



Figure 1. Imagem original



Figure 2. Saída do programa regions.cpp

1.2 Exercício 1.2 - Troca Regiões (trocaregiones.cpp)

Desenvolva um programa chamado [\[trocaregiones.cpp\]](#) com base no código de exemplo "pixels.cpp". O programa deve trocar os quadrantes diagonais na imagem. Para realizar essa troca, utilize a classe "Mat" e seus construtores para criar as regiões que serão intercambiadas. Ao executar o programa, a imagem exibida terá os quadrantes diagonais trocados.

1.1.2 Código e Resultado.

trocaregiones.cpp

```
// Renato Emanuel Medeiros de Lira
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

int main(int, char** argv) {

    Mat image;
    image=imread(argv[1], IMREAD_GRAYSCALE);

    if (image.empty()) {
        cout << "Imagem não foi carregada" << endl;
        return 1;
    }

    int x = image.rows;
    int y = image.cols;

    Mat temp = image.clone();

    Rect region1(0, 0, y/2, x/2);
    Rect region2(y/2, 0, y/2, x/2);
    Rect region3(0, x/2, y/2, x/2);
    Rect region4(y/2, x/2, y/2, x/2);

    Mat roi1 = temp(region1);
    Mat roi2 = temp(region2);
    Mat roi3 = temp(region3);
    Mat roi4 = temp(region4);

    roi3.copyTo(image(region2)); // R3 -> R2
    roi4.copyTo(image(region1)); // R4 -> R1
    roi1.copyTo(image(region4)); // R1 -> R4
    roi2.copyTo(image(region3)); // R2 -> R3

    imshow("janela", image);
    waitKey(0);

    imwrite("trocaregiones.png", image);

    return 0;
}
```

Abaixo temos a imagem original [biel](#) e em seguida a imagem resultante do código [trocaregiones](#), na

qual os quadrantes foram trocados de lugar.



Figure 3. Saída do programa `trocaregiones.cpp`

2. Serialização de dados em ponto flutuante via FileStorage

A serialização de dados em ponto flutuante usando o FileStorage envolve a gravação e leitura de valores numéricos em formato de ponto flutuante em um arquivo estruturado. Ao utilizar a classe FileStorage, é possível gravar os dados em um formato como YAML ou XML, permitindo a preservação da precisão fracionária. A leitura dos dados serializados também é facilitada pela classe FileStorage, tornando possível acessar e utilizar os valores gravados posteriormente. Esse processo é especialmente útil em cenários onde é necessário armazenar e recuperar informações numéricas precisas, como em aplicações científicas, processamento de imagens e simulações.

2.1. Exercício 2 - `filestorage.cpp`

Utilizando o programa "filestorage.cpp" como base, crie um novo programa que gere uma imagem de 256x256 pixels contendo uma senoide horizontal com 4 períodos e amplitude de 127, semelhante à figura apresentada na [\[filestorage_exemplo\]](#). Grave a imagem em formato PNG e em formato YML. Em seguida, compare os arquivos gerados, extraíndo uma linha de cada imagem gravada e calculando a diferença entre elas. Trace um gráfico dessa diferença ao longo da linha correspondente nas imagens. Observe os resultados e faça as observações relevantes.

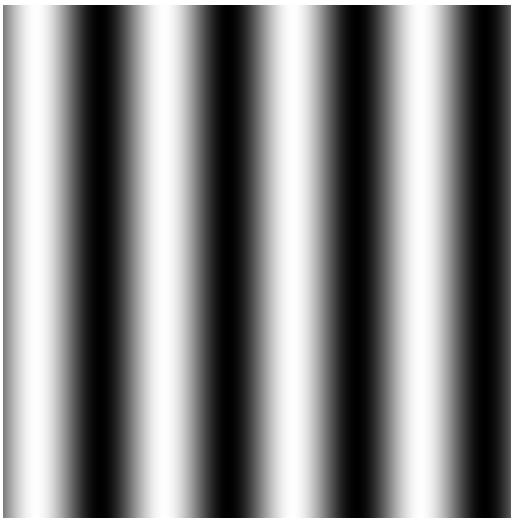


Figure 4. Imagem de senóide

2.1.1. Código e Resultado.

filestorage.cpp

```
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main() {
    int side = 256;
    int periodos = 4;

    Mat image(side, side, CV_8UC1);

    for (int i = 0; i < side; i++) {
        for (int j = 0; j < side; j++) {
            float value = 127 * sin(2 * CV_PI * periodos * j / side) + 128;
            image.at<uchar>(i, j) = static_cast<uchar>(value);
        }
    }

    string filenameYML = "senoide.yml";
    string filenamePNG = "senoide.png";

    FileStorage fs(filenameYML, FileStorage::WRITE);
    fs << "mat" << image;
    fs.release();

    imwrite(filenamePNG, image);

    FileStorage fs_read(filenameYML, FileStorage::READ);
    fs_read["mat"] >> image;
    fs_read.release();
}
```

```
imshow("image", image);  
waitKey();  
  
return 0;  
}
```

Abaixo temos a imagem resultante do código [filestorage](#), na qual é representado uma senoide horizontal com 4 periodos e amplitude de 127. A imagem é semelhante a figura [\[filestorage_exemplo\]](#) usada como exemplo.

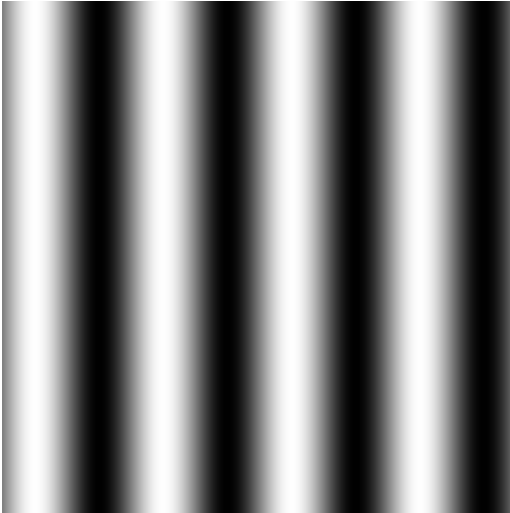


Figure 5. Saída do programa `filestorage.cpp`

3. Decomposição de imagens em planos de bits

3.1. Exercício 3 - `esteg-encode.cpp`

3.1.1. Código e Resultado.

4. Preenchendo regiões

4.1. Exercício 4 - `labeling.cpp`

4.1.1. Código e Resultado.

5. Manipulação de histogramas

5.1. Exercício 5 - `histogram.cpp`

5.1.1. Código e Resultado.

6. Filtragem no domínio espacial I

6.1. Exercício 6 - filtroespacial.cpp

6.1.1. Código e Resultado.

7. Filtragem no domínio espacial II

7.1. Exercício 7 - addweighted.cpp

7.1.1. Código e Resultado.

8. A Transformada Discreta de Fourier

8.1. Exercício 8 - dftimage.cpp

8.1.1. Código e Resultado.