PROCESSAMENTO DIGITAL DE IMAGENS.
Aluno(a): Matheus Augusto Monteiro Da Silva.

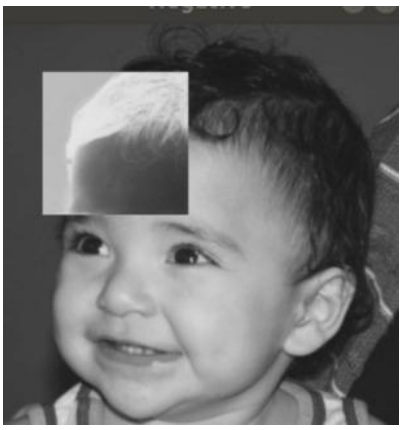
PRIMEIRA UNIDADE
Manipulação pixel em uma imagem.

PRIMEIRA UNIDADE.
1.1 EXERCÍCIO
Solução:

Para solucionar o problema, foi necessário introduzir opções para o usuário selecionar qual região ele desejava converter para negativo. Para realizar essa conversão, foi utilizado um loop "for" para percorrer a área selecionada e aplicar a seguinte operação: "image.at<uchar>(i,j) = 255 - image.at<uchar>(i,j)". Essa operação permite que o pixel da imagem seja transformado em seu negativo.

original:

negativo:



```
#include <iostream>
```

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

using namespace cv;
using namespace std;

int main(int, char** argv){
  Mat image;
  int x1,y1,x2,y2;

  namedWindow("Original",WINDOW_AUTOSIZE);
  image = imread(argv[1],IMREAD_GRAYSCALE);

  if(!image.data){
    cout << "nao abriu bolhas.png" << std::endl;
  }
  imshow("Original", image);
  waitKey();

  cout << "Ditite as coordenadas do primeiro ponto: "<<endl;
  cin >> x1 >> y1;
  cout << "Ditite as coordenadas do primeiro ponto: "<<endl;
  cin >> x2 >> y2;

  //Verifica se as coordenadas são válidas
  if(x1!=x2 && y1!=y2 &&x1>=0 && x2<=image.rows && y1>=0 && y2<=image.rows){
    //converte a região selecionada para negativo
    for(int i=x1;i<x2;i++){
      for(int j=y1;j<y2;j++){
        image.at<uchar>(i,j)=255 - pixel);
      }
    }

    namedWindow("Negativo", WINDOW_AUTOSIZE);
    imshow("Negativo", image);
    waitKey();
  }else{
    cout<<"coordenadas inválidas"<<endl;
  }
  return 0;
}
```

EXERCÍCIO 1.2

Solução

Neste caso, em vez de lidar com regiões selecionadas pelo usuário, o objetivo é realizar manipulações na imagem de forma programática. Para isso, a classe Mat e as funções Rect foram utilizadas para recortar partes da imagem e armazená-las em uma matriz temporária. Em seguida, o método copyTo foi empregado para modificar a imagem original, permitindo assim a alteração das áreas desejadas. Dessa forma, o processo foi realizado sem a necessidade de interação direta com o usuário.

Original:

regiões trocadas:

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

using namespace cv;
using namespace std;

int main(int, char** argv){
  Mat image;

  namedWindow("Original",WINDOW_AUTOSIZE);
  image = imread(argv[1],IMREAD_GRAYSCALE);
```

```
  if(!image.data){
    cout << "nao abriu bolhas.png" << std::endl;
  }
  imshow("Original", image);
  waitKey();

  //recorte superior esquerdo da imagem
  Mat rec1 = image(Rect(0,0,image.rows/2,image.cols/2));
  //recorte superior direito da imagem
  Mat rec2 = image(Rect(image.rows/2,0,image.rows/2,image.cols/2));
  // recorte inferior esquerdo
  Mat rec3 = image(Rect(0,image.cols/2,image.rows/2,image.cols/2));
  // recorte inferior direito
  Mat rec4 = image(Rect(image.rows/2,image.cols/2,image.rows/2,image.cols/2));

  Mat troca(image.rows,image.cols,image.type());

  // trocando inferior direito para o superior esquerdo
  rec4.copyTo(troca(Rect(0,0,image.rows/2,image.cols/2)));
  // trocando inferior esquerdo para o superior direito
  rec3.copyTo(troca(Rect(image.rows/2,0,image.rows/2,image.cols/2)));
  // trocando superior direito para inferior direito
  rec2.copyTo(troca(Rect(0,image.cols/2,image.rows/2,image.cols/2)));
  // trocando superior esquerdo para inferior direito
  rec1.copyTo(troca(Rect(image.rows/2,image.cols/2,image.rows/2,image.cols/2)));

  namedWindow("regioes trocadas", WINDOW_AUTOSIZE);
  imshow("regioes trocadas", troca);
  waitKey();



  return 0;
}
```

PREENCHENDO REGIÕES

EXERCÍCIO 2.1

Solução:

Para solucionar a situação em que a imagem possui mais de 255 objetos a serem rotulados, é possível adotar uma estratégia alternativa. Nesse caso, a

abordagem consiste em atribuir rótulos decimais aos objetos ou utilizar a operação de módulo 255 durante o processo de rotulagem.

EXERCÍCIO 2.2
Solução:

Para remover as bolhas e buracos presentes nas bordas da imagem, é possível adotar uma abordagem que segue a mesma lógica. Primeiramente, é necessário percorrer as bordas da imagem em busca de pixels que correspondam à cor dos objetos. Ao encontrar um pixel desse tipo, pode-se utilizar esse ponto como uma semente para aplicar o algoritmo "floodFill". Dessa forma, as bolhas assumirão a mesma tonalidade do fundo da imagem.

No caso de contar os buracos, é possível utilizar uma estratégia em que o fundo da imagem seja preenchido com uma cor cinza, utilizando o algoritmo "floodFill". Isso fará com que o interior dos buracos ainda mantenha a cor do fundo original. Em seguida, é possível contar a quantidade de buracos existentes.

Sabendo a quantidade de buracos, é possível aplicar novamente o algoritmo "floodFill" na imagem e verificar quantos objetos foram encontrados. Ao subtrair o número de buracos desta contagem, obtemos a quantidade de bolhas e buracos restantes.

Manipulação de Histogramas

Exercício 3.1

Solução:

A fim de simular uma imagem em tons de cinza, é necessário converter a imagem para essa representação utilizando a função "cvtColor". Em seguida, para equalizar o histograma, utiliza a função "equalizeHist". Posteriormente, será gerado o histograma tanto da imagem original quanto da imagem equalizada, permitindo assim a comparação entre as duas.

Exercício 3.2

Solução:

A solução para esse exercício envolveu a criação de um histograma que armazena o último histograma do frame anterior e o comparasse com o histograma mais recente. Essa comparação foi realizada por meio da função "compareHist", que calcula a correlação entre os histogramas. Com base nessa correlação, foi possível criar uma estrutura condicional (if) para verificar se a correlação era alta ou baixa.

Caso a correlação fosse alta, foi exibido um alerta em verde no canto inferior da tela, indicando a detecção de movimento.

Exercício 4

Solução:

Para resolver essa questão, a solução consiste em adicionar a máscara do filtro Laplaciano do Gaussiano juntamente com as máscaras dos outros filtros existentes. Além disso, é necessário implementar a opção de escolha, permitindo ao usuário digitar uma tecla específica para acionar essa funcionalidade.

Ao analisar o efeito do filtro Laplaciano do Gaussiano, vai ser emitido um realce nos contornos, resultando em linhas mais espessas e uma maior visibilidade dos contornos. O filtro Laplaciano é aplicado primeiro, seguido pelo filtro Laplaciano do Gaussiano para obter esse efeito desejado.