Trabalho - PDI

Nome: Pedro Klisley F. da Silva
Samuel Alves da Costa

Disciplina: Processamento Digital de Imagem

DCA
UFRN
Março/2016

Sumário

1 Resolução

3.1 Utilizando o programa exemplos/pixels.cpp como referência, implemente um programa regions.cpp. Esse programa deverá solicitar ao usuário as coordenadas de dois pontos P1P1 e P2P2 localizados dentro dos limites do tamanho da imagem e exibir que lhe for fornecida. Entretanto, a região definida pelo retângulo de vértices opostos definidos pelos pontos P1P1 e P2P2 será exibida com o negativo da imagem na região correspondente.

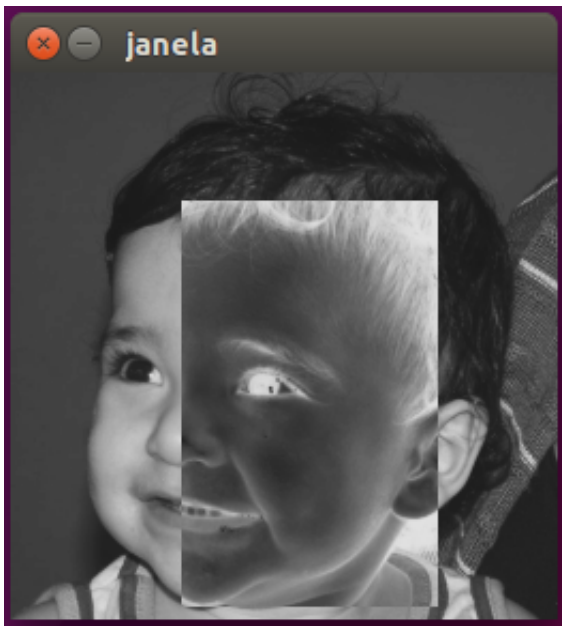
Nesse exercício, foi usada as funções "image.rows" e "image.cols" para fornecer respectivamente o numero de linhas e de colunas (dimensão da matriz) da figura a ser criado o efeito do negativo.

Conhecendo esses valore, é solicitado ao usuário que entre com dois pontos dentro do intervalo do tamanho da matrix. Ao ler os pontos fornecidos pelo usuário, o efeito do negativo é mostrado na imagem, conforme ilustra a Figura 3.1.

«Efeito do Negativo»

Para essa imagem o usuário entrou com os valores

«Valores» «»Imagem

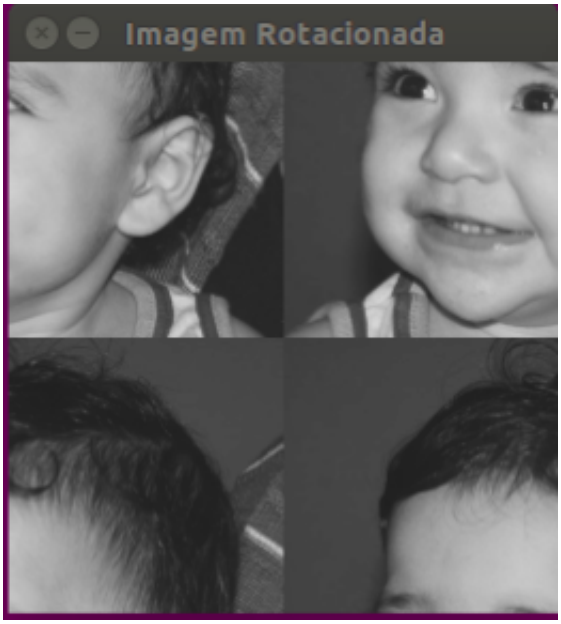



```
#include <iostream>
#include <cv.h>
#include <highgui.h>

using namespace cv;
using namespace std;

int main(int, char**){
  Mat image;
  Vec3b val;

 int linhas = image.rows;
```

```
 int colunas = image.cols;

 int x1=0, y1=0,x2=0,y2=0;

  image= imread("biel.png",CV_LOAD_IMAGE_GRAYSCALE);


if(!image.data)
    cout << "nao abriu biel.png" << endl;

namedWindow("janela",WINDOW_AUTOSIZE);



        cout << "digite a coordenada x1, valor entre 0 e "<< linhas << endl;
        cin >> x1;

        cout << "digite a coordenada y1, valor entre 0 e "<< colunas << endl;
        cin >> y1;

        cout << "digite a coordenada x2 valor entre 0 e "<< linhas<< endl;
        cin >> x2;

        cout << "digite a coordenada y2, valor entre 0 e "<< colunas << endl;
        cin >> y2;


  for( int i= x1;i< x2;i++){
    for(int j = y1;j< y2;j++){
      image.at<uchar>(i,j)= 255 - image.at<uchar>(i,j);
    }
  }


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


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

3.2 Utilizando o programa exemplos/pixels.cpp como referência, implemente um programa troca-regioes.cpp. Seu programa deverão trocar aleatoriamente regiões da imagem, formando uma espécie de quebra-cabeças. Explore o uso da classe Mat e seus construtores para criar as regiões que serão trocadas. O efeito é ilustrado na Figura Troca de regiões.

«»Imagem



5.1 Utilizando o programa exemplos/histogram.cpp como referência, implemente um programa equalize.cpp. Este deverá, para cada imagem capturada, realizar a equalização do histogram antes de exibir a imagem. Teste sua implementação apontando a câmera para ambientes com iluminações variadas e observando o efeito gerado.

Nesse exercício foi utilizada a função "equalizeHist", em C++ ela recebe os seguintes parametros: C++: void equalizeHist(InputArray src, OutputArray dst)

Parametros: src – Canal de origem . dst – Canal de destino.

Esse Função permitiu incrementar ao código original a funcionalidade de equalização.

A qualização do histograma é feita da seguinte forma:

1 - Calcula o histograma H por src.

2 - Normaliza o histograma de modo que a soma das barras seja 255.

3 - Calcula a integral do histograma usando a seguinte expressão:

«»Imagem

4- Transforma a imagem usando

Utilizando a camera e posicionando para diferentes niveis de iluminação, são vistos os resutados das Figura XX, Figura XX+1 e Figura XX+2:

Observa-se que na Figura XX, comparado o lado direito e esquerdo, tem-se no lado direito uma imagem com o contraste melhorado, isso se da porque essa função atua de modo a esticar a faixa de intensidade da figura original.

Essa mesma comparação entre a imagem original e a Equalizada pode ser vista nas figuras Figura XX+1 e Figura XX+2.

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

using namespace cv;
using namespace std;

Mat image1, image2, image3, image4;
Mat image, imageRot;

int width, height;
int width2, height2;

int main(int argvc, char** argv){
  image = imread("biel.png");
  image.copyTo(imageRot);
  imshow("Imagem Normal",image);

  //Pega tamanhos
  width = image.size().width;
  height = image.size().height;
  width2 = width/2;
  height2 = height/2;

  //Divide imagem original em 4 menores
  image1 = image(Rect(0, 0, width2, height2));
  image2 = image(Rect(width2, 0, width2, height2));
  image3 = image(Rect(0, height2, width2, height2));
  image4 = image(Rect(width2, height2, width2, height2));

  //Monta nova imagem
  image4.copyTo(imageRot(Rect(0, 0, width2, height2)));
  image3.copyTo(imageRot(Rect(width2, 0, width2, height2)));
  image2.copyTo(imageRot(Rect(0, height2, width2, height2)));
  image1.copyTo(imageRot(Rect(width2, height2, width2, height2)));
```

```
  imshow("Imagem Rotacionada",imageRot);
  waitKey(0);
  return 0;
}
```

5.2 Utilizando o programa exemplos/histogram.cpp como referência, implemente um programa motiondetector.cpp. Este deverá continuamente calcular o histograma da imagem (apenas uma componente de cor é suficiente) e compará-lo com o último histograma calculado. Quando a diferença entre estes ultrapassar um limiar pré-estabelecido, ative um alarme. Utilize uma função de comparação que julgar conveniente.

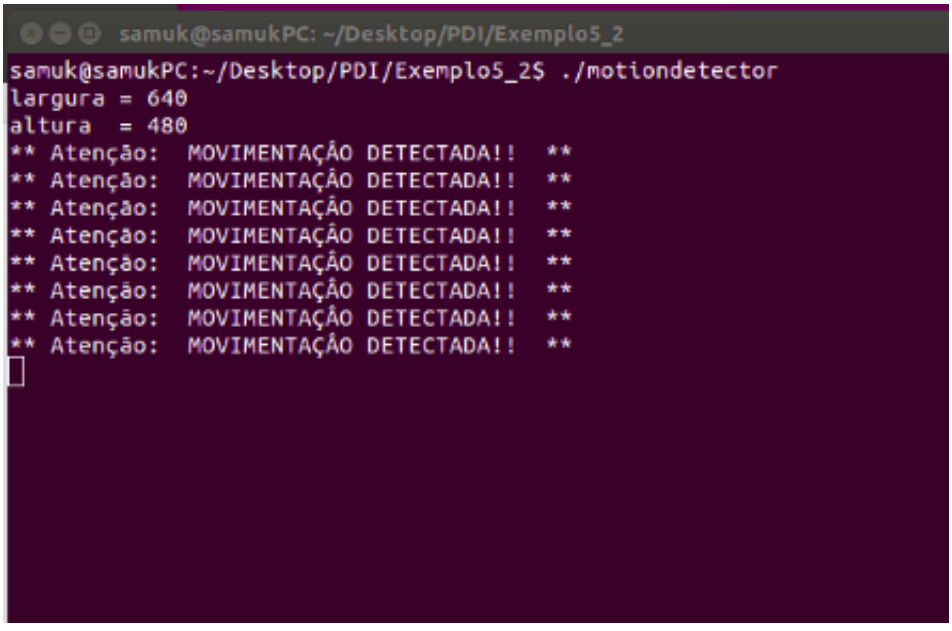
A análise foi feita utizilando a Componente R

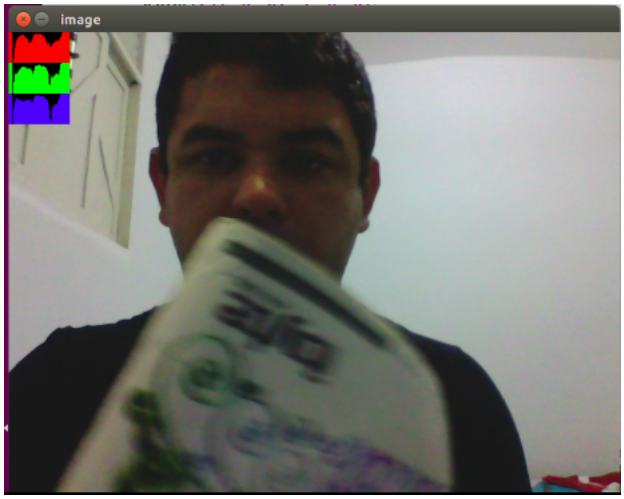
Nessa atividade é calculado o histograma atual da imagem na componente R, e comparado com a mesma componente do último histograma calculado. Fazendo um cálculo de

$$HistogramaAtual - HistogrmaAnterior > 1$$

Este deverá continuamente calcular o histograma da imagem (apenas uma componente de cor é suficiente) e compará-lo com o último histograma calculado.

pedro, deixe pra ver isso aqui depois srsrsr umas 18 h





Observase que

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

using namespace cv;
using namespace std;

int main(int argc, char** argv){
  Mat image;
  int width, height;
  VideoCapture cap;
  vector<Mat> planes;
  Mat histR, histG, histB;
  int nbins = 64;
  float range[] = {0, 256}, histograma_agora, histograma_antes;
  const float *histrange = { range };
  bool uniform = true;
  bool acummulate = false;


  int soma_histImgR = 0, soma_histImgBufR= 0;


  cap.open(0);

  if(!cap.isOpened()){
    cout << "cameras indisponiveis";
    return -1;
  }

  width  = cap.get(CV_CAP_PROP_FRAME_WIDTH);
  height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

  cout << "largura = " << width << endl;
  cout << "altura = " << height << endl;
```

```
  int histw = nbins, histh = nbins/2;
  Mat histImgR(histh, histw, CV_8UC3, Scalar(0,0,0));
  Mat histImgG(histh, histw, CV_8UC3, Scalar(0,0,0));
  Mat histImgB(histh, histw, CV_8UC3, Scalar(0,0,0));

  Mat histImgBufR(histh, histw, CV_8UC3, Scalar(0,0,0));

  histImgBufR.setTo(Scalar(0));

  int i = 0;
  while(1){
    cap >> image;
    split (image, planes);
    calcHist(&planes[0], 1, 0, Mat(), histR, 1,
             &nbins, &histrange,
             uniform, acummulate);
    calcHist(&planes[1], 1, 0, Mat(), histG, 1,
             &nbins, &histrange,
             uniform, acummulate);
    calcHist(&planes[2], 1, 0, Mat(), histB, 1,
             &nbins, &histrange,
             uniform, acummulate);

    normalize(histR, histR, 0, histImgR.rows, NORM_MINMAX, -1, Mat());
    normalize(histG, histG, 0, histImgR.rows, NORM_MINMAX, -1, Mat());
    normalize(histB, histB, 0, histImgR.rows, NORM_MINMAX, -1, Mat());

    histImgR.setTo(Scalar(0));
    histImgG.setTo(Scalar(0));
    histImgB.setTo(Scalar(0));

    for(int i=0; i<nbins; i++){
      line(histImgR, Point(i, histh),
           Point(i, cvRound(histR.at<float>(i))),
           Scalar(0, 0, 255), 1, 8, 0);
      line(histImgG, Point(i, histh),
           Point(i, cvRound(histG.at<float>(i))),
           Scalar(0, 255, 0), 1, 8, 0);
      line(histImgB, Point(i, histh),
           Point(i, cvRound(histB.at<float>(i))),
           Scalar(255, 0, 0), 1, 8, 0);
    }



//Verificando se a cena foi alterada, e caso sim, emissao de alarme
    for(int i = 0; i<histh; i++){
      for(int j = 0; j<histw; j++){
          soma_histImgR = soma_histImgR + histImgR.at<uchar>(i,j);
          soma_histImgBufR = soma_histImgBufR + histImgBufR.at<uchar>(i,j);
      }
    }
```

```
    float dividendo = histh*histw;
    histograma_agora = soma_histImgR/dividendo;
    histograma_antes = soma_histImgBufR/dividendo;


    if(abs(histograma_agora - histograma_antes)>1 && i != 0){
      cout<<"**  Ateno :   MOVIMENTAO  DETECTADA!! **\n";

    }



    histImgR.copyTo(image(Rect(0, 0    ,nbins, histh)));
    histImgG.copyTo(image(Rect(0, histh ,nbins, histh)));
    histImgB.copyTo(image(Rect(0, 2*histh ,nbins, histh)));
    imshow("image", image);
    if(waitKey(30) >= 0) break;
    histImgBufR = histImgR.clone();
    soma_histImgR = 0;
    soma_histImgBufR = 0;
    i++;
  }
  return 0;
}
```