Introdução ao Processamento Digital de Imagens - Exercícios em OpenCV

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

Disciplina: Processamento Digital de Imagens
Professor: Agostinho de Medeiros Brito Júnior

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 3.1 a ser criado o efeito do negativo.

Figura 3.1: Imagem Original

Conhecendo esses valores, é solicitado ao usuário que entre com dois pontos, (x1,y1) e (x2,y2), dentro do intervalo do tamanho da imagem.

Ao ler os pontos fornecidos pelo usuário, o efeito do negativo é mostrado, conforme ilustra a figura 3.2.

Para essa imagem o usuário entrou com os valores:

x1= 60, y1= 80, x2= 250, y2= 200;

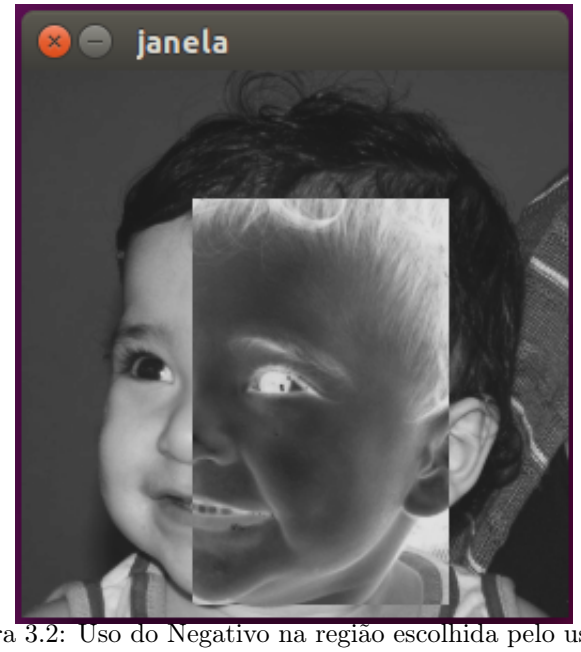


Figura 3.2: Uso do Negativo na região escolhida pelo usuário

Código costruído para fazer o negativo da 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 *trocaregioes.cpp*. Seu programa deverão trocar 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.

Utilizando a classe *Mat* e seus contrutores, como é visto no código a seguir. A imagem Original Figura 3.1 é dividida em 4 outras subimagnes, chamadas de imagem1, imagem2, imagem3 e imagem4, todas de mesmo tamanho e que serão reagrupadas, de forma embaralhada, formando uma imagem maior composta pelas quatro subimagens como é visto da Figura 3.2 .

Figura 3.2: Troca de Regiões.

Código utilizado para fazer o embaralhamento da imagem original:

```
#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 quato imagens 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));

  //A nova imagem passa a ser montada (embaralhada), com as imagens 1, 2, 3 e 4
  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;
}
```

4.1 *Observando-se o programa labeling.cpp como exemplo, é possível verificar que caso existam mais de 255 objetos na cena, o processo de rotulação poderá ficar comprometido. Identifique a situação em que isso ocorre e proponha uma solução para este problema.*

Em casos com imagens que contêm mais de 256 objetos não é possível rotular todos os objetos utilizando apenas os 256 tons de cinza reproduzíveis a partir de 8 bits. Logo, uma solução seria utilizar 3 canais de cores, como o padrão RGB, em que há 255 possibilidades para canal, resultando em $256^3 = 16777216$ possibilidades de rotulação.

4.2 *Aprimore o algoritmo de contagem apresentado para identificar regiões com ou sem buracos internos que existam na cena. Assuma que objetos com mais de um buraco podem existir. Inclua suporte no seu algoritmo para não contar bolhas que tocam as bordas da imagem. Não se pode presumir, a priori, que elas tenham buracos ou não.*

O algoritmo aprimorado é mostrado abaixo:

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

using namespace cv;
using namespace std;

void excluiObjetosDeBorda(Mat* image, int width, int height)
{
        CvPoint p;
        for(int i=0; i<height; i++)
        {
                if(image->at<uchar>(i,0) == 255)
                {   //Borda esquerda
                        p.x=0;
```

```
                p.y=i;
                floodFill(*image,p,0);
        }
        if(image->at<uchar>(i,width-1) == 255)
        {   //Borda direita
                p.x=width-1;
                p.y=i;
                floodFill(*image,p,0);
        }
    }


    for(int j=0; j<width; j++)
    {
        if(image->at<uchar>(0,j) == 255) //Borda superior
        {
                p.x=j;
                p.y=0;
                floodFill(*image,p,0);
        }
        if(image->at<uchar>(height-1,j) == 255) //Borda inferior
        {
                // achou um objeto
                p.x=j;
                p.y=height-1;
                floodFill(*image,p,0);
        }
    }

}

int main(int argc, char** argv){

    Mat image, thr;
    int width, height;
    int nobjects;

    CvPoint p;
    image = imread(argv[1],CV_LOAD_IMAGE_GRAYSCALE);

    if(!image.data)
    {
        std::cout << "imagem nao carregou corretamente\n";
        return(-1);
    }
    width=image.size().width;
    height=image.size().height;

    excluiObjetosDeBorda(&image, width, height);

    threshold(image,thr,200,255,THRESH_BINARY_INV);

    vector< vector <Point> > contours; // Vector for storing contour
    vector< Vec4i > hierarchy;
```

```
        int count=0;

        // busca objetos com buracos presentes
        findContours(thr, contours, hierarchy,CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE
            ); // Find the contours in the image
        for( int i = 0; i< contours.size(); i=hierarchy[i][0] ) // iterate through
            each contour.
        {
                Rect r= boundingRect(contours[i]);
                p.x = r.x;
                p.y = r.y;
                if(hierarchy[i][2]<0)
                {
                        if(image.at<uchar>(p.x,p.y) != 255)
                        {
                                p.x -= 1;
                        }
                        floodFill(image,p,count+128);
                        count++;
                }
        }
        cout << "O numero de objetos com buraco foi: " << count << endl;
        // busca objetos sem buracos presentes
        nobjects=0;
        for(int i=0; i<height; i++)
        {
                for(int j=0; j<width; j++)
                {
                        if(image.at<uchar>(i,j) == 255)
                        {
                                // achou um objeto
                                nobjects++;
                                p.x=j;
                                p.y=i;
                                floodFill(image,p,nobjects);
                        }
                }
        }

        cout << "O numero de objetos sem buracos foi: " << nobjects << endl;

        imshow("image", image);
        imwrite("labelingAprimorado.png", image);
        waitKey();
        return 0;
}
```

Primeiramente, esse programa lê a imagem de um arquivo através do seu caminho passado por parâmetro. Então os objetos que tocam as bordas da imagem são removidos pela função excluiObjetosDeBorda, que "varre"as bordas da imagem e altera a cor de cinza dos objetos que tocam a borda com a cor de cinza do fundo da imagem. Depois, a função threshold inverte as cores da imagem para facilitar a localização dos contornos. Após isso, a função findContours procura os contornos internos dos objetos e caso sejam encontrados são identifi-

cados a partir de tons de cinza de intensidade média. A imagem resultado é salva no arquivo labelingAprimorado.png

O arquivo de entrada utilizado foi:

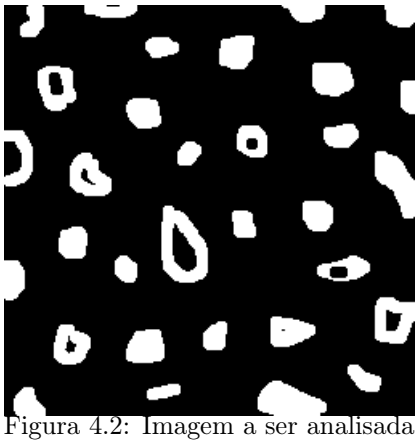
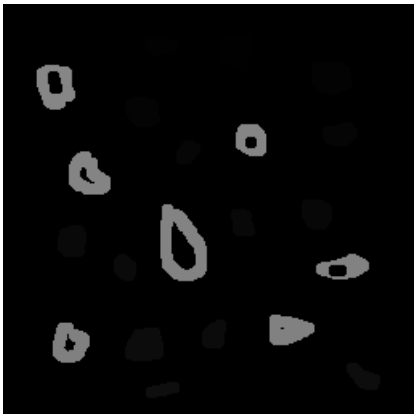


Figura 4.2: Imagem a ser analisada

E abaixo estão ilustrados as saídas dos programas labeling.cpp e labelingAprimorado.cpp

Saída Normal

Saída Aprimorada



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++ tal função recebe os seguintes parametros:

src – Canal de origem . // dst – Canal de destino.

void equalizeHist(InputArray src, OutputArray dst)

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

A equalização 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:

$$H' = \sum_{0 \leq j < i} H(j)$$

4- Transforma a imagem usada usando H': $dst(x, y) = H'(src(x, y))$

Utilizando a webcam e posicionando-a para diferentes niveis de iluminação, pode-se ter a imagem equalizada conforme a ilusta a figura 4.1, onde do lado equerdo tem a imagem originalmente capturada e do lado direto a imagem equalizada.

Observa-se que na na Figura 4.1, comparado o lado direito e esquerdo, tem-se no lado direito uma imagem com o contraste melhorado, isso se dá porque essa função atua de modo a esticar a faixa de intensidade da figura original.

Figura 4.1:Imagem original (à Esquerda) e Imagem equalizada

O código referente à equalização da imagem é visto a seguir:

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

using namespace cv;
using namespace std;

int main(int argc, char** argv){
  Mat image, equaliza;
  int width, height;//
  VideoCapture cap;
  vector<Mat> planes;
  vector<Mat> canais;//
  Mat histR, histG, histB;
  int nbins = 64;
  float range[] = {0, 256};
  const float *histrange = { range };
```

```
bool uniform = true;
bool acummulate = false;

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 histImgR_equaliza(histh, histw, CV_8UC3, Scalar(0,0,0));
Mat histImgG_equaliza(histh, histw, CV_8UC3, Scalar(0,0,0));
Mat histImgB_equaliza(histh, histw, CV_8UC3, Scalar(0,0,0));


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);
    }

 //equaliza histograma na canal
    split (image, canais);
// C++: void equalizeHist(InputArray src, OutputArray dst)
    equalizeHist(canais[0], canais[0]);
    equalizeHist(canais[1], canais[1]);
    equalizeHist(canais[2], canais[2]);
    merge(canais,equaliza);
    calcHist(&canais[0], 1, 0, Mat(), histR, 1,
            &nbins, &histrange,
            uniform, acummulate);
    calcHist(&canais[1], 1, 0, Mat(), histG, 1,
            &nbins, &histrange,
            uniform, acummulate);
    calcHist(&canais[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_equaliza.setTo(Scalar(0));
    histImgG_equaliza.setTo(Scalar(0));
    histImgB_equaliza.setTo(Scalar(0));

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

    //add

    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;
```

```
    histImgR_equaliza.copyTo(equaliza(Rect(0, 0  ,nbins, histh)));
    histImgG_equaliza.copyTo(equaliza(Rect(0, histh ,nbins, histh)));
    histImgB_equaliza.copyTo(equaliza(Rect(0, 2*histh ,nbins, histh)));
    imshow("equaliza", equaliza);
    if(waitKey(30) >= 0) break;
  }
  return 0;
}
```

5.2 Utilizando o programa exemplos/histogram.cpp como referência, implemente um programa motion-detector.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.

Nessa atividade é calculado o histograma atual da imagem na componente R, e comparado com a mesma componente do último histograma calculado. Fazendo uma comparação com a expressão

$$HistogramaAtual - HistogrmaAnterior > 1$$

Caso essa expressão seja verdadeira, significa que houve uma alteração do histograma atual em relação ao histograma anterior, logo existe uma movimentação detectada e o sistema emite um alerta conforme Figura 5.1 para, por exemplo, a movimentação vista na Figura 5.2.

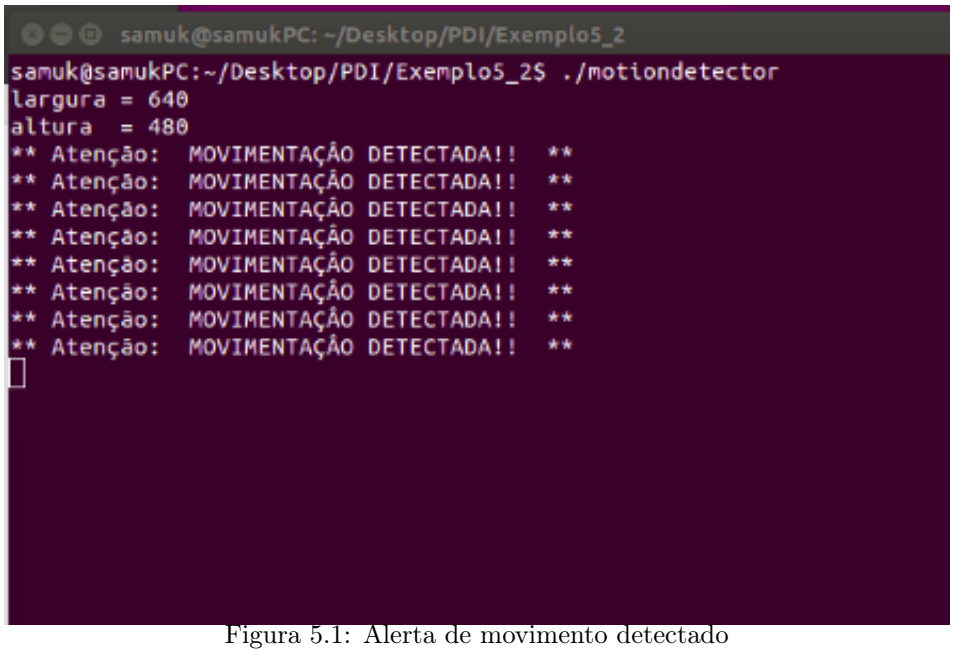


Figura 5.1: Alerta de movimento detectado

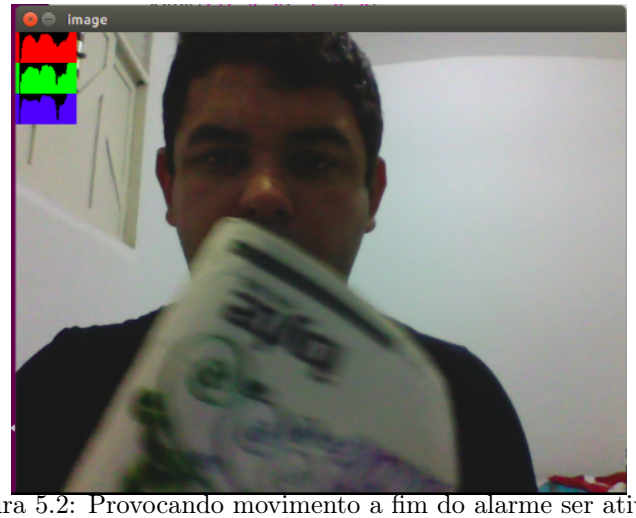


Figura 5.2: Provocando movimento a fim do alarme ser ativado

O código utilizado para detectar movimento é visto a seguir:

```
#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;
}
```

6.1 *Utilizando o programa exemplos/filtroespacial.cpp como referência, implemente um programa laplgauss.cpp. O programa deverá acrescentar mais uma funcionalidade ao exemplo fornecido, permitindo que seja calculado o laplaciano do gaussiano das imagens capturadas. Compare o resultado desse filtro com a simples aplicação do filtro laplaciano.*

O programa laplgauss.cpp que contém os filtros utilizados no programa filtroespacial adicionado do filtro laplaciano do gaussiano é mostrado abaixo:

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

using namespace cv;
using namespace std;

void printmask(Mat &m){
  for(int i=0; i<m.size().height; i++){
    for(int j=0; j<m.size().width; j++){
      cout << m.at<float>(i,j) << ",";
    }
    cout << endl;
  }
}

void menu(){
  cout << "\npressione a tecla para ativar o filtro: \n"
       "a - calcular modulo\n"
```

```
    "m - media\n"
    "g - gauss\n"
    "v - vertical\n"
        "h - horizontal\n"
    "l - laplaciano\n"
        "esc - sair\n";
}

int main(int argvc, char** argv){
  VideoCapture video;
  float media[] = {1,1,1,
                               1,1,1,
                               1,1,1};
  float gauss[] = {1,2,1,
                               2,4,2,
                               1,2,1};
  float horizontal[]={-1,0,1,
                                    -2,0,2,
                                    -1,0,1};
  float vertical[]={-1,-2,-1,
                                   0,0,0,
                                   1,2,1};
  float laplacian[]={0,-1,0,
                                    -1,4,-1,
                                    0,-1,0};

  Mat cap, frame, frame32f, frameFiltered;
  Mat mask(3,3,CV_32F), mask1, mask2;
  Mat result, result1;
  double width, height, min, max;
  int absolut;
  char key, currentOption;

  video.open(0);
  if(!video.isOpened())
    return -1;
  width=video.get(CV_CAP_PROP_FRAME_WIDTH);
  height=video.get(CV_CAP_PROP_FRAME_HEIGHT);
  std::cout << "largura=" << width << "\n";;
  std::cout << "altura =" << height<< "\n";;

  namedWindow("filtroespacial",1);

  mask = Mat(3, 3, CV_32F, media);
  scaleAdd(mask, 1/9.0, Mat::zeros(3,3,CV_32F), mask1);
  swap(mask, mask1);
  absolut=1; // calcs abs of the image

  menu();
  for(;;){
    video >> cap;
    cvtColor(cap, frame, CV_BGR2GRAY);
    flip(frame, frame, 1);
    imshow("original", frame);
```

```
frame.convertTo(frame32f, CV_32F);
if(currentOption == 'k')
{
    mask1 = Mat(3, 3, CV_32F, gauss);
    scaleAdd(mask1, 1/9.0, Mat::zeros(3,3,CV_32F), mask2);
    filter2D(frame32f, frame32f, frame32f.depth(), mask2, Point(1,1),0);
}

filter2D(frame32f, frameFiltered, frame32f.depth(), mask, Point(1,1),0);
if(absolut){
  frameFiltered=abs(frameFiltered);
}
frameFiltered.convertTo(result, CV_8U);
imshow("filtroespacial", result);
key = (char) waitKey(10);
if( key == 27 ) break; // esc pressed!
switch(key){
case 'a':
      menu();
  currentOption = 'a';
  absolut=!absolut;
  break;
case 'm':
      menu();
  currentOption = 'm';
  mask = Mat(3, 3, CV_32F, media);
  scaleAdd(mask, 1/9.0, Mat::zeros(3,3,CV_32F), mask1);
  mask = mask1;
  printmask(mask);
  break;
case 'g':
      menu();
  currentOption = 'g';
  mask = Mat(3, 3, CV_32F, gauss);
  scaleAdd(mask, 1/16.0, Mat::zeros(3,3,CV_32F), mask1);
  mask = mask1;
  printmask(mask);
  break;
case 'h':
      menu();
  currentOption = 'h';
  mask = Mat(3, 3, CV_32F, horizontal);
  printmask(mask);
  break;
case 'v':
      menu();
  currentOption = 'v';
  mask = Mat(3, 3, CV_32F, vertical);
  printmask(mask);
  break;
case 'l':
      menu();
  currentOption = 'l';
  mask = Mat(3, 3, CV_32F, laplacian);
```

```
      printmask(mask);
      break;
    case 'k':
         menu();
      currentOption = 'k';
      mask = Mat(3, 3, CV_32F, laplacian);
      printmask(mask);
      break;
    default:
      break;
    }
  }
  return 0;
}
```

O programa acima armazena os quadros capturados pela câmera padrão do computador e aplica o filtro atual escolhido pelo usuário. No caso do filtro laplaciano do gaussiano foi utilizada uma condição para que quando essa opção fosse escolhida que fosse aplicado um filtro gaussiano antes que fosse aplicada a convolução da máscara que representa a operação do laplaciano com a saída do filtro anterior.

As sáidas do filtros laplaciano e laplaciano do gaussiano são mostradas abaixo.

Figura 6.1: Laplaciano

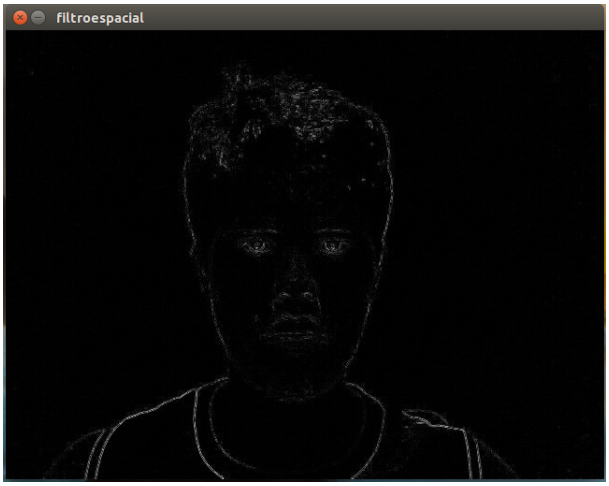
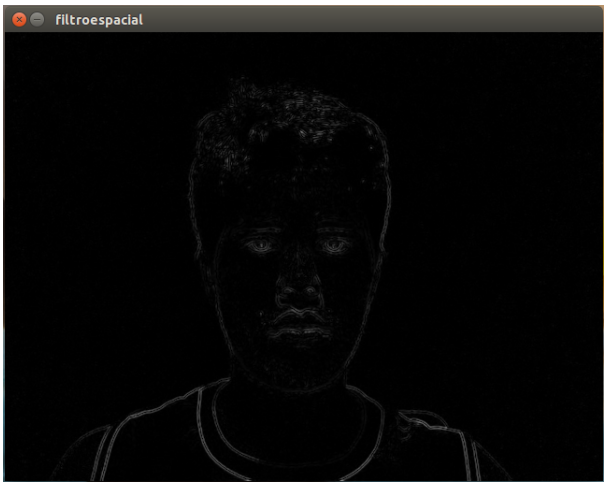


Figura 6.2: Laplaciano do Gaussiano



Analisando as saídas resultantes do programa, observa-se que o filtro laplaciano do gaussiano suaviza a região de borda da imagem, reduzindo o ruído presente nas sucessivas imagens e que o filtro laplaciano é mais sensível a esses ruídos, resultando em imagens mais instáveis principalmente nas regiões de borda, onde os pixels variam com alta frequência entre valores de preto e de branco.

7.1 *Utilizando o programa exemplos/addweighted.cpp como referência, implemente um programa tiltshift.cpp. Três ajustes deverão ser providos na tela da interface:*

- *um ajuste para regular a altura da região central que entrará em foco*;

- *um ajuste para regular a força de decaimento da região borrada*;

- *um ajuste para regular a posição vertical do centro da região que entrará em foco. Finalizado o programa, a imagem produzida deverá ser salva em arquivo.*

O programa tiltshift.cpp lê uma imagem e aplica os 3 ajustes citados acima para simular o efeito tiltshift encontrado em câmeras profissionais. Com essa técnica, uma imagem com objetos de tamanho real pode parecer que é uma imagem de miniaturas desses mesmos objetos, dependendo de fatores como: ângulo da foto e tamanho dos objetos em foco em relação ao tamanho da imagem, etc.

O programa tiltsift.cpp é mostrado abaixo:

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

using namespace cv;
using namespace std;

double alfa;
int alfa_slider = 0;
int alfa_slider_max = 100;

int top_slider = 0;
int top_slider_max = 100;

int center_slider = 50;
int center_slider_max = 100;

Mat image1, image2, blended, imagehsv;
Mat imageTop;
Mat frame32f, mask1, mask2;
vector<Mat> channels;

int width, height;

char TrackbarName[50];

void on_trackbar_blend(int, void*){
 alfa = (double) alfa_slider/alfa_slider_max ;
 addWeighted( image1, alfa, imageTop, 1-alfa, 0.0, blended);
 imshow("addweighted", blended);
}

int limitHeightBound(int param)
{
    if(param <= 0){
        return 1;
    }
    else if(param >= height){
        return height-1;
```

```
    }
    else{
        return param;
    }
}

void on_trackbar_height(int, void*){
  image1.copyTo(imageTop);
  int center = center_slider*(height-1)/100;
  int focusedHeight = top_slider*(height-1)/200;
  Mat tmp = image2(Rect(0, 0, width, limitHeightBound(center-focusedHeight)));
  tmp.copyTo(imageTop(Rect(0, 0, width, limitHeightBound(center-focusedHeight))));
  tmp = image2(Rect(0, limitHeightBound(center+focusedHeight), width,
      limitHeightBound(height-(center+focusedHeight))));
  tmp.copyTo(imageTop(Rect(0, limitHeightBound(center+focusedHeight), width,
      limitHeightBound(height-(center+focusedHeight)))));
  on_trackbar_blend(alfa_slider,0);
}

int main(int argvc, char** argv){
  image1 = imread("taxi.png", IMREAD_COLOR);

  width = image1.size().width;
  height = image1.size().height;

  /*cvtColor(image1, imagehsv, CV_BGR2HSV);
  split(imagehsv, channels);

  for(int i=0; i<height; i++){
    for(int j=0; j<width; j++){
        if(channels[1].at<uchar>(i,j) <= 130)
        {
                channels[1].at<uchar>(i,j) = 130;
        }
    }
  }

  merge(channels, imagehsv);
  cvtColor(imagehsv, image1, CV_HSV2BGR);
  */
  image1.copyTo(image2);
  image2.copyTo(imageTop);
  namedWindow("addweighted", 1);

  //Colocar Media
  float media[] = {1,1,1,
                   1,1,1,
                   1,1,1};

  image2.convertTo(image2, CV_32FC3);
  mask1 = Mat(3, 3, CV_32F, media);
  scaleAdd(mask1, 1/9.0, Mat::zeros(3,3,CV_32F), mask2);
  for(int i = 0; i < 3; i++)
  {
```

```
        filter2D(image2, image2, image2.depth(), mask2, Point(1,1),0);
    }
    image2.convertTo(image2, CV_8UC3);

    sprintf( TrackbarName, "Alpha x %d", alfa_slider_max );
    createTrackbar( TrackbarName, "addweighted",
                                &alfa_slider,
                                alfa_slider_max,
                                on_trackbar_blend );
    on_trackbar_blend(alfa_slider, 0);

    sprintf( TrackbarName, "Heightline x %d", top_slider_max );
    createTrackbar( TrackbarName, "addweighted",
                                &top_slider,
                                top_slider_max,
                                on_trackbar_height );
    sprintf( TrackbarName, "Centerline x %d", center_slider_max );
    createTrackbar( TrackbarName, "addweighted",
                                &center_slider,
                                center_slider_max,
                                on_trackbar_height );
    on_trackbar_height(top_slider, 0);

    waitKey(0);
    return 0;
}
```

O programa acima lê uma imagem de nome taxi.png e faz uma cópia dessa imagem. Na cópia é aplicado o filtro da média 5 vezes para que a imagem cópia fique com aspecto de imagem borrada. Então, a partir dos valores do trackbar, ajusta-se a posição da imagem original e posição da imagem borrada e o decaimento da imagem original na parte borrada.

O trackbar alpha controla a força de decaimento da imagem real na parte borrada, assim para valor alpha 0, apenas a imagem borrada aparece na posição delimitada para a imagem cópia, com alpha igual a 100, apenas a imagem original aparece nessa região e com valores intermediários ocorre uma mescla das 2 imagens.

O trackbar centerline indica onde é o centro da imagem original, para valor 0, o centro é o topo da imagem resultante, valor 50 indica o centro da imagem resultante e valor 100 a extremidade inferior.

Já o trackbar heightline indica o tamanho da imagem original, para valor 0, apenas a imagem borrada aparece, para valor 100 a imagem original aparece com metade da altura acima do centerline e metade da altura abaixo do centerline

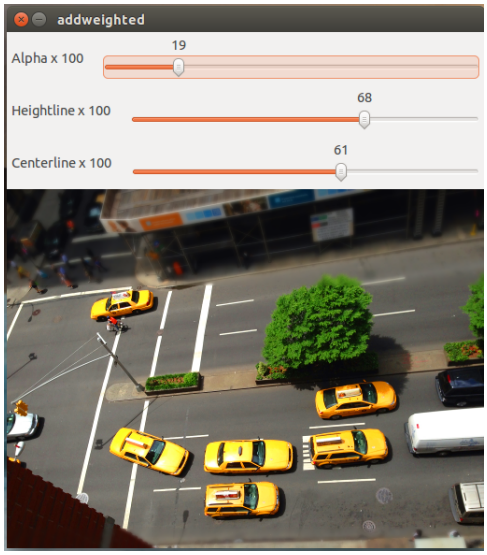
Tentou-se manipular a saturação da imagem, entretanto, com a imagem testada, pequenos incrementos no canal de saturação já resultavam em imagens insatisfatórias em que pixels com cor branca assumia valores correspondentes à cor vermelha ápos à alteração na saturação.

As imagem de entrada e saída do programa são mostrados abaixo:

Entrada

Saída

A interface do programa com o efeito de tiltshift é mostrado abaixo:



7.2 *Utilizando o programa exemplos/addweighted.cpp como referência, implemente um programa tiltshiftvideo.cpp. Tal programa deverá ser capaz de processar um arquivo de vídeo, produzir o efeito de tilt-shift nos quadros presentes e escrever o resultado em outro arquivo de vídeo. A ideia é criar um efeito de miniaturização de cenas. Descarte quadros em uma taxa que julgar conveniente para evidenciar o efeito de stop motion, comum em vídeos desse tipo.*

O programa tiltshiftvideo.cpp é mostrado abaixo:

```
// Vdeo  utilizado: https://www.youtube.com/watch?v=WrQ0bJpx7D4

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

```
using namespace cv;
using namespace std;

double alfa;
int alfa_slider = 0;
int alfa_slider_max = 100;

int top_slider = 0;
int top_slider_max = 100;

int center_slider = 50;
int center_slider_max = 100;

Mat image1, image2, blended;
Mat imageTop;
Mat frame32f, mask1, mask2;

int width, height, length;

int limitHeightBound(int param)
{
    if(param < 0){
        return 0;
    }
    else if(param > height){
        return height;
    }
    else{
        return param;
    }
}

Mat frame; //frame atual

int main(int argvc, char** argv){

  VideoCapture video;
  video.open("Apresentacao_Korea.mp4");
  if(!video.isOpened())
    return -1;

  length=video.get(CV_CAP_PROP_FRAME_COUNT);
  width=video.get(CV_CAP_PROP_FRAME_WIDTH);
  height=video.get(CV_CAP_PROP_FRAME_HEIGHT);

  // Mscara  do filtro da mdia Media
  float media[] = {1,1,1,
                   1,1,1,
                   1,1,1};

  //Montagem do Vdeo de  sada
  VideoWriter output_cap("Korea_Mini.mp4", video.get(CV_CAP_PROP_FOURCC),
      video.get(CV_CAP_PROP_FPS), Size(width, height));
```

```
  for(;;)
    {
        if (!video.read(image1))
            break;

        //Efeito Stop and motion
        video.grab();

        //Borra imagem cpia
        image1.copyTo(image2);
        image2.convertTo(image2, CV_32FC3);
        mask1 = Mat(3, 3, CV_32F, media);
        scaleAdd(mask1, 1/9.0, Mat::zeros(3,3,CV_32F), mask2);
        for(int i = 0; i < 3; i++)
        {
                filter2D(image2, image2, image2.depth(), mask2, Point(1,1),0);
        }
        image2.convertTo(image2, CV_8UC3);

        //Monta frame com imagem original e filtrada
        image1.copyTo(imageTop);
        int center = height/2;
        int focusedHeight = height/4;
        if(focusedHeight > 0){
                Mat tmp = image2(Rect(0, 0, width,
                    limitHeightBound(center-focusedHeight)));
                tmp.copyTo(imageTop(Rect(0, 0, width,
                    limitHeightBound(center-focusedHeight))));
                tmp = image2(Rect(0, limitHeightBound(center+focusedHeight), width,
                    limitHeightBound(height-(center+focusedHeight))));
                tmp.copyTo(imageTop(Rect(0, limitHeightBound(center+focusedHeight), width,
                    limitHeightBound(height-(center+focusedHeight)))));
        }
        alfa = 0.15;
        addWeighted(image1, alfa, imageTop, 1-alfa, 0.0, blended);

        //Salva frames modificados na sada
        output_cap.write(blended);

        char key = cvWaitKey(10);
        if (key == 27) // ESC
            break;

    }
  return 0;
}
```

Nesse programa os frames são capturados do arquivo de vídeo Apresentacao_Korea.mp4, a cada frame mostrado 1 frame é descartado para reproduzir o efeito de stop motion. Utilizando o mesmo método da atividade anterior, é montado um frame com as extremidades borradas e mescladas com a imagem original. Na parte final do programa o vídeo modificado é salvo no arquivo de sáida.

O arquivo de entrada é encontrado em https://www.youtube.com/watch?v=WrQ0bJpx7D4 e o arquivo de saída é encontrado em http://pedroklisley.github.io/

Referências

[1] Tutorial OpenCV - Professor Agostinho Brito, UFRN. Disponível em: <<http://agostinhobritojr.github.io/tutoriais/pdi/>>. Acesso entre 12 fev 2016 a 21 mar 2016.

[2] Tutorias OpenCV.ORG. Disponível em: <<http://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html?highlight=equalizehist>>. Acesso entre 1 e 21 mar 2016.

[3] PDI - Aula 2, IX Escola do CBPF. Disponível em: <<http://mesonpi.cat.cbpf.br/e2012/arquivos/g06/Aula2_G06.pdf>>. Acesso entre 1 e 21 mar 2016.