1. APRESENTAR NA TELA UMA IMAGEM

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

int main(int argc, char** argv){
  cv::Mat image;
  image = cv::imread(argv[1],cv::IMREAD_GRAYSCALE);
  cv::imshow("image", image);
  cv::waitKey();
  return 0;
}
```

2. MANIPULAR PIXEL

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

using namespace cv;
using namespace std;

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

  image= cv::imread("bolhas.png",cv::IMREAD_GRAYSCALE);
  if(!image.data)
    std::cout << "nao abriu bolhas.png" << std::endl;

  cv::namedWindow("janela", cv::WINDOW_AUTOSIZE);

  for(int i=200;i<210;i++){
    for(int j=10;j<200;j++){
      image.at<uchar>(i,j)=0;
    }
  }

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

  image= cv::imread("bolhas.png",cv::IMREAD_COLOR);

  val[0] = 0;   //B
  val[1] = 0;   //G
  val[2] = 255; //R

  for(int i=200;i<210;i++){
    for(int j=10;j<200;j++){
      image.at<Vec3b>(i,j)=val;
    }
  }

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

3. CONTAGEM E PREENCHIMENTO DE REGIÕES

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

using namespace cv;

int main(int argc, char** argv){
  cv::Mat image, realce;
  int width, height;
  int nobjects;

  cv::Point p;
  image = cv::imread(argv[1], cv::IMREAD_GRAYSCALE);

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

  width=image.cols;
  height=image.rows;
  std::cout << width << "x" << height << std::endl;

  p.x=0;
  p.y=0;

  // busca objetos 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;
          // preenche o objeto com o contador
            cv::floodFill(image,p,nobjects);
      }
    }
  }
  std::cout << "a figura tem " << nobjects << "
bolhas\n";
  cv::equalizeHist(image, realce);
  cv::imshow("image", image);
```

```
  cv::imshow("realce", realce);
  cv::imwrite("labeling.png", image);
  cv::waitKey();
  return 0;
}
```

4. Manipulação de histogramas

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

int main(int argc, char** argv){
  cv::Mat image;
  int width, height;
  cv::VideoCapture cap;
  std::vector<cv::Mat> planes;
  cv::Mat histR, histG, histB;
  int nbins = 64;
  float range[] = {0, 255};
  const float *histrange = { range };
  bool uniform = true;
  bool acummulate = false;
  int key;

      cap.open(2);

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

  cap.set(cv::CAP_PROP_FRAME_WIDTH, 640);
  cap.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
  width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
  height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);

  std::cout << "largura = " << width << std::endl;
  std::cout << "altura  = " << height << std::endl;

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

```
  cv::Mat histImgG(histh, histw, CV_8UC3,
cv::Scalar(0,0,0));
  cv::Mat histImgB(histh, histw, CV_8UC3,
cv::Scalar(0,0,0));

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

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

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

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

```
                cv::Scalar(255, 0, 0), 1, 8, 0);
    }
    histImgR.copyTo(image(cv::Rect(0, 0        ,nbins,
histh)));
    histImgG.copyTo(image(cv::Rect(0, histh    ,nbins,
histh)));
    histImgB.copyTo(image(cv::Rect(0, 2*histh ,nbins,
histh)));
    cv::imshow("image", image);
    key = cv::waitKey(30);
    if(key == 27) break;
  }
  return 0;
}
```

5. Filtragem no domínio espacial I

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

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

int main(int, char **) {
  cv::VideoCapture cap;  // open the default camera
  float media[] = {0.1111, 0.1111, 0.1111, 0.1111,
0.1111,
                   0.1111, 0.1111, 0.1111, 0.1111};
  float gauss[] = {0.0625, 0.125,  0.0625, 0.125, 0.25,
                   0.125,  0.0625, 0.125,  0.0625};
  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};
  float boost[] = {0, -1, 0, -1, 5.2, -1, 0, -1, 0};

  cv::Mat frame, framegray, frame32f, frameFiltered;
  cv::Mat mask(3, 3, CV_32F);
  cv::Mat result;
  double width, height;
  int absolut;
  char key;

  cap.open(0);

  if (!cap.isOpened())  // check if we succeeded
    return -1;

  cap.set(cv::CAP_PROP_FRAME_WIDTH, 640);
  cap.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
  width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
  height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);
  std::cout << "largura=" << width << "\n";
```

```
  ;
  std::cout << "altura =" << height << "\n";
  ;
  std::cout << "fps    =" << cap.get(cv::CAP_PROP_FPS) <<
"\n";
  std::cout << "format =" << cap.get(cv::CAP_PROP_FORMAT)
<< "\n";

  cv::namedWindow("filtroespacial", cv::WINDOW_NORMAL);
  cv::namedWindow("original", cv::WINDOW_NORMAL);

  mask = cv::Mat(3, 3, CV_32F, media);

  absolut = 1;  // calcs abs of the image

  for (;;) {
    cap >> frame;  // get a new frame from camera
    cv::cvtColor(frame, framegray, cv::COLOR_BGR2GRAY);
    cv::flip(framegray, framegray, 1);
    cv::imshow("original", framegray);
    framegray.convertTo(frame32f, CV_32F);
    cv::filter2D(frame32f, frameFiltered,
frame32f.depth(), mask,
                 cv::Point(1, 1), 0);
    if (absolut) {
      frameFiltered = cv::abs(frameFiltered);
    }

    frameFiltered.convertTo(result, CV_8U);

    cv::imshow("filtroespacial", result);

    key = (char)cv::waitKey(10);
    if (key == 27) break;  // esc pressed!
    switch (key) {
      case 'a':
        absolut = !absolut;
        break;
      case 'm':
        mask = cv::Mat(3, 3, CV_32F, media);
        printmask(mask);
        break;
      case 'g':
        mask = cv::Mat(3, 3, CV_32F, gauss);
```

```
        printmask(mask);
        break;
      case 'h':
        mask = cv::Mat(3, 3, CV_32F, horizontal);
        printmask(mask);
        break;
      case 'v':
        mask = cv::Mat(3, 3, CV_32F, vertical);
        printmask(mask);
        break;
      case 'l':
        mask = cv::Mat(3, 3, CV_32F, laplacian);
        printmask(mask);
        break;
      case 'b':
        mask = cv::Mat(3, 3, CV_32F, boost);
        break;
      default:
        break;
    }
  }
  return 0;
}
```

6. Filtragem no domínio espacial II

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

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

int top_slider = 0;
int top_slider_max = 100;

cv::Mat image1, image2, blended;
cv::Mat imageTop;

char TrackbarName[50];

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

void on_trackbar_line(int, void*){
  image1.copyTo(imageTop);
  int limit = top_slider*255/100;
  if(limit > 0){
    cv::Mat tmp = image2(cv::Rect(0, 0, 256, limit));
    tmp.copyTo(imageTop(cv::Rect(0, 0, 256, limit)));
  }
  on_trackbar_blend(alfa_slider,0);
}

int main(int argvc, char** argv){
  image1 = cv::imread("blend1.jpg");
  image2 = cv::imread("blend2.jpg");
  image2.copyTo(imageTop);
  cv::namedWindow("addweighted", 1);

  std::sprintf( TrackbarName, "Alpha x %d",
alfa_slider_max );
  cv::createTrackbar( TrackbarName, "addweighted",
```

```
                        &alfa_slider,
                        alfa_slider_max,
                        on_trackbar_blend );
  on_trackbar_blend(alfa_slider, 0 );

  std::sprintf( TrackbarName, "Scanline x %d",
top_slider_max );
  cv::createTrackbar( TrackbarName, "addweighted",
                        &top_slider,
                        top_slider_max,
                        on_trackbar_line );
  on_trackbar_line(top_slider, 0 );

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

7. Filtragem no domínio da frequência