OpenCV

Fernando Ribeiro

2018

Utilização nas práticas

- SOFTWARE UTILIZADO
- OpenCV 3.3.0
- Qt Creator 4.4.0

- VIDEOS DE AJUDA PARA INSTALAÇÃO
- https://wiki.qt.io/How_to_setup_Qt_and_openCV_on_Windows
- https://www.youtube.com/watch?v=ox6NFWE8xg4

Algumas referências

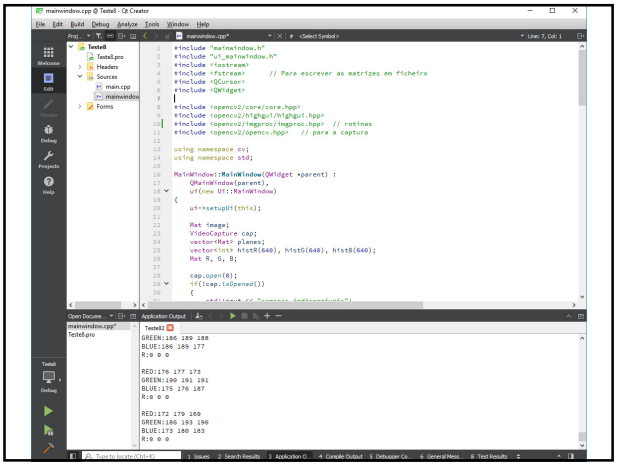
- Slides das aulas
- https://opencv.org/
- https://docs.opencv.org/3.3.0/
- Vídeos no Youtube
- Livros em PDF
- Sites pessoais: http://ubaa.net/shared/processing/opencv/

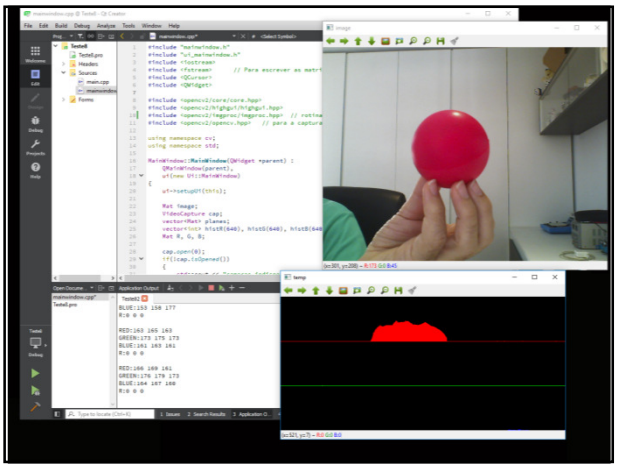
Ficheiro .PRO (project)

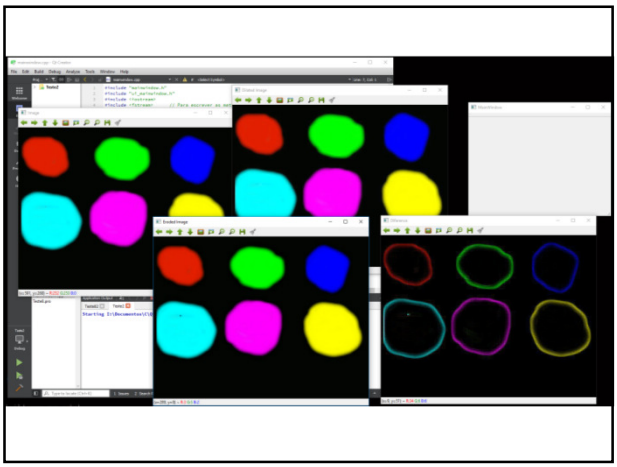
```
QT += core gui

greaterThan(QT_MAJOR_VERSION, 4) : QT += widgets

TARGET = Teste8
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS

SOURCES += \
        main.cpp \
        mainwindow.cpp

HEADERS += \
        mainwindow.h

FORMS += \
        mainwindow.ui

INCLUDEPATH += C:\opencv\opencv-original\build\include

LIBS += C:\opencv\bin\libopencv_core330.dll
LIBS += C:\opencv\bin\libopencv_highgui330.dll
LIBS += C:\opencv\bin\libopencv_imgcodecs330.dll
LIBS += C:\opencv\bin\libopencv_imgproc330.dll
LIBS += C:\opencv\bin\libopencv_features2d330.dll
LIBS += C:\opencv\bin\libopencv_calib3d330.dll
LIBS += C:\opencv\bin\libopencv_videoio330.dll
```

Includes

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <iostream>
#include <fstream> // Para utilização de ficheiros
#include <QWidget>
#include <QCursor>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>  // rotinas de proc. imagem
#include <opencv2/opencv.hpp>           // para captura de vídeo

using namespace cv;
using namespace std;
```







Matrizes

```
cv::Mat A = cv::Mat(3, 2, CV_8U);          // 8 bits unsigned char
cv::Mat B = cv::Mat::ones( 4, 4, CV_32F);      // matriz a 1...

cv::Mat C = cv::Mat::zeros(5, 5, CV_32F);      // matriz a 0...
cv::Mat D = cv::Mat::eye(4, 4, CV_32F);        // identidade
cv::Mat E = cv::Mat::ones( 3, 3, CV_32F)/(float)(3*3);
cv::Mat F = cv::Mat::eye(4, 4, CV_32F)*0.1;    // 0.1 das diagonais
cv::Mat G = (cv::Mat_<double>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
cv::Mat H = cv::Mat(3, 2, CV_8UC3);            // 3 bytes (ex: RGB)
B = B/7.0;

cv::Mat imagem;

cv::Mat linha = C.row(1).clone();          // cria matriz linha
```

Matrizes

```
A
[ 0, 0;
  0, 0;
  0, 0]

B
[1, 1, 1, 1;
 1, 1, 1, 1;
 1, 1, 1, 1;
 1, 1, 1, 1]

C
[0, 0, 0, 0, 0;
 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0]

D
[1, 0, 0, 0;
 0, 1, 0, 0;
 0, 0, 1, 0;
 0, 0, 0, 1]

E
[0.11111111, 0.11111111, 0.11111111;
 0.11111111, 0.11111111, 0.11111111;
 0.11111111, 0.11111111, 0.11111111]

F
[0.1, 0, 0, 0;
 0, 0.1, 0, 0;
 0, 0, 0.1, 0;
 0, 0, 0, 0.1]

G
[ 0, -1,  0;
 -1,  5, -1;
  0, -1,  0]

H
[ 0, 0, 0, 0, 79, 112;
 101, 110, 67, 76, 32, 49;
 46, 48, 32, 67, 85, 68]

B
[0.14285715, 0.14285715, 0.14285715, 0.14285715;
 0.14285715, 0.14285715, 0.14285715, 0.14285715;
 0.14285715, 0.14285715, 0.14285715, 0.14285715;
 0.14285715, 0.14285715, 0.14285715, 0.14285715]
```

Constituição de uma imagem

Imagem em tons de cinzento

```
CCC...
CCC...
CCC...
...
```

C	C	C	...	C
C	C	C	...	C
C	C	C	...	C
...	...	...	...	...
C	C	C	...	C

Imagem a cores

```
[BGR][BGR][BGR]...
[BGR][BGR][BGR]...
[BGR][BGR][BGR]...
...
```

RGB	RGB	RGB	...	RGB
RGB	RGB	RGB	...	RGB
RGB	RGB	RGB	...	RGB
...	...	...	...	...
RGB	RGB	RGB	...	RGB

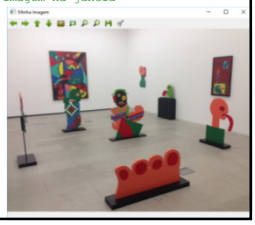
Imagens

```
cv::Mat variavel(240,320,CV_8U,cv::Scalar(100));

cv::Mat imagem = cv::imread("i://QtTest//test.jpg", 1);    // Lê uma imagem
std::cout << "Size: " << imagem.size().height << ", " << imagem.size().width ;
std::cout << imagem
```

```
640, 480
[211, 217, 223, 210, 216, 223, 212, 218, 225, 213, 219, 226, 214, 219, 226, 214, 217, 225, 215, 218, 226, 215, 219, 226, 215, 221, 228, 213,
 219, 226, 214, 220, 227, 214, 220, 227, 215, 221, 228, 218, 220, 229, 215, 220, 228, 211, 223, 227, 212, 222, 227, 217, 222, 229, 218, 223,
 230, 216, 222, 229, 216, 222, 229, 217, 223, 230, 219, 223, 232, 220, 223, 231, 220, 223, 231, 219, 224, 231, 219, 225, 232, 219, 225, 232,
 219, 225, 232, 219, 224, 233, 221, 225, 233, 222, 225, 233, 221, 224, 232, 222, 225, 233, 223, 226, 234, 223, 225, 233, 223, 226, 233, 224,
 227, 233, 224, 227, 232, 224, 226, 233, 224, 226, 234, 223, 228, 237, 224, 228, 237, 223, 227, 236, 224, 227, 235, 226, 230, 235, 227, 230,
 234, 229, 229, 235, 228, 230, 235, 228, 231, 236, 228, 231, 236, 229, 231, 239, 228, 230, 238, 230, 231, 239, 231, 230, 239, 230, 230, 239,
 229, 231, 239, 229, 231, 239, 230, 232, 240, 231, 232, 239, 232, 232, 238, 233, 233, 239, 233, 233, 239, 233, 233, 239, 232, 234, 239, 232,
 235, 240, 232, 235, 240, 232, 235, 240, 233, 233, 239, 233, 233, 239, 233, 235, 240, 232, 235, 240, 234, 233, 241, 234, 234, 241, 235, 233,
 239, 235, 234, 240, 233, 233, 239, 233, 233, 239, 234, 234, 240, 233, 235, 240, 232, 235, 239, 234, 235, 239, 234, 235, 239, 234, 234, 240,
 235, 236, 241, 232, 235, 240, 235, 235, 241, 237, 236, 242, 238, 236, 242, 238, 236, 242, 238, 236, 242, 238, 236, 241, 237, 237, 241, 235,
 237, 241, 234, 237, 241, 233, 236, 240, 234, 237, 242, 234, 238, 242, 234, 237, 242, 236, 236, 240, 239, 237, 241, 237, 238, 242, 236, 237,
 241, 236, 237, 241, 235, 236, 240, 237, 238, 242, 239, 238, 242, 237, 237, 241, 235, 236, 240, 237, 238, 242, 235, 235, 241, 236, 236, 241,
 237, 238, 241, 236, 237, 239, 235, 236, 240, 236, 237, 241, 238, 237, 241, 239, 236, 241, 240, 235, 242, 239, 235, 241, 238, 236, 242, 237,
 236, 242, 236, 236, 242, 235, 234, 242, 236, 236, 242, 235, 235, 241, 236, 235, 243, 236, 234, 242, 237, 235, 241, 237, 236, 240, 236, 235,
 239, 236, 235, 239, 235, 236, 240, 234, 235, 239, 235, 236, 240, 233, 235, 238, 234, 234, 240, 234, 234, 240, 235, 235, 239, 234, 233, 237,
 233, 234, 238, 234, 236, 238, 234, 236, 237, 233, 235, 237, 233, 234, 237, 235, 235, 239, 235, 233, 239, 235, 233, 239, 235, 233, 239, 234,
 232, 238, 235, 233, 239, 235, 235, 239, 232, 233, 237, 233, 233, 237, 236, 234, 239, 234, 232, 239, 234, 233, 238, 235, 233, 238, 235, 233,
 239, 234, 233, 237, 234, 233, 237, 234, 233, 237, 233, 232, 236, 234, 232, 237, 233, 232, 236, 233, 232, 236, 233, 232, 236, 233, 232, 236,
 237, 242, 234, 238, 242, 234, 237, 242, 236, 236, 240, 239, 237, 241, 237, 238, 242, 236, 237, 241, 236, 237, 241, 235, 236, 240, 237, 238,
 242, 239, 238, 242, 237, 237, 241, 235, 236, 240, 237, 238, 242, 235, 235, 241, 236, 236, 241, 237, 238, 241, 236, 237, 239, 235, 236, 240,
 236, 237, 241, 238, 237, 241, 239, 236, 241, 240, 235, 242, 239, 235, 241, 238, 236, 242, 237, 236, 242, 236, 236, 242, 235, 234, 242, 236,
 236, 242, 235, 235, 241, 236, 235, 243, 236, 234, 242, 237, 235, 241, 237, 236, 240, 236, 235, 239, 236, 235, 239, 235, 236, 240, 234, 235,
```

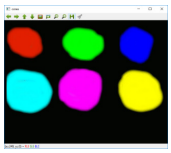
Imagens

```
cv::Mat variavel(240,320,CV_8U,cv::Scalar(100));
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);  // Lê uma imagem
if (!imagem.data)           // se der erro na leitura
     return -1;

cv::namedWindow("Minha Imagem"); // cria janela chamada "Minha Imagem"
cv::namedWindow("Output", CV_WINDOW_NORMAL);

cv::imshow("Minha Imagem", imagem); // mostra imagem na janela
           // se não existir cria-a

// Coluna, Linha
cv::moveWindow("Output" , 0, 0);

// Destroi janela
cv::destroyWindow("Output");

// grava imagem em ficheiro
cv::imwrite("i://QtTest//output.bmp", imagem);
```



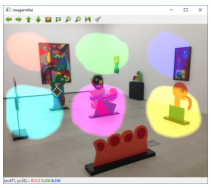
Funções

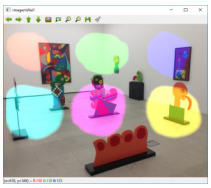
```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::Mat cores = cv::imread("i://QtTest//cores.jpg", 1);
cv::imshow("Image",imagem);
cv::imshow("cores",cores);

cv::Mat grande, invertida, imagemWei, imagemWei1, imagemAdd, outra;
cv::flip(imagem, invertida, 1); // >0 horizontal, 0 vertical, <0 ambas
cv::imshow("invertida",invertida);

cv::resize(imagem, grande, cv::Size(800, 600));
cv::imshow("grande", grande);
```

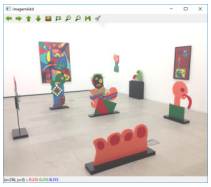


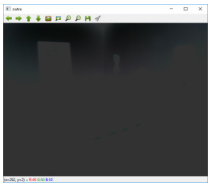
Funções

```
// adiciona duas imagens, 0.9 de uma e 0.5 de outra
cv::addWeighted(imagem, 0.9, cores, 0.5, imagemWei);
cv::imshow("imagemWei", imagemWei);
imagemWei1 = 0.9*imagem+0.5*cores;// Sim, é possível
cv::imshow("imagemWei1", imagemWei1);

// adiciona o valor 50 a cada canal da imagem
cv::add(imagem, cv::Scalar(50, 50, 50), imagemAdd);
cv::imshow("imagemAdd", imagemAdd);

cv::subtract(imagemAdd, imagem, outra);
cv::imshow("outra", outra);
```









Aceder aos valores

```
 std::cout << element5 ;
     [  0,   1,   2,   3,   4;
        1,   2,   3,   4,   5;
        2,   3,   4,   5,   6;
        3,   4,   5,   6,   7;
        4,   5,   6,   7,   8]

if (image.channels() == 1) // imagem em tons de cinzento
        image.at<uchar>(j,i)= 0;

if (image.channels() == 3) // imagem RGB
{
     image.at<cv::Vec3b>(j,i)[0]=   0;// canal 0, BLUE
     image.at<cv::Vec3b>(j,i)[1]=   0; // canal 1, GREEN
     image.at<cv::Vec3b>(j,i)[2]= 255; // canal 2, RED
}

 // Para aceder aos pixeis com apontadores
    uchar* data= image.ptr<uchar>(27);
    *data++= *data/9 + 25;
```

Aceder aos valores

```
using namespace cv;
using namespace std;
```

```
Mat imagem;
Vec3b valor;

imagem = imread("i://gray.png",CV_LOAD_IMAGE_GRAYSCALE);
if(!imagem.data)
     cout << "nao abriu ficheiro de cinzentos" << endl;

for(int linha=200;linha<220;linha++){
     for(int coluna=10;coluna<200;coluna++){
          imagem.at<uchar>(linha,coluna)=0;
     }
}
imshow("cinzentos", imagem);
```

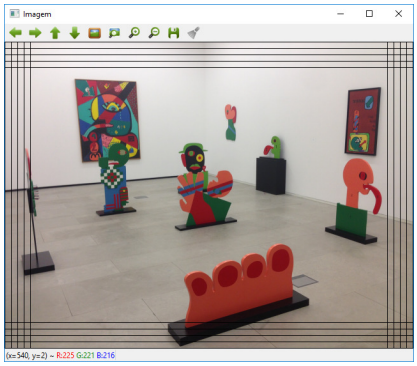
Aceder aos valores

```
using namespace cv;
using namespace std;
```

```
Mat imagem;
Vec3b valor;

imagem = imread("i://QtTest//jg.png",CV_LOAD_IMAGE_COLOR);
if(!imagem.data)
     cout << "nao abriu ficheiro de cores" << endl;

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

for(int linha=200;linha<220;linha++){
     for(int coluna=10;coluna<200;coluna++){
          imagem.at<Vec3b>(linha,coluna)=valor;
     }
}
imshow("cores", imagem);
```

Aceder aos valores

```
cv::Mat imagem= cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::imshow("Imagem", imagem);

for (int i=0;i<50;i+=10)
{
    imagem.row(i).setTo(cv::Scalar(0)); // Top row
    imagem.row(imagem.rows - 1 - i).setTo(cv::Scalar(0)); // Bottom row
    imagem.col(i).setTo(cv::Scalar(0)); // Left column
    imagem.col(imagem.cols - 1 -i).setTo(cv::Scalar(0)); // Right column
}
cv::imshow("Imagem", imagem);
```



Funções geométricas

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);

cv::Point p1, p2;
p1.x=330; p1.y= 90;
p2.x=385; p2.y=170;

cv::rectangle(imagem, p1, p2, cv::Scalar(255,0,0));
cv::line(imagem, p1, p2, cv::Scalar(0, 0, 255));
cv::line(imagem, cv::Point(330,170), cv::Point(385,90), cv::Scalar(0, 255, 0));
cv::putText(imagem, "Ola Mundo", cv::Point(200, 50), FONT_HERSHEY_PLAIN, 2,
cv::Scalar(255, 255, 0), 3, 8);
cv::imshow("Image",imagem);
```

Funções geométricas

```
cv::Point cb[1][7];
cb[0][0] = Point( 20, 40 );
cb[0][1] = Point( 40, 60 );
cb[0][2] = Point(100, 60 );
cb[0][3] = Point(120, 40 );
cb[0][4] = Point( 90, 40 );
cb[0][5] = Point( 70, 10 );
cb[0][6] = Point( 50, 40 );
const Point* ppt[1] = { cb[0] };
int npt[] = { 7 };
cv::fillPoly( imagem, ppt, npt, 1, Scalar( 0, 255, 255 ), 4 );

cv::imshow("Image",imagem);
```

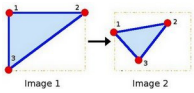
Funções geométricas

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

cv::Point centroC(200, 200), centroE(300, 100);
cv::circle(imagem, centroC, 50, cv::Scalar(255,255,0), 1, cv::LINE_8, 0);
//   circle(imagem, cv::Point(150,200), 50, cv::Scalar(255,255,0), 1, cv::LINE_8, 0);
//   circle(imagem, cv::Point(p.rx(),p.ry()), 50, cv::Scalar(255,255,0), 1, cv::LINE_8, 0);
//   circle(imagem, centro, 50, cv::Scalar(255), 1, cv::LINE_8, 0);
cv::ellipse(imagem, centroE, cv::Size(100,60), 30.0, 0.0, 270.0, cv::Scalar(255,255,0), -1,
8, 0); // -1 preenche a elipse

cv::imshow("imagem",imagem);
```

Warp - Affine

What is an Affine Transformation?

1.It is any transformation that can be expressed in the form of a *matrix multiplication* (linear transformation) followed by a *vector addition* (translation).
2.From the above, We can use an Affine Transformation to express:
1. Rotations (linear transformation)
2. Translations (vector addition)
3. Scale operations (linear transformation)

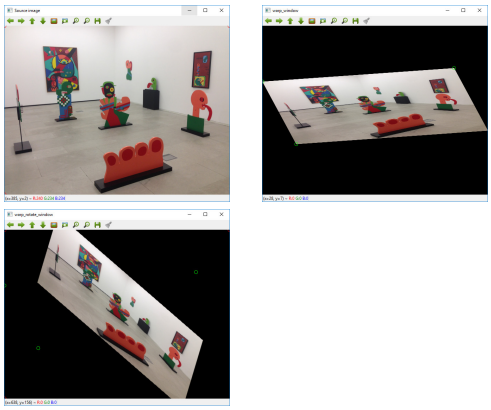


Warp - Affine

```
Point2f coor_origem[3];
Point2f coor_destino[3];

Mat mat_rotacao( 2, 3, CV_32FC1 );
Mat mat_warp( 2, 3, CV_32FC1 );
Mat original, imagem_warped, imagem_warped_rodada;

// Lê image original
original = cv::imread("i://QtTest//jg.png");

// Define matriz com as mesmas caracteristicas que a original
imagem_warped = Mat::zeros( original.rows, original.cols, original.type() );

// Define os 3 pontos para calcular a transformação de Affine (na Origem)
coor_origem[0] = Point2f( 0,0 );
coor_origem[1] = Point2f( original.cols - 1, 0 );
coor_origem[2] = Point2f( 0, original.rows - 1 );

// Define os 3 pontos para calcular a transformação de Affine (no destino)
coor_destino[0] = Point2f( original.cols*0.0, original.rows*0.33 );
coor_destino[1] = Point2f( original.cols*0.85, original.rows*0.25 );
coor_destino[2] = Point2f( original.cols*0.15, original.rows*0.7 );
```

Warp - Affine

```
// Calcula a Transformada de Affine
mat_warp = getAffineTransform( coor_origem, coor_destino );

// Aplica a Transformada de Affine
warpAffine( original, imagem_warped, mat_warp, imagem_warped.size() );

// Calcula a matriz de rotacao em relação ao centro da imagem
Point axis = Point( imagem_warped.cols/2, imagem_warped.rows/2 );
double angle = -45.0;   // definir a rotacao desejada
double scale = 1.0;     // definir o escalamento desejado

// Calcula a matriz de rotação
mat_rotacao = getRotationMatrix2D( axis, angle, scale );

// Aplica a rotação na imagem warped
warpAffine( imagem_warped, imagem_warped_rodada, mat_rotacao, imagem_warped.size() );

// Mostra o resultado
imshow("Source image", original );
imshow("warp_window", imagem_warped );
imshow("warp_rotate_window", imagem_warped_rodada );
```

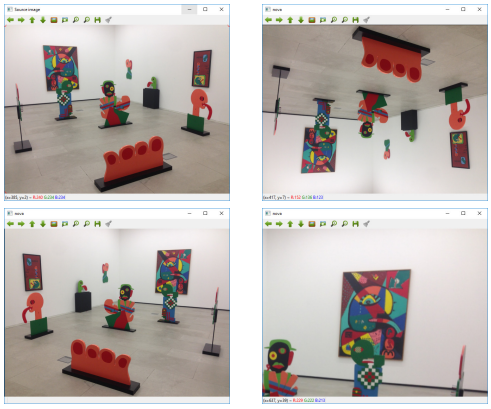


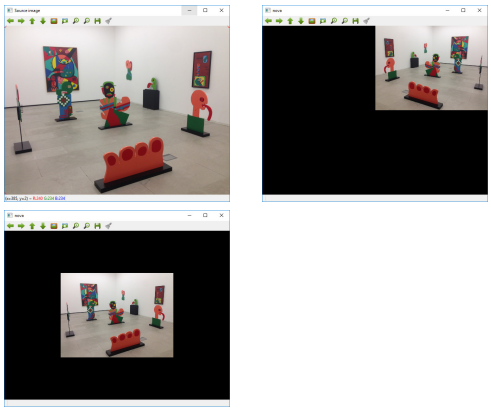
Remap

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

cv::Mat nova, map_x, map_y;
nova.create( imagem.size(), imagem.type() );
map_x.create( imagem.size(), CV_32FC1 );
map_y.create( imagem.size(), CV_32FC1 );

for( int i = 0; i < imagem.rows; i++ )
{
    for( int j = 0; j < imagem.cols; j++ )
    {
        map_x.at<float>(i,j) = (float)j;
        map_y.at<float>(i,j) = (float)(imagem.rows - i) ;
    }
}

cv::remap( imagem, nova, map_x, map_y, CV_INTER_LINEAR,
     cv::BORDER_CONSTANT, cv::Scalar( 0, 0, 0) );
cv::imshow("nova", nova);
```

Remap

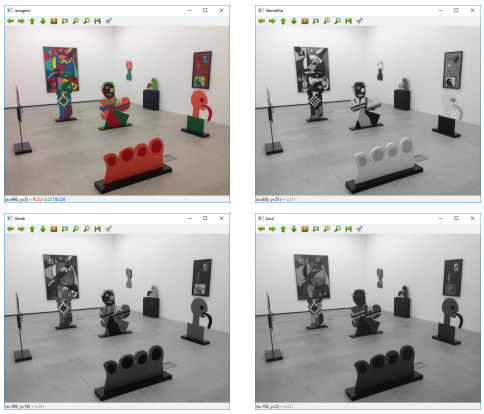
```
map_x.at<float>(i,j) = (float)j;
map_y.at<float>(i,j) = (float)(imagem.rows - i);

map_x.at<float>(i,j) = (float)(imagem.cols -j);
map_y.at<float>(i,j) = (float)i ;

map_x.at<float>(i,j) = (float)(imagem.cols -j)/2;
map_y.at<float>(i,j) = (float)i/2 ;

map_x.at<float>(i,j) = (float)(imagem.cols -j)*2;
map_y.at<float>(i,j) = (float)i*2 ;

map_x.at<float>(i,j) = (float)(j - imagem.cols/4)*2;
map_y.at<float>(i,j) = (float)(i - imagem.rows/4)*2 ;
```

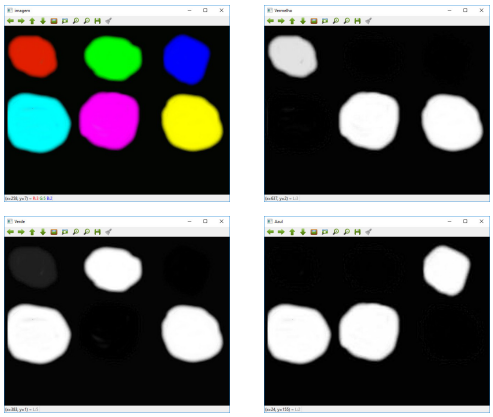




Canais de cor

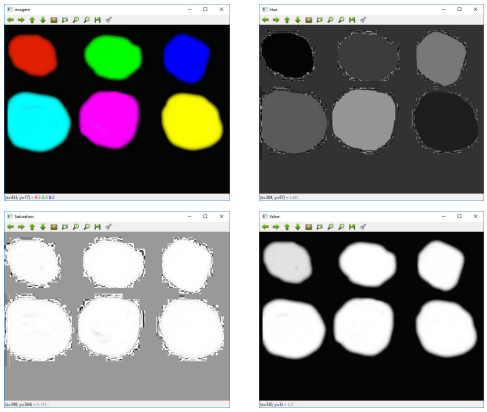
```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

std::vector<cv::Mat> planos;  // cria vector de 3 imagens
cv::split(imagem, planos);        // separa em 3 canais

cv::imshow("Azul", planos[0]);
cv::imshow("Verde", planos[1]);
cv::imshow("Vermelho", planos[2]);
```





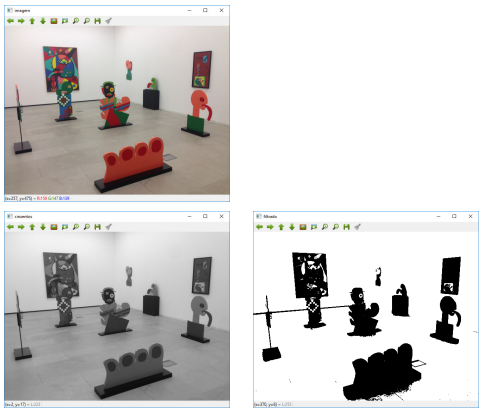
Canais de cor

```
cv::Mat imagem = cv::imread("i://QtTest//cores.jpg", 1);
cv::imshow("imagem",imagem);

std::vector<cv::Mat> v;   // cria vector de 3 imagens
cv::cvtColor(imagem, hsv, CV_BGR2HSV);
cv::split(hsv, v);     // Split the image

cv::imshow("Hue",v[0]);
cv::imshow("Saturation",v[1]);
cv::imshow("Value",v[2]);
```



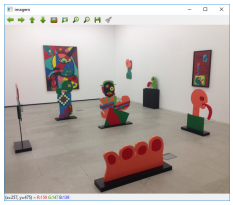
Conversões / Thresholds

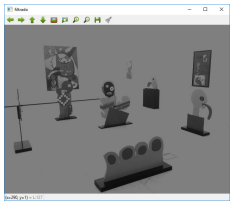
```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

cv::Mat cinzentos, filtrado, RGB;

cv::cvtColor(imagem, RGB, CV_BGR2RGB); // inverte o formato de RGB para BGR

cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY); // converte em cinzentos
      //precisa de #include <opencv2/imgproc/imgproc.hpp>

cv::threshold(cinzentos, filtrado, 127, 255, cv::THRESH_BINARY);
                // TYPE = BINARY, BINARY_INV, TRUNC, TOZERO, TOZERO_INV

cv::imshow("cinzentos", cinzentos);
cv::imshow("filtrado", filtrado);
```



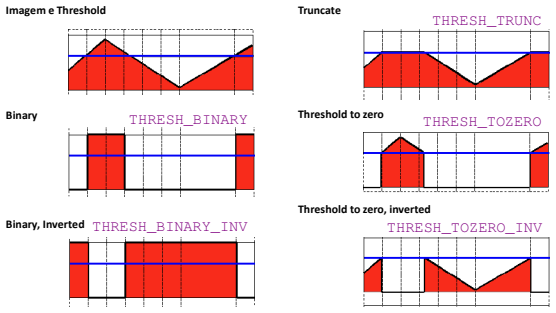
Conversões / Thresholds

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

cv::Mat cinzentos, filtrado, RGB;

cv::cvtColor(imagem, RGB, CV_BGR2RGB);// formato de RGB para BGR

// converte em cinzentos
cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY);
            //precisa de #include <opencv2/imgproc/imgproc.hpp>

cv::threshold(cinzentos, filtrado, 127, 255, cv::THRESH_TRUNC);
             // TYPE = BINARY, BINARY_INV, TRUNC, TOZERO, TOZERO_INV

cv::imshow("cinzentos", cinzentos);
cv::imshow("filtrado", filtrado);
```



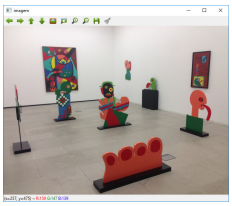


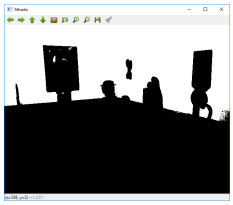
Tipos de Threshold



Thresholds - inRange

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::Mat cinzentos, filtrado;

cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

cv::inRange(cinzentos, 180, 255, filtrado);
cv::imshow("filtrado", filtrado);
```

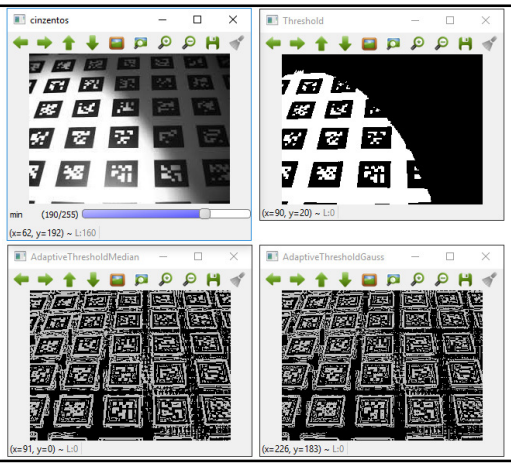




Thresholds - AdaptiveThreshold

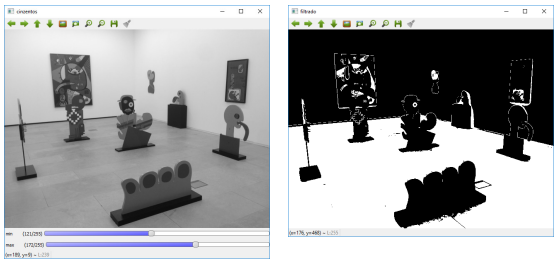
```
int th = 127;

cv::Mat cinzentos, thre, thre_ada_gauss, thre_ada_median;

void on_trackbar( int, void* )
{
    cv::threshold(cinzentos, thre, th, 255, cv::THRESH_BINARY);
    cv::imshow("Threshold", thre);
    cv::adaptiveThreshold(cinzentos, thre_ada_gauss, th, cv::ADAPTIVE_THRESH_GAUSSIAN_C,
cv::THRESH_BINARY, 5, 0);
    cv::imshow("AdaptiveThresholdGauss", thre_ada_gauss);
    cv::adaptiveThreshold(cinzentos, thre_ada_median, th, cv::ADAPTIVE_THRESH_MEAN_C,
cv::THRESH_BINARY, 5, 0);
    cv::imshow("AdaptiveThresholdMedian", thre_ada_median);
}

cv::Mat imagem = cv::imread("i://QtTest//At1.jpg", 1);
cv::imshow("imagem",imagem);

cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

cv::createTrackbar("min", "cinzentos", &th, 255, on_trackbar );
on_trackbar(0, 0);
```

```
adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod,
int thresholdType, int blockSize, double C)
```



Thresholds com TrackBar

```
int threshold_min = 127;
int threshold_max = 200;

cv::Mat cinzentos, filtrado;

void on_trackbar( int, void* )
{
        cv::inRange(cinzentos, threshold_min, threshold_max, filtrado);
        cv::imshow("filtrado", filtrado);
}

cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY);
cv::namedWindow("cinzentos", 1);
cv::imshow("cinzentos", cinzentos);

cv::createTrackbar("min", "cinzentos", &threshold_min, 255, on_trackbar );
cv::createTrackbar("max", "cinzentos", &threshold_max, 255, on_trackbar );

cv::inRange(cinzentos, threshold_min, threshold_max, filtrado);
cv::imshow("filtrado", filtrado);
```

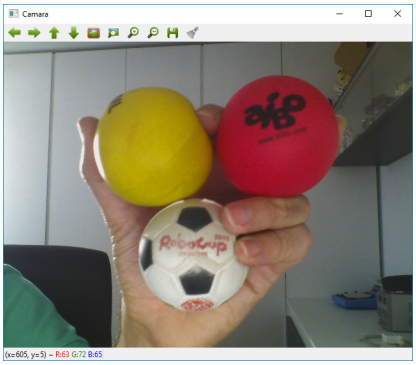


Thresholds com TrackBar

```
int threshold_min = 127;
int threshold_max = 200;

cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::Mat cinzentos, filtrado;
cv::cvtColor(imagem, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

cv::createTrackbar("min", "cinzentos", &threshold_min, 255, NULL );
cv::createTrackbar("max", "cinzentos", &threshold_max, 255, NULL );

while (1)
{
    cv::inRange(cinzentos, threshold_min, threshold_max, filtrado);
    cv::imshow("filtrado", filtrado);
    cv::waitKey(30);
}
```

Extração de vídeo

```
// Precisa deste include para a captura
#include <opencv2/opencv.hpp>

cv::Mat frame;
cv::namedWindow("Camara");
cv::VideoCapture cap(0);
while (1)
{
     cap >> frame;
     cv::imshow("Camara", frame);

     if (cv::waitKey(10)>=0)
          break;
}
```



Time

```
cv::waitKey(5000);      // 5 segundos
cv::waitKey(0);         // espera por uma tecla

cv::Mat imagem = cv::imread("i://QtTest//test.jpg", 1);
cv::imshow("Image",imagem);

//  Contar o tempo em segundos:
double duration = 0.0;
duration = static_cast<double>(cv::getTickCount());

salt(imagem,3000); // função que pretendemos medir o tempo

duration = static_cast<double>(cv::getTickCount())- duration;
duration /= cv::getTickFrequency(); // the elapsed time in ms
std::cout << "Duration: " << duration ;

cv::imshow("Image",imagem);
```

Duration: 0.00100884

Time (fps)

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", 1);
cv::imshow("imagem",imagem);

double duration = 0.0;
while (1)
{
        salt(imagem,3000); // função que pretendemos medir o tempo
        cv::imshow("Image",imagem);
        cv::waitKey(10);

        duration = (static_cast<double>(cv::getTickCount()) - duration) / cv::getTickFrequency();
        std::cout << "fps: " << std::setprecision(3) << (1.0/duration);
        std::cout << " Time: " << duration*1000 << std::endl;

        duration = static_cast<double>(cv::getTickCount());     // pronta para o próximo ciclo
}
```

fps: 77.3 Time: 12.9
fps: 76.5 Time: 13.1
fps: 63.2 Time: 15.8
fps: 62.7 Time: 16
fps: 69.2 Time: 14.5
fps: 64.7 Time: 15.5
...

Rato

```
#include <QCursor>

QPoint p = QCursor::pos();
std::cout << p.rx() << "," << p.ry() << std::endl;

int x=p.rx(); int y=p.ry();
std::cout << "xy = " << x << "," << y << std::endl;
```

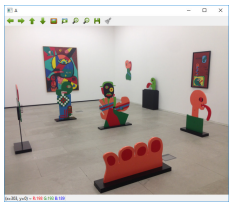
Rato

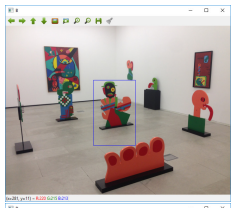
```
static void onMouse( int event, int x, int y, int, void* )
{
    if( event == cv::EVENT_LBUTTONDOWN )
    {
        cv::Point seed = cv::Point(x,y);
        std::cout << "Coord: " << seed.x << ", " << seed.y << std::endl;
        ppx = seed.x; ppy = seed.y;
        flag=1;
    }
}
cv::VideoCapture cap(0);
cv::namedWindow("Frame");
cv::setMouseCallback("Frame", onMouse, 0 );
while (1)
{
    if (flag==1)
    {
        cv::Vec3b pixelRGB=frame.at<cv::Vec3b>(ppy,ppx);
        int R=pixelRGB.val[2];
        int G=pixelRGB.val[1];
        int B=pixelRGB.val[0];
        std::cout << "RGB = " << R << " " << G << " " << B << std::endl;
        flag=0;
    }

    cap >> frame;
    cv::imshow("Frame", frame);
    if (cv::waitKey(5)>=0) break;
}
```

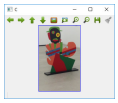
Coord: 3, 2
RGB = 174 190 181
...

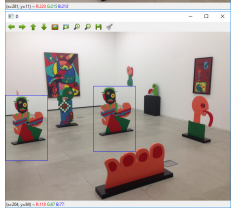
Região de Interesse

```
cv::Mat Original = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::imshow("A", Original);

cv::Rect Rec(250, 154, 120, 187);  // Constroi Rectangle x=100, y=100, larg=200, alt=200
cv::rectangle(Original, Rec, cv::Scalar(255), 1, 8, 0);
            // Scalar Cor, 1 espessura, 8 tipo de linha, 0 shift
cv::imshow("B", Original);

cv::Mat Roi = Original(Rec);   // Seleciona area descrita por REC e escreve em Roi
cv::imshow("C", Roi);

cv::Rect WhereRec(0, 180, Roi.cols, Roi.rows);// Colocar Roi na origem
Roi.copyTo(Original(WhereRec));    // Copia Roi para Imagem Original dado por WhereRec
cv::imshow("D", Original);
```

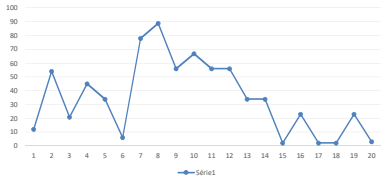






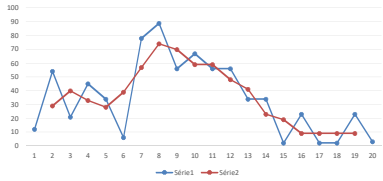


filtros



12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

filtros



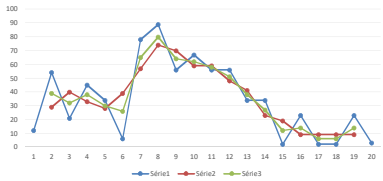
12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

1	1	1

12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

29	40	33	28	39	57	74	70	59	59	48	41	23	19	9	9	9	9		

filtros



12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

1	3	1

12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

39	32	38	30	26	65	80	64	62	58	51	38	27	12	14	6	6	14		

filtros

12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

1	1	1	1	1

12	54	21	45	34	6	78	89	56	67	56	56	34	34	2	23	2	2	23	3

33	32	36	50	52	59	69	64	53	49	36	29	19	12	10	10				

Filtros (2D)

54	44	30	18	73	93	86	98	39	46	52
46	58	86	31	15	18	58	50	99	96	33
15	82	59	53	75	90	31	51	48	46	83
64	64	44	62	61	84	40	83	91	66	85
47	4	88	28	74	80	86	47	78	60	95
43	15	99	59	22	21	4	34	70	49	51
72	30	36	68	8	62	95	97	7	57	99
2	63	59	31	77	85	82	0	83	0	52
80	22	6	85	16	51	13	2	93	36	15
78	96	51	38	35	81	36	95	58	28	12
79	22	64	20	93	53	99	75	88	60	53

1	1	1
1	1	1
1	1	1

1	3	1
3	5	3
1	3	1

53	51	49	52	60	64	62	64	42
58	60	54	54	52	56	61	70	50
52	54	60	67	69	66	62	63	43
52	51	60	55	52	53	59	64	46
48	47	54	47	50	58	58	55	36
47	51	51	48	51	53	52	44	30
41	44	43	54	54	54	52	42	31
51	50	44	55	53	49	51	44	33
55	45	45	52	53	56	62	59	40

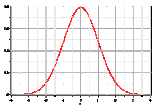
58	55	43	44	53	56	65	68	65
59	62	55	58	59	53	56	66	67
53	57	55	69	71	63	64	71	70
43	55	58	56	63	55	61	68	70
40	58	55	40	42	48	52	53	59
43	53	48	44	53	64	54	49	48
40	43	52	51	65	57	48	43	43
49	43	44	50	51	40	42	54	38
58	48	45	51	55	58	61	65	42

Filtros - *Bluring*

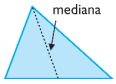
- Averaging

$$F = \frac{1}{K_{largura}\ x\ K_{altura}} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

- Gaussian



- Median



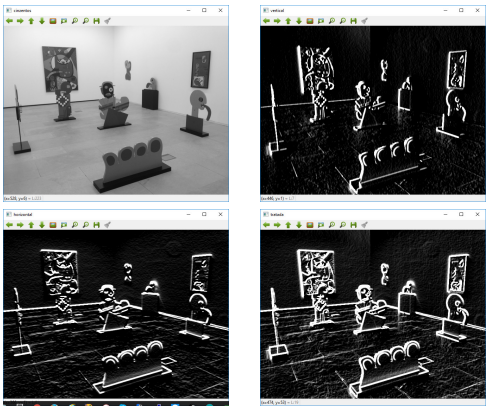
filtros

```
cv::Mat Original = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::Mat cinzentos, horizontal, vertical, tratada;
cv::imshow("Original", Original);

cv::cvtColor(Original, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

cv::Mat V = cv::Mat::zeros( 5, 5, CV_32F);
cv::Mat H = cv::Mat::zeros( 5, 5, CV_32F);

// atribuição das matrizes
// … V.at<float>(2,0) = 4;

cv::filter2D(cinzentos, vertical, 0, V, cv::Point(-1,-1), 0.0, cv::BORDER_DEFAULT);
cv::imshow("vertical", vertical);
cv::filter2D(cinzentos, horizontal, 0, H, cv::Point(-1,-1), 0.0, cv::BORDER_DEFAULT);
cv::imshow("horizontal", horizontal);
cv::add(vertical, horizontal, tratada);
cv::imshow("tratada", tratada);
```

2	1	0	-1	-2
2	1	0	-1	-2
4	2	0	-2	-4
2	1	0	-1	-2
2	1	0	-1	-2

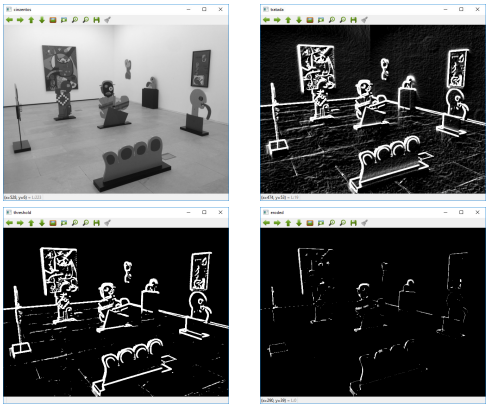
2	2	4	2	2
1	1	2	1	1
0	0	0	0	0
-1	-1	-2	-1	-1
-2	-2	-4	-2	-2

void filter2D(InputArray **src**, OutputArray **dst**, int **ddepth**, InputArray **kernel**, Point **anchor**=Point(-1,-1), double **delta**=0, int **borderType**=BORDER_DEFAULT)



filtros

```
(...)

cv::filter2D(cinzentos, vertical  , 0, V, cv::Point(-1,-1), 0.0, cv::BORDER_DEFAULT);
cv::filter2D(cinzentos, horizontal, 0, H, cv::Point(-1,-1), 0.0, cv::BORDER_DEFAULT);
cv::add(vertical, horizontal, tratada);
cv::imshow("tratada", tratada);

cv::Mat threshold, eroded;
cv::Mat F = cv::Mat::ones( 5, 5, CV_32F);

cv::threshold(tratada, threshold, 127, 255, 3);
                // TYPE = BINARY, BINARY_INV, TRUNC, TOZERO, TOZERO_INV
cv::imshow("threshold", threshold);

cv::erode(threshold, eroded, F);
cv::imshow("eroded", eroded);
```



Filtros - *blur*

```
cv::Mat Original = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::Mat cinzentos, blured3, blured5, blured7;

cv::cvtColor(Original, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

int i;

i=3;
cv::blur(cinzentos, blured3, cv::Size(i,i));
cv::imshow("blured3", blured3);

i=5;
cv::blur(cinzentos, blured5, cv::Size(i,i));
cv::imshow("blured5", blured5);

i=7;
cv::blur(cinzentos, blured7, cv::Size(i,i));
cv::imshow("blured7", blured7);
```

1	1	1
1	1	1
1	1	1

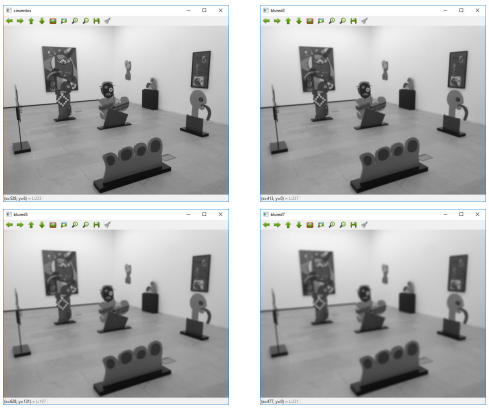
/9

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

/25

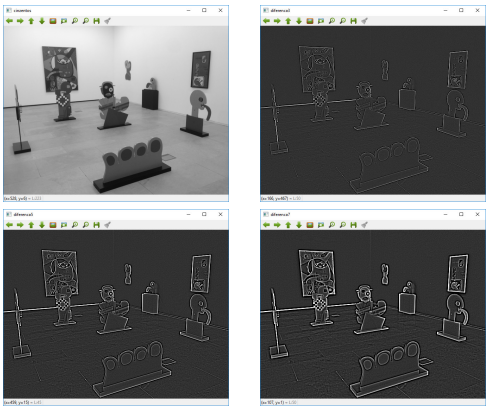
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

/49



Filtros - *blur*

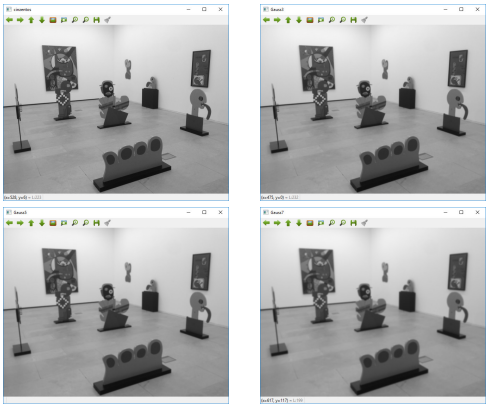
```
cv::Mat diferenca3, diferenca5, diferenca7;

(...)

i=3;
cv::blur(cinzentos, blured3, cv::Size(i,i));

i=5;
cv::blur(cinzentos, blured5, cv::Size(i,i));

i=7;
cv::blur(cinzentos, blured7, cv::Size(i,i));

diferenca3 = 50 + (blured3 - cinzentos)*5;
cv::imshow("diferenca3", diferenca3);

diferenca5 = 50 + (blured5 - cinzentos)*5;
cv::imshow("diferenca5", diferenca5);

diferenca7 = 50 + (blured7 - cinzentos)*5;
cv::imshow("diferenca7", diferenca7);
```



Filtros - *Gaussian*

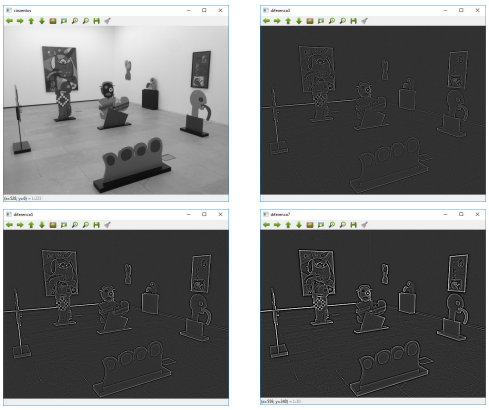
```
cv::Mat Original = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::Mat cinzentos, Gauss3, Gauss5, Gauss7

cv::cvtColor(Original, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

int i;

i=3;
cv::GaussianBlur(cinzentos, Gauss3, cv::Size(i,i), 0, 0, cv::BORDER_DEFAULT);
                 // desvio padrão em X, desvio padrão em Y
cv::imshow("Gauss3", Gauss3);

i=5;
cv::GaussianBlur(cinzentos, Gauss5, cv::Size(i,i), 0, 0, cv::BORDER_DEFAULT);
cv::imshow("Gauss5", Gauss5);

i=7;
cv::GaussianBlur(cinzentos, Gauss7, cv::Size(i,i), 0, 0, cv::BORDER_DEFAULT);
cv::imshow("Gauss7", Gauss7);
```

void GaussianBlur(InputArray **src**, OutputArray **dst**, Size **ksize**,
double **sigmaX**, double **sigmaY**=0, int **borderType**=BORDER_DEFAULT)



Filtros - *Gaussian*

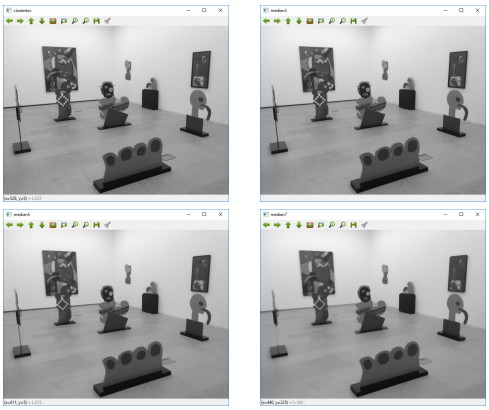
```
cv::Mat cinzentos, diferenca3, diferenca5, diferenca7;

(...)

diferenca3 = 50 + (Gauss3 - cinzentos)*5;
cv::imshow("diferenca3", diferenca3);

diferenca5 = 50 + (Gauss5 - cinzentos)*5;
cv::imshow("diferenca5", diferenca5);

diferenca7 = 50 + (Gauss7 - cinzentos)*5;
cv::imshow("diferenca7", diferenca7);
```



Filtros - *Median*

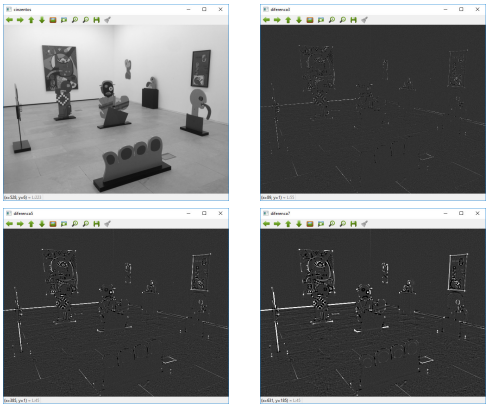
```
cv::Mat Original = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
cv::Mat cinzentos, median3, median5, median7;
//cv::imshow("Original", Original);

cv::cvtColor(Original, cinzentos, cv::COLOR_BGR2GRAY);
cv::imshow("cinzentos", cinzentos);

int i;

i=3;
cv::medianBlur(cinzentos, median3, i);
cv::imshow("median3", median3);

i=5;
cv::medianBlur(cinzentos, median5, i);
cv::imshow("median5", median5);

i=7;
cv::medianBlur(cinzentos, median7, i);
cv::imshow("median7", median7);
```

void medianBlur(InputArray **src**, OutputArray **dst**, int **ksize**)



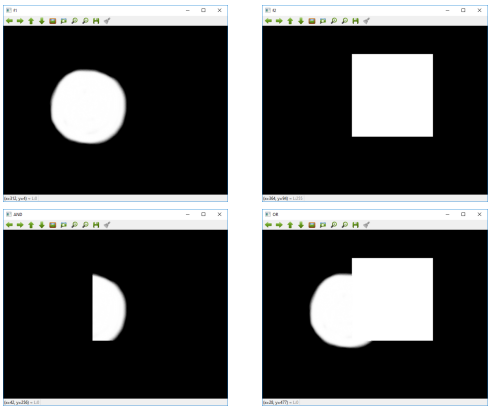
Filtros - *Median*

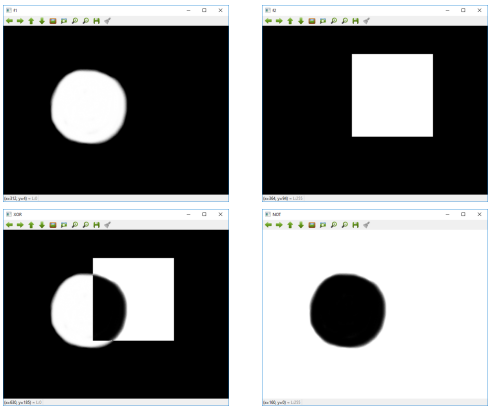
```
cv::Mat cinzentos, diferenca3, diferenca5, diferenca7;

(...)

diferenca3 = 50 + (median3 - cinzentos)*5;
cv::imshow("diferenca3", diferenca3);

diferenca5 = 50 + (median5 - cinzentos)*5;
cv::imshow("diferenca5", diferenca5);

diferenca7 = 50 + (median7 - cinzentos)*5;
cv::imshow("diferenca7", diferenca7);
```



Operações Aritméticas

```
cv::Mat f1 = cv::imread("i://QtTest//f1.png", 1);
cv::Mat f2 = cv::imread("i://QtTest//f2.png", 1);
cv::Mat r_and, r_or, r_xor, r_not;

cv::cvtColor(f1, f1, cv::COLOR_BGR2GRAY);
cv::cvtColor(f2, f2, cv::COLOR_BGR2GRAY);
cv::imshow("f1", f1);
cv::imshow("f2", f2);

cv::bitwise_and(f1,f2,r_and);
cv::bitwise_or(f1,f2,r_or);
cv::bitwise_xor(f1,f2,r_xor);
cv::bitwise_not(f1,r_not);

cv::imshow("AND", r_and);
cv::imshow("OR", r_or);
cv::imshow("XOR", r_xor);
cv::imshow("NOT", r_not);
```

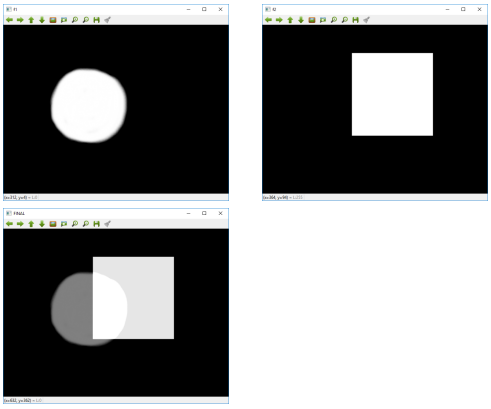




Operações Aritméticas

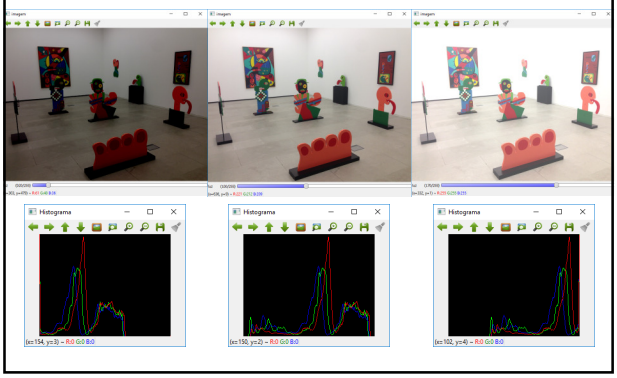
```
cv::Mat f1 = cv::imread("i://QtTest//f1.png", 1);
cv::Mat f2 = cv::imread("i://QtTest//f2.png", 1);
cv::Mat final;

cv::cvtColor(f1, f1, cv::COLOR_BGR2GRAY);
cv::cvtColor(f2, f2, cv::COLOR_BGR2GRAY);
cv::imshow("f1", f1);
cv::imshow("f2", f2);

cv::addWeighted(f1, 0.5, f2, 0.9, 0, final, -1);
cv::imshow("FINAL", final);
```

void addWeighted(InputArray **src1**, double **alpha**, InputArray **src2**, double **beta**, double **gamma**, OutputArray **dst**, int **dtype**=-1)



Histogramas

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png", cv::IMREAD_COLOR);
std::vector<cv::Mat> pl_RGB;

cv::imshow("imagem", imagem);
cv::split(imagem, pl_RGB);

int h_tamanho = 256;
float gama[] = { 0, 256 } ;
const float* h_gama = { gama };
cv::Mat h_B, h_G, h_R;
cv::calcHist( &pl_RGB[0], 1, 0, cv::Mat(), h_B, 1, &h_tamanho, &h_gama, 1, 0);
cv::calcHist( &pl_RGB[1], 1, 0, cv::Mat(), h_G, 1, &h_tamanho, &h_gama, 1, 0);
cv::calcHist( &pl_RGB[2], 1, 0, cv::Mat(), h_R, 1, &h_tamanho, &h_gama, 1, 0);

int h_largura = 256; int h_altura = 100;
int racio = cvRound( (double) h_largura/h_tamanho );
cv::Mat imagem_h( h_altura, h_largura, CV_8UC3, cv::Scalar( 0,0,0) );
cv::normalize(h_B, h_B, 0, imagem_h.rows, cv::NORM_MINMAX, -1, cv::Mat());
cv::normalize(h_G, h_G, 0, imagem_h.rows, cv::NORM_MINMAX, -1, cv::Mat());
cv::normalize(h_R, h_R, 0, imagem_h.rows, cv::NORM_MINMAX, -1, cv::Mat());

for( int i = 1; i <= h_tamanho; i++ )
{
    cv::line( imagem_h, cv::Point( racio*(i-1), h_altura - cvRound(h_B.at<float>(i-1)) ) ,
              cv::Point(racio*(i), h_altura-cvRound(h_B.at<float>(i) ), cv::Scalar(255,0,0), 1, 8, 0  );
    cv::line( imagem_h, cv::Point( racio*(i-1), h_altura - cvRound(h_G.at<float>(i-1)) ) ,
              cv::Point(racio*(i), h_altura-cvRound(h_G.at<float>(i))), cv::Scalar(0,255,0), 1, 8, 0  );
    cv::line( imagem_h, cv::Point( racio*(i-1), h_altura - cvRound(h_R.at<float>(i-1)) ) ,
              cv::Point(racio*(i), h_altura-cvRound(h_R.at<float>(i))), cv::Scalar(0,0,255), 1, 8, 0  );
}
cv::imshow("Histograma", imagem_h );
```



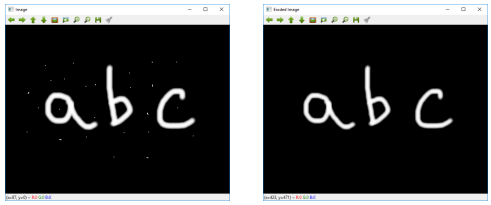
Histogramas de posição

```
cv::Mat image, R, G, B;
cv::VideoCapture cap(0);
std::vector<cv::Mat> planos;
std::vector<int> histR(640), histG(640), histB(640);

while(1)
{
    cap >> image;
    cv::split (image, planos);

    R = planos[2]-planos[1]-planos[0];  // Só vermelho
    G = planos[1]-planos[2]-planos[0];  // Só verde
    B = planos[0]-planos[1]-planos[2];  // Só azul

    for(int i=0; i<image.cols; i++)
    {
       histR[i] = histG[i] = histB[i] = 0;
       for(int j=0; j<image.rows; j++)
       {
           if (B.at<uchar>(j,i) > 0) histB[i]++;
           if (G.at<uchar>(j,i) > 0) histG[i]++;
           if (R.at<uchar>(j,i) > 0) histR[i]++;
       }
       histB[i] /=4; histG[i] /=4; histR[i] /=4;
    }

    int hz = 100;   // linha do horizonte
    cv::Mat temp(hz*3,image.cols,CV_8UC3,cv::Scalar(0, 0, 0));
    for(int i=0; i<image.cols; i++)
    {
       line(temp, cv::Point(i, hz)  , cv::Point(i, hz  -cvRound(histR[i])), cv::Scalar(0, 0, 255), 1, 8, 0);
       line(temp, cv::Point(i, hz*2), cv::Point(i, hz*2-cvRound(histG[i])), cv::Scalar(0, 255, 0), 1, 8, 0);
       line(temp, cv::Point(i, hz*3), cv::Point(i, hz*3-cvRound(histB[i])), cv::Scalar(255, 0, 0), 1, 8, 0);
    }
    cv::imshow("image", image);
    cv::imshow("temp", temp);
}
```

CORRER O TESTE 12

Operações Morfológicas

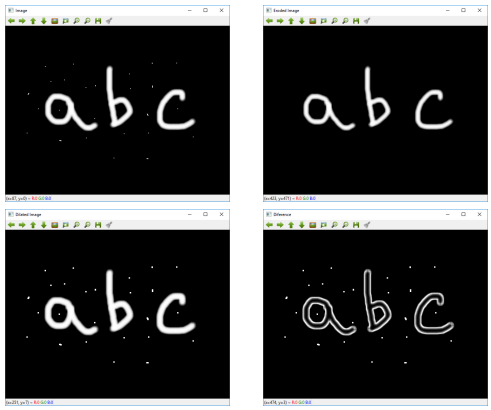
- EROSÃO
- DILATAÇÃO

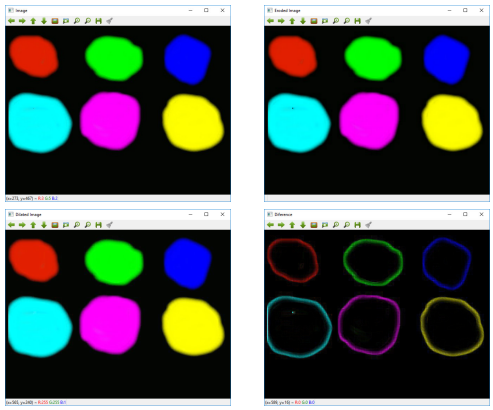


Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//cores.jpg");
cv::Mat eroded, dilated, final;

cv::imshow("Image",image);

cv::erode(image,eroded,cv::Mat());
cv::imshow("Eroded Image",eroded);

cv::dilate(image,dilated,cv::Mat());
cv::imshow("Dilated Image",dilated);

final = (dilated - eroded)*2;
cv::imshow("Diference",final);
```

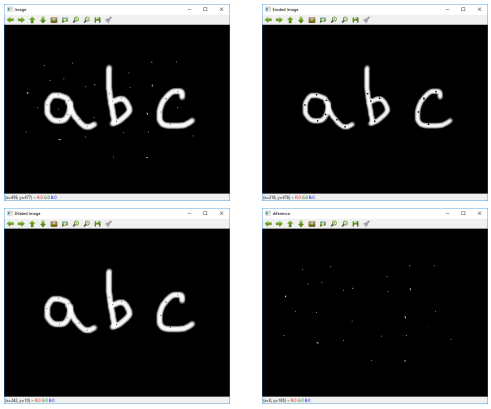
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1





Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//nome2.png"); // open the image
cv::Mat eroded, dilated, diferenca;

cv::imshow("Image",image);

cv::erode(image,eroded,cv::Mat());
cv::imshow("Eroded Image",eroded);

cv::dilate(eroded,dilated,cv::Mat());
cv::imshow("Dilated Image",dilated);

diferenca = image - dilated;    // O inverso dá tudo preto
cv::imshow("diferenca", diferenca);
```



Operações Morfológicas

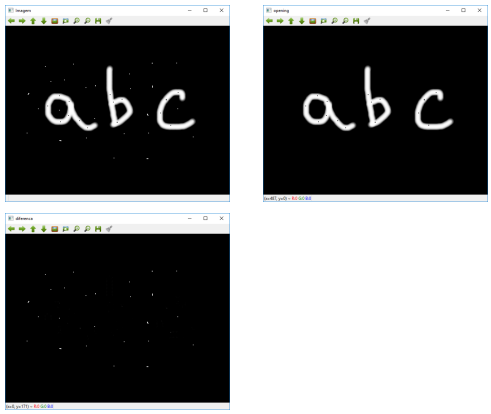
Outras operações compostas

OPENING – Erosão + dilatação – remove ruido
CLOSING – Dilatação + erosão – preenche imperfeições
MORPHOLOGICAL GRADIENT – Dilatação – erosão – perfil do objecto
TOP HAT – imagem_original - Opening
BLACK HAT – Closing - imagem_original

Tem elemento estruturante (matriz)

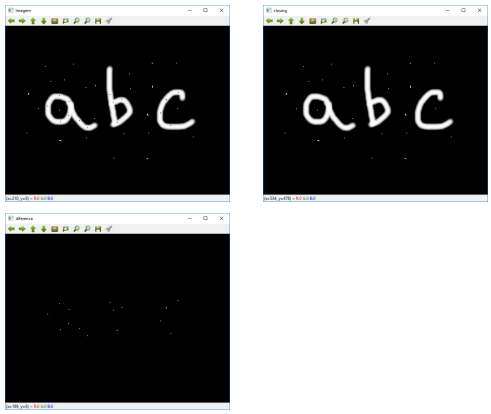
Operações Morfológicas

```
cv::Mat imagem = cv::imread("i://QtTest//nome2.png"); // open the image
cv::Mat opening, diferenca;

cv::imshow("Imagem",imagem);

cv::Mat E(3, 3, CV_8U,cv::Scalar(1)); // Matriz de 7 x 7

cv::morphologyEx(imagem, opening, cv::MORPH_OPEN, E);
cv::imshow("opening", opening);

diferenca = imagem - opening;
cv::imshow("diferenca", diferenca);
```



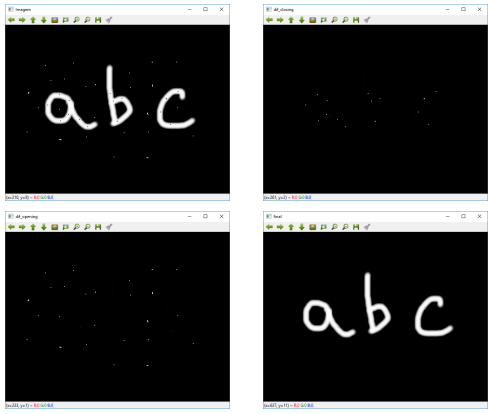
Operações Morfológicas

```
cv::Mat imagem = cv::imread("i://QtTest//nome2.png"); // open the image
cv::Mat closing, diferenca;

cv::imshow("Imagem",imagem);

cv::Mat E(3, 3, CV_8U,cv::Scalar(1)); // Matriz de 7 x 7

cv::morphologyEx(imagem, closing, cv::MORPH_CLOSE, E);
cv::imshow("closing", closing);

diferenca = closing- imagem;
cv::imshow("diferenca", diferenca);
```



Operações Morfológicas

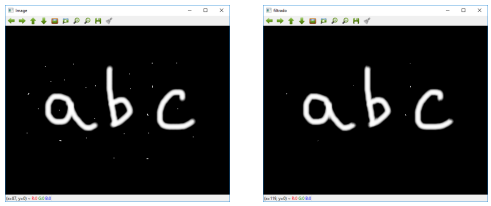
```
cv::Mat imagem = cv::imread("i://QtTest//nome2.png"); // open the image
cv::Mat closing, opening, dif_closing, dif_opening, final;

cv::imshow("Imagem",imagem);

cv::Mat E(3, 3, CV_8U,cv::Scalar(1)); // Matriz de 7 x 7

cv::morphologyEx(imagem, closing, cv::MORPH_CLOSE, E);
cv::morphologyEx(imagem, opening, cv::MORPH_OPEN, E);
cv::imshow("closing", closing);
cv::imshow("opening", opening);

dif_closing = closing - imagem;
dif_opening = imagem - opening;

final = imagem - dif_opening + dif_closing;

cv::imshow("final", final);
```



Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//nome1.png"); // open the image
cv::Mat eroded, dilated, final, filtrado;

cv::imshow("Image",image);

cv::Mat E(7, 7, CV_8U,cv::Scalar(1)); // Matriz de 7 x 7
E.at<float>(0,0)= 0; E.at<float>(0,1)= 0; E.at<float>(0,5)= 0; E.at<float>(0,6)= 0;
E.at<float>(1,0)= 0;                                           E.at<float>(1,6)= 0;
E.at<float>(5,0)= 0;                                           E.at<float>(5,6)= 0;
E.at<float>(6,0)= 0; E.at<float>(6,1)= 0; E.at<float>(6,5)= 0; E.at<float>(6,6)= 0;

cv::morphologyEx(image, filtrado, cv::MORPH_CLOSE, E);
cv::imshow("filtrado", filtrado);

 // Operação:
        // - Opening: MORPH_OPEN : 2
        // - Closing: MORPH_CLOSE: 3
        // - Gradient: MORPH_GRADIENT: 4
        // - Top Hat: MORPH_TOPHAT: 5
        // - Black Hat: MORPH_BLACKHAT: 6
```

$$\begin{matrix} 0&0&1&1&1&0&0 \\ 0&1&1&1&1&1&0 \\ 1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1 \\ 0&1&1&1&1&1&0 \\ 0&0&1&1&1&0&0 \end{matrix}$$

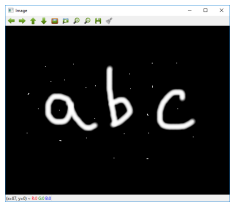


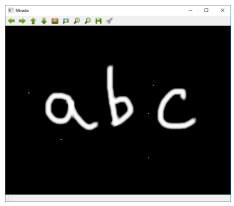
Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//nome1.png"); // open the image
cv::Mat eroded, dilated, final, filtrado;

cv::imshow("Image",image);

cv::Mat E(7, 7, CV_8U,cv::Scalar(0)); // Matriz de 7 x 7
for (int i=0;i<7;i++)
{
    E.at<float>(3,i)= 1;
    E.at<float>(i,3)= 1;
}

cv::morphologyEx(image, filtrado, cv::MORPH_CLOSE, E);
cv::imshow("filtrado", filtrado);
```

			1			
			1			
			1			
1	1	1	1	1	1	1
			1			
			1			
			1			

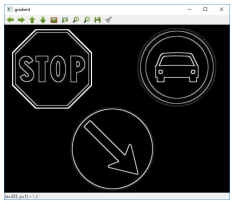
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
1	1	1	1	1	1	1
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0

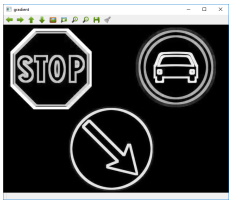


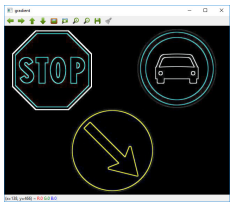


Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//sinais.jpg");
cv::Mat gradient, diferenca;
cv::cvtColor(image, image, cv::COLOR_BGR2GRAY);
cv::imshow("Image",image);

cv::Mat E(3, 3, CV_8U,cv::Scalar(1));  // tamanho 3x3

cv::morphologyEx(image, gradient, cv::MORPH_GRADIENT, E, cv::Point(-1,-1), 1,
cv::BORDER_CONSTANT);
cv::imshow("gradient", gradient);

diferenca = image - gradient;
cv::imshow("diferenca", diferenca);
```



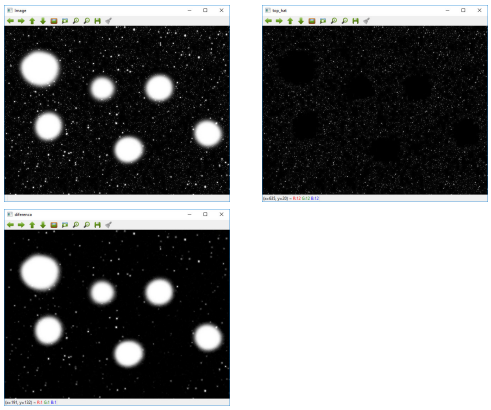




Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//top_hat.jpg"); // open the image
cv::Mat top_hat, diferenca;
cv::imshow("Image",image);

cv::Mat E(3, 3, CV_8U,cv::Scalar(1));

cv::morphologyEx( image, top_hat, cv::MORPH_TOPHAT, E );
        // Operação:
        // - Opening: MORPH_OPEN : 2
        // - Closing: MORPH_CLOSE: 3
        // - Gradient: MORPH_GRADIENT: 4
        // - Top Hat: MORPH_TOPHAT: 5
        // - Black Hat: MORPH_BLACKHAT: 6
cv::imshow("top_hat", top_hat);
diferenca = image - top_hat;
cv::imshow("diferenca", diferenca);
```

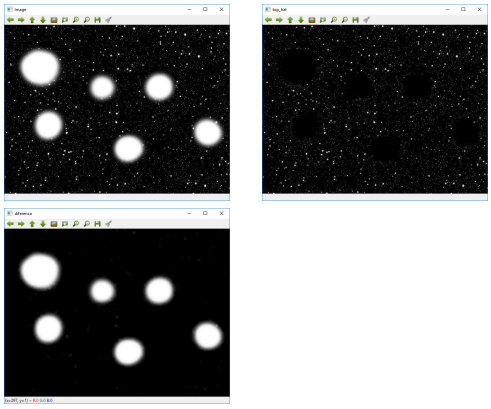
1	1	1
1	1	1
1	1	1



Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//top_hat.jpg"); // open the image
cv::Mat top_hat, diferenca;
cv::imshow("Image",image);

cv::Mat E(7, 7, CV_8U,cv::Scalar(1));

cv::morphologyEx( image, top_hat, cv::MORPH_TOPHAT, E );
        // Operação:
        // - Opening: MORPH_OPEN : 2
        // - Closing: MORPH_CLOSE: 3
        // - Gradient: MORPH_GRADIENT: 4
        // - Top Hat: MORPH_TOPHAT: 5
        // - Black Hat: MORPH_BLACKHAT: 6
cv::imshow("top_hat", top_hat);
diferenca = image - top_hat;
cv::imshow("diferenca", diferenca);
```

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1



Operações Morfológicas

```
cv::Mat image = cv::imread("i://QtTest//top_hat2.jpg"); // open the image
cv::Mat top_hat, diferenca;
cv::cvtColor(image, image, cv::COLOR_BGR2GRAY);
cv::imshow("Image",image);

cv::Mat E = cv::Mat::ones(7, 7, CV_8U);
ReadMatToFile(E, "i://QtTest//mat7.txt");

cv::morphologyEx(image, top_hat, cv::MORPH_TOPHAT, E, cv::Point(-1,-1), 1,
cv::BORDER_CONSTANT);
cv::imshow("top_hat", top_hat);

diferenca = image - top_hat;
cv::imshow("diferenca", diferenca);
```

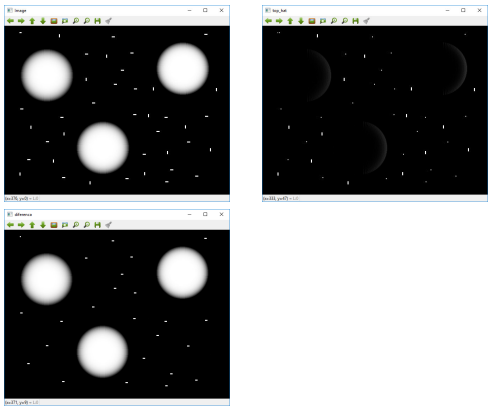
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

0	0	0	0	0	0	0
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Ler ficheiro de texto para matriz

```
void ReadMatToFile(cv::Mat& temp, const char* nome)
{
    std::ifstream ficheiro(nome);
    if(!ficheiro)
    {
        std::cout<<"Ficheiro nao aberto"<<std::endl;   return;
    }
    for(int i=0; i<temp.rows; i++)
    {
        for(int j=0; j<temp.cols; j++)
            ficheiro >> temp.at<float>(i,j);
    }
    ficheiro.close();
}


cv::Mat E = cv::Mat::ones(7, 7, CV_8U);
ReadMatToFile(E, "i://QtTest//mat7.txt");
```

0	0	0	0	0	0	0
0	0	0	0	0	0	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

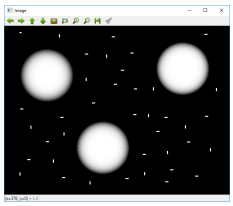


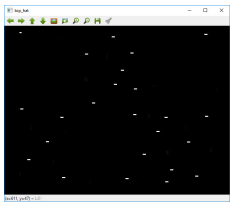
Operações Morfológicas

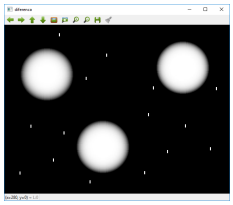
```
cv::Mat image = cv::imread("i://QtTest//top_hat2.jpg"); // open the image
cv::Mat top_hat, diferenca;
cv::cvtColor(image, image, cv::COLOR_BGR2GRAY);
cv::imshow("Image",image);

cv::Mat E = cv::Mat::ones(7, 7, CV_32F);
ReadMatToFile(E, "i://QtTest//mat7.txt");
std::cout << E ;

cv::Mat F;
E.convertTo(F, CV_8U);
std::cout << F ;

cv::morphologyEx(image, top_hat, cv::MORPH_TOPHAT, F, cv::Point(-1,-1), 1,
cv::BORDER_CONSTANT);
cv::imshow("top_hat", top_hat);

diferenca = image - top_hat;
cv::imshow("diferenca", diferenca);
```

		1	1	1		
		1	1	1		
		1	1	1		
		1	1	1		
		1	1	1		
		1	1	1		
		1	1	1		

0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0
0	0	1	1	1	0	0





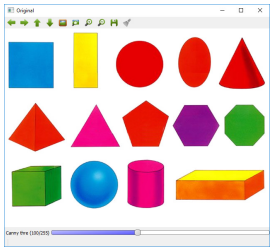


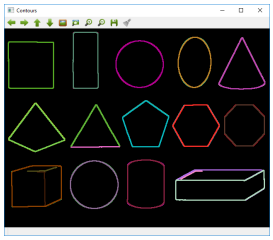
Contour

```
cv::Mat original, cinzentos;
int thresh = 100;
cv::RNG rng(12345);

void thresh_callback(int, void* )
{
   cv::Mat canny;
   std::vector<std::vector<cv::Point> > contours;
   std::vector<cv::Vec4i> hierarchy;
   cv::Canny(cinzentos, canny, thresh, thresh*2, 3 );
   cv::FindContours(canny, contours, hierarchy, cv::RETR_TREE, cv::CHAIN_APPROX_SIMPLE, cv::Point(0,0));
   cv::Mat drawing = cv::Mat::zeros( canny.size(), CV_8UC3 );
   for( size_t i = 0; i< contours.size(); i++ )
   {
      cv::Scalar color = cv::Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
      cv::drawContours( drawing, contours, (int)i, color, 2, 8, hierarchy, 0, cv::Point() );
   }
   cv::namedWindow("Contours", cv::WINDOW_AUTOSIZE );
   cv::imshow( "Contours", drawing );
}

      original = cv::imread("i://QtTest//geometricas2.png", 1);
      cv::cvtColor( original, cinzentos, cv::COLOR_BGR2GRAY );
      cv::blur( cinzentos, cinzentos, cv::Size(3,3) );
      cv::namedWindow("Original", cv::WINDOW_AUTOSIZE );
      cv::imshow("Original", original );
      cv::createTrackbar("Canny thresh:", "Original", &thresh, 255, thresh_callback );
      thresh_callback( 0, 0 );
      cv::waitKey(0);
```

Contour

```
void cv::findContours (Matriz          image,
                       ArrayOfArrays   contours,
                       OutputArray     hierarchy,
                       int             mode,
                       int             method,
                       Point           offset = Point()
)
```



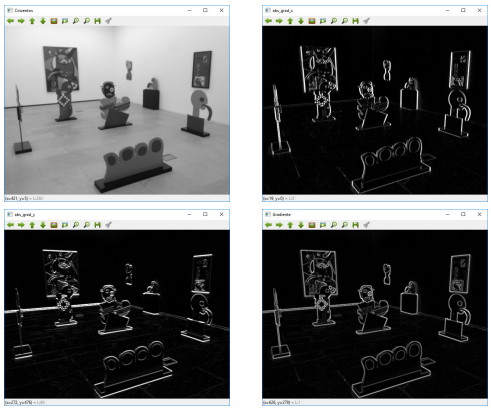


Sobel

```
cv::Mat imagem = cv::imread("i://QtTest//jg.png");
cv::Mat cinzentos;
cv::imshow("Imagem",imagem);

cv::GaussianBlur( imagem, imagem, cv::Size(3,3), 0, 0, cv::BORDER_DEFAULT );
cv::cvtColor( imagem, cinzentos, CV_BGR2GRAY );
cv::imshow("Cinzentos", cinzentos);

cv::Mat grad_x, grad_y;
cv::Mat img_grad_x, img_grad_y;

// Gradient X
cv::Sobel( cinzentos, grad_x, CV_16S, 1, 0, 3, 1, 0, cv::BORDER_DEFAULT );
cv::convertScaleAbs( grad_x, img_grad_x );
cv::imshow("abs_grad_x", img_grad_x);

// Gradient Y
cv::Sobel( cinzentos, grad_y, CV_16S, 0, 1, 3, 1, 0, cv::BORDER_DEFAULT );
cv::convertScaleAbs( grad_y, img_grad_y );
cv::imshow("abs_grad_y", img_grad_y);

// Gradient
cv::Mat gradiente;
cv::addWeighted( img_grad_x, 0.5, img_grad_y, 0.5, 0, gradiente );
cv::imshow("Gradiente", gradiente);
```

Sobel

```
void cv::Sobel  (Matriz  src,
                 Matriz  dst,
                 int     ddepth,
                 int     dx,
                 int     dy,
                 int     ksize = 3,
                 double  scale = 1,
                 double  delta = 0,
                 int     borderType = BORDER_DEFAULT
)
```



Canny

```
int top_slider = 50;
int top_slider_max = 200;

cv::Mat image, border;

void on_trackbar_canny(int, void*)
{
  cv::Canny(image, border, top_slider, 3*top_slider);
  cv::imshow("canny", border);
}
```

```
image = cv::imread("i://QtTest//jg-gray.png");
cv::namedWindow("canny", 1);

cv::createTrackbar("nome", "canny", &top_slider, top_slider_max, on_trackbar_canny );
on_trackbar_canny(top_slider, 0 );
```

Canny

Vantagens:
• Low error rate: Meaning a good detection of only existent edges.
• Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
• Minimal response: Only one detector response per edge.

Passos:
- Filter out any noise. The Gaussian filter is used for this purpose
- Find the intensity gradient of the image. For this, we follow a procedure analogous to Sobel:

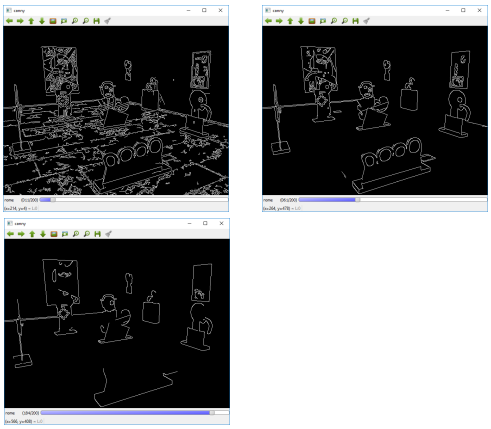
a) Apply a pair of convolution masks (in x and y directions:
b) Find the gradient strength and direction with:

- Non-maximum suppression is applied. This removes pixels that are not considered to be part of an edge.
- Hysteresis: The final step. Canny does use two thresholds (upper and lower):
  (a) If a pixel gradient is higher than the upper threshold, the pixel is accepted as an edge
  (b) If a pixel gradient value is below the lower threshold, then it is rejected.

(c) If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the upper threshold.

https://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

```
void cv::Canny  (Matriz   image,
                 Matriz   edges,
                 double   threshold1,    // histerese
                 double   threshold2,    // histerese
                 int      apertureSize = 3,    // abertura para o sobel
                 bool     L2gradient = false
)
```



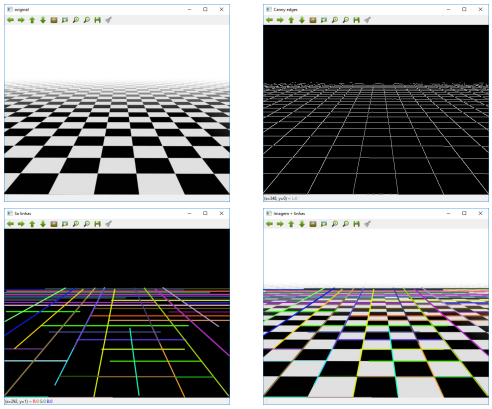
Hough Lines

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

Hough Lines

```
Mat imagem = cv::imread("i://QtTest//quadriculado.jpg", 0);
cv::resize(imagem, imagem, cv::Size(640,480), 0, 0, INTER_LINEAR);
cv::imshow("original", imagem);

// Canny edge
cv::Mat edges;
cv::Canny(imagem, edges, 95, 100);
cv::imshow("Canny edges", edges);

vector<Vec4i> linhas;
HoughLinesP(edges, linhas, 1, CV_PI / 180, 100, 100, 10); // Find hough lines

// Prepara matriz com as mesmas dimensões da original
Mat imagem_linhas(imagem.rows, imagem.cols, CV_8UC3, Scalar(0, 0, 0));

cv::cvtColor(imagem, imagem, CV_GRAY2BGR);
// Desenhas as linhas
for (size_t i = 0; i < linhas.size(); i++)
{
    Vec4i l = linhas[i];
    cv::Scalar cor(rand() % 255,rand() % 255,rand() % 255);
    line(imagem, Point(l[0], l[1]), Point(l[2], l[3]), cor, 2, CV_AA);
    line(imagem_linhas, Point(l[0], l[1]), Point(l[2], l[3]), cor, 2, CV_AA);
}
imshow("So linhas", imagem_linhas);
imshow("Imagem + linhas", imagem);     // imagem original com as linhas por cima
```

```
void cv::HoughLines (Matriz  image,
                     Matriz  lines,
                     double  rho,
                     double  theta,
                     int threshold,
                     double  srn = 0,
                     double  stn = 0,
                     double  min_theta = 0,
                     double  max_theta = CV_PI
)
```

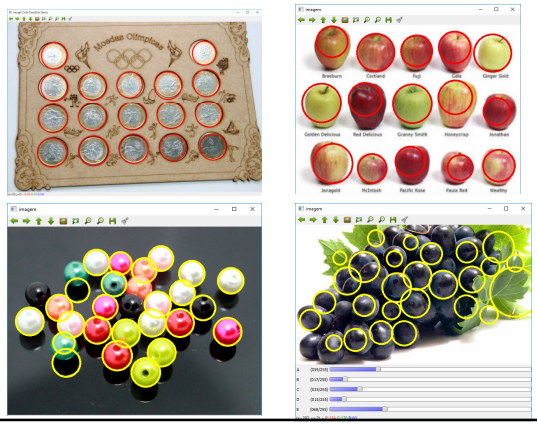


Hough Circles

```
HoughCircles(gray, circles, HOUGH_GRADIENT, 1, gray.rows/16,// change this value to detect circles with different distances to each other100, 30, 1, 30// cha
```

https://docs.opencv.org/trunk/d4/d70/tutorial_hough_circle.html

Hough Circles

```
cv::Mat imagem, gray, gaussian;
imagem = cv::imread( "i://QtTest//moedas2.jpg", 1 );
cv::cvtColor( imagem, gray, CV_BGR2GRAY );

// Reduce the noise so we avoid false circle detection
cv::GaussianBlur( gray, gaussian, cv::Size(7, 7), 2, 2 );

std::vector<cv::Vec3f> circles;
cv::HoughCircles( gaussian, circles, CV_HOUGH_GRADIENT, 1, 30, 200, 50, 0, 100 );
            // HOUGH_STANDARD, HOUGH_PROBABILISTIC,
                HOUGH_MULTI_SCALE, HOUGH_GRADIENT
            // 1
            // distancia minima entre circulos
            // threshold1, threshold2, raio_minimo, raio_máximo

for( size_t i = 0; i < circles.size(); i++ )    // Desenha os circulos detectados
{
    cv::Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
    int radius = cvRound(circles[i][2]);
    cv::circle( imagem, center, radius, cv::Scalar(0,0,255), 3, 8, 0 );// circle outline
}

cv::imshow("Hough Circle Transform Demo", imagem);
```

```
void cv::HoughCircles (Matriz  image,
                       Matriz  circles,
                       int     method,
                       double  dp,
                       double  minDist,
                       double  param1 = 100,
                       double  param2 = 100,
                       int     minRadius = 0,
                       int     maxRadius = 0
)
```