QtCon BR

Visão Computacional com QtQuick e OpenCV

Caio Jordão Carvalho
caiojcarvalho@gmail.com
https://carvalho.site

whoami

Análise e Desenvolvimento de Sistemas - **IFBA**
Back-end Developer no **Escavador**
Contribuidor na comunidade **KDE**
Google Summer of Code 2018

Our Agenda

Visão Computacional

Reconhecimento de Objetos

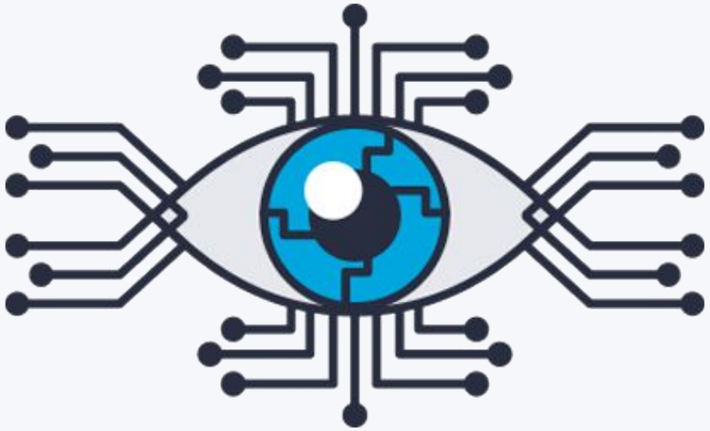
OpenCV

Integrando Qt e OpenCV

QtCon BR


Visão Computacional

Visão Computacional

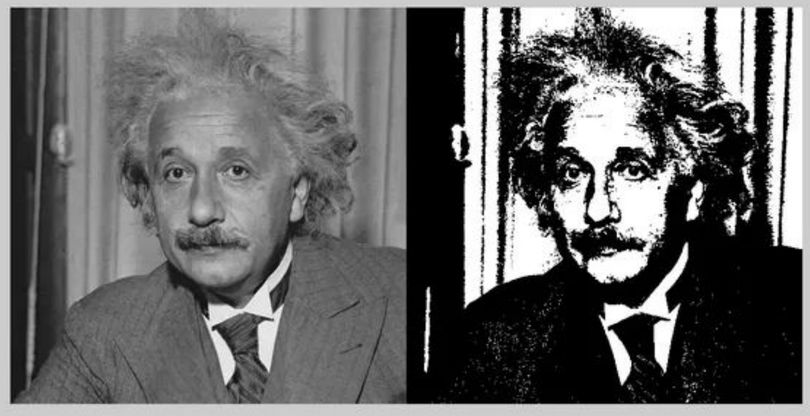


1. Ciência que obtém informações de imagens ou dados multi-dimensionais.

2. Reconhecimento de objetos, estimativa de movimento, restauração e reconstrução de imagens.

QtCon BR

Processamento de Imagens



1. Necessidade de conversão da imagem antes da análise.

2. Aplicação de filtros para remoção de ruídos e/ou interferências, uso de escala de cinza, redimensionamento.

Computer Vision is the future!

QtCon BR

Reconhecimento de Objetos

QtCon BR

Reconhecimento de Objetos

- Sub-área da Visão Computacional que se preocupa em detectar e classificar objetos presentes em imagens.

- Uso de Reconhecimento de Padrões.

Fases do Reconhecimento




Captura → Tratamento → Detecção → Classificação

QtCon BR

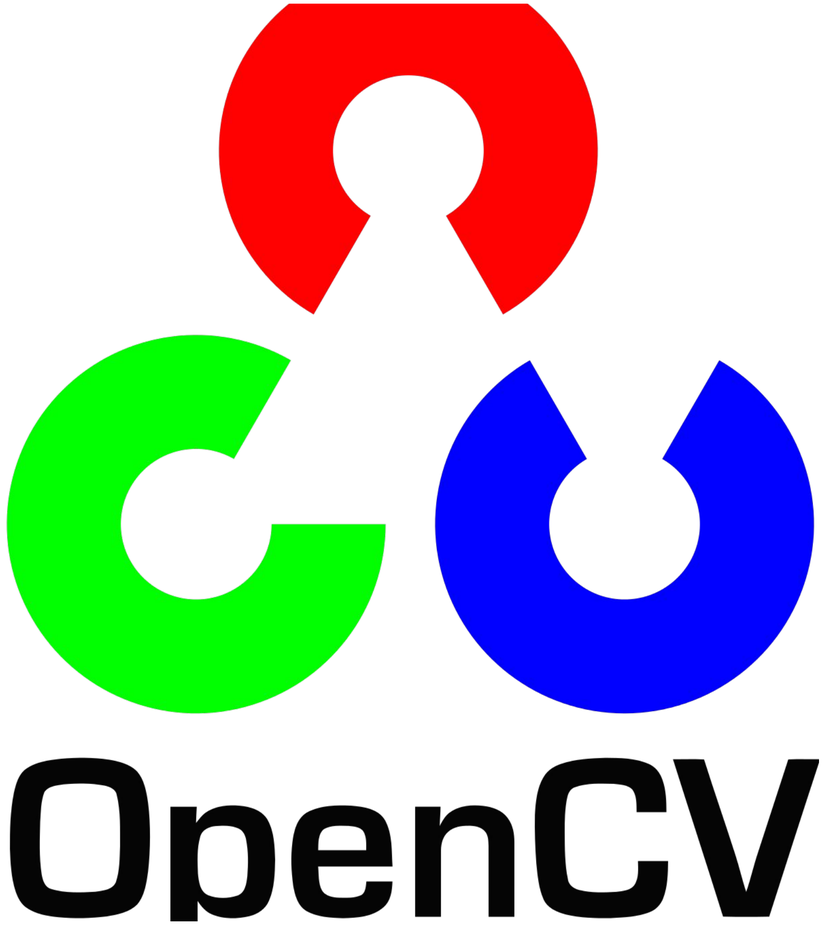
Alguns Algoritmos

- Viola-Jones.
- Histograms of Oriented Gradients.
- Deep Learning.
- ...


OpenCV

OpenCV

- Open Source Computer Vision Library.
- Intel, 2000.
- BSD.
- 3.4
- Módulos de Processamento de Imagens e Vídeos, Estruturas de Dados para Álgebra Linear e GUIs.



Principais Módulos

- core
- highgui
- imgproc
- objdetect
- videoio
- ml
- gpu
- dnn

Integrando Qt e OpenCV

QtCon BR

cv::Mat

- n-dimensional array
- Uma das principais estruturas do OpenCV.
- Utilizado principalmente para representação de imagens.
- InputArray, OutputArray.

Relação entre cv::Mat e QImage

- QImage(uchar *data, int width, int height, QImage::Format format);
- Mat (int rows, int cols, int type, void *data, size_t step);

cv::Mat -> QImage

1. Mat mat = imread(“test.png”);
2. cvtColor(mat, mat, CV_BGR2RGB);
3. QImage image(mat.data, mat.cols, mat.rows, QImage::Format_RGB888);

QImage -> cv::Mat

1. QImage im(“test.png”);
2. im = im.convertToFormat(QImage::Format_RGB888);
3. Mat mat(im.height(), im.width(), CV_8UC(3), im.bits(), im.bitsPerLine());

Linkagem

- INCLUDEPATH += /usr/include/opencv
- LIBS += -L/usr/lib64 -lopencv_core -lopencv_imgproc -lopencv_highgui -lopencv_ml -lopencv_videoio -lopencv_features2d -lopencv_calib3d -lopencv_objdetect -lopencv_flann -lopencv_imgcodecs

Exemplo QML + OpenCV

- ImageProcessor
- ImageViewer
- https://github.com/cjlcarvalho/qml-recognition

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

class ImageProcessor : public QObject {

public:
    ImageProcessor();
    Q_INVOKABLE void processImage(const QString &path);

signals:
    void imageProcessed(const QImage& image);
}
```

QtCon BR

```cpp
void ImageProcessor::processImage(const QString& path)
{
    Mat im = cv::imread(path.toStdString());
    Mat gray;

    if (!im.empty()) {
        cv::cvtColor(im, gray, CV_BGR2GRAY);
        CascadeClassifier fCascade("haarcascade_frontalface_default.xml");

        vector<cv::Rect> faces;

        fCascade.detectMultiScale(gray, faces, 1.3, 5);

        for (Rect &face : faces)
            rectangle(im, face, Scalar(255, 0, 0), 2);

        QImage resultedImage(im.data, im.cols, im.rows, im.step, QImage::Format_RGB888);

        emit imageProcessed(resultedImage);
    }
}
```

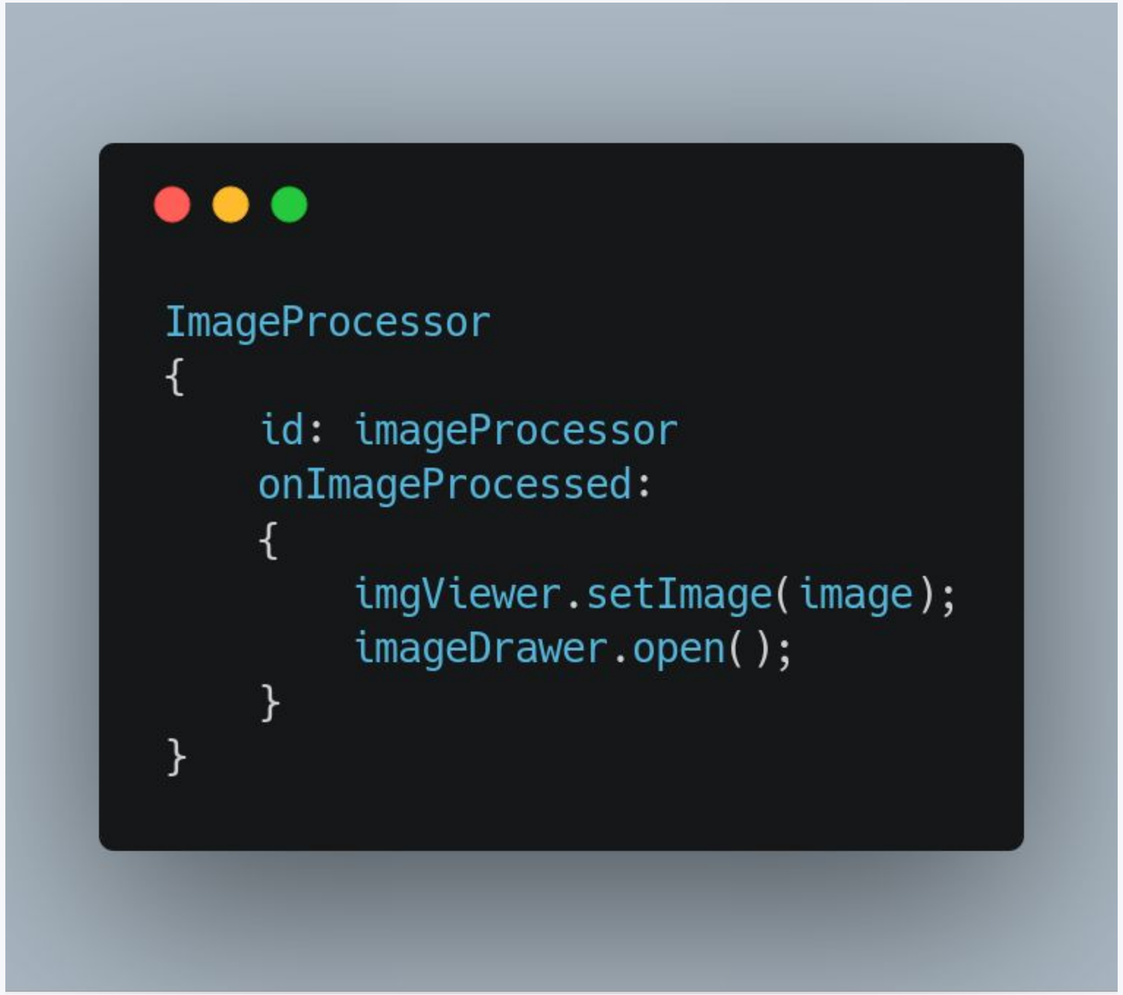
```cpp
#include <QQuickItem>
#include <QQuickPaintedItem>
#include <QImage>
#include <QPainter>

class ImageViewer : public QQuickPaintedItem
{
    Q_OBJECT
public:
    ImageViewer(QQuickItem *parent = Q_NULLPTR);
    Q_INVOKABLE void setImage(const QImage &image);

private:
    void paint(QPainter *painter);

private:
    QImage currentImage;
};
```

```cpp
#include "imageviewer.h"

ImageViewer::ImageViewer(QQuickItem *parent) :
    QQuickPaintedItem(parent)
{
}

void ImageViewer::setImage(const QImage& image)
{
     currentImage = image.copy();
     update();
}

void ImageViewer::paint(QPainter *painter)
{
     QSizeF scaled = QSizeF(currentImage.width(), currentImage.height())
                     .scaled(boundingRect().size(), Qt::KeepAspectRatio);

     QRect centerRect(qAbs(scaled.width() - width()) / 2.0,
                      qAbs(scaled.height() - height()) / 2.0,
                      scaled.width(), scaled.height());
     painter->drawImage(centerRect, currentImage);
}
```

```
qmlRegisterType<ImageProcessor>("com.caio.classes", 1, 0, "ImageProcessor");
qmlRegisterType<ImageViewer>("com.caio.classes", 1, 0, "ImageViewer");
```


Camera
{
id: camera
imageCapture
{
onImageSaved:
{
imgProcessor.processImage(path);
}
}
}

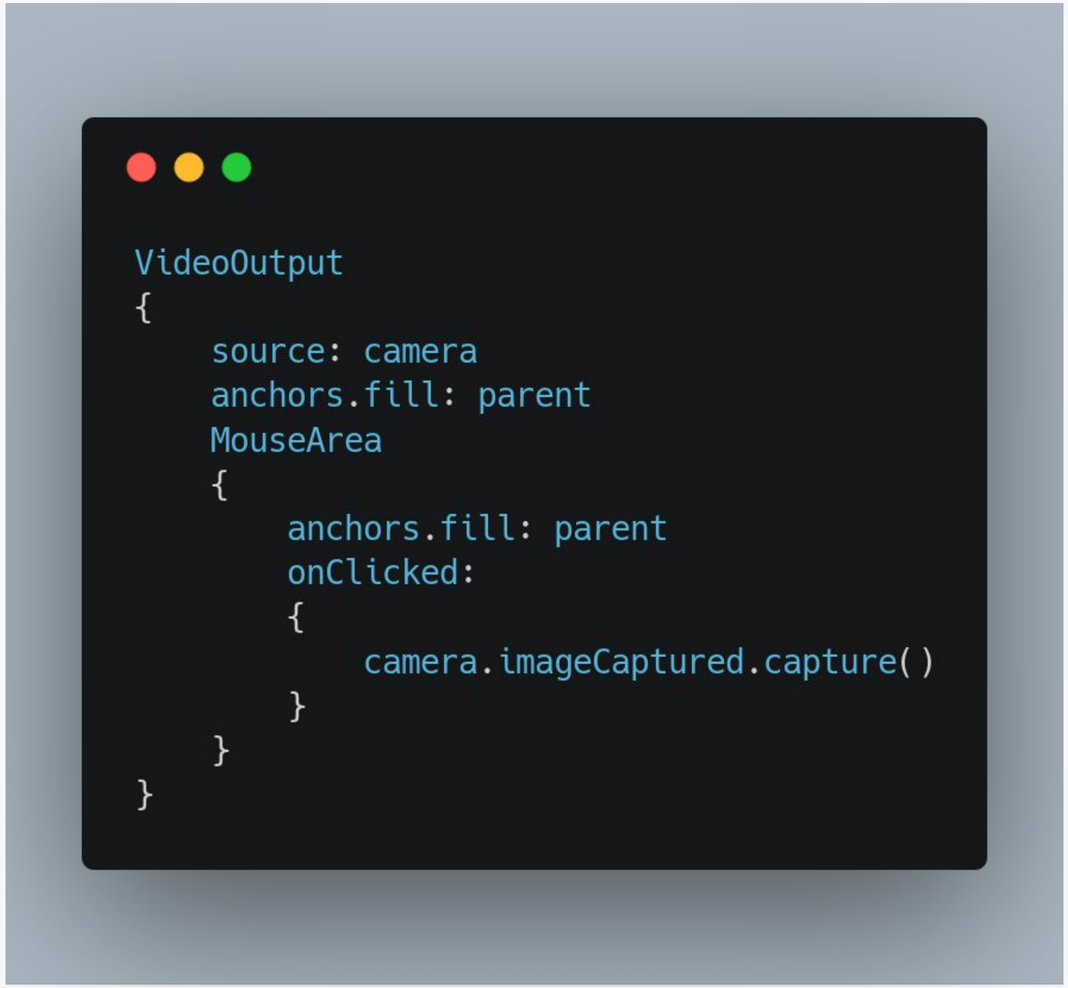
QtCon BR


ImageProcessor
{
id: imageProcessor
onImageProcessed:
{
imgViewer.setImage(image);
imageDrawer.open();
}
}

QtCon BR


Drawer
{
id: imageDrawer
width: parent.width
height: parent.height
ImageViewer:
{
id: imgViewer
anchors.fill: parent
}
}

QtCon BR



```
VideoOutput
{
    source: camera
    anchors.fill: parent
    MouseArea
    {
        anchors.fill: parent
        onClicked:
        {
            camera.imageCaptured.capture()
        }
    }
}
```

Visão Computacional no mobile

QtCon BR

Qt + OpenCV no Mobile

- https://opencv.org/platforms/android/
- Bibliotecas pré-compiladas do OpenCV para iOS e Android.

Conclusões

QtCon BR

QtCon BR

Obrigado!

perguntas?

Caio Jordão Carvalho
caiojcarvalho@gmail.com
https://carvalho.site