人脸识别封装类

```
#include <QObject>
#include"common.h"
#include <opencv2\imgproc\types_c.h>

class MyFaceDetect : public QObject
{
    Q_OBJECT
public:
    explicit MyFaceDetect(QObject *parent = 0);

signals:

public slots:

    // 0.人脸识别的初始化
    static void  FaceDetectInit();
    // 1.获取摄像头图片后 识别出人脸的位置, 返回位置对应的矩形框
    static void  detectAndDisplay(Mat &frame , std::vector<Rect> &faces);
};
```

定义实现

```
#include "myfacedetect.h"

#include "objdetect/objdetect.hpp"
#include<QCoreApplication>
#include<qDebug>

CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;

String window_name_face = "Capture - Face";


void MyFaceDetect::FaceDetectInit()
{
    // 将 xml 文件放在 exe 同级的目录下面
    QString face_cascade_name = QCoreApplication::applicationDirPath()
            +"/haarcascade_frontalface_default.xml";
            //+"/haarcascade_frontalface_alt_tree.xml";
            //+ "/lbpcascade_frontalface.xml" ;
    //"haarcascade_frontalface_alt.xml" //lbpcascade_frontalface.xml;
```

```
    QString eyes_cascade_name = QCoreApplication::applicationDirPath()
           // +"/haarcascade_eye.xml";
            +"/haarcascade_eye_tree_eyeglasses.xml";
    //haarcascade_eye_tree_eyeglasses.xml;

    // 根据路径加载 xml 文件
    qDebug() << face_cascade_name;
    //-- 1. Load the cascade
    if( !face_cascade.load( face_cascade_name.toStdString() ) )
    {
        qDebug()<< "--(!)Error loading face " ;
        return;
    }
    qDebug() << eyes_cascade_name;
    if( !eyes_cascade.load( eyes_cascade_name.toStdString() ) )
    {
        qDebug()<<"--(!)Error loading eyes " ;
        return;
    }
}

// 1.获取摄像头图片后 识别出人脸的位置, 返回位置对应的矩形框
void MyFaceDetect::detectAndDisplay(Mat &frame, std::vector<Rect> &faces)
{
    Mat frame_gray;
    //首先 , 得到无颜色的图片用于做识别
    cvtColor( frame, frame_gray, CV_BGR2GRAY );

    equalizeHist( frame_gray, frame_gray );

    //-- 多尺寸检测人脸
    //第三个参数 每个图像比例下图像大小减少多少的参数。
    //第四个参数 指定每个候选矩形应该为邻居保留多少个像素。
    //第五个参数 参数与函数 cvHaarDetectObjects 中的旧级联具有相同的含义。是新版本还是
旧
    //版本
    //第六个参数 最小可能对象大小。小于该值的对象将被忽略。
    //第七个参数 最大可能对象大小。大于该值的对象将被忽略
    face_cascade.detectMultiScale( frame_gray, faces
                , 1.1, 6, 0, Size(100,100) , Size(300, 300) );

    //绘制识别的人脸矩形
    for( auto ite = faces.begin() ; ite != faces.end() ; ++ite )
    {
        Rect rct = *ite;
        Point center( rct.x + rct.width*0.5, rct.y + rct.height*0.5 );
        ellipse( frame, center, Size( rct.width*0.5, rct.height*0.5), 0, 0, 360,
                 Scalar( 255, 0, 255 ), 4, 8, 0 );
    }
```

```
    //imshow( "capture_face", frame );

    //为了防止误识别, 再识别眼睛
    for( auto ite = faces.begin() ; ite != faces.end() ; )
    {
        Rect rct = *ite;
        Mat faceROI = frame_gray( rct );
        std::vector<Rect> eyes;
        //-- 在每张人脸上检测双眼
        eyes_cascade.detectMultiScale( faceROI, eyes , 1.1, 2, 0 , Size(20, 20) );
        //正常 鼻子的大小 应该不超过脸的 1/3
        if( eyes.size() != 2 )
        {
            ite = faces.erase(ite);
        }else
        {
            if( rct.height*0.5 < eyes[0].y ){
                ite = faces.erase(ite);
                continue;
            }
            // ellipse( frame, center, Size( rct.width*0.5, rct.height*0.5), 0, 0, 360,
            //Scalar( 255, 0, 255 ), 4, 8, 0 );
            // Rect rctnose;
            // rctnose.x = nose[0].x + rct.x;
            // rctnose.y = nose[0].y + rct.y;
            // rctnose.width = nose[0].width;
            // rctnose.height = nose[0].height;
            // Point nosecenter( rctnose.x + rctnose.width*0.5, rctnose.y
+rctnose.height*0.5 );
            // ellipse( frame, nosecenter, Size( rctnose.width*0.5,
rctnose.height*0.5),0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );
            ++ite;
        }
    }
    //-- 显示结果图像
    //imshow( window_name_face, frame );
}
```

萌拍实现

构造图片

现支持两种, 一个是兔子耳朵, 一个是圣诞帽子

```
int m_funnyPic; // 用于存储当前的萌拍效果 1 兔耳朵 2 帽子

enum funnyPic_type{ fp_tuer = 1 , fp_hat };

QImage m_tuer;
QImage m_hat;
```

```
m_tuer(":/images/tuer.png") , m_hat(":/images/hat.png")
m_funnyPic = 0; //根据下拉控件的选项，决定使用哪种效果，保存在这个变量中
```

采集过程的修改

```
//人脸识别 ， 如果失败，使用上一次的人脸矩形
std::vector<Rect> faces;
//存储上一次识别人脸的矩形
//m_vecLastFace;

if( m_funnyPic!= 0 )
    MyFaceDetect::detectAndDisplay( frame , faces );

//加载图片
QImage* tmpImg = nullptr;
switch( m_funnyPic )
{
    case fp_tuer:
        tmpImg = &m_tuer; break;
    case fp_hat:
        tmpImg = &m_hat; break;
}

if( faces .size() > 0)
    m_vecLastFace = faces;
//将道具绘制到图片上
if( tmpImg )
if( m_funnyPic == fp_tuer || m_funnyPic == fp_hat )
{
    //QPainter 使用
    QPainter paint( &image );
    //遍历所有人脸的矩形，画道具
    for( int i = 0 ; i < m_vecLastFace.size(); ++i)
    {
        Rect rct = m_vecLastFace[i];
        int x = rct.x + rct.width*0.5 - tmpImg -> width()*0.5 + 20;
//20 是图片的尺寸偏移
        int y = rct.y - tmpImg -> height();
        QPoint p (x , y);
        paint.drawImage(  p , * tmpImg );
    }
}
```