

```

#include "cv.h"
#include "highgui.h"
#include <stdio.h>

#ifdef _EiC
#define WIN32
#endif

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;

void detect_and_draw( IplImage* image );

const char* cascade_name =
    "haarcascade_frontalface_alt.xml";//人脸检测分类器

int main( int argc, char** argv )
{
    CvCapture* capture = 0;
    IplImage *frame, *frame_copy = 0;
    int optlen = strlen("--cascade=");
    const char* input_name;

    if( argc > 1 && strcmp( argv[1], "--cascade=", optlen ) == 0 )
    {
        cascade_name = argv[1] + optlen;
        input_name = argc > 2 ? argv[2] : 0;
    }
    else
    {
        cascade_name = "E:\\毕业设计\\智能机器人动态人脸识别系统\\陈建州程序 .xml";//
        分类器路径

        input_name = argc > 1 ? argv[1] : 0;
    }

    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    if( !cascade )//如果没有找到分类器，输出以下
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        fprintf( stderr,
            "Usage: facedetect --cascade=\"<cascade_path>\" [filename|camera_index]\n" );
        return -1;
    }
}

```

```

}
storage = cvCreateMemStorage(0);

capture = cvCaptureFromCAM( !input_name ? 0 : input_name[0] - '0' );//读取摄像头
if(!capture)//如果没有摄像头读取视频文件

    capture = cvCaptureFromAVI("检测.avi");

cvNamedWindow( "result", 1);//创建窗口

if( capture )
{
    for(;;)
    {
        if( !cvGrabFrame( capture ) )//从摄像头中抓取帧
            break;
        frame = cvRetrieveFrame( capture );//读取上边抓取的帧
        if( !frame )
            break;
        if( !frame_copy )
            frame_copy = cvCreateImage( cvSize(frame->width,frame->height),
                                         IPL_DEPTH_8U, frame->nChannels );
        if( frame->origin == IPL_ORIGIN_TL )
            cvCopy( frame, frame_copy, 0 );
        else
            cvFlip( frame, frame_copy, 0 );

        detect_and_draw( frame_copy );

        if( cvWaitKey( 10 ) >= 0 )
            break;
    }

    cvReleaseImage( &frame_copy );
    cvReleaseCapture( &capture );
}
else//没检测到视频文件或者摄像头
{
    const char* filename = (char*)"检测.jpg";//读图片
    IplImage* image = cvLoadImage( filename, 1 );

    if( image )
    {
        detect_and_draw( image );
    }
}

```

```

        cvWaitKey(0);
        cvReleaseImage( &image );
    }
    else
    {

        FILE* f = fopen( filename, "rt" );
        if( f )
        {
            char buf[1000+1];
            while( fgets( buf, 1000, f ) )
            {
                int len = (int)strlen(buf);
                while( len > 0 && isspace(buf[len-1]) )
                    len--;
                buf[len] = '\0';
                image = cvLoadImage( buf, 1 );
                if( image )
                {
                    detect_and_draw( image );
                    cvWaitKey(0);
                    cvReleaseImage( &image );
                }
            }
            fclose(f);
        }
    }

}

cvDestroyWindow("result");

return 0;
}

```

```

void detect_and_draw( IplImage* img )
{
    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
    }
}

```

```

        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}}
    };
    double scale = 1.3;
    IplImage* gray = cvCreateImage( cvSize(img->width,img->height), 8, 1 );
    IplImage* small_img = cvCreateImage( cvSize( cvRound (img->width/scale),
                                                cvRound (img->height/scale)),
                                        8, 1 );

    int i;

    cvCvtColor( img, gray, CV_BGR2GRAY );
    cvResize( gray, small_img, CV_INTER_LINEAR );
    cvEqualizeHist( small_img, small_img );
    cvClearMemStorage( storage );

    if( cascade )
    {
        double t = (double)cvGetTickCount();
        CvSeq* faces = cvHaarDetectObjects( small_img, cascade, storage,
                                            1.1,
                                            2,
0/*CV_HAAR_DO_CANNY_PRUNING*/,
                                            cvSize(30, 30) );//检测人脸返回矩形人
脸

        t = (double)cvGetTickCount() - t;
        printf( "detection time = %gms\n", t/((double)cvGetTickFrequency()*1000.) );
        for( i = 0; i < (faces ? faces->total : 0); i++ )//找到矩形中心，把矩形转化为圆形
        {
            CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
            CvPoint center;
            int radius;
            center.x = cvRound((r->x + r->width*0.5)*scale);
            center.y = cvRound((r->y + r->height*0.5)*scale);
            radius = cvRound((r->width + r->height)*0.25*scale);
            cvCircle( img, center, radius, colors[i%8], 3, 8, 0 );
        }
    }

    cvShowImage( "result", img );
    cvReleaseImage( &gray );
    cvReleaseImage( &small_img );
}

```

OpenCV 的人脸检测主要是调用训练好的 cascade (Haar 分类器) 来进行模式匹配。

cvHaarDetectObjects, 先将图像灰度化, 根据传入参数判断是否进行 canny 边缘处理(默认不使用), 再进行匹配。匹配后收集找出的匹配块, 过滤噪声, 计算相邻个数如果超过了规定值 (传入的 min\_neighbors) 就当成输出结果, 否则删去。

匹配循环: 将匹配分类器放大 scale (传入值) 倍, 同时原图缩小 scale 倍, 进行匹配, 直到匹配分类器的大小大于原图, 则返回匹配结果。匹配的时候调用 cvRunHaarClassifierCascade 来进行匹配, 将所有结果存入 CvSeq\* Seq (可动态增长元素序列), 将结果传给 cvHaarDetectObjects。

cvRunHaarClassifierCascade 函数整体是根据传入的图像和 cascade 来进行匹配。并且可以根据传入的 cascade 类型不同 (树型、stump (不完整的树) 或其他的), 进行不同的匹配方式。函数 cvRunHaarClassifierCascade 用于对单幅图片的检测。在函数调用前首先利用 cvSetImagesForHaarClassifierCascade 设定积分图和合适的比例系数 (=> 窗口尺寸)。当分析的矩形框全部通过级联分类器每一层的时返回正值(这是一个候选目标), 否则返回 0 或负值。

为了了解 OpenCV 人脸检测中寻找匹配图像的详细过程, 就把 cvHaarDetectObjects 和 cvRunHaarClassifierCascade 的源文件详细看了一遍, 并打上了注释。方便大家阅读。

附 cvHaarDetectObjects 代码:

```
CV_IMPL CvSeq*
cvHaarDetectObjects( const CvArr* _img,
                    CvHaarClassifierCascade* cascade,
                    CvMemStorage* storage, double scale_factor,
                    int min_neighbors, int flags, CvSize min_size )
{
    int split_stage = 2;
    CvMat stub, *img = (CvMat*)_img;
    //CvMat 多通道矩阵 *img=_img 指针代换传入图
    CvMat *temp = 0, *sum = 0, *tilted = 0, *sqsum = 0, *norm_img = 0, *sumcanny = 0,
    *img_small = 0;
    CvSeq* seq = 0;
    CvSeq* seq2 = 0;
    //CvSeq 可动态增长元素序列
    CvSeq* idx_seq = 0;
    CvSeq* result_seq = 0;
    CvMemStorage* temp_storage = 0;
    CvAvgComp* comps = 0;
    int i;
#ifdef _OPENMP
    CvSeq* seq_thread[CV_MAX_THREADS] = {0};
    int max_threads = 0;
#endif
    CV_FUNCNAME( "cvHaarDetectObjects" );
    __BEGIN__;
```

```

double factor;
int npass = 2, coi;
//npass=2
int do_canny_pruning = flags & CV_HAAR_DO_CANNY_PRUNING;
//true 做 canny 边缘处理
if( !CV_IS_HAAR_CLASSIFIER(cascade) )
    CV_ERROR( !cascade ? CV_StsNullPtr : CV_StsBadArg, "Invalid classifier
cascade" );
if( !storage )
    CV_ERROR( CV_StsNullPtr, "Null storage pointer" );
CV_CALL( img = cvGetMat( img, &stub, &coi ));
if( coi )
    CV_ERROR( CV_BadCOI, "COI is not supported" );
//一些出错代码
if( CV_MAT_DEPTH(img->type) != CV_8U )
    CV_ERROR( CV_StsUnsupportedFormat, "Only 8-bit images are supported" );
CV_CALL( temp = cvCreateMat( img->rows, img->cols, CV_8UC1 ));
CV_CALL( sum = cvCreateMat( img->rows + 1, img->cols + 1, CV_32SC1 ));
CV_CALL( sqsum = cvCreateMat( img->rows + 1, img->cols + 1, CV_64FC1 ));
CV_CALL( temp_storage = cvCreateChildMemStorage( storage ));
#ifdef _OPENMP
    max_threads = cvGetNumThreads();
    for( i = 0; i < max_threads; i++ )
    {
        CvMemStorage* temp_storage_thread;
        CV_CALL( temp_storage_thread = cvCreateMemStorage(0));
//CV_CALL 就是运行，假如出错就报错。
        CV_CALL( seq_thread[i] = cvCreateSeq( 0, sizeof(CvSeq),
//CvSeq
可动态增长元素序列

        sizeof(CvRect), temp_storage_thread ));
    }
#endif
if( !cascade->hid_cascade )
    CV_CALL( icvCreateHidHaarClassifierCascade(cascade) );
if( cascade->hid_cascade->has_tilted_features )
    tilted = cvCreateMat( img->rows + 1, img->cols + 1, CV_32SC1 ); //多通道
矩阵 图像长宽+1 4 通道
seq = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvRect), temp_storage ); //创建序列
seq 矩形
seq2 = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvAvgComp), temp_storage ); //创建序列
seq2 矩形和邻近
result_seq = cvCreateSeq( 0, sizeof(CvSeq), sizeof(CvAvgComp), storage ); //创建序列
result_seq 矩形和邻近

```

```

    if( min_neighbors == 0 )
        seq = result_seq;
    if( CV_MAT_CN(img->type) > 1 )
    {
        cvCvtColor( img, temp, CV_BGR2GRAY );
//img 转为灰度
        img = temp;
    }
    if( flags & CV_HAAR_SCALE_IMAGE )
//flag && 匹配图
    {
        CvSize win_size0 = cascade->orig_window_size;
//CvSize win_size0 为分类器的原始大小
        int use_ipp = cascade->hid_cascade->ipp_stages != 0 &&
            icvApplyHaarClassifier_32s32f_C1R_p != 0; //IPP
        相关函数
        if( use_ipp )
            CV_CALL( norm_img = cvCreateMat( img->rows, img->cols, CV_32FC1 ));
//图像的矩阵化 4 通道.
            CV_CALL( img_small = cvCreateMat( img->rows + 1, img->cols + 1, CV_8UC1 ));
//小图矩阵化 单通道 长宽+1
            for( factor = 1; ; factor *= scale_factor )
//成 scale_factor 倍数匹配
            {
                int positive = 0;
                int x, y;
                CvSize win_size = { cvRound(win_size0.width*factor),
                                    cvRound(win_size0.height*factor) };
//win_size 分类器行列 (扩大 factor 倍)
                CvSize sz = { cvRound( img->cols/factor ), cvRound( img->rows/factor ) };
//sz 图像行列 (缩小 factor 倍) 三个 Cvsize
                CvSize sz1 = { sz.width - win_size0.width, sz.height - win_size0.height };
//sz1 图像 减 分类器行列
                CvRect rect1 = { icv_object_win_border, icv_object_win_border,
                                win_size0.width - icv_object_win_border*2,
//icv_object_win_border (int) 初始值=1
                                win_size0.height - icv_object_win_border*2 };
//矩形框 rect1
                CvMat img1, sum1, sqsum1, norm1, tilted1, mask1;
//多通道矩阵
                CvMat* _tilted = 0;
                if( sz1.width <= 0 || sz1.height <= 0 )
//图片宽或高小于分类器 ->跳出
                    break;

```

```

        if( win_size.width < min_size.width || win_size.height < min_size.height )    //分
        类器高或宽小于给定的 mini_size 的高或宽 ->继续
            continue;
//CV_8UC1 见定义.
#define CV_MAKETYPE(depth,cn) ((depth) + (((cn)-1)<< CV_CN_SHIFT))
//深度+ (cn-1) 左移 3 位    depth, depth+8, depth+16, depth+24.
        img1 = cvMat(  sz.height,  sz.width,  CV_8UC1,  img_small->data.ptr );
//小图的矩阵化 img1 单通道
        sum1 = cvMat(  sz.height+1,  sz.width+1,  CV_32SC1,  sum->data.ptr );
//长宽+1 4 通道 8 位    多通道矩阵
        sqsum1 = cvMat(  sz.height+1,  sz.width+1,  CV_64FC1,  sqsum->data.ptr );
//长宽+1 4 通道 16 位
        if( tilted )
        {
            tilted1 = cvMat( sz.height+1, sz.width+1, CV_32SC1, tilted->data.ptr );    //
            长宽+1 4 通道 8 位
            _tilted = &tilted1;
//长宽+1 4 通道 8 位
        }
        norm1 = cvMat(  sz1.height,  sz1.width,  CV_32FC1,  norm_img ?
norm_img->data.ptr : 0 ); //norm1 图像 减 分类器行列 4 通道
        mask1 = cvMat(  sz1.height,  sz1.width,  CV_8UC1,  temp->data.ptr );
//mask1 灰度图
        cvResize(      img,      &img1,      CV_INTER_LINEAR      );
//img 双线性插值 输出到 img1
        cvIntegral(      &img1,      &sum1,      &sqsum1,      _tilted      );
//计算积分图像
        if( use_ipp && icvRectStdDev_32s32f_C1R_p( sum1.data.i, sum1.step,
sqsum1.data.db, sqsum1.step, norm1.data.fl, norm1.step, sz1, rect1 ) < 0 )
            use_ipp = 0;
        if( use_ipp )
//如果 ipp=true (intel 视频处理加速等的函数库)
        {
            positive = mask1.cols*mask1.rows;
//mask1 长乘宽 ->positive
            cvSet(      &mask1,      cvScalarAll(255)      );
//mask1 赋值为 255
            for( i = 0; i < cascade->count; i++ )
            {
                if( icvApplyHaarClassifier_32s32f_C1R_p(sum1.data.i, sum1.step,
norm1.data.fl, norm1.step, mask1.data.ptr, mask1.step,
sz1, &positive, cascade->hid_cascade->stage_classifier[i].threshold,
cascade->hid_cascade->ipp_stages[i]) < 0 )
                {

```



```

        use_ipp = 0;

//ipp=false;

        break;
    }
    if( positive <= 0 )
        break;
}

}

if( !use_ipp )

//如果 ipp=false
{
    cvSetImagesForHaarClassifierCascade( cascade, &sum1, &sqsum1, 0, 1. );
    for( y = 0, positive = 0; y < sz1.height; y++ )
        for( x = 0; x < sz1.width; x++ )
        {
            mask1.data.ptr[mask1.step*y + x] =
                cvRunHaarClassifierCascade( cascade, cvPoint(x,y), 0 ) > 0;

//匹配图像.

            positive += mask1.data.ptr[mask1.step*y + x];
        }
    if( positive > 0 )
    {
        for( y = 0; y < sz1.height; y++ )
            for( x = 0; x < sz1.width; x++ )
                if( mask1.data.ptr[mask1.step*y + x] != 0 )
                {
                    CvRect obj_rect = { cvRound(y*factor), cvRound(x*factor),
                                         win_size.width, win_size.height };
                    cvSeqPush( seq, &obj_rect );

//将匹配块放到 seq 中
                }
            }
        }
    }
    else
//!(flag && 匹配图)
    {
        cvIntegral( img, sum, sqsum, tilted );
        if( do_canny_pruning )
        {
            sumcanny = cvCreateMat( img->rows + 1, img->cols + 1, CV_32SC1 );

//如果 做 canny 边缘检测
            cvCanny( img, temp, 0, 50, 3 );

```

```

        cvIntegral( temp, sumcanny );
    }
    if( (unsigned)split_stage >= (unsigned)cascade->count ||
        cascade->hid_cascade->is_tree )
    {
        split_stage = cascade->count;
        npass = 1;
    }
    for( factor = 1; factor*cascade->orig_window_size.width < img->cols - 10 &&
//匹配
        factor*cascade->orig_window_size.height < img->rows - 10;
        factor *= scale_factor )
    {
        const double ystep = MAX( 2, factor );
        CvSize win_size = { cvRound( cascade->orig_window_size.width * factor ),
                            cvRound( cascade->orig_window_size.height * factor ) };
        CvRect equ_rect = { 0, 0, 0, 0 };
        int *p0 = 0, *p1 = 0, *p2 = 0, *p3 = 0;
        int *pq0 = 0, *pq1 = 0, *pq2 = 0, *pq3 = 0;
        int pass, stage_offset = 0;
        int stop_height = cvRound((img->rows - win_size.height) / ystep);
        if( win_size.width < min_size.width || win_size.height < min_size.height )
//超边跳出
            continue;
        cvSetImagesForHaarClassifierCascade( cascade, sum, sqsum, tilted, factor );
//匹配
        cvZero(
                                temp
                                );
//清空 temp 数组
        if(
                                do_canny_pruning
                                )
//canny 边缘检测
        {
            equ_rect.x = cvRound(win_size.width*0.15);
            equ_rect.y = cvRound(win_size.height*0.15);
            equ_rect.width = cvRound(win_size.width*0.7);
            equ_rect.height = cvRound(win_size.height*0.7);

            p0 = (int*)(sumcanny->data.ptr + equ_rect.y*sumcanny->step) + equ_rect.x;
            p1 = (int*)(sumcanny->data.ptr + equ_rect.y*sumcanny->step)
                + equ_rect.x + equ_rect.width;
            p2 = (int*)(sumcanny->data.ptr + (equ_rect.y +
equ_rect.height)*sumcanny->step) + equ_rect.x;
            p3 = (int*)(sumcanny->data.ptr + (equ_rect.y +
equ_rect.height)*sumcanny->step)
                + equ_rect.x + equ_rect.width;

```

```

        pq0 = (int*)(sum->data.ptr + equ_rect.y*sum->step) + equ_rect.x;
        pq1 = (int*)(sum->data.ptr + equ_rect.y*sum->step)
            + equ_rect.x + equ_rect.width;
        pq2 = (int*)(sum->data.ptr + (equ_rect.y + equ_rect.height)*sum->step) +
equ_rect.x;
        pq3 = (int*)(sum->data.ptr + (equ_rect.y + equ_rect.height)*sum->step)
            + equ_rect.x + equ_rect.width;
    }
    cascade->hid_cascade->count = split_stage;
//分裂级
    for( pass = 0; pass < npass; pass++ )
    {
#ifdef _OPENMP

```

人脸识别在 opencv 下作人脸检测（转）OPENCV 2010-03-04 15:17:51 阅读 103 评论 0 字号：大中小

人脸识别的第一步，就是人脸检测。把人的脸部从一张照片中用计算机自动识别出来，作为下一步人脸识别的基础。

在 opencv 中，库中自带了一个利用 harr 特征的人脸检测训练及检测函数：cvHaarDetectObjects。它利用训练好的检测器，在图片中间检测你想要的物体，如人脸。opencv 自带了很多检测器，在%opencv%data\haarcascades 目录下，你可以随意取用。或者你也可以自己用图片训练自己的检测器，之后拿来使用。

下面是检测人脸的源代码：

```

// Usage: DetectFaces <imagefilename>

#include <stdio.h>

#include "cv.h"

#include "highgui.h"

// *** Change this to your install location! ***

// *****

#define OPENCV_ROOT "C:/Program Files/OpenCV"

// *****

```

```

void displayDetections(IplImage * pInpImg, CvSeq * pFaceRectSeq, char* FileName);

int main(int argc, char** argv)

{

// variables

IplImage * pInpImg = 0;

CvHaarClassifierCascade * pCascade = 0; // the face detector

CvMemStorage * pStorage = 0; // memory for detector to use

CvSeq * pFaceRectSeq; // memory-access interface

// usage check

if(argc < 2)

{

printf("Missing name of image file!\n"

      "Usage: %s <imagefilename>\n", argv[0]);

exit(-1);

}

// initializations

pInpImg = (argc > 1) ? cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR) : 0;

pStorage = cvCreateMemStorage(0);

pCascade = (CvHaarClassifierCascade *)cvLoad

    ((OPENCV_ROOT"/data/haarcascades/haarcascade_frontalface_default.xml"),

    0, 0, 0);

// validate that everything initialized properly

```

```

if( !pInpImg || !pStorage || !pCascade )

{

printf("Initialization failed: %s\n",

(!pInpImg)? "can't load image file" :

(!pCascade)? "can't load haar-cascade -- "

"make sure path is correct" :

"unable to allocate memory for data storage", argv[1]);

exit(-1);

}

// detect faces in image

pFaceRectSeq = cvHaarDetectObjects

(pInpImg, pCascade, pStorage,

1.1, // increase search scale by 10% each pass

3, // merge groups of three detections

CV_HAAR_DO_CANNY_PRUNING, // skip regions unlikely to contain a face

cvSize(40,40)); // smallest size face to detect = 40x40

// display detected faces

displayDetections(pInpImg, pFaceRectSeq, argv[1]);

// clean up and release resources

cvReleaseImage(&pInpImg);

if(pCascade) cvReleaseHaarClassifierCascade(&pCascade);

if(pStorage) cvReleaseMemStorage(&pStorage);

```

```

return 0;

}

void displayDetections(IplImage * pInpImg, CvSeq * pFaceRectSeq, char* FileName)

{

const char * DISPLAY_WINDOW = "Haar Window";

int i;

// create a window to display detected faces

cvNamedWindow(DISPLAY_WINDOW, CV_WINDOW_AUTOSIZE);

// draw a rectangular outline around each detection

for(i=0;i<(pFaceRectSeq? pFaceRectSeq->total:0); i++) )

{

CvRect* r = (CvRect*)cvGetSeqElem(pFaceRectSeq, i);

CvPoint pt1 = { r->x, r->y };

CvPoint pt2 = { r->x + r->width, r->y + r->height };

cvRectangle(pInpImg, pt1, pt2, CV_RGB(0,255,0), 3, 4, 0);

cvSetImageROI(pInpImg, *r);

//char* FileName = argv[1];

IplImage* dst = cvCreateImage( cvSize(92,112), pInpImg->depth, pInpImg->nChannels);

cvResize(pInpImg, dst, CV_INTER_LINEAR);

strcat(FileName, ".pgm");

cvSaveImage(FileName, dst);

}

```

```
// display face detections

cvShowImage(DISPLAY_WINDOW, pInpImg);

cvWaitKey(0);

cvDestroyWindow(DISPLAY_WINDOW);

}
```

程序会将人脸检测出来，显示在屏幕上，并存回原来的文件将其覆盖。

在 window xp， devcpp 4.9.9.2 下编译通过。