X_vision SDK快速入门手册

1. 快速开始

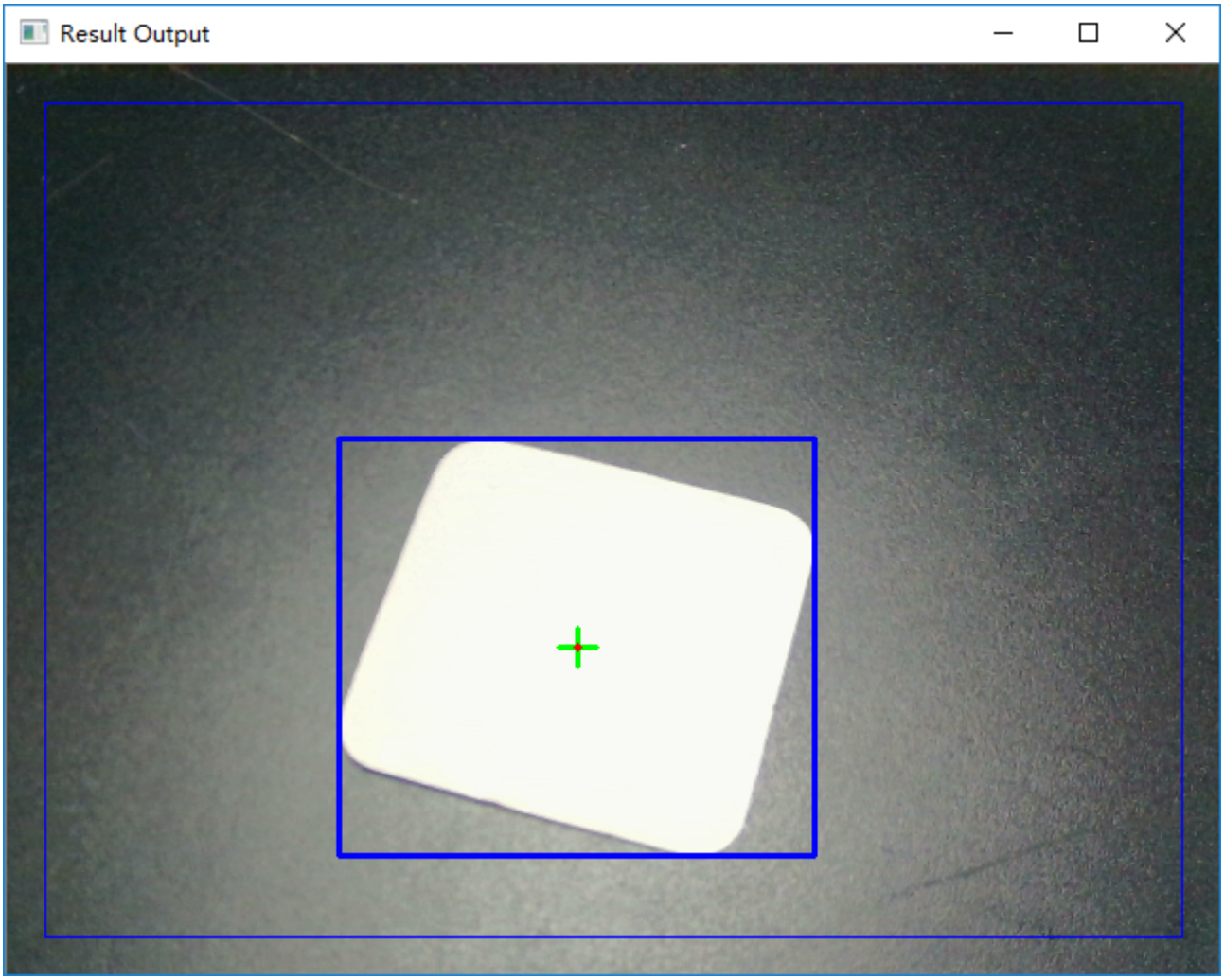
为了调用底层算法，你必须将输入图像转换成mvInputImage类型。首先初始化一个算法实例mvInstanceAlloc()，然后调用mvAlgProcess()执行算法处理进程，在此之前你可以预设置感兴趣检测区域。同时，你还可以选择输出算法处理后的结果mvResult。下面是一个简单的演示demo。

```
#include <stdio.h>
#include <stdlib.h>
#include <opencv2/opencv.hpp>
#include "DllmvInterface.h"
#include <windows.h>
#include <iostream>

#define MV_CFG_PATH "./imvcfg/"
initParam param;

int main(int argc, char* argv[])
{
    using namespace cv;
    mvInputImage orgImage;
    mvDetRoi det;
    mvResult *pRes;
    //读取一张图片
    Mat img = imread("test.jpg");
    //初始化结果保存图像
    IplImage* iplRes = cvCreateImage(cvSize(img.cols, img.rows), IPL_DEPTH_8U, 3);
    //加载算法配置文件
    strcpy(param.cfgPath, MV_CFG_PATH);
    //构造联通物体检测算法实例
    pAlg = (algDllHandle*)mvInstanceAlloc(img.cols, img.rows, MV_ALG_COMPONENT_DET,
&param);
```

```
//绘制第1个矩形
det.polys[0].ppnts[0].x = 40;
det.polys[0].ppnts[0].y = 40;
det.polys[0].ppnts[1].x = 360;
det.polys[0].ppnts[1].y = 40;
det.polys[0].ppnts[2].x = 360;
det.polys[0].ppnts[2].y = 160;
det.polys[0].ppnts[3].x = 40;
det.polys[0].ppnts[3].y = 160;
//回到起始点
det.polys[0].ppnts[4] = det.polys[0].ppnts[0];
//共绘制了5个点
det.polys[0].num = 5;

det.polys[0].lable = 0;  //polygon zone lable
det.polys[0].uc = 44;  //lable color
det.polys[0].valid = 1;  //set true if this is an valid area
det.polys[0].uniteFlag = 0;  //set true if need to use unite

//seed: (50, 50)
det.polys[0].seed.x = det.polys[0].ppnts[0].x + 10;
det.polys[0].seed.y = det.polys[0].ppnts[0].y + 10;
det.roiMap = NULL; //roi image, set to null if not want to use it

//绘制第2个矩形
det.polys[1].ppnts[0].x = 200;
det.polys[1].ppnts[0].y = 200;
det.polys[1].ppnts[1].x = 824;
det.polys[1].ppnts[1].y = 200;
det.polys[1].ppnts[2].x = 824;
det.polys[1].ppnts[2].y = 568;
det.polys[1].ppnts[3].x = 200
det.polys[1].ppnts[3].y = 568;
det.polys[1].ppnts[4] = det.polys[1].ppnts[0];
det.polys[1].num = 5;

det.polys[1].lable = 1;
det.polys[1].uc = 45;
det.polys[1].valid = 1;
det.polys[1].uniteFlag = 0;

// seed: (210,210)
det.polys[1].seed.x = det.polys[1].ppnts[0].x + 10;
det.polys[1].seed.y = det.polys[1].ppnts[0].y + 10;
det.roiMap = NULL;
```

```
    //设置感兴趣检测区域
    MvSetDetRoiArea(pAlg, det, NULL);

    //将img转换成mvInputImage类型
    orgImage.frameIndex = 0;
    orgImage.pFrame = (void*)img.data;
    orgImage.width = img.cols;
    orgImage.height = img.rows;
    orgImage.nChannel = img.channels();
    orgImage.widthStep = img.cols * img.channels();
    orgImage.depth = img.depth();
    orgImage.type = MV_BGR24;

    //调用算法处理
    ret = mvAlgProcess(pAlg, (mvInputImage*)&orgImage);

    //输出处理结果
    if (ret)
    {
        pRes = (mvResult*)&pAlg->result;
        //内部输出
        mvMatchObjsDrawAndDisplay(pAlg, pRes);

        //获取算法处理结果
        iplRes->imageData = (char*)pAlg->imgCr.imageData;
        //转换成Mat类型
        Mat matRes = cvarrToMat(iplRes);
        //外部调用OpenCV输出
        imshow("Result Output", matRes);
        //按键等待
        waitKey(0);
    }
}
```

执行以上程序，你将会看到以下输出：



你可以通过修改算法参数配置文件 cfgparam.txt 调整算法以获得更佳的处理效果。

2. 通用算法实例构造接口

```
/**
* 该方法用于构造算法实例并返回一个void指针，你需要转换成 algDllHandle* 类型
* @width 图像宽度，类型为int
* @height 图像高度，类型为int
* @type 算法类型，类型为algType
* @para 算法初始化参数，类型为initParam*
*/
void* mvInstanceAlloc(int width, int height, algType type, initParam* para);
```

3. 设置感兴趣检测区域

```
/**
* 该方法用于设置算法工作区域，设置成功返回非0，否则返回0
* @ppAlg 算法实例，类型为void*
* @roi 检测区域，类型为mvDetRoi
* @cfgpath 算法配置文件目录，类型为char*
*/
int MvSetDetRoiArea(void *ppAlg, mvDetRoi roi, char *cfgpath);
```

4. 通用算法调用接口

```
/**
* 该方法用于调用底层算法对图像进行处理，算法调用成功返回非0，否则返回0
* @ppAlg 算法实例，类型为void*
* @imageInput 待处理图像，类型为mvInputImage*
*/
int mvAlgProcess(void* ppAlg, mvInputImage* imageInput);
```

5. 输出算法处理结果

```
/**
* 该方法用于内部输出算法处理的结果
* @ppAlg 算法实例，类型为void*
* @res 算法处理结果，类型为mvResult*
*/
void mvMatchObjsDrawAndDisplay(void* ppAlg, mvResult* res);
```

6. 算法类型

```
typedef enum
{
    MV_ALG_SHAPE_MATCH = 0,   //形状匹配
    MV_ALG_SHAPE_TMP,   //形状匹配的模板方法
    MV_ALG_FEATURE_LOC,   //特征点定位
    MV_ALG_LOC_MATCH,   //特征点匹配
    MV_ALG_FEATURE_TMP = 4,   //模板建立
    MV_ALG_QRDECODE_DET,   //二维码识别
```

```
    MV_ALG_LOC_TMP_LOC,  //基于特征点的定位处理
    MV_ALG_ZJLOC_DET,  //铸件定位
    MV_ALG_REVERTING_IMAGE,  //图像定位融合/模板定位
    MV_ALG_SURFACE_DET,  //表面检测
    MV_ALG_TANZUAN_DET,  //探钻检测
    MV_ALG_DOCKRECOG,  //dock识别
    MV_ALG_TRACKING_TMP,  //目标跟踪的模板方法
    MV_ALG_OBJECT_TRACKING,  //目标跟踪
    MV_ALG_FACE_TRACKING,  //人脸跟踪
    MV_ALG_ZJSURFACE_DET,  //铸件表面检测
    MV_ALG_CIR_DET,  //圆检测
    MV_ALG_LINE_DET,  //线检测
    MV_ALG_FACE_DET,  //人脸检测
    MV_ALG_OBJECT_MATCH,  //匹配目标
    MV_ALG_PHONE_RECYCLE,  //phone recycle system
    MV_ALG_COMPONENT_DET = 21,  //联通物体识别
    MV_ALG_PHONE_MEAUREMENT,  //phone recycle system
    MV_ALG_ZMCLOTH_DET1 = 22,  //zm cloth surface detect
    MV_ALG_ZMCLOTH_DET2 = 23,  //zm cloth surface detect
    MV_ALG_ZMCLOTH_DET_DL1 = 24,  //deep-learning model 1 use ssd network
    MV_ALG_ZMCLOTH_DET_DL2,  //deep-learning model 2 use fcn network
    MV_ALG_ZMCLOTH_DET_DL3,  //deep-learning model 2 use fcn network
    MV_ALG_ZMCLOTH_DET_CAFFE_SSD = 27,  //deep-leanning model use caffe-ssd network
    MV_ALG_ZMCLOTH_DET_CAFFE_FCN,  //deep-leanning model use caffe-fcn network
    MV_ALG_LZCOUNTER = 30,  //目标检测和计数
    MV_ALG_IMAGE_QULITY_EVAL = 31,  //调光
    MV_ALG_FOCUS_EVAL = 32,  //image focus evaluation
    MV_ALG_DARKNET_PROCESS = 33,  //物体识别
}algType;
```