# 边缘识别算子快速入门手册

目录

[边缘识别算子快速入门手册 1](#_Toc533175912)

[1．引言 3](#_Toc533175913)

[1.1编写目的 3](#_Toc533175914)

[1.2其它说明 3](#_Toc533175915)

[2. 算子说明 3](#_Toc533175916)

[2.1 边缘识别算子 3](#_Toc533175917)

[2.1.1类型定义及说明 3](#_Toc533175918)

[2.1.2相关数据结构 3](#_Toc533175919)

[2.1.3 调用流程 4](#_Toc533175920)

[2.1.4 使用案例 5](#_Toc533175921)

[2.1.5 注意事项 12](#_Toc533175922)

[2.1.6 算子配置 12](#_Toc533175923)

[2.1.7 函数说明： 13](#_Toc533175924)

[3. 附录 14](#_Toc533175925)

# 1．引言

## 1.1编写目的

此文档是为了说明边缘识别算子的功能，以及调用算子的具体流程，方便其他人使用开发算子。

## 1.2其它说明

算子实例的通用构造、执行接口、算法配置文件等说明可见附录《X\_vision SDK快速入门手册》。

# 2. 算子说明

## 2.1 边缘识别算子

### 2.1.1类型定义及说明

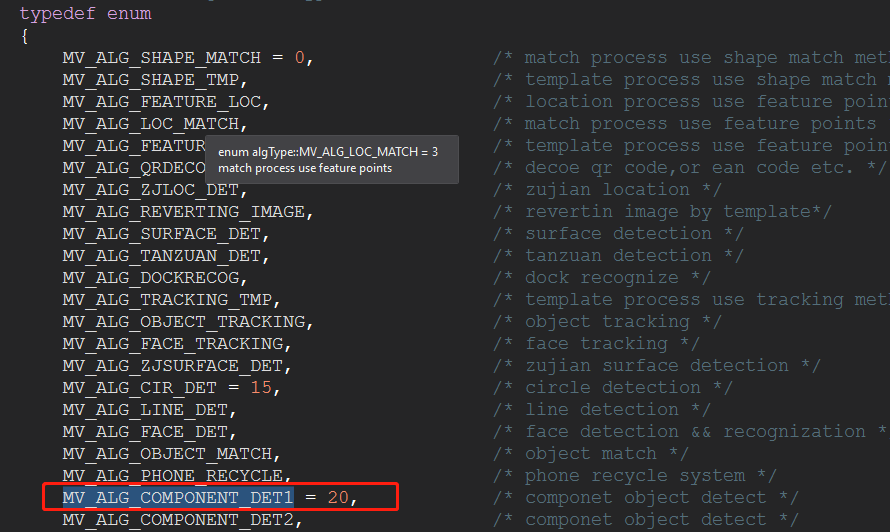
算子类型为：algType::MV\_ALG\_COMPONENT\_DET1/ algType::MV\_ALG\_COMPONENT\_DET2，主要功能是进行边缘的识别以及检测。

### 2.1.2相关数据结构

相关数据结构定义如下：

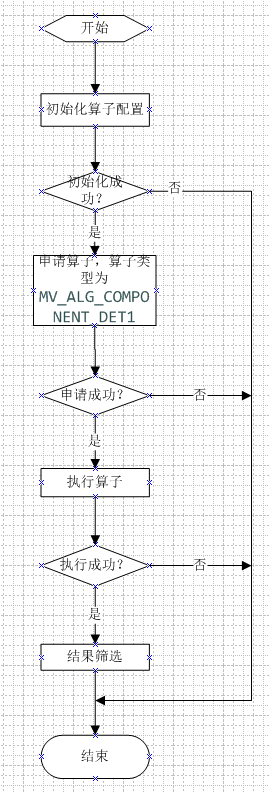
/\* alg sub\_type \*/

#define MV\_ALG\_COMPONENT\_DET1 20



### 2.1.3 调用流程

边缘识别算子调用流程如下：



边缘识别流程图

### 2.1.4 使用案例

#include <stdio.h>

#include <stdlib.h>

#include <opencv2/opencv.hpp>

#include "DllmvInterface.h"

#include <windows.h>

#include "mvSDK\_interface.h"

#include <iostream>

#include <vector>

#include<opencv2/ml/ml.hpp>

using namespace cv;

using namespace std;

algDllHandle \*pAlg = NULL;

RNG rng(12345);

//alg算法配置

//算法缩放系数

#define MV\_ALG\_SCL 1

//算法配件路径

#define MV\_CONFIG\_PATH "D:/VS/TestComponent/x64/Release/imvcfg/"

#define TEST\_SAMPLE\_PATH "D:/VS/TestComponent/x64/Release/pic/"

//输入图像缩放系数

#define MV\_INTPUT\_IMAGE\_SCAL 1

//算法类型

#define MV\_ALG\_TYPE (MV\_ALG\_COMPONENT\_DET1)

#define INPUT\_WIDTH 1024

#define INPUT\_HEIGHT 750

#define TEST\_PATH "imageList.txt"

void load\_images(const string & prefix, const string & filename, vector< Mat > & img\_lst, int flag, Size ct)

{

string line;

ifstream file;

Mat img1, img2;

int width = ct.width;

int height = ct.height;

file.open((prefix + filename).c\_str());

if (!file.is\_open())

{

cerr << "Unable to open the list of images from " << filename << " filename." << endl;

exit(-1);

}

bool end\_of\_parsing = false;

while (!end\_of\_parsing)

{

getline(file, line);

if (line.empty()) // no more file to read

{

end\_of\_parsing = true;

break;

}

Mat img = imread((prefix + line).c\_str()); // load the image

if (img.empty()) // invalid image, just skip it.

continue;

#ifndef \_DEBUGX

//imshow("image", img);

//waitKey(10);

#endif

if (flag)

{

//resize

resize(img, img2, Size(width, height));

#ifdef \_DEBUG

//imshow("image", img2);

//waitKey(10);

#endif

img\_lst.push\_back(img2.clone());

}

else

img\_lst.push\_back(img.clone());

}

}

void cclTtemsDraw(mvCCLItems \*pComps, cv::Mat img)

{

int i;

mvCCLItem \*pComp;

mvRect rc;

mvPoint pt;

int x, xx, y, yy;

cv::Point pp1, pp2;

vector<mvFPoint> subpixels;

for (i = pComps->num\_comp - 1; i >= 0; i--)

{

cv::Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));

pComp = &pComps->ccomps[i];

if (!pComp->vflag)

continue;

x = pComp->rec\_bound.up\_left.x;

y = pComp->rec\_bound.up\_left.y;

xx = pComp->rec\_bound.dw\_right.x;

yy = pComp->rec\_bound.dw\_right.y;

rc = pComp->rec\_bound;

pt = pComp->pnt\_mass;

cv::rectangle(img, cv::Rect(x, y, xx - x, yy - y), Scalar(0, 0, 255));

for (int k = 0; k < pComp->pnt\_countour.nump; k++)

{

pp1.x = pComp->pnt\_countour.pnts[k].x;

pp1.y = pComp->pnt\_countour.pnts[k].y;

cv::circle(img, pp1, 0.1, color, 1, 8, 0);

}

//draw "+"

int len = 10;

pp1.x = pComp->pnt\_center.x;

pp1.y = pComp->pnt\_center.y - len;

pp2.x = pComp->pnt\_center.x;

pp2.y = pComp->pnt\_center.y + len;

rectangle(img, pp1, pp2, CV\_RGB(0, 255, 0));

pp1.x = pComp->pnt\_center.x - len;

pp1.y = pComp->pnt\_center.y;

pp2.x = pComp->pnt\_center.x + len;

pp2.y = pComp->pnt\_center.y;

rectangle(img, pp1, pp2, CV\_RGB(0, 255, 0));

/\*object centor \*/

pp1.x = pComp->pnt\_center.x, pp1.y = pComp->pnt\_center.y;

line(img, pp1, pp1, cvScalar(0, 0, 255), 2);

}

return;

}

void\* algLocationProcessWrap(algType type)

{

int ret;

mvInputImage orgImage;

initParam param;

mvResult \*pRes;

int keyval = 0;

int key;

mvImage clone;

vector< Mat > pos\_lst;

vector< Mat > test\_lst;

vector< Mat > full\_neg\_lst;

vector< Mat > neg\_lst;

vector< Mat > gradient\_lst;

vector< int > labels;

//string neg = NEG\_PATH;

string test\_dir = TEST\_SAMPLE\_PATH;

string test\_sample = TEST\_PATH;

int frameNum;

//设置alg param初始化参数

strcpy\_s(param.cfg\_path, MV\_CONFIG\_PATH);

//param.tmp\_offset = offset;

//param.tmp\_roi = loc;

param.online\_temp = 0; //not used

param.tmp\_img.pframe = NULL;

param.tmp\_img.width = 0; //temp->width;

param.tmp\_img.height = 0; //temp->height;

param.tmp\_img.channels = 0; //temp->nChannels;

param.tmp\_img.depth = 0; //temp->depth;

param.tmp\_img.wstep = 0; //temp->wstep;

param.tmp\_img.type = MV\_CV\_IPL; //MV\_CV\_IPL;

IplImage \*org = NULL;

pAlg = (algDllHandle\*)mvInstanceAlloc(INPUT\_WIDTH, INPUT\_HEIGHT, type, &param);

if (!pAlg)

return NULL;

pAlg->sub\_type = 2; //sub\_type of this algorithm

pRes = (mvResult\*)&pAlg->result;

//读取本地图像

load\_images(test\_dir, test\_sample, test\_lst, 1, Size(INPUT\_WIDTH, INPUT\_HEIGHT));

vector< Mat >::const\_iterator img = test\_lst.begin();

vector< Mat >::const\_iterator end = test\_lst.end();

frameNum = 0;

mvDetRoi det;

for (int i = 0; img != end; img++)

{

static int flag = 0;

//cv::Mat frame = cv::imread(TEST\_IMAGE, -1);

cv::Mat frame = img->clone();

if (flag == 0)

{

org = cvCreateImage(cvSize(frame.cols, frame.rows), IPL\_DEPTH\_8U, 3);

flag = 1;

det.num\_poly = 3;

int startxy = 120;

int rww, rhh;

rww = 200; rhh = 200;

det.polys[0].ppnts[0].x = 40;

det.polys[0].ppnts[0].y = 40;

det.polys[0].ppnts[1].x = rww - det.polys[0].ppnts[0].x;

det.polys[0].ppnts[1].y = det.polys[0].ppnts[0].y;

det.polys[0].ppnts[2].x = rww - det.polys[0].ppnts[0].x;

det.polys[0].ppnts[2].y = rhh - det.polys[0].ppnts[0].y;

det.polys[0].ppnts[3].x = det.polys[0].ppnts[0].x;

det.polys[0].ppnts[3].y = rhh - det.polys[0].ppnts[0].y;

det.polys[0].ppnts[4] = det.polys[0].ppnts[0];

det.polys[0].num = 5;

det.polys[0].index = 0;

det.polys[0].uc = 80;

det.polys[0].valid = 1;

det.polys[0].uflag = 0;

det.polys[0].seed.x = det.polys[0].ppnts[0].x + 2;

det.polys[0].seed.y = det.polys[0].ppnts[0].y + 2;

cv::Mat frame2 = img->clone();

clone.channels = 1;

clone.width = frame2.cols;

clone.height = frame.rows;

det.roi\_map = NULL; //important

}

Mat input2;

frame.copyTo(input2);

orgImage.index = frameNum++;

orgImage.pframe = (void\*)frame.data; /\* 图像数据地址 \*/

orgImage.width = frame.cols; /\* 图像宽度 \*/

orgImage.height = frame.rows; /\* 图像高度 \*/

orgImage.channels = frame.channels(); /\* 图像通道\*/

orgImage.wstep = frame.cols \* frame.channels(); /\* 图像 wstep = 宽度\* 通道数 \*/

orgImage.depth = frame.depth(); /\* 图像深度 \*/

orgImage.type = MV\_BGR24; /\*帧的格式\*/

int64 nTick = ::GetTickCount();

double ptime;

nTick = getTickCount();

/\* 算法处理 \*/

imshow("org", frame);

ret = mvAlgProcess(pAlg, (mvInputImage\*)&orgImage);

ptime = ((double)getTickCount() - nTick)\*1000. / getTickFrequency();

printf("processed time = %.2f ms\n", ptime);

cout << pAlg->result.cc\_items->ccomps->com\_label;

if (ret > 0)

{

//绘制结果

//mvMatchObjsDrawAndDisplay(pAlg, pRes);

cclTtemsDraw(pRes->cc\_items, input2);

imshow("ccls", input2);

//strcpy(strsss, TEST\_SAMPLE\_PATH);

//sprintf(nnn, "result-%02d.png", frameNum);

//strcat(strsss, nnn);

//

//imwrite(strsss, input2);

Mat bin, edge, com;

bin = Mat(cvSize(pAlg->bin\_img.width, pAlg->bin\_img.height), CV\_8UC1, (void\*)pAlg->bin\_img.pdata);

imshow("bin-image", bin);

edge = Mat(cvSize(pAlg->edge\_img.width, pAlg->edge\_img.height), CV\_8UC1, (void\*)pAlg->edge\_img.pdata);

imshow("edge-image", edge);

mvImage imm;

mvImageCreate(&imm, pAlg->com\_img.width, pAlg->com\_img.height, 1);

mvSetComponentRoiArea(imm, det);

mvImageShow("mvImage", imm);

mvImageDestroy(&imm);

int cnt = mvDetRoiMapFilter(det, pAlg->com\_img.width, pAlg->com\_img.height, pAlg->com\_img.pdata);

cout << "cnt=" << cnt << endl;

com = Mat(cvSize(pAlg->com\_img.width, pAlg->com\_img.height), CV\_8UC1, (void\*)pAlg->com\_img.pdata);

///\* 将ccl 的lable 二值化 \*/

/\*for (i = 0; i < pAlg->com\_img.nsize; i++)

{

if (pAlg->com\_img.pdata[i] > 0)

pAlg->com\_img.pdata[i] = 255;

}\*/

imshow("com-image", com);

key = cvWaitKey();

}

}

while (1);

key = cvWaitKey();

printf("stop.\n");

cvWaitKey();

if (1)

{

mvInstanceDelete(pAlg);

}

return (void\*)pAlg;

}

int main(int argc, char\* argv[])

{

int key;

////process

algLocationProcessWrap(MV\_ALG\_TYPE);

key = cvWaitKey(0);

while (key != 'q' && key != 'Q')

{

cvWaitKey(0);

}

return 0;

}

### 2.1.5 注意事项

①初始化算子时，配置文件的文件夹路径要以“/”结束，例如：”D:/imvcfg/”

②存放图片的文件夹必须有图片的列表文件imagelist.txt,并将图片名字写在该文件中，每行一个，不能有空格。

### 2.1.6 算子配置

###################

algType=MV\_ALG\_COMPONENT\_DET1

#edge threshold

edge\_thres=80

conf\_thres=0.1

use\_tracker=0

#alg parameters#

#use filter

use\_filter=0

#minimal object pixels

min\_pixels=0

#maxmal object pixels，if = -1,not use

max\_pixels=50000

#diff val, default: 50

diff\_val=50

#minimal area of object

min\_area=32

#maxmal area, if = -1,not use

max\_area=-1

#minimal gray value

min\_gray=-1

#maxmal gray value

max\_gray=-1

#min\_wh\_rate: min(w,h)/max(w,h)

min\_wh\_rate=0

#max\_wh\_rate

max\_wh\_rate=0

#ppa edge-pixels/area

ppa=0

gr\_thres=0

co\_thres=0

conLikeness=0

#remove edge flag

edge\_flag=0

#use classifier

use\_clasfier=0

#use mapper

use\_mapper=0

#smooth factor

use\_deepl=0

#binary

useOtsu=1

#use smooth=1

useSmooth=1

#use binary,bin metho: 1,2,3,4

useBin=4

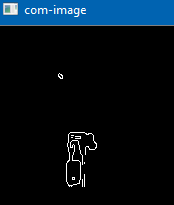
#disp\_level=3281073

### 2.1.7 函数说明：

mvSetComponentRoiArea，设置ROI区域



mvDetRoiMapFilter，检测结果筛选：



# 3. 附录

[1] 《X\_vision SDK快速入门手册》