专业：计算机科学与技术
姓名：卢佳盈
学号：3180103570
日期：2020/12/9

浙江大学实验报告

课程名称：计算机视觉 指导老师：宋明黎 成绩：_____

实验名称：制作个人视频

一、实验目的和要求

调用 CvBox2D **cvFitEllipse2**(const CvArr* points)实现椭圆拟合.

二、实验内容和原理

（简述实验有关的基本原理）

2.1 开发环境

- Windows X64
- Visual Studio 2017
- opencv-3.4.0

2.2 运行方式

双击执行

或命令行输入 hw2.exe 图片路径

如 hw2.exe C:/input.png

目标路径请使用反斜杠

```
D:\openCV\code\Project1\x64\Debug>Project1.exe input.png
chose cvThreshold or cvAdaptiveThreshold?(1 or 0):1
use cvThreshold

D:\openCV\code\Project1\x64\Debug>Project1.exe input.png
chose cvThreshold or cvAdaptiveThreshold?(1 or 0):0
use cvAdaptiveThreshold
```

2.3 原理介绍

【局部二值化】

相关函数：cvFindContours

功能：该函数从二值图像中提取轮廓，返回值为轮廓的数目。

API：

```
void cvAdaptiveThreshold(
    const CvArr* src, CvArr* dst, double max_value,
    int adaptive_method = CV_ADAPTIVE_THRESH_MEAN_C,
    int threshold_type = CV_THRESH_BINARY,
    int block_size = 3, double param1 = 5);
```

src	输入图像
dst	输出图像

max_value	使用 CV_THRESH_BINARY 和 CV_THRESH_BINARY_INV 的最大值	
adaptive_method	自适应阈值算法	
	CV_ADAPTIVE_THRESH_MEAN_C	加权平均取阈值
	CV_ADAPTIVE_THRESH_GAUSSIAN_C	高斯函数取阈值
threshold_type	取阈值类型	
	CV_THRESH_BINARY	if src(x,y)>threshold dst(x,y) = max_value; otherwise dst(x,y)= 0
	CV_THRESH_BINARY_INV	if src(x,y)>threshold dst(x,y) = 0 ; otherwise dst(x,y) = max_value
	CV_THRESH_TRUNC	if src(x,y)>threshold dst(x,y) = threshold; otherwise dst(x,y) = src(x,y)
	CV_THRESH_TOZERO	if (x,y)>threshold dst(x,y) = src(x,y); otherwise dst(x,y) = 0
	CV_THRESH_TOZERO_INV	if src(x,y)>threshold dst(x,y) = 0 ;otherwise dst(x,y) = src(x,y)
block_size	用来计算阈值的象素邻域大小	
param1	与方法有关的参数	

【腐蚀】

相关函数：erode

功能：图像腐蚀

API：

```
void erode(
    const Mat& src, Mat& dst,
    const Mat& element, Point anchor = Point(-1, -1),
    int iterations = 1, int borderType = BORDER_CONSTANT,
    const Scalar& borderValue = morphologyDefaultBorderValue());
```

src	原图像
dst	目标图像
element	腐蚀操作的内核。 如果不指定，默认为一个简单的 3*3 矩阵
anchor	默认为 Point(-1,-1),内核中心点。省略时为默认值
iterations	腐蚀次数。省略时为默认值 1
borderType	推断边缘类型
borderValue	边缘值

【边缘检测】

相关函数：cvFindContours

功能：该函数从二值图像中提取轮廓，返回值为轮廓的数目。

API：

```
int cvFindContours(
```

```
    CvArr * image,
    CvMemStorage * storage,
    CvSeq ** first_contour,
    int header_size = sizeof(CvContour),
    int mode = CV_RETR_LIST,
    int method = CV_CHAIN_APPROX_SIMPLE,
    CvPoint offset = cvPoint(0, 0)
)
```

image	指向 8 位单通道的源图像	
storage	得到的轮廓的存储容器	
first_contour	输出参数，包含指向第一个输出轮廓的指针	
header_size	如果 method=CV_CHAIN_CODE，则序列头的大小设置为 sizeof(CvChain)，否则设置为 sizeof(CvContour)	
mode	提取模式	
	CV_RETR_EXTERNAL	只提取最外层的轮廓
	CV_RETR_LIST	提取所有轮廓，并且放置在 list 中
	CV_RETR_CCOMP	提取所有轮廓，并且将其组织为两层的 hierarchy（层级）:顶层为连通域的外围边界，次层为洞的内层边界
	CV_RETR_TREE	提取所有轮廓，并且重构嵌套轮廓的全部 hierarchy （层级）
method	逼近方法	
	CV_CHAIN_CODE	链码（freeman）的输出轮廓.其它方法输出多边形(定点序列). 链码：用曲线起始点的坐标和边界点方向代码来描述曲线或边界的方法
	CV_CHAIN_APPROX_NONE	将所有点由链码形式翻译(转化）为点序列形式
	CV_CHAIN_APPROX_SIMPLE	压缩水平、垂直和对角分割，即函数只保留末端的象素点
	CV_CHAIN_APPROX_TC89_L1	应用 Teh-Chin 链逼近算法. CV_LINK_RUNS -通过连接为 1 的水平碎片使用完全不同的轮廓提取算法。仅有 CV_RETR_LIST 提取模式可以在本方法中应用
	CV_CHAIN_APPROX_TC89_KCOS	
offset	每一个轮廓点的偏移量	

【查找轮廓】

相关函数：cvFindContours

功能：对给定的一组二维点集作椭圆的最佳拟合(最小二乘意义上的)。返回的结构中 size 表示椭圆轴的整个长度

API：CvBox2D cvFitEllipse2

```
CvBox2D cvFitEllipse2(const CvArr* points);
```

points	点集的序列或数组

三、实验步骤与分析

（每个步骤结合对应部分的源代码分析）

【以灰度图形式读取图像】

```
const char* filename = argv[1];
imageSrc = cvLoadImage(filename, 0)
```

【动态变量的创建】

```
CvMemStorage* storage;
CvSeq* contour;
storage = cvCreateMemStorage(0);
contour = cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq), sizeof(CvPoint), storage);
```

【对灰度图进行二值化】

```
IplImage* imageThreshold = cvCloneImage(imageSrc);
if (threType == 1) {
        cout << "use cvThreshold" << endl;
        cvThreshold(imageSrc, imageThreshold, 125, 255, CV_THRESH_BINARY);
    }
    else {
        cout << "use cvAdaptiveThreshold" << endl;
        cvAdaptiveThreshold(imageSrc, imageThreshold, 255, CV_ADAPTIVE_THRESH_MEAN_C,
CV_THRESH_BINARY, 125, 10);
    }
```

发现如果使用 cvTreshold 函数，会因图片亮度不统一使得二值化结果产生白区或黑区，因此使用局部二值化函数。但当图片中椭圆个数较少时，使用 cvTreshold 函数结果拟合程度更高。

【对二值化结果进行腐蚀处理】

```
IplImage* imageErode = cvCloneImage(imageSrc);
cvErode(imageThreshold, imageErode);
```

为消除二值化图像中存在的噪点，避免轮廓结果中出现空孔，对图像进行腐蚀操作

【从腐蚀结果获取边缘图像】

```
IplImage* imageCanny= cvCloneImage(imageSrc);
cvCanny(imageErode, imageCanny, 100, 150, 3);
```

【从边缘图像获取轮廓】

```
cvFindContours(imageCanny, storage, &contour, sizeof(CvContour),CV_RETR_LIST, CV_CHAIN_APPROX_NONE,
cvPoint(0, 0));
```

【在每一个轮廓中拟合椭圆】

```
IplImage* imageCanny= cvCloneImage(imageSrc);
    cvCanny(imageErode, imageCanny, 100, 150, 3);
    //获取轮廓
    cvFindContours(imageCanny, storage, &contour, sizeof(CvContour),CV_RETR_LIST,
CV_CHAIN_APPROX_NONE, cvPoint(0, 0));
    //ellipse
    IplImage* imageOut = cvCloneImage(imageSrc);
```

```
int i = 0;
for (; contour; contour = contour->h_next) {
    int count = contour->total; // 轮廓中点的数量
    CvPoint center;
    CvSize size;
    CvBox2D box;

    // 为避免出现过小的椭圆，设置点数量的最小值
    if (count < 10)
        continue;
    i++;
    cout << count << "\t";
    CvMat* points_f = cvCreateMat(1, count, CV_32FC2);
    CvMat points_i = cvMat(1, count, CV_32SC2, points_f->data.ptr);
    cvCvtSeqToArray(contour, points_f->data.ptr, CV_WHOLE_SEQ);
    cvConvert(&points_i, points_f);

    //对当前轮廓进行椭圆拟合
    box = cvFitEllipse2(points_f);

    // 椭圆的绘制
    center = cvPointFrom32f(box.center);
    size.width = cvRound(box.size.width*0.5);
    size.height = cvRound(box.size.height*0.5);

    cvEllipse(imageOut, center, size,box.angle, 0, 360,CV_RGB(0, 0, 255), 1, CV_AA, 0);
    cvReleaseMat(&points_f);
}
```

【图像保存】

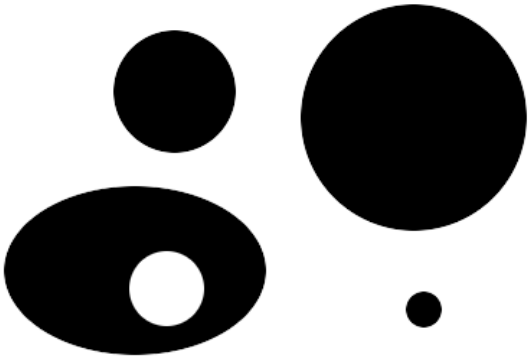
```
cvSaveImage("./output.png", imageOut);
```

四、实验结果

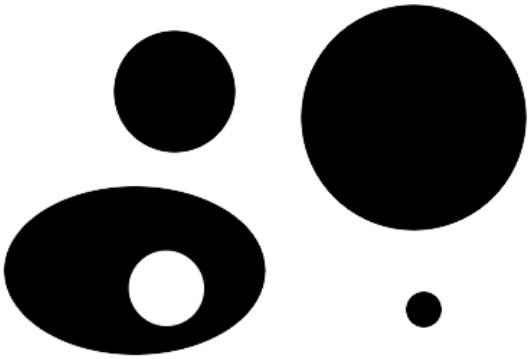
（展示实验用到的输入输出图像等）

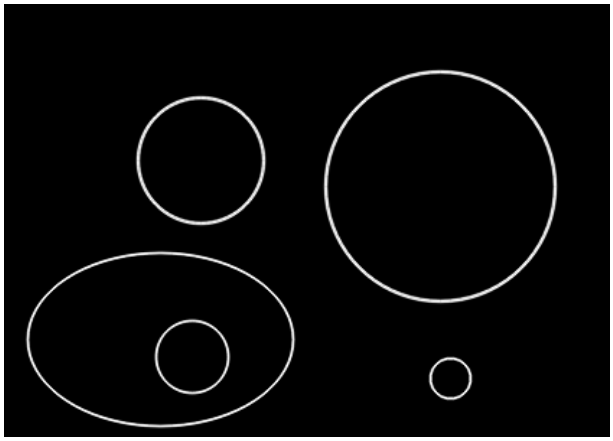
【输入素材-demo1】

thresholdType 输入 1



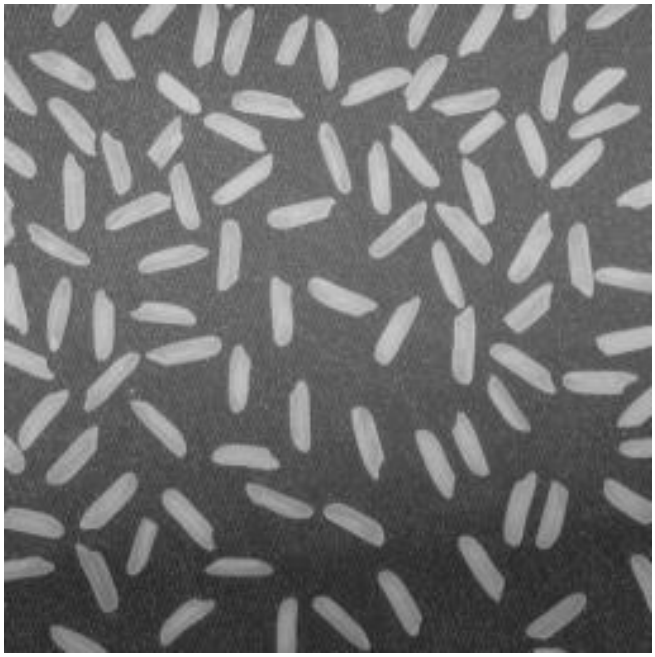
【输出结果-demo1】



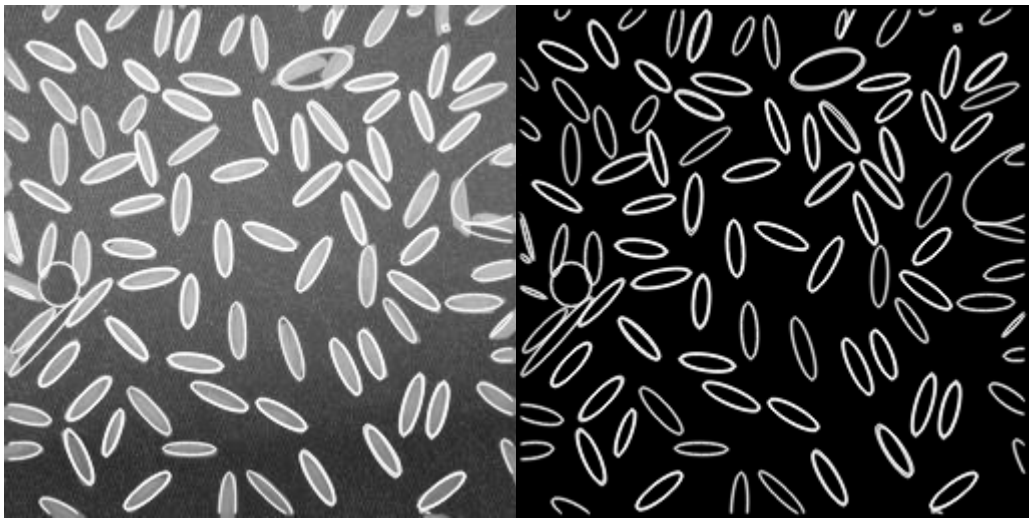


【输入素材-demo2】

thresholdType 输入 0



【输出结果-demo2】



可以看出，算法对于结果的输出是较为正确的，一些错误的区域是由于原图像两个椭圆相邻较近而产生的错误拟合。

五、心得体会

在本次实验中，我使用了较长时间来区分 cvFitEllipse2 与 fitEllipse 函数，发现两者的 API 调用是不同的，当我以 Mat 形式处理图片并使用 fitEllipse 函数时，可以得到相比 cvFitEllipse2 函数更好的拟合效果，这让我百思不得其解，因为这两个函数的功能显然是一样的。在经过很长时间的 debug 之后，发现问题出现在对图像的轮廓化过程中，当以 Mat 形式处理图片时，我使用的是 Canny 函数，而以 IplImage 形式处理图片时，我使用的是 cvThreshold 搭配 cvCanny 函数，前者可以实现较好的阈值处理，而后者因为图像亮度不统一的原因很难对全局进行较好的二值化处理。为了实现实验要求的“使用 cvFitEllipse2 函数实现椭圆拟合”，我对进行轮廓处理前的图像，依次进行了阈值化（cvThreshold、cvThresholdAdaptive）→腐蚀操作后，再进行轮廓处理，最终获得了较好的拟合结果。