# 1、矩

在概率论、统计学甚至计量经济学中常会看到“矩”这个字，那什么是“矩”？

**矩的物理意义：**数学中矩的概念来自物理学。在物理学中，矩是表示距离和物理量乘积的物理量，表征物体的空间分布。由其定义，矩通常需要一个参考点（基点或参考系）来定义距离。如力和参考点距离乘积得到的力矩（或扭矩），原则上任何物理量和距离相乘都会产生力矩，质量，电荷分布等。

**矩的数学意义：**矩是物体形状识别的重要参数指标。在统计学中，矩表征随机量的分布。一阶矩，期望，表位置；二阶矩，方差，表胖瘦；三阶矩，偏度，表歪斜；四阶矩，峰度，表尾巴胖瘦

# 2、图像的矩

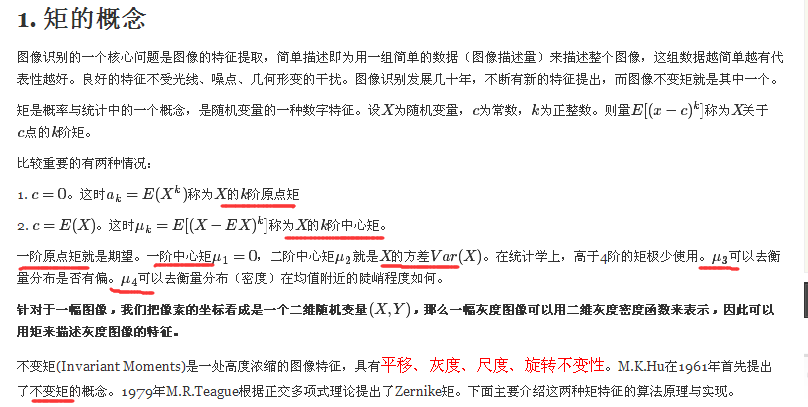
矩是通过对轮廓上所有点进行积分运算（或者认为是求和运算）而得到的一个粗略特征。

图像识别的一个核心问题是图像的特征提取，简单描述即为用一组简单的数据(数据描述量)来描述整个图像，这组数据越简单越有代表性越好。良好的特征不受光线、噪点、几何形变的干扰，图像识别技术的发展中，不断有新的描述图像特征提出，而图像不变矩就是其中一个。

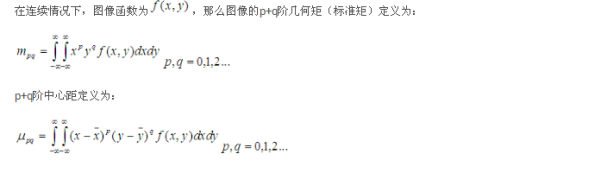
从图像中计算出来的矩通常描述了图像不同种类的几何特征如：大小、灰度、方向、形状等。

图像矩广泛应用于模式识别、目标分类、目标识别与防伪估计、图像编码与重构等领域。介绍图像矩的描述方法，将为目标物体特征的选择和提取提供有力的证据。

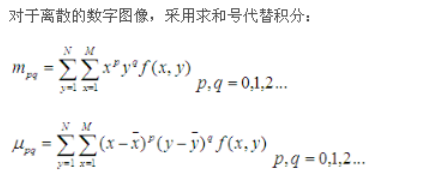
背景知识1： <https://www.cnblogs.com/ronny/p/3985810.html>



opencv中的矩主要包括以下几种：空间矩，中心矩和中心归一化矩。



一副M\*N的数字图像f(x,y)，其p+q阶 几何矩 mpq中心矩表示为



N M分别表示图像的高度和宽度。

对离散的图像f(x,y)为在图像坐标点（x,y）处的灰度值，那么中心矩反映的是图像灰度相对于其灰度质心的分布情况。

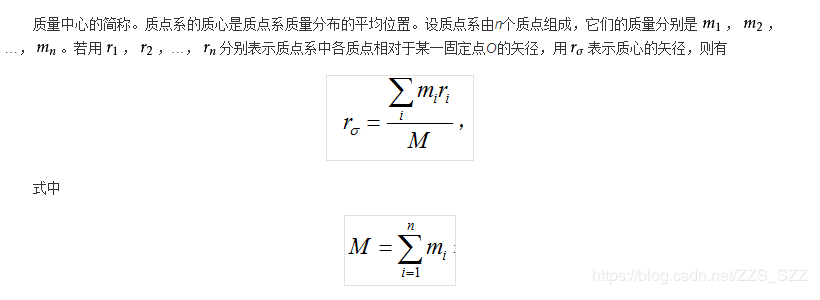
m00 零阶矩表示物体的面积或质量，

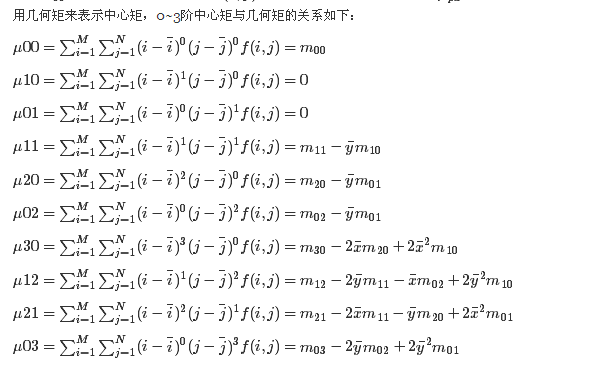
所有的一阶矩和高阶矩除以m00后，与物体的大小无关。

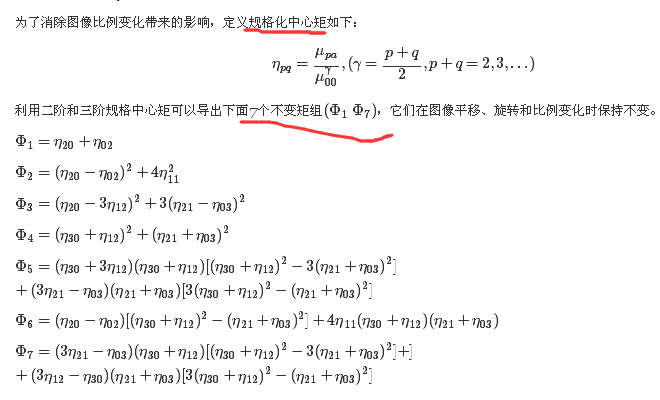
m10对二值图像来讲就是物体上所有点的x坐标的总和

m01对二值图像来讲就是物体上所有点的y坐标的总和

将m00看做是图像的灰度质量，则为图像的质心坐标，质量中心的简称。







矩函数在图像分析中有着广泛的应用，如模式识别、目标分类、目标识别、与方位估计图像编码与重构等，一个从一幅数字图像计算出来的矩集，通常描述了该图像形状的全部特征，并且提供了大量关于该图像不同类型的的几何特征信息，比如大小，方向，位置，以及形状等图像矩的这种特征被广泛地应用到各种图像处理和计算机视觉和机器人技术的领域的目标识别与方位估计中，一阶矩与图像的形状有关，而二阶矩显示曲线围绕直线平均度的扩展程度，三阶矩则是关于平均值的对称性测量，由二阶矩和三阶矩可以导出7组不变矩，而不变矩是图像的统计特性，满足平移，伸缩，旋转均不变性，在图像识别领域得到了广泛的利用。

# 3、计算矩的编程步骤：

1、提取图像的边缘：Canny

2、发现轮廓

3、计算每个轮廓对象的矩：

moments计算图像的中心矩、几何矩、规格化中心矩，最高到三阶

4、计算每个对象的弧长、面积：

arcLength来计算轮廓或者曲线的长度

contoursArea来计算图像的轮廓的面积

m00

## moments函数来计算图像的矩

Moments moments(InputArray array, bool binaryImage=false )

矩用来计算形状的重心，面积，主轴和其他形状特征。

参数：

array是一幅单通道，8-bits的图像，或一个二维浮点数组(Point of Point2f)。

binaryImage用来指示输出图像是否为一幅二值图像，如果是二值图像，则图像中所有非0像素看作为1进行计算。

输出参数：Moments是一个类：

class Moments

{public:

Moments();

Moments(double m00, double m10, double m01, double m20, double m11,

double m02, double m30, double m21, double m12, double m03 );

//Moments类中包括以下数据：

//! @name spatial moments 几何矩

double m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;

//! @name central moments 中心矩

double mu20, mu11, mu02, mu30, mu21, mu12, mu03;

//! @name central normalized moments 归一化中心矩

double nu20, nu11, nu02, nu30, nu21, nu12, nu03;

Moments( const CvMoments& moments );

operator CvMoments() const;

}

## contourArea函数

double contourArea ( InputArray contour,

bool oriented = false

)

contour:是一个向量，二维点，可以是vector或Mat类型，可以输入轮廓数据

oriented:有默认值false，面向区域标识符，如果为true，该函数返回一个带符号的面积，其正负取决于轮廓的方向(顺时针还是逆时针)。根据这个特性可以根据面积的符号来确定轮廓的位置。如果是默认值false，则面积以绝对值的形式返回。

该函数使用Green formula计算轮廓面积，返回面积和非零像素数量如果使用drawContours或fillPoly绘制轮廓，可能导致不同。

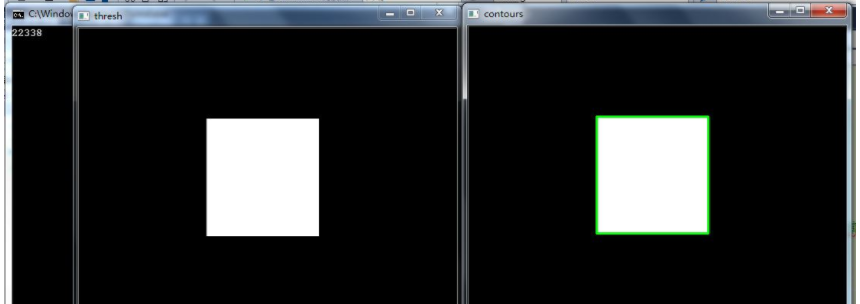
## arcLength函数

double arcLength( InputArray curve, bool closed );

计算轮廓的长度

curve输入曲线数据 closed 是否是封闭曲线

## 3.1计算图像的矩



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

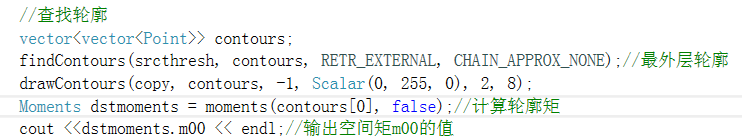
Mat src = imread("e:\\template.png", 1);

Mat copy = src.clone();

Mat srcgray,srcthresh;

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

threshold(srcgray, srcthresh, 100, 255, THRESH\_BINARY);//确保黑中找白

imshow("thresh", srcthresh);  imshow("contours", copy);

waitKey(0);

return 0;

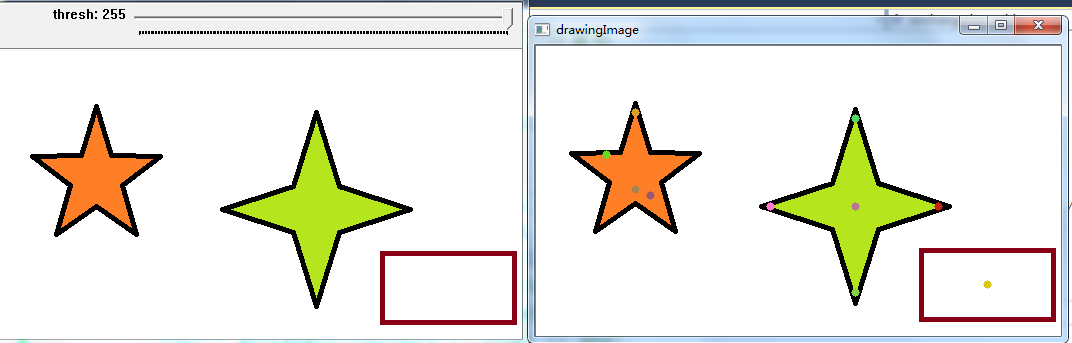
}

## 3.2 滑块+canny+矩

在二值图像的情况下，m00表示区域的面积。

令Xc,Yc表示区域重心的坐标，则：Xc = M10/M00; Yc = M01/M00;

功能：计算轮廓的矩，并根据1阶中心矩得到轮廓的质心



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int thresh = 80;

Mat src, srcgray,edge;

void onchange(int, void \*);

RNG rng(1234);

int main()

{

src = imread("e:\\ju.png", 1);

if (!src.data)

{

cout << "图像加载失败...";

return -1;

}

else

cout << "图像加载成功..." << endl;

imshow("原图", src);

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

GaussianBlur(srcgray, srcgray, Size(3, 3), 0);

createTrackbar("thresh", "原图", &thresh, 255, onchange);

onchange(0, 0);

waitKey(0);

return 0;

}

void onchange(int, void \*)

{

Canny(src, edge, thresh, thresh \* 2, 3);

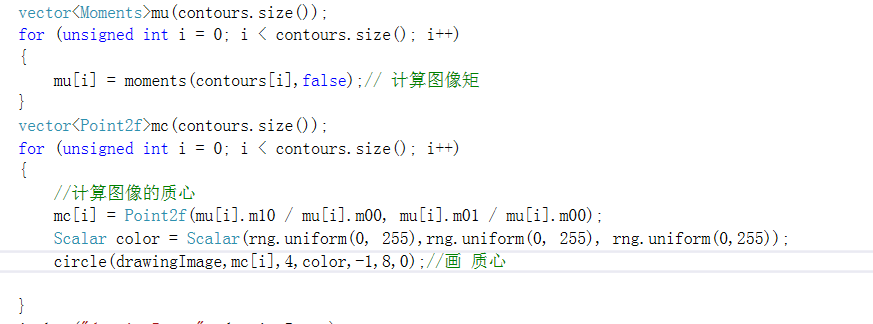
vector<vector<Point>>contours;

vector<Vec4i>hierarchy;

findContours(edge, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

Mat drawingImage = Mat::zeros(edge.size(), CV\_8UC3);

src.copyTo(drawingImage);

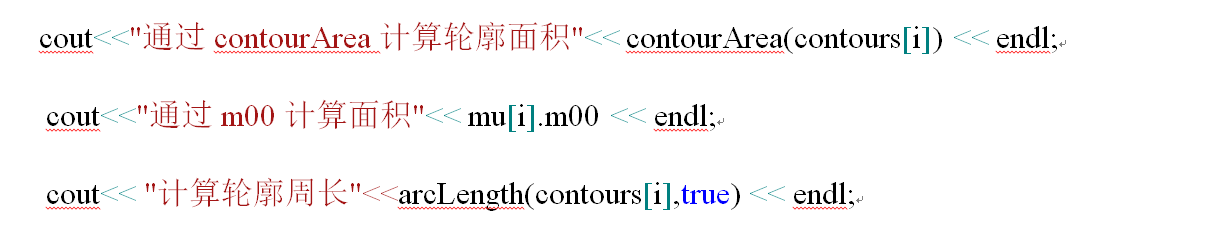


imshow("drawingImage", drawingImage);

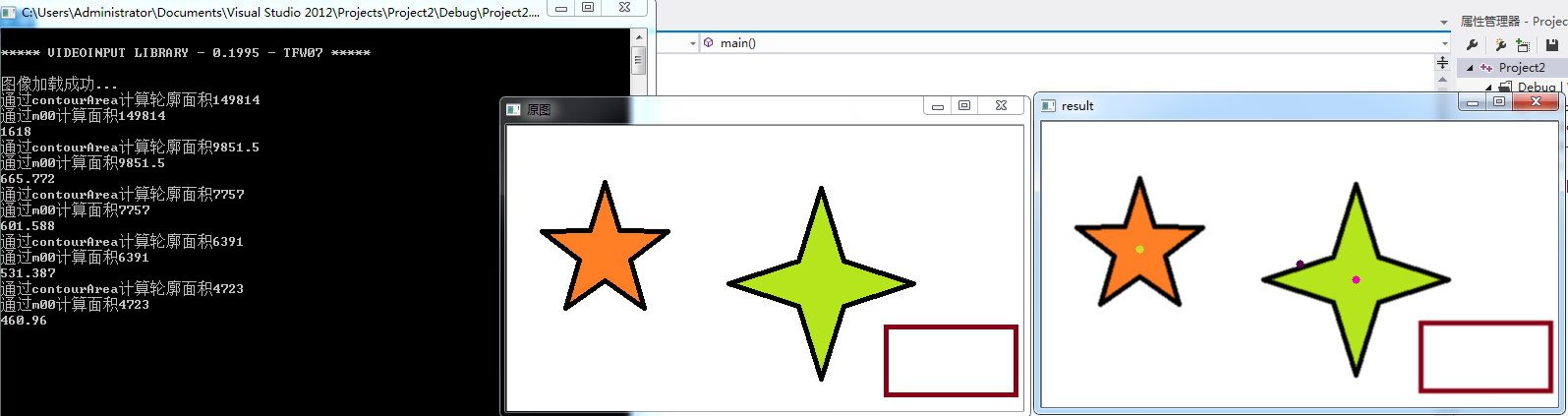
}

//计算图像的质心

mc[i]=Point2f(static\_cast<float>(mu[i].m10/mu[i].m00), static\_cast<float>(mu[i].m01 / mu[i].m00));

放入第二个for循环中： 

## 3.3函数实现计算图像的矩



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

void cacMoments(Mat src);

int main()

{

Mat src = imread("e:\\ju.png", 1);

if (!src.data)

{

cout << "图像加载失败...";

return -1;

}

else

cout << "图像加载成功..." << endl;

imshow("原图", src);

cacMoments(src);

waitKey(0);

return 0;

}

void cacMoments(Mat src)

{

Mat srcgray;

RNG &rng = theRNG();

GaussianBlur(src, src, Size(3, 3), 0);

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

vector<vector<Point>>contours;

vector<Vec4i>hierarchy;

findContours(srcgray, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

vector<Moments>mu(contours.size());

for (unsigned int i = 0; i < contours.size(); i++)

{

mu[i] = moments(contours[i], false);//计算矩

}

vector<Point2f>mc(contours.size());

for (unsigned int i = 0; i < contours.size(); i++)

{

//计算图像的质心

mc[i] = Point2f(static\_cast<float>(mu[i].m10 / mu[i].m00), static\_cast<float>(mu[i].m01 / mu[i].m00));

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

circle(src, mc[i], 4, color, -1, 8, 0);

cout << "通过contourArea计算轮廓面积" << contourArea(contours[i]) << endl;

cout << "通过m00计算面积" << mu[i].m00 << endl;

cout << arcLength(contours[i], true) << endl;

}

imshow("result", src);

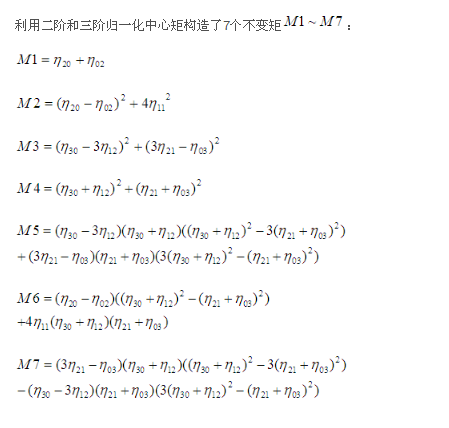
}

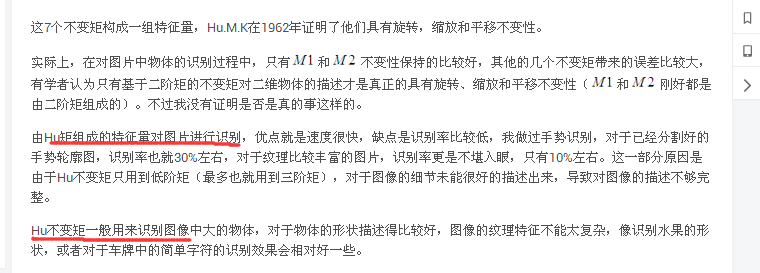
# 4、Hu矩具有平移、旋转、缩放不变的特性。

我们在实际中如何应用这些轮廓呢？比如，有两个轮廓如何进行比较匹配，或者如何比较一个轮廓和一个抽象模板。比较两个轮廓最简洁的方式是比较它们的轮廓矩。

在OpenCV中，还可以很方便的得到Hu不变距，Hu不变矩在图像旋转、缩放、平移等操作后，仍能保持矩的不变性，所以有时候用Hu不变距更能识别图像的特征。

Hu不变矩主要是利用归一化中心矩构造了7个不变特征矩：





不变矩作用主要目的是描述事物（图像）的特征。人眼识别图像的特征往往又表现为“求和”的形式，因此不变矩是对图像元素进行了积分操作。

不变矩能够描述图像整体特征就是因为它具有平移不变形、比例不变性和旋转不变性等性质。然而，另一方面图像的各阶不变矩究竟代表的什么特征很难进行直观的物理解释。

## HuMoments函数

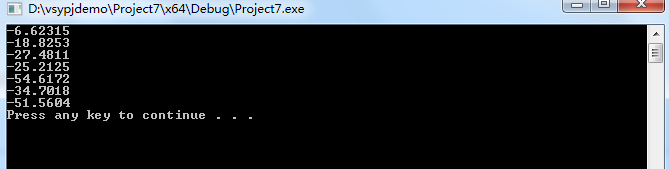
void HuMoments(const Moments& moments, double\* hu)

HuMoments函数用于由中心矩计算Hu矩

参数：moments即为上面一个函数计算得到的moments类型。

输出参数：hu是一个含有7个数的数组。

# 4.1简单使用moments函数、HuMoments函数的代码



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\lena.jpg");

Mat srcgray;

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

Moments mts = moments(srcgray, false);

double hu[7];

HuMoments(mts, hu);

for (int i = 0; i < 7; i++)

{

cout << log(abs(hu[i])) << endl;//最终输出的值为log|Φi|

}

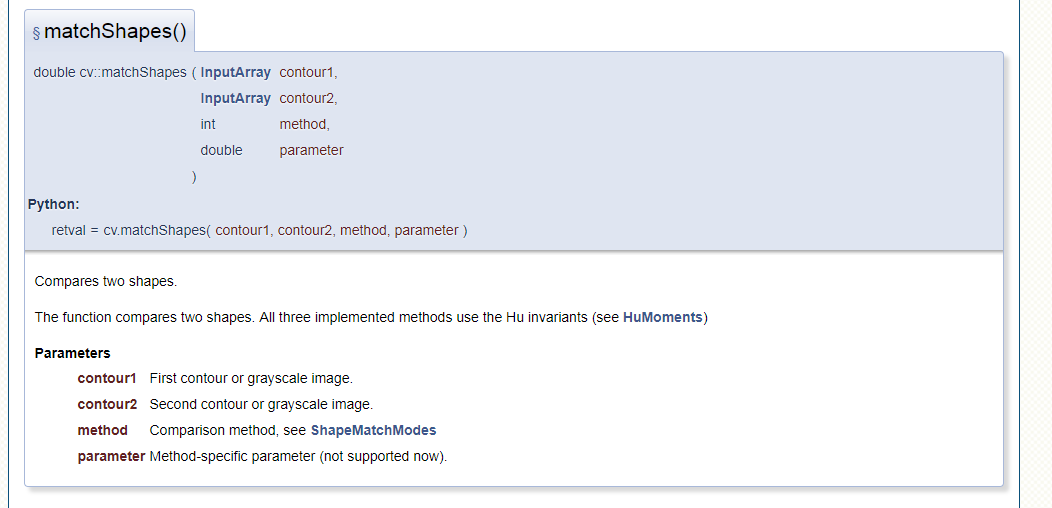
waitKey(0);

system("pause");

return 0;

}

## matchShapes函数



double matchShapes( InputArray contour1, InputArray contour2,int method, double parameter );

opencv提供了输入图像直接进行hu矩匹配的函数，返回的是两个图像或轮廓之间hu矩的相似度。函数返回值代表相似度大小，完全相同的图像返回值是0，返回值最大是1。这可以用在在一堆照片中搜索出两张相同或相同程度最大的图像。

前两个参数contour1，contour2输入“灰度图像”时，并不是想当然的那样，其内容包含待匹配轮廓图案的灰度图；而是使用一行或一列双通道灰度图或者两列灰度图，该图中的每个像素不是什么图片，而是代表多边形轮廓上各节点的X,Y坐标。输入轮廓时每个参数只能是一个轮廓。

第三个参数method是比较的方法，在opencv3.0之前，带CV前缀，opencv4.0,不带CV前缀

CV\_CONTOURS\_MATCH\_I1

CV\_CONTOURS\_MATCH\_I2

CV\_CONTOURS\_MATCH\_I3

enum ShapeMatchModes {

CV\_CONTOURS\_MATCH\_I1 =1,

CV\_CONTOURS\_MATCH\_I2 =2,

CV\_CONTOURS\_MATCH\_I3 =3

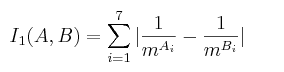
};

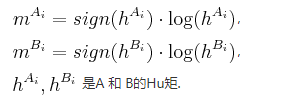
三种输入，这其实就是三种不同的判定物体相似的方法，匹配的返回值也是不一样的其返回值分别通过不同的公式计算得出。

函数 MatchShapes 比较两个形状。 三个实现方法全部使用 Hu 矩

(A ~ object1, B - object2):

method=CONTOUR\_MATCH\_I1:

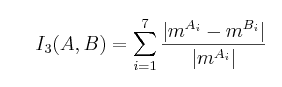




method=CONTOUR\_MATCH\_I2:



method=CONTOUR\_MATCH\_I3:



第四个参数parameter： 方法相应的参数，目前暂时不用。

## 4.2匹配两张图像的相似度

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src1 = imread("e:\\ju.png"); //lena.jpg

Mat src2 = imread("e:\\template.png"); // lena.jpg 两张都是lena图像，输出结果是0

cvtColor(src1, src1, COLOR\_BGR2GRAY);

cvtColor(src2, src2, COLOR\_BGR2GRAY);

threshold(src1, src1, 100, 255, THRESH\_BINARY);

threshold(src2, src2, 100, 255, THRESH\_BINARY);

Canny(src1, src1, 50, 120);

Canny(src2, src2, 50, 120);

vector<vector<Point>>contours1, contours2;

vector<Vec4i>hierarchy1, hierarchy2;

findContours(src1, contours1, hierarchy1, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

findContours(src2, contours2, hierarchy2, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);



return 0;

}