图像处理

一，掩膜操作（mask，亦称作kernal）

实现图像的对比度调整：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

```
filter2D(src, dest, src.depth(), kernal);// Src.depth()表示位图深度
// Define a kernal
Mat kernal = (Mat_<char>(3,3)<<0,-1,0,-1,5,-1,0,-1,0);
```

二，调整图像亮度和对比度

$$g(i,j) = \alpha f(i,j) + \beta$$

α控制对比度β控制亮度

（补）HSV色彩空间

HSV変換とは Q.5で用いた処理であるが、RGBをH(色相)、S(彩度)、V(明度)に変換する手法である。

- Saturation(彩度) 彩度が小さいほど白、彩度が大きいほど色が濃くなる。 0<=S<=1
- Value (明度) 明度が小さいほど黒くなり、明度が大きいほど色がきれいになる。 0<=V<=1
- Hue(色相) 色を0<=H<=360の角度で表し、具体的には次のように表される。

```
赤 黄色  緑  水色  青  紫  赤
0  60  120  180 240 300 360
0  30   60   90 120 150 180 (opencv)
```

三，模糊原理（Smooth/Blur）

最简单常用的操作处理，常用来给图像去噪，核心是卷积操作，因为一般是线性的故又称作线性滤波。

均值滤波：Blur

高斯滤波：GaussianBlur

中值滤波：统计排序滤波， 对椒盐噪声有很好的抑制作用

双边滤波：均值模糊无法克服边缘像素信息丢失缺陷，原因是均值滤波是基于平均权重

高斯滤波部分客服了该缺陷，但无法完全避免，因为没有考虑像素值的不同

高斯双边模糊-是边缘保留的滤波方法，避免了边缘信息丢失，保留了图像的轮廓不变。

```
medianBlur(src, dest, size);
bilateralFilter(src, dest, d, sigmaColor, sigmaSpace);
```

四，形态学操作

- 膨胀： 最大值替换（扩大白色范围）
- 腐蚀： 最小值替换（扩大黑色范围）
- 开操作：先腐蚀，后膨胀。（先取小再取大）

  可以去掉白噪声，断开白边缘。
- 闭操作：先膨胀，后腐蚀。（先取大后取小）

  可以去除黑噪声
- 形态学梯度Morphological Gradient：膨胀减去腐蚀，又称作基本梯度（其他还包括-内部梯度，方向梯度）
- 顶帽：TopHat：原图像与开操作之间的差值图像。(得到白噪声)
- 黑帽：BlackHat：原图像与闭操作之间的差值图像。（得到黑噪声： 颜色取反）

```
getStructuringElement(int shape, Size ksize, Point anchor);
dilate(src, dest, kernal);
erode(src, dest, kernal);
morphologyEx(src, dest, MORPH_GRADIENT, kernal);
```

※getStructuringElement() ：返回一个特定大小与形状的结构元素用于形态学操作，除此之外，也可以自己构建一个任意形状的二进制掩码，作为结构元素

五，图形学

```
Mat hline = getStructuringElement(MORPH_RECT, Size(src.cols / 8, 1)); // 水平线
Mat vline = getStructuringElement(MORPH_RECT, Size(1, src.rows / 8)); // 竖直线
Mat kernal = getStructuringElement(MORPH_RECT, Size(3, 3));           // 提取块

morphologyEx(binValue, dest, MORPH_OPEN, hline/vline/kernal);
```

六，图形金字塔

- 高斯金字塔： 对图像进行降采样
  - 对当层进行高斯模糊

- 删除当前层的偶数行

※高斯不同(Difference of Gaussian-DOG):

同一张图像在不同参数下做高斯模糊之后的结果相减, 得到的图像称作高斯不同.

是图像的内在特征, 在灰度图像增强, 角点检测中经常用到.

- 拉普拉斯金字塔：根据上层降采样图片用来重建一张图片
  -

API

- pyrUp
- pyrDown

七，阈值

- 阈值二值化(threshold binary)
- 截断(truncate)
- 阈值取零(threshold to zero)
- 阈值反取零(threshold to zero reversed)
- 大津阈值(OTSU): 通过大津算法帮助我们寻找 一个合适的阈值
- 三角阈值: 通过三角算法帮助我们寻找 一个合适的阈值

八，Sobel算子（对噪声敏感， 故一般先Blur）

边缘是什么 : 像数值发生跃迁的地方,是图像的显著特征之一.

如何捕捉/提取边缘 - 对图像求一阶导数 - Sobel算子

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

最终的图像梯度:

$$G = \sqrt{G_x^2 + G_y^2} \quad or \quad G = |G_x| + |G_y|$$

Sobel算子: 是离散的微分算子, 用来计算图像灰度的近似梯度.

是高斯平滑和微分求导的统合

又称一阶微分算子, 求导算子，在水平和垂直两个方向上求导，得到图像X和Y两个方向的梯度。

改进版（Scharr）：

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}, G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

- 高斯平滑
- 转灰度
- 求梯度（分别求X，Y方向）
- 振幅图像

九，Laplace算子

在二阶导数的时候， 最大变化处的值为0。据此， 我们可以通过二阶导来提取边缘。

- 高斯模糊
- 转为灰度图
- 拉普拉斯 - 二阶导数计算
- 取绝对值
- 显示结果

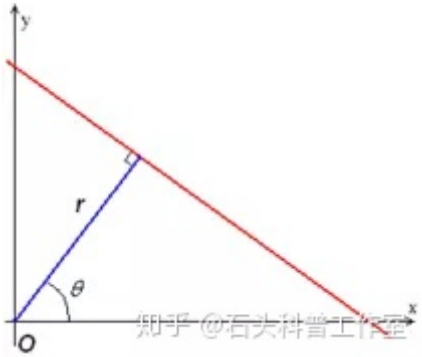
十，Canny边缘检测

- 高斯模糊
- 灰度转换
- 计算梯度（Soble/Scharr）（大小和方向）
- 非最大值抑制
- 高低阈值输出二值图像

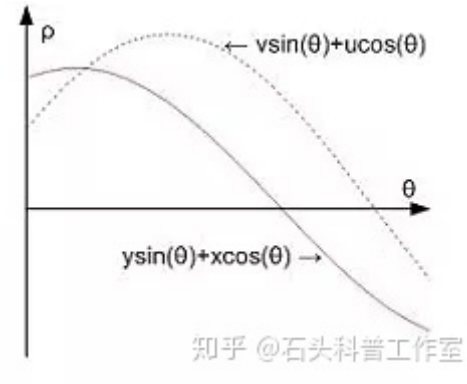
十一， Hough直线检测

基本原理： 笛卡尔空间 -> Hesse仿射坐标空间 放射空间下的高亮（峰值）即为直线。

$$x = \rho cos\theta, \ y = \rho sin\theta$$
$$\rho^2 = x^2 + y^2$$
$$tan\theta = y/x$$



※上图可知一点可以在放射空间下绘制过一点的所有可能直线；



前提条件： 边缘检测已经完成

缺点：无法得到端点信息（一般很少用， 因为得到的是极坐标还需手动逆变换）

👇

统计概率的霍夫直线检测：HoughLineP

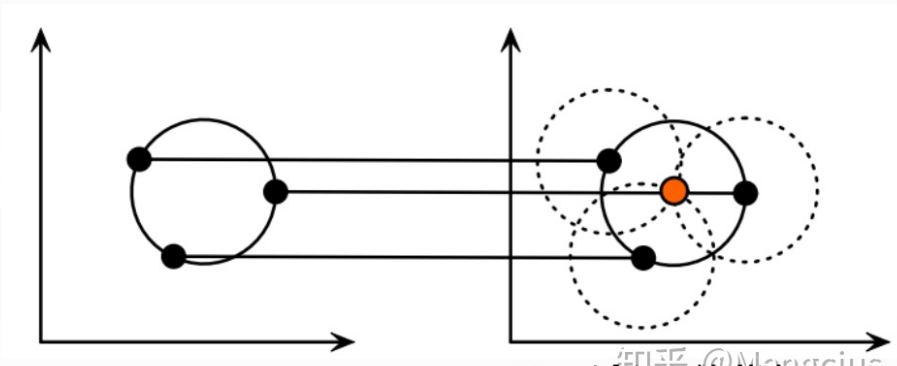
- 灰度图-> 边缘检测 -> （坐标空间转换 -> NMS -> 极值逆变换）

十二， 霍夫圆检测

基本原理：基本原理与直线检测类似，只是点对应的二维极径极角空间被三维的圆心点（x，y）和半径所替代。

从平面坐标圆上的点到极坐标转换的三个参数C(x0,y0,r)其中x0,y0是圆心，r 取一固定值时theta扫描360度，x y 跟着变化， 若多个边缘点对应的三维空间曲线交于一点，则他们在共同圆上，在圆心处有累积最大值，也可以用同样的阈值的方法来判断一个圆是否被检测到。

$$x = a + R\cos(\theta)$$
$$y = b + R\sin(\theta)$$



※注意

- 因为霍夫圆检测对噪声比较敏感，所以首先要对图像做滤波（模糊降噪）。（比如椒盐噪声用中值滤波，其他的也可以用高斯模糊）
- 基于效率考虑，opencv中实现的霍夫变换圆检测是基于图像梯度的实现。
  - Canny检测边缘， 发现可能的圆心

    圆心一定是在圆上的每个点的模向量上, 这些圆上点模向量的交点就是圆心, 霍夫梯度法的第一步就是找到这些圆心, 这样三维的累加平面就又转化为二维累加平面。
  - 从候选圆心开始计算最佳的半径的的大小。

第二步根据所有候选中心的边缘非0像素对其的支持程度来确定半径。

十三，像素重映射

简单来说，就是把图像中每一个像素按照一定的规则映射到另外一张图像的对应位置上去，形成一张新的图像。

十四，直方图（一）

※直方图均衡化是一种提高图像对比度的方法， 拉伸推向灰度值范围。

如果一幅图像的像素占有**很多的灰度**级而且**分布均匀**， 那么这样的图像往往有**很高的**对**比度**和**多**变**的灰度色**调。

直方图均衡化是仅依靠图像的直方图信息自动达成这种效果的变换函数。

实现方法：remap将图像灰度从一个分布映射到另外一个分布， 然后在得到映射后是像素值即可。

※直方图的比较方法：

对两张图像计算直方图H1与H2，归一化得到相同尺度空间，然后可以通过计算H1与H2之间的距离得到两个直方图的相似程度进而比较图像本身的相似程度。

- Correlation ：相关性比较

  CV_COMP_CORREL

$$d(H_1,H_2)=\frac{\sum_I\left(H_1(I)-\bar{H}_1(I)\right)*\left((H_2(I)-\bar{H}_2(I)\right)}{\sqrt{\sum_I\left(H_1(I)-\bar{H}_1(I)\right)^2*\sum_I\left((H_2(I)-\bar{H}_2(I)\right)^2}}$$

- Chi-square：卡方比较

$$d(H_1,H_2)=\sum_i\frac{(H_1(I)-H_2(I))^2}{H_1(I)}$$

  卡方比较来源于卡方检验，卡方检验就是统计样本的实际观测值与理论推断值之间的偏离程度，实际观测值与理论推断值之间的偏离程度就决定卡方值的大小，卡方值越大，越不符合；卡方值越小，偏差越小，越趋于符合，若两个值完全相等时，卡方值就为0，表明理论值完全符合。卡方检验的公式如下，其中fi是观测频率，npi是期望频率，X²是卡方值。

- Intersection： 十字交叉性

$$d(H_1,H_2)=\sum_I min(H_1(I),H_2(I))$$

- Bhattacharyya distance： 巴氏距离

$$d(H_1,H_2)=\sqrt{1-\frac{1}{\sqrt{\bar{H}_1(I)*\bar{H}_2(I)*N^2}}\sum_I\sqrt{H_1(I)*H_2(I)}}$$

在直方图相似度计算时，巴氏距离获得的效果最好，但计算是最为复杂的。

巴氏距离的计算结果，其值完全匹配为0，完全不匹配则为1。

在统计学中，巴氏距离（巴塔恰里雅距离 / Bhattacharyya distance）用于测量两离散概率分布。它常在分类中测量类之间的可分离性。

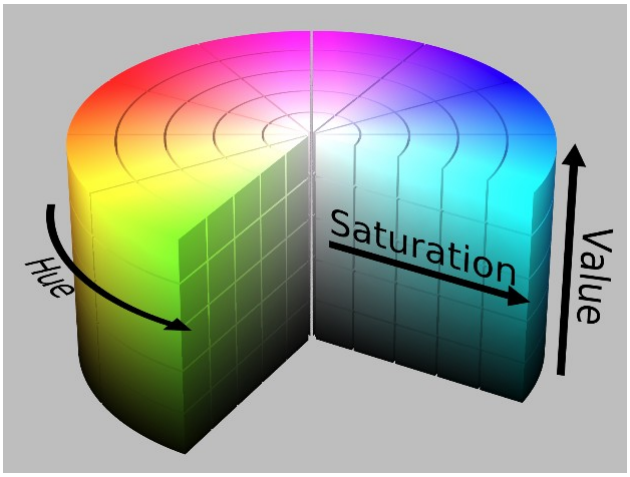
※直方图比较的一般步骤：

- BGR -> HSV， （取H,S分量， 因为对其敏感）；
- 计算图像的直方图，然后归一化到[0~1]之间，用到函数 calcHist() 和 normalize()；
- 使用上述的四种方法之一进行比较，用到函数compareHist()。

十四，直方图（二）

直方图反向投射：

- 反向投影是反映直方图模型在目标图像中的分布情况。
- 简单来说就是用直方图模型去目标图像中寻找是否有相似的对象。
- 通常用HSV色彩空间和HS两个通道直方图模型。



反向投影-步骤：

- 建立直方图模型
- 计算待检测物体的直方图并映射到模型中
- 从模型反向计算生成图像

一些API

- equalizeHist： 直方图均衡化
- split： 将多通道图像分为多个单通道他图像
- calcHist： 计算图像直方信息
- compare

十五，模板匹配

- 什么是模板匹配

  类似卷积，计算匹配度。

十六，轮廓发现

- 输入图像转为灰度图像
- 使用Canny进行边缘检测，得到二值图像
- 使用findContour寻找图像
- 使用drawContour绘制图像

十七，凸包

- 什么是凸包：

  在一个多边形的边缘或者内部任意两点的连线都包含在多边形的边界或者内部。

  包含点集S中所有点的最小的凸多边形（**正式定**义）

- 检测算法
  - Graham扫描算法
    - 首先选择Y方向最低的点作为起始点 p_0。
    - 从p_0开始极坐标扫描， 依次添加$p_1...p_n$（根据极坐标的角度大小， 逆时针方向）
    - 对每个点p_i来说， 如果添加p_i到凸包中导致一个左转向（逆时针方向）则添加该点到凸包；

      反之， 如果导致一个又转向（顺时针方向）则从该凸包中删除p_i。

参考

- 步骤：
  - 获得灰度图
  - -> 转为二值图像
  - findContour得到候选点
  - 凸包
  - 绘制

- 轮廓周围绘制矩形和圆形框
  - approxPoluDP()： 基于RDP算法实现， 目的是减少多边形轮廓点数。
  - boundingRect()：得到轮廓周围最小矩形左上角点坐标和右下角坐标， 绘制一个矩形。
  - minAreaRect()：得到一个旋转的矩形， 返回旋转矩形。
  - minEnclosingCircle()：得到最小区域圆形。
  - fitEllipse()：获得最小位置的椭圆。

- 步骤
  - 二值图像
  - findContour
  - API
  - Draw

十八，圆形矩

- 矩
  - 空间矩（Spatial Moments）：$M_{ij} = \sum_{i,j} P(x,y) * x^i * y^j$

    实质是面积或者质量，可以通过一阶矩来计算质心或重心：

    重心： $\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}}$
  - 中心矩（Central Moments）：$\mu_{ij} = \sum_{i,j} P(x,y) * (x - \bar{x})^i * (y - \bar{y})^j$

    体现图像强度的最大和最小方向，可以构建图像的协方差矩阵。只具有平移不变性，所以用中心矩来做匹配效果一般不会很好。
  - 中心归一化矩（Central Normalized Moments）：$\nu_{ij} = \frac{\mu_{ij}}{m_{00}^{\frac{i+j}{2}+1}}$

    归一化后的矩具有尺度不变性。
  - Hu矩：由于具有尺度，旋转，平移不变性，可以用来做匹配。
    $hu[0] = \eta_{20} + \eta_{02}$
    $hu[1] = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}$
    ...
- 应用（对象测量）
  - moments： 计算矩
  - contourArea：计算面积
  - arcLength：计算弧长
- 步骤
  - 提取图像边缘
  - 发现轮廓
  - 计算每个轮廓的矩
  - 计算每个对象的中心，弧长， 面积

十九，点多边形测试

- 测试一个点是否在给定的多边形内部，边缘或者外部

二十，图像分割

- 距离变换与分水岭介绍

- 距离变换常见算法有两种
  - 不断膨胀/腐蚀得到
  - 基于倒角距离
- 分水岭（Watershed）变换的常见算法

  分水岭是基于地理形态的分析的图像分割算法，模仿地理结构（比如山川、沟壑，盆地）来实现对不同物体的分类。

  - 基于浸泡理论实现
    - 测地线距离（Geodestic Distance）

      图像的灰度空间很像地球表面的整个地理结构，每个像素的灰度值代表高度。其中的灰度值较大的像素连成的线可以看做山脊，也就是分水岭。其中的水就是用于二值化的gray threshold level，二值化阈值可以理解为水平面，比水平面低的区域会被淹没，刚开始用水填充每个孤立的山谷(局部最小值)。

      当水平面上升到一定高度时，水就会溢出当前山谷，可以通过在分水岭上修大坝，从而避免两个山谷的水汇集，这样图像就被分成2个像素集，一个是被水淹没的山谷像素集，一个是分水岭线像素集。最终这些大坝形成的线就对整个图像进行了分区，实现对图像的分割。
    - 分水岭算法的整个过程：
      1. 把梯度图像中的所有像素按照灰度值进行分类，并设定一个测地距离阈值。
      2. 找到灰度值最小的像素点（默认标记为灰度值最低点），让threshold从最小值开始增长，这些点为起始点。
      3. 水平面在增长的过程中，会碰到周围的邻域像素，测量这些像素到起始点（灰度值最低点）的测地距离，如果小于设定阈值，则将这些像素淹没，否则在这些像素上设置大坝，这样就对这些邻域像素进行了分类。
      4. 随着水平面越来越高，会设置更多更高的大坝，直到灰度值的最大值，所有区域都在分水岭线上相遇，这些大坝就对整个图像像素的进行了分区。

      用上面的算法对图像进行分水岭运算，由于噪声点或其它因素的干扰，可能会得到密密麻麻的小区域，即图像被分得太细（over-segmented，过度分割），这因为图像中有非常多的局部极小值点，每个点都会自成一个小区域。

      其中的解决方法：

      1. 对图像进行高斯平滑操作，抹除很多小的最小值，这些小分区就会合并。
      2. 不从最小值开始增长，可以将相对较高的灰度值像素作为起始点（需要用户手动标记），从标记处开始进行淹没，则很多小区域都会被合并为一个区域，这被称为**基于图像标记(mark)的分水岭算法**。

- 相关API
  - distanceTransform()

    - watershed()
- 处理流程
    - 对图进行灰度化和二值化得到二值图像
    - 通过膨胀得到确定的背景区域，通过距离转换得到确定的前景区域，剩余部分为不确定区域
    - 对确定的前景图像进行连接组件处理，得到标记图像
    - 根据标记图像对原图像应用分水岭算法，更新标记图

※补充

获取像素矩阵的指针， 索引i表示第几行。

```
Mat.ptr<uchar>(int i=0);
```

获取当前指针

```
const uchar* current = myImage.ptr<uchar>(row);
```

copyMakeBorder(): pad任意大小的边缘.

-- THE END OF THE FIRST PART