# 课设学习笔记

## 对课题的一些理解

目标分割Target Segmentation：像素级的对前景与背景进行分类，将背景剔除；

目标检测Target Detection：定位目标，确定目标位置及大小；

目标识别Target Recognition：定性目标，确定目标是什么，一种分类；

目标跟踪Target Tracking：追踪目标运动轨迹。

图像分割 与 目标分割？

高斯混合模型（Gaussian Mixed Model，GMM ）的定义：

1999年Stauffer等人提出了经典的混合高斯背景建模法，该方法通常能够很好地适应复杂场景，并通过模型参数的自动计算来调整背景模型，但是由于增加了高斯分布的个数，计算量也增大。在过去的十几年里，提出了很多基于混合高斯模型的改进算法，如：2004年Zivkovic等人提出的高斯模型个数自适应的算法，使得算法效率、鲁棒性得以提升。

对于特定像素点的视频序列可看作时间序列,表示像素颜色值, 如果是RGB彩色空间, 则; 且该时间序列可表示为K个高斯分布的叠加,即混合高斯分布将混合高斯分布按排序, 则前B个高斯分布为背景模型,其中T为阈值, 余下的分布则认为是前景。

K-Nearest，KNN

Geometric Multigid，GMG

## 目标识别的几种基本方法

目标检测简要综述，看个一百遍（https://blog.csdn.net/real\_myth/article/details/52370701）

## 基于混合高斯模型的目标识别可能需要的基本步骤

1. 背景建模

使用高斯混合模型（BackgroundSubtractorMOG2）对背景建模，提取出前景

1. 排除干扰
   1. 阴影：
      1. YUV阴影消除
      2. 色彩特征不变量阴影消除
      3. SHV阴影消除
   2. 光照变化
      1. GMM一直在更新背景模型，对缓慢的光线变化有一定的鲁棒性
   3. 噪声
      1. 通过滤波和形态学运算去除
2. 滤波

使用中值滤波去掉椒盐噪声

1. 形态学运算

闭运算和开运算填充空洞

1. 特征匹配

选出目标

KeyPoint这数据结构中有如下数据结构(http://docs.opencv.org/3.3.0/d2/d29/classcv\_1\_1KeyPoint.html)：

angle：角度，表示关键点的方向，通过Lowe大神的论文可以知道，为了保证方向不变形，SIFT算法通过对关键点周围邻域进行梯度运算，求得该点方向。-1为初值。

class\_id：当要对图片进行分类时，我们可以用class\_id对每个特征点进行区分，未设定时为-1，需要靠自己设定

octave：代表是从金字塔哪一层提取的得到的数据。

pt：关键点点的坐标

response：响应程度，代表该点强壮大小—response代表着该关键点how good，更确切的说，是该点角点的程度。

size：该点直径的大小

## 图像分割的几种具体方法

1. 背景减除

背景估计、更新，定位精确，但对光照和环境变化敏感

1. 帧差法

背景减除的一种，背景模型为上一帧图像，所以不需要对背景建模，对于日照变化等缓慢的光线变化适应性好

1. 平均背景法
2. Codebook背景模型（可以解决动态背景建模的问题）
3. 分水岭算法

## 各种滤波

**线性：**

1. 方框滤波

方框滤波，用K\*K窗口中的像素值平均后输出。以下几种滤波就是窗口中的参数不一样。

1. 均值滤波(Averaging Filtering)

This is done by convolving the image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replaces the central element with this average.通过将图像与一个归一化的方框过滤器进行卷积来实现的,取区域内所有像素的平均值替换中心的像素点，图像会模糊、丢失细节。两种方式实现cv2.blur() ；cv2.boxFilter()（可以自定box）

1. 高斯滤波（Gaussian Filtering）

In this approach, instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used. It is done with the function, cv2.GaussianBlur().Gaussian filtering is highly effective in removing Gaussian noise from the image.高斯滤波的系数满足高斯分布，而均值滤波系数是相同的，高斯滤波对图像中的高斯噪声有很好滤除作用。可以通过cv2.getGaussianKernel().自定义内核。

**非线性：**

1. 中值滤波（Median Filtering）

The function cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise.中值滤波原理是直接把一个区域的像素点中值替换区域的中心点像素，对椒盐噪声非常有效，而且不会像均值滤波那样模糊化太过于严重。

1. 双边滤波（Bilateral Filtering）

As we noted, the filters we presented earlier tend to blur edges. This is not the case for the bilateral filter,cv2.bilateralFilter(), which was defined for, and is highly effective at noise removal while preserving edges. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the a neighborhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is,nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity value and does not consider whether the pixel lies on an edge or not. The resulting effect is that Gaussian filters tend to blur edges, which is undesirable.双边滤波最大的特点是保持边缘锐利，双边滤波的思想是抑制与中心像素值差别太大的像素，输出像素值依赖于邻域像素值的加权合。

## 形态学处理

1. 腐蚀（Erosion）（二值图像的位运算/逻辑运算，以下操作也可以对灰度图实现）

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).腐蚀可以提取前景骨架，平滑前景边缘，消除一些噪声，原理是只要区域内不全是1则变为0，即被腐蚀。腐蚀与细化不同，细化可以用于提取文字骨架，提取图像特征。

1. 膨胀（Dilation）

It is just opposite of erosion. Here, a pixel element is ‘1’ if atleast one pixel under the kernel is ‘1’. So it increases the white region in the image or size of foreground object increases. It is also useful in joining broken parts of an object.与腐蚀相反，可以扩张前景、增加目标区域，也可以连接前景断开的部分。

1. 开运算（Opening）

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above.Here we use the function, cv2.morphologyEx().先腐蚀，再膨胀，可以消除前景区域外的噪声（小白点）。

1. 闭运算（Closing）

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foregroundobjects, or small black points on the object.与开运算相反，先膨胀再腐蚀，可以消除前景区域内的噪声（小黑点）。

1. Morphological Gradient
2. Top Hat
3. Black Hat

## 检测运动目标的几种方法

1. 光流计算法
   1. 光流法是利用运动目标随时间变化的光流特性进行检测，但由于其计算量大，且抗噪性能差，不适用于有较高实时性要求的应用；
2. 帧间差分法
   1. 帧间差分法计算简单，对运动变化区域敏感，但是往往检测出的物体在运动方向上被拉伸，运动目标位置不精确，部分运动目标信息被去除，不能完整地提取运动 目标；
3. 背景减除法（研究阴影的去处非常重要）
   1. 均值法
   2. 中值法
   3. 单高斯分布模型法
   4. 混合高斯分布模型法

## 图像金字塔定义（摘自网络）

我们将一层层的图像比喻为金字塔，层级越高，则图像尺寸越小，分辨率越低。

两种类型的金字塔：

**高斯金字塔：**用于下采样，主要的图像金字塔；

**拉普拉斯金字塔：**用于重建图像，也就是预测残差（我的理解是，因为小图像放大，必须插入一些像素值，那这些像素值是什么才合适呢，那就得进行根据周围像素进行预测），对图像进行最大程度的还原。比如一幅小图像重建为一幅大图像，

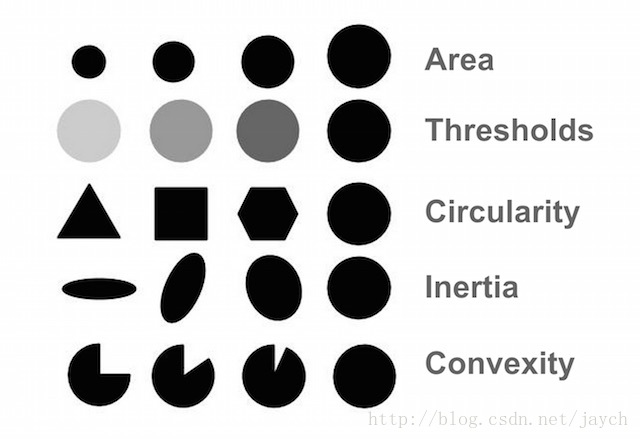
图像金字塔有两个高频出现的名词：上采样和下采样。现在说说他们俩。

**上采样：**就是图片放大（所谓上嘛，就是变大），使用PryUp函数

**下采样：**就是图片缩小（所谓下嘛，就是变小），使用PryDown函数

下采样将步骤：对图像进行高斯内核卷积，将所有偶数行和列去除，下采样就是图像压缩，会丢失图像信息。

上采样步骤：将图像在每个方向放大为原来的两倍，新增的行和列用0填充；使用先前同样的内核（乘以4）与放大后的图像卷积，获得新增像素的近似值。上、下采样都存在一个严重的问题，那就是图像变模糊了，因为缩放的过程中发生了信息丢失的问题。要解决这个问题，就得看拉普拉斯金字塔了。

simpleblob detector的介绍（[https://blog.csdn.net/jaych/article/details/62947714）](https://blog.csdn.net/jaych/article/details/62947714）)

参数介绍（<https://blog.csdn.net/guyuealian/article/details/72026839>）





在*ubuntu*中*clock*返回的时间不准，实测与*time.time*差三倍左右，在新版本中不推荐使用*clock*

time.perf\_counter()

Return the value (in fractional seconds) of a performance counter, i.e. a clock with the highest available resolution to measure a short duration. It does include time elapsed during sleep and is system-wide. The reference point of the returned value is undefined, so that only the difference between the results of consecutive calls is valid.

New in version 3.3.

time.process\_time()

Return the value (in fractional seconds) of the sum of the system and user CPU time of the current process. It does not include time elapsed during sleep. It is process-wide by definition. The reference point of the returned value is undefined, so that only the difference between the results of consecutive calls is valid.

New in version 3.3.