# Opencv3.2算法实现

## orb

orb的源码地址：opencv3.2\opencv-3.2.0\modules\features2d\src

orb使用方法：

Ptr<ORB> orb = ORB::create();

vector<KeyPoint> Keypoints1,Keypoints2;

Mat descriptors1,descriptors2;

orb->detectAndCompute(rgbd1, Mat(), Keypoints1, descriptors1);

### 1.1关于金字塔存储的方式

Orb算法中使用矩阵变量imagePyramid这来存储不同层的金字塔图像，但是这个存储的方式有点神奇下面我们开始讲解。

首先我们这里假定我们输入的图像宽是752，高是480，边界是默认的31，这样比较具体也比较好讲：



上图就是imagePyramid的存储结构，不知道为什么作者一定要将这个矩阵的列数必须是16的倍数因此从814变成了816，我感觉没有必要啊，难道是因为能够快速计算？

layerInfo变量存储的就是各层图像的不包含边界的也就是图中蓝色矩形的数据，layerOfs存储的是各层图像的起始地址，如上图中黄色圆圈所示。

上图中整个黑色的方框表示的就是imagePyramid数据结构，红色方框表示的是添加的边界。1.2 每层金字塔图像保留的最多特征点个数

假设我们总的保留特征点的个数是N，每层的缩放系数是f(对应程序中是1.2)，我们金字塔层数为o(对应程序中是8层)则第1到o-1层的图像保留的最多特征点个数是：



其中sum=(就是等比数列之和)

第八层的最多保留特征点个数=N-前面层数的最大特征点个数保留之和

## 2、sift

sift源码地址：opencv3.2\opencv\_contrib-3.2.0\modules\xfeatures2d\src

sift使用方法：

Ptr<Feature2D> sift = xfeatures2d::SIFT::create(50);  
 sift->detectAndCompute(src,Mat(),keypointsa,a);

目前sift算法还没有来得及看源码只是对算法有了大概的了解。

## 光流

参考博客：<http://blog.sina.com.cn/s/blog_74a459380101obhr.html>

opencv提供了如下光流算法：

**calcOpticalFlowPyrLk**: opencv3.2\opencv-3.2.0\modules\video\src\lkpyramid.cpp，Pyramidal Implementation of the Lucas Kanade Feature TrackerDescription of the algorithm.是稀疏光流，是LK光流的改进方法。

**cvCalcOpticalFlowFarneback**:文件所在位置与上面相同。利用Gunnar Farneback的算法计算全局性的稠密光流算法（即图像上所有像素点的光流都计算出来），由于要计算图像上所有点的光流，故计算耗时，速度慢。它的核心思想主要源于Two-Frame Motion Estimation Based on PolynomialExpansion论文。

**calcOpticalFlowBM**：opencv3.2\opencv-3.2.0\modules\cudalegacy\src\bm.cpp, 通过块匹配的方法来计算光流。

**calcOpticalFlowSF：**opencv3.2\opencv\_contrib-3.2.0\modules\optflow\src\simpleflow.cpp,

impleFlow: A Non-iterative, Sublinear Optical FlowAlgorithm2012年的文章

**calcOptical****FlowHS:**计算速度较慢, Horn-Schunck, Determining Optical Flow,**好像opencv3中没有了这种方法**。稠密光流。

### 2.1 calcOpticalFlowPyrLk函数分析：

参考博客：<http://www.cnblogs.com/gnuhpc/archive/2012/12/04/2802124.html>

<https://blog.csdn.net/banyao2006/article/details/39484113>

#### cv::detail::LKTrackerInvoker::operator()函数

其实整个函数的分析都落在了上面这个operator()函数中：这我们假设我们的原始图像是752(宽)\*480(高)这样比较方便具体的讲解。

其实这个函数由一个大的for循环构成，而这个大的for循环又包含了三个小的for循环下面逐一讲解这大的for循环和三个小的for循环。



这里，，

# 相机标定calibrateCamera函数分析

所在位置/opencv-3.2.0/modules/calib3d/src/calibration.cpp文件calibrateCamera函数。

# epnp函数

所在位置/opencv-3.2.0/modules/calib3d/src/epnp.cpp compute\_pose函数