1,深度图里有很多“洞”。因为RGB-D相机不是万能的，它有一个探测距离的限制啦！太远或太近的东西都是看不见的呢。关于这些“洞”，我们暂时睁一只眼闭一只眼，不去理它。以后我们也可以靠**双边bayes滤波器去填这些洞**。但是！这是RGB-D相机本身的局限性。软件算法顶多给它修修补补，并不能完全弥补它的缺陷。

不过请你放心，在我们给出的这两个图中，都进行了预处理。你可以认为**“深度图就是彩色图里每个像素距传感器的距离”**啦！

*s*是scaling factor，**即深度图里给的数据与实际距离的比例**。由于深度图给的都是short (mm单位)，*s*通常为1000。

// 使用智能指针，创建一个空点云。这种指针用完会自动释放。

在OpenCV2里，图像是以矩阵(cv::MAt)作为基本的数据结构。Mat结构既可以帮你管理内存、像素信息，还支持一些常见的矩阵运算，是非常方便的结构。彩色图像含有R,G,B三个通道，每个通道占8个bit（也就是unsigned char），故称为8UC3（8位unsigend char, 3通道）结构。而深度图则是单通道的图像，每个像素由16个bit组成（也就是C++里的unsigned short），像素的值代表该点离传感器的距离。通常1000的值代表1米，所以我们把camera\_factor设置成1000. 这样，深度图里每个像素点的读数除以1000，就是它离你的真实距离了。

2,我们对上述的描述子进行匹配。在OpenCV中，你需要选择一个匹配算法，例如**粗暴式（bruteforce），近似最近邻（Fast Library for Approximate Nearest Neighbour, FLANN）等等。**这里我们构建一个**FLANN的匹配算法**：

不管如何说，仅靠描述子的匹配似乎是太多了些，把许多不相似的东西也匹配起来了。(由于这两个图像只有水平旋转，所以水平的匹配线才是对的，其他的都是误匹配）。因此，需要筛选一下这些匹配，例如，把distance太大的给去掉（源文件69到88行）。

　　筛选的准则是：去掉大于最小距离四倍的匹配。

　　小萝卜：为什么是四倍呢？

师兄：这只是个经验数值啦，就像你平时都买一斤半的枣糕呀，为什么不买两斤呢？

小萝卜：对了，那个Ransac和inlier是什么呢？

　　师兄：尽管经过了筛选，我们提供的匹配里还是存在误匹配的情况。根据误匹配计算运动是不靠谱的。这时该怎么办呢？OpenCV会利用一种“随机采样一致性”（Random Sample Consensus）的思路（见https://en.wikipedia.org/wiki/RANSAC）。意思即为，**在现有的匹配中随机取一部分，估计其运动。因为正确的匹配结果肯定是相似的，而误匹配的结果肯定满天乱飞。只要把收敛的结果取出来即可。**

小萝卜：这个就叫做“幸福的家庭都是相似的，不幸的家庭各有各的不幸”吧。

由于我们的程序变复杂了，出现了一些内部的参数，如特征点类型，筛选准则，ransac参数等等。如果我们把这些参数值定义在源代码里，那么每修改一次，就要重新编译一遍程序。这非常麻烦。所以，我们希望把参数定义在外部文件中，在程序刚开始时读取此文件。这样一来，只要更改此文件即可完成参数的调节，不必重新编译程序了。因此，请你完成一个读取参数的类，放进slamBase中。

3,该矩阵的左上部分是一个3×3的旋转矩阵，它是一个正交阵。右上部分是3×1的位移矢量。左下是3×1的缩放矢量，在SLAM中通常取成0，**因为环境里的东西不太可能突然变大变小（又没有缩小灯）**。右下角是个1. 这样的一个阵可以对点或者其他东西进行齐次变换：

对！OpenCV认为旋转矩阵*R*，虽然有3×3那么大，自由变量却只有三个，不够节省空间。所以在OpenCV里使用了一个向量来表达旋转。向量的方向是旋转轴，大小则是转过的弧度.

我们已经实现了一个只有两帧的SLAM程序。然而，也许你还不知道，这已经是一个视觉里程计(Visual Odometry)啦！只要不断地把进来的数据与上一帧对比，就可以得到完整的运动轨迹以及地图了呢！

4,在得到匹配之后，我们判断了匹配是否成功，并把失败的数据丢弃。为什么这样做呢？因为之前的算法，对于任意两张图像都能做出一个结果。对于无关的图像，就明显是不对的。所以要去除匹配失败的情形。

如何检测匹配失败呢？我们采用了三个方法：

去掉goodmatch太少的帧，最少的goodmatch定义为：

min\_good\_match=10

去掉solvePnPRASNAC里，inlier较少的帧，同理定义为：

min\_inliers=5

去掉求出来的变换矩阵太大的情况。因为假设运动是连贯的，两帧之间不会隔的太远： max\_norm=0.3　如何知道两帧之间不隔太远呢？我们计算了一个度量运动大小的值：‖Δ*t*‖+min(2*π*−‖*r*‖,‖*r*‖)。**它可以看成是位移与旋转的范数加和**。当这个数大于阈值max\_norm时，我们就认为匹配出错了。

5,我们介绍了如何使用两两匹配，搭建一个视觉里程计。那么，这个里程计有什么不足呢？

1. 一旦出现了错误匹配，整个程序就会跑飞。
2. 误差会累积。常见的现象是：相机转过去的过程能够做对，但转回来之后则出现明显的偏差。
3. 效率方面不尽如人意。在线的点云显示比较费时。

　　累积误差是里程计中不可避免的，后续的相机姿态依赖着前面的姿态。想要保证地图的准确，必须要保证每次匹配都精确无误，而这是难以实现的。所以，我们希望用更好的方法来做slam。不仅仅考虑两帧的信息，而要把所有整的信息都考虑进来，成为一个全slam问题（full slam）。

1. 为什么说slam里的BA问题稀疏呢？因为**同样的场景很少出现在许多位置中**。这导致上面的pose graph中，图*G*离全图很远，只有少部分的节点存在直接边的联系。这就是姿态图的稀疏性。

7.

**关键帧的提取。**把每一帧都拼到地图是去是不明智的。因为帧与帧之间距离很近，导致地图需要频繁更新，浪费时间与空间。所以，我们希望，当机器人的运动超过一定间隔，就增加一个“关键帧”。最后只需把关键帧拼到地图里就行了。

**回环的检测。**回环的本质是识别曾经到过的地方。最简单的回环检测策略，就是把新来的关键帧与之前所有的关键帧进行比较，不过这样会导致越往后，需要比较的帧越多。所以，稍微快速一点的方法是在过去的帧里随机挑选一些，与之进行比较。更进一步的，也可以用图像处理/模式识别的方法计算图像间的相似性，对相似的图像进行检测。

把这两者合在一起，就得到了我们slam程序的基本流程。以下为伪码：

初始化关键帧序列：F并将第一帧f0放入F· (2)对于新来的一帧I，计算F中最后一帧与I的运动，并估计该运动的大小e 。有以下几种可能性：

若e>Eerror ，说明运动太大，可能是计算错误，丢弃该帧；  若没有匹配上（match太少），说明该帧图像质量不高，丢弃；  若e<Ekey ，说明离前一个关键帧很近，同样丢弃； 剩下的情况，只有是特征匹配成功，运动估计正确，同时又离上一个关键帧有一定距离，则把I作为新的关键帧，进入回环检测程序： **近距离回环：**匹配I与F末尾m 个关键帧。匹配成功时，在图里增加一条边。 **随机回环**：随机在F里取n个帧，与I 进行匹配。若匹配上，在图里增加一条边。 将I放入F末尾。若有新的数据，则回(2)； 若无，则进行优化与地图拼接。

回环检测是很怕"false positive"的，即“将实际上不同的地方当成了同一处”，这会导致地图出现明显的不一致。所以，在使用g2o时，**要在边里添加"robust kernel"**，保证一两个错误的边不会影响整体结果。

**/////////////////////////////////////////////////////////////////**

仅仅生成链接库，不生成可执行文件

ADD\_LIBRARY( slambase slamBase.cpp )

TARGET\_LINK\_LIBRARIES( slambase

${OpenCV\_LIBS}

${PCL\_LIBRARIES} )

CMakelists.txt添加自己生成的连接库

<https://blog.csdn.net/KYJL888/article/details/85109782>

添加自己的链接库，并生成可执行文件

ADD\_EXECUTABLE(detectFeatures detectFeatures.cpp)

TARGET\_LINK\_LIBRARIES(detectFeatures ${OpenCV\_LIBS} ${PCL\_LIBRARIES} **libslambase.a** )

**RGBSLAM2 总结**

表示在机器时间1305031910.765238采集了一张RGB图像，存放于rgb/1305031910.765238.png中。

这种存储方式的一个特点是，没有直接的rgb-depth一一对应关系。由于采集时间的差异，几乎没有两张图像是同一个时刻采集的。然而，我们在处理图像时，需要把一个RGB和一个depth当成一对来处理。所以，我们需要一步预处理，找到rgb和depth图像的一一对应关系。

TUM为我们提供了一个工具来做这件事，详细的说明请看：http://vision.in.tum.de/data/datasets/rgbd-dataset/tools 该网页整理了一些常用工具，包括时间配对，ground-truth误差比对、图像到点云的转换等。对于现在预处理这一步，我们需要的是一个 associate.py 文件。

如果读者熟悉python，就很容易看懂它的用法。实际上，只要给它两个文件名即可，它会输出一个匹配好的序列，像这样

python associate.py rgb.txt depth.txt

对！程序默认时间差在0.02内的就可以当成一对图像。为了保存这个结果，我们可以把它输出到一个文件中去，如：

python associate.py rgb.txt depth.txt > associate.txt

**第二件事，因为外部那个运动捕捉装置的记录频率比较高，得到的轨迹点也比图像密集很多，如何查找每个图像的真实位置呢？**

python associate.py associate.txt groundtruth.txt > associate\_with\_groundtruth.txt

这时，我们的新文件 associate\_with\_groundtruth.txt 中就含有每个帧的位姿信息了：

我们**把这个类的指针定义成了shared\_ptr**，以后尽量使用这个指针管理此类的对象，**这样可以免出一些变量作用域的问题。并且，智能指针可以自己去delete，不容易出现问题。**