

## 理解图优化，一步步带你看懂g2o框架

小白：师兄师兄，最近我在看SLAM的优化算法，有种方法叫“图优化”，以前学习算法的时候还有一个优化方法叫“凸优化”，这两个不是一个东西吧？

师兄：哈哈，这个问题有意思，虽然它们中文发音一样，但是意思差别大着呢！我们来看看英文表达吧，图优化的英文是 graph optimization 或者 graph-based optimization，你看，它的“图”其实是数据结构中的graph。而凸优化的英文是 convex optimization，这里的“凸”其实是凸函数的意思，所以单从英文就能区分开它们。

小白：原来是这样，我看SLAM中图优化用的很多啊，我看了一下高博的书，还是迷迷糊糊的，求科普啊师兄

师兄：图优化真的蛮重要的，概念其实不负责，主要是编程稍微有点复杂。。

小白：不能同意更多。。，那个代码看的我一脸懵逼

## 图优化有什么优势？

师兄：按照惯例，我还是先说说图优化的背景吧。SLAM的后端一般分为两种处理方法，一种是以扩展卡尔曼滤波（EKF）为代表的滤波方法，一种是以图优化为代表的非线性优化方法。不过，目前SLAM研究的主流热点几乎都是基于图优化的。

小白：据我所知，滤波方法很早就有了，前人的研究也很深。为什么现在图优化变成了主流了？

师兄：你说的没错。滤波方法尤其是EKF方法，在SLAM发展很长的一段历史中一直占据主导地位，早期的大神们研究了各种各样的滤波器来改善滤波效果，那会入门SLAM，EKF是必须要掌握的。顺便总结下滤波方法的优缺点：

优点：在当时计算资源受限、待估计量比较简单（特征点）的情况下，EKF为代表的滤波方法比较有效，经常用在激光SLAM中。

缺点：它的一个大缺点就是存储量和状态量是平方增长关系，因为存储的是协方差矩阵，因此不适合大型场景。而现在基于视觉的SLAM方案，路标点（特征点）数据很大，滤波方法根本吃不消，所以此时滤波的方法效率非常低。

小白：原来如此。那图优化在视觉SLAM中效率很高吗？

师兄：这个其实说来话长了。很久很久以前，其实就是不到十年前吧(感觉好像很久)，大家还都是用滤波方法，因为在图优化里，Bundle Adjustment（后面简称BA）起到了核心作用。但是那会SLAM的研究者们发现包含大量特征点和相机位姿的BA计算量其实很大，根本没办法实时。

小白：啊？后来发生了什么？（认真听故事ing）

师兄：后来SLAM研究者们发现了其实在视觉SLAM中，虽然包含大量特征点和相机位姿，但其实BA是稀疏的，稀疏的就好办了，就可以加速了啊！比较代表性的就是2009年，几个大神发表了自己的研究成果《SBA：A software package for generic sparse bundle adjustment》，而且计算机硬件发展也很快，因此基于图优化的视觉SLAM也可以实时了！

小白：厉害厉害！向大牛们致敬！

## 图优化是什么？

---

小白：图优化既然是主流，那我可以跳过滤波方法直接学习图优化吧，反正滤波方法也看不懂。。

师兄：额，图优化确实是主流，以后有需要你可以再去看滤波方法，那我们今天就只讲图优化好啦

小白：好滴，那问题来了，究竟什么是图优化啊？

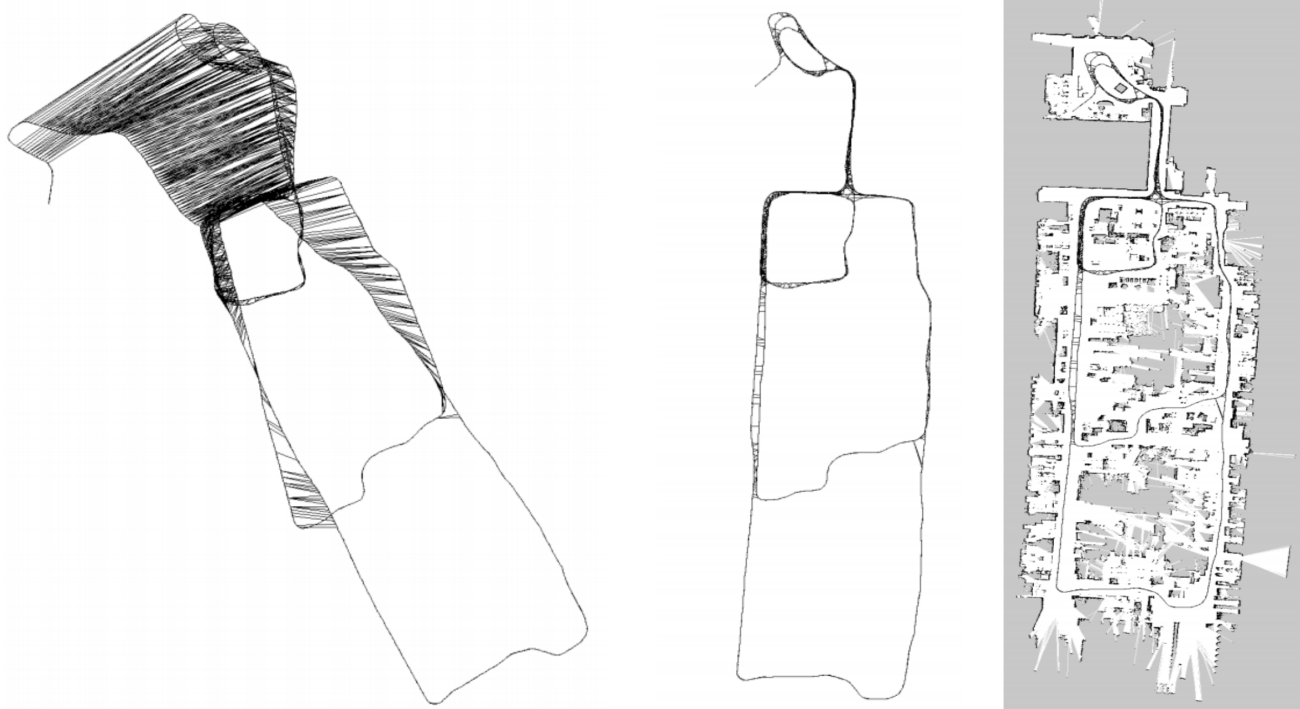
师兄：图优化里的图就是数据结构里的图，一个图由若干个顶点（vertex），以及连接这些顶点的边（edge）组成，给你举个例子

比如一个机器人在房屋里移动，它在某个时刻  $t$  的位姿（pose）就是一个顶点，这个也是待优化的变量。而位姿之间的关系就构成了一个边，比如时刻  $t$  和时刻  $t+1$  之间的相对位姿变换矩阵就是边，边通常表示误差项。

在SLAM里，图优化一般分解为两个任务：

- 1、构建图。机器人位姿作为顶点，位姿间关系作为边。
- 2、优化图。调整机器人的位姿（顶点）来尽量满足边的约束，使得误差最小。

下面就是一个直观的例子。我们根据机器人位姿来作为图的顶点，这个位姿可以来自机器人的编码器，也可以是ICP匹配得到的，图的边就是位姿之间的关系。由于误差的存在，实际上机器人建立的地图是不准的，如下图左。我们通过设置边的约束，使得图优化向着满足边约束的方向优化，最后得到了一个优化后的地图（如下图中所示），它和真正的地图（下图右）非常接近。



小白：哇塞，这个图优化效果这么明显啊！刚开始误差那么大，最后都校正过来了

师兄：是啊，所以图优化在SLAM中举足轻重啊，一定得掌握！

小白：好，有学习的动力了！我们开启编程模式吧！

## 先了解g2o 框架

---

师兄：前面我们简单介绍了图优化，你也看到了它的神通广大，那如何编程实现呢？

小白：对啊，有没有现成的库啊，我还只是个“调包侠”。。

师兄：这个必须有啊！在SLAM领域，基于图优化的一个用的非常广泛的库就是g2o，它是General Graphic Optimization 的简称，是一个用来优化非线性误差函数的c++框架。这个库可以满足你调包侠的梦想~



# g2o:General Graph Optimization

小白：哈哈，太好了，否则打死我也写不出来啊！那这个g2o怎么用呢？

师兄：我先说安装吧，其实g2o安装很简单，参考GitHub上官网：

<https://github.com/RainerKuemmerle/g2o>

按照步骤来安装就行了。需要注意的是安装之前确保电脑上已经安装好了第三方依赖。

小白：好的，这个看起来很好装。不过问题是，我看相关的代码，感觉很复杂啊，不知如何下手啊

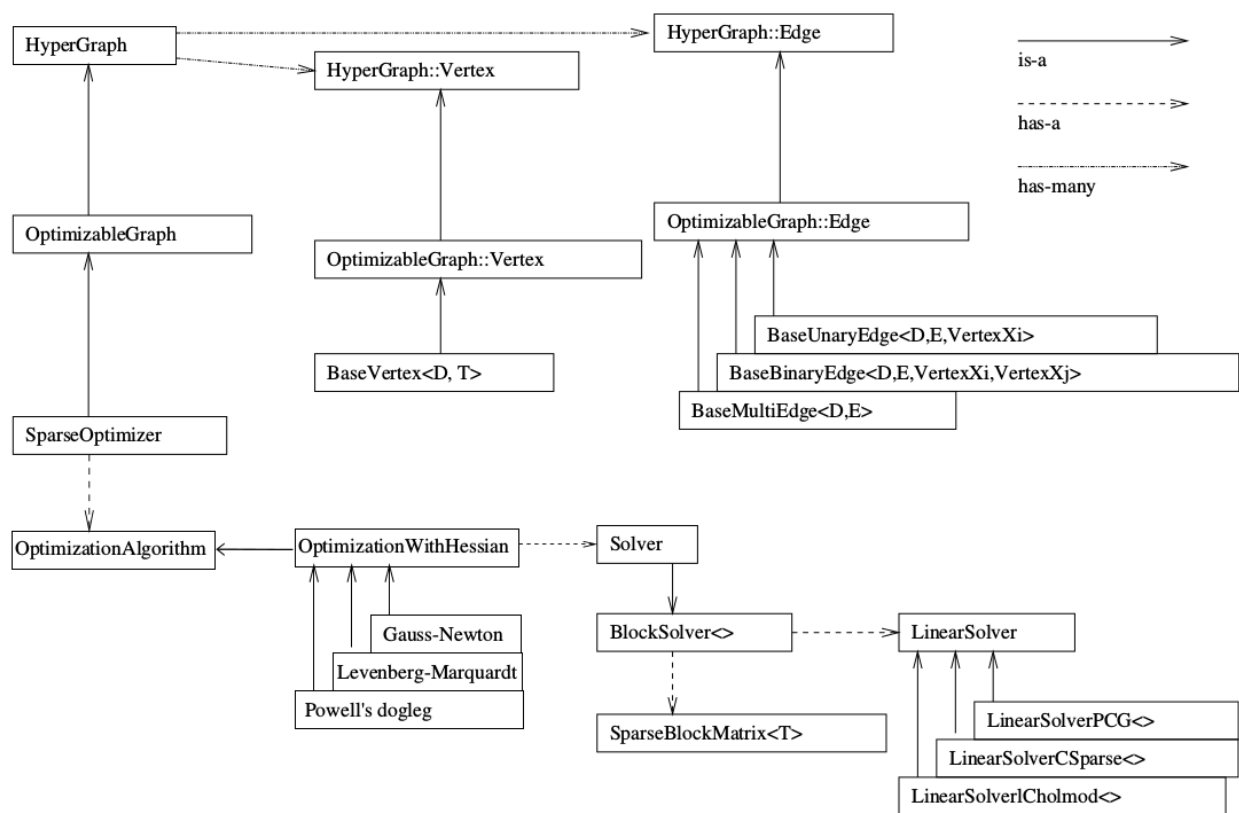
师兄：别急，第一次接触g2o，确实有这种感觉，而且官网文档写的也比较“不通俗不易懂”，不过如果你能捋顺了它的框架，再去看代码，应该很快能够入手了

小白：是的，先对框架了然于胸才行，不然即使能凑合看懂别人代码，自己也不会写啊！

师兄：嗯嗯，其实g2o帮助我们实现了很多内部的算法，只是在进行构造的时候，需要遵循一些规则，在我看来这是可以接受的，毕竟一个程序不可能满足所有的要求，因此在以后g2o的使用中还是应该多看多记，这样才能更好的使用这个库。

小白：记住了。养成记笔记的好习惯，还要多练习。

师兄：好，那我们首先看一下下面这个图，是g2o的基本框架结构。如果你查资料的话，你会在很多地方都能看到。看图的时候要注意箭头类型



## 1、图的核心

小白：师兄，这个图该从哪里开始看？感觉好多东西。。

师兄：如果你想要知道这个图中哪个最重要，就去看看箭头源头在哪里

小白：我看看。。。好像是最左侧的SparseOptimizer？

师兄：对的，SparseOptimizer是整个图的核心，我们注意右上角的 is-a 实心箭头，这个SparseOptimizer它是一个Optimizable Graph，从而也是一个超图（HyperGraph）。

小白：我去，师兄，怎么突然冒出来这么多奇怪的术语，都啥意思啊？

师兄：这个你不需要一个个弄懂，不然可能黄花菜都凉了。你先暂时只需要了解一下它们的名字，有些以后用不到，有些以后用到了再回看。目前如果遇到重要的我会具体解释。

小白：好。那下一步看哪里？

## 2、顶点和边

师兄：我们先来看上面的结构吧。注意看 has-many 箭头，你看这个超图包含了许多顶点（HyperGraph::Vertex）和边（HyperGraph::Edge）。而这些顶点继承自 Base Vertex，也就是OptimizableGraph::Vertex，而边可以继承自 BaseUnaryEdge（单边），BaseBinaryEdge（双边）或BaseMultiEdge（多边），它们都叫做OptimizableGraph::Edge

小白：头有点晕了，师兄

师兄：哈哈，不用一个个记，现阶段了解这些就行。顶点和边在编程中很重要的，关于顶点和边的定义我们以后会详细说的。下面我们来看底部的结构。

小白：嗯嗯，知道啦！

### 3、配置SparseOptimizer的优化算法和求解器

师兄：你看下面，整个图的核心SparseOptimizer 包含一个优化算法（ OptimizationAlgorithm ）的对象。OptimizationAlgorithm是通过OptimizationWithHessian 来实现的。其中迭代策略可以从Gauss-Newton（高斯牛顿法，简称GN），Levenberg-Marquardt（简称LM法），Powell's dogleg 三者中间选择一个（我们常用的是GN和LM）

小白：GN和LM就是我们以前讲过的非线性优化方法中常用的两种吧

师兄：是的，如果不了解的话具体看《XXX》《XXX》这两篇文章。

### 4、如何求解

师兄：那么如何求解呢？OptimizationWithHessian 内部包含一个求解器（ Solver ），这个Solver实际是由一个BlockSolver组成的。这个BlockSolver有两个部分，一个是SparseBlockMatrix，用于计算稀疏的雅可比和Hessian矩阵；一个是线性方程的求解器（ LinearSolver ），它用于计算迭代过程中最关键的一步 $H\Delta x = -b$ ，LinearSolver有几种方法可以选择：PCG, CSparse, Choldmod，具体定义后面会介绍

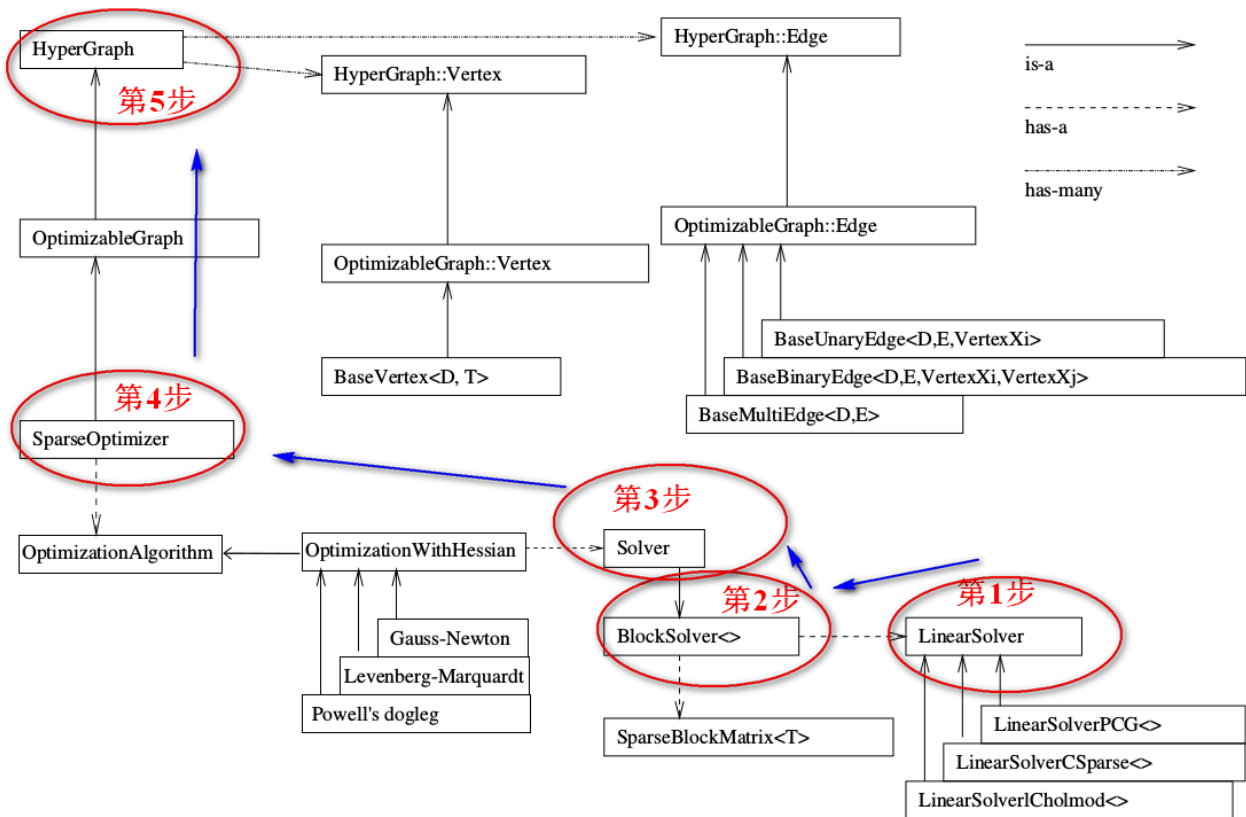
到此，就是上面图的一个简单理解。

## 一步步带你看懂g2o编程流程

---

小白：师兄，看完了我也不知道编程时具体怎么编呢！

师兄：我正好要说这个。首先这里需要说一下，我们梳理是从顶层到底层，但是编程实现时需要反过来，像建房子一样，从底层开始搭建框架一直到顶层。g2o的整个框架就是按照下图中我标的这个顺序来写的。



高博在十四讲中g2o求解曲线参数的例子来说明，源代码地址

[https://github.com/gaoxiang12/slambook/edit/master/ch6/g2o\\_curve\\_fitting/main.cpp](https://github.com/gaoxiang12/slambook/edit/master/ch6/g2o_curve_fitting/main.cpp)

为了方便理解，我重新加了注释。如下所示，

```
typedef g2o::BlockSolver< g2o::BlockSolverTraits<3,1> > Block; // 每个误差项优化变量维度为
3, 误差值维度为1

// 第1步：创建一个线性求解器LinearSolver
Block::LinearSolverType* linearSolver = new
g2o::LinearSolverDense<Block::PoseMatrixType>();

// 第2步：创建BlockSolver。并用上面定义的线性求解器初始化
Block* solver_ptr = new Block( linearSolver );

// 第3步：创建总求解器solver。并从GN, LM, DogLeg 中选一个，再用上述块求解器BlockSolver初始化
g2o::OptimizationAlgorithmLevenberg* solver = new g2o::OptimizationAlgorithmLevenberg(
solver_ptr );

// 第4步：创建终极大boss 稀疏优化器 ( SparseOptimizer )
g2o::SparseOptimizer optimizer; // 图模型
optimizer.setAlgorithm( solver ); // 设置求解器
optimizer.setVerbose( true ); // 打开调试输出

// 第5步：定义图的顶点和边。并添加到SparseOptimizer中
CurveFittingVertex* v = new CurveFittingVertex(); // 往图中增加顶点
v->setEstimate( Eigen::Vector3d(0,0,0) );
```

```

v->setId(0);
optimizer.addVertex( v );
for ( int i=0; i<N; i++ )    // 往图中增加边
{
    CurveFittingEdge* edge = new CurveFittingEdge( x_data[i] );
    edge->setId(i);
    edge->setVertex( 0, v );           // 设置连接的顶点
    edge->setMeasurement( y_data[i] ); // 观测数值
    edge->setInformation( Eigen::Matrix<double,1,1>::Identity()*1/(w_sigma*w_sigma) ); //
    信息矩阵：协方差矩阵之逆
    optimizer.addEdge( edge );
}

// 第6步：设置优化参数，开始执行优化
optimizer.initializeOptimization();
optimizer.optimize(100);

```

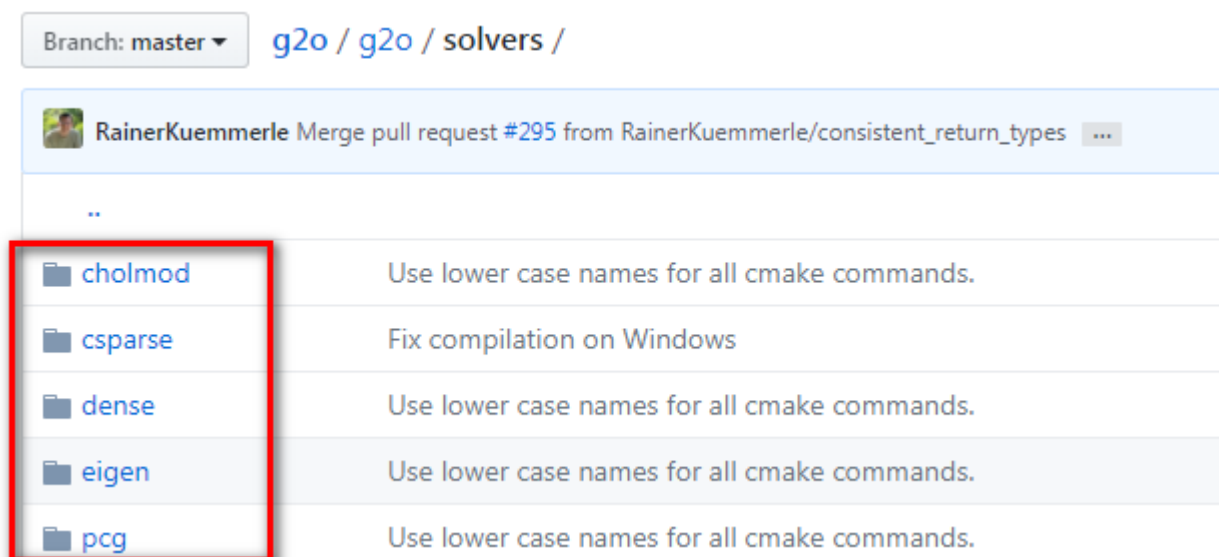
结合上面的流程图和代码。下面一步步解释具体步骤。

## 1、创建一个线性求解器LinearSolver

我们要求的增量方程的形式是： $H\Delta X=-b$ ，通常情况下想到的方法就是直接求逆，也就是 $\Delta X=-H.inv*b$ 。看起来好像很简单，但这有个前提，就是H的维度较小，此时只需要矩阵的求逆就能解决问题。但是当H的维度较大时，矩阵求逆变得很困难，求解问题也变得很复杂。

小白：那有什么办法吗？

师兄：办法肯定是有的。此时我们就需要一些特殊的方法对矩阵进行求逆，你看下图是GitHub上g2o相关部分的代码



如果你点进去看，可以分别查看每个方法的解释，如果不想挨个点进去看，看看下面我的总结就行了



LinearSolverCholmod : 使用sparse cholesky分解法。继承自LinearSolverCCS  
LinearSolverCsparse : 使用Csparse法。继承自LinearSolverCCS  
LinearSolverPCG : 使用preconditioned conjugate gradient 法, 继承自LinearSolver  
LinearSolverDense : 使用dense cholesky分解法。继承自LinearSolver  
LinearSolverEigen : 依赖项只有eigen, 使用eigen中sparse Cholesky 求解, 因此编译好后可以方便的在其他地方使用, 性能和Csparse差不多。继承自LinearSolver

## 2、创建BlockSolver。并用上面定义的线性求解器初始化。

BlockSolver 内部包含 LinearSolver, 用上面我们定义的线性求解器LinearSolver来初始化。它的定义在如下文件夹内:

[g2o/g2o/core/block\\_solver.h](#)

你点进去会发现 BlockSolver有两种定义方式

一种是指定的固定变量的solver, 我们来看一下定义

```
using BlockSolverPL = BlockSolver< BlockSolverTraits<p, l> >;
```

其中p代表pose的维度 ( 注意一定是流形manifold下的最小表示 ), l表示landmark的维度

另一种是可变尺寸的solver, 定义如下

```
using BlockSolverX = BlockSolverPL<Eigen::Dynamic, Eigen::Dynamic>;
```

小白: 为何会有可变尺寸的solver呢?

师兄: 这是因为在某些应用场景, 我们的Pose和Landmark在程序开始时并不能确定, 那么此时这个块状求解器就没办法固定变量, 此时使用这个可变尺寸的solver, 所有的参数都在中间过程中被确定












另外你看block\_solver.h的最后, 预定义了比较常用的几种类型, 如下所示:

```
BlockSolver_6_3 : 表示pose 是6维, 观测点是3维。用于3D SLAM中的BA  
BlockSolver_7_3 : 在BlockSolver_6_3 的基础上多了一个scale  
BlockSolver_3_2 : 表示pose 是3维, 观测点是2维
```

以后遇到了知道这些数字是什么意思就行了

## 3、创建总求解器solver。并从GN, LM, DogLeg 中选一个, 再用上述块求解器BlockSolver初始化

我们来看[g2o/g2o/core/](#)目录下, 发现Solver的优化方法有三种: 分别是高斯牛顿 ( GaussNewton ) 法, LM ( Levenberg-Marquardt ) 法、Dogleg法, 如下图所示, 也和前面的图相匹配

 <a href="#">optimization_algorithm_dogleg.cpp</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm_dogleg.h</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm_factory.cpp</a>	Improve listSolvers output
 <a href="#">optimization_algorithm_factory.h</a>	some documentation of tne factory
 <a href="#">optimization_algorithm_gauss_newt...</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm_gauss_newt...</a>	Simplified Memory Ownership
 <a href="#">optimization_algorithm levenberg.c...</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm levenberg.h</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm_property.h</a>	change license of most files to BSD
 <a href="#">optimization_algorithm_with_hessia...</a>	- added ability to change the floating point precision (float/double)...
 <a href="#">optimization_algorithm_with_hessia...</a>	- added ability to change the floating point precision (float/double)...

小白：师兄，上图最后那个OptimizationAlgorithmWithHessian 是干嘛的？

师兄：你点进去 GN、LM、Doglet算法内部，会发现他们都继承自同一个类：OptimizationWithHessian，如下图所示，这也和我们最前面那个图是相符的

```

namespace g2o {

    /**
     * \brief Implementation of the Gauss Newton Algorithm
     */
    class G2O_CORE_API OptimizationAlgorithmGaussNewton : public OptimizationAlgorithmWithHessian
    {
    public:
        /**
         * construct the Gauss Newton algorithm, which use the given Solver for solving the
         * linearized system.
         */

namespace g2o {

    /**
     * \brief Implementation of the Levenberg Algorithm
     */
    class G2O_CORE_API OptimizationAlgorithmLevenberg : public OptimizationAlgorithmWithHessian
    {
    public:
        /**
         * construct the Levenberg algorithm, which will use the given Solver for solving the
         * linearized system.
         */

namespace g2o {

    class BlockSolverBase;

    /**
     * \brief Implementation of Powell's Dogleg Algorithm
     */
    class G2O_CORE_API OptimizationAlgorithmDogleg : public OptimizationAlgorithmWithHessian
    {
    public:
        /** \brief type of the step to take */
        enum {

```

然后，我们点进去看 OptimizationAlgorithmWithHessian，发现它又继承自 OptimizationAlgorithm，这也和前面的相符

```

namespace g2o {

    class Solver;

    /**
     * \brief Base for solvers operating on the approximated Hessian, e.g., Gauss-Newton, Levenberg
     */
    class G2O_CORE_API OptimizationAlgorithmWithHessian : public OptimizationAlgorithm
    {
    public:
        explicit OptimizationAlgorithmWithHessian(Solver& solver);

```

总之，在该阶段，我们可以选则三种方法：

```
g2o::OptimizationAlgorithmGaussNewton
g2o::OptimizationAlgorithmLevenberg
g2o::OptimizationAlgorithmDogleg
```

#### 4、创建终极大boss 稀疏优化器（SparseOptimizer），并用已定义求解器作为求解方法。

创建稀疏优化器

```
g2o::SparseOptimizer    optimizer;
```

用前面定义好的求解器作为求解方法：

```
SparseOptimizer::setAlgorithm(OptimizationAlgorithm* algorithm)
```

其中setVerbose是设置优化过程输出信息用的

```
SparseOptimizer::setVerbose(bool verbose)
```

不信我们来看一下它的定义

```
if (verbose()) {
    computeActiveErrors();
    cerr << "iteration= -1\t chi2= " << activeChi2()
        << "\t time= 0.0"
        << "\t cumTime= 0.0"
        << "\t (using initial guess from " << costFunction.name() << ")" << endl;
}
```

#### 5、定义图的顶点和边。并添加到SparseOptimizer中。

这部分比较复杂，我们下一次再介绍。

#### 6、设置优化参数，开始执行优化。

设置SparseOptimizer的初始化、迭代次数、保存结果等。

初始化

```
SparseOptimizer::initializeOptimization(HyperGraph::EdgeSet& eset)
```

设置迭代次数，然后就开始执行图优化了。

```
SparseOptimizer::optimize(int iterations, bool online)
```

小白：终于搞明白g2o流程了！谢谢师兄！必须给你个「好看」啊！

注：以上内容部分参考了如下文章，感谢原作者：

<https://www.jianshu.com/p/e16ffb5b265d>

<https://blog.csdn.net/heyjia0327/article/details/47686523>

## 讨论

我们知道（不知道的话，去查一下十四讲）用g2o和ceres库都能用来进行BA优化，这两者在使用过程中有什么不同？

欢迎留言讨论，更多学习视频、文档资料、参考答案等关注计算机视觉life公众号，菜单栏点击“知识星球”查看「从零开始学习SLAM」星球介绍，快来和其他小伙伴一起学习交流~



# 从零开始学习SLAM

自制视频课程  
批改讲解作业  
交流互动答疑  
越早进越优惠



微信扫一扫

## 推荐阅读

[从零开始一起学习SLAM | 为什么要学SLAM？](#) [从零开始一起学习SLAM | 学习SLAM到底需要学什么？](#) [从零开始一起学习SLAM | SLAM有什么用？](#) [从零开始一起学习SLAM | C++新特性要不要学？](#) [从零开始一起学习SLAM | 为什么要用齐次坐标？](#) [从零开始一起学习SLAM | 三维空间刚体的旋转](#) [从零开始一起学习SLAM | 为啥需要李群与李代数？](#) [从零开始一起学习SLAM | 相机成像模型](#) [从零开始一起学习SLAM | 不推公式，如何真正理解对极约束？](#) [从零开始一起学习SLAM | 神奇的单应矩阵](#) [从零开始一起学习SLAM | 你好，点云](#) [从零开始一起学习SLAM | 给点云加个滤网](#) [从零开始一起学习SLAM | 点云平滑法线估计](#) [零基础小白，如何入门计算机视觉？](#) [SLAM领域牛人、牛实验室、牛研究成果梳理](#) [我用MATLAB撸了一个2D LiDAR SLAM](#) [可视化理解四元数，愿你不再掉头发](#) [最近一年语义SLAM有哪些代表性工作？](#) [视觉SLAM技术综述](#)

小白：师兄，上一次看的g2o框架《[从零开始一起学习SLAM | 理解图优化，一步步带你看懂g2o代码](#)》真的很清晰，我现在再去看g2o的那些优化的部分，基本都能看懂了呢！

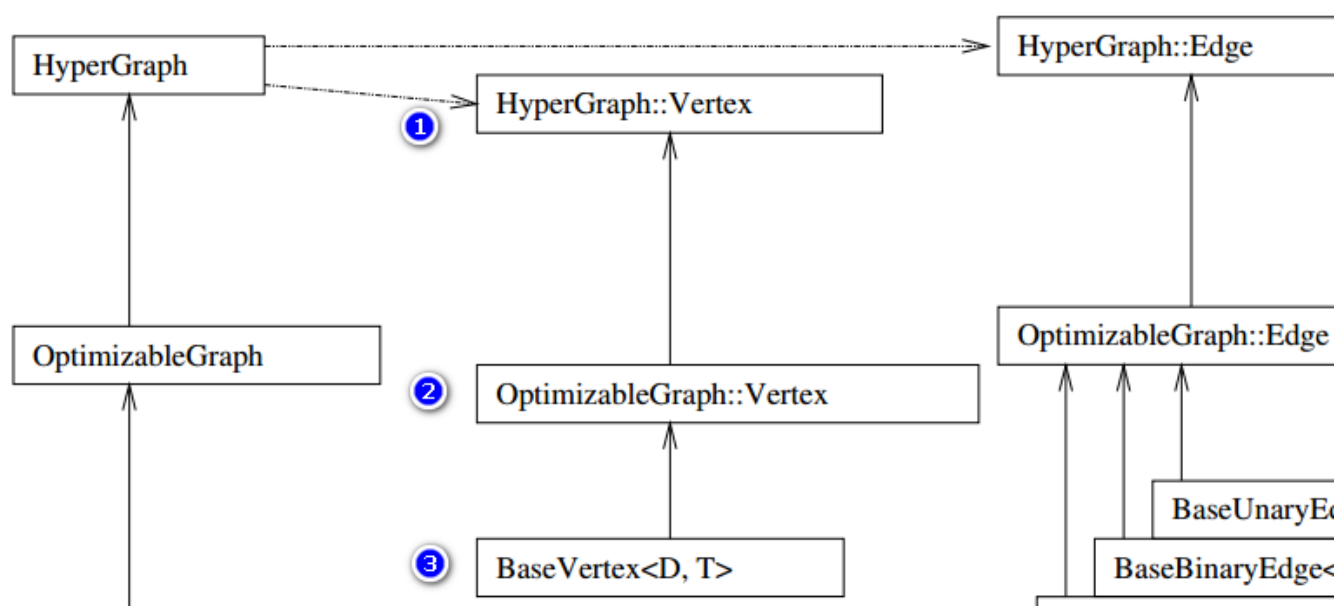
师兄：那太好了，以后多练习练习，加深理解

小白：嗯，我开始编程时，发现g2o的顶点和边的定义也非常复杂，光看十四讲里面，就有好几种不同的定义，完全懵圈状态。。。师兄，能否帮我捋捋思路啊

师兄：嗯，你说的没错，入门的时候确实感觉很乱，我最初也是花了些时间才搞懂的，下面分享一下。

## g2o的顶点 ( Vertex) 从哪里来的？

师兄：在《g2o: A general Framework for (Hyper) Graph Optimization》这篇文档里，我们找到那张经典的类结构图。也就是上次讲框架用到的那张结构图。其中涉及到顶点（vertex）的就是下面加了序号的3个东东了。



小白：记得呢，这个图很关键，帮助我理清了很多思路，原来来自这篇文章啊

师兄：对，下面我们一步步来看吧。先来看看上图中和vertex有关的第①个类：HyperGraph::Vertex，在g2o的GitHub上（<https://github.com/RainerKuemmerle/g2o>），它在这个路径

g2o/core/hyper\_graph.h

这个 HyperGraph::Vertex 是个abstract vertex，必须通过派生来使用。如下图所示

```

53     class G2O_CORE_API HyperGraph
54     {
55     public:
56
57         /**
58
59         ..
60
138
139         //: abstract Vertex, your types must derive from that one
140         class G2O_CORE_API Vertex : public HyperGraphElement {
141         public:
142             //! creates a vertex having an ID specified by the argument
143             explicit Vertex(int id=InvalidId);
144             virtual ~Vertex();

```

然后我们看g2o 类结构图中第②个类，我们看到HyperGraph::Vertex 是通过类OptimizableGraph 来继承的，而OptimizableGraph的定义在

g2o/core/optimizable\_graph.h

我们找到vertex定义，发现果然，OptimizableGraph 继承自 HyperGraph，如下图所示

```

61     struct G2O_CORE_API OptimizableGraph : public HyperGraph {
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100    /**
101     * \brief A general case Vertex for optimization
102     */
103    class G2O_CORE_API Vertex : public HyperGraph::Vertex, public HyperGraph::DataContainer {
104    private:
105        friend struct OptimizableGraph;
106    public:
107        Vertex();
108    ---

```

不过，这个OptimizableGraph::Vertex 也非常底层，具体使用时一般都会进行扩展，因此g2o中提供了一个比较通用的适合大部分情况的模板。就是g2o 类结构图中 对应的第③个类：

BaseVertex<D, T>

那么它在哪里呢？在这个路径：

g2o/core/base\_vertex.h

```

40 namespace g2o {
41
42 /**
43  * \brief Templated BaseVertex
44  *
45  * Templated BaseVertex
46  * D : minimal dimension of the vertex, e.g., 3 for rotation in 3D
47  * T : internal type to represent the estimate, e.g., Quaternion for rotation in 3D
48  */
49 template <int D, typename T>
50 class BaseVertex : public OptimizableGraph::Vertex {
51 public:
52     typedef T EstimateType;
53     typedef std::stack<EstimateType,
54                     std::vector<EstimateType, Eigen::aligned_allocator<EstimateType> > >
55         BackupStackType;
56
57     static const int Dimension = D;          ///< dimension of the estimate (minimal) in the manifold space
58
59     typedef Eigen::Map<Eigen::Matrix<number_t, D, D, Eigen::ColMajor>, Eigen::Matrix<number_t, D, D, Eigen::ColMajor>::Flags >
60

```

小白：哇塞，原来是这样抽丝剥茧的呀，学习了，授人以鱼不如授人以渔啊！

师兄：嗯，其实就是根据那张图结合g2o GitHub代码就行了

## g2o的顶点 ( Vertex) 参数如何理解？

小白：那是不是就可以开始用了？

师兄：别急，我们来看看参数吧，这个很关键。

我们来看一下模板参数 D 和 T，翻译一下上图红框：

D是int 类型的，表示vertex的最小维度，比如3D空间中旋转是3维的，那么这里 D = 3

T是待估计vertex的数据类型，比如用四元数表达三维旋转的话，T就是Quaternion 类型

小白：哦哦，大概理解了，但还是有点模糊

师兄：我们进一步来细看一下D, T。这里的D 在源码里面是这样注释的

```

static const int Dimension = D; ///< dimension of the estimate (minimal) in the manifold
space

```

可以看到这个D并非是顶点（更确切的说是状态变量）的维度，而是其在流形空间（manifold）的最小表示，这里一定要区别开，另外，源码里面也给出了T的作用



```
typedef T EstimateType;  
EstimateType _estimate;
```

可以看到，这里T就是顶点（状态变量）的类型，跟前面一样。

小白：Got it!

## 如何自己定义顶点？

小白：师兄，我们是不是可以开始写顶点定义了？

师兄：嗯，我们知道了顶点的基本类型是 `BaseVertex<D, T>`，那么下一步关心的就是如何使用了，因为在不同的应用场景（二维空间，三维空间），有不同的待优化变量（位姿，空间点），还涉及不同的优化类型（李代数位姿、李群位姿）

小白：这么多啊，那要自己根据 `BaseVertex` 一个个实现吗？

师兄：那不需要！g2o本身内部定义了一些常用的顶点类型，我给找出来了，大概这些：

```
VertexSE2 : public BaseVertex<3, SE2> //2D pose vertex, (x,y,theta)  
VertexSE3 : public BaseVertex<6, Isometry3> //6d vector (x,y,z,qx,qy,qz) (note that we  
leave out the w part of the quaternion)  
VertexPointXY : public BaseVertex<2, Vector2>  
VertexPointXYZ : public BaseVertex<3, Vector3>  
VertexSBAPointXYZ : public BaseVertex<3, Vector3>  
  
// SE3 vertex parameterized internally with a transformation matrix and externally with  
its exponential map  
VertexSE3Expmap : public BaseVertex<6, SE3Quat>  
  
// SBACam vertex, (x,y,z,qw,qx,qy,qz),(x,y,z,qx,qy,qz) (note that we leave out the w part  
of the quaternion.  
// qw is assumed to be positive, otherwise there is an ambiguity in qx,qy,qz as a  
rotation  
VertexCam : public BaseVertex<6, SBACam>  
  
// Sim3 vertex, (x,y,z,qw,qx,qy,qz),7d vector,(x,y,z,qx,qy,qz) (note that we leave out  
the w part of the quaternion.  
VertexSim3Expmap : public BaseVertex<7, Sim3>
```

小白：好全啊，我们可以直接用啦！

师兄：当然我们可以直接用这些，但是有时候我们需要的顶点类型这里面没有，就得自己定义了。

重新定义顶点一般需要考虑重写如下函数：

```
virtual bool read(std::istream& is);
virtual bool write(std::ostream& os) const;
virtual void oplusImpl(const number_t* update);
virtual void setToOriginImpl();
```

小白：这些函数啥意思啊，我也就能看懂 read 和 write（/尴尬脸），还有每次定义都要重新写这几个函数吗？

师兄：是的，这几个是主要要改的地方。我们来看一下他们都是什么意义：

read，write：分别是读盘、存盘函数，一般情况下不需要进行读/写操作的话，仅仅声明一下就可以

setToOriginImpl：顶点重置函数，设定被优化变量的原始值。

oplusImpl：顶点更新函数。非常重要的一个函数，主要用于优化过程中增量 $\Delta x$ 的计算。我们根据增量方程计算出增量之后，就是通过这个函数对估计值进行调整的，因此这个函数的内容一定要重视。

自己定义 顶点一般是下面的格式：

```
class myVertex: public g2::BaseVertex<Dim, Type>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    myVertex(){}

    virtual void read(std::istream& is) {}
    virtual void write(std::ostream& os) const {}

    virtual void setOriginImpl()
    {
        _estimate = Type();
    }
    virtual void oplusImpl(const double* update) override
    {
        _estimate += /*update*/;
    }
}
```

小白：看不太懂啊，师兄

师兄：没事，我们看例子就知道了，先看一个简单例子，来自十四讲中的曲线拟合，来源如下

[ch6/g2o\\_curve\\_fitting/main.cpp](#)

// 曲线模型的顶点，模板参数：优化变量维度和数据类型

```

class CurveFittingVertex: public g2o::BaseVertex<3, Eigen::Vector3d>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    virtual void setToOriginImpl() // 重置
    {
        _estimate << 0,0,0;
    }

    virtual void oplusImpl( const double* update ) // 更新
    {
        _estimate += Eigen::Vector3d(update);
    }
    // 存盘和读盘: 留空
    virtual bool read( istream& in ) {}
    virtual bool write( ostream& out ) const {}
};

```

我们可以看到下面代码中顶点初值设置为0，更新时也是直接把更新量 update 加上去的，知道为什么吗？

小白：更新不就是  $x + \Delta x$  吗，这是定义吧

师兄：嗯，对于这个例子是可以直接加，因为顶点类型是Eigen::Vector3d，属于向量，是可以通过加法来更新的。但是但是有些例子就不行，比如下面这个复杂点例子：李代数表示位姿VertexSE3Expmap

来自g2o官网，在这里

[g2o/types/sba/types\\_six\\_dof\\_expmap.h](https://g2o.cc/docs/en/quickstart/types/sba/types_six_dof_expmap.h)

```

/**

 \* \brief SE3 vertex parameterized internally with a transformation matrix
 and externally with its exponential map

 */

class G2O_TYPES_SBA_API VertexSE3Expmap : public BaseVertex<6, SE3Quat>{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    VertexSE3Expmap();
    bool read(std::istream& is);
    bool write(std::ostream& os) const;
    virtual void setToOriginImpl() {
        _estimate = SE3Quat();
    }

    virtual void oplusImpl(const number_t* update_) {
        Eigen::Map<const Vector6> update(update_);
        setEstimate(SE3Quat::exp(update)*estimate()); //更新方式
    }
}

```

```
};
```

小白：师兄，这个里面的6, SE3Quat 分别是什么意思？

师兄：书中都写了，以下来自十四讲的介绍：

第一个参数6 表示内部存储的优化变量维度，这是个6维的李代数

第二个参数是优化变量的类型，这里使用了g2o定义的相机位姿类型：SE3Quat。

在这里可以具体查看g2o/types/slam3d/se3quat.h

它内部使用了四元数表达旋转，然后加上位移来存储位姿，同时支持李代数上的运算，比如对数映射（log函数）、李代数上增量（update函数）等操作

说完了，那我现在问你个问题，为啥这里更新时没有像上面那样直接加上去？

小白：这个表示位姿，好像是不能直接加的我记得，原因有点忘了

师兄：嗯，是不能直接加，原因是变换矩阵不满足加法封闭。那我再问你，为什么相机位姿顶点类VertexSE3Expmap使用了李代数表示相机位姿，而不是使用旋转矩阵和平移矩阵？

小白：不造啊。。

师兄：其实也是上述原因的拓展：这是因为旋转矩阵是有约束的矩阵，它必须是正交矩阵且行列式为1。使用它作为优化变量就会引入额外的约束条件，从而增大优化的复杂度。而将旋转矩阵通过李群-李代数之间的转换关系转换为李代数表示，就可以把位姿估计变成无约束的优化问题，求解难度降低。

小白：原来如此啊，以前学的东西都忘了。。

师兄：以前学的要多看，温故而知新。我们继续看例子，刚才是位姿的例子，下面是三维点的例子，空间点位置VertexPointXYZ，维度为3，类型是Eigen的Vector3，比较简单，就不解释了

```
class G2O_TYPES_SBA_API VertexSBAPointXYZ : public BaseVertex<3, Vector3>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    VertexSBAPointXYZ();
    virtual bool read(std::istream& is);
    virtual bool write(std::ostream& os) const;
    virtual void setToOriginImpl() {
        _estimate.fill(0);
    }

    virtual void oplusImpl(const number_t* update)
    {
        Eigen::Map<const Vector3> v(update);
        _estimate += v;
    }
};
```

## 如何向图中添加顶点？

师兄：往图中增加顶点比较简单，我们还是先看看第一个曲线拟合的例子，setEstimate(type) 函数来设定初始值；setId(int) 定义节点编号

```
// 往图中增加顶点
CurveFittingVertex* v = new CurveFittingVertex();
v->setEstimate( Eigen::Vector3d(0,0,0) );
v->setId(0);
optimizer.addVertex( v );
```

这个是添加 VertexSBAPointXYZ 的例子，都很容易看懂

/ch7/pose\_estimation\_3d2d.cpp

```
int index = 1;
for ( const Point3f p:points_3d )    // landmarks
{
    g2o::VertexSBAPointXYZ* point = new g2o::VertexSBAPointXYZ();
    point->setId ( index++ );
    point->setEstimate ( Eigen::Vector3d ( p.x, p.y, p.z ) );
    point->setMarginalized ( true );
    optimizer.addVertex ( point );
}
```

至此，我们讲完了g2o 的顶点的来源，定义，自定义方法，添加方法，基本上你以后再看到顶点就不会陌生啦！

小白：太感谢啦！

## 编程练习

- 题目：给定一组世界坐标系下的3D点(p3d.txt)以及它在相机中对应的坐标(p2d.txt)，以及相机的内参矩阵。使用bundle adjustment 方法（g2o库实现）来估计相机的位姿T。初始位姿T为单位矩阵。
- 本程序学习目标：熟悉g2o库编写流程，熟悉顶点定义方法。

代码框架、数据及预期结果已经为你准备好了，公众号「计算机视觉life」后台回复：**顶点**，即可获得。

欢迎留言讨论，更多学习视频、文档资料、参考答案等关注计算机视觉life公众号，**菜单栏点击“知识星球”查看「从零开始学习SLAM」星球介绍**，快来和其他小伙伴一起学习交流~

自制视频课程  
批改讲解作业  
交流互动答疑  
越早进越优惠



微信扫一扫

本文参考：

高翔《视觉SLAM十四讲》

<https://www.jianshu.com/p/e16ffb5b265d>

## 推荐阅读

[从零开始一起学习SLAM | 为什么要学SLAM?](#) [从零开始一起学习SLAM | 学习SLAM到底需要学什么?](#) [从零开始一起学习SLAM | SLAM有什么用?](#) [从零开始一起学习SLAM | C++新特性要不要学?](#) [从零开始一起学习SLAM | 为什么要用齐次坐标?](#) [从零开始一起学习SLAM | 三维空间刚体的旋转](#) [从零开始一起学习SLAM | 为啥需要李群与李代数?](#) [从零开始一起学习SLAM | 相机成像模型](#) [从零开始一起学习SLAM | 不推公式，如何真正理解对极约束?](#) [从零开始一起学习SLAM | 神奇的单应矩阵](#) [从零开始一起学习SLAM | 你好，点云](#) [从零开始一起学习SLAM | 给点云加个滤网](#) [从零开始一起学习SLAM | 点云平滑法线估计](#) [从零开始一起学习SLAM | 点云到网格的进化](#) [从零开始一起学习SLAM | 理解图优化，一步步带你看懂g2o代码](#) [零基础小白，如何入门计算机视觉?](#) [SLAM领域牛人、牛实验室、牛研究成果梳理](#) [我用MATLAB撸了一个2D LiDAR SLAM](#) [可视化理解四元数，愿你不再掉头发](#) [最近一年语义SLAM有哪些代表性工作?](#) [视觉SLAM技术综述 汇总](#) | [VIO、激光SLAM相关论文分类集锦](#)

本文讲解g2o边的定义使用方法

小白：师兄，g2o框架《[从零开始一起学习SLAM | 理解图优化，一步步带你看懂g2o代码](#)》，以及顶点《[从零开始一起学习SLAM | 掌握g2o顶点编程套路](#)》我都学完啦，今天给我讲讲g2o中的边吧！是不是也有什么套路？

师兄：嗯，g2o的边比顶点稍微复杂一些，不过前面你也了解了许多g2o的东西，有没有发现g2o的编程基本都是固定的格式（套路）呢？

小白：是的，我现在按照师兄说的g2o框架和顶点设计方法，再去看g2o实现不同功能的代码，发现都是一个模子出来的，只不过在某些地方稍微改改就行了啊

师兄：是这样的。我们来看看g2o的边到底是咋回事。

## 初步认识g2o的边

师兄：在《g2o: A general Framework for (Hyper) Graph Optimization》这篇文档里，我们找到那张经典的类结构图，里面关于边（edge）的部分是这样的，重点是下图中红色框内。

上一次我们讲顶点的时候，还专门去追根溯源查找顶点类之间的继承关系，边其实也是类似的，我们在g2o官方GitHub上这些 `g2o/g2o/core/hyper_graph.h` `g2o/g2o/core/optimizable_graph.h` `g2o/g2o/core/base_edge.h`

头文件下就能看到这些继承关系了，我们就不像之前顶点那样一个个去追根溯源了，如果有兴趣你可以自己去试试看。我们主要关注一下上面红框内的三种边。

`BaseUnaryEdge`，`BaseBinaryEdge`，`BaseMultiEdge` 分别表示一元边，两元边，多元边。

小白：他们有啥区别啊？师兄：一元边你可以理解为一条边只连接一个顶点，两元边理解为一条边连接两个顶点，也就是我们常见的边啦，多元边理解为一条边可以连接多个（3个以上）顶点



一个比较丑的示例

下面我们来看看他们的参数有什么区别？你看主要就是几个参数：`D`, `E`, `VertexXi`, `VertexXj`，他们的分别代表：

`D` 是 `int` 型，表示测量值的维度（dimension）`E` 表示测量值的数据类型 `VertexXi`，`VertexXj` 分别表示不同顶点的类型

比如我们用边表示三维点投影到图像平面的重投影误差，就可以设置输入参数如下：

```
BaseBinaryEdge<2, Vector2D, VertexSBAPointXYZ, VertexSE3Expmap>
```

你说说看 这个定义是什么意思？小白：首先这个是个二元边。第1个2是说测量值是2维的，也就是图像像素坐标x,y的差值，对应测量值的类型是Vector2D，两个顶点也就是优化变量分别是三维点 VertexSBAPointXYZ，和李群位姿 VertexSE3Expmap？

师兄：对的，就是这样~当然除了输入参数外，定义边我们通常需要复写一些重要的成员函数 小白：听着和顶点类似哦，也是复写成员函数，顶点里主要复写了顶点更新函数oplusImpl和顶点重置函数setToOriginImpl，边的话是不是也差不多？师兄：边和顶点的成员函数还是差别比较大的，边主要有以下几个重要的成员函数

```
virtual bool read(std::istream& is);
virtual bool write(std::ostream& os) const;
virtual void computeError();
virtual void linearizeOplus();
```

下面简单解释一下 read, write：分别是读盘、存盘函数，一般情况下不需要进行读/写操作的话，仅仅声明一下就可以 computeError函数：非常重要，是使用当前顶点的值计算的测量值与真实的测量值之间的误差 linearizeOplus函数：非常重要，是在当前顶点的值下，该误差对优化变量的偏导数，也就是我们说的Jacobian

除了上面几个成员函数，还有几个重要的成员变量和函数也一并解释一下：

```
_measurement：存储观测值
_error：存储computeError() 函数计算的误差
_vertices[]：存储顶点信息，比如二元边的话，_vertices[] 的大小为2，存储顺序和调用setVertex(int,
vertex) 是设定的int 有关（0 或1）
setId(int)：来定义边的编号（决定了在H矩阵中的位置）
setMeasurement(type) 函数来定义观测值
setVertex(int, vertex) 来定义顶点
setInformation() 来定义协方差矩阵的逆
```

后面我们写代码的时候经常会遇到他们的。

## 如何自定义g2o的边？

小白：前面你介绍了g2o中边的基本类型、重要的成员变量和成员函数，那么如果我们要定义边的话，具体如何编程呢？师兄：我这里正好有个模板给你看看，基本上定义g2o中的边，就是如下套路：

```
class myEdge: public g2o::BaseBinaryEdge<errorDim, errorType, Vertex1Type, Vertex2Type>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    myEdge(){}
    virtual bool read(istream& in) {}
    virtual bool write(ostream& out) const {}
    virtual void computeError() override
    {
        // ...
        _error = _measurement - Something;
    }
    virtual void linearizeOplus() override
```



```

{
    _jacobianOplusXi(pos, pos) = something;
    // ...
    /*
    _jacobianOplusXj(pos, pos) = something;
    ...
    */
}
private:
// data
}

```

我们可以发现，最重要的就是computeError(), linearizeOplus()两个函数了

小白：嗯，看起来好像也不难啊 师兄：我们先来看一个简单例子，地址在 [https://github.com/gaoxiang12/slambook/blob/master/ch6/g2o\\_curve\\_fitting/main.cpp](https://github.com/gaoxiang12/slambook/blob/master/ch6/g2o_curve_fitting/main.cpp) 这个是个一元边，主要是定义误差函数了，如下所示，你可以发现这个例子基本就是上面例子的一丢丢扩展，是不是感觉so easy？

```

// 误差模型 模板参数：观测值维度，类型，连接顶点类型
class CurveFittingEdge: public g2o::BaseUnaryEdge<1,double,CurveFittingVertex>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    CurveFittingEdge( double x ): BaseUnaryEdge(), _x(x) {}
    // 计算曲线模型误差
    void computeError()
    {
        const CurveFittingVertex* v = static_cast<const CurveFittingVertex*>
(_vertices[0]);
        const Eigen::Vector3d abc = v->estimate();
        _error(0,0) = _measurement - std::exp( abc(0,0)*_x*_x + abc(1,0)*_x + abc(2,0) );
    }
    virtual bool read( istream& in ) {}
    virtual bool write( ostream& out ) const {}
public:
    double _x; // x 值, y 值为 _measurement
};

```

小白：嗯，这个能看懂 师兄：下面是一个复杂一点例子，3D-2D点的PnP 问题，也就是最小化重投影误差问题，这个问题非常常见，使用最常见的二元边，弄懂了这个基本跟边相关的代码也差不多都一通百通了

代码在g2o的GitHub上这个地方可以看到 [g2o/types/sba/types\\_six\\_dof\\_expmap.h](#) 这里根据自己理解对代码加了注释，方便理解

```

//继承了BaseBinaryEdge类，观测值是2维，类型Vector2D,顶点分别是三维点、李群位姿
class G2O_TYPES_SBA_API EdgeProjectXYZ2UV : public BaseBinaryEdge<2, Vector2D,
VertexSBAPointXYZ, VertexSE3Expmap>{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    //1. 默认初始化
    EdgeProjectXYZ2UV();

```

```

//2. 计算误差
void computeError() {
    //李群相机位姿v1
    const VertexSE3Expmap* v1 = static_cast<const VertexSE3Expmap*>(_vertices[1]);
    // 顶点v2
    const VertexSBAPointXYZ* v2 = static_cast<const VertexSBAPointXYZ*>(_vertices[0]);
    //相机参数
    const CameraParameters * cam
        = static_cast<const CameraParameters *>(parameter(0));
    //误差计算, 测量值减去估计值, 也就是重投影误差obs-cam
    //估计值计算方法是T*p, 得到相机坐标系下坐标, 然后在利用camera2pixel()函数得到像素坐标。
    Vector2D obs(_measurement);
    _error = obs - cam->cam_map(v1->estimate().map(v2->estimate()));
}
//3. 线性增量函数, 也就是雅克比矩阵J的计算方法
virtual void linearizeOplus();
//4. 相机参数
CameraParameters * _cam;
bool read(std::istream& is);
bool write(std::ostream& os) const;
};

```

有一个地方比较难理解

```
_error = obs - cam->cam_map(v1->estimate().map(v2->estimate()));
```

小白：我确实看不懂这一句。。 师兄：其实就是：误差 = 观测 - 投影

下面我给你捋捋思路。我们先来看看cam\_map 函数，它的定义在 g2o/types/sba/types\_six\_dof\_expmap.cpp  
cam\_map 函数功能是把相机坐标系下三维点（输入）用内参转换为图像坐标（输出），具体代码如下所示

```

Vector2 CameraParameters::cam_map(const Vector3 & trans_xyz) const {
    Vector2 proj = project2d(trans_xyz);
    Vector2 res;
    res[0] = proj[0]*focal_length + principle_point[0];
    res[1] = proj[1]*focal_length + principle_point[1];
    return res;
}

```

然后看 .map函数，它的功能是把世界坐标系下三维点变换到相机坐标系，函数在 g2o/types/sim3/sim3.h 具体定义是

```

Vector3 map (const Vector3& xyz) const {
    return s*(r*xyz) + t;
}

```

因此下面这个代码

```
v1->estimate().map(v2->estimate())
```

就是用V1估计的pose把V2代表的三维点，变换到相机坐标系下。

小白：原来如此，以前我都忽视了这些东西了，没想到里面是这样的关联的。师兄：嗯，我们继续，前面主要是对 `computeError()` 的理解，还有一个很重要的函数就是 `linearizeOplus()`，用来定义雅克比矩阵 我摘取了相关代码（来自：`g2o/g2o/types/sba/types_six_dof_expmap.cpp`），并进行了标注，相信会更容易理解

十四讲第169页中的雅克比矩阵完全是按照书上 式子（7.45）、（7.47）来编程的，不难理解 小白：后面就是直接照抄书上就行，哈哈

## 如何向图中添加边？

师兄：前面我们讲过如何往图中增加顶点，可以说非常easy了，往图中增加边会稍微多一些内容，我们还是先从最简单的例子说起：一元边的添加方法

下面代码来自GitHub上，仍然是前面曲线拟合的例子 `slambook/ch6/g2o_curve_fitting/main.cpp`

```
// 往图中增加边
for ( int i=0; i<N; i++ )
{
    CurveFittingEdge* edge = new CurveFittingEdge( x_data[i] );
    edge->setId(i);
    edge->setVertex( 0, v );           // 设置连接的顶点
    edge->setMeasurement( y_data[i] ); // 观测数值
    edge->setInformation( Eigen::Matrix<double,1,1>::Identity()*1/(w_sigma*w_sigma) ); // 信息矩阵：协方差矩阵之逆
    optimizer.addEdge( edge );
}
```

小白：setMeasurement 函数的输入的观测值具体是指什么？师兄：对于这个曲线拟合，观测值就是实际观测到的数据点。对于视觉SLAM来说，通常就是我们观测到的特征点坐标，下面就是一个例子。这个例子比刚才的复杂一点，因为它是二元边，需要用边连接两个顶点 代码来自GitHub上 `slambook/ch7/pose_estimation_3d2d.cpp`

```
index = 1;
for ( const Point2f p:points_2d )
{
    g2o::EdgeProjectXYZ2UV* edge = new g2o::EdgeProjectXYZ2UV();
    edge->setId ( index );
    edge->setVertex ( 0, dynamic_cast<g2o::VertexSBAPointXYZ*> ( optimizer.vertex ( index ) ) );
    edge->setVertex ( 1, pose );
    edge->setMeasurement ( Eigen::Vector2d ( p.x, p.y ) );
    edge->setParameterId ( 0,0 );
    edge->setInformation ( Eigen::Matrix2d::Identity() );
    optimizer.addEdge ( edge );
    index++;
}
```

小白：这里的setMeasurement函数里的p来自向量points\_2d，也就是特征点的图像坐标(x,y)了吧！师兄：对，这正好呼应我刚才说的。另外，你看setVertex 有两个一个是 0 和 VertexSBAPointXYZ 类型的顶点，一个是1 和 pose。你觉得这里的0和1是什么意思？能否互换呢？

小白：0, 1应该是分别指代哪个顶点吧，直觉告诉我不能互换，可能得去查查顶点定义部分的代码 师兄：你的直觉没错！我帮你 查过啦，你看这个是setVertex在g2o官网的定义：

```
// set the ith vertex on the hyper-edge to the pointer supplied
void setVertex(size_t i, Vertex* v) { assert(i < _vertices.size() && "index out of bounds"); _vertices[i]=v;}
```

这段代码在 g2o/core/hyper\_graph.h 里可以找到。你看 \_vertices[i] 里的i就是我们这里的0和1，我们再去看看这里的类型：g2o::EdgeProjectXYZ2UV 的定义，前面我们也放出来了，就这两句

```
class G2O_TYPES_SBA_API EdgeProjectXYZ2UV
{
    ....
    //李群相机位姿v1
    const VertexSE3Expmap* v1 = static_cast<const VertexSE3Expmap*>(_vertices[1]);
    // 顶点v2
    const VertexSBAPointXYZ* v2 = static_cast<const VertexSBAPointXYZ*>(_vertices[0]);
}
```

你看 vertices[0] 对应的是 VertexSBAPointXYZ 类型的顶点，也就是三维点，vertices[1] 对应的是 VertexSE3Expmap 类型的顶点，也就是位姿pose。因此前面 1 对应的就应该是 pose，0对应的 应该就是三维点。

小白：原来如此，之前都没注意这些，看来g2o不会帮我区分顶点的类型啊，以后这里编程要对应好，不然错了都找不到原因呢！谢谢师兄，今天又是收获满满的一天！

## 练习

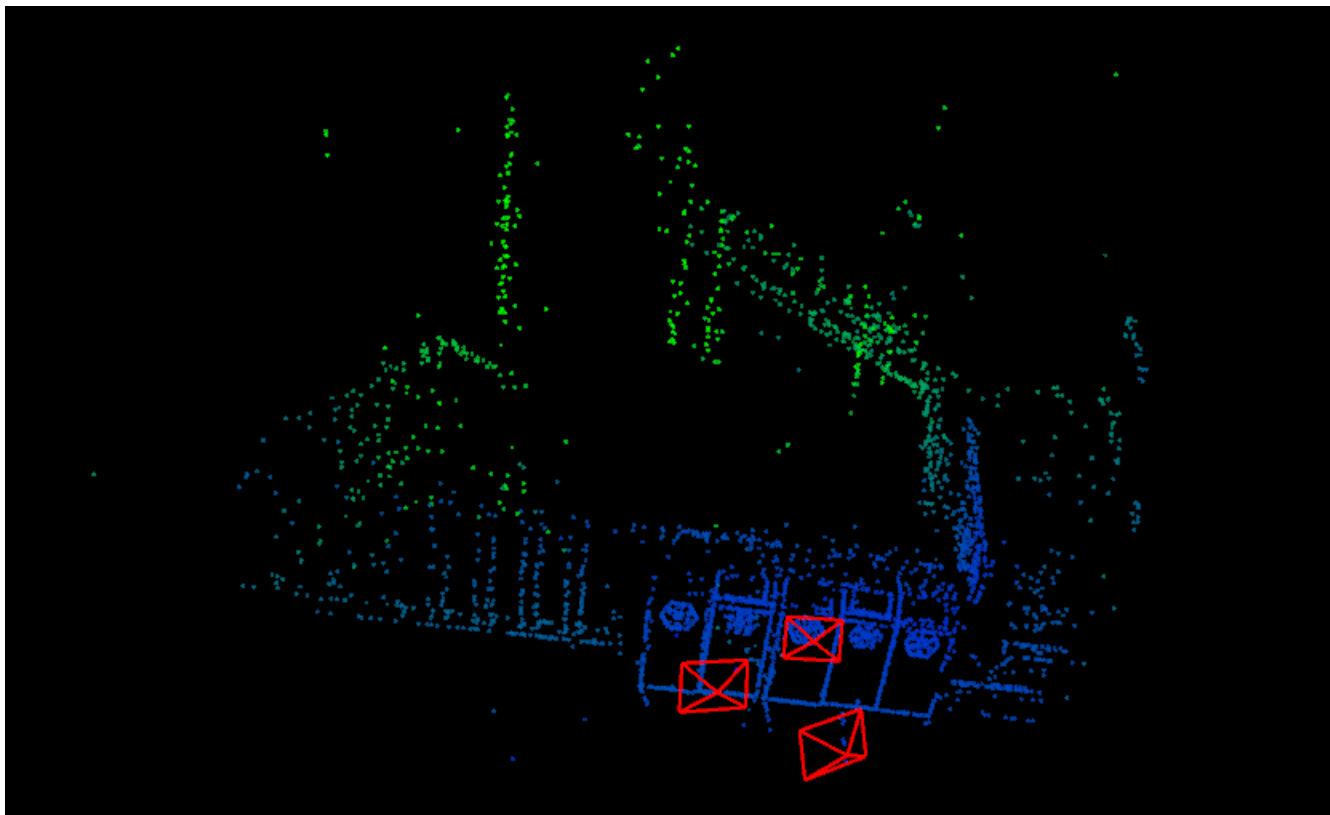
题目：用直接法Bundle Adjustment 估计相机位姿。给定3张图片，两个txt文件，其中poses.txt中存储3张图片对应的相机初始位姿（Tcw），格式为：timestamp, tx, ty, tz, qx, qy, qz, qw，分别对应时间戳、平移、旋转（四元数），而points.txt中存储的是3D点集合以及该点周围 4x4 窗口的灰度值，记做  $I(p)_i$ ，格式为：

x, y, z, 灰度1, 灰度2..., 灰度16

我们把每个3D点投影到对应图像中，用投影后点周围的灰度值与原始窗口的灰度值差异作为待优化误差。

请使用g2o进行优化，并绘制结果（绘制函数已经写好）。

代码框架中需要你填写顶点、边的定义。如果正确，输出结果如下图所示：



## 参考：

高翔《视觉 SLAM十四讲》 [https://blog.csdn.net/try\\_again\\_later/article/details/81813639](https://blog.csdn.net/try_again_later/article/details/81813639)

代码框架、数据、窗口值具体顺序、优化目标函数、预期输出结果已经为你准备好了，公众号「计算机视觉life」后台回复：**边**，即可获得。

欢迎留言讨论，更多学习视频、文档资料、参考答案等关注计算机视觉life公众号，，菜单栏点击“知识星球”查看「从零开始学习SLAM」星球介绍，快来和其他小伙伴一起学习交流~



知识星球

# 从零开始学习SLAM

自制视频课程  
批改讲解作业  
交流互动答疑  
越早进越优惠



微信扫一扫

推荐阅读

[从零开始一起学习SLAM | 为什么要学SLAM?](#) [从零开始一起学习SLAM | 学习SLAM到底需要学什么?](#) [从零开始一起学习SLAM | SLAM有什么用?](#) [从零开始一起学习SLAM | C++新特性要不要学?](#) [从零开始一起学习SLAM | 为什么要用齐次坐标?](#) [从零开始一起学习SLAM | 三维空间刚体的旋转](#) [从零开始一起学习SLAM | 为啥需要李群与李代数?](#) [从零开始一起学习SLAM | 相机成像模型](#) [从零开始一起学习SLAM | 不推公式，如何真正理解对极约束?](#) [从零开始一起学习SLAM | 神奇的单应矩阵](#) [从零开始一起学习SLAM | 你好，点云](#) [从零开始一起学习SLAM | 给点云加个滤网](#) [从零开始一起学习SLAM | 点云平滑法线估计](#) [从零开始一起学习SLAM | 点云到网格的进化](#) [从零开始一起学习SLAM | 理解图优化，一步步带你看懂g2o代码](#) [从零开始一起学习SLAM | 掌握g2o顶点编程套路](#) [零基础小白，如何入门计算机视觉？](#) [SLAM领域牛人、牛实验室、牛研究成果梳理](#) [我用MATLAB撸了一个2D LiDAR SLAM](#) [可视化理解四元数，愿你不再掉头发](#) [最近一年语义SLAM有哪些代表性工作？](#) [视觉SLAM技术综述 汇总](#) [VIO、激光SLAM相关论文分类集锦](#) [研究SLAM，对编程的要求有多高？](#) [2018年SLAM、三维视觉方向求职经验分享](#) [深度学习遇到SLAM | 如何评价基于深度学习的DeepVO，VINet，VidLoc？](#)