从零开始一起学习SLAM | 掌握g2o顶点编程套路

小白：师兄，上一次将的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！

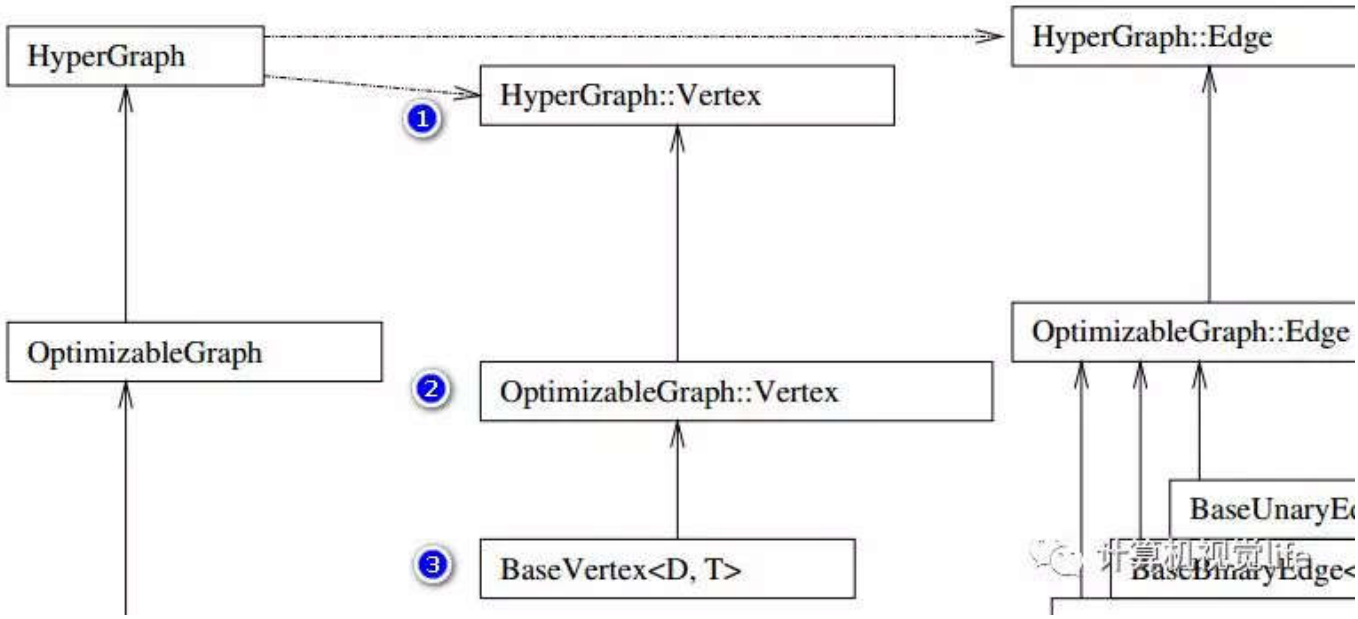
师兄：那太好啦，以后多练习练习，加深理解

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        /**

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

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();
```


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

那么它在哪里呢？ 在这个路径：

g2o/core/base_vertex.h

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
```

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，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
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
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 par
// qw is assumed to be positive, otherwise there is an ambiguity in qx,qy,qz as a rotati
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 t
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Δx 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 + △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

```
/**

 \* \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。

本文参考：

高翔《视觉SLAM十四讲》

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

推荐阅读