GTSAM 库用于 SFM（以 class 类型排版）

1. SFM 简介

[该类型的内容暂不支持下载]

1.1 SFM 定义

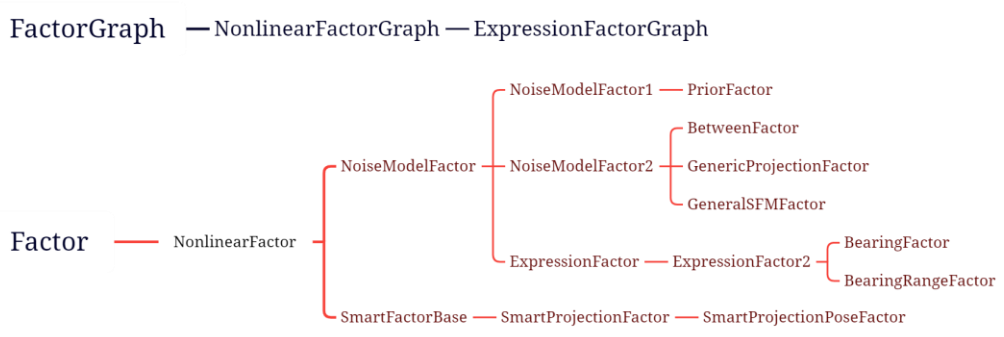
```
C++
通过相机的移动来确定目标的空间和几何关系，是三维重建的一种常见方法。
它与 Kinect 这种 3D 摄像头最大的不同在于，它只需要普通的 RGB 摄像头即可，
因此成本更低廉，且受环境约束较小，在室内和室外均能使用。
```

1.2 SFM 算法流程

计算前两个摄像机之间的位姿变换:

```
C++
Ⅰ、特征点提取与特征点匹配

Ⅱ、基础矩阵估计 F：五点法或者八点法（RANSAC）

Ⅲ、本质矩阵估计 E

Ⅳ、本质矩阵分解为 R 和 T

Ⅴ、三维点云计算

Ⅵ、重投影

Ⅶ、计算第三个摄像机到到世界坐标系的变换矩阵(R 和 T)

Ⅷ、更多摄像相机的变换矩阵计算

Ⅸ、重构的细化与优化：BA、Ceres slover、gtsam
```

2. Graph



2.1 NonlinearFactorGraph

继承自 FactorGraph:

```
C++
/**
     * Directly add ExpressionFactor that implements |h(x)-z|^2_R
     * @param h expression that implements measurement function
     * @param z measurement
     * @param R model
     */
    template<typename T>
    void addExpressionFactor(const SharedNoiseModel& R, const T&
z,
                              const Expression<T>& h) {
      push_back(boost::make_shared<ExpressionFactor<T> >(R, z,
h));
    }
```

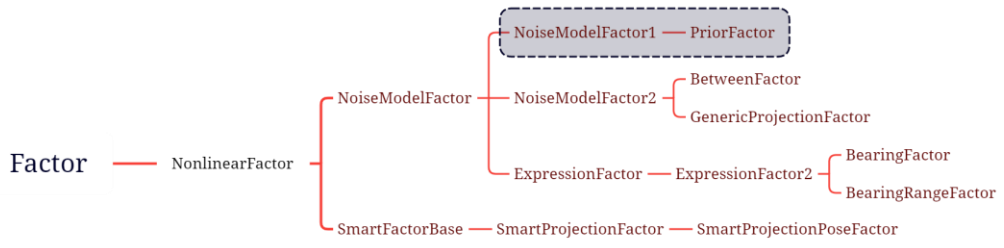
2.2 ExpressionFactorGraph

```
C++
/**
   * Directly add ExpressionFactor that implements |h(x)-z|^2_R
   * @param h expression that implements measurement function
   * @param z measurement
   * @param R model
   */
  template<typename T>
  void addExpressionFactor(const Expression<T>& h, const T& z,
      const SharedNoiseModel& R) {
```

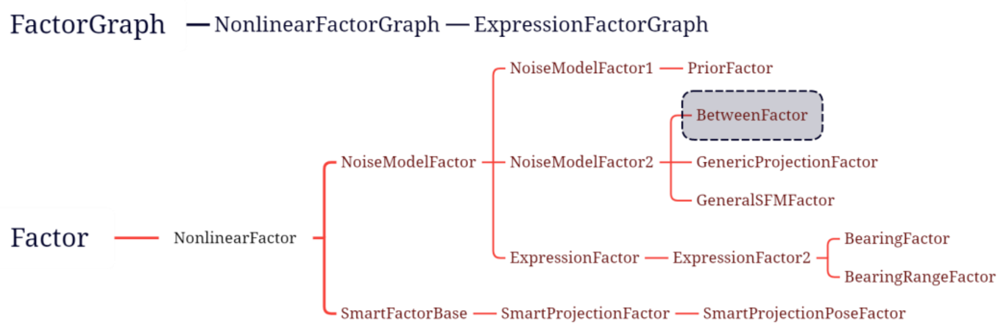
```
    using F = ExpressionFactor<T>;

push_back(boost::allocate_shared<F>(Eigen::aligned_allocator<F>(),
R, z, h));
  }
```

3. Factors

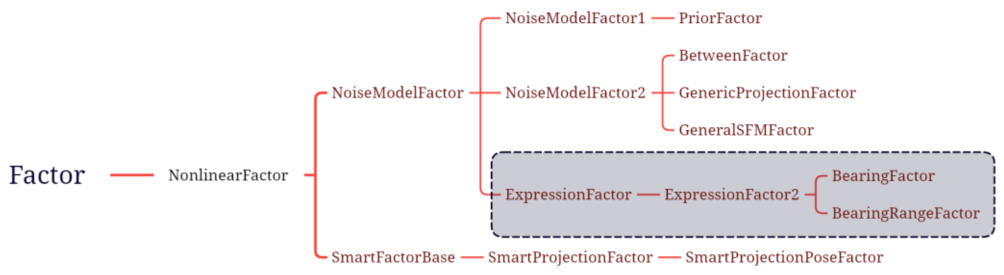
3.1 PriorFactor



3.2 BetweenFactor



3.3 Expression & ExpressionFactor



```
C++
/**
   * Constructor: creates a factor from a measurement and
measurement function
   *   @param noiseModel the noise model associated with a
measurement
   *   @param measurement actual value of the measurement, of type
T
   *   @param expression predicts the measurement from Values
   * The keys associated with the factor, returned by keys(), are
sorted.
   */
 /**
* 构造函数:从测量和测量函数创建因子
* @param noiseModel:     与测量相关联的噪声模型
* @param measurement:    测量的实际值，类型为 T
* @param expression :    从 Values 中预测测量值
*由 keys()返回的与因子相关的键被排序。
*/
  ExpressionFactor(const SharedNoiseModel& noiseModel,  //
                   const T& measurement, const Expression<T>&
expression)
      : NoiseModelFactor(noiseModel), measured_(measurement) {
    initialize(expression);
  }
```

引用<Eigen/Dense>，jacbian 矩阵。

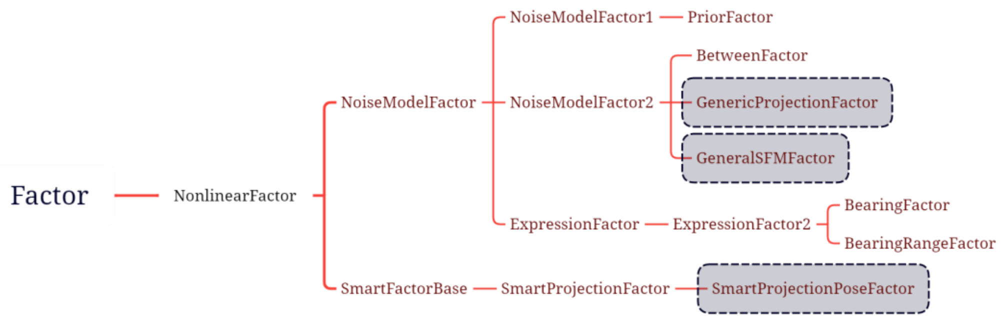
相关函数为：MakeOptionalJacobian

```
C++
// Expressions wrap trees of functions that can evaluate their own
derivatives.
  // The meta-functions below are useful to specify the type of
those functions.
  // Example, a function taking a camera and a 3D point and
yielding a 2D point:
  //
Expression<Point2>::BinaryFunction<SimpleCamera,Point3>::type
  template<class A1>
  struct UnaryFunction {
    typedef boost::function<
        T(const A1&, typename MakeOptionalJacobian<T, A1>::type)>
type;
```

```
  };

  template<class A1, class A2>
  struct BinaryFunction {
    typedef boost::function<
        T(const A1&, const A2&, typename MakeOptionalJacobian<T,
A1>::type,
            typename MakeOptionalJacobian<T, A2>::type)> type;
  };

  template<class A1, class A2, class A3>
  struct TernaryFunction {
    typedef boost::function<
        T(const A1&, const A2&, const A3&,
            typename MakeOptionalJacobian<T, A1>::type,
            typename MakeOptionalJacobian<T, A2>::type,
            typename MakeOptionalJacobian<T, A3>::type)> type;
  };
```

3.4 GenericProjectionFactor & SmartFactor & GeneralSFMFactor

```
C++
// Make the typename short so it looks much cleaner
typedef SmartProjectionPoseFactor<Cal3_S2> SmartFactor;
```



区别:

1.SmartFactor 设定所有时刻相机的内参为常值

```
C++
/**
 * This factor assumes that camera calibration is fixed, and that
 * the calibration is the same for all cameras involved in this
factor.
 * The factor only constrains poses (variable dimension is 6).
 * This factor requires that values contains the involved poses
(Pose3).
 * If the calibration should be optimized, as well, use
SmartProjectionFactor instead!
 * @addtogroup SLAM
 */
 /**
 该因素假设摄像机校准是固定的，并且
 *该系数中涉及的所有摄像机的校准是相同的。
 *该因子仅约束姿势(可变维度为 6)。
 *该因子要求值包含所涉及的姿势(姿势 3)。
 *如果校准也需要优化，请使用 SmartProjectionFactor!
  */
```

4. Optimizer

4.1 LevenbergMarquardtOptimizer

4.2 GaussNewtonOptimizer

4.3 DoglegOptimizer

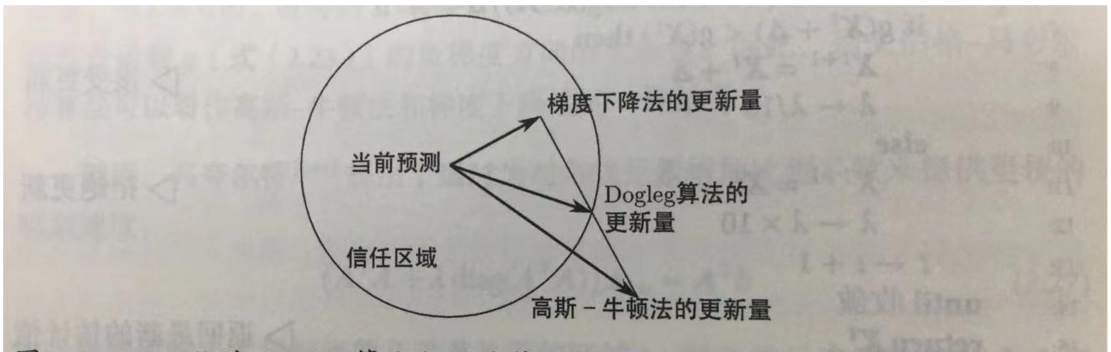


图 2-2　Powell 的 Dogleg 算法分别计算了高斯–牛顿法的更新量和梯度下降法的更新量，并将它们结合起来

4.4 ISAM2