Pose Graph 3D (位姿图优化)

有关的Ceres的`pose_graph_3d.cc`的代码在[Ceres官方][1]的GitHub库里

下面会一行行地介绍官方的代码, 相关的有两个文件, 分别是`pose_graph_3d.cc` 和 `pose_graph_3d_error_term.h`

pose_graph_3d.cc

这是主文件用来做整个位姿图优化的, 接下来我们从`main`这个函数开始说起

main

```
  google::InitGoogleLogging(argv[0]);
  GFLAGS_NAMESPACE::ParseCommandLineFlags(&argc, &argv, true);

  CHECK(FLAGS_input != "") << "Need to specify the filename to read.";
```

这里主要是确认有没有输入一个文件名, 这个文件是用来存着这个位姿图优化的问题的

```
ceres::examples::MapOfPoses poses;
ceres::examples::VectorOfConstraints constraints;
```

位姿图优化需要加载一串的位姿和相关位姿之间的constraint, 仔细看一下位姿和constraint的定义

```
typedef std::map<int,
                 Pose3d,
                 std::less<int>,
                 Eigen::aligned_allocator<std::pair<const int, Pose3d>>>
    MapOfPoses;
```

位姿的定义需要这个位姿的ID, 3D的实际位姿

```
typedef std::vector<Constraint3d, Eigen::aligned_allocator<Constraint3d>>
    VectorOfConstraints;
```

constraint的定义需要再深入看一下这个 `Constraint3d` 的定义

```
struct Constraint3d {
  int id_begin;
```

```
  int id_end;

  // The transformation that represents the pose of the end frame E w.r.t. the
  // begin frame B. In other words, it transforms a vector in the E frame to
  // the B frame.
  Pose3d t_be;

  // The inverse of the covariance matrix for the measurement. The order of the
  // entries are x, y, z, delta orientation.
  Eigen::Matrix<double, 6, 6> information;

  // The name of the data type in the g2o file format.
  static std::string name() { return "EDGE_SE3:QUAT"; }

  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};
```

这里可以看出, 定义一个constraint, 需要知道这个constraint连接的是哪两个位姿, 这里用位姿的ID来表示, 另外还需要知道这两个位姿之间的相对位姿. 除此之外, 还需要用到有关这个相对位子的information matrix, 也就是inverse covariance matrix

```
CHECK(ceres::examples::ReadG2oFile(FLAGS_input, &poses, &constraints))
    << "Error reading the file: " << FLAGS_input;
```

这里回到main的函数, 接下来会把文件里的位姿图问题给加载到之前的两个容器里

这里举两个例子

```
VERTEX_SE3:QUAT 7 29.583 0.104272 -0.194878 -0.00678458 0.0141998 0.012983
0.999792
```

在一个g2o文件里, 这一行可以用来表示一个节点, 也就是之前所说的一个位姿

```
EDGE_SE3:QUAT 879 880 4.1973 0.687599 0.00927741 0.000133331 -0.00104311
0.20361 0.979052 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 4.00002 4.69166e-06 -0.00814815
4 -0.000822411 3.83419
```

这一行可以用来表示两个节点之间的相对位姿

```
ceres::Problem problem;
ceres::examples::BuildOptimizationProblem(constraints, &poses, &problem);
```

BuildOptimizationProblem

下面这里可以开始定义我们要解决的位姿图优化的问题, 我们可以进入`BuildOptimizationProblem`来看一下细节

```
ceres::LossFunction* loss_function = NULL;
ceres::LocalParameterization* quaternion_local_parameterization =
    new EigenQuaternionParameterization;
```

这部分是应对quaternion常用的设置

```
for (VectorOfConstraints::const_iterator constraints_iter =
         constraints.begin();
     constraints_iter != constraints.end();
     ++constraints_iter) {
```

接下来要循环每一个加载的constraint

```
const Eigen::Matrix<double, 6, 6> sqrt_information =
    constraint.information.llt().matrixL();
```

对每一个constraint, 提前做Cholesky Decomposition并且把得到的LL^T的L算出来留待之后所用

```
ceres::CostFunction* cost_function =
    PoseGraph3dErrorTerm::Create(constraint.t_be, sqrt_information);
```

对每一个constraint, 现在开始创造一个残差函数, 这样会引入到下一个文件

pose_graph_3d_error_term.h

```
static ceres::CostFunction* Create(
    const Pose3d& t_ab_measured,
    const Eigen::Matrix<double, 6, 6>& sqrt_information) {
  return new ceres::AutoDiffCostFunction<PoseGraph3dErrorTerm, 6, 3, 4, 3, 4>(
      new PoseGraph3dErrorTerm(t_ab_measured, sqrt_information));
}
```

这里可以看到, 对每一个constraint所创造出来的残差函数, 会输入一个constraint的测量值, 也就是从之前g2o文件的`EDGE_SE3:QUAT`提取出来的, 这里的`sqrt_information`则是刚才提到的Cholesky Decomposition里的L

这里会发现, 每一个残差函数是输出维度为6的残差

```
bool operator()(const T* const p_a_ptr,
                const T* const q_a_ptr,
                const T* const p_b_ptr,
                const T* const q_b_ptr,
                T* residuals_ptr) const {
```

进入到具体的残差函数的定义, 会发现之前的`<PoseGraph3dErrorTerm, 6, 3, 4, 3, 4>`分别定义了残差, 节点A的位置, 节点A的旋转, 节点B的位置, 节点B的旋转的维度

```
// Compute the relative transformation between the two frames.
Eigen::Quaternion<T> q_a_inverse = q_a.conjugate();
Eigen::Quaternion<T> q_ab_estimated = q_a_inverse * q_b;
```

这里通过对quaternion的运算, 可以用已知的节点A和节点B的旋转计算出这两个节点的相对旋转

```
// Represent the displacement between the two frames in the A frame.
Eigen::Matrix<T, 3, 1> p_ab_estimated = q_a_inverse * (p_b - p_a);
```

同理, 可以计算出节点A和节点B的相对位移

```
// Compute the error between the two orientation estimates.
Eigen::Quaternion<T> delta_q =
    t_ab_measured_.q.template cast<T>() * q_ab_estimated.conjugate();
```

把计算出的相对旋转和之前的测量值相比, 可以得出旋转的残差

```
// Compute the residuals.
// [ position         ]   [ delta_p          ]
// [ orientation (3x1)] = [ 2 * delta_q(0:2) ]
Eigen::Map<Eigen::Matrix<T, 6, 1>> residuals(residuals_ptr);
residuals.template block<3, 1>(0, 0) =
    p_ab_estimated - t_ab_measured_.p.template cast<T>();
residuals.template block<3, 1>(3, 0) = T(2.0) * delta_q.vec();
```

同理, 可以得出相对位移的残差

```
// Scale the residuals by the measurement uncertainty.
residuals.applyOnTheLeft(sqrt_information_.template cast<T>());
```

将这两种残差(旋转和相对位移)使用之前的L来scale到同一个level

这样就是基本完成了对整个残差函数的定义

```
problem->AddResidualBlock(cost_function,
                          loss_function,
                          pose_begin_iter->second.p.data(),
                          pose_begin_iter->second.q.coeffs().data(),
                          pose_end_iter->second.p.data(),
                          pose_end_iter->second.q.coeffs().data());

problem->SetParameterization(pose_begin_iter->second.q.coeffs().data(),
                             quaternion_local_parameterization);
problem->SetParameterization(pose_end_iter->second.q.coeffs().data(),
                             quaternion_local_parameterization);
```

还剩下的这些步骤就是将残差函数添加到之前定义的`problem`里去

```
// Returns true if the solve was successful.
bool SolveOptimizationProblem(ceres::Problem* problem) {
  CHECK(problem != NULL);

  ceres::Solver::Options options;
  options.max_num_iterations = 200;
  options.linear_solver_type = ceres::SPARSE_NORMAL_CHOLESKY;

  ceres::Solver::Summary summary;
  ceres::Solve(options, problem, &summary);

  std::cout << summary.FullReport() << '\n';

  return summary.IsSolutionUsable();
}
```

接下来可以回到最开始的main函数里并且通过`ceres::SPARSE_NORMAL_CHOLESKY`来解这个问题

各种链接

[1] https://github.com/ceres-solver/ceres-solver/blob/master/examples/slam/pose_graph_3d/pose_graph_3d.cc