SLAM14讲—06非线性优化

6.1状态估计问题

6.1.1最大后验与最大似然

经典SLAM模型由一个状态方程和一个运动方程构成：

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j} \end{cases}. \tag{6.1}$$

$\mathbf{x}_k$是相机的位姿，可以使用变换矩阵或李代数表示它。至于观测方程即针孔相机模型。

- 位姿变量$\mathbf{x}_k$可以由$\mathbf{T}_k$或者$\exp(\hat{\boldsymbol{\xi}}_k)$表达，二者是等价的。
- 假设在$\mathbf{x}_k$处对路标$\mathbf{y}_j$进行了一次观测，对应到图像上的像素位置$\mathbf{z}_{k,j}$，那么观测方程可以表示为：

  $$s\boldsymbol{z}_{k,j} = \boldsymbol{K}\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{y}_j. \tag{6.2}$$

  这里的$\mathbf{K}$是相机内参，s为像素点的距离。同时，这里的$\mathbf{z}_{k,j}$和$\mathbf{y}_j$都必须以齐次坐标来描述，且中间有一次齐次到非齐次的转换。

现在，考虑数据受噪声的影响后，会发生什么变化。在运动和观测方程中，我们通常假设两个噪声项$\mathbf{w}_k$，$\mathbf{v}_{k,j}$满足零均值的高斯分布：

$$\boldsymbol{w}_k \sim N\left(0, \boldsymbol{R}_k\right), \boldsymbol{v}_k \sim N\left(0, \boldsymbol{Q}_{k,j}\right). \tag{6.3}$$

在这些噪声的影响下，我们希望通过带噪声的数据$\mathbf{z}$和$\mathbf{u}$，推断位姿$\mathbf{x}$和地图$\mathbf{y}$（以及它们的概率分布），这构成了一个状态估计问题。可以通过滤波器（扩展卡尔曼滤波器）或非线性优化方法进行解决。

$$P(\boldsymbol{x}|\boldsymbol{z}) = \frac{P(\boldsymbol{z}|\boldsymbol{x})\,P(\boldsymbol{x})}{P(\boldsymbol{z})} \propto P(\boldsymbol{z}|\boldsymbol{x})\,P(\boldsymbol{x})\,. \tag{6.5}$$

贝叶斯法则左侧通常称为**后验概率**。它右侧的 $P(\boldsymbol{z}|\boldsymbol{x})$ 称为**似然**，另一部分 $P(\boldsymbol{x})$ 称为**先验**。**直接求后验分布是困难的，但是求一个状态最优估计，使得在该状态下，后验概率最大化（Maximize a Posterior，MAP），则是可行的：**

$$\boldsymbol{x}^*{}_{MAP} = \arg\max P(\boldsymbol{x}|\boldsymbol{z}) = \arg\max P(\boldsymbol{z}|\boldsymbol{x})P(\boldsymbol{x}). \tag{6.6}$$

请注意贝叶斯法则的分母部分与待估计的状态 $\boldsymbol{x}$ 无关，因而可以忽略。贝叶斯法则告诉我们，求解最大后验概率，**相当于最大化似然和先验的乘积**。进一步，我们当然也可以说，对不起，我不知道机器人位姿大概在什么地方，此时就没有了**先验**。那么，可以求解 $\boldsymbol{x}$ 的**最大似然估计（Maximize Likelihood Estimation, MLE）：**

$$\boldsymbol{x}^*{}_{MLE} = \arg\max P(\boldsymbol{z}|\boldsymbol{x}). \tag{6.7}$$

直观地说，似然是指“在现在的位姿下，可能产生怎样的观测数据”。由于我们知道观测数据，所以最大似然估计，可以理解成：**“在什么样的状态下，最可能产生现在观测到的数据”**。这就是最大似然估计的直观意义。

6.1.2最小二乘的引出

6.2非线性最小二乘

1. 给定某个初始值 $\boldsymbol{x}_0$。
2. 对于第 k 次迭代，寻找一个增量 $\Delta\boldsymbol{x}_k$，使得 $\|f(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\|_2^2$ 达到极小值。
3. 若 $\Delta\boldsymbol{x}_k$ 足够小，则停止。
4. 否则，令 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$，返回 2.

- Gauss-Newton

1. 给定初始值 $\boldsymbol{x}_0$。
2. 对于第 k 次迭代，求出当前的雅可比矩阵 $\boldsymbol{J}(\boldsymbol{x}_k)$ 和误差 $f(\boldsymbol{x}_k)$。
3. 求解增量方程：$\boldsymbol{H}\Delta\boldsymbol{x}_k = \boldsymbol{g}$.
4. 若 $\Delta\boldsymbol{x}_k$ 足够小，则停止。否则，令 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$，返回 2.

- Levenberg-Marquadt

1. 给定初始值 $\boldsymbol{x}_0$，以及初始优化半径 μ。
2. 对于第 k 次迭代，求解：

$$\min_{\Delta\boldsymbol{x}_k} \frac{1}{2}\|f(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\|^2, \quad s.t. \|\boldsymbol{D}\Delta\boldsymbol{x}_k\|^2 \leq \mu, \tag{6.24}$$

   这里 μ 是信赖区域的半径，$\boldsymbol{D}$ 将在后文说明。
3. 计算 ρ。
4. 若 $\rho > \frac{3}{4}$，则 $\mu = 2\mu$;
5. 若 $\rho < \frac{1}{4}$，则 $\mu = 0.5\mu$;
6. 如果 ρ 大于某阈值，认为近似可行。令 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$。
7. 判断算法是否收敛。如不收敛则返回 2，否则结束。

6.3实践：Ceres

安装代码如下：

```
sudo apt-get install liblapack-dev libsuitesparse-dev libcxsparse3.1.2 libgflags-dev libgoogle-glog-dev
 libgtest-dev
```

然后，进入 Ceres 库，使用 cmake 编译并安装它。这个过程我们已经做过很多遍了，此处就不再赘述。安装完成后，在/usr/local/include/ceres 下找到 Ceres 的头文件，并在/usr/local/lib/下找到名为 libceres.a 的库文件。有了头文件和库文件，就可以使用 Ceres 进行优化计算了。

第三个库无法安装时，如下操作：

1. 打开source.list

```
sudo gedit /etc/apt/sources.list
```

2.然后将下面的源粘贴到source.list的最上方

```
deb http://cz.archive.ubuntu.com/ubuntu trusty main universe
```

3.更新源：

```
sudo apt-get update
```

4.安装依赖项

```
sudo apt-get install libcxsparse3.1.4
```

后面的glog和gflags事先已经装过（可具体看ICE-BA配置环境血泪史）

6.4实践：g2o

在使用一个库之前，我们需要对它进行编译和安装。读者应该已经体验很多次这个过程了，它们基本都是大同小异的。关于 g2o，读者可以从 github 下载它：https://github.com/RainerKuemmerle/g2o，或从本书提供的第三方代码库中获得。

解压代码包后，你会看到 g2o 库的所有源码，它也是一个 CMake 工程。我们先来安装它的依赖项（部分依赖项与 Ceres 有重合）：


```
sudo apt-get install libqt4-dev qt4-qmake libqglviewer-dev libsuitesparse-dev libcxsparse3.1.2
libcholmod-dev
```

然后，按照 cmake 的方式对 g2o 进行编译安装即可，我们略去该过程的说明。安装完成后，g2o 的头文件将在/usr/local/g2o 下，库文件在/usr/local/lib/下。现在，我们重新考虑 Ceres 例程中的曲线拟合实验，在 g2o 中实验一遍。

libcholmod-dev找不到时，输入以下命令后按tab键自动补全后安装对应版本

```
sudo apt-get install libcholmod
```

本次代码中有误需进行修改：
错误代码如下：

```
typedef g2o::BlockSolver< g2o::BlockSolverTraits<3,1> > Block;
Block::LinearSolverType* linearSolver = new g2o::LinearSolverDense<Block::PoseMatrixType>();
Block* solver_ptr = new Block( linearSolver );
g2o::OptimizationAlgorithmLevenberg* solver = new g2o::OptimizationAlgorithmLevenberg(
solver_ptr );
g2o::SparseOptimizer optimizer;
optimizer.setAlgorithm( solver );
optimizer.setVerbose( true );
```

修改正确代码如下所示：

```
typedef g2o::BlockSolver< g2o::BlockSolverTraits<3,1> > Block;
Block::LinearSolverType* linearSolver = new g2o::LinearSolverDense<Block::PoseMatrixType>();
Block* solver_ptr = new Block( std::unique_ptr<Block::LinearSolverType>(linearSolver) );
g2o::OptimizationAlgorithmLevenberg* solver = new
g2o::OptimizationAlgorithmLevenberg(std::unique_ptr<Block>(solver_ptr) );
g2o::SparseOptimizer optimizer;
optimizer.setAlgorithm( solver );
optimizer.setVerbose( true );
```

错误原因如下：
unique_ptr“唯一”拥有其所指对象，同一时刻只能有一个unique_ptr指向给定对象（通过禁止拷贝语义、只有移动语义来实现）。
将初始化中用等号赋值的地方改成用std::unique_ptr赋值，将linearSolver和solver_ptr用移动语言也就是std::move来实现。