2018 年第 7 次课习题答案

——Vance 吴方熠

2 Bundle Adjustment (5 分，约 3 小时)

2.1 文献阅读 (2 分)

我们在第五讲中已经介绍了 Bundle Adjustment，指明它可以用于解 PnP 问题。现在，我们又在后端中说明了它可以用于解大规模的三维重构问题，但在实时 SLAM 场合往往需要控制规模。事实上，Bundle Adjustment 的历史远比我们想象的要长。请阅读 Bill Triggs 的经典论文 Bundle Adjustment: A Modern Synthesis（见 paper/目录），了解 BA 的发展历史，然后回答下列问题：

1. 为何说 Bundle Adjustment is slow 是不对的？
2. BA 中有哪些需要注意参数化的地方？Pose 和 Point 各有哪些参数化方式？有何优缺点。
3. * 本文写于 2000 年，但是文中提到的很多内容在后面十几年的研究中得到了印证。你能看到哪些方向在后续工作中有所体现？请举例说明。

答：

1. Bundle Adjustment is slow 不对是因为之前的研究中没有考虑到雅克比矩阵 J 和海森矩阵 H 的稀疏性质。根据《十四讲》中 10.2.3 小节，对于稀疏的 H 可以用 Schur 消元法等方法来实现边缘化（Marginalization），从而加速 H 的计算，进而提高整体 BA 的速度。
2. BA 中需要注意参数化的地方有：3D 路标点和旋转。3D 路标点可以表示成非齐次和齐次形式，若用非齐次形式表示，当点的距离很远时，点的坐标数值将会很大，此时的代价函数会变得很单调，步长将会变得很大；若使用齐次坐标来表示，则可以控制步长。而旋转的参数化方式可以用欧拉角、四元素和旋转矩阵。用欧拉角会存在约束和万向锁的问题。
3. 海森矩阵 H 的稀疏性可以让 BA 实现实时，这在 07 年 Georg Klein 和 David Murray 提出的 PTAM 上得到了体现。

2.2 BAL-dataset（3 分）

BAL（Bundle Adjustment in large）数据集（http://grail.cs.washington.edu/projects/bal/）是一个大型 BA 数据集，它提供了相机与点初始值与观测，你可以用它们进行 Bundle Adjustment。现在，请你使用 g2o，自己定义 Vertex 和 Edge（不要使用自带的顶点类型，也不要像本书例程那边调用 Ceres 来求导），书写 BAL 上的 BA 程序。你可以挑选其中一个数据，运行你的 BA，并给出优化后的点云图。

本题不提供代码框架，请独立完成。提示：

1. 注意 BAL 的投影模型比教材中介绍的多了个负号；

解：

根据 BAL 网站介绍的模型及数据格式，可确定相机的数据维度为 9 维，空间点的维度为 3 维，误差维度为 2 维。已知：

$$\begin{cases} \boldsymbol{P}_c = \boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t} = exp(\boldsymbol{\xi}^{\wedge})\boldsymbol{P}_w \\ \boldsymbol{P}_n = -\boldsymbol{P}_c/\boldsymbol{P}_c(z) \\ \boldsymbol{P}'_{\boldsymbol{uv}} = f \cdot r(\boldsymbol{P}_n) \cdot \boldsymbol{P}_n \\ r(\boldsymbol{P}_n) = 1.0 + k_1 \cdot \left|\left|\boldsymbol{P}_n\right|\right|^2 + k_2 \cdot ||\boldsymbol{P}_n||^4 \end{cases} \tag{2.1}$$

其中$\boldsymbol{P}_w$、$\boldsymbol{P}_c$、$\boldsymbol{P}_n$、$\boldsymbol{P}_{uv}$分别表示点 $\boldsymbol{P}$ 在世界坐标系、相机坐标系、归一化相机坐标系和像素坐标系的坐标。

可确定误差函数为：

$$\boldsymbol{e}(\boldsymbol{\xi}, f, k_1, k_2, \boldsymbol{P}_w) = \boldsymbol{P}_{uv} - \boldsymbol{P}'_{\boldsymbol{uv}} \tag{2.2}$$

则整体的雅克比矩阵为：

$$\begin{aligned} \boldsymbol{J} &= \begin{bmatrix} \boldsymbol{J}_{\delta\xi}^{2\times6} & \boldsymbol{J}_f^{2\times1} & \boldsymbol{J}_{k_1}^{2\times1} & \boldsymbol{J}_{k_2}^{2\times1} & \boldsymbol{J}_{P_w}^{2\times3} \end{bmatrix} \\ &= \begin{bmatrix} \dfrac{\partial e}{\partial \delta \boldsymbol{\xi}} & \dfrac{\partial e}{\partial f} & \dfrac{\partial e}{\partial k_1} & \dfrac{\partial e}{\partial k_2} & \dfrac{\partial e}{\partial \boldsymbol{P}_w} \end{bmatrix} \end{aligned} \tag{2.3}$$

下面我们分别讨论以上几项的值。根据公式（2.1）、（2.2）及链式求导法则可得：

$$\frac{\partial e}{\partial \delta \boldsymbol{\xi}} = \frac{\partial e}{\partial \boldsymbol{P}_c} \cdot \frac{\partial \boldsymbol{P}_c}{\partial \delta \boldsymbol{\xi}} \tag{2.4}$$

将公式（2.2）展开，有：

$$\begin{aligned} e &= \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} + f(1 + k_1 r^2 + k_2 r^4) \begin{bmatrix} x_c/z_c \\ y_c/z_c \\ 1 \end{bmatrix} \\ e_u &= u + f(1 + k_1 r^2 + k_2 r^4)\frac{x_c}{z_c} \\ e_v &= u + f(1 + k_1 r^2 + k_2 r^4)\frac{y_c}{z_c} \end{aligned} \tag{2.5}$$

其中，$r^2=(\frac{x_c}{z_c})^2+(\frac{y_c}{z_c})^2$。对公式（2.5）进行求导可得：

$$\begin{aligned}
&\frac{\partial e_u}{\partial x_c}=\frac{f}{z_c}(1+k_1r^2+k_2r^4)+\frac{2fx_c{}^2}{z_c{}^3}(k_1+2k_2r^2)\\
&\frac{\partial e_u}{\partial y_c}=\frac{2fx_cy_c}{z_c{}^3}(k_1+2k_2r^2)\\
&\frac{\partial e_u}{\partial z_c}=-\frac{fx_c}{z_c{}^2}(1+k_1r^2+k_2r^4)-\frac{2fx_cr^2}{z_c{}^2}(k_1+2k_2r^2)\\
&\frac{\partial e_v}{\partial x_c}=\frac{2fx_cy_c}{z_c{}^3}(k_1+2k_2r^2)\\
&\frac{\partial e_v}{\partial y_c}=\frac{f}{z_c}(1+k_1r^2+k_2r^4)+\frac{2fy_c{}^2}{z_c{}^3}(k_1+2k_2r^2)\\
&\frac{\partial e_v}{\partial z_c}=-\frac{fy_c}{z_c{}^2}(1+k_1r^2+k_2r^4)-\frac{2fy_cr^2}{z_c{}^2}(k_1+2k_2r^2)
\end{aligned}\tag{2.6}$$

将公式（2.6）结合起来，即为$\frac{\partial e}{\partial \boldsymbol{P}_c}$，对于$\frac{\partial \boldsymbol{P}_c}{\partial \delta\boldsymbol{\xi}}$ 参考《十四讲》中的 4.3.5 小节，有：

$$\frac{\partial \boldsymbol{P}_c}{\partial \delta\boldsymbol{\xi}}=[I\quad -\boldsymbol{P}_c{}^{\wedge}]\tag{2.7}$$

将公式（2.6）与（2.7）结合即为雅克比矩阵的第一项，对于第 2 至 5 项，根据（2.5）有：

$$\frac{\partial e}{\partial f}=\begin{bmatrix}\frac{x_c}{z_c}(1+k_1r^2+k_2r^4)\\ \frac{y_c}{z_c}(1+k_1r^2+k_2r^4)\end{bmatrix}\tag{2.8}$$

$$\frac{\partial e}{\partial k_1}=\begin{bmatrix}\frac{fx_c}{z_c}r^2\\ \frac{fy_c}{z_c}r^2\end{bmatrix}\tag{2.9}$$

$$\frac{\partial e}{\partial k_2}=\begin{bmatrix}\frac{fx_c}{z_c}r^4\\ \frac{fy_c}{z_c}r^4\end{bmatrix}\tag{2.10}$$

$$\frac{\partial e}{\partial \boldsymbol{P}_w}=\frac{\partial e}{\partial \boldsymbol{P}_c}\cdot\frac{\partial \boldsymbol{P}_c}{\partial \boldsymbol{P}_w}=\frac{\partial e}{\partial \boldsymbol{P}_c}\cdot\boldsymbol{R}\tag{2.11}$$

至此雅克比矩阵以全部求得。

本题源码见附件。

以下为 BAL 数据集“problem-52-64053-pre.txt”的运行结果的截图：

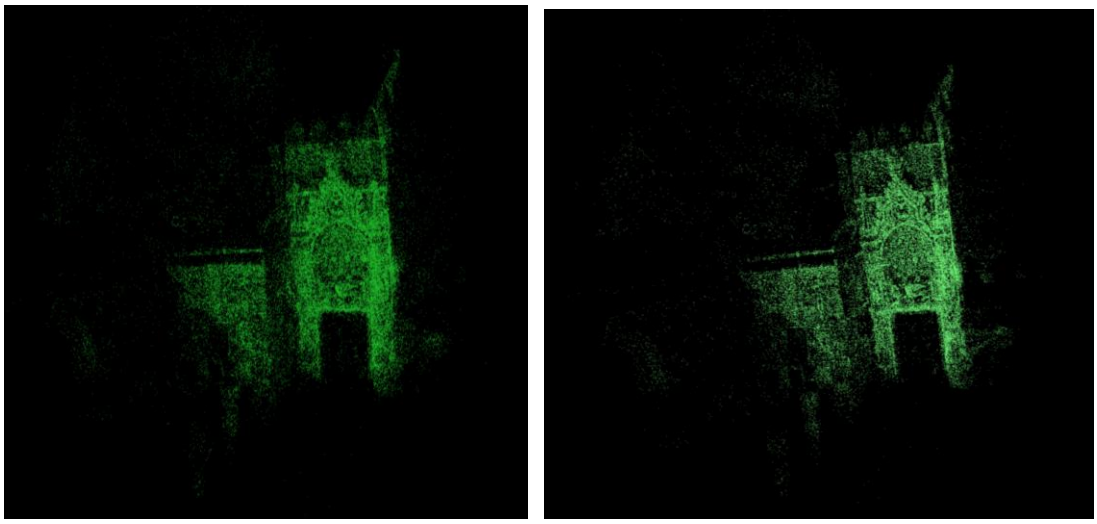
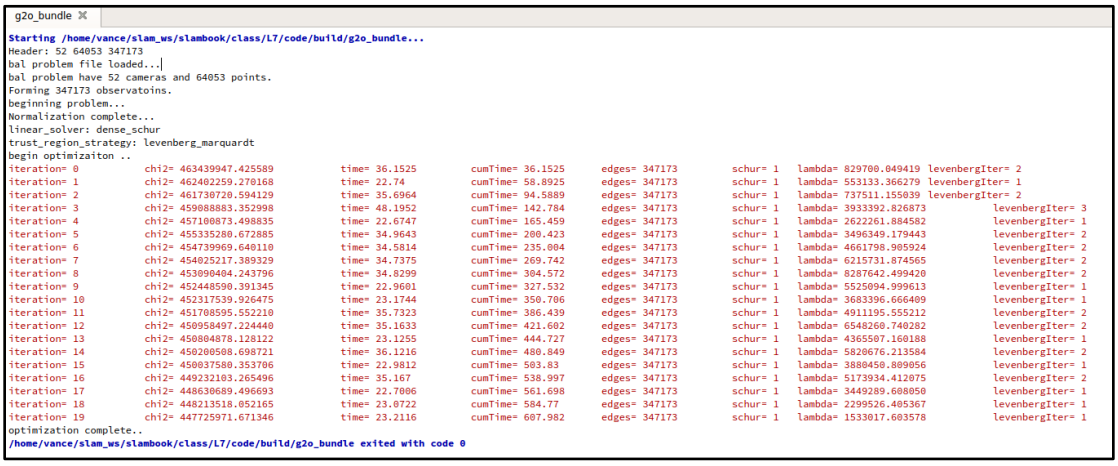


图 1 BAL 数据集优化前（左）后（右）对比图



```
g2o_bundle
Starting /home/vance/slam_ws/slambook/class/L7/code/build/g2o_bundle...
Header: 52 64053 347173
bal problem file loaded...
bal problem have 52 cameras and 64053 points.
Forming 347173 observatoins.
beginning problem...
Normalization complete...
linear_solver: dense_schur
trust_region_strategy: levenberg_marquardt
begin optimizaiton ..
iteration= 0          chi2= 463439947.425589          time= 36.1525          cumTime= 36.1525          edges= 347173          schur= 1   lambda= 829700.049419 levenbergIter= 2
iteration= 1          chi2= 462402259.270168          time= 22.74            cumTime= 58.8925          edges= 347173          schur= 1   lambda= 553133.366279 levenbergIter= 1
iteration= 2          chi2= 461730720.594129          time= 35.6964          cumTime= 94.5889          edges= 347173          schur= 1   lambda= 737511.155039 levenbergIter= 2
iteration= 3          chi2= 459088883.352998          time= 48.1952          cumTime= 142.784          edges= 347173          schur= 1   lambda= 3933392.826873          levenbergIter= 3
iteration= 4          chi2= 457100873.498835          time= 22.6747          cumTime= 165.459          edges= 347173          schur= 1   lambda= 2622261.884582          levenbergIter= 1
iteration= 5          chi2= 455335280.672885          time= 34.9643          cumTime= 200.423          edges= 347173          schur= 1   lambda= 3496349.179443          levenbergIter= 2
iteration= 6          chi2= 454739969.640110          time= 34.5814          cumTime= 235.004          edges= 347173          schur= 1   lambda= 4661798.905924          levenbergIter= 2
iteration= 7          chi2= 454025217.389329          time= 34.7375          cumTime= 269.742          edges= 347173          schur= 1   lambda= 6215731.874565          levenbergIter= 2
iteration= 8          chi2= 453090404.243796          time= 34.8299          cumTime= 304.572          edges= 347173          schur= 1   lambda= 8287642.499420          levenbergIter= 2
iteration= 9          chi2= 452448590.391345          time= 22.9601          cumTime= 327.532          edges= 347173          schur= 1   lambda= 5525094.999613          levenbergIter= 1
iteration= 10         chi2= 452317539.926475          time= 23.1744          cumTime= 350.706          edges= 347173          schur= 1   lambda= 3683396.666409          levenbergIter= 1
iteration= 11         chi2= 451708595.552210          time= 35.7323          cumTime= 386.439          edges= 347173          schur= 1   lambda= 4911195.555212          levenbergIter= 2
iteration= 12         chi2= 450958497.224440          time= 35.1633          cumTime= 421.602          edges= 347173          schur= 1   lambda= 6548260.740282          levenbergIter= 2
iteration= 13         chi2= 450804878.128122          time= 23.1255          cumTime= 444.727          edges= 347173          schur= 1   lambda= 4365507.160188          levenbergIter= 1
iteration= 14         chi2= 450200508.698721          time= 36.1216          cumTime= 480.849          edges= 347173          schur= 1   lambda= 5820676.213584          levenbergIter= 2
iteration= 15         chi2= 450037580.353706          time= 22.9812          cumTime= 503.83           edges= 347173          schur= 1   lambda= 3880450.809056          levenbergIter= 1
iteration= 16         chi2= 449232103.265496          time= 35.167           cumTime= 538.997          edges= 347173          schur= 1   lambda= 5173934.412075          levenbergIter= 2
iteration= 17         chi2= 448630689.496693          time= 22.7006          cumTime= 561.698          edges= 347173          schur= 1   lambda= 3449289.608050          levenbergIter= 1
iteration= 18         chi2= 448213518.052165          time= 23.0722          cumTime= 584.77           edges= 347173          schur= 1   lambda= 2299526.405367          levenbergIter= 1
iteration= 19         chi2= 447725971.671346          time= 23.2116          cumTime= 607.982          edges= 347173          schur= 1   lambda= 1533017.603578          levenbergIter= 1
optimization complete..
/home/vance/slam_ws/slambook/class/L7/code/build/g2o_bundle exited with code 0
```

图 2 BAL 数据集优化过程截图

3 直接法的 Bundle Adjustment (5 分，约 3 小时)

3.1 数学模型

特征点法的 BA 以最小化重投影误差作为优化目标。相对的，如果我们以最小化光度误差为目标，就得到了直接法的 BA。之前我们在直接法 VO 中，谈到了如何用直接法去估计相机位姿。但是直接法亦可用来处理整个 Bundle Adjustment。下面，请你推导直接法 BA 的数学模型，并完成它的 g2o 实现。注意本题使用的参数化形式与实际的直接法还有一点不同，我们用 x, y, z 参数化每一个 3D 点，而实际的直接法多采用逆深度参数化 [1]。

本题给定 7 张图片，记为 0.png 至 6.png，每张图片对应的相机位姿初始值为 $\mathbf{T}_i$，以 $\mathbf{T}_{cw}$ 形式存储在 poses.txt 文件中，其中每一行代表一个相机的位姿，格式如之前作业那样：

$$\text{time}, t_x, t_y, t_z, q_x, q_y, q_z, q_w$$

平移在前，旋转（四元数形式）在后。同时，还存在一个 3D 点集 P，共 N 个点。其中每一个点的初始坐标记作 $\mathbf{p}_i = [x, y, z]_i^{\mathrm{T}}$。每个点还有自己的固定灰度值，我们用 16 个数来描述，这 16 个数为该点周围 4x4 的小块读数，记作 $I(p)_i$，顺序见图 1。换句话说，小块从 $u-2, v-2$ 取到 $u+1, v+1$，先迭代 v。那么，我们知道，可以把每个点投影到每个图像中，然后再看投影后点周围小块与原始的 4x4 小块有多大差异。那么，整体优化目标函数为：

$$\min \sum_{j=1}^{7} \sum_{i=1}^{N} \sum_{W} \| I(\mathbf{p}_i) - I_j(\pi(\mathbf{K}\mathbf{T}_j\mathbf{p}_i)) \|_2^2 \tag{1}$$

即最小化任意点在任意图像中投影与其本身颜色之差。其中 $\mathbf{K}$ 为相机内参（在程序内以全局变量形式给定），π 为投影函数，W 指代整个 patch。下面，请回答：

1. 如何描述任意一点投影在任意一图像中形成的 error?
2. 每个 error 关联几个优化变量?
3. error 关于各变量的雅可比是什么?

答：

1. 对任意一点$\boldsymbol{P}_w$投影在任意一图像j中的误差项 error 可表示为：

$$e(\boldsymbol{P}_w, \boldsymbol{\xi}) = I(\boldsymbol{P}_i) - I_j\left(\frac{1}{z_c}\boldsymbol{K}exp(\boldsymbol{\xi}^{\wedge})\boldsymbol{P}_w\right)$$

其中z_c为点$\boldsymbol{P}_w$投影到相机坐标系下的 z 坐标的值。

2. 每个误差项 error 关联 2 个优化变量，分别是$\boldsymbol{P}_w$和$\boldsymbol{\xi}$。

3. 误差项 error 关于各变量的雅克比矩阵如下：

第一项为$\boldsymbol{J}_{\boldsymbol{P}_w}$：

$$\boldsymbol{J}_{\boldsymbol{P}_w} = \frac{\partial e}{\partial \boldsymbol{P}_w} = \frac{\partial I_j}{\partial \boldsymbol{P}_{uv}} \cdot \frac{\partial \boldsymbol{P}_{uv}}{\partial \boldsymbol{P}_c} \cdot \frac{\partial \boldsymbol{P}_c}{\partial \boldsymbol{P}_w} \tag{3.1}$$

其中$\frac{\partial I_j}{\partial \boldsymbol{P}_{uv}}$为投影点(u, v)的灰度梯度，参考《十四讲》公式（7.42)、(7.47)、(8.13）有：

$$\frac{\partial I_j}{\partial \boldsymbol{P}_{uv}} = \begin{bmatrix} \frac{I_j(u+1,v) - I_j(u-1,v)}{2} & \frac{I_j(u,v+1) - I_j(u,v-1)}{2} \end{bmatrix} \tag{3.2}$$

$$\boldsymbol{P}_{uv}=\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_c/z_c\\y_c/z_c\\1\end{bmatrix}$$

$$u=\frac{f_x x_c}{z_c}\qquad v=\frac{f_y y_c}{z_c}$$

$$\frac{\partial \boldsymbol{P}_{uv}}{\partial \boldsymbol{P}_c}=\begin{bmatrix}\frac{f_x}{z_c} & 0 & -\frac{f_x x_c}{{z_c}^2}\\0 & \frac{f_y}{z_c} & -\frac{f_y y_c}{{z_c}^2}\end{bmatrix} \tag{3.3}$$

$$\frac{\partial \boldsymbol{P}_c}{\partial \boldsymbol{P}_w}=\boldsymbol{R} \tag{3.4}$$

第二项为$\boldsymbol{J}_{\xi}$，可参考《十四讲》8.4.1 小节，得出：

$$\boldsymbol{J}_{\xi}=\frac{\partial e}{\partial \delta \boldsymbol{\xi}}=-\frac{\partial I_j}{\partial \boldsymbol{P}_{uv}}\cdot\frac{\partial \boldsymbol{P}_{uv}}{\partial \boldsymbol{P}_c}\cdot\frac{\partial \boldsymbol{P}_c}{\partial \delta \boldsymbol{\xi}} \tag{3.5}$$

前两项上面已经得出，最后一项可参考《十四讲》4.3.5 小节，有：

$$\frac{\partial \boldsymbol{P}_c}{\partial \delta \boldsymbol{\xi}}=[I\quad -\boldsymbol{P}_c{}^{\wedge}] \tag{3.6}$$

3.2 实现

下面，请你根据上述说明，使用 g2o 实现上述优化，并用 pangolin 绘制优化结果。程序框架见 code/directBA.cpp 文件。实现过程中，思考并回答以下问题：

1. 能否不要以 $[x, y, z]^{\mathrm{T}}$ 的形式参数化每个点？
2. 取 4x4 的 patch 好吗？取更大的 patch 好还是取小一点的 patch 好？
3. 从本题中，你看到直接法与特征点法在 BA 阶段有何不同？
4. 由于图像的差异，你可能需要鲁棒核函数，例如 Huber。此时 Huber 的阈值如何选取？

提示：

1. 构建 Error 之前先要判断点是否在图像中，去除一部分边界的点。
2. 优化之后，Pangolin 绘制的轨迹与地图如图 1 所示。
3. 你也可以不提供雅可比的计算过程，让 g2o 自己计算一个数值雅可比。
4. 以上数据实际取自 DSO[1]。

答：

1. 如果不要以$[x,y,z]^{\mathrm{T}}$的形式参数化每个点的话，可以使用逆深度来参数化。根据论文《Inverse Depth Parametrization for Monocular SLAM》中描述，逆深度参数化点的公式如下图所示：

C. Inverse Depth Point Parametrization

In our new scheme, a scene 3-D point i can be defined by the 6-D state vector:

$$\mathbf{y}_i = (x_i \quad y_i \quad z_i \quad \theta_i \quad \phi_i \quad \rho_i)^\top \tag{3}$$

which models a 3-D point located at (see Fig. 1)

$$\mathbf{x}_i = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i) \tag{4}$$

$$\mathbf{m} = (\cos\,\phi_i \sin\,\theta_i, -\sin\,\phi_i, \cos\,\phi_i \cos\,\theta_i)^\top . \tag{5}$$

The $\mathbf{y}_i$ vector encodes the *ray from the first camera position from which the feature was observed* by x_i, y_i, z_i, the camera optical center, and θ_i, ϕ_i azimuth and elevation (coded in the world frame) defining unit directional vector $\mathbf{m}(\theta_i, \phi_i)$. The point's depth along the ray d_i is encoded by its inverse $\rho_i = 1/d_i$.

图 3 逆深度参数化公式截图

其中$[X_i, Y_i, Z_i]^T$为世界坐标系下点的坐标，$[x_i, y_i, z_i]^{\mathrm{T}}$为相机坐标系下点的坐标，$m(\theta, \emptyset)$ 表示相机光轴到路标点的旋转角度，ρ为相机光心到路标点的距离的导数。

2.patch 取小一点更好。因为在不同的视角下，对于同一个三维点 P 来说，其附近的点的投影点的灰度值会不同。如果 patch 取的太大，要计算的周围点的灰度值变化程度更高，不利于匹配。故取小 patch 有助于提高匹配成功率。

3.直接法中的雅克比需要考虑灰度梯度，而特征点法不需要考虑；直接法中需要引入 patch 块，考虑特征点周围一定数目点的误差项，而特征点法仅考虑重投影误差，只有两个误差项。

4.假设误差项服从高斯分布，则误差项的平方服从卡方分布。根据表 1 中的卡方分布，先确定误差项的自由度（这里是 16 维），再确定置信度（一般假设为 0.95），最后根据自由度和置信度查找卡方分布表就能得知 Huber 的阈值是多少。显然，阈值越低，其置信度就越高。

表 1 卡方分布表

x^2分布临界值表（卡方分布）

n'	P												
	0.995	0.99	0.975	0.95	0.9	0.75	0.5	0.25	0.1	0.05	0.025	0.01	0.005
1	---	---	---	---	0.02	0.1	0.45	1.32	2.71	3.84	5.02	6.63	7.88
2	0.01	0.02	0.02	0.1	0.21	0.58	1.39	2.77	4.61	5.99	7.38	9.21	10.6
3	0.07	0.11	0.22	0.35	0.58	1.21	2.37	4.11	6.25	7.81	9.35	11.34	12.84
4	0.21	0.3	0.48	0.71	1.06	1.92	3.36	5.39	7.78	9.49	11.14	13.28	14.86
5	0.41	0.55	0.83	1.15	1.61	2.67	4.35	6.63	9.24	11.07	12.83	15.09	16.75
6	0.68	0.87	1.24	1.64	2.2	3.45	5.35	7.84	10.64	12.59	14.45	16.81	18.55
7	0.99	1.24	1.69	2.17	2.83	4.25	6.35	9.04	12.02	14.07	16.01	18.48	20.28
8	1.34	1.65	2.18	2.73	3.4	5.07	7.34	10.22	13.36	15.51	17.53	20.09	21.96
9	1.73	2.09	2.7	3.33	4.17	5.9	8.34	11.39	14.68	16.92	19.02	21.67	23.59
10	2.16	2.56	3.25	3.94	4.87	6.74	9.34	12.55	15.99	18.31	20.48	23.21	25.19
11	2.6	3.05	3.82	4.57	5.58	7.58	10.34	13.7	17.28	19.68	21.92	24.72	26.76
12	3.07	3.57	4.4	5.23	6.3	8.44	11.34	14.85	18.55	21.03	23.34	26.22	28.3
13	3.57	4.11	5.01	5.89	7.04	9.3	12.34	15.98	19.81	22.36	24.74	27.69	29.82
14	4.07	4.66	5.63	6.57	7.79	10.17	13.34	17.12	21.06	23.68	26.12	29.14	31.32
15	4.6	5.23	6.27	7.26	8.55	11.04	14.34	18.25	22.31	25	27.49	30.58	32.8
16	5.14	5.81	6.91	7.96	9.31	11.91	15.34	19.37	23.54	26.3	28.85	32	34.27
17	5.7	6.41	7.56	8.67	10.09	12.79	16.34	20.49	24.77	27.59	30.19	33.41	35.72
18	6.26	7.01	8.23	9.39	10.86	13.68	17.34	21.6	25.99	28.87	31.53	34.81	37.16
19	6.84	7.63	8.91	10.12	11.65	14.56	18.34	22.72	27.2	30.14	32.85	36.19	38.58
20	7.43	8.26	9.59	10.85	12.44	15.45	19.34	23.83	28.41	31.41	34.17	37.57	40

本题源码见附件，程序运行的结果截图如下：

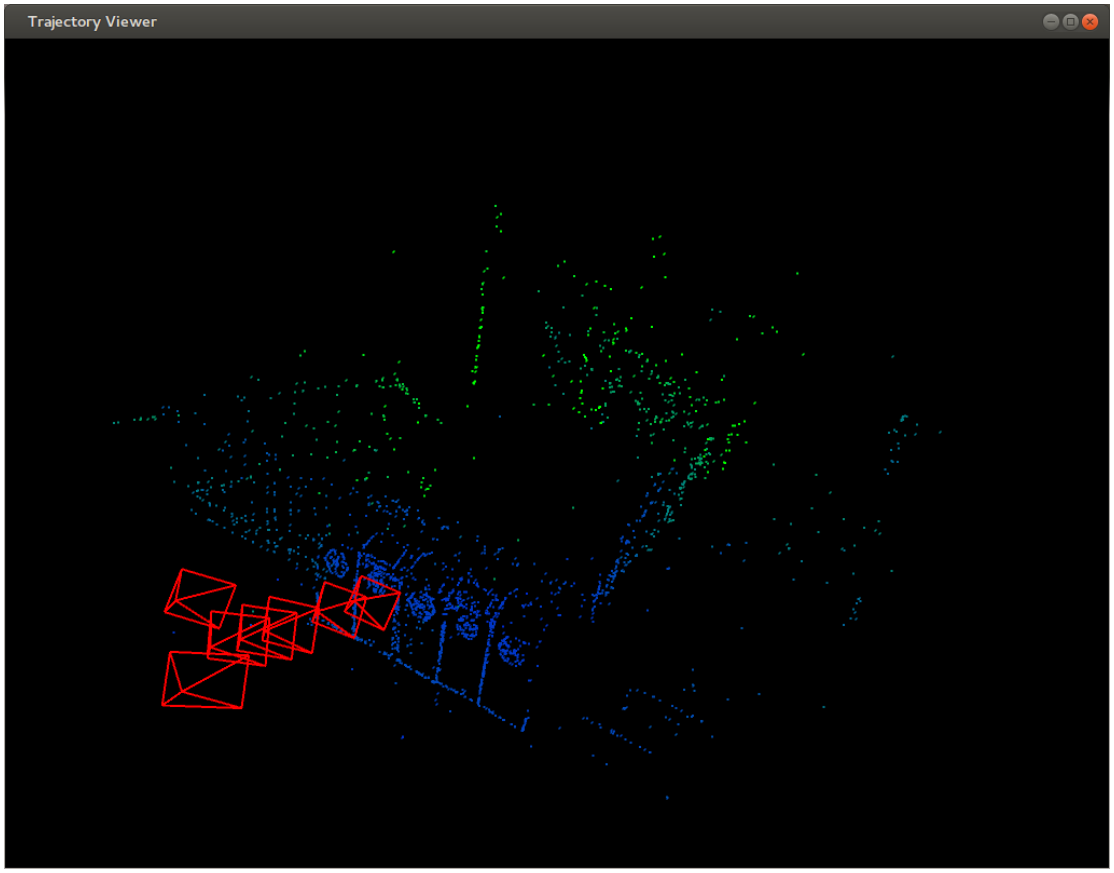


图 4 光流法 BA 优化效果

directBA (disabled)

```
iteration= 0    chi2= 3390801.837720  time= 4.28044    cumTime= 4.28044    edges= 24130    schur= 1    lambda= 353084.637036 levenbergIter= 1
iteration= 1    chi2= 3254418.937140  time= 4.24391    cumTime= 8.52435    edges= 24130    schur= 1    lambda= 117694.879012 levenbergIter= 1
iteration= 2    chi2= 3161874.852353  time= 4.20855    cumTime= 12.7329    edges= 24130    schur= 1    lambda= 39231.626337  levenbergIter= 1
iteration= 3    chi2= 3096958.128049  time= 4.2246     cumTime= 16.9575    edges= 24130    schur= 1    lambda= 13077.208779  levenbergIter= 1
iteration= 4    chi2= 3045653.406289  time= 4.28372    cumTime= 21.2412    edges= 24130    schur= 1    lambda= 4359.069593   levenbergIter= 1
iteration= 5    chi2= 3010667.997942  time= 4.22744    cumTime= 25.4687    edges= 24130    schur= 1    lambda= 2906.046395   levenbergIter= 1
iteration= 6    chi2= 2975290.581957  time= 4.20445    cumTime= 29.6731    edges= 24130    schur= 1    lambda= 1937.364264   levenbergIter= 1
iteration= 7    chi2= 2957734.120737  time= 4.20255    cumTime= 33.8757    edges= 24130    schur= 1    lambda= 1291.576176   levenbergIter= 1
iteration= 8    chi2= 2933629.871687  time= 4.19955    cumTime= 38.0752    edges= 24130    schur= 1    lambda= 861.050784    levenbergIter= 1
iteration= 9    chi2= 2917446.643709  time= 4.19798    cumTime= 42.2732    edges= 24130    schur= 1    lambda= 574.033856    levenbergIter= 1
iteration= 10   chi2= 2901171.247891  time= 4.20682    cumTime= 46.48      edges= 24130    schur= 1    lambda= 382.689237    levenbergIter= 1
iteration= 11   chi2= 2892459.243423  time= 4.19859    cumTime= 50.6786    edges= 24130    schur= 1    lambda= 255.126158    levenbergIter= 1
iteration= 12   chi2= 2885617.324537  time= 4.20777    cumTime= 54.8864    edges= 24130    schur= 1    lambda= 170.084105    levenbergIter= 1
iteration= 13   chi2= 2882725.432777  time= 4.19857    cumTime= 59.0849    edges= 24130    schur= 1    lambda= 113.389404    levenbergIter= 1
iteration= 14   chi2= 2881379.575439  time= 4.20091    cumTime= 63.2859    edges= 24130    schur= 1    lambda= 75.592936     levenbergIter= 1
iteration= 15   chi2= 2879017.915669  time= 4.20232    cumTime= 67.4882    edges= 24130    schur= 1    lambda= 50.395291     levenbergIter= 1
iteration= 16   chi2= 2874343.612996  time= 4.19978    cumTime= 71.688     edges= 24130    schur= 1    lambda= 33.596860     levenbergIter= 1
iteration= 17   chi2= 2873961.028388  time= 5.20803    cumTime= 76.896     edges= 24130    schur= 1    lambda= 179.183255    levenbergIter= 3
iteration= 18   chi2= 2868564.405924  time= 4.19375    cumTime= 81.0897    edges= 24130    schur= 1    lambda= 119.455503    levenbergIter= 1
iteration= 19   chi2= 2863855.640269  time= 5.70658    cumTime= 86.7963    edges= 24130    schur= 1    lambda= 5096.768146   levenbergIter= 4
iteration= 20   chi2= 2861508.683252  time= 4.19212    cumTime= 90.9884    edges= 24130    schur= 1    lambda= 3397.845430   levenbergIter= 1
iteration= 21   chi2= 2859040.711592  time= 5.20896    cumTime= 96.1974    edges= 24130    schur= 1    lambda= 18121.842295  levenbergIter= 3
iteration= 22   chi2= 2854295.729088  time= 5.20018    cumTime= 101.398    edges= 24130    schur= 1    lambda= 96649.825575  levenbergIter= 3
iteration= 23   chi2= 2849673.207300  time= 4.19318    cumTime= 105.591    edges= 24130    schur= 1    lambda= 64433.217050  levenbergIter= 1
iteration= 24   chi2= 2845469.261514  time= 5.24006    cumTime= 110.831    edges= 24130    schur= 1    lambda= 343643.824268 levenbergIter= 3
iteration= 25   chi2= 2844186.461191  time= 4.27093    cumTime= 115.102    edges= 24130    schur= 1    lambda= 229095.882845 levenbergIter= 1
iteration= 26   chi2= 2841616.590509  time= 5.26775    cumTime= 120.369    edges= 24130    schur= 1    lambda= 1221844.708508          levenbergIter= 3
iteration= 27   chi2= 2841218.573518  time= 4.29292    cumTime= 124.662    edges= 24130    schur= 1    lambda= 814563.139006 levenbergIter= 1
iteration= 28   chi2= 2841046.206970  time= 4.71636    cumTime= 129.379    edges= 24130    schur= 1    lambda= 1086084.185341          levenbergIter= 2
iteration= 29   chi2= 2840931.297194  time= 4.25347    cumTime= 133.632    edges= 24130    schur= 1    lambda= 724056.123560 levenbergIter= 1
iteration= 30   chi2= 2840751.527782  time= 4.74213    cumTime= 138.374    edges= 24130    schur= 1    lambda= 965408.164747 levenbergIter= 2
iteration= 31   chi2= 2840189.463518  time= 4.78918    cumTime= 143.164    edges= 24130    schur= 1    lambda= 1287210.886330          levenbergIter= 2
iteration= 32   chi2= 2839953.853373  time= 4.74442    cumTime= 147.908    edges= 24130    schur= 1    lambda= 1716281.181773          levenbergIter= 2
iteration= 33   chi2= 2839180.472603  time= 5.21928    cumTime= 153.127    edges= 24130    schur= 1    lambda= 9153499.636123          levenbergIter= 3
iteration= 34   chi2= 2839141.636368  time= 5.22347    cumTime= 158.351    edges= 24130    schur= 1    lambda= 48818664.725988         levenbergIter= 3
iteration= 35   chi2= 2839111.850547  time= 4.197      cumTime= 162.548    edges= 24130    schur= 1    lambda= 32545776.483992         levenbergIter= 1
iteration= 36   chi2= 2839106.386823  time= 5.19789    cumTime= 167.746    edges= 24130    schur= 1    lambda= 173577474.581292        levenbergIter= 3
iteration= 37   chi2= 2839100.221774  time= 4.19661    cumTime= 171.942    edges= 24130    schur= 1    lambda= 115718316.387528        levenbergIter= 1
iteration= 38   chi2= 2839099.341928  time= 5.19912    cumTime= 177.141    edges= 24130    schur= 1    lambda= 617164354.066815        levenbergIter= 3
iteration= 39   chi2= 2839099.174836  time= 5.20439    cumTime= 182.346    edges= 24130    schur= 1    lambda= 3291543221.689680       levenbergIter= 3
iteration= 40   chi2= 2839099.106276  time= 5.19579    cumTime= 187.542    edges= 24130    schur= 1    lambda= 17554897182.344959      levenbergIter= 3
iteration= 41   chi2= 2839099.064060  time= 4.19674    cumTime= 191.738    edges= 24130    schur= 1    lambda= 11703264788.229973      levenbergIter= 1
iteration= 42   chi2= 2839099.025164  time= 4.1927     cumTime= 195.931    edges= 24130    schur= 1    lambda= 7802176525.486649       levenbergIter= 1
iteration= 43   chi2= 2839099.003658  time= 5.70125    cumTime= 201.632    edges= 24130    schur= 1    lambda= 166446432543.715149     levenbergIter= 4
iteration= 44   chi2= 2839098.964616  time= 4.20133    cumTime= 205.834    edges= 24130    schur= 1    lambda= 55482144181.238380      levenbergIter= 1
iteration= 45   chi2= 2839098.952702  time= 4.19609    cumTime= 210.03     edges= 24130    schur= 1    lambda= 36988096120.825584      levenbergIter= 1
iteration= 46   chi2= 2839098.949528  time= 6.20307    cumTime= 216.233    edges= 24130    schur= 1    lambda= 12625270142575.132812   levenbergIter= 5
iteration= 47   chi2= 2839098.948942  time= 5.21211    cumTime= 221.445    edges= 24130    schur= 1    lambda= 67334774093734.039062   levenbergIter= 3
iteration= 48   chi2= 2839098.948694  time= 4.69796    cumTime= 226.143    edges= 24130    schur= 1    lambda= 89779698791645.375000   levenbergIter= 2
iteration= 49   chi2= 2839098.947548  time= 5.20648    cumTime= 231.349    edges= 24130    schur= 1    lambda= 239412530111054.312500  levenbergIter= 3
iteration= 50   chi2= 2839098.947548  time= 5.19815    cumTime= 236.547    edges= 24130    schur= 1    lambda= 1532240192710747 6.000000 levenbergIter= 3
/home/vance/slam_ws/slambook/class/L7/code/build/directBA exited with code 0
```

图 5 光流法 BA 优化过程截图