

一、Bundle Adjustment

1. 文献阅读

阅读Bill Triggs经典论文Bundle Adjustment: A Modern Synthesis, 了解BA的发展历史, 回答下列问题

1. 为何说BA is slow 是不对的?

因为没有考虑H矩阵的稀疏性

2. BA中有那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?

3D路标点 + pose表示 (旋转与平移)

point可以用齐次和非其次表示

pose: 欧拉角, 四元数, 旋转矩阵, 李代数

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。

根据H的稀疏性可以实现实时BA,比如07年的PTAM上实现, 在现如今, 已经是SLAM必备了。

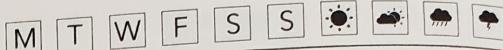
2. BAL-dataset

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。

提示: 注意BAL的投影模型比教材中介绍的多了负号。

分析:

我这里把BA投影模型的全部过程都复习了一遍, 如果只看本题相关的雅克比矩阵部分, 直接看最后三页就好! ! !



MINUTES 會議記錄

主持人
Compere日期
Date時間
Time出席者
Member地址
Place主題
Subject

名稱推導：(基於重投影誤差的BA)

* 第一塊：

投影模型： $P \rightarrow u, v$ 1. 使用相機外參 R, t 。

$$P' = R_{cw}P_w + t_{new} = [x', y', z']^T$$

2. 將 P' 投至歸一化平面：

$$P' = [x'/z, y'/z, 1]^T = [u_o, v_o, 1]^T$$

3. 去畸變：(僅徑向畸變)

$$\begin{cases} u' = u_o (1 + k_1 r_o^2 + k_2 r_o^4) \\ v' = v_o (1 + k_1 r_o^2 + k_2 r_o^4) \end{cases}$$

4. 根據內參 求像素坐標。

$$\begin{cases} u = f_x u' + c_x \\ v = f_y v' + c_y \end{cases} \quad (f_x \text{ 和 } f_y \text{ 通常相似})$$

MINUTES 會議記錄

主持人 Compere	日期 Date	時間 Time
地址 Place		
出席者 Member		

M T W F S S ☀️ 🌧️ 🌩️ 🌡️

* 第二块：
整个BA在做什么？
目的是将给的一组 Pose 和一组 Landmark (point) 整体
进行调优，输出优化后的 Camera_Pose 和 landmark.
利用观测量值 (轨迹) (map)

我这里再说得简单一些，观测数据中是带误差的。本来 Landmark: P 通过投影模型与观测应该一致，因为各种噪声的影响，会导致存在误差 ϵ ：

$E = \epsilon - h(\xi, p)$

该公式表示的是外参对对应的李代数相机

$h(\xi, p)$ 就是第一块中阐述的投影模型
考虑所有误差的时候：

$$\frac{1}{2} \sum_{j=1}^n \|E_{ij}\|^2 = \frac{1}{2} \sum_{j=1}^n \|t_j - h(\xi_j, p_j)\|^2 \quad (1)$$

相机下标 观测到的路标下标

MINUTES 會議記錄												
主持人 Compere	日期 Date	時間 Time	M	T	W	F	S	S				
出席者 Member	地址 Place	主題 Subject										
<p>★ 第三块：求解：</p> <p>通过①和②我们可以推导出目标函数(1)， 回忆之前讲到的非线性优化的方法，#GN, LM 等， 都是需要我们 通过梯度的表示公式，然后逐步迭代优 化，生成对应的自变量增量 ΔX。</p> <p>在这里，我们将 GN 或 LM 展开详细介绍一下如何迭代进行优化 的：</p> <p><因为有李代数在其中表示旋转和平移，所以想通过解析 形式求导不现实></p> <p>① 将 $f(x)$ 在 x 附近泰勒展开</p> <p style="text-align: center;">↑</p> <p>初始值</p> $\ f(x) + \Delta x\ _2^2 = \ f(x)\ _2^2 + J(x) \Delta x + \frac{1}{2} \Delta x^T H \Delta x.$ <p style="text-align: center;">↗ ↘</p> <p>雅可比 赫塞</p>												

MINUTES 賽前會

主持人 Compere

日期 Date _____

時間 Time _____

地址 Place _____

主題 Subject _____

出席者 Member _____

A. 若不保留一阶项 = $\Delta x^* = -J^T(x)$. 一阶梯度下降.

直观意思就是沿仅向梯度前进即可.

B. 若保留二阶项: 牛顿法.

为了避免 H 矩阵的计算, 引入两种更加实用的方法?

A. 高斯牛顿法.

$\Delta x^* = \arg \min_{\Delta x} \frac{1}{2} \|f(x) + J(x)\Delta x\|^2$.

这里 $f(x)$ 是 J 关于 x 的导数.

我的理解: 在小 $f(x)$ 上就做展开, 而不是形成整体的 $J(x)$ 捆勒.

对上式求导(对 Δx 求导).

$$\begin{aligned} \frac{1}{2} \|f(x) + J\Delta x\|^2 &= \frac{1}{2} ((f(x) + J\Delta x)^T \cdot (f(x) + J\Delta x)) \\ &= \frac{1}{2} (\|f(x)\|^2 + 2f(x)^T J \Delta x + \Delta x^T J^T J \Delta x) \\ \Rightarrow J^T J \Delta x &= -J^T f(x) \\ \Rightarrow \Delta x &= \frac{(J^T J)^{-1} (-J^T f(x))}{H} \end{aligned}$$

M T W Th F S

MINUTES 會議記錄

主持人 Compere

日期 Date

地址 Place

時間 Time

出席者 Member

主題 Subject

B. LM

添加 P 做信赖区城，
 $P = \frac{f(x+\Delta x) - f(x)}{\| \Delta x \|}$: 实际下降值的值
 $J(x), \Delta x$: 近似模型下降的值

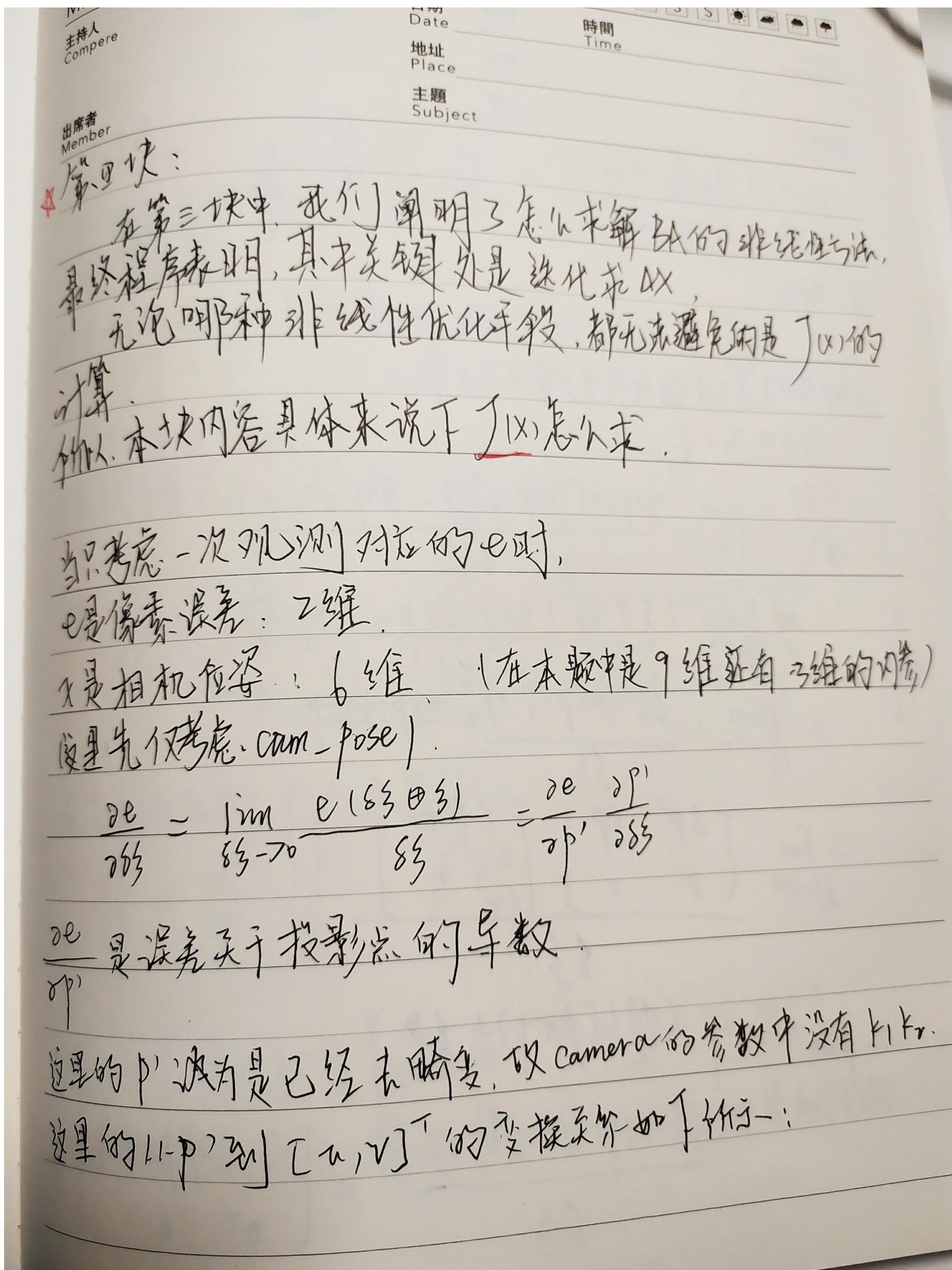
ρ 越靠近 1，说明近似情况越好。
 求的是 $\min_{\Delta x} \frac{1}{2} \| f(x_k) + J(x_k) \Delta x_k \|^2 + \frac{\lambda}{2} \| \Delta x \|^2$

提示：引入 L 以是因为 LN 中的 $H = J^T J$ 的半正定性，存在不稳定性。

最终是在求 $(H + \lambda I)^T \Delta x = g$ 。

三步二

总结，整体的迭代式求解是一步步来做的，首次求出一个当前状态的最小的 Δx ，然后将 Δx 代入到原本 x 中，再去迭代求，直到 Δx 足够小，退出循环。



MINUTES 會議記錄		日期 Date	時間 Time
主持人 Compere		地址 Place	
出席者 Member		主題 Subject	

$\begin{cases} u = f_x \frac{x}{\tau^1} + c_x \\ v = f_y \frac{y}{\tau^1} + c_y \end{cases} \Rightarrow \frac{\partial p}{\partial \theta} = \begin{pmatrix} \frac{f_x}{\tau^1} & 0 & -\frac{f_x x^1}{\tau^{12}} \\ 0 & \frac{f_y}{\tau^1} & -\frac{f_y y^1}{\tau^{12}} \end{pmatrix}$

第二部分：
 $\frac{\partial p}{\partial \theta}$ 表示相机坐标系下的 P' 点关于运动参数的导数。
 这部分的详细推导可见第 4.3.5 节。
 这里也写一下：

$\frac{\partial \frac{\partial p}{\partial \theta}}{\partial \theta} = \lim_{\delta \theta \rightarrow 0} \frac{\exp(\delta \theta^1) \exp(\delta \theta^2) p - \exp(\delta \theta^1) p}{\delta \theta}$

$= \lim_{\delta \theta \rightarrow 0} \frac{(I + \delta \theta^1) \exp(\delta \theta^2) p - \exp(\delta \theta^2) p}{\delta \theta}$

$= \lim_{\delta \theta \rightarrow 0} \frac{\delta \theta^1 \exp(\delta \theta^2) p - \exp(\delta \theta^2) p}{\delta \theta}$

$= \lim_{\delta \theta \rightarrow 0} \frac{\delta \theta^1 \left[\begin{matrix} \delta \theta^1 & \delta p \\ 0 & 0 \end{matrix} \right] \left[\begin{matrix} R^1 p + t^1 \\ 1 \end{matrix} \right]}{\delta \theta}$

$= \lim_{\delta \theta \rightarrow 0} \frac{\left[\begin{matrix} \delta \theta^1 (R^1 p + t^1) + \delta p \\ 0 \end{matrix} \right]}{\delta \theta} = \left[\begin{matrix} I - (R^1 p + t^1)^1 \\ 0^T \end{matrix} \right]$

MINUTES

主持人 Comperer	日期 Date	时间 Time						
			<input type="checkbox"/>					

出席者
Member

时-数-
每日非零次前三行组成 $[I, -P^A]$

写出来就是：

$$\begin{bmatrix} 1 & 0 & 0 & \cancel{0} & -x' & y' \\ 0 & 1 & 0 & \cancel{x'} & 0 & -x \\ 0 & 0 & 1 & -y' & x' & 0 \end{bmatrix}$$

根据链式求导法 将 $\frac{\partial e}{\partial p^1}, \frac{\partial P^1}{\partial \xi} = \frac{\partial e}{\partial \xi}$

$$\frac{\partial e}{\partial \xi} = \begin{bmatrix} \frac{f_x}{\xi^1} & 0 & -\frac{f_{xx}x'}{\xi^{12}} - \frac{f_{xx}x'y'}{\xi^{12}} + f_{xx}\left(\frac{f_x x'^2}{\xi^{12}}\right) - \frac{f_x y'}{\xi^1} \\ 0 & \frac{f_y}{\xi^1} & -\frac{f_{yy}y'}{\xi^{12}} - (f_y + \frac{f_{yy}y'^2}{\xi^{12}}) + \frac{f_{yy}x'}{\xi^{12}} + \frac{f_{yx}x'}{\xi^1} \end{bmatrix}$$

若有单矩阵中的求导

$$\frac{\partial e}{\partial p} = \frac{\partial e}{\partial p^1} \frac{\partial p^1}{\partial p} = R \frac{\partial e}{\partial p^1}$$

注意：以上推导都是 $error = [预测] - [观测]$

若相反的话，我们将两个矩阵可以添加负号就可。

MINUTES 會議記錄

M T W F S S

時間
Time日期
Date地址
Place主題
Subject主持人
Compere出席者
Member

第五块：BA求解：

在整体BA目标函数上，我们将 x 定义成所有待优化变量：

$$x = [\underbrace{s_1 \dots s_m}_x, \underbrace{p_1 \dots p_n}_p]^T$$

此时，目标函数可以简化成：

$$\frac{1}{2} \|f(x) + \Delta x\|^2 = \frac{1}{2} \|e + J^T x + E \Delta x_p\|^2$$

所有的 J^T 的拼接。
 $2 \times 3n$

 $2 \times 6m$

根据GN或LM：目标函数最终求的是：

$$H \Delta x = g \quad : \quad \left\{ \begin{array}{l} H = J^T J \quad (\text{高斯牛顿}) \\ \end{array} \right.$$

$$H = J^T J + \lambda I \quad (LM)$$

$$J = [F \quad E] \Rightarrow H = \begin{bmatrix} F^T F & F^T E \\ E^T F & E^T E \end{bmatrix}$$

MINUTES 會議記錄

日期 Date M T W F S S
地址 Place 時間 Time

主題 Subject

出席者 Member

第 1 單元：本題求解：

推導：怎樣將帶內參的 J_f 形式推導出來？

回答：

由數據集中有負号（相機到 normalized 坐標系）
所以误差模型推導出來的結果如下：

$$\begin{cases} ex = u + \frac{f \cdot x_c}{r_c} (1 + k_1 r^2 + k_2 r^4) + c_x \\ ey = v + \frac{f \cdot y_c}{r_c} (1 + k_1 r^2 + k_2 r^4) + c_y \end{cases}$$

其中： $r^2 = \sqrt{x_c^2 + y_c^2}$

2. J_f 形式：對相機參數的轉換可以

我們的 cam 對單個的一條邊 error 的是 2×9 , 具體形式
如下：

$$J_f = [J_{83}^{2 \times 6}, J_f^{2 \times 1}, J_{k1}^{2 \times 1}, J_{k2}^{2 \times 1}]$$

我們的 point 的是 2×3 , 形式和之前無差別。
因為有變形參數和焦距參數的引入，導致 $J_{83}^{2 \times 6}$ 和前文中
的 $J_{2 \times 6}$ 稍有不同，具體描述如下：

M	T	W	F	S	S
---	---	---	---	---	---

MINUTES 會議記錄

主持人
Compere日期
Date時間
Time出席者
Member地址
Place主題
Subject2.1 J_{2x6}

根据J中的目标函数：

$$\frac{\partial ex}{\partial x_0} = \frac{f}{T_0} (1 + k_1 r^2 + k_2 r^4) + \frac{2fx_0}{T_0^3} (k_1 + 2k_2 r^2)$$

$$\frac{\partial ex}{\partial y_0} = \frac{2fy_0r}{T_0^3} (k_1 + 2k_2 r^2)$$

$$\frac{\partial ex}{\partial T_0} = -\frac{fx_0}{T_0^2} (1 + k_1 r^2 + k_2 r^4) - \frac{2fx_0r^2}{T_0^2} (k_1 + 2k_2 r^2)$$

$$\frac{\partial ey}{\partial x_0} = \frac{2fy_0x_0}{T_0^3} (k_1 + 2k_2 r^2)$$

$$\frac{\partial ey}{\partial y_0} = \frac{f}{T_0} (1 + k_1 r^2 + k_2 r^4) + \frac{2fy_0^2}{T_0^3} (k_1 + 2k_2 r^2)$$

$$\frac{\partial ey}{\partial T_0} = -\frac{fy_0}{T_0^2} (1 + k_1 r^2 + k_2 r^4) - \frac{2fy_0r^2}{T_0^2} (k_1 + 2k_2 r^2)$$

以上求出 $\frac{\partial e}{\partial \{}} = \frac{\partial e}{\partial p_c} \cdot \frac{\partial p_c}{\partial \{}}$ 的第一项.

第二项形式设为, 和 J_{2x6} 中保持一致 $[1 - p_i^{\wedge}]$

MINUTES 會議記錄

主持人 Compere

出席者 Member

日期 Date

時間 Time

地址 Place

主題 Subject

$\frac{\partial e}{\partial f} = \left[\begin{array}{c} \frac{x_c}{f_c} (1 + k_1 r^2 + k_2 r^4) \\ \frac{y_c}{f_c} (1 + k_1 r^2 + k_2 r^4) \end{array} \right]$

$\frac{\partial e}{\partial k_1} = \left[\begin{array}{c} \frac{x_c}{f_c} \cdot r^2 \\ \frac{y_c}{f_c} \cdot r^2 \end{array} \right]$

$\frac{\partial e}{\partial k_2} = \left[\begin{array}{c} \frac{f_x c}{f_c} \cdot r^4 \\ \frac{f_y c}{f_c} \cdot r^4 \end{array} \right]$

3. J_E 、对路标的求导雅可比

$J_E^{(2x)} = \frac{\partial e}{\partial p_c} \cdot \frac{\partial p_c}{\partial p_w} = \frac{\partial e}{\partial p_w} \cdot R$ (该形式与原来一致, 但是 $\frac{\partial e}{\partial p_w}$ 有内容变化)

代码部分：（这里仅展示雅克比求导矩阵部分）（完整项目见附件）

```

virtual void linearizeOplus() override {
    VertexCameraBAL *cam = static_cast<VertexCameraBAL *>( vertex(0));
    VertexPointBAL *point = static_cast<VertexPointBAL *>( vertex(1));

    Vector3d Pc;
    Pc = cam->estimate()._SE3 * point->estimate();
    // 相机坐标系
    double xc = Pc[0];
    double yc = Pc[1];
    double zc = Pc[2];
    // 归一化坐标系
    double xc1 = -xc / zc;
    double yc1 = -yc / zc;
    double zc1 = -1;

    // Apply second and fourth order radial distortion
    const double &k1 = cam->estimate()._k1;
    const double &k2 = cam->estimate()._k2;

    double r2 = xc1 * xc1 + yc1 * yc1;
    double distort = double(1.0) + k1 * r2 + k2 * r2 * r2;

    const double &f = cam->estimate()._f;

    Matrix<double, 2, 6> J_e_kesi;
    Matrix<double, 2, 3> J_e_pc;
    Matrix<double, 3, 6> J_pc_kesi;// = new Eigen::Matrix<double,3,6>::Zero();
    Matrix<double, 2, 1> J_e_f;
    Matrix<double, 2, 2> J_e_k;
    Matrix3d pc_hat;

    double zc_2 = zc * zc;
    double zc_3 = zc_2 * zc;
    double d2 = k1 + 2 * k2 * r2;

    J_e_pc(0, 0) = f / zc * distort + 2 * f * xc * xc / zc_3 * d2;
    J_e_pc(0, 1) = 2 * f * xc * yc / zc_3 * d2;
    J_e_pc(0, 2) = -f * xc / zc_2 * distort - 2 * f * xc * r2 / zc_2 * d2;
    J_e_pc(1, 0) = 2 * f * xc * yc / zc_3 * d2;
    J_e_pc(1, 1) = f / zc * distort + 2 * f * yc * yc / zc_3 * d2;
    J_e_pc(1, 2) = -f * yc / zc_2 * distort - 2 * f * yc * r2 / zc_2 * d2;

    pc_hat << 0, zc, -yc,
              -zc, 0, xc,
              yc, -xc, 0;
    J_pc_kesi.block(0,0,3,3) = Matrix3d::Identity();
    J_pc_kesi.block(0,3,3,3) = pc_hat;

    J_e_kesi = J_e_pc * J_pc_kesi;

    J_e_f(0, 0) = xc / zc * distort;
    J_e_f(1, 0) = yc / zc * distort;

    J_e_k(0, 0) = f * xc * r2 / zc;
    J_e_k(0, 1) = f * xc * r2 * r2 / zc;

    J_e_k(1, 0) = f * yc * r2 / zc;
    J_e_k(1, 1) = f * yc * r2 * r2 / zc;

    _jacobianOplusXi.block(0, 0, 2, 6) = J_e_kesi;
    _jacobianOplusXi.block(0, 6, 2, 1) = J_e_f;
}

```

```

_jacobianOplusXi.block(0, 7, 2, 2) = J_e_k;

Matrix<double, 2, 3> J_e_pw;
J_e_pw = J_e_pc * cam->estimate().SE3.rotation_matrix();
_jacobianOplusXj = J_e_pw;
}

```

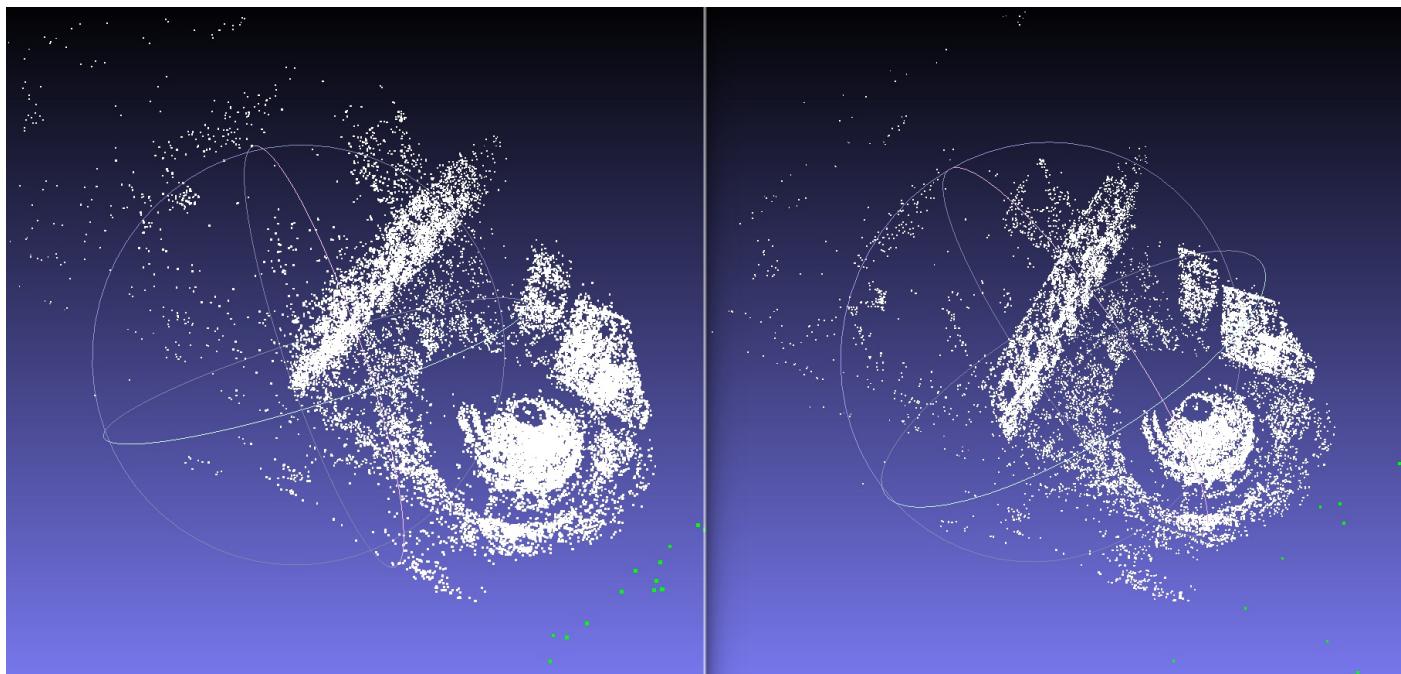
结果验证：

```

iteration= 0    chi2= 447035230.450231  time= 0.168474  cumTime= 0.168474  edges= 83718  schur= 1  lambda= 436534.999128  levenbergIter= 1
iteration= 1    chi2= 258852450.802893  time= 0.129436  cumTime= 0.297911  edges= 83718  schur= 1  lambda= 145511.666376  levenbergIter= 1
iteration= 2    chi2= 174964328.713253  time= 0.115251  cumTime= 0.413162  edges= 83718  schur= 1  lambda= 48503.888792  levenbergIter= 1
iteration= 3    chi2= 85706104.448015  time= 0.10933  cumTime= 0.513492  edges= 83718  schur= 1  lambda= 16167.962931  levenbergIter= 1
iteration= 4    chi2= 27668273.069986  time= 0.105841  cumTime= 0.619333  edges= 83718  schur= 1  lambda= 5389.320977  levenbergIter= 1
iteration= 5    chi2= 6980791.094378  time= 0.0985411  cumTime= 0.717874  edges= 83718  schur= 1  lambda= 1796.440326  levenbergIter= 1
iteration= 6    chi2= 1757637.732163  time= 0.105289  cumTime= 0.823163  edges= 83718  schur= 1  lambda= 598.813442  levenbergIter= 1
iteration= 7    chi2= 1032066.143545  time= 0.102487  cumTime= 0.92565  edges= 83718  schur= 1  lambda= 199.604481  levenbergIter= 1
iteration= 8    chi2= 801661.446760  time= 0.0978172  cumTime= 1.02347  edges= 83718  schur= 1  lambda= 66.534827  levenbergIter= 1
iteration= 9    chi2= 671182.111648  time= 0.0974594  cumTime= 1.12093  edges= 83718  schur= 1  lambda= 22.178276  levenbergIter= 1
iteration= 10   chi2= 563684.093964  time= 0.101104  cumTime= 1.22203  edges= 83718  schur= 1  lambda= 7.392759  levenbergIter= 1
iteration= 11   chi2= 443413.608686  time= 0.0937824  cumTime= 1.31581  edges= 83718  schur= 1  lambda= 2.464253  levenbergIter= 1
iteration= 12   chi2= 323277.686779  time= 0.08827  cumTime= 1.40408  edges= 83718  schur= 1  lambda= 0.821418  levenbergIter= 1
iteration= 13   chi2= 215440.417907  time= 0.104391  cumTime= 1.50847  edges= 83718  schur= 1  lambda= 0.273806  levenbergIter= 1
iteration= 14   chi2= 137747.094196  time= 0.107898  cumTime= 1.61637  edges= 83718  schur= 1  lambda= 0.106475  levenbergIter= 1
iteration= 15   chi2= 98662.715031  time= 0.10513  cumTime= 1.7215  edges= 83718  schur= 1  lambda= 0.070984  levenbergIter= 1
iteration= 16   chi2= 77524.649216  time= 0.100995  cumTime= 1.8225  edges= 83718  schur= 1  lambda= 0.047322  levenbergIter= 1
iteration= 17   chi2= 66876.459971  time= 0.0955265  cumTime= 1.91802  edges= 83718  schur= 1  lambda= 0.031548  levenbergIter= 1
iteration= 18   chi2= 59480.293276  time= 0.0893905  cumTime= 2.00741  edges= 83718  schur= 1  lambda= 0.021032  levenbergIter= 1
iteration= 19   chi2= 51254.132727  time= 0.0904226  cumTime= 2.09784  edges= 83718  schur= 1  lambda= 0.014021  levenbergIter= 1
optimization complete..

```

这里还是使用《十四讲》上的BAL数据集，因为其他数据集输出效果不理想，图都挤在一起，太大的数据集干脆无法显示。



二、直接法的BA

1. 数学模型

特征点法的BA以最小化重投影误差为目标，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，谈到如何用直接法去估计相机位姿。但是直接法也可以处理整个BA。

下面，请你推到直接法BA的数学模型，完成g2o的实现。

注意：我们用x,y,z参数化每个3D点，而实际的直接法多采用逆深度参数化。使用逆深度会有一种类似归一化的效果，防

止因为距离太远，导致对其他参数不敏感。

本题给定7张照片，每张图片对应相机位姿初始值为 T_i ，以 T_{cw} 存储在poses.txt文件中，其中每一行代表一个相机位姿，格式同之前一致。同时，还有一个3D点集 P ，共 N 个点。其中每个点的初始坐标基座 p_i 。每个带你都有自己的固定灰度值，16个该点周围 4×4 小块读数表示，顺序是列优先。

我们知道，可以把每个点投影到每个图像上，然后再看投影后点周围的小块与原始的 4×4 小块的差别。那么，整体优化目标函数如下：

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

也就是最小化任意点在任意图像中投影与其本身颜色之差。其中 K 是相机内参， p_i 是投影函数， W 是整个patch。下面请回答：

1. 如何描述任意一点投影在任意一张图像中的error？
2. 每个error关联几个优化变量？
3. error关于各变量的雅克比是什么？

图片上的2,3,4对应上述问题做了回答

分析：

1. 本题和 ch6 中的直接法的区别？

第 6 节课后题的直接法属于前端 VO 范畴，它是以第一张图像做 ref，其余 5 张图像做 current，直接估计其余 5 张 picture 的 T 。这只是小范围的移动中对相机的粗略位姿估计。

本次作业，其实是后端范畴，我们可以将阳包集当做前端 VO 粗略估计输出结果。统一对全部的 pose 和 point 做 BA，使得光度误差最小化。

2. 该 BA 中的误差项是什么？

我们单独看一个 point 和一幅图像的时候，error 就是光度误差，将 point 投影到对应图像上，取 4×4 patch，用 ~~error~~ 工具做差。

数学表示如下：

$$\sum_w \| I(p_i) - I_j(w | KT_j p_i) \|^2$$

其中 p_i 表示第 i 个 point， T_j 是图像位姿， $T_j p_i$ 表示 j 图下 p_i 的三维坐标， w 表示 4×4 patch。

3. 每个 error 关联几个优化变量？

两部分，一部分是 $p_i(3 \times 1)$ ，一部分是 T_j ，用李代数表示的话就是 6×1 的向量。

4. error 关于各变量的雅可比是什么？

~~斜率是~~ 首先 e 是标量，故最终的 J 的维度是 1×9 .

其中 $\frac{\partial e}{\partial x}$ 是 $\frac{\partial J}{\partial x}$. 1×3 是 $\frac{\partial J}{\partial w}$. $\frac{\partial J}{\partial b}$

$\left. \begin{array}{l} \text{针对一次误差} \\ \text{全} \end{array} \right\}$

$\frac{\partial J}{\partial x}$ 的形式：可以根据《机器学习》P195页

中的阐述写成如下形式：

$$\frac{\partial J}{\partial x} = -\frac{\partial J}{\partial u} \begin{vmatrix} \frac{\partial u}{\partial q} & \frac{\partial q}{\partial x} \end{vmatrix}$$

↓ ↓ ↓
n处的像素梯度

1×2

老生常谈的 2×6 矩阵.

$$\frac{\partial J}{\partial w} = -\frac{\partial J}{\partial u} \begin{vmatrix} \frac{\partial J}{\partial u} & \frac{\partial u}{\partial q} & \frac{\partial q}{\partial w} \end{vmatrix}$$

↓
前两项与 $\frac{\partial J}{\partial x}$ 相同

→ 这部分很简单，就是 R (旋转矩阵).

以上就是 error 关于各变量的雅可比形式。

2. 实现

根据上面说明，使用g2o实现上述优化。用pangolin绘制优化结果。

```

//  

// Created by xuzhi.  

// this program shows how to perform direct bundle adjustment  

//  

#include <iostream>  
  

using namespace std;  
  

#include <g2o/core/base UnaryEdge.h>  

#include <g2o/core/base BinaryEdge.h>  

#include <g2o/core/base Vertex.h>  

#include <g2o/core/block_solver.h>  

#include <g2o/core/optimization_algorithm_levenberg.h>  

#include <g2o/solvers/dense/linear_solver_dense.h>  

#include <g2o/core/robust_kernel.h>  

#include <g2o/core/robust_kernel_impl.h>  

#include <g2o/types/sba/types_six_dof_expmap.h>  
  

#include <Eigen/Core>  

#include <sophus/se3.h>  

#include <opencv2/opencv.hpp>  
  

#include <pangolin/pangolin.h>  

#include <boost/format.hpp>  
  

#define ROBUST true  

using namespace Sophus;  

using namespace pangolin;  

using namespace g2o;  
  

typedef vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> VecSE3;  

typedef vector<Eigen::Vector3d, Eigen::aligned_allocator<Eigen::Vector3d>> VecVec3d;  
  

// global variables  

// Tcw形式存储相机pose  

string pose_file = "../poses.txt";  

// 每行表示一个p, 前三维表示初始坐标x,y,z, 后面的16维表示周围的patch灰度值  

string points_file = "../points.txt";  
  

// intrinsics  

float fx = 277.34;  

float fy = 291.402;  

float cx = 312.234;  

float cy = 239.777;  
  

// bilinear interpolation  

// 返回值是浮点类型  

inline float GetPixelValue(const cv::Mat &img, float x, float y) {  

    uchar *data = &img.data[int(y) * img.step + int(x)];  

    float xx = x - floor(x);  

    float yy = y - floor(y);  

    return float(  

        (1 - xx) * (1 - yy) * data[0] +  

        xx * (1 - yy) * data[1] +  

        (1 - xx) * yy * data[img.step] +  

        xx * yy * data[img.step + 1]  

    );  

}
}

```

```

// g2o vertex that use sophus::SE3 as pose
class VertexSophus : public g2o::BaseVertex<6, Sophus::SE3> {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    VertexSophus() {}

    ~VertexSophus() {}

    bool read(std::istream &is) {}

    bool write(std::ostream &os) const {}

    virtual void setToOriginImpl() {
        _estimate = Sophus::SE3();
    }

    virtual void oplusImpl(const double *update_) {
        Eigen::Map<const Eigen::Matrix<double, 6, 1>> update(update_);
        setEstimate(Sophus::exp(update_) * estimate());
    }
};

// TODO edge of projection error, implement it
// 16x1 error, which is the errors in patch
typedef Eigen::Matrix<double, 16, 1> Vector16d;
class EdgeDirectProjection : public g2o::BaseBinaryEdge<16, Vector16d,
g2o::VertexSBAPointXYZ, VertexSophus> {
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;

    EdgeDirectProjection(float *color, cv::Mat &target) {
        this->origColor = color;
        this->targetImg = target;
        this->w = targetImg.cols;
        this->h = targetImg.rows;
    }

    ~EdgeDirectProjection() {}

    virtual void computeError() override {
        // TODO START YOUR CODE HERE
        // compute projection error ...
        const VertexSBAPointXYZ *vertexPw = static_cast<const VertexSBAPointXYZ * >(vertex(0));
        const VertexSophus *vertexTcw = static_cast<const VertexSophus * >(vertex(1));
        Vector3d Pc = vertexTcw->estimate() * vertexPw->estimate();
        float u = Pc[0] / Pc[2] * fx + cx;
        float v = Pc[1] / Pc[2] * fy + cy;
        if (u - 2 < 0 || v - 2 < 0 || u+1 >= w || v + 1 >= h) {
            this->setLevel(1);
            for (int n = 0; n < 16; n++)
                _error[n] = 0;
        } else {
            for (int i = -2; i < 2; i++) {
                for (int j = -2; j < 2; j++) {
                    int num = 4 * i + j + 10;
                    _error[num] = origColor[num] - GetPixelValue(targetImg, u + i, v + j);
                }
            }
        }
    }
};

```

```

        }
    }
    // END YOUR CODE HERE
}

// Let g2o compute jacobian for you
virtual void linearizeOplus() override {
    if (level() == 1) {
        _jacobianOplusXi = Matrix<double, 16, 3>::Zero();
        _jacobianOplusXj = Matrix<double, 16, 6>::Zero();
        return;
    }
    const VertexSBAPointXYZ *vertexPw = static_cast<const VertexSBAPointXYZ * >(vertex(0));
    const VertexSophus *vertexTcw = static_cast<const VertexSophus * >(vertex(1));
    Vector3d Pc = vertexTcw->estimate() * vertexPw->estimate();
    float x = Pc[0];
    float y = Pc[1];
    float z = Pc[2];
    float inv_z = 1.0 / z;
    float inv_z2 = inv_z * inv_z;
    float u = x * inv_z * fx + cx;
    float v = y * inv_z * fy + cy;

    Matrix<double, 2, 3> J_Puv_Pc;
    J_Puv_Pc(0, 0) = fx * inv_z;
    J_Puv_Pc(0, 1) = 0;
    J_Puv_Pc(0, 2) = -fx * x * inv_z2;
    J_Puv_Pc(1, 0) = 0;
    J_Puv_Pc(1, 1) = fy * inv_z;
    J_Puv_Pc(1, 2) = -fy * y * inv_z2;

    Matrix<double, 3, 6> J_Pc_kesi = Matrix<double, 3, 6>::Zero();
    J_Pc_kesi(0, 0) = 1;
    J_Pc_kesi(0, 4) = z;
    J_Pc_kesi(0, 5) = -y;
    J_Pc_kesi(1, 1) = 1;
    J_Pc_kesi(1, 3) = -z;
    J_Pc_kesi(1, 5) = x;
    J_Pc_kesi(2, 2) = 1;
    J_Pc_kesi(2, 3) = y;
    J_Pc_kesi(2, 4) = -x;

    Matrix<double, 1, 2> J_I_Puv;
    for (int i = -2; i < 2; i++)
        for (int j = -2; j < 2; j++) {
            int num = 4 * i + j + 10;
            J_I_Puv(0, 0) =
                (GetPixelValue(targetImg, u + i + 1, v + j) -
                 GetPixelValue(targetImg, u + i - 1, v + j)) / 2;
            J_I_Puv(0, 1) =
                (GetPixelValue(targetImg, u + i, v + j + 1) -
                 GetPixelValue(targetImg, u + i, v + j - 1)) / 2;
            _jacobianOplusXi.block<1, 3>(num, 0) = -J_I_Puv * J_Puv_Pc *
            vertexTcw->estimate().rotation_matrix();
            _jacobianOplusXj.block<1, 6>(num, 0) = -J_I_Puv * J_Puv_Pc *
            J_Pc_kesi;
        }
    }

    virtual bool read(istream &in) {}
}

```

```

virtual bool write(ostream &out) const {}

private:
    cv::Mat targetImg; // the target image
    float *origColor = nullptr; // 16 floats, the color of this point
    int w;
    int h;
};

// plot the poses and points for you, need pangolin
void Draw(const VecSE3 &poses, const VecVec3d &points);

int main(int argc, char **argv) {
    VecSE3 poses;
    VecVec3d points;
    ifstream fin(pose_file);

    while (!fin.eof()) {
        double timestamp = 0;
        fin >> timestamp;
        if (timestamp == 0) break;
        double data[7];
        for (auto &d: data) fin >> d;
        poses.push_back(Sophus::SE3(
            Eigen::Quaterniond(data[6], data[3], data[4], data[5]),
            Eigen::Vector3d(data[0], data[1], data[2]))
        ));
        if (!fin.good()) break;
    }
    fin.close();

    vector<float *> color;
    fin.open(points_file);
    while (!fin.eof()) {
        double xyz[3] = {0};
        for (int i = 0; i < 3; i++) fin >> xyz[i];
        if (xyz[0] == 0) break;
        points.push_back(Eigen::Vector3d(xyz[0], xyz[1], xyz[2]));
        // 用指针往后递延
        float *c = new float[16];
        for (int i = 0; i < 16; i++) fin >> c[i];
        color.push_back(c);

        if (fin.good() == false) break;
    }
    fin.close();
    cout << "poses: " << poses.size() << ", points: " << points.size() << endl;

    // read images
    vector<cv::Mat> images;
    boost::format fmt("../%d.png");
    for (int i = 0; i < 7; i++) {
        images.push_back(cv::imread((fmt % i).str(), 0));
    }
    cout << "images: " << images.size() << endl;

    // build optimization problem
    typedef g2o::BlockSolver<g2o::BlockSolverTraits<6, 3>> DirectBlock; // 求解的向量
}

```

是6*1的

```

    DirectBlock::LinearSolverType *linearSolver = new
g2o::LinearSolverDense<DirectBlock::PoseMatrixType>();
    DirectBlock *solver_ptr = new DirectBlock(linearSolver);
    g2o::OptimizationAlgorithmLevenberg *solver = new
g2o::OptimizationAlgorithmLevenberg(solver_ptr); // L-M
    g2o::SparseOptimizer optimizer;
    optimizer.setAlgorithm(solver);
    optimizer.setVerbose(true);

    // TODO add vertices, edges into the graph optimizer
    // START YOUR CODE HERE
    for (int i = 0; i < points.size(); i++) {
        VertexSBAPointXYZ *vertexPw = new VertexSBAPointXYZ();
        vertexPw->setEstimate(points[i]);
        vertexPw->setId(i);
        vertexPw->setMarginalized(true);
        optimizer.addVertex(vertexPw);
    }
    for (int j = 0; j < poses.size(); j++) {
        VertexSophus *vertexTcw = new VertexSophus();
        vertexTcw->setEstimate(poses[j]);
        vertexTcw->setId(j + points.size());
        optimizer.addVertex(vertexTcw);
    }

    for (int c = 0; c < poses.size(); c++)
        for (int p = 0; p < points.size(); p++) {
            EdgeDirectProjection *edge = new EdgeDirectProjection(color[p],
images[c]);
            edge->setVertex(0, dynamic_cast<VertexSBAPointXYZ *>
(optimizer.vertex(p)));
            edge->setVertex(1, dynamic_cast<VertexSophus *>(optimizer.vertex(c +
points.size())));
            edge->setInformation(Matrix<double, 16, 16>::Identity());
            RobustKernelHuber *rk = new RobustKernelHuber;
            rk->setDelta(1.0);
            edge->setRobustKernel(rk);
            optimizer.addEdge(edge);
        }

    // END YOUR CODE HERE

    // perform optimization
    optimizer.initializeOptimization(0);
    optimizer.optimize(200);

    // TODO fetch data from the optimizer
    // START YOUR CODE HERE
    for (int c = 0; c < poses.size(); c++)
        for (int p = 0; p < points.size(); p++) {
            Vector3d Pw = dynamic_cast<VertexSBAPointXYZ *>(optimizer.vertex(p))->estimate();
            points[p] = Pw;
            SE3 Tcw = dynamic_cast<VertexSophus *>(optimizer.vertex(c +
points.size()))->estimate();
            poses[c] = Tcw;
        }
    // END YOUR CODE HERE
    // plot the optimized points and poses
    Draw(poses, points);

    // delete color data

```

```

for (auto &c: color) delete[] c;
return 0;
}

void Draw(const VecSE3 &poses, const VecVec3d &points) {
    if (poses.empty() || points.empty()) {
        cerr << "parameter is empty!" << endl;
        return;
    }

    // create pangolin window and plot the trajectory
    pangolin::CreateWindowAndBind("Trajectory Viewer", 1024, 768);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    pangolin::OpenGLRenderState s_cam(
        pangolin::ProjectionMatrix(1024, 768, 500, 500, 512, 389, 0.1, 1000),
        pangolin::ModelViewLookAt(0, -0.1, -1.8, 0, 0, 0, 0.0, -1.0, 0.0)
    );

    pangolin::View &d_cam = pangolin::CreateDisplay()
        .SetBounds(0.0, 1.0, pangolin::Attach::Pix(175), 1.0, -1024.0f / 768.0f)
        .SetHandler(new pangolin::Handler3D(s_cam));

    while (pangolin::ShouldQuit() == false) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        d_cam.Activate(s_cam);
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

        // draw poses
        float sz = 0.1;
        int width = 640, height = 480;
        for (auto &Tcw: poses) {
            glPushMatrix();
            Sophus::Matrix4f m = Tcw.inverse().matrix().cast<float>();
            glMultMatrixf((GLfloat *) m.data());
            glColor3f(1, 0, 0);
            glLineWidth(2);
            glBegin(GL_LINES);
            glVertex3f(0, 0, 0);
            glVertex3f(sz * (0 - cx) / fx, sz * (0 - cy) / fy, sz);
            glVertex3f(0, 0, 0);
            glVertex3f(sz * (0 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(0, 0, 0);
            glVertex3f(sz * (width - 1 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(0, 0, 0);
            glVertex3f(sz * (width - 1 - cx) / fx, sz * (0 - cy) / fy, sz);
            glVertex3f(sz * (width - 1 - cx) / fx, sz * (0 - cy) / fy, sz);
            glVertex3f(sz * (width - 1 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(sz * (width - 1 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(sz * (0 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(sz * (0 - cx) / fx, sz * (height - 1 - cy) / fy, sz);
            glVertex3f(sz * (0 - cx) / fx, sz * (0 - cy) / fy, sz);
            glVertex3f(sz * (0 - cx) / fx, sz * (0 - cy) / fy, sz);
            glEnd();
            glPopMatrix();
        }

        // points
    }
}

```

```

glPointSize(2);
glBegin(GL_POINTS);
for (size_t i = 0; i < points.size(); i++) {
    glColor3f(0.0, points[i][2]/4, 1.0-points[i][2]/4);
    glVertex3d(points[i][0], points[i][1], points[i][2]);
}
glEnd();

pangolin::FinishFrame();
usleep(5000); // sleep 5 ms
}
}

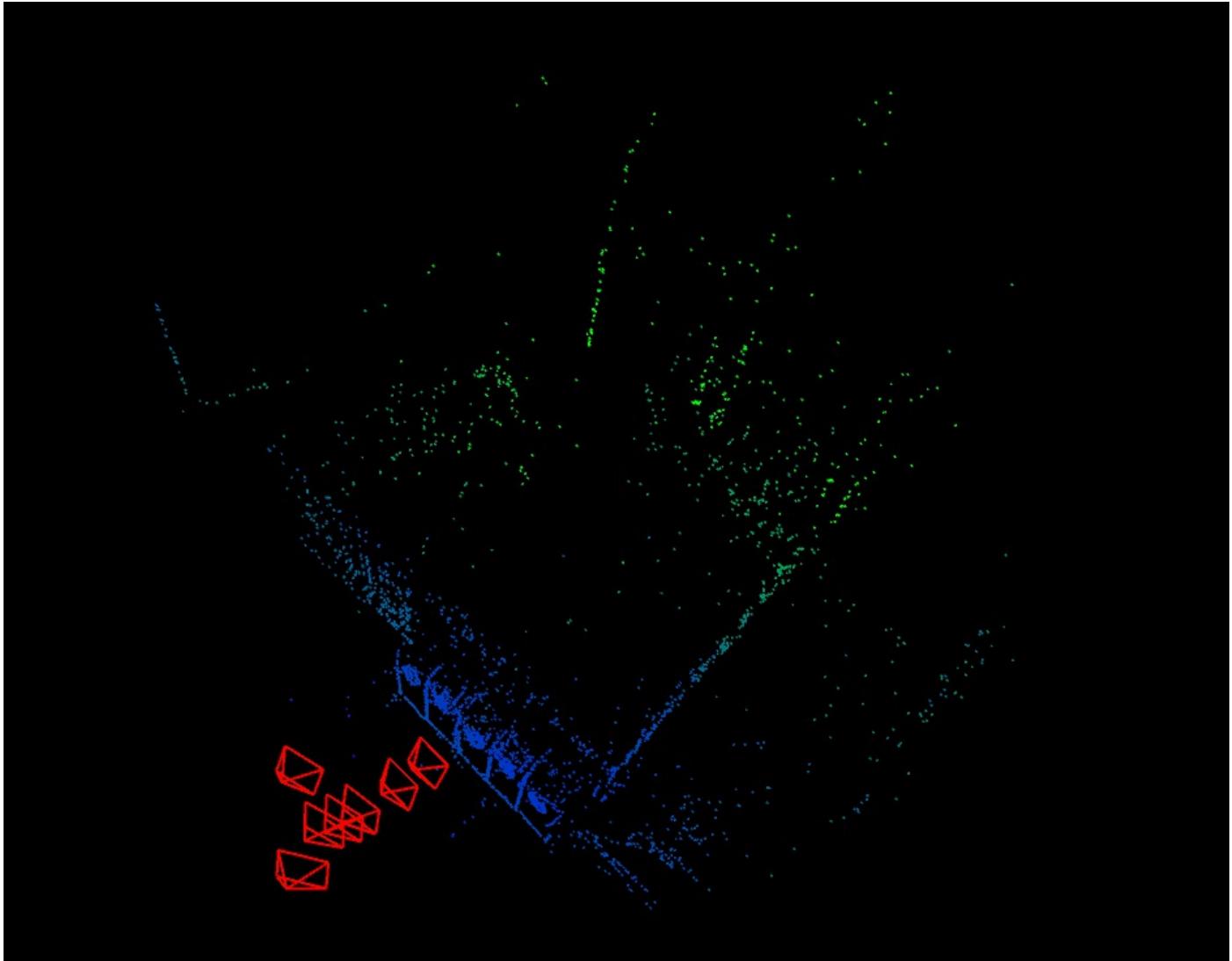
```

输出结果：

```

poses: 7, points: 4118
images: 7
iteration= 0 chi2= 3477120.470006 time= 0.291905 cumTime= 0.291905 edges= 28826 schur= 1 lambda= 370502.769268 levenbergIter= 1
iteration= 1 chi2= 3344617.246140 time= 0.31794 cumTime= 0.609845 edges= 28826 schur= 1 lambda= 123500.923089 levenbergIter= 1
iteration= 2 chi2= 3251169.050613 time= 0.207628 cumTime= 0.817473 edges= 28826 schur= 1 lambda= 41166.974363 levenbergIter= 1
iteration= 3 chi2= 3185504.282237 time= 0.229522 cumTime= 1.047 edges= 28826 schur= 1 lambda= 13722.324788 levenbergIter= 1
iteration= 4 chi2= 3133284.288598 time= 0.222119 cumTime= 1.26911 edges= 28826 schur= 1 lambda= 4574.108263 levenbergIter= 1
iteration= 5 chi2= 3096793.581097 time= 0.240709 cumTime= 1.50982 edges= 28826 schur= 1 lambda= 3049.405508 levenbergIter= 1
iteration= 6 chi2= 3060754.008807 time= 0.244027 cumTime= 1.75385 edges= 28826 schur= 1 lambda= 2032.937006 levenbergIter= 1
iteration= 7 chi2= 3037563.797124 time= 0.209707 cumTime= 1.96356 edges= 28826 schur= 1 lambda= 1355.291337 levenbergIter= 1
iteration= 8 chi2= 3019122.537340 time= 0.182949 cumTime= 2.14651 edges= 28826 schur= 1 lambda= 903.527558 levenbergIter= 1
iteration= 9 chi2= 2999333.811957 time= 0.207872 cumTime= 2.35438 edges= 28826 schur= 1 lambda= 602.351705 levenbergIter= 1
iteration= 10 chi2= 2989392.432606 time= 0.187216 cumTime= 2.54159 edges= 28826 schur= 1 lambda= 401.567804 levenbergIter= 1
iteration= 11 chi2= 2983449.064713 time= 0.188378 cumTime= 2.72997 edges= 28826 schur= 1 lambda= 267.711869 levenbergIter= 1
iteration= 12 chi2= 2975394.329805 time= 0.192648 cumTime= 2.92262 edges= 28826 schur= 1 lambda= 178.474579 levenbergIter= 1
iteration= 13 chi2= 2974850.362243 time= 0.179563 cumTime= 3.10218 edges= 28826 schur= 1 lambda= 118.983053 levenbergIter= 1
iteration= 14 chi2= 2972033.061321 time= 0.178025 cumTime= 3.28021 edges= 28826 schur= 1 lambda= 79.322035 levenbergIter= 1
iteration= 15 chi2= 2970780.935768 time= 0.176808 cumTime= 3.45702 edges= 28826 schur= 1 lambda= 52.881357 levenbergIter= 1
iteration= 16 chi2= 2966677.064687 time= 0.239296 cumTime= 3.69631 edges= 28826 schur= 1 lambda= 282.033903 levenbergIter= 3
iteration= 17 chi2= 2961559.999565 time= 0.173869 cumTime= 3.87018 edges= 28826 schur= 1 lambda= 188.022662 levenbergIter= 1
iteration= 18 chi2= 2958385.288552 time= 0.246322 cumTime= 4.1165 edges= 28826 schur= 1 lambda= 1002.787211 levenbergIter= 3
iteration= 19 chi2= 2953919.219271 time= 0.176466 cumTime= 4.29297 edges= 28826 schur= 1 lambda= 668.524808 levenbergIter= 1
iteration= 20 chi2= 2953399.928673 time= 0.249576 cumTime= 4.54254 edges= 28826 schur= 1 lambda= 3565.465640 levenbergIter= 3
iteration= 21 chi2= 2950292.394281 time= 0.23655 cumTime= 4.77909 edges= 28826 schur= 1 lambda= 19015.816749 levenbergIter= 3
iteration= 22 chi2= 2950006.998763 time= 0.21679 cumTime= 4.99588 edges= 28826 schur= 1 lambda= 25354.422332 levenbergIter= 2
iteration= 23 chi2= 2946643.222502 time= 0.175316 cumTime= 5.1712 edges= 28826 schur= 1 lambda= 16902.948221 levenbergIter= 1
iteration= 24 chi2= 2941632.900212 time= 0.252906 cumTime= 5.42411 edges= 28826 schur= 1 lambda= 90149.057179 levenbergIter= 3
iteration= 25 chi2= 2940590.362794 time= 0.173529 cumTime= 5.59764 edges= 28826 schur= 1 lambda= 60099.371453 levenbergIter= 1
iteration= 26 chi2= 2937414.139562 time= 0.23558 cumTime= 5.83322 edges= 28826 schur= 1 lambda= 320529.981082 levenbergIter= 3
iteration= 27 chi2= 2935938.623278 time= 0.1765 cumTime= 6.00971 edges= 28826 schur= 1 lambda= 213686.654055 levenbergIter= 1
iteration= 28 chi2= 2933566.527947 time= 0.248844 cumTime= 6.25856 edges= 28826 schur= 1 lambda= 1139662.154959 levenbergIter= 3
iteration= 29 chi2= 2932444.943189 time= 0.179863 cumTime= 6.43842 edges= 28826 schur= 1 lambda= 759774.769972 levenbergIter= 1
iteration= 30 chi2= 2931471.888185 time= 0.235734 cumTime= 6.67416 edges= 28826 schur= 1 lambda= 4052132.106519 levenbergIter= 3
iteration= 31 chi2= 2931054.355416 time= 0.177986 cumTime= 6.85214 edges= 28826 schur= 1 lambda= 2701421.404346 levenbergIter= 1
iteration= 32 chi2= 2930943.208411 time= 0.204936 cumTime= 7.05708 edges= 28826 schur= 1 lambda= 3601895.205795 levenbergIter= 2
iteration= 33 chi2= 2930832.433599 time= 0.177852 cumTime= 7.23493 edges= 28826 schur= 1 lambda= 2491263.479530 levenbergIter= 1
iteration= 34 chi2= 2930765.023085 time= 0.211164 cumTime= 7.44609 edges= 28826 schur= 1 lambda= 3201684.627373 levenbergIter= 2
iteration= 35 chi2= 2930704.135509 time= 0.206334 cumTime= 7.65243 edges= 28826 schur= 1 lambda= 4268912.836498 levenbergIter= 2
iteration= 36 chi2= 2930663.791521 time= 0.174816 cumTime= 7.82724 edges= 28826 schur= 1 lambda= 2845941.890999 levenbergIter= 1
iteration= 37 chi2= 2930622.296338 time= 0.209682 cumTime= 8.03693 edges= 28826 schur= 1 lambda= 3794589.187998 levenbergIter= 2
iteration= 38 chi2= 2930594.017854 time= 0.192238 cumTime= 8.22916 edges= 28826 schur= 1 lambda= 2529726.125332 levenbergIter= 1
iteration= 39 chi2= 2930450.112159 time= 0.244661 cumTime= 8.47383 edges= 28826 schur= 1 lambda= 13491872.668437 levenbergIter= 3
iteration= 40 chi2= 2930443.917394 time= 0.210172 cumTime= 8.684 edges= 28826 schur= 1 lambda= 17989163.557917 levenbergIter= 2
iteration= 41 chi2= 2930441.114242 time= 0.207474 cumTime= 8.89147 edges= 28826 schur= 1 lambda= 2398551.410555 levenbergIter= 2
iteration= 42 chi2= 2930441.114206 time= 0.413906 cumTime= 9.05358 edges= 28826 schur= 1 lambda= 4292381620202590.000000 levenbergIter= 8
iteration= 43 chi2= 2930441.113997 time= 0.198155 cumTime= 9.50353 edges= 28826 schur= 1 lambda= 2861587746801726.500000 levenbergIter= 1
iteration= 44 chi2= 2930441.113997 time= 0.206516 cumTime= 9.71005 edges= 28826 schur= 1 lambda= 5723175493603453.000000 levenbergIter= 1

```



同时思考并回答如下问题：

1. 能否不用[x,y,z]形式参数化每个点？

回答：

可以的，使用uv+深度信息也可以。

2. patch的大小选取有什么说法？

回答：

取小误差增大，取大了计算量指数增长，所以我认为这里的 4×4 挺合理的。

3. 从本题中，你看到直接法跟特征点法在BA阶段有何不同？

回答：

大部分都差不多，最大的差异体现在雅克比矩阵的计算以及误差的定义上。

4. 由于图像的差异，你可能需要鲁棒核函数，例如Huber。那Huber的阈值怎么选取？**回答：**

如果误差服从高斯分布，误差项的平方服从卡方分布，根据确定的误差项自由度、置信度，查卡方分布表可以得出其 p 值，即

Huber 阈值

一般置信度假设 0.95

4×4 的 patch, 其 error 维度为 16, 自由度为 16。