从 DSO 来看视觉里程计

文坤
22

关于 DSO

Direct+Sparse 的结合，使得算法既能很好的适应场景，而且实时性也很好。本文档将从算法和代码的各个层面展现 DSO 算法的细节，综合各种已有的资料，力求完整详细。

目录

1 视觉 SLAM 分类 **1**

1.1 稀疏-稠密法 1

1.2 直接-间接法 1

2 视觉中的数学问题 **3**

2.1 最大似然估计 3

2.2 边缘化 3

2.3 流形—lie group 3

2.3.1 李群对李代数的导数 4

2.3.2 SE(3) 的伴随矩阵 5

2.3.3 ξ_{ij} 对 ξ_i、ξ_j 的导数 5

2.4 矩阵正交化 7

3 DSO 原理 **9**

3.1 残差形式 9

3.2 参数形式 11

3.3 雅可比 11

3.3.1 图像雅可比 J_I 12

3.3.2 几何雅可比 J_{geo} 12

3.4 边缘化 12

4 DSO 代码分析–前端跟踪 **13**

4.1 标定矫正 13

4.2 系统初始化 13
4.3 帧间跟踪 13

5 DSO 代码分析–后端优化 **15**
5.1 点的深度计算 15
5.1.1 深度滤波 15
5.1.2 深度优化 16
5.2 Bundle Adjustment 16
5.3 边缘化 16

6 代码细节 **17**
6.1 比例因子 17

第 1 章　视觉 SLAM 分类

视觉 SLAM 可以从多个角度对其进行分类。

1.1　稀疏-稠密法

1.2　直接-间接法

第 2 章　视觉中的数学问题

2.1　最大似然估计

2.2　边缘化

2.3　流形—lie group

流形上的优化:

位置、速度定义在欧式空间，可以直接进行优化处理。在欧式空间定义，特性是对加法操作封闭，比如 $\mathbf{t}_2 = \mathbf{t}_1 + \triangle\mathbf{t}$。但是姿态，只对乘法操作封闭 $\mathbf{R}_2 = \triangle\mathbf{R} * \mathbf{R}_1$，需要在流形空间中进行优化。
简单说，流形是一个非线性空间，但是在局部空间内对其进行线性化，就可以用线性空间进行拟合。　　对于一般的优化问题:

$$x = \tag{2.1}$$

在流形上优化的好处:
1、有约束优化问题转化为无约束优化问题，如 det(R)=1，有约束的优化问题会引入拉格朗日因子，优化变量维数会更高，转换李代数后，没有什么约束了，

2.3.1 李群对李代数的导数

对于 $\xi \in SE(3)$ 定义

$$\xi^{\wedge} = \begin{bmatrix} \rho \\ \phi \end{bmatrix}^{\wedge} = \begin{bmatrix} \phi^{\wedge} & \rho \\ 0^T & 0 \end{bmatrix} \in R^{4x4}, \rho, \phi \in R^3 \tag{2.2}$$

$$\xi^{\curlywedge} = \begin{bmatrix} \rho \\ \phi \end{bmatrix}^{\curlywedge} = \begin{bmatrix} \phi^{\wedge} & \rho^{\wedge} \\ 0^T & \phi^{\wedge} \end{bmatrix} \in R^{6x6}, \rho, \phi \in R^3 \tag{2.3}$$

其中 ρ 为平移分量，ϕ 为旋转分量。

假设某空间点 p 经过一次变换 T(对应李代数 ξ), 得到 Tp。现在，给 T 左乘一个扰动 $\triangle T = exp(\delta\xi^{\wedge})$, 假设扰动项的李代数为 $\delta\xi = [\delta\rho, \delta\phi]^{\wedge})$, 那么：

$$
\begin{aligned}
\frac{\partial(Tp)}{\partial\delta\xi} &= \lim_{\delta\xi\longrightarrow 0}\frac{exp(\delta\xi^{\wedge})exp(\xi^{\wedge})p - exp(\xi^{\wedge})p}{\delta\xi} \\
&\approx \lim_{\delta\xi\longrightarrow 0}\frac{(I+\delta\xi^{\wedge})exp(\xi^{\wedge})p - exp(\xi^{\wedge})p}{\delta\xi} \\
&= \lim_{\delta\xi\longrightarrow 0}\frac{\delta\xi^{\wedge}exp(\xi^{\wedge})p}{\delta\xi} \\
&= \lim_{\delta\xi\longrightarrow 0}\frac{\begin{bmatrix}\delta\phi^{\wedge} & \delta\rho \\ \mathbf{0}^T & 0\end{bmatrix}\begin{bmatrix}\mathbf{R}p+\mathbf{t} \\ 1\end{bmatrix}}{\delta\xi} \\
&= \lim_{\delta\xi\longrightarrow 0}\frac{\begin{bmatrix}\delta\phi^{\wedge}(\mathbf{R}p+\mathbf{t})+\delta\rho \\ \mathbf{0}^T\end{bmatrix}}{\delta\xi} \\
&= \lim_{\delta\xi\longrightarrow 0}\frac{\begin{bmatrix}-(\mathbf{R}p+\mathbf{t})^{\wedge}\delta\phi+\delta\rho \\ \mathbf{0}^T\end{bmatrix}}{\delta\xi} \\
&= \begin{bmatrix}\boldsymbol{I} & -(\mathbf{R}p+\mathbf{t}) \\ \mathbf{0}^T & \mathbf{0}^T\end{bmatrix} = (\mathbf{T}p)^{\odot}
\end{aligned}
\tag{2.4}
$$

2.3.2 SE(3) 的伴随矩阵

李群伴随矩阵的作用，相当于实现李群乘法的交换律：

$$\mathbf{T}exp(\xi^{\wedge}) = exp((Ad(\mathbf{T})\xi)^{\wedge})\mathbf{T} \tag{2.5}$$

稍加改变，用 $\mathbf{T}^{-1}$ 代替 T 得：(推导有待继续查，参考 14 讲 11.1.2 节)

$$exp(\xi^{\wedge})\mathbf{T} = \mathbf{T}exp((Ad(\mathbf{T}^{-1})\xi)^{\wedge}) \tag{2.6}$$

伴随矩阵的解释，还可参照 JingeTu 的理解

2.3.3 ξ_{ij} 对 ξ_i、ξ_j 的导数

这部分的推到参考：

1）视觉 slam14 讲 11.1.2 节 Pose Graph 的优化

2)JingeTu 的博客 https://www.cnblogs.com/JingeTU/p/9077372.html ，只是这里的推导方式不是很好，结合 1）中的扰动法，推导过程如下。

参考坐标系为全局坐标系时

假设 pose 在世界坐标系 w 下定义，ξ_i、ξ_j 即 ξ_{wi}、ξ_{wj}，则

$$\xi_{ij} = \xi_{wi}^{-1} \circ \xi_{wj} = In(exp((-\xi_{wi})^{\wedge})exp(\xi_{wj}^{\wedge}))^{\vee} \tag{2.7}$$

对应的李群：

$$\mathbf{T}_{ij} = \mathbf{T}_{wi}^{-1}\mathbf{T}_{wj} = exp(\xi_{ij}^{\wedge}) \tag{2.8}$$

采用扰动法对 ξ_{wi}、ξ_{wj} 进行求导

$$\begin{aligned}
\xi_{ij} &= In(\mathbf{T}_{wi}^{-1} * \mathbf{T}_{wj})^{\vee} \\
&= In((exp(\delta\xi_{wi}^{\wedge})\mathbf{T}_{wi})^{-1}exp(\delta\xi_{wj}^{\wedge})\mathbf{T}_{wj})^{\vee} \\
&= In(\mathbf{T}_{wi}^{-1}exp((-\delta\xi_{wi})^{\wedge})exp(\delta\xi_{wj}^{\wedge})\mathbf{T}_{wj})^{\vee} \\
&= In(\mathbf{T}_{wi}^{-1}exp(-\delta\xi_{wi})\mathbf{T}_{wj}exp((Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wj})^{\wedge}))^{\vee} \\
&= In(\mathbf{T}_{wi}^{-1}\mathbf{T}_{wj}exp((-Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wi})^{\wedge})exp((Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wj})^{\wedge}))^{\vee} \\
&\approx In(\mathbf{T}_{wi}^{-1}\mathbf{T}_{wj}[\mathbf{I} - Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wi} + Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wj}])^{\vee} \\
&\approx In(exp(\xi_{ij}^{\wedge})[\mathbf{I} - Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wi} + Ad(\mathbf{T}_{wj}^{-1})\delta\xi_{wj}])^{\vee} \\
&\approx \xi_{ij} + \frac{\partial\xi_{ij}}{\partial\delta\xi_{wi}}\delta\xi_{wi} + \frac{\partial\xi_{ij}}{\partial\delta\xi_{wj}}\delta\xi_{wj}
\end{aligned} \tag{2.9}$$

按照李代数上的求导法则，可求出 ξ_{ij} 对两个位姿 ξ_i、ξ_j 的雅可比矩阵：

$$\frac{\partial\xi_{ij}}{\partial\delta\xi_{wi}} = -\mathcal{J}^{-1}(\xi_{ij})Ad(T_j^{-1}) \tag{2.10}$$

$$\frac{\partial\xi_{ij}}{\partial\delta\xi_{wj}} = \mathcal{J}^{-1}(\xi_{ij})Ad(T_j^{-1}) \tag{2.11}$$

由于 se(3) 上的左右雅可比 $\mathcal{J}$ 形式过于复杂，通常取其近似。如果误差接近于零，就可以将它近似于 I 或者

$$\mathcal{J}^{-1} \approx I + \frac{1}{2}\begin{bmatrix} \phi^\wedge & \rho^\wedge \\ 0 & \phi^\wedge \end{bmatrix} \tag{2.12}$$

参考坐标系为局部坐标系时

当参考坐标系选取为当前载体坐标系时，即 ξ_i、ξ_j 即 ξ_{iw}、ξ_{jw}，则

$$\frac{\partial \xi_{ij}}{\partial \delta \xi_{iw}} = \mathbf{I} \tag{2.13}$$

$$\frac{\partial \xi_{ij}}{\partial \delta \xi_{jw}} = -Ad(\mathbf{T}_{th}) \tag{2.14}$$

2.4 矩阵正交化

矩阵正交化本是矩阵问题，放在几何部分中，是因为想从几何的角度中来阐述。

本小节内容参见：https://blog.csdn.net/tengweitw/article/details/41174555、https://blog.csdn.net/tengweitw/article/details/41775545

矩阵正交投影

图中，$\boldsymbol{e} = \boldsymbol{b} - \boldsymbol{p} = \boldsymbol{b} - x\boldsymbol{a}$，向量 $\boldsymbol{e}$ 为投影残差

向量 $\boldsymbol{a}$ 与向量 $\boldsymbol{e}$ 垂直，

$\boldsymbol{a}^T\boldsymbol{e} = 0 \longrightarrow \boldsymbol{a}^T(\boldsymbol{b} - x\boldsymbol{a}) = 0 \longrightarrow x\boldsymbol{a}^T\boldsymbol{a} = \boldsymbol{a}^T\boldsymbol{b} \longrightarrow x = \frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$

x 是一个标量值，刻画了向量 b 投影到向量 a 上的长度。

$$\boldsymbol{p} = \boldsymbol{a}x = \boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{b}}{\boldsymbol{a}^T\boldsymbol{a}}$$

$\mathbf{P}$ 为投影矩阵，$\mathbf{P}\boldsymbol{b} = \boldsymbol{p}$, 则：

$$\mathbf{P} = \frac{\boldsymbol{a}\boldsymbol{a}^T}{\boldsymbol{a}^T\boldsymbol{a}}$$

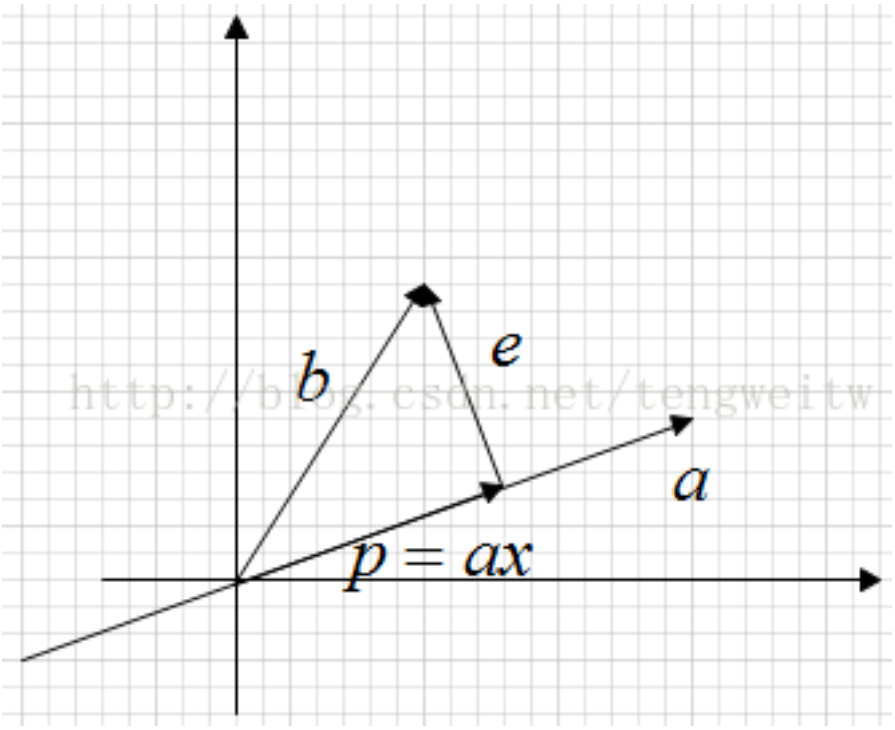

图 2.1: 向量 b 在向量 a 上的投影.

Gram-Schmidt 正交化

第 3 章　DSO 原理

在 DSO 中，pose 表示在载体坐标系，即 $\mathbf{T}_i$ 为 $\mathbf{T}_{iw}$。

3.1　残差形式

残差公式 r^k 是 Host 帧 i 到 Target 帧 j 上的匹配点的 K 的某个 patch 投影的 residual:

$$\begin{aligned} r_{ij}^k &= r^k(x \boxplus \xi_0) \\ &= I_j[p^{'}(T_i, T_j, d_k, c)] - b_j - \frac{t_j e^{a_j}}{t_i e^{a_i}}(I_i(p^k) - b_i) \end{aligned} \tag{3.1}$$

其中

$$\begin{aligned} p^{'} &= \pi(R\pi^{-1}(p, d_p) + t) \\ &= \pi(T_j T_i^{-1} \pi^{-1}(p, d_p)) \end{aligned} \tag{3.2}$$

而位姿的 se(3) 表示为

$$T_j = exp(\hat{\xi}_j), T_i = exp(\hat{\xi}_i) \tag{3.3}$$

投影和反投影方程分别为

$$\pi(P) = (\frac{f_x P_x}{P_z} + c_x, \frac{f_y P_y}{P_z} + c_y) \tag{3.4}$$

$$\pi^{-1}(P, d_p) = (\frac{p_x - c_x d_p}{f_x}, \frac{p_y - c_y d_p}{f_y}, d_p) \tag{3.5}$$

注意：

1、残差方程中，涉及到 i、j 两个时刻的 pose。而基于 feature point 的 slam 中只于当前时刻的 pose 相关，将全局坐标系下面的 point 转换到当前帧下计算重投影误差。

2、考虑了亮度值的仿射变换 a、b。

以上对应论文中的残差形式，而代码中涉及到更为细节，下面进行展开。这里考虑在优化时有两个关键帧，一个称为 Host, 一个称为 Target。取 Host 帧中的一个像素点：

$$x_H = [u_H, v_H, 1]_H^T \tag{3.6}$$

其中 u_H, v_H 为该点的像素坐标系，使用齐次坐标为了方便矩阵运算。同时，点的逆深度为：

$$\rho_H = \frac{1}{d_H} \tag{3.7}$$

在不考虑亮度放射变换时，点 p 从 Host 帧 i 变换到到 Target 帧 j 上的过程为：

$$\begin{aligned} &\overbrace{\rho_T^{-1} K^{-1} x_T = T_{TW}}^{P_W} \overbrace{T_{HW}^{-1} \frac{1}{\rho_H} K^{-1} x_H}^{P_W} \\ &\underbrace{\rho_T^{-1} K_{-1} x_T}_{p_T} = \underbrace{T_{TW} T_{HW}^{-1}}_{T_{TH}} \underbrace{\frac{1}{\rho_H} K^{-1} x_H}_{p_H} \end{aligned} \tag{3.8}$$

将 SE(3) 形式展开

$$T_{TH} = \begin{bmatrix} R_{TH} & t_{TH} \\ 0 & 1 \end{bmatrix} \tag{3.9}$$

代入上式得：

$$x_T = \frac{\rho_T}{\rho_H} (K R_{TH} K^{-1} x_H + K t_{TH} \rho_H) \tag{3.10}$$

注：

1) 公式中省略了齐次坐标和非齐次坐标的转换过程。推导时请自行脑补。

2) 对于逆深度的操作，有些和常规习惯不同，但是代码中就是这样实现的，而且感觉挺方便的，后面会结合具体代码进行分析。

3) 公式 (3.2) 中的 $p^{'}$ 表示一个投影过程, 而 (3.10) 的 x_H 表示从 Host 中变换到 Target 后的一个点，二者实质是一样的。即 (3.10) 是 (3.2) 的展开形式。

3.2 参数形式

涉及到的参数为 ξ_i、ξ_j、d、c、a_i、aj、b_i、b_j。其中 ξ_i、$\xi_j \in SE(3)$ 为两帧的外参，d 为 point 在其 host 帧中的深度，$c \in R^4$ 为相机的内参，a_i, aj, b_i, b_j 分别为光照参数。

按照 Jacobian 的结构可以分为 2 类，T_i、T_j、d、c= (f_x, f_y, c_x, c_y) 为 geometry 参数 δ_{geo}, 而 a_i、a_j、b_i、b_j 为 photometric 参数 δ_{photo}。

每个点以某一个 pattern 来构成,pattern 形式见论文"Figure 4.**Residual pattern**"。按照层级，参数又可以分为全局属性 c、帧属性 ξ_i、ξ_j、ab_i、ab_j，点属性 d_{k_i}(注意，这里是 i, 表示的是点在 host 帧下面的深度)。自变量中 d 的数量是最多。

同时，按照参数的来源的分类，即可以分为 Host 参数 (下标为 i，代码中表示为为 h) 和 Target 参数 (下标为 j，代码中表示为 t)

3.3 雅可比

Jacobian 定义为:

$$J_k = \frac{\partial r^k((\delta + x) \boxplus \xi_0)}{\partial \delta} \tag{3.11}$$

论文里, 根据残差形式 (3.1) 将其划分为两部分

$$J_k = [\underbrace{\frac{\partial I_j}{\partial p^{'}}}_{J_I} \underbrace{\frac{\partial p^{'}(\delta + x) \boxplus \xi}{\partial \delta_{geo}}}_{J_{geo}}, \underbrace{\frac{\partial r_k((\delta + x) \boxplus x_0)}{\partial \delta_{photo}}}_{J_{photo}}] \tag{3.12}$$

其中，$J_I = \frac{\partial I_j}{\partial p^{'}} = (\frac{\partial I_j}{\partial p_x^{'}}, \frac{\partial I_j}{\partial p_y^{'}}) \in R_{1x2}$ 是当前像素的的亮度的梯度值
在完整 DSO 中，雅可比由三部分组成:
图像雅可比 J_I，即图像梯度;

几何雅可比 J_{geo}，描述各量相对几何量，例如旋转和平移的变化率；
光度雅可比 J_{photo}，描述各个量相对光度参数的雅可比；

3.3.1 图像雅可比 J_I

$J_I = \frac{\partial I_j}{\partial p'} = \frac{\partial I_j}{\partial x_T}$ 即图像的梯度

3.3.2 几何雅可比 J_{geo}

对 pose 求导

几何部分包括相机的位姿和特征点的深度，需要对两部分求雅可比。
记 ξ_T 和 ξ_H 分别为 T_{TW}、T_{HW} 的李代数形式, 位姿部分的雅可比:

$$\frac{\partial x_H}{\partial \xi_T} = \begin{bmatrix} \rho_T f_x & 0 & -\rho_T u f_x & -uv f_x & (1+u^2) f_x & -v f_x \\ 0 & \rho_T f_y & -f_y \rho_T v & -(1+v^2) f_y & f_y uv & f_y u \end{bmatrix} \tag{3.13}$$

$$\frac{\partial x_H}{\partial \xi_H} = ???????? \tag{3.14}$$

对逆深度求导

再考虑对 Host 帧中的逆深度 ρ_H 的雅可比:

$$\frac{\partial x_T}{\partial \rho_H} = \begin{bmatrix} \frac{\partial x_T}{\partial u} & \frac{\partial x_T}{\partial v} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \rho_H} \\ \frac{\partial v}{\partial \rho_H} \end{bmatrix} \tag{3.15}$$

对相机内参求导

3.4 边缘化

第 4 章　DSO 代码分析–前端跟踪

整个项目代码量比较大，文件很多。而且作者为了提高计算效率，优化相关的部分，很多中间结果共用，也没有采用开源的优化库 ceres、gtsam 等，所有部分纯手撸。而且还进行了 SSE 加速，也没使用 OpenCV 中的算法。因此，代码及其复杂，还涉及到分厂多的参数，这些参数与作者的使用调试经验强相关。整个关键代码部分，可以分为三个部分：

- 初始化
- 帧间跟踪
- 局部优化

初始化　作者在 github 的项目主页上说到，初始化并不是很鲁棒，用户可以自己实现自己的初始化操作。

帧间跟踪　帧间跟踪主要是进行图像对齐，计算帧间的 pose，类似基于特征点法计算 H 或者 F 分解得到 Pose 的过程，同时还要更新点的深度信息。

局部优化　局部优化中做的事儿就有点儿多了，重头戏。

4.1　标定矫正

4.2　系统初始化

4.3　帧间跟踪

DSO 中帧间跟踪时，只计算当前帧与前一个**关键帧**间的 pose，即粗略跟踪。

完成粗跟踪的类是 CoarseTracker，该类在 FullSystem 中共有两个对象，分别是 coarseTracker_forNewKF、coarseTracker。
二者的区别是：
coarseTracker: 用于所有帧的跟踪，每一个非关键帧到来时，都计算相对于 coarseTracker 中保存的关键帧的相对 pose。两个关键帧间的所有非关键帧使用的 coarseTracker 对象是同一个。
coarseTracker_forNewKF: 用于更新存储当前关键帧的信息。即当前帧是关键帧时，更新该对象，然后在下一帧 (非关键帧) 到来时，复制给 coarseTracker 对象。更新的信息包括 refFrameID 信息，在 makeKeyFrame() 函数中完成，且是在完成关键帧之后和边缘化之前。

新帧到来时，若 coarseTracker_forNewKF 中的 refFrameID 比 coarseTracker 中的 refFrameID 大 (因为其在前一关键帧中被更新，而 coarseTracker 中保存的还是前前关键帧)，就将 coarseTracker_forNewKF 赋给 coarseTracker。若没有，coarseTracker 维持之前的状态，即最后一个 keyframe。更新过程只发生在关键帧及其下一帧中间。

第 5 章　DSO 代码分析–后端优化

后端优化在 makeKeyFrame() 中，依次有十几项步骤：

5.1　点的深度计算

关键点的深度的计算都在后端完成，前端之负责计算两帧间的 pose 的初值。分为两步：

1) 通过深度滤波器更新点的深度范围，直到点的深度收敛，作为深度的初值。

2) 在完成帧间 pose 的优化后，进一步优化点的深度值。

5.1.1　深度滤波

通过深度滤波器完成深度范围的更新,在函数 ImmaturePoint::traceOn() 中，主要维持两个变量:idepth_min、idepth_min。将点投影到 Target 帧中，形成对应的两个点 ptpMin、ptpMax，如果两个点的距离小于一定的阈值，则认为该点深度收敛：

Listing 5.1 ImmaturePoint::traceOn()

```
Vec3f pr = hostToFrame_KRKi * Vec3f(u,v, 1);
Vec3f ptpMin = pr + hostToFrame_Kt*idepth_min;
.....
Vec3f ptpMax;
ptpMax = pr + hostToFrame_Kt*idepth_max;
......
if(!(uMax > 4 && vMax > 4 && uMax < wG[0]-5 && vMax < hG[0]-5))
{
  if(debugPrint) printf("OOB␣uMax␣␣%f␣%f␣-␣%f␣%f!\n",u,v, uMax, vMax);
  lastTraceUV = Vec2f(-1,-1);
  lastTracePixelInterval=0;
```

```
12:
13:   return lastTraceStatus = ImmaturePointStatus::IPS_OOB;
14: }
```

对深度进行深度更新：

$$\begin{aligned} u - \delta edx &= \frac{(pr + \rho_{min} Kt)[0]}{(pr + \rho_{min} Kt)[2]} \\ u + \delta edx &= \frac{(pr + \rho_{max} Kt)[0]}{(pr + \rho_{max} Kt)[2]} \end{aligned} \tag{5.1}$$

式中，$pr = KR_{TH}K^{-1} * p_H$ 是点从 Host 帧中旋转到 Target 帧后的样子 (即 Point Rotation 的简写)，未考虑平移。平移和深度相关，深度相关的和前面重点强调的是一样的。变形后得到代码中的形式：

$$\begin{aligned} \rho_{min} &= \frac{(pr[2] * (u - \delta edx) - pr[0]}{(kt)[0] - (Kt)[2] * (u - \delta edx)} \\ \rho_{max} &= \frac{(pr[2] * (u + \delta edx) - pr[0]}{(kt)[0] - (Kt)[2] * (u + \delta edx)} \end{aligned} \tag{5.2}$$

5.1.2 深度优化

在优化时，会专门对深度进行优化

5.2 Bundle Adjustment

这一部分是整篇中最为核心的地方。作者将 Hessian 矩阵分为 active、linear、marginal 三部分。在 AccumulatedTopHessianSSE::addPoint() 函数中，模板参数 mode 的值 0、1、2 分别表示 active、linearized、marginalize。但是经测试 linearized 下面的实际部分并未执行，作者在逻辑中将其屏蔽掉了。

active 部分

marginal 部分

5.3 边缘化

第 6 章　代码细节

6.1　比例因子

代码中好多地方出现比例因子，这些因子作用是提高求解方程式

$$H \triangle x = b$$

的数值稳定性和精确度.
出现的地方:
1、方程求解的过程中
2、FrameHessian 中