DSO 初始化笔记

赵世博, 王鹏

August 2019

1 DSO 介绍

DSO 的全称是 Direct Sparse Odometry, 是一种将直接法和稀疏法相结合的视觉里程计. 由于它不是完整的 SLAM 系统, 因此它不包含回环检测, 地图复用等功能, 无法消除累计误差. 由于 DSO 是直接对整张图像内提取强度梯度明显变化的像素点, 而不依赖环境中某个特定的特征点和描述子, 因此该方法有望解决弱纹理场景下的位姿估计问题, 即室内白墙等.

直接法与特征点法区别 这里简单总结一下直接法与特征点法的区别 [7], 如图 1所示.

1.1 DSO 算法程序结构

DSO 整体代码由四个部分组成：系统与各算法集成于 src/FullSystem，后端优化位于 src/OptimizationBackend，这二者组成了 DSO 大部分核心内容。src/utils 和 src/IOWrapper 为一些去畸变、数据集读写和可视化 UI 代码。本文主要解析 DSO 初始化操作.

2 DSO 初始化流程

DSO 初始化主要是对前两帧进行处理, 本文将对初始化过程所遇到的主要函数进行介绍, 主要的步骤如图 2所示。

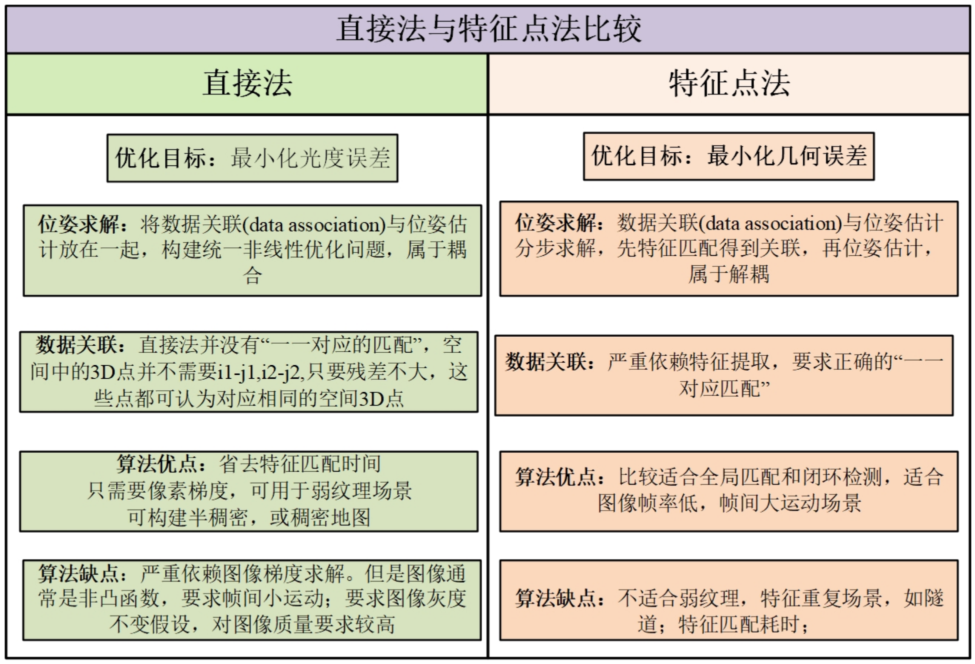

图 1: 直接法与特征点法区别

3 getImage()| 输入图像

DSO 程序的真正的入口在 ImageAndExposure* getImage(int id, bool forceLoadDirectly=false) 函数中. 该函数主要完成以下几个步骤:

- 去除光度非线性响应、光晕对图像的影响.(processFrame 函数中)
- 采用插值方式添加图像的随机几何噪声 (**option**)
- 添加图像的光度噪声.(applyBlurNoise(float* img) 函数中)(**option**)

3.1 直接法为什么要进行光度标定？

基于特征点法的位姿估计: 传统的视觉特征如 SIFT，AKAZE，ORB 等，都一般具有光照不变性，对图像的亮度值并不敏感，因此可以直接拿来求解。

基于直接法的位姿估计: 位姿估计主要依赖图像的亮度值，因此亮度值的准确度会直接影响算法的稳定性和精度。因此，DSO 的作者引入光度标

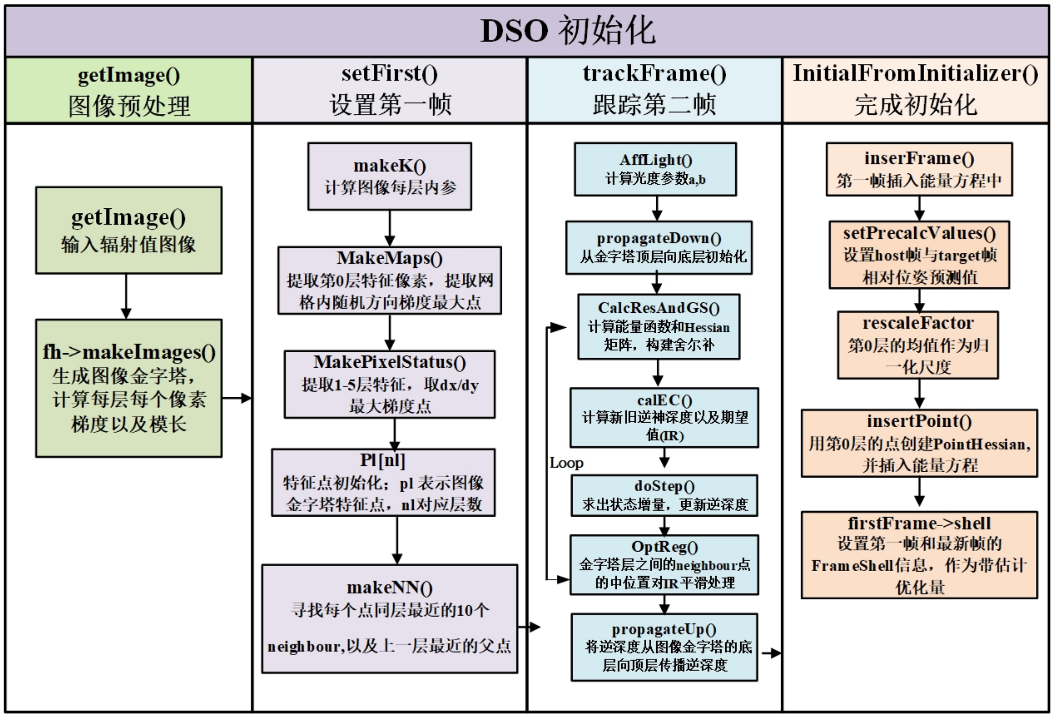

图 2: DSO 初始化结构图

定概念，通过构建光度模型，从而实现图像亮度纠正。图 3展示的是相机拍摄的一张纯色图片，理论上这张图片的各个像素值应该是相等的，但是实际统计过程中，我们发现真实的像素值并不一样。

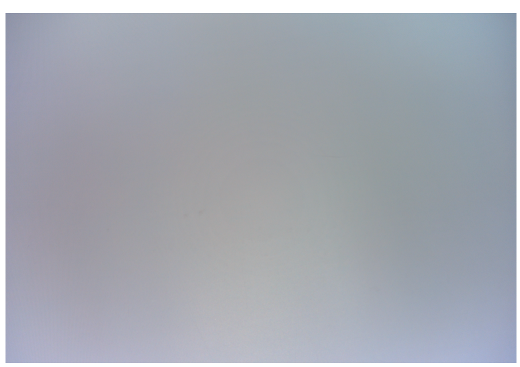

图 3: 相机拍摄的纯色图片

统计结果如 4所示，你会发现图像边缘的像素与图像中心点的像素可能会差 2 倍，这种图像如果不进行光度标定，显然可能会影响 DSO 估计位姿的精度。

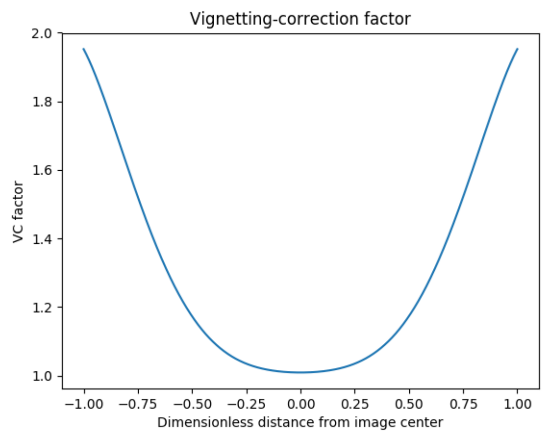

图 4: 纯色图片像素值变化曲线

3.2 光度标定都标定哪些参数？

根据相关光度标定论文 [1]，作者给出的相机光度模型如下所示:

$$I(\mathbf{x}) = G(tV(\mathbf{x})B(\mathbf{x})) \tag{1}$$

- G 为相机的响应函数 (response function), 其阈值是离散的 (0-255), 是传感器的固有属性
- V 为归一化的渐晕函数 (vignette)，作者用一个和图像一样大的权重矩阵，来表示其对每个像素的影响
- t 表示曝光时间。对于 rolling shutter 相机，每一帧的 t 都不相同
- B 表示不受光学模块影响的是辐射值图像
- I 表示输出的图像

3.3 作者是怎么标定的？

作者主要标定两个参数，第一个是相机响应函数 (Response Calibration); 第二个是渐晕标定 (Vignette Calibration)[2]

相机响应函数标定 (Response Calibration)

作者采用非参数估计的方法进行标定。具体来讲，作者将相机固定，对着在静态固定的场景，在相机不同的曝光时间下，重复拍照得到一系列图像。由于场景不变，因此相机的辐射值图像 B 是不变的。这里我们令 $B'(\mathbf{x}) := V(\mathbf{x})B(\mathbf{x})$. 公式 1就可以写成如下形式:

$$I(\mathbf{x}) = G\left(tB'(\mathbf{x})\right) \tag{2}$$

由于作者求解的是响应函数的反函数 $U(I(\mathbf{x})) = G^{-1}$,并且假设 $U(I(\mathbf{x}))$ 上有一个零均值的高斯噪声，那么公式 2 可以写成如下形式：

$$U\left(I_i(\mathbf{x})\right) = t_i B'(\mathbf{x}) + n_i, \quad n_i \sim N\left(0, \sigma_i^2\right) \tag{3}$$

通过最大似然估计，给出最小二乘的形式：

$$E\left(U, B'\right) = \sum_i \sum_{\mathbf{x}\in\Omega} \left(U\left(I_i(\mathbf{x})\right) - t_i B'(\mathbf{x})\right)^2 \tag{4}$$

第一个求和对应的是不同曝光时间下的所有图像，第二个求和对应的是图像平面上所有的像素点，通过对上式最小化，轮流迭代分别求解出 U 和 B'. 最终可得到:

$$U(k)^* = \arg\min_{U(k)} E\left(U, B'\right) = \frac{\sum_{\Omega_k} t_i B'(\mathbf{x})}{|\Omega_k|} \tag{5}$$

$$B'(\mathbf{x})^* = \arg\min_{B'(\mathbf{x})} E\left(U, B'\right) = \frac{\sum_i t_i U\left(I_i(\mathbf{x})\right)}{\sum_i t_i^2} \tag{6}$$

公式 5 $U(k)$ 的求解，是通过非参数方式进行的。由于 $U(k)$ 对应一维数组，对应固定的 k 值，$U(k)$ 也是常量。因此在在求解公式 4 中 $U(k)$ 相当于对一个变量求导，因此就有:

$$\sum_i \sum_{\mathbf{x}\in\Omega'} \left(U(k) - t_i B'(\mathbf{x})\right) = \sum_i \sum_{\mathbf{x}\in\Omega'} U(k) - \sum_i \sum_{\mathbf{x}\in\Omega'} t_i B'(\mathbf{x}) = \sum_{\mathbf{x}\in\Omega_k} U(k) - \sum_{\mathbf{x}\in\Omega_k} t_i B'(\mathbf{x}) = 0 \tag{7}$$

其中 $\Omega' := \{\mathbf{x}|I_i(\mathbf{x}) = k\}$ 表示第 i 帧图像所有像素值为 k 的像素点集合. 这里 $\Omega_k := \{i, \mathbf{x}|I_i(\mathbf{x}) = k\}$ 表示所有图像中，像素值等于 k 的像素点集合。

对于 $B'(\mathbf{x})$ 同理，我们将公式 8，分别对每一个 $B'(\mathbf{x})$ 求偏导，可得:

$$E\left(U, B'\right) = \sum_{\mathbf{x}\in\Omega} \underbrace{\sum_i \left(U\left(I_i(\mathbf{x})\right) - t_i B'(\mathbf{x})\right)^2}_{E(U,B',\mathbf{x})} \tag{8}$$

$$2\sum_i t_i \left(U\left(I_i(\mathbf{x})\right) - t_i B'(\mathbf{x})\right) = 2\sum_i t_i U\left(I_i(\mathbf{x})\right) - 2\sum_i t_i^2 B'(\mathbf{x}) \tag{9}$$

渐晕标定 (Vignette Calibration)

渐晕标定特使采用非参数化的形式，作者采用渐晕映射表 $V : \Omega \to [0, 1]$。标定过程是对一块白墙进行采集，作者假设白墙是理想的朗伯反射面 (Lambertian Suface), 即固定的照明下，从任意视角上观察都有相同的亮度平面。作者将 Marker 贴在墙上，如下图所示：

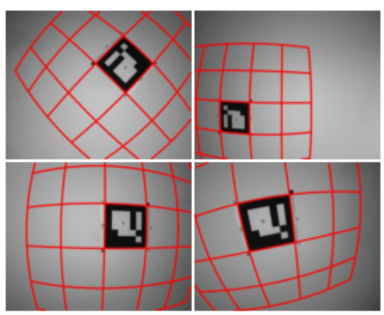

图 5: 晕影标定

从图 5可知，作者是在不同的角度对 marker 进行拍摄，白墙上不同位置的三维点相对于镜头的位置是不一样的，晕影影响也不一样。因此需要将白墙变换到相机坐标系，这里就引入位姿估计问题。定义 3D 空间到图像平面的映射 $\pi : \mathcal{P} \to \Omega$. 通过最大似然化，得到如下误差方程:

$E(C, V) = \sum_{i,x\in p} \left(t_i V\left([\pi_i(\mathbf{x})]\right) C(\mathbf{x}) - U\left(I_i\left(\pi_i(\mathbf{x})\right)\right)\right)^2$

其中 C 表示平面点到相机的辐射度 irradiance，属于未知变量。通过交

替迭代求解最小化 C 和 V, 可以得到 (方法与求解响应函数类似):

$$C^*(\mathbf{x}) = \arg\min_{C(\mathbf{x})} E(C, V) = \frac{\sum_i t_i V([\pi_i(\mathbf{x})]) U(I_i(\pi_i(\mathbf{x})))}{\sum_i (t_i V([\pi_i(\mathbf{x})]))^2} \tag{10}$$

$$V^*(\mathbf{x}) = \arg\min_{V(\mathbf{x})} E(C, V) = \frac{\sum_i t_i C(\mathbf{x}) U(I_i(\pi_i(\mathbf{x})))}{\sum_i (t_i C(\mathbf{x}))^2} \tag{11}$$

最终目标: 通过前文标定出相机的响应函数 G 以及渐晕 V，我们就可以通过公式反解算出辐射值 B。最终我们通过辐射值 B 去做直接法.

$$I_i'(\mathbf{x}) := t_i B_i(\mathbf{x}) = \frac{G^{-1}(I_i(\mathbf{x}))}{V(\mathbf{x})} \tag{12}$$

3.4 makeImages()| 构建金字塔

- makeImages 函数对每一帧图像构建金字塔, 并计算各层金字塔图像的像素值和梯度.
- 其中 dIp 表示每一层图像的像素值、x 方向梯度、y 方向梯度；absSquaredGrad 存储 xy 方向梯度值的平方和

4 setFirst()| 设置第一帧

4.1 makeK()| 计算图像每层内参

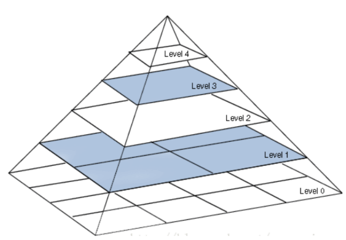

图 6: 图像金字塔

- 构建 0-5 层图像金字塔，并设置每层对应的内参，见图 6

4.2 MakeMaps()| 图像第 0 层提取梯度最大点

该函数主要进行的操作有:

4.2.1 makeHists()| 计算梯度直方图

- 将图像划分为固定的 block, 每个 block 大小固定为 32*32
- 在每个 block 内创建直方图 hist0
- 统计直方图 hist0 中像素数目占百分之 50 对应的梯度作为阈值 ths
- 对阈值进行 3*3 的均值滤波，得到 thsSmoothed

4.2.2 PixelSelector::select()| 在当前帧选择符合条件的像素

按照不同层选取点的密度 density[]=0.03,0.05,0.15,0.5,1 在当前帧上选择符合条件的像素，同时通过划分网格保证提取的特征点分布均匀，见算法 1

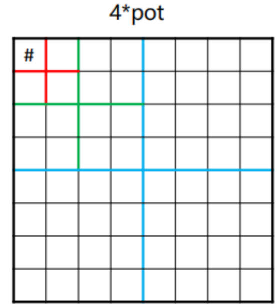

图 7: 图像金字塔

Algorithm 1 第 0 层金字塔图像特征选择

Input: 输入需要的梯度点的数目 N_{des}

Result: 输出选择的梯度点

1: 初始化选择的梯度点为空 as $\{\emptyset\}$, $N_{sel} = 0$;

2: **while** $N_{sel} < N_{des}$ **do**

3: 将图像 M 划分为 $d \times d$ blocks

4: **for** 每个 $4d \times 4d$ 蓝色 *block* **do**

for 每个 $2d \times 2d$ 绿色 *block* **do**

for 每个 $d \times d$ 红色 *blocks* **do**

遍历每一个像素，对应图像金字塔第 0 层梯度，阈值为 (thsSmoothed)，并且梯度最大的像素;

end for

if 如果红色 *block* 没有选择梯度点 **then**

在绿色 block 中选择梯度点，对应金字塔第 1 层的梯度，阈值为红色 block 的 0.75 倍;

end if

end for

if 如果绿色 *block* 中没有选择梯度点 **then**

在蓝色 block 中选择梯度点，对应图像金字塔第 2 层，阈值为绿色的 0.75 倍;

end if

end for

$N_{sel} = N_{sel} +$ 新第 0,1,2 层梯度点的数目;

end while

4.3 makePixelStatus()| 提取 1-5 层特征点

- 同样动态调节 pot 的大小，保证提取合适数目
- 每个 pot 内梯度大于阈值，且 gradx/grady/gradx-grady/gradx+grady 中有最大值则被选中

```
  if(sqgd > TH*TH)
{
float agx = fabs((float)g[1]);
if(agx > bestXX) {bestXX=agx; bestXXID=idx;}

float agy = fabs((float)g[2]);
if(agy > bestYY) {bestYY=agy; bestYYID=idx;}

float gxpy = fabs((float)(g[1]-g[2]));
if(gxpy > bestXY) {bestXY=gxpy; bestXYID=idx;}

float gxmy = fabs((float)(g[1]+g[2]));
```

```
if(gxmy > bestYX) {bestYX=gxmy; bestYXID=idx;}
}
```

4.4 pl[nl]| 构建特征点

特征点初始化，在选中的像素中，添加特征点信息 pl[nl]，表示图像金字塔第几层提取特征点，nl 表示第几层 point.

```
pl[nl].u = x+0.1;
pl[nl].v = y+0.1;
pl[nl].idepth = 1;
pl[nl].iR = 1;
pl[nl].isGood=true;
pl[nl].energy.setZero();
pl[nl].lastHessian=0;
pl[nl].lastHessian_new=0;
pl[nl].my_type= (lvl!=0) ? 1 : statusMap[x+y*wl];
```

4.5 makeNN() |Kdtree 寻找每个点的临近点与父点

寻找每个点同层最近 10 个 neighbour 以及上一层最近的父点，见图 8

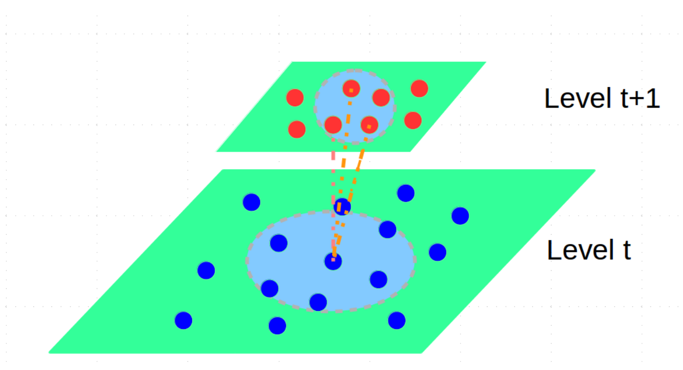

图 8: makeKNN

5 TrackFrame()| 跟踪第二帧

该函数主要进行的操作有:

- AffLight() 函数粗略估计相邻两帧的相对光度参数.
- propagateDown() 函数从金字塔顶层向底层初始化，主要是 iR.
- calcResAndGS() 函数构建相邻两帧的能量函数误差以及对应的 hessian 矩阵, 舍尔补
- calEC() 构建新旧帧逆深度以及期望值 IR
- 迭代求解状态变量, 在 dostep 函数中更新逆深度
- OptReg() 金字塔层之间的 neighbour 点中的位置对 IR 平滑处理
- propagateUp() 将逆深度从图像金字塔的底层向顶层传播

5.1 基于直接法的光度误差公式推导

这部分内容主要参考涂金戈和林突破两位大佬的公式推导 [5][6].

5.1.1 光度仿射变换

与传统 Visual SLAM 方法不同, DSO 每一帧输入图像都需要完成光度仿射变换。这是因为传统的 Visual SLAM 方法大多依赖视觉特征, 这些特征大多具有光照不变性。**而直接法则是直接使用图像的像素值, 这就导致图像亮度准确度将直接影响算法的精度和稳定性**。因此，作者通过引入光度仿射变换来对图像的亮度进行补偿。

光度仿射变换是建立前后两帧图像之间辐射值对应关系, 对应光度参数 $[a, b]$. 假设第一帧 (参考帧) 辐射值而为 I_1, 第二帧 (当前帧) 辐射值为 I_2, 那么参考帧与当前帧辐射值存在如下关系，

$$I_2\left(\mathbf{p}_2\right)=\exp(a_{21}) I_1\left(\mathbf{p}_1\right)+b_{21} \tag{13}$$

$$\exp(a_{21})=\frac{e^{a_2} \Delta t_2}{e^{a_1} \Delta t_1} \tag{14}$$

$$b_{21}=b_2-\exp(a_{21}) b_1 \tag{15}$$

其中 Δt_1,Δt_2 表示曝光的时间.

由于本文现在讨论的是初始化过程,DSO 在初始化时设置的参数是参考帧的 $[a_1, b_1] = [0, 0], a_2 = a_1$, 所以上式便可以退化如下所示:

$$\exp(a_{21}) = \frac{\Delta t_2}{\Delta t_1} \tag{16}$$

$$b_{21} = b_2 \tag{17}$$

$\exp(a_{21}) = \frac{\Delta t_2}{\Delta t_1}$ 表示前后帧相对光度变换参数. 在 DSO 初始化代码中 $a_{21} = ln(\frac{\Delta t_2}{\Delta t_1})$ 是 DSO 初始化中真正优化的光度参数.

```
if(firstFrame->ab_exposure>0 && newFrame->ab_exposure>0)
refToNew_aff_current = AffLight(logf(newFrame->ab_exposure /
    firstFrame->ab_exposure),0); // coarse approximation.
```

5.1.2 光度误差

假设参考帧坐标系下的一个空间 3D 点为 $\mathbf{P}_1$, 其对应参考坐标系下的坐标为 $\mathbf{p_1}$, 其对应的逆深度初始化为 $\rho_1 = 1$, 并且已知内参 K. 那么存在如下逆投影变换关系 $\pi^{-1}(\mathbf{x})$, 即:

$$\mathbf{P}_1 = \pi^{-1}\left(\mathbf{p}_1\right) = \mathbf{K}^{-1}\mathbf{p}_1/\rho_1 \tag{18}$$

那么当前帧中的像素坐标 $\mathbf{p}_2$ 可以写成如下的形式 (即 $\mathbf{p}_2$ 是由 $\mathbf{p}_1$ 投影而来, 投影的过程需要知道两帧之间的相对位姿 ξ_{21} 和 $\mathbf{p}_1$, 以及参考帧对应的逆深度 ρ_1)

$$\mathbf{p}_2 = f\left(p_1, \xi_{21}, \rho_1\right) = \pi\left(\exp\left(\xi_{21}^{\wedge}\right)\mathbf{P}_1\right) = \pi\left(\exp\left(\xi_{21}^{\wedge}\right)\pi^{-1}\left(\mathbf{p}_1\right)\right) \tag{19}$$

根据光照不变性假设, 我们可以定义单个像素点的误差存在如下关系:

$$r = w_h\left(I_2\left[p_2\right] - \left(exp(a_{21})I_1\left[p_1\right] + b_{21}\right)\right) \tag{20}$$

其中 w_h 是 huber 权重.

值得一提的是，DSO 算法中每个投影点是与周围 8 个点组成 Pattern, 来共同构建参差, 这 8 个点是为了方便 SSE, 在优化中共享中间点的深度.

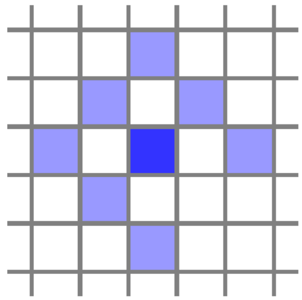

图 9: pattern 图案

对每个点的误差加权求和之后得到整体的残差能量函数 (Huber 范数形式):

$$E_{\mathbf{p}} = \sum_{\mathbf{p}_i \in \mathcal{N}(\mathbf{p})} w_{\mathbf{h}} \left\| I_2\left(\mathbf{p}_2\right) - exp(a_{21}) I_1\left(\mathbf{p}_1\right) - b_{21} \right\|_{\gamma} = \sum_{\mathbf{p}_1 \in \mathcal{N}(\mathbf{p})} w_{\mathbf{h}} H\left(r\left(\mathbf{p}_1\right)\right) \tag{21}$$

其中:

$$w_{\mathbf{h}} = \frac{c^2}{c^2 + \|\nabla I(\mathbf{p})\|_2^2} \tag{22}$$

用来平衡高梯度点的权重正常的 Huber 范数可以写成如下形式:

$$H(r) = \begin{cases} r^2/2 & ,|r| < \sigma \\ \sigma(|r| - \sigma/2) & ,|r| >= \sigma \end{cases} \tag{23}$$

在实际 DSO 代码中 σ 是一个常数，设为 9:

$$w_h = \begin{cases} 1 & ,|r| < \sigma \\ \sigma/|r| & ,|r| >= \sigma \end{cases} \tag{24}$$

5.2 光度误差函数求导

首先再次确定一下误差方程:

$$f(x) = E_{\mathbf{p}} = \sum_{\mathbf{p}_i \in \mathcal{N}(\mathbf{p})} w_{\mathbf{h}} \left\| I_2\left(\mathbf{p}_2\right) - exp(a_{21}) I_1\left(\mathbf{p}_1\right) - b_{21} \right\|_{\gamma} \tag{25}$$

由于误差函数待优化的变量有光度参数 $[a, b]$, 估计位姿 ξ_{21}, 以及逆深度 ρ_1，即状态变量为 $x = \left[\rho_1^{(1)}, \ldots, \rho_1^{(N)}, \xi_{21}, a, b\right]^T_{(N+8)\times 1} = [\mathbf{x}_\alpha, \mathbf{x}_\beta]^T$ 因此我们需要对他们分别求导

- 光度参数求导
- 位姿状态变量求导
- 逆深度求导

5.2.1 光度参数求导

$$\frac{\partial f(\mathbf{x})}{\partial a_{21}} = -w_h \exp(a_{21}) I_1(\mathbf{p}_1) \tag{26}$$

$$\frac{\partial r_{21}}{\partial b_{21}} = -w_h \tag{27}$$

5.2.2 相对位姿求导

根据链式法则, 则有:

$$\frac{\partial f(\mathbf{x})}{\partial \xi_{21}} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{p}_2} \frac{\partial \mathbf{p}_2}{\partial \xi_{21}} \tag{28}$$

设 ∇I_x,∇I_y 分别为 $\mathbf{p}_2$ 处的水平和垂直方向上的梯度，则公式 (28) 第一部分:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{p}_2} = w_h \frac{\partial I_2(\mathbf{p}_2)}{\partial \mathbf{p}_2} = w_h \left[\nabla I_x, \quad \nabla I_y\right] \tag{29}$$

公式 (28) 第二部分同样根据链式法则有：

$$\frac{\partial \mathbf{p}_2}{\partial \xi_{21}} = \frac{\partial \mathbf{p}_2}{\partial \mathbf{P}_2} \frac{\partial \mathbf{P}_2}{\partial \xi_{21}} \tag{30}$$

我们先看公式 (30) 第一部分，关于像素坐标和空间坐标的关系，通过相机模型公式 (31) 可知:

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \rho_2 \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} \tag{31}$$

$$u = f_x \frac{X_2}{Z_2} + c_x, \quad v = f_y \frac{Y_2}{Z_2} + c_y \tag{32}$$

这里 $\rho_2 = 1/Z_2$,$\mathbf{P}_2 = [X_2, Y_2, Z_2]^T$ 如果不考虑齐次项的话, 我们可以得到公式 (33):

$$\frac{\partial \mathbf{p}_2}{\partial \mathbf{P}_2} = \begin{bmatrix} \frac{\partial u_2}{\partial X_2} & \frac{\partial u_2}{\partial Y_2} & \frac{\partial u_2}{\partial Z_2} \\ \frac{\partial v_2}{\partial X_2} & \frac{\partial v_2}{\partial Y_2} & \frac{\partial v_2}{\partial Z_2} \end{bmatrix} = \begin{bmatrix} \rho_2 f_x & 0 & -\rho_2^2 f_x X_2 \\ 0 & \rho_2 f_y & -\rho_2^2 f_y Y_2 \end{bmatrix} \tag{33}$$

接下来, 我们考虑公式 (30) 第二部分，根据左乘扰动模型，给定一个微小扰动 $\Delta\mathbf{T} = \exp(\delta\xi^\wedge)$, 其中 $\delta\xi = [\delta\eta, \delta\phi]^T$ ，根据视觉 SLAM14 讲, 我们存在如下关系:

$$\begin{aligned} \frac{\partial \mathbf{P}_2}{\partial \xi} &= \lim_{\delta\xi \to 0} \frac{\exp(\delta\xi^\wedge)\mathbf{P}_2 - \mathbf{P}_2}{\delta\xi} \\ &\approx \lim_{\delta\xi \to 0} \frac{(\mathbf{I} + \delta\xi^\wedge)\mathbf{P}_2 - \mathbf{P}_2}{\delta\xi} \\ &= \lim_{\delta\xi \to 0} \frac{\delta\xi^\wedge \mathbf{P}_2}{\delta\xi} \\ &= \lim_{\delta\xi \to 0} \frac{\begin{bmatrix} \delta\phi^\wedge & \delta\eta \\ \mathbf{0}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_2 \\ 1 \end{bmatrix}}{\delta\xi} \\ &= \begin{bmatrix} \mathbf{I} & -\mathbf{P}_2^\wedge \\ \mathbf{0}^T & \mathbf{0}^T \end{bmatrix} \end{aligned} \tag{34}$$

我们将公式 (34) 求导结果展开, 可以得到公式 (35)

$$\frac{\partial \mathbf{P}_2}{\partial \xi} = \begin{bmatrix} \mathbf{I} & -\mathbf{P}_2^\wedge \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & Z_2 & -Y_2 \\ 0 & 1 & 0 & -Z_2 & 0 & X_2 \\ 0 & 0 & 1 & Y_2 & -X_2 & 0 \end{bmatrix} \tag{35}$$

我们将公式 (35), 公式 (33) 的结果带入到公式 (30), 可以得到:

$$\begin{aligned} \frac{\partial \mathbf{p}_2}{\partial \xi} &= \begin{bmatrix} \rho_2 f_x & 0 & -\rho_2^2 f_x X_2 & -\rho_2^2 f_x X_2 Y_2 & f_x + \rho_2^2 f_x X_2^2 & -\rho_2 f_x Y_2 \\ 0 & \rho_2 f_y & -\rho_2^2 f_y Y_2 & -f_y - \rho_2^2 f_y Y_2^2 & \rho_2^2 f_y X_2 Y_2 & \rho_2 f_y x_2 \end{bmatrix} \\ &= \begin{bmatrix} \rho_2 f_x & 0 & -\rho_2 f_x u_2' & -f_x u_2' v_2' & f_x + f_x u_2'^2 & -f_x v_2' \\ 0 & \rho_2 f_y & -\rho_2 f_y v_2' & -f_y - f_y v_2'^2 & f_y u_2' v_2' & f_y u_2' \end{bmatrix} \end{aligned} \tag{36}$$

其中 $u_2' = X_2/Z_2, v_2' = Y_2/Z_2$ 为归一化坐标. 最终公式 (29) 求导结果如下所示:

$$\frac{\partial f(\mathbf{x})}{\partial \xi} = w_h \begin{bmatrix} \nabla I_x \rho_2 f_x \\ \nabla I_y \rho_2 f_y \\ -\rho_2 \left(\nabla I_x f_x u_2' - \nabla I_y f_y v_2'\right) \\ -\nabla I_x f_x u_2' v_2' - \nabla I_y f_y \left(1 + v_2'\right) \\ \nabla I_x f_x \left(1 + u_2'^2\right) + \nabla I_y f_y u_2' v_2' \\ -\nabla I_x f_x v_2' + \nabla I_y f_y u_2' \end{bmatrix} \tag{37}$$

5.2.3 逆深度求导

根据链式法则

$$\frac{\partial f(\mathbf{x})}{\partial \rho_1} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{p}_2} \frac{\partial \mathbf{p}_2}{\partial \rho_1} \tag{38}$$

上式第一部分已有公式 (29) 给出，第二部分继续分解:

$$\frac{\partial \mathbf{p}_2}{\partial \rho_1} = \frac{\partial \mathbf{p}_2}{\partial \mathbf{P}_2} \frac{\partial \mathbf{P}_2}{\partial \rho_1} \tag{39}$$

其中公式 (39) 第一部分已经公式 (33) 中得到，现考虑第二部分，已知

$$\mathbf{P}_2 = \mathbf{R}\mathbf{K}^{-1}\mathbf{P}_1/\rho_1 + \mathbf{t} \tag{40}$$

那么就有:

$$\frac{\partial \mathbf{P}_2}{\partial \rho_1} = -\frac{\mathbf{R}\mathbf{K}^{-1}\mathbf{P}_1}{\rho_1^2} = -\rho_1^{-1}\left(\mathbf{P}_2 - \mathbf{t}\right) = -\rho_1^{-1}\left[X_2 - t_x \quad Y_2 - t_y \quad Z_2 - t_z\right]^T \tag{41}$$

结合公式 (33) 和（39)，公式 (41), 我们可以得到:

$$\frac{\partial \mathbf{p}_2}{\partial \rho_1} = -\rho_1^{-1}\rho_2 \begin{bmatrix} f_x \left(u_2' t_z - t_x\right) \\ f_y \left(v_2' t_z - t_y\right) \end{bmatrix} \tag{42}$$

将结果带入到公式 (38) 得:

$$\frac{\partial f(\mathbf{x})}{\partial \rho_1} = \rho_1^{-1}\rho_2 \left(\nabla I_x f_x \left(t_x - u_2' t_z\right) + \nabla I_y f_y \left(t_y - v_2' t_z\right)\right) \tag{43}$$

至此, 我们已经对所有的状态变量完成求导过程, 逆深度状态变量 $\mathbf{X}_\alpha$ 的雅克比矩阵我们用 $\mathbf{J}_\alpha = \left[\frac{\partial f(\mathbf{x})}{\partial \rho_1^{(1)}}, \ldots, \frac{\partial f(\mathbf{x})}{\partial \rho_1^{(N)}}\right]_{1\times N}$ 表示. 估计的位姿状态以及光度参数 a,b, 我们用 $\mathbf{J}_\beta = \left[\frac{\partial f(\mathbf{x})}{\partial \xi}, \frac{\partial f(\mathbf{x})}{\partial a}, \frac{\partial f(\mathbf{x})}{\partial b}\right]_{1\times 8}$. 总的雅克比矩阵表示为 $\mathbf{J} = [\mathbf{J}_\alpha, \mathbf{J}_\beta]_{1\times(N+8)}$

5.3 估计状态求解

首先我们回顾一下基础知识, 已知误差方程可以在当前状态一阶展开:

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{J}\Delta\mathbf{x} \tag{44}$$

我们让误差函数对状态增量求导, 可得:

$$\frac{\partial \frac{1}{2} f^2(\mathbf{x} + \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} = f(\mathbf{x} + \Delta\mathbf{x})^T \frac{\partial f(\mathbf{x} + \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \approx (f(\mathbf{x}) + \mathbf{J}\Delta\mathbf{x})^T \mathbf{J} \tag{45}$$

另上式等于 0, 则得到增量方程:

$$\mathbf{J}^T\mathbf{J}\Delta\mathbf{x} = -\mathbf{J}^T f(\mathbf{x}) \tag{46}$$

令 $\mathbf{H} = \mathbf{J}^T\mathbf{J}, \mathbf{g} = -\mathbf{J}^T f(\mathbf{x})$, 则可以写成:

$$\mathbf{H}\Delta\mathbf{x} = \mathbf{g} \tag{47}$$

这里我们通过舍尔补的方式对 Hessan 矩阵进行求解.

$$\begin{bmatrix} \mathbf{H}_{\alpha\alpha} & \mathbf{H}_{\alpha\beta} \\ \mathbf{H}_{\beta\alpha} & \mathbf{H}_{\beta\beta} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\alpha \\ \mathbf{x}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{g}_\alpha \\ \mathbf{g}_\beta \end{bmatrix} \tag{48}$$

其中 $\mathbf{H}_{a\alpha} = \mathbf{J}_\alpha^T\mathbf{J}_\alpha, \mathbf{H}_{\alpha\beta} = \mathbf{J}_\alpha^T\mathbf{J}_\beta, \mathbf{H}_{\beta\alpha} = \mathbf{J}_\beta^T\mathbf{J}_\alpha, \mathbf{H}_{\beta\beta} = \mathbf{J}_\beta^T\mathbf{J}_\beta, \mathbf{g}_\alpha = -\mathbf{J}_\alpha^T f(\mathbf{x}), \mathbf{g}_\beta = -\mathbf{J}_\beta^T f(\mathbf{x})$

通过舒尔补消元:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{H}_{\alpha\alpha} & \mathbf{H}_{\alpha\beta} \\ \mathbf{H}_{\beta\alpha} & \mathbf{H}_{\beta\beta} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\alpha \\ \mathbf{x}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{g}_\alpha \\ \mathbf{g}_\beta \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{H}_{\alpha\alpha} & \mathbf{H}_{\alpha\beta} \\ \mathbf{0} & -\mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1}\mathbf{H}_{\alpha\beta} + \mathbf{H}_{\beta\beta} \end{bmatrix} \begin{bmatrix} \mathbf{x}_\alpha \\ \mathbf{x}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{g}_\alpha \\ -\mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1}\mathbf{g}_\alpha + \mathbf{g}_\beta \end{bmatrix} \tag{49}$$

令 $\mathbf{H}_\phi = \mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1}\mathbf{H}_{\alpha\beta}, \mathbf{g}_\phi = \mathbf{H}_{\beta\alpha}\mathbf{H}_{\alpha\alpha}^{-1}\mathbf{g}_\alpha$

$$\mathbf{x}_\beta = \left(\mathbf{H}_{\beta\beta} - \mathbf{H}_\phi\right)^{-1}\left(\mathbf{g}_\beta - \mathbf{g}_\phi\right) \tag{50}$$

$$\mathbf{x}_\alpha = \mathbf{H}_{\alpha\alpha}^{-1}\left(\mathbf{g}_\phi - \mathbf{H}_{\alpha\beta}\mathbf{x}_\beta\right) \tag{51}$$

我们将 $\mathbf{H}_\phi, \mathbf{g}_\phi$ 展开,$\mathbf{H}_{a\alpha}^{-1} = \left(\mathbf{H}_{\mathrm{aa}}\mathbf{H}_{\mathrm{aa}}\mathbf{H}_{\mathrm{a}\alpha}^{-1}\right)^{-1} = \left(\mathbf{J}_\alpha^{\mathrm{T}}\mathbf{J}_\alpha\mathbf{J}_\alpha^T\mathbf{J}_\alpha\mathbf{H}_{\alpha\alpha}^{-1}\right)^{-1} = \left(\mathbf{J}_\alpha^T\mathbf{J}_\alpha\mathbf{H}_{\alpha\alpha}^{-1}\right)^{-1} / \left(\mathbf{J}_\alpha\mathbf{J}_\alpha^T\right) = \mathbf{I}/\left(\mathbf{J}_\alpha\mathbf{J}_\alpha^T\right)$

$$\mathbf{H}_\phi = \frac{1}{\mathbf{J}_\alpha\mathbf{J}_\alpha^T}\left(\mathbf{J}_\alpha^T\mathbf{J}_\beta\right)^T\mathbf{J}_\alpha^T\mathbf{J}_\beta \tag{52}$$

$$\mathbf{g}_\phi = -\frac{1}{\mathbf{J}_\alpha\mathbf{J}_\alpha^T}\left(\mathbf{J}_\alpha^T\mathbf{J}_\beta\right)^T\mathbf{J}_\alpha^T f(\mathbf{x}) \tag{53}$$

这里的 $\mathbf{H}_\phi, \mathbf{g}_\phi$ 就是代码中的 Hsc,bsc，$\mathbf{H}_{\beta\beta}, \mathbf{g}_\beta$ 就是代码中的 H,b，此外，DSO 在迭代的过程中设置了一个参数 lambda 来调整下一次的迭代，类似于列文伯格方法中的 ：

$$\left(\mathbf{H} + \lambda\mathbf{D}^T\mathbf{D}\right)\Delta\mathbf{x} = \mathbf{g} \tag{54}$$

DSO 为了使 $[H, b]$(见图 10) 变为对角阵，在 $[H, b]$ 右下角加了一个元素，组成了 9×9 的方阵，由于 H 矩阵具有对称性，所以只需求解 $\frac{9+1}{2}*9=45$ ，此外为了加速求解 H，b 的系数，采用了 SSE 加速，SSE 可以同时对 4 个单精度的浮点数计算，所以需要存储 45×4 个 float 数

x	y	z	$roll$	$pitch$	yaw	a	b	$b_{\alpha\alpha}$
${}^1J_x^T\,{}^1J_x$	${}^1J_x^T\,{}^1J_y$	${}^1J_x^T\,{}^1J_z$	${}^1J_x^T\,{}^1J_{roll}$	${}^1J_x^T\,{}^1J_{pitch}$	${}^1J_x^T\,{}^1J_{yaw}$	${}^1J_x^T\,{}^1J_a$	${}^1J_x^T\,{}^1J_b$	${}^1J_x^T*e_1$
	${}^1J_y^T\,{}^1J_y$	${}^1J_y^T\,{}^1J_z$	${}^1J_y^T\,{}^1J_{roll}$	${}^1J_y^T\,{}^1J_{pitch}$	${}^1J_y^T\,{}^1J_{yaw}$	${}^1J_y^T\,{}^1J_a$	${}^1J_y^T\,{}^1J_b$	${}^1J_y^T*e_1$
		${}^1J_z^T\,{}^1J_z$	${}^1J_z^T\,{}^1J_{roll}$	${}^1J_z^T\,{}^1J_{pitch}$	${}^1J_z^T\,{}^1J_{yaw}$	${}^1J_z^T\,{}^1J_a$	${}^1J_y^T\,{}^1J_b$	${}^1J_z^T*e_1$
			${}^1J_{roll}^T\,{}^1J_{roll}$	${}^1J_{roll}^T\,{}^1J_{pitch}$	${}^1J_{roll}^T\,{}^1J_{yaw}$	${}^1J_{roll}^T\,{}^1J_a$	${}^1J_{roll}^T\,{}^1J_b$	${}^1J_{roll}^T*e_1$
				${}^1J_{pitch}^T\,{}^1J_{pitch}$	${}^1J_{pitch}^T\,{}^1J_{yaw}$	${}^1J_{pitch}^T\,{}^1J_a$	${}^1J_{pitch}^T\,{}^1J_b$	${}^1J_{pitch}^T*e_1$
					${}^1J_{yaw}^T\,{}^1J_{yaw}$	${}^1J_{yaw}^T\,{}^1J_a$	${}^1J_{yaw}^T\,{}^1J_b$	${}^1J_{yaw}^T*e_1$
						${}^1J_a^T\,{}^1J_a$	${}^1J_a^T\,{}^1J_b$	${}^1J_a^T*e_1$
							${}^1J_b^T\,{}^1J_b$	${}^1J_b^T*e_1$
								e_1*e_1

图 10: $[H, b]$ 矩阵的结构

6 IntialFromInitializer()| 完成初始化

- insertFrame() 将第一帧设置成关键帧并插入能量函数中。
- setPrecalcValues() 设置 host 与 target 帧的相对位姿变换，得到估计状态增量 (包括位姿，逆深度).
- resacleFactor 对图像金字塔第 0 层求出归一化尺度
- insertPoint() 用第 0 层的点创建 pointHessian, 并插入能量方程
- firstFrame->shell 设置第一帧和最新帧的待优化量

参考文献

[1] Jakob Engel. A photometrically calibrated benchmark for monocular visual odometry. Accessed April 2, 2019.

[2] kokerf. Dso 光度标定. https://blog.csdn.net/kokerf/article/details/80170257. Accessed April 2, 2019.

[3] Shibo Zhao. Direct depth slam: Sparse geometric feature enhanced direct depth slam system for low-texture environments. https://www.mdpi.com/1424-8220/18/10/3339. Accessed April 2, 2019.

[4] Shibo Zhao. A robust laser-inertial odometry and mapping method for large-scale highway. https://youtu.be/AQJ9wFK8jwQ. Accessed April 2, 2019.

[5] 林突破. Dso 初始化. https://blog.csdn.net/xxxlinttp/article/details/89379785. Accessed April 4, 2019.

[6] 涂金戈. 直接法光度误差导数推导. https://www.cnblogs.com/JingeTU/p/8297076.html. Accessed April 3, 2019.

[7] 高博. Dso 详解. https://zhuanlan.zhihu.com/p/29177540.