

## DSO windowed optimization 代码 (2)

## 3 非 Schur Complement 部分信息计算

参考《DSO windowed optimization 公式》，非Schur Complement 部分指  $H_{XX}$  和  $J_X^T r$ 。

## 3.1 AccumulatedTopHessianSSE::addPoint() 优化的局部信息计算

EnergyFunctional::accumulateAF\_MT() 与 EnergyFunctional::accumulateLF\_MT() 遍历每一个点，对每一个点调用 AccumulatedTopHessianSSE::addPoint()，在 AccumulatedTopHessianSSE::addPoint() 中遍历点的每一个 residual。计算所有优化系统的信息，存储在每个点的局部变量和 EnergyFunctional 的局部变量中。

## 3.1.1 resApprox

首先搞定 resApprox。由 VecNRf 可知，这东西是 8x1 的矩阵（也就是每个 residual 都是八个像素点的组合）。

https://github.com/JakobEngel/dso/blob/5fb2c065b1638e10bccf049a6575ede4334ba673/src/OptimizationBackend/AccumulatedTopHessian.cpp#L72

```
VecNRf resApprox;
if(mode==0) // active
    resApprox = rJ->resF;
if(mode==2) // marginalize
    resApprox = r->res_toZeroF;
if(mode==1) // linearized
{
    // compute Jp*delta
    _m128 Jp_delta_x = _mm_set1_ps(rJ->Jpdx[0].dot(dp.head<6>())+rJ->Jpdc[0].dot(dc)+rJ->Jpdd[0]*dd);
    _m128 Jp_delta_y = _mm_set1_ps(rJ->Jpdx[1].dot(dp.head<6>())+rJ->Jpdc[1].dot(dc)+rJ->Jpdd[1]*dd);
    _m128 delta_a = _mm_set1_ps((float)(dp[6]));
    _m128 delta_b = _mm_set1_ps((float)(dp[7]));

    for(int i=0;i<patternNum;i+=4)
    {
        // PATTERN: rtz = resF - [JI*Jp Ja]*delta.
        _m128 rtz = _mm_load_ps(((float*)&r->res_toZeroF)+i);
        rtz = _mm_add_ps(rtz,_mm_mul_ps(_mm_load_ps(((float*)&rJ->JIdx)+i),Jp_delta_x));
        rtz = _mm_add_ps(rtz,_mm_mul_ps(_mm_load_ps(((float*)&rJ->JIdx)+i),Jp_delta_y));
        rtz = _mm_add_ps(rtz,_mm_mul_ps(_mm_load_ps(((float*)&rJ->JabF)+i),delta_a));
        rtz = _mm_add_ps(rtz,_mm_mul_ps(_mm_load_ps(((float*)&rJ->JabF)+i),delta_b));
        _mm_store_ps(((float*)&resApprox)+i, rtz);
    }
}
```

Residual 有三种情况：

- active 情况最简单，直接是 residual。
- marginalize 的情况比较复杂，res\_toZeroF 在 EFResidual::fixLinearizationF() 赋值，而 res\_toZeroF 与下面计算的 rtz 是类似的。
- linearized 在这里已经给出了其赋值的方法，下面会说到，linearized residual 是不存在的。

所谓的 linearied residual 是指 EFResidual::isActive() 与 EFResidual::isLinearized 都为 true 的 Residual。初始阶段 isLinearized 为 false，只要搞清楚 isLinearized 在什么时候设置为 true 就可以了解到 linearized residual 是何种意思。查找了 EFResidual::isLinearized 只在 EFResidual::fixLinearizationF 中设置为 true，而 EFResidual::fixLinearizationF() 仅仅只在 FullSystem::flagPointsForRemoval() 中调用。在此处，将那些符合 2 种情况（1. 因为 residual 太少造成了 Out Of Boundary（这里考虑到将要被 marginalize 掉的帧的影响），2. 主帧要被 marginalize 掉）的点的 residual 设置为 linearized，但是这些点紧接着又会在 EnergyFunctional::marginalizePointsF() 中被 marg 掉，被删除掉。最终也没有进入 FullSystem::optimize() 的优化过程中。我在 AccumulatedTopHessianSSE::addPoint() 的这个位置设置了 conditional breakpoint (mode==1)，或者 assert(mode!=1)，实验证明 linearized residual 是不存在的。

- active residual 时，resApprox 对应的就是简单的  $r_{21}$ 。
- linearized residual 时，还要看这个代码是什么意思。

$$\begin{bmatrix} Jp\_delta\_x \\ Jp\_delta\_y \end{bmatrix} = \frac{\partial r_2}{\partial \xi_1} \delta \xi_1 + \frac{\partial r_2}{\partial \xi_2} \delta \xi_2 + \frac{\partial r_2}{\partial C} \delta C + \frac{\partial r_2}{\partial \rho_1} \delta \rho_1$$

$$\begin{bmatrix} delta\_a \\ delta\_b \end{bmatrix} = \frac{\partial r_1}{\partial l_1} \delta l_1 + \frac{\partial r_1}{\partial l_2} \delta l_2$$

$$rtz = \frac{\partial r_1}{\partial \xi_1} \delta \xi_1 + \frac{\partial r_1}{\partial \xi_2} \delta \xi_2 + \frac{\partial r_1}{\partial C} \delta C + \frac{\partial r_1}{\partial \rho_1} \delta \rho_1 + \frac{\partial r_1}{\partial l_1} \delta l_1 + \frac{\partial r_1}{\partial l_2} \delta l_2$$

res\_toZeroF 与 rtz 相同。resApprox = res\_toZeroF + rtz。

## 3.1.2 acc

在 AccumulatedTopHessianSSE::addPoint() 函数计算了 Hessian 矩阵。而这里的 Hessian 矩阵是存储了两个帧之间的相互信息，所有的信息存储在 AccumulatedTopHessianSSE::acc 中，acc 是一个数组，大小是 8\*8 个，位置 (i, j) 上对应的是 i 帧与 j 帧的相互信息。

AccumulatorApprox 也就是 AccumulatedTopHessianSSE::acc 变量的“基础”类型。这个类型对应着 13x13 的矩阵。这个矩阵经过阅读代码，可以知道存储的是以下信息。

$$H = \begin{bmatrix} J^T \\ r^T \end{bmatrix} \begin{bmatrix} J & r \end{bmatrix}$$
$$J = \begin{bmatrix} \frac{\partial r_1}{\partial C} 8 \times 4 & \frac{\partial r_1}{\partial \xi_1} 8 \times 6 & \frac{\partial r_1}{\partial l_1} 8 \times 2 \end{bmatrix}_{8 \times 12}$$
$$r = \begin{bmatrix} r_{21} \end{bmatrix}_{8 \times 1}$$
$$H = \begin{bmatrix} J^T \\ r^T \end{bmatrix} \begin{bmatrix} J & r \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1}{\partial C}^T 4 \times 8 \\ \frac{\partial r_1}{\partial \xi_1}^T 6 \times 8 \\ \frac{\partial r_1}{\partial l_1}^T 2 \times 8 \\ r_{21}^T 1 \times 8 \end{bmatrix} \begin{bmatrix} \frac{\partial r_1}{\partial C} 8 \times 4 & \frac{\partial r_1}{\partial \xi_1} 8 \times 6 & \frac{\partial r_1}{\partial l_1} 8 \times 2 & r_{21} 8 \times 1 \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial C} 4 \times 4 & \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial \xi_1} 4 \times 6 & \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial l_1} 4 \times 2 & \frac{\partial r_1}{\partial C}^T r_{21} 4 \times 1 \\ \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial C} 6 \times 4 & \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial \xi_1} 6 \times 6 & \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial l_1} 6 \times 2 & \frac{\partial r_1}{\partial \xi_1}^T r_{21} 6 \times 1 \\ \frac{\partial r_1}{\partial l_1}^T \frac{\partial r_1}{\partial C} 2 \times 4 & \frac{\partial r_1}{\partial l_1}^T \frac{\partial r_1}{\partial \xi_1} 2 \times 6 & \frac{\partial r_1}{\partial l_1}^T \frac{\partial r_1}{\partial l_1} 2 \times 2 & \frac{\partial r_1}{\partial l_1}^T r_{21} 2 \times 1 \\ r_{21}^T \frac{\partial r_1}{\partial C} 1 \times 4 & r_{21}^T \frac{\partial r_1}{\partial \xi_1} 1 \times 6 & r_{21}^T \frac{\partial r_1}{\partial l_1} 1 \times 2 & r_{21}^T r_{21} 1 \times 1 \end{bmatrix}$$

代码中的 BotRight 对应矩阵右下角 3x3 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial \xi_1} 2 \times 2 & \frac{\partial r_1}{\partial \xi_1}^T r_{21} 2 \times 1 \\ r_{21}^T \frac{\partial r_1}{\partial \xi_1} 1 \times 2 & r_{21}^T r_{21} 1 \times 1 \end{bmatrix}$$

TopRight 对应矩阵右上角 10x3 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial \xi_1} 4 \times 2 & \frac{\partial r_1}{\partial C}^T r_{21} 4 \times 1 \\ \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial \xi_1} 6 \times 2 & \frac{\partial r_1}{\partial \xi_1}^T r_{21} 6 \times 1 \end{bmatrix}$$

Data 对应左上角 10x10 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial C} 4 \times 4 & \frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial \xi_1} 4 \times 6 \\ \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial C} 6 \times 4 & \frac{\partial r_1}{\partial \xi_1}^T \frac{\partial r_1}{\partial \xi_1} 6 \times 6 \end{bmatrix}$$

这个 AccumulatorApprox 中存储的 13x13 矩阵并不是优化过程中整体的大矩阵，只是对应着窗口中两帧之间的相互信息。注意到代码中计算调用 acc 变量时是这么调用的 acc[tid][htIDX]，int htIDX = r->hostIDX + r->targetIDX \* nframes[tid];，不考虑 tid 线程编号，acc 共有 8\*8=64 个。

继续讲完 AccumulatedTopHessianSSE::addPoint 函数。

函数的目标除了计算不同帧之间的相互信息（变量 acc），还需要计算每一个点对于所有 residual 的信息和。即 EFPPoint 中的成员变量

Hdd\_accAF, bd\_accAF, Hcd\_accAF, Hdd\_accLF, bd\_accLF, Hcd\_accLF，如果这个点是 active 点，那么设置 AF 相关的变量，否则设置 LF 相关变量，如果是 marginalize 点，清除 AF 相关变量的信息。这三个成员变量将用于计算逆深度的优化量。

局部变量 Hdd\_acc, bd\_acc, Hcd\_acc 对应着这些 EFPPoint 的成员变量，最后赋值到成员变量。

## 3.1.3 bd\_acc, Hdd\_acc, Hcd\_acc

https://github.com/JakobEngel/dso/blob/5fb2c065b1638e10bccf049a6575ede4334ba673/src/OptimizationBackend/AccumulatedTopHessian.cpp#L118

```
JI_r[0] += resApprox[i] * rJ->JIdx[0][i];
JI_r[1] += resApprox[i] * rJ->JIdx[1][i];
...
Vec2f Ji2_Jpdd = rJ->JIdx2 * rJ->Jpdd;
bd_acc += JI_r[0]*rJ->Jpdd[0] + JI_r[1]*rJ->Jpdd[1];
Hdd_acc += Ji2_Jpdd.dot(rJ->Jpdd);
Hcd_acc += rJ->Jpdc[0]*Ji2_Jpdd[0] + rJ->Jpdc[1]*Ji2_Jpdd[1];
```

JI\_r 对应  $\frac{\partial r_1}{\partial r_2} \left( \frac{\partial r_1}{\partial \xi_1} \delta \xi_1 + \frac{\partial r_1}{\partial \xi_2} \delta \xi_2 + \frac{\partial r_1}{\partial C} \delta C + \frac{\partial r_1}{\partial \rho_1} \delta \rho_1 + \frac{\partial r_1}{\partial l_1} \delta l_1 + \frac{\partial r_1}{\partial l_2} \delta l_2 \right)$ ，2x1。

Ji2\_Jpdd 对应  $\frac{\partial r_1}{\partial r_2} \frac{\partial r_1}{\partial \rho_1}$ ，2x1。

bd\_acc 对应 (1) active 时， $\frac{\partial r_1}{\partial \rho_1}^T r_{21}$ ；(2) marginalize 时， $\frac{\partial r_1}{\partial \rho_1}^T \left( \frac{\partial r_1}{\partial \xi_1} \delta \xi_1 + \frac{\partial r_1}{\partial \xi_2} \delta \xi_2 + \frac{\partial r_1}{\partial C} \delta C + \frac{\partial r_1}{\partial \rho_1} \delta \rho_1 + \frac{\partial r_1}{\partial l_1} \delta l_1 + \frac{\partial r_1}{\partial l_2} \delta l_2 \right)$ 。1x1。

Hdd\_acc 对应  $\frac{\partial r_1}{\partial \rho_1}^T \frac{\partial r_1}{\partial \rho_1}$ ，1x1。

Hcd\_acc 对应  $\frac{\partial r_1}{\partial C}^T \frac{\partial r_1}{\partial \rho_1}$ ，4x1。

## 3.2 AccumulatedTopHessianSSE::stitchDoubleInternal() 优化信息统计

循环 for(int k=min;k<max;k++) 循环是遍历所有可能的 (host\_frame,target\_frame) 组合。

内层循环累积计算 accH 就不用看了，这个循环是用于累加多个线程的结果，accH 就是 acc[h+nframes\*t]，参照 3.1。

下面的 H（对应  $H_{XX}$ ）和 b（对应  $J_X^T r$ ）的累加，使用了 EnergyFunctional::adHost 和 EnergyFunctional::adTarget。这是因为前面计算的 Jacobian 都是对相对状态的偏导，这两个变量存储的是相对状态对绝对状态的偏导。

adHost[h+nframes\*t] 下标是 (t,h)，对应公式  $\frac{\partial X_R^{(h)}}{\partial X_R}^T$ 。

adTarget[h+nframes\*t] 下标是 (t,h)，对应公式  $\frac{\partial X_R^{(h)}}{\partial X_R^{(t)}}^T$ 。

$X_R^{(i)}$  是 i 帧的所有状态，包括 se(3) 和 AffLight 参数，即  $\begin{bmatrix} \xi_i \\ l_i \end{bmatrix}$ 。

分类: SLAM



JingeTU  
关注 - 4  
粉丝 - 71

+加关注

« 上一篇: OKVIS 代码框架

» 下一篇: DSO windowed optimization 代码 (3)

posted @ 2018-03-16 22:34 JingeTU 阅读(1505) 评论(1) 编辑 收藏

## 评论列表

#1楼 2019-07-20 16:30 ByLeeBravo

有个问题想请教一下博主，是否DSO的BA部分没有使用huber? 只有普通帧的pose-only optimization使用了huber核? 我看DSO中的huber核数导一般叫做hw，没有出现在BA部分。谢谢博主~

支持(0) 反对(0)

注册用户登录后才能发表评论，请 登录 或 注册，访问 网站首页。

【推荐】了解你才能更懂你，博客园首发问卷调查，助力社区新升级  
【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库  
【推荐】独家下载电子书 | 前端必看！阿里这样实现前端代码智能生成



相关博文：

- DSO 代码框架
- VINS-Mono代码分析与总结(一) IMU预积分
- 贝叶斯优化(BayesianOptimization)深入理解
- DSO之光度标定
- MarkDown 中使用 LaTeX 数学式
- » 更多推荐...

最新 IT 新闻：

- 日烧6000万 免费模式能否颠覆To B市场?
- 明星直播带货不动货，销售额水分高达99%
- 腾讯云与工业富联合作升级，联手打造新基建领域的数字“灯塔工厂”标杆
- 抖音出海，可能比华为更难
- 理想VS蔚来，首轮PK谁赢了?
- » 更多新闻...

## 公告

昵称: JingeTU  
年龄: 3年5个月  
粉丝: 71  
关注: 4  
+加关注

2020年7月						
日	一	二	三	四	五	六
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

## 搜索

<input type="text"/>	<input type="button" value="找找看"/>
<input type="text"/>	<input type="button" value="谷歌搜索"/>

## 常用链接

我的随笔  
我的评论  
我的参与  
最新评论  
我的标签

## 随笔分类

LeetCode(7)  
SLAM(38)

## 随笔档案

2020年5月(2)

2019年12月(1)  
2019年10月(3)  
2019年8月(1)  
2019年5月(1)  
2019年4月(1)  
2018年6月(1)  
2018年5月(2)  
2018年3月(3)  
2018年1月(5)  
2017年11月(1)  
2017年10月(1)  
2017年9月(5)  
2017年8月(1)  
2017年3月(5)  
2017年2月(13)

## 最新评论

1. Re:直接法光度误差导数推导  
@OldYangtze 求导链式法则，公式 (34) 最后一行。你是问这? ...  
--JingeTU

2. Re:直接法光度误差导数推导  
大佬（36）公式是怎么得出来的?  
--OldYangtze

3. Re:DSO 代码框架  
大神你好，请教一个问题。在 trackNewestCoarse()这个函数中有一句 float extrapolFac = 1; if(lambda < lambdaExtrapolationLimit) ...  
--胡公子

4. Re:DSO windowed optimization 公式  
@daichuang 没有计算吗？你看这一行。具体的你去确认这附近的代码，我很久没看了。 ...  
--JingeTU

5. Re:DSO windowed optimization 公式  
@JingeTU 是我理解有问题，这里ksi包含8个参数（se(3)和两个光度参数）。这里还有个问题，相对光度参数delta(bij)对host帧光度参数的伴随，代码中只求取了对bi的导数，而没有求对...  
--daichuang

## 阅读排行榜

1. 矩阵求导的思考(6054)
2. DSO 代码框架(5954)
3. IMU 预积分推导(5464)
4. ORB\_SLAM2 源码阅读 ORB\_SLAM2::ORBEx tractor(4262)
5. 直接法光度误差导数推导(3545)

## 评论排行榜

1. DSO 代码框架(10)
2. DSO windowed optimization 公式(9)
3. IMU 预积分推导(6)
4. Adjoint of SE(3)(5)
5. 直接法光度误差导数推导(5)

## 推荐排行榜

1. DSO 代码框架(4)
2. 【SLAM】安装 g2o\_viewer(3)
3. 直接法光度误差导数推导(3)
4. DSO windowed optimization 代码 (1)(2)
5. DSO windowed optimization 代码 (3)(2)