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 $J_r^T r$。

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->Jpdxi[0].dot(dp.head<6>())+rJ->Jpdc[0].dot(dc)+rJ->Jpdd[0]*dd);
  __m128 Jp_delta_y = _mm_set1_ps(rJ->Jpdxi[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*)&rJ->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+1))+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+1))+i),delta_b));
    _mm_store_ps(((float*)&resApprox)+i, rtz);
  }
}
```

Residual 有三种情况：

1. active 情况最简单，直接是 residual。
2. 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。
3. linearized 在这里已经经出了其赋值的方法，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。

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

$$\begin{bmatrix} \text{Jp\_delta\_x} \\ \text{Jp\_delta\_y} \end{bmatrix} = \frac{\partial r_1}{\partial \xi_1}\delta\xi_1 + \frac{\partial r_1}{\partial \xi_2}\delta\xi_2 + \frac{\partial r_1}{\partial C}\delta C + \frac{\partial r_1}{\partial \rho_1}\delta\rho_1$$

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

$$\text{rtz} = \frac{\partial r_1}{\partial \xi_1}\delta\xi_1 + \frac{\partial r_1}{\partial \xi_2}\delta\xi_2 + \frac{\partial r_1}{\partial C}\delta C + \frac{\partial r_1}{\partial \rho_1}\delta\rho_1 + \frac{\partial r_{21}}{\partial l_1}\delta l_1 + \frac{\partial r_{21}}{\partial l_2}\delta l_2$$

`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。

$$H = \begin{bmatrix} J^T \\ r^T \end{bmatrix} \begin{bmatrix} J & r \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial r_{21}}{\partial C}_{8\times4} & \frac{\partial r_{21}}{\partial \xi_1}_{8\times6} & \frac{\partial r_{21}}{\partial \xi_2}_{8\times2} \end{bmatrix}_{8\times12}$$

$$r = \begin{bmatrix} r_{21}{}_{8\times1} \end{bmatrix}$$

$$H = \begin{bmatrix} J^T \\ r^T \end{bmatrix} \begin{bmatrix} J & r \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial r_{21}}{\partial C}^T{}_{4\times8} \\ \frac{\partial r_{21}}{\partial \xi_1}^T{}_{6\times8} \\ \frac{\partial r_{21}}{\partial \xi_2}^T{}_{2\times8} \\ r_{21}{}^T{}_{1\times8} \end{bmatrix} \begin{bmatrix} \frac{\partial r_{21}}{\partial C}_{8\times4} & \frac{\partial r_{21}}{\partial \xi_1}_{8\times6} & \frac{\partial r_{21}}{\partial \xi_2}_{8\times2} & r_{21}{}_{8\times1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial r_{21}}{\partial C}^T\frac{\partial r_{21}}{\partial C}_{4\times4} & \frac{\partial r_{21}}{\partial C}^T\frac{\partial r_{21}}{\partial \xi_1}_{4\times6} & \frac{\partial r_{21}}{\partial C}^T\frac{\partial r_{21}}{\partial \xi_2}_{4\times2} & \frac{\partial r_{21}}{\partial C}^T r_{21}{}_{4\times1} \\ \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial C}_{6\times4} & \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial \xi_1}_{6\times6} & \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial \xi_2}_{6\times2} & \frac{\partial r_{21}}{\partial \xi_1}^T r_{21}{}_{6\times1} \\ \frac{\partial r_{21}}{\partial \xi_2}^T\frac{\partial r_{21}}{\partial C}_{2\times4} & \frac{\partial r_{21}}{\partial \xi_2}^T\frac{\partial r_{21}}{\partial \xi_1}_{2\times6} & \frac{\partial r_{21}}{\partial \xi_2}^T\frac{\partial r_{21}}{\partial \xi_2}_{2\times2} & \frac{\partial r_{21}}{\partial \xi_2}^T r_{21}{}_{2\times1} \\ r_{21}{}^T\frac{\partial r_{21}}{\partial C}_{1\times4} & r_{21}{}^T\frac{\partial r_{21}}{\partial \xi_1}_{1\times6} & r_{21}{}^T\frac{\partial r_{21}}{\partial \xi_2}_{1\times2} & r_{21}{}^T r_{21}{}_{1\times1} \end{bmatrix}$$

代码中的 `BotRight` 对应矩阵右下角 3x3 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial \xi_2}^T\frac{\partial r_{21}}{\partial \xi_2}_{2\times2} & \frac{\partial r_1}{\partial \xi_2}^T r_{21}{}_{2\times1} \\ r_{21}{}^T\frac{\partial r_{21}}{\partial \xi_2}_{1\times2} & r_{21}{}^T r_{21}{}_{1\times1} \end{bmatrix}$$

`TopRight` 对应矩阵右上角 10x3 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial C}^T\frac{\partial r_{21}}{\partial \xi_2}_{4\times2} & \frac{\partial r_1}{\partial C}^T r_{21}{}_{4\times1} \\ \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial \xi_2}_{6\times2} & \frac{\partial r_{21}}{\partial \xi_1}^T r_{21}{}_{6\times1} \end{bmatrix}$$

`Data` 对应左上角 10x10 的分块：

$$\begin{bmatrix} \frac{\partial r_1}{\partial C}^T\frac{\partial r_{21}}{\partial C}_{4\times4} & \frac{\partial r_1}{\partial C}^T\frac{\partial r_{21}}{\partial \xi_1}_{4\times6} \\ \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial C}_{6\times4} & \frac{\partial r_{21}}{\partial \xi_1}^T\frac{\partial r_{21}}{\partial \xi_1}_{6\times6} \end{bmatrix}$$

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。

继续调究 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 `EFPoint` 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 `EFPoint` 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#L128

```
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 $\frac{\partial r_1}{\partial r_1}^T(\frac{\partial r_1}{\partial \xi_1}\delta\xi_1 + \frac{\partial r_1}{\partial \xi_2}\delta\xi_2 + \frac{\partial r_1}{\partial C}\delta C + \frac{\partial r_1}{\partial \rho_1}\delta\rho_1 + \frac{\partial r_{21}}{\partial l_1}\delta l_1 + \frac{\partial r_{21}}{\partial l_2}\delta l_2)$，2x1。

`JI2_Jpdd` 对应 $\frac{\partial r_1}{\partial r_1}^T\frac{\partial r_1}{\partial r_1}$，2x1。

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

`Hdd_acc` 对应 $\frac{\partial r_1}{\partial r_1}^T\frac{\partial r_1}{\partial r_1}$，1x1。

`Hcd_acc` 对应 $\frac{\partial r_1}{\partial C}^T\frac{\partial r_{21}}{\partial r_1}$，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 $J_r^T r$）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 $\frac{\partial X_t^{(n)}}{\partial X_h^{(n)}}^T$。

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

$X_h^{(i)}$ 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核导数一般叫做hw，没有出现在BA部分，谢谢博主~

支持(0) 反对(0)