DSO windowed optimization 代码 (1)

这里不想解释怎么 marginalize，什么是 First-Estimates Jacobian (FEJ)。这里只看看代码，看看Hessian矩阵是怎么构造出来的。

1 优化流程

整个优化过程，也是 Levenberg-Marquardt 的优化过程，这个优化过程在函数 FullSystem::makeKeyFrame() 中被调用，也是在确定当前帧成为关键帧，并且用当前帧激活了窗口中其他帧的 immaturePoints 之后。过程在 FullSystem::optimize() 函数中。

优化的目标值，是所有需要优化点的逆深度、相机的4个内参数、窗口中8个帧的状态量。

FullSystem::optimize() 函数流程大致如下。首先 FullSystem::linearizeAll(false) 把相关的导数计算一下。然后在所有的 residual 中找到 isLinearized 为 false 的 residual，调用 PointFrameResidual::applyRes(true)，设置它们的 PointFrameResidual::ResState （按照 FullSystem::optimizeImmaturePoint 的结果），如果是正常的点(ResState::IN)调用EFResidual::takeDataF()把 EFResidual::JpJdF 设置一下。这就算完成了优化的准备工作。

随后进入循环体，进行最高有限次的优化循环。每一次优化都有可能使得整体能量升高，所以每次优化前调用 FullSystem::backupState() 保存当前所有待优化参数的 state 和 step，FullSystem::solveSystem() 进行优化（得到的优化变化量 step），FullSystem::doStepFromBackup() 将优化结果生效，FullSystem::linearizeAll(false) 计算新的能量值，如果能量升高了，就使用 FullSystem::loadSateBackup() 将结果回滚。

在跳出循环体之后调用一次 FullSystem::linearizeAll(true)，效果是将优化之后成为 outlier 的 residual 剔除，剩下正常的 residual 调用一次 EFResidual::takeDataF() 计算新的 EFResidual::JpJdF （这里涉及到 FEJ）。注意一下 FullSystem::linearizeAll() 参数为 false 和 true 的区别。

FullSystem::solveSystem() 设置优化变化量 step 的过程在 EnergyFunctional::resubstituteF_MT() 中，能帮助理解 idepth 是如何更新的（Schur Complement 将 idepth 剔除出最终需要矩阵求逆的系统，然而这些 idepth 的优化变化量 step 是如何计算的？）。

2 导数准备

需要遍历每一个 PointFrameResidual 将与这个 Residual 相关的导数计算出来，再进行优化。而这些计算出来的相关导数被存储在 RawResidualJacobian 中。

https://github.com/JakobEngel/dso/blob/master/src/OptimizationBackend/RawResidualJacobian.h#L32

在优化过程中 Frame, PointHessian, PointFrameResidual 都有与之对应的实体，分别是 EFFrame, EFPoint, EFResidual。通过保留指针，保存层次之间的索引。

在计算 RawResidualJacobian 时，是计算 PointFrameResidual 的 J，尔后会将这个 J 转移到 EFResidual 的 J，并且计算 EFResidual::JpJdF，这个过程在 EFResidual::takeDataF() 中。所以这里把 JpJdF 的计算过程写出来，弄清 JpJdF 的意义。

https://github.com/JakobEngel/dso/blob/5fb2c065b1638e10bccf049a6575ede4334ba673/src/OptimizationBackend/EnergyFunctionalStructs.cpp#L37

```
struct RawResidualJacobian
{
        EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
        // ================== new structure: save independently =============.
        VecNRf resF; // typdef Eigen::Matrix<float,MAX_RES_PER_POINT,1> VecNRf; MAX_RES_PER_POINT == 8

        // the two rows of d[x,y]/d[xi].
        Vec6f Jpdxi[2];                 // 2x6

        // the two rows of d[x,y]/d[C].
        VecCf Jpdc[2];                  // 2x4

        // the two rows of d[x,y]/d[idepth].
        Vec2f Jpdd;                             // 2x1

        // the two columns of d[r]/d[x,y].
        VecNRf JIdx[2];                 // 9x2

        // = the two columns of d[r] / d[ab]
        VecNRf JabF[2];                 // 9x2


        // = JIdx^T * JIdx (inner product). Only as a shorthand.
        Mat22f JIdx2;                           // 2x2
        // = Jab^T * JIdx (inner product). Only as a shorthand.
        Mat22f JabJIdx;                 // 2x2
        // = Jab^T * Jab (inner product). Only as a shorthand.
        Mat22f Jab2;                    // 2x2

};
```

以上变量的类型中出现 `NR`，说明该变量是存储了每一个 pattern 点的信息。

现在将这些变量对应的导数一一列出：

1. `VecNRf resF;` 对应 r_{21}，1x8，这里的 r_{21} 是对于一个点，八个 pattern residual 组成的向量。
2. `Vec6f Jpdxi[2];` 对应 $\frac{\partial x_2}{\partial \xi_{21}}$，2x6，注意这里的 x_2 是像素坐标。（我一般把像素坐标写成 x，对应代码中的变量 `Ku`，归一化坐标写成 x'，对应代码中的变量 `u`。）
3. `VecCf Jpdc[2];` 对应 $\frac{\partial x_2}{\partial C}$，2x4，这里的 C 指相机内参 $[f_x, f_y, c_x, c_y]^T$。
4. `Vec2f Jpdd;` 对应 $\frac{\partial x_2}{\partial \rho_1}$，2x1，注意是对 host 帧的逆深度求导。
5. `VecNRf JIdx[2];` 对应 $\frac{\partial r_{21}}{\partial x_2}$，8x2，这个和 target 帧上的影像梯度相关。
6. `VecNRf JabF[2];` 对应 $\frac{\partial r_{21}}{\partial a_{21}}, \frac{\partial r_{21}}{\partial b_{21}}$，8x1，8x1。
7. `Mat22f JIdx2;` 对应 $\frac{\partial r_{21}}{\partial x_2}^T \frac{\partial r_{21}}{\partial x_2}$，2x8 8x2，2x2。
8. `Mat22f JabJIdx;` 对应 $\frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial x_2}$，2x8 8x2，2x2，这里的 l_{21} 指 $\begin{bmatrix} a_{21} \\ b_{21} \end{bmatrix}$。
9. `Mat22f Jab2;` 对应 $\frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial l_{21}}$，2x8 8x2，2x2。
10. `JpJdF` 对应 $\begin{bmatrix} \frac{\partial r_{21}}{\partial \xi_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \\ \frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \end{bmatrix}$，8x1。

在 PointFrameResidual::linearize 中对这些变量进行了计算。

https://github.com/JakobEngel/dso/blob/master/src/FullSystem/Residuals.cpp#L78

在计算时使用了投影过程中的变量，现在将这些变量与公式对应。投影过程标准公式如下：

$$\begin{aligned} x_2 &= K\rho_2(R_{21}\rho_1^{-1}K^{-1}x_1 + t_{21}) \\ &= Kx_2' \end{aligned}$$

变量的对应关系如下：

`KliP` = $K^{-1}x_1$ = x_1'

`ptp` = $R_{21}K^{-1}x_1 + \rho_1 t_{21}$ = $\rho_2^{-1}\rho_1 K^{-1}x_2$

`drescale` = $\rho_2\rho_1^{-1}$

[`u` , `v` , 1]^T = $K^{-1}x_2$ = x_2'

[`Ku` , `Kv` , 1]^T = x_2

1. `Vec2f Jpdd;` $\frac{\partial x_2}{\partial \rho_1}$

```
d_d_x = drescale * (PRE_tTll_0[0]-PRE_tTll_0[2]*u)*SCALE_IDEPTH*HCalib->fxl();
d_d_y = drescale * (PRE_tTll_0[1]-PRE_tTll_0[2]*v)*SCALE_IDEPTH*HCalib->fyl();
```

计算 $\frac{\partial x_2}{\partial \rho_1}$，这个在博客《直接法光度误差导数推导》中已经讲了如何求解。得到的结果是：

$$\begin{bmatrix} f_x\rho_1^{-1}\rho_2(t_{21}^x - u_2' t_{21}^z) \\ f_y\rho_1^{-1}\rho_2(t_{21}^y - v_2' t_{21}^z) \end{bmatrix}$$

2. `VecCf Jpdc[2];` $\frac{\partial x_2}{\partial C}$

```
d_C_x[2] = drescale*(PRE_RTll_0(2,0)*u-PRE_RTll_0(0,0));
d_C_x[3] = HCalib->fxl() * drescale*(PRE_RTll_0(2,1)*u-PRE_RTll_0(0,1)) * HCalib->fyli();
d_C_x[0] = KliP[0]*d_C_x[2];
d_C_x[1] = KliP[1]*d_C_x[3];

d_C_y[2] = HCalib->fyl() * drescale*(PRE_RTll_0(2,0)*v-PRE_RTll_0(1,0)) * HCalib->fxli();
d_C_y[3] = drescale*(PRE_RTll_0(2,1)*v-PRE_RTll_0(1,1));
d_C_y[0] = KliP[0]*d_C_y[2];
d_C_y[1] = KliP[1]*d_C_y[3];

d_C_x[0] = (d_C_x[0]+u)*SCALE_F;
d_C_x[1] *= SCALE_F;
d_C_x[2] = (d_C_x[2]+1)*SCALE_C;
d_C_x[3] *= SCALE_C;

d_C_y[0] *= SCALE_F;
d_C_y[1] = (d_C_y[1]+v)*SCALE_F;
d_C_y[2] *= SCALE_C;
d_C_y[3] = (d_C_y[3]+1)*SCALE_C;
```

链式的求导过程。

$$\frac{\partial x_2}{\partial C} = \begin{bmatrix} \frac{\partial u_2}{\partial f_x} & \frac{\partial u_2}{\partial f_y} & \frac{\partial u_2}{\partial c_x} & \frac{\partial u_2}{\partial c_y} \\ \frac{\partial v_2}{\partial f_x} & \frac{\partial v_2}{\partial f_y} & \frac{\partial v_2}{\partial c_x} & \frac{\partial v_2}{\partial c_y} \end{bmatrix}$$

$$x_2 = Kx_2'$$

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_2' \\ v_2' \\ 1 \end{bmatrix}$$

$$\begin{aligned} u_2 &= f_x u_2' + c_x \\ v_2 &= f_y v_2' + c_y \end{aligned}$$

$$\begin{aligned} \frac{\partial u_2}{\partial f_x} &= u_2' + f_x\frac{\partial u_2'}{\partial f_x} & \frac{\partial u_2}{\partial f_y} &= f_x\frac{\partial u_2'}{\partial f_y} \\ \frac{\partial u_2}{\partial c_x} &= f_x\frac{\partial u_2'}{\partial c_x} + 1 & \frac{\partial u_2}{\partial c_y} &= f_x\frac{\partial u_2'}{\partial c_y} \end{aligned}$$

$$\begin{aligned} \frac{\partial v_2}{\partial f_x} &= f_y\frac{\partial v_2'}{\partial f_x} & \frac{\partial v_2}{\partial f_y} &= v_2' + f_y\frac{\partial v_2'}{\partial f_y} \\ \frac{\partial v_2}{\partial c_x} &= f_y\frac{\partial v_2'}{\partial c_x} & \frac{\partial v_2}{\partial c_y} &= f_y\frac{\partial v_2'}{\partial c_y} + 1 \end{aligned}$$

先求 $\frac{\partial x_2'}{\partial C}$，再使用链式法则求 $\frac{\partial x_2}{\partial C}$。

$$\begin{aligned} x_2' &= \rho_2\rho_1^{-1}(R_{21}K^{-1}x_1 + \rho_1 t_{21}) \\ &= \rho_2\rho_1^{-1}R_{21}K^{-1}x_1 + \rho_2 t_{21} \\ &= \rho_2\rho_1^{-1}\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} f_x^{-1} & 0 & -f_x^{-1}c_x \\ 0 & f_y^{-1} & -f_y^{-1}c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} + \rho_2 t_{21} \\ &= \rho_2\rho_1^{-1}\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} f_x^{-1}(u_1 - c_x) \\ f_y^{-1}(v_1 - c_y) \\ 1 \end{bmatrix} + \rho_2 \begin{bmatrix} t_{21}^x \\ t_{21}^y \\ t_{21}^z \end{bmatrix} \end{aligned}$$

$$\begin{aligned} u_2' &= \frac{\rho_2\rho_1^{-1}(r_{11}f_x^{-1}(u_1 - c_x) + r_{12}f_y^{-1}(v_1 - c_y) + r_{13}) + \rho_2 t_{21}^x}{\rho_2\rho_1^{-1}(r_{31}f_x^{-1}(u_1 - c_x) + r_{32}f_y^{-1}(v_1 - c_y) + r_{33}) + \rho_2 t_{21}^z} = \frac{A}{C} \\ v_2' &= \frac{\rho_2\rho_1^{-1}(r_{21}f_x^{-1}(u_1 - c_x) + r_{22}f_y^{-1}(v_1 - c_y) + r_{23}) + \rho_2 t_{21}^y}{\rho_2\rho_1^{-1}(r_{31}f_x^{-1}(u_1 - c_x) + r_{32}f_y^{-1}(v_1 - c_y) + r_{33}) + \rho_2 t_{21}^z} = \frac{B}{C} \end{aligned}$$

u_2', v_2' 的分母的计算结果都为 1，但是这个计算过程和内参 f_x, f_y, c_x, c_y 都有关系。求导过程不能省略分母，感谢某同学指出我这里认知上的错误。（为方便表达，我用字母替代分子、分母，$C = 1$。）

求导示例：

$$\begin{aligned} \frac{\partial u_2'}{\partial f_x} &= \frac{\partial A}{\partial f_x}\frac{1}{C} + A\frac{1}{C^2}(-1)\frac{\partial C}{\partial f_x} \\ &= \rho_2\rho_1^{-1}r_{11}(u_1 - c_x)f_x^{-2}(-1)\frac{1}{C} - \frac{A}{C}\frac{1}{C}\rho_2\rho_1^{-1}r_{31}(u_1 - c_x)f_x^{-2}(-1) \\ &= \frac{1}{C}(\rho_2\rho_1^{-1}r_{11}(u_1 - c_x)f_x^{-2}(-1) + \rho_2\rho_1^{-1}r_{31}(u_1 - c_x)f_x^{-2}u_2') \\ &= \rho_2\rho_1^{-1}(r_{31}u_2' - r_{11})f_x^{-2}(u_1 - c_x) \end{aligned}$$

同理得到以下结果：

$$\begin{aligned} \frac{\partial u_2'}{\partial f_x} &= \rho_2\rho_1^{-1}(r_{31}u_2' - r_{11})f_x^{-2}(u_1 - c_x) & \frac{\partial u_2'}{\partial f_y} &= \rho_2\rho_1^{-1}(r_{32}u_2' - r_{12})f_y^{-2}(v_1 - c_y) \\ \frac{\partial u_2'}{\partial c_x} &= \rho_2\rho_1^{-1}(r_{31}u_2' - r_{11})f_x^{-1} & \frac{\partial u_2'}{\partial c_y} &= \rho_2\rho_1^{-1}(r_{32}u_2' - r_{12})f_y^{-1} \end{aligned}$$

$$\begin{aligned} \frac{\partial v_2'}{\partial f_x} &= \rho_2\rho_1^{-1}(r_{31}v_2' - r_{21})f_x^{-2}(u_1 - c_x) & \frac{\partial v_2'}{\partial f_y} &= \rho_2\rho_1^{-1}(r_{32}v_2' - r_{22})f_y^{-2}(v_1 - c_y) \\ \frac{\partial v_2'}{\partial c_x} &= \rho_2\rho_1^{-1}(r_{31}v_2' - r_{21})f_x^{-1} & \frac{\partial v_2'}{\partial c_y} &= \rho_2\rho_1^{-1}(r_{32}v_2' - r_{22})f_y^{-1} \end{aligned}$$

链式：

$$\begin{aligned} \frac{\partial u_2}{\partial f_x} &= u_2' + f_x\frac{\partial u_2'}{\partial f_x} \\ &= u_2' + \rho_2\rho_1^{-1}(r_{31}u_2' - r_{11})f_x^{-1}(u_1 - c_x) \\ \frac{\partial u_2}{\partial f_y} &= f_x\frac{\partial u_2'}{\partial f_y} \\ &= f_x f_y^{-1}\rho_2\rho_1^{-1}(r_{32}u_2' - r_{12})f_y^{-1}(v_1 - c_y) \\ \frac{\partial u_2}{\partial c_x} &= f_x\frac{\partial u_2'}{\partial c_x} + 1 \\ &= \rho_2\rho_1^{-1}(r_{31}u_2' - r_{11}) + 1 \\ \frac{\partial u_2}{\partial c_y} &= f_x\frac{\partial u_2'}{\partial c_y} \\ &= f_x f_y^{-1}\rho_2\rho_1^{-1}(r_{32}u_2' - r_{12}) \end{aligned}$$

$$\begin{aligned} \frac{\partial v_2}{\partial f_x} &= f_y\frac{\partial v_2'}{\partial f_x} \\ &= f_y f_x^{-1}\rho_2\rho_1^{-1}(r_{31}v_2' - r_{21})f_x^{-1}(u_1 - c_x) \\ \frac{\partial v_2}{\partial f_y} &= v_2' + f_y\frac{\partial v_2'}{\partial f_y} \\ &= v_2' + \rho_2\rho_1^{-1}(r_{32}v_2' - r_{22})f_y^{-1}(v_1 - c_y) \\ \frac{\partial v_2}{\partial c_x} &= f_y\frac{\partial v_2'}{\partial c_x} \\ &= f_y f_x^{-1}\rho_2\rho_1^{-1}(r_{31}v_2' - r_{21}) \\ \frac{\partial v_2}{\partial c_y} &= f_y\frac{\partial v_2'}{\partial c_y} + 1 \\ &= \rho_2\rho_1^{-1}(r_{32}v_2' - r_{22}) + 1 \end{aligned}$$

3. `Vec6f Jpdxi[2];` $\frac{\partial x_2}{\partial \xi_{21}}$

```
d_xi_x[0] = new_idepth*HCalib->fxl();
d_xi_x[1] = 0;
d_xi_x[2] = -new_idepth*u*HCalib->fxl();
d_xi_x[3] = -u*v*HCalib->fxl();
d_xi_x[4] = (1+u*u)*HCalib->fxl();
d_xi_x[5] = -v*HCalib->fxl();

d_xi_y[0] = 0;
d_xi_y[1] = new_idepth*HCalib->fyl();
d_xi_y[2] = -new_idepth*v*HCalib->fyl();
d_xi_y[3] = -(1+v*v)*HCalib->fyl();
d_xi_y[4] = u*v*HCalib->fyl();
d_xi_y[5] = u*HCalib->fyl();
```

计算 $\frac{\partial x_2}{\partial \xi_{21}}$，这个在博客《直接法光度误差导数推导》中已经讲了如何求解。得到的结果是：

$$\frac{\partial x_2}{\partial \xi_{21}} = \begin{bmatrix} f_x\rho_2 & 0 & -f_x\rho_2 u_2' & -f_x u_2' v_2' & f_x(1 + u_2'^2) & -f_x v_2' \\ 0 & f_y\rho_2 & -f_y\rho_2 v_2' & -f_y(1 + v_2'^2) & f_y u_2' v_2' & f_y u_2' \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4. `VecNRf JIdx[2];` 对应 $\frac{\partial r_{21}}{\partial x_2}$

```
J->JIdx[0][idx] = hitColor[1];
J->JIdx[1][idx] = hitColor[2];
```

计算 $\frac{\partial r_{21}}{\partial x_2}$，这个在博客《直接法光度误差导数推导》中已经讲了如何求解。得到的结果是：

$$\frac{\partial r_{21}}{\partial x_2} = w_h\frac{\partial I_2[x_2]}{\partial x_2} = w_h\,[\,g_x, g_y\,]$$

注意代码中这个变量是8维。

5. `VecNRf JabF[2];` $\frac{\partial r_{21}}{\partial a_{21}}, \frac{\partial r_{21}}{\partial b_{21}}$

```
float drdA = (color[idx]-b0);
...
J->JabF[0][idx] = drdA*hw;
J->JabF[1][idx] = hw;
```

计算 $\frac{\partial r_{21}}{\partial l_{21}}$，这个在博客《直接法光度误差导数推导》中已经讲了如何求解。得到的结果是（结果有点不符合，需要再对照一下）：

$$\begin{aligned} \frac{\partial r_{21}}{\partial a_{21}} &= -w_h e^{a_{21}} I_1[x_1] \\ \frac{\partial r_{21}}{\partial b_{21}} &= -w_h \end{aligned}$$

10. `JpJdF` 对应 $\begin{bmatrix} \frac{\partial r_{21}}{\partial \xi_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \\ \frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \end{bmatrix}$

```
void EFResidual::takeDataF()
{
        std::swap<RawResidualJacobian*>(J, data->J);

        Vec2f JI_JI_Jd = J->JIdx2 * J->Jpdd;

        for(int i=0;i<6;i++)
                JpJdF[i] = J->Jpdxi[0][i]*JI_JI_Jd[0] + J->Jpdxi[1][i] * JI_JI_Jd[1];

        JpJdF.segment<2>(6) = J->JabJIdx*J->Jpdd;
}
```

1. `JI_JI_Jd` 对应 $\frac{\partial r_{21}}{\partial x_2}^T \frac{\partial r_{21}}{\partial \rho_1}$，2x8 8x1，2x1。
2. `JpJdF.segment<6>(0)` 对应 $\frac{\partial x_2}{\partial \xi_{21}}^T \frac{\partial r_{21}}{\partial x_2}^T \frac{\partial r_{21}}{\partial \rho_1} = \frac{\partial r_{21}}{\partial \xi_{21}}^T \frac{\partial r_{21}}{\partial \rho_1}$，6x2 2x8 8x1，6x1。
3. `JpJdF.segment<2>(6)` 对应 $\frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial x_2} \frac{\partial x_2}{\partial \rho_1} = \frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial \rho_1}$，2x8 8x2 2x1，2x1。
4. `JpJdF` 对应 $\begin{bmatrix} \frac{\partial r_{21}}{\partial \xi_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \\ \frac{\partial r_{21}}{\partial l_{21}}^T \frac{\partial r_{21}}{\partial \rho_1} \end{bmatrix}$，8x1。

分类: SLAM

« 上一篇： DSO 代码框架
» 下一篇： OKVIS 代码框架

posted @ 2018-01-31 21:55 JingeTU 阅读(2436) 评论(1) 编辑 收藏

评论列表

#1楼 2019-09-22 01:31 Ah_Xiang

前辈你好，关于雅克比有一个小问题。代码中有一些scale量，应该是用来平衡不同优化量间是数量级的，但是直接输出Jacobian量来看，各个数量级在Jxxx层面也并不统一。想请问下，是这些scale的设置主要是为了追求哪个层面的数值平衡呢？在仿写代码时，应该如何去调整这些scale数值呢？谢谢！

支持(0) 反对(0)

公告

昵称： JingeTU
园龄： 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

找找看
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 extrapFac = 1; if(lambda < lambdaExtrapolationLimit) ...
--胡公子
4. Re:DSO windowed optimization 公式
@daichujiang 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...
--daichujiang

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::ORBextractor(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)