第五章作业思路分享

主讲人 曹永

纲要

- 第一部分：公式推导
- 第二部分：代码实现

公式推导

- 内参模型（按照开源代码中的定义）

a.安装误差T（代码原先使用上三角形式，改为下三角形式）,刻度系数误差K,零偏B

$$T=\begin{bmatrix}1 & 0 & 0\\ \alpha_{xz} & 1 & 0\\ -\alpha_{xy} & \alpha_{yx} & 1\end{bmatrix} \qquad K=\begin{bmatrix}\mathrm{s}_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & s_z\end{bmatrix} \qquad B=\begin{bmatrix}b_x\\ b_y\\ b_z\end{bmatrix}$$

b.待估计参数

$$\theta^{acc}=[\alpha_{xz}\quad \alpha_{xy}\quad \alpha_{yx}\quad s_x\quad s_y\quad s_z\quad b_x\quad b_y\quad b_z]$$

公式推导

- 内参模型（按照开源代码中的定义）

c.给定加速度读数为 X ，对应的真实值为 $X^{'}$ ，其计算公式如下：

$$X^{'}=T*K*(X-B)$$

公式推导

- 残差

$$f(\theta^{acc}) = \|g\|_2 - \|X'\|_2$$

- 雅可比，按照链式求导分解

$$\frac{\partial f}{\partial \theta^{acc}} = \frac{\partial f}{\partial \|X'\|_2} \frac{\partial \|X'\|_2}{\partial X'} \frac{\partial X'}{\partial \theta^{acc}}$$

公式推导

深蓝学院
shenlanxueyuan.com

- 雅可比，逐项求导

$$X = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} \qquad X^{'} = T * K * (X - B) = \begin{bmatrix} s_x(A_x - b_x) \\ \alpha_{xz} s_x (A_x - b_x) + s_y (A_y - b_y) \\ -\alpha_{xy} s_x (A_x - b_x) + \alpha_{yx} s_y (A_y - b_y) + s_z (A_z - b_z) \end{bmatrix}$$

$$\frac{\partial f}{\partial \left\| X^{'} \right\|_2} = \frac{\partial (\left\| g \right\|_2 - \left\| X^{'} \right\|_2)}{\partial \left\| X^{'} \right\|_2} = -1 \qquad \frac{\partial \left\| X^{'} \right\|_2}{\partial X^{'}} = \frac{X^{'}}{\left\| X^{'} \right\|_2}$$

$$\frac{\partial X^{'}}{\partial \theta^{acc}} = \begin{bmatrix} 0 & 0 & 0 & A_x - b_x & 0 & 0 & -s_x & 0 & 0 \\ s_x(A_x - b_x) & 0 & 0 & \alpha_{xz}(A_x - b_x) & A_y - b_y & 0 & -\alpha_{xz} s_x & -s_y & 0 \\ 0 & -s_x(A_x - b_x) & s_y(A_y - b_y) & -\alpha_{xy}(A_x - b_x) & \alpha_{yx}(A_y - b_y) & A_z - b_z & \alpha_{xy} s_x & -\alpha_{yx} s_y & -s_z \end{bmatrix}$$

代码实现

- 自动求导，修改MultiPosAccResidual为下三角形式

```
//   // TODO: implement lower triad model here
//   //
//   // mis_yz, mis_zy, mis_zx:
//   params[0], params[1], params[2],
//   // mis_xz, mis_xy, mis_yx:
//   _T2(0), _T2(0), _T2(0),
//   //    s_x,     s_y,     s_z:
//   params[3], params[4], params[5],
//   //    b_x,     b_y,     b_z:
//   params[6], params[7], params[8]
// );
CalibratedTriad_<_T2> calib_triad( _T2(0), _T2(0), _T2(0),
                                   params[0], params[1], params[2],
                                   params[3], params[4], params[5],
                                   params[6], params[7], params[8]);

// apply undistortion transform:
Eigen::Matrix< _T2, 3 , 1> calib_samp = calib_triad.unbiasNormalize( raw_samp );

residuals[0] = _T2 (g_mag_) - calib_samp.norm();
```

代码实现

深蓝学院
shenlanxueyuan.com

- 修改calibrateAcc中的参数为下三角形式

```
//
// TODO: implement lower triad model here
//
acc_calib_params[0] = init_acc_calib_.misXZ();
acc_calib_params[1] = init_acc_calib_.misXY();
acc_calib_params[2] = init_acc_calib_.misYX();

acc_calib_params[3] = init_acc_calib_.scaleX();
acc_calib_params[4] = init_acc_calib_.scaleY();
acc_calib_params[5] = init_acc_calib_.scaleZ();

acc_calib_params[6] = init_acc_calib_.biasX();
acc_calib_params[7] = init_acc_calib_.biasY();
acc_calib_params[8] = init_acc_calib_.biasZ();
```

代码实现

深蓝学院 shenlanxueyuan.com

- 修改calibrateAcc中的参数为下三角形式

```
//    // TODO: implement lower triad model here
//    //
//    // mis_yz, mis_zy, mis_zx:
//    params[0], params[1], params[2],
//    // mis_xz, mis_xy, mis_yx:
//    _T2(0), _T2(0), _T2(0),
//    //    s_x,    s_y,    s_z:
//    params[3], params[4], params[5],
//    //    b_x,    b_y,    b_z:
//    params[6], params[7], params[8]
// );
CalibratedTriad_<_T2> calib_triad( _T2(0), _T2(0), _T2(0),
                                   params[0], params[1], params[2],
                                   params[3], params[4], params[5],
                                   params[6], params[7], params[8]);

// apply undistortion transform:
Eigen::Matrix< _T2, 3 , 1> calib_samp = calib_triad.unbiasNormalize( raw_samp );

residuals[0] = _T2 (g_mag_) - calib_samp.norm();
```

代码实现

深蓝学院
shenlanxueyuan.com

- 解析求导，残差计算

```cpp
virtual bool Evaluate(double const* const* params, double *residuals, double **jacobians) const {
    const double Txz = params[0][0];
    const double Txy = params[0][1];
    const double Tyx = params[0][2];
    const double Sx = params[0][3];
    const double Sy = params[0][4];
    const double Sz = params[0][5];
    const double bx = params[0][6];
    const double by = params[0][7];
    const double bz = params[0][8];

    // 下三角模型
    // 安装误差矩阵
    Eigen::Matrix<double, 3, 3> T;
    T << 1   , 0   , 0,
         Txz , 1   , 0,
         -Txy, Tyx , 1;

    // 刻度误差系数矩阵
    Eigen::Matrix3d K = Eigen::Matrix3d::Identity();
    K(0, 0) = Sx;
    K(1, 1) = Sy;
    K(2, 2) = Sz;

    // 偏差向量
    Eigen::Vector3d bias(bx, by, bz);

    Eigen::Matrix<double,3,1 > sample(double(sample_(0)),double(sample_(1)),double(sample_(2)));

    Eigen::Vector3d calib_samp = T * K * (sample.col(0) - bias);
    residuals[0] = double (g_mag_) - calib_samp.norm();
```

代码实现

- 解析求导，雅可比计算

```cpp
    if (jacobians != nullptr) {
        if (jacobians[0] != nullptr) {
            Eigen::Vector3d samp_norm = calib_samp / (calib_samp.norm());
            Eigen::Vector3d unbias_samp = sample.col(0) - bias;
            Eigen::Matrix<double, 3, 9> J_theta = Eigen::Matrix<double, 3, 9>::Zero();
            J_theta(0, 3) = unbias_samp(0);
            J_theta(0, 6) = -Sx;
            J_theta(1, 0) = Sx * unbias_samp(0);
            J_theta(1, 3) = Txz * unbias_samp(0);
            J_theta(1, 4) = unbias_samp(1);
            J_theta(1, 6) = -Txz * Sx;
            J_theta(1, 7) = -Sy;
            J_theta(2, 1) = -Sx * unbias_samp(0);
            J_theta(2, 2) = Sy * unbias_samp(1);
            J_theta(2, 3) = -Txy * unbias_samp(0);
            J_theta(2, 4) = Tyx * unbias_samp(1);
            J_theta(2, 5) = unbias_samp(2);
            J_theta(2, 6) = Txy * Sx;
            J_theta(2, 7) = -Tyx * Sy;
            J_theta(2, 8) = -Sz;
            Eigen::Matrix<double, 1, 9> J_se3 = - samp_norm.transpose() * J_theta;
            jacobians[0][0] = J_se3(0, 0);
            jacobians[0][1] = J_se3(0, 1);
            jacobians[0][2] = J_se3(0, 2);
            jacobians[0][3] = J_se3(0, 3);
            jacobians[0][4] = J_se3(0, 4);
            jacobians[0][5] = J_se3(0, 5);
            jacobians[0][6] = J_se3(0, 6);
            jacobians[0][7] = J_se3(0, 7);
            jacobians[0][8] = J_se3(0, 8);
        }
    }
    return true;
}
```

附：

深蓝学院
shenlanxueyuan.com

The Euclidean norm of a vector $\mathbf{x}$ is represented by $||\mathbf{x}||_2 = \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}$ where, $\mathbf{x} = [x_1, x_2, \ldots x_n]^\top$, a column vector. The norm is a scalar value. The derivative of a scalar with respect to the vector $\mathbf{x}$ must result in a vector (similar to a gradient of a function from $f: R^n \to R$). To estimate the derivative of a scalar with respect to a vector, we estimate the partial derivative of the scalar with respect to each component of the vector and arrange the partial derivatives to form a vector. The derivative is represented by the grad operator ∇

$$\nabla_{\mathbf{x}} ||\mathbf{x}||_2 = [\frac{\partial}{\partial x_1} ||\mathbf{x}||_2, \frac{\partial}{\partial x_2} ||\mathbf{x}||_2, \ldots \frac{\partial}{\partial x_n} ||\mathbf{x}||_2]^\top$$

The i^{th} component of the derivative is given by:
$\frac{\partial}{\partial x_i} ||\mathbf{x}|| = \frac{\partial}{\partial x_i} \sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)} = \frac{1}{2} \frac{2x_i}{(x_1^2 + x_2^2 + \ldots + x_n^2)^{1/2}} = \frac{x_i}{\sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}}$. Since $\frac{d}{dx} f(x)^n = n f(x)^{n-1} \frac{d}{dx} f(x)$. Putting all the partial derivatives (x_i) together, we get,

$$\nabla_{\mathbf{x}} ||\mathbf{x}||_2 = \frac{\mathbf{x}}{||\mathbf{x}||_2}$$

在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听!

Thanks for Listening