这是标题

陈烁龙

2022 年 9 月 28 日

目录

插图

表格

1 OpenMVG 相机位姿

1.1 相机位姿表示

在 OpenMVG 库中，相机的位姿是通过结构体 *Pose3* 来表示的，其由两个成员构成：表示位置的 *center* 和表示姿态的 *rotation*，但是代码里并没有说明其位姿转换的方向 (矩阵的含义)。

事实上，*center* 等同于 ${}^{W}\boldsymbol{p}_{C}$，*rotation* 等同于 ${}^{C}_{W}\boldsymbol{R}$，有：

$$ {}^{C}\boldsymbol{p}_{i} = {}^{C}_{W}\boldsymbol{R}({}^{W}\boldsymbol{p}_{i} - {}^{W}\boldsymbol{p}_{C}) $$

即：

$$ \begin{cases} {}^{C}\boldsymbol{p}_{i} = {}^{C}_{W}\boldsymbol{R}{}^{W}\boldsymbol{p}_{i} + \boldsymbol{t} \\ \quad \boldsymbol{t} = -{}^{C}_{W}\boldsymbol{R}{}^{W}\boldsymbol{p}_{C} \end{cases} $$

换句话说，我们可以通过 $\left[{}^{C}_{W}\boldsymbol{R}|\boldsymbol{t}\right]$ 将世界坐标系下的点投影到相机坐标系下。其逆变换为：

$$ \begin{cases} {}^{C}_{W}\boldsymbol{R}^{-1}{}^{C}\boldsymbol{p}_{i} - {}^{C}_{W}\boldsymbol{R}^{-1}\boldsymbol{t} = {}^{W}\boldsymbol{p}_{i} \\ \quad {}^{C}_{W}\boldsymbol{R}^{-1}({}^{C}\boldsymbol{p}_{i} - \boldsymbol{t}) = {}^{W}\boldsymbol{p}_{i} \end{cases} $$

1.2 复合位姿

另外，复合两个位姿：

$$ {}^{C_i}_{C_j}\boldsymbol{T} = \begin{bmatrix} {}^{C_i}_{C_j}\boldsymbol{R} & -{}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}\boldsymbol{p}_{C_i} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} $$

$$ {}^{C_j}_{W}\boldsymbol{T} = \begin{bmatrix} {}^{C_j}_{W}\boldsymbol{R} & -{}^{C_j}_{W}\boldsymbol{R}{}^{W}\boldsymbol{p}_{C_j} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} $$

$$ {}^{C_i}_{C_j}\boldsymbol{T}{}^{C_j}_{W}\boldsymbol{T} = \begin{bmatrix} {}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}_{W}\boldsymbol{R} & -{}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}_{W}\boldsymbol{R}{}^{W}\boldsymbol{p}_{C_j} - {}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}\boldsymbol{p}_{C_i} \\ \boldsymbol{0}_{1\times3} & 1 \end{bmatrix} $$

$$ \begin{cases} {}^{C_i}_{W}\boldsymbol{R} = {}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}_{W}\boldsymbol{R} \\ {}^{C_i}_{W}\boldsymbol{R}^{-1} = {}^{W}_{C_j}\boldsymbol{R}{}^{C_j}_{C_i}\boldsymbol{R} \\ {}^{W}\boldsymbol{p}_{C_i} = {}^{C_i}_{W}\boldsymbol{R}^{-1}({}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}_{W}\boldsymbol{R}{}^{W}\boldsymbol{p}_{C_j} + {}^{C_i}_{C_j}\boldsymbol{R}{}^{C_j}\boldsymbol{p}_{C_i}) \\ \qquad = {}^{W}\boldsymbol{p}_{C_j} + {}^{W}_{C_j}\boldsymbol{R}{}^{C_j}\boldsymbol{p}_{C_i} \\ \qquad = {}^{W}\boldsymbol{p}_{C_j} + {}^{C_j}_{W}\boldsymbol{R}^{-1}{}^{C_j}\boldsymbol{p}_{C_i} \end{cases} $$

这也就是下面代码块的含义了：

Listing 1: 位姿复合

```
Pose3 operator * ( const Pose3& P ) const
{
    return {
        rotation_ * P.rotation_,
        P.center_ + P.rotation_.transpose() * center_
    };
}
```

不过，这有什么含义呢？我们已知第 j 个坐标系相对于世界坐标系的位姿和第 j 个坐标系相对于第 i 个坐标系，那么，我们就可以复合得到第 i 个坐标系相对于世界坐标系的位姿。

1.3 归化参考坐标系

另外，由于使用 *Ceres* 库进行解算时，网是不受控的 (BA 算法会整体调整网结构)。所以一般没有相机的位姿和世界坐标系对齐。但是在 V-SLAM 中，我们又一般都以第一帧相机作为世界坐标系参考。所以再完成 SfM 结算后，我们需要将所有的数据 (相机位姿，控制点，路标等) 转到第一帧坐标系下。如果用 c 表示世界到相机的变换，那么：

$$ {}^{C_i}_{C_0}\boldsymbol{T} = {}^{C_i}_{W}\boldsymbol{T} \times {}^{W}_{C_0}\boldsymbol{T} = {}^{C_i}_{W}\boldsymbol{T} \times {}^{C_0}_{W}\boldsymbol{T}^{-1} $$

也就是说，我们需要在每一个相机的位姿基础上右乘一个参考相机位姿的逆。对于路标而言：

$$ {}^{C_0}\boldsymbol{p}_{i} = {}^{C_0}_{W}\boldsymbol{T}{}^{W}\boldsymbol{p}_{i} $$

也就是说，我们需要对路标左乘一个参考相机的位姿。

好在上面的一切都已经有一个函数实现了，就是：

Listing 2: 位姿复合

```
void ApplySimilarity
( const geometry::Similarity3 & sim,
  SfM_Data & sfm_data,
  bool transform_priors );
```

你要将哪个相机作为参考系，只需要把这个相机的位子传到 sim 参数里就行了。