GTSAM在视觉里程计中的应用

单目相机位姿估计 `CameraResectioning.cpp`

1 问题描述

已知：相机内参、世界坐标系3d点、3d点对应的图像平面观测

求解：相机位姿

2 基本概念回顾

A. 相机内参

下面公式包含了相机直角坐标系->归一化平面坐标系->像素坐标系的转换

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \overset{\text{def}}{=} \frac{1}{Z} \boldsymbol{K} \boldsymbol{P}. \tag{5.6}$$

引入世界坐标系 $P = RP_w + t$

$$Z\boldsymbol{P}_{uv} = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P}_{\mathrm{w}} + \boldsymbol{t}) = \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_{\mathrm{w}}. \tag{5.8}$$

一般标定内参的代码中除了 `fx` 、 `fy` 、 `cx` 、 `cy` 以外，还有个 `s` 参数，这个主要是表示坐标系xy轴非正交产生的误差

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

B. 重投影

$$\boldsymbol{T}^* = \arg\min_{\boldsymbol{T}} \frac{1}{2} \sum_{i=1}^{n} \left\| \boldsymbol{u}_i - \frac{1}{s_i} \boldsymbol{K} \boldsymbol{T} \boldsymbol{P}_i \right\|_2^2. \tag{7.36}$$

该问题的误差项，是将 3D 点的投影位置与观测位置作差，所以称为**重投影误差**。使用齐次坐标时，这个误差有 3 维。不过，由于 $\boldsymbol{u}$ 最后一维为 1，该维度的误差一直为零，因而我们更多时候使用非齐次坐标，于是误差就只有 2 维了。如图 7-14 所示，我们通过特征匹配知道了 p_1 和 p_2 是同一个空间点 P 的投影，但是不知道相机的位姿。在初始值中，P 的投影 $\hat{p}_2$ 与实际的 p_2 之间有一定的距离。于是我们调整相机的位姿，使得这个距离变小。不过，由于这个调整需要考虑很多个点，所以最后的效果是整体误差的缩小，而每个点的误差通常都不会精确为零。

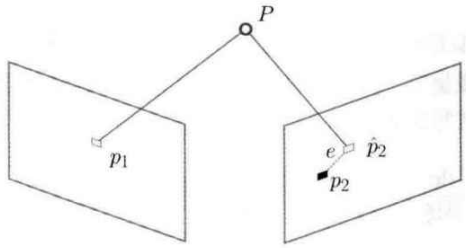

图 7-14　重投影误差示意图

残差为2维，待优化变量为6维（相机位姿），雅可比矩阵为2*6，求解时使用链式法则

数。利用链式法则，可以列写如下：

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = \lim_{\delta \boldsymbol{\xi} \to 0} \frac{\boldsymbol{e}(\delta \boldsymbol{\xi} \oplus \boldsymbol{\xi}) - \boldsymbol{e}(\boldsymbol{\xi})}{\delta \boldsymbol{\xi}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} \frac{\partial \boldsymbol{P}'}{\partial \delta \boldsymbol{\xi}}. \tag{7.42}$$

这里的 $\oplus$ 指李代数上的左乘扰动。第一项是误差关于投影点的导数，在式 (7.41) 中已经列出了变量之间的关系，易得

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} = -\begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}. \tag{7.43}$$

P' 是相机直角坐标系下的表示，既先将残差对*相机直角系坐标*求导，再将*相机直角系坐标*对*位姿*求导

而第二项为变换后的点关于李代数的导数，根据 4.3.5 节中的推导，得

$$\frac{\partial (\boldsymbol{T}\boldsymbol{P})}{\partial \delta \boldsymbol{\xi}} = (\boldsymbol{T}\boldsymbol{P})^{\odot} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{P}'^{\wedge} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} \end{bmatrix}. \tag{7.44}$$

而在 $\boldsymbol{P}'$ 的定义中，我们取出了前 3 维，于是得

$$\frac{\partial \boldsymbol{P}'}{\partial \delta \boldsymbol{\xi}} = [\boldsymbol{I}, -\boldsymbol{P}'^{\wedge}]. \tag{7.45}$$

将这两项相乘，就得到了 2×6 的雅可比矩阵：

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}. \tag{7.46}$$

详见十四讲第7讲

3 代码讲解

3.1 相机参数设置

```
Cal3_S2::shared_ptr calib(new Cal3_S2(1, 1, 0, 50, 50));
```

`class GTSAM_EXPORT Cal3_S2 : public Cal3`

A. 关于 `GTSAM_EXPORT` 说明，详见 *`Using-GTSAM-EXPORT.md`*

在windows平台，从外部访问需要显示的声明库中的所有函数

Using GTSAM_EXPORT:

On Windows it is necessary to explicitly export all functions from the library which should be externally accessible. To do this, include the macro GTSAM_EXPORT in your class or function definition.

For example:

```
class GTSAM_EXPORT MyClass { ... };

GTSAM_EXPORT myFunction();
```

B. `Cal3_S2` 继承于 `Cal3` ， `Cal3` 用于表示相机内参， `Cal3_S2` 在此基础上添加了一些3d-2d坐标系转换的实现

C. 参数说明

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
// Cal3_S2构造函数
Cal3_S2(double fx, double fy, double s, double u0, double v0)
    : Cal3(fx, fy, s, u0, v0) {}

// Cal3构造函数
Cal3(double fx, double fy, double s, double u0, double v0)
    : fx_(fx), fy_(fy), s_(s), u0_(u0), v0_(v0) {}

// Cal3内参构建
virtual Matrix3 K() const {
  Matrix3 K;
  K << fx_, s_, u0_, 0.0, fy_, v0_, 0.0, 0.0, 1.0;
  return K;
```

```
14 }
```

3.2 自定义因子

```
1   // 噪声模型 因子id 相机内参 像素观测 3d路标点
2   graph.emplace_shared<ResectioningFactor>(measurementNoise, X(1), calib,
3           Point2(55, 45), Point3(10, 10, 0));
```

下面分析一下自定义因子编写

```
1  class ResectioningFactor: public NoiseModelFactor1<Pose3> {
2    typedef NoiseModelFactor1<Pose3> Base;
3
4    Cal3_S2::shared_ptr K_; ///< camera's intrinsic parameters
5    Point3 P_;              ///< 3D point on the calibration rig
6    Point2 p_;              ///< 2D measurement of the 3D point
7
8  public:
9
10   /// Construct factor given known point P and its projection p
11   ResectioningFactor(const SharedNoiseModel& model, const Key& key,
12       const Cal3_S2::shared_ptr& calib, const Point2& p, const Point3& P) :
13       Base(model, key), K_(calib), P_(P), p_(p) {
14   }
15
16   /// evaluate the error
17   Vector evaluateError(const Pose3& pose, boost::optional<Matrix&> H =
18       boost::none) const override {
19     PinholeCamera<Cal3_S2> camera(pose, *K_);
20     return camera.project(P_, H, boost::none, boost::none) - p_;
21   }
22 };
```

A. boost::optional C++17新特性，可以理解为指针，用来表达无效值

https://www.codenong.com/cs110825744/

B. 待优化变量 `Pose3` ，由旋转矩阵+位移构成

```
1   // 待优化变量初始化(先旋转后平移)
2   Values initial;
3   initial.insert(X(1),
```

```
4       Pose3(Rot3(1, 0, 0, 0, -1, 0, 0, 0, -1), Point3(0, 0, 2)));
```

C. 残差 & 雅可比求解

evaluateError编写规则：return残差、雅可比存入H

残差是将3d landmark投影到图像平面和图像平面作差，`project(...)`会返回`Point2`类型，与`p_`一致

```
1 // P_ 世界系下的3d点，p_ 2d观测，
2 return camera.project(P_, H, boost::none, boost::none) - p_;
```

下面分析重投影部分：

看**调用关系**：

`PinholeCamera::project`(`camera.project(P_, H, boost::none, boost::none)`)调用1

-> `_project`(`_project(pw, Dpose, Dpoint, Dcal)`)调用2 求解H矩阵(Dpose)

-> `PinholeBase::project2` 调用2.1求解H后半部分

-> `pose().transformTo` 调用2.1.1

-> `PinholeBase::Project` 调用2.1.2

-> `PinholeBase::Dpose` 调用2.1.3

-> `calibration().uncalibrate`(`Cal3_S2::uncalibrate`)调用2.2求解H前半部分

值得注意的是，这里待优化变量的顺序和求导方式与十四讲有所不同

首先将(7.16)的内参部分提取出来

将这两项相乘，就得到了 2×6 的雅可比矩阵：

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}. \tag{7.46}$$

$$\begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} * \begin{bmatrix} \frac{1}{Z'} & 0 & -\frac{X'}{Z'^2} & -\frac{X'Y'}{Z'^2} & 1+\frac{X'^2}{Z'^2} & -\frac{Y'}{Z'} \\ 0 & \frac{1}{Z'} & -\frac{Y'}{Z'^2} & -1-\frac{Y'^2}{Z'^2} & \frac{X'Y'}{Z'^2} & \frac{X'}{Z'} \end{bmatrix}$$

$X'Y'Z'$ 是相机直角坐标系下表示，将其换成归一化坐标系(uv1)和逆深度(d)表示

$$\begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} * \begin{bmatrix} d & 0 & -ud & -uv & 1+u^2 & -v \\ 0 & d & -vd & -1-v^2 & uv & u \end{bmatrix}$$

代码中是预测-观测（回看 `evaluateError`），而十四讲是观测-预测，所以差了个负号（其实无所谓，计算的是二范数）

$$\begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} * \begin{bmatrix} -d & 0 & ud & uv & -1-u^2 & v \\ 0 & -d & vd & 1+v^2 & -uv & -u \end{bmatrix}$$

代码中是先旋转后平移，十四讲是先平移后旋转，所以0-2列和3-5列需要互换

```
uv, -1-uu, v, -d, 0, d * u
1+vv, -uv, -u, 0, -d, d * v
```

4 结果

```
tioning
Final result:

Values with 1 values:
Value x1: (gtsam::Pose3)
R: [
        0.8, 0, 0;
        0, -1.2, 1.17484e-16;
        0, -2.9371e-17, -0.3
]
t:           0 -1.4057e-14     6.92406
```

双目相机位姿估计 `StereoVOExample.cpp`

1 问题描述

已知：相机内参、基线、左右相机观测到的landmark像素坐标

求解：相机位姿、landmark坐标

2 代码讲解

A. 固定第一个顶点在原点

```
    // graph.emplace_shared<NonlinearEquality<Pose3> >(1, Pose3());
```

```
2   graph.emplace_shared<PriorFactor<Pose3>>(1, Pose3());
```

`Pose3()` 默认构造函数旋转矩阵为单位阵，平移为0

```
1 Pose3() : R_(traits<Rot3>::Identity()), t_(traits<Point3>::Identity()) {}
2
3 static Fixed Identity() { return Fixed::Zero();}
```

B. 设置相机内参

```
1 // 单目(fx fy s cx cy)
2 Cal3_S2::shared_ptr calib(new Cal3_S2(1, 1, 0, 50, 50));
3
4 // 双目(fx fy s cx cy b)
5 const Cal3_S2Stereo::shared_ptr K(
6     new Cal3_S2Stereo(1000, 1000, 0, 320, 240, 0.2));
```

C. 构建观测

双目观测结构体

```
1 // 即v(纵轴)相同,u不同
2 StereoPoint2(double uL, double uR, double v) :
3     uL_(uL), uR_(uR), v_(v) {
4 }
```

双目因子结构体

```
1   /**
2    * Constructor
3    * @param measured 双目量测
4    * @param model 噪声模型
5    * @param poseKey 位姿key
6    * @param landmarkKey
7    * @param K the constant calibration 相机内参
8    * @param body_P_sensor 相机body之间的外参,默认单位阵
9    */
10 graph.emplace_shared<GenericStereoFactor<Pose3,Point3> >(StereoPoint2(520, 480,
```

3 结果

```
Final result:

Values with 5 values:
Value 1: (gtsam::Pose3)
R: [
        1, 0, 0;
        0, 1, 0;
        0, 0, 1
]
t: 0 0 0

Value 2: (gtsam::Pose3)
R: [
        1, 4.04802e-17, -7.81373e-16;
        -4.04803e-17, 1, -1.00645e-15;
        7.81373e-16, 1.00645e-15, 1
]
t:  6.13491e-13 -6.06151e-13            1

Value 3: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        1;
        1;
        5
]

Value 4: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        -1;
        1;
        5
]

Value 5: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        -6.5639e-17;
        -0.5;
        5
]
```

固定起点

```
Final result:

Values with 5 values:
Value 1: (gtsam::Pose3)
R: [
        0.99997, -0.00167097, -0.00754129;
        0.00173733, 0.99996, 0.00880227;
        0.00752628, -0.00881511, 0.999933
]
t:    0.04467 -0.0482035  0.0228357

Value 2: (gtsam::Pose3)
R: [
        0.99997, -0.00167097, -0.00754129;
        0.00173733, 0.99996, 0.00880227;
        0.00752628, -0.00881511, 0.999933
]
t:  0.0371287 -0.0394012    1.02277

Value 3: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        1.00526;
        0.997505;
        5.02121
]

Value 4: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        -0.994678;
        0.99403;
        5.00616
]

Value 5: (Eigen::Matrix<double, 3, 1, 0, 3, 1>)
[
        0.00779903;
        -0.504172;
        5.02691
]
```

不固定起点