从零开始手写VIO 第一课作业

边城量子 2019.6.14

1. VIO文献阅读

- 阅读VIO相关综述文档如a, 回答如下问题:
  - 视觉与IMU进行融合之后有何优势?
  - 有哪些常见的视觉+IMU融合方案? 有没有工业界应用的例子?
  - 在学术界, VIO研究有哪些新进展?有没有将学习方法用到VIO中的例子?

参考文献1: Jianjun Gui et al. “A review of visual inertial odometry from filtering and optimisation perspectives”. In: Advanced Robotics 29.20 (2015), 1289–1301. issn: 0169-1864. doi: {10.1080/01691864.2015.1057616}.

回答:

---

2. 四元数和李代数更新

- 课件提到了可以使用四元数或旋转矩阵存储旋转变量, 当我们用计算出来的ω对某旋转更新时,有两种不同方式: $\mathbf{R} \leftarrow \mathbf{R}\exp(\boldsymbol{\omega}^{\wedge})$ $\mathbf{q} \leftarrow \mathbf{q} \otimes \left[1, \frac{1}{2}\omega\right]^{\mathrm{T}}$
- 请编程验证对于小量$\omega = [0.01, 0.02, 0.03]^{\mathrm{T}}$, 两种方法得到的结果非常接近, 实践当中可视为等同. 因此在后文提到旋转时, 我们并不刻意区分旋转本身是$\mathbf{q}$还是$\mathbf{R}$, 也不区分其更新方式为上式的哪一种。

回答:

代码和执行结果如下。 使用了四种方式计算exp(w^), 分别是AngleAxisd, 泰勒展开, 罗德里格斯公式, Sophus库，然后和四元数方式进行比较，把更新后的结果都换成矩阵形式并打印，发现两者的误差很小。

```
/*
Function:
    使用四种方式计算 exp(w^),  验证如下结论:
    在w为小量时(w=[0.01,0.02,0.03]), 以下两种方式对旋转的更新结果接近:
          (1)  R <- R exp(w^)
          (2)  q <- q ✕ [1,0.5w].transpose()
     四种方式计算exp(w^):
        1.AngleAxisd的toRotationMatrix()
        2.泰勒展开取前4项
        3.罗德里格斯公式
        4.Sophus库的exp()函数和matrix()函数
    基本原理:
       增量四元数 q' =[cos0.5θ,n*sin0.5θ],
       当θ很小时,q'趋于 [1, 0.5w ],其中w为旋转向量(注意,此处w并非单位向量)
*/
// for M_PI definition on VC++ compiler
#define  _USE_MATH_DEFINES
#include <iostream>
#include <Eigen/Core>
```

```cpp
#include <Eigen/Geometry>
#include <Sophus/SO3.hpp>
#include <cmath>      // for cos, sin, M_PI

using namespace std;

// 方式1: 使用AngleAxisd计算exp(w^)
// 原理: 选装向量w生成AngleAxisd,调用.toRotationMatrix()方法
//   w: 旋转向量
//   return:  旋转向量对应的旋转矩阵
Eigen::Matrix3d caculateMatrixByAngleAxisd(Eigen::Vector3d w) {
    // 向量模
    double norm = sqrt(w(0) * w(0) + w(1) * w(1) + w(2) * w(2));
    // w归一化
    Eigen::Vector3d w_norm(w(0) / norm, w(1) / norm, w(2) / norm);
    // 定义AngleAxisd
    Eigen::AngleAxisd v_rot = Eigen::AngleAxisd(norm, w_norm);
    // AngeleAxisd to RotationMatrix
    return v_rot.toRotationMatrix();
}

// 方式2: 使用泰勒展开计算exp(w^), 计算量大,不推荐,推荐方式3代替之
// 原理: 使用exp的泰勒表达式展开, 忽略高阶项求和
//   w: 旋转向量
//   return: 旋转向量对应的旋转矩阵
Eigen::Matrix3d caculateMatrixByTaylor(Eigen::Vector3d w) {
    // 得到w^
    Eigen::Matrix3d w_;
    w_ << 0, -w(2), w(1),
    w(2), 0, -w(0),
    -w(1), w(0), 0;
    // 通过直接泰勒展开求exp(w^), 省略4次及以上高阶项及
    Eigen::Matrix3d I = Eigen::Matrix3d::Identity(3,3);
    Eigen::Matrix3d  m_rot = I + w_ + w_ * w_ / 2
                      + w_ * w_ * w_ / (3 * 2)
                      + w_*w_*w_*w_/(4*3*2);
    return m_rot;
}

// 方式3: 使用罗德里格斯公式计算exp(w^)
// 原理:   exp(w^) = cosθI + (1 - cosθ)αα^ T + sinθα^
//   w:旋转向量
//   return: 旋转向量对应的旋转矩阵
Eigen::Matrix3d caculateMatrixByRodrigues(Eigen::Vector3d w) {
    // 罗德里格斯公式 exp(\phi^) = cosθI + (1 - cosθ)αα ^ T + sinθα ^
    Eigen::Matrix3d I = Eigen::Matrix3d::Identity();
    // 角度等于w模长
    double norm = sqrt(w(0) * w(0) + w(1) * w(1) + w(2) * w(2));
    // w归一化
    Eigen::Vector3d w_norm(w(0) / norm, w(1) / norm, w(2) / norm);
    // 定义α^对应的矩阵
    Eigen::Matrix3d m_alpha;
    m_alpha <<   0, -w_norm(2),  w_norm(1),
        w_norm(2),    0,  -w_norm(0),
        -w_norm(1), w_norm(0),  0;
    Eigen::Matrix3d  m_rot = cos(norm) * I
                           + (1 - cos(norm)) * w_norm * w_norm.transpose()
                           + sin(norm) * m_alpha;
```

```
    return m_rot;
}

// 方式4: 使用Sophus计算exp(w^)
// 原理:  exp(w)得到SO3, 调用matrix()转为矩阵
//   w:旋转向量
//   return: 旋转向量对应的旋转矩阵
Eigen::Matrix3d caculateMatrixBySophus(Eigen::Vector3d w){
    Sophus::SO3<double> SO3_w = Sophus::SO3<double>::exp(w);
    return SO3_w.matrix();
}

int main(int argn, char** argv) {
    // 任意定义原始R向量, 设置为绕z轴旋转PI/4的一个向量
    Eigen::AngleAxisd angle = Eigen::AngleAxisd(M_PI / 4, Eigen::Vector3d(0, 0,
1));
    // 转成旋转矩阵
    Eigen::Matrix3d R = angle.toRotationMatrix();
    // 转成四元数
    Eigen::Quaterniond Qr = Eigen::Quaterniond(R);

    // 定义w向量
    Eigen::Vector3d w(0.01, 0.02, 0.03);

    // 四种方式计算旋转矩阵 exp(w^): AngleAxisd, 泰勒展开, 罗德里格斯, Sophus
    // w转成AngleAxisd,调用 toRotationMatrix()方法
    Eigen::Matrix3d w_exp01 = caculateMatrixByAngleAxisd(w);
    // 泰勒展开 exp(w^)
    Eigen::Matrix3d w_exp02 = caculateMatrixByTaylor(w);
    // 罗德里格斯公式
    Eigen::Matrix3d w_exp03 = caculateMatrixByRodrigues(w);
    // Sophus库exp()后matrix()
    Eigen::Matrix3d w_exp04 = caculateMatrixBySophus(w);

    // 使用各个旋转矩阵 w_exp 更新R, 得到R_01,...,R_04
    Eigen::Matrix3d R_01 = R * w_exp01;
    Eigen::Matrix3d R_02 = R * w_exp02;
    Eigen::Matrix3d R_03 = R * w_exp03;
    Eigen::Matrix3d R_04 = R * w_exp04;

    // 使用w定义四元数
    Eigen::Quaterniond q = Eigen::Quaterniond(1, 0.5 * w(0), 0.5 * w(1), 0.5 *
w(2));
    // 使用四元数更新Qr(即R), 得到q_update
    Eigen::Matrix3d q_update = (Qr*q ).normalized().toRotationMatrix();

    // 显示旋转矩阵更新(4种) 和 四元数更新 的结果对比
    cout << "Matrix R updated by rotationMatrix: " << endl;
    cout << "----------------------------------------" << endl;
    cout << "R*w_exp01: use  AngleAxiesd" << endl << R_01 << endl;
    cout << "----------------------------------------" << endl;
    cout << "R*w_exp02: use Taylor " << endl << R_02 << endl;
    cout << "----------------------------------------" << endl;
    cout << "R*w_exp03: use Rodrigues " << endl << R_03 << endl;
    cout << "----------------------------------------" << endl;
    cout << "R*w_exp04: use Sophus " << endl << R_04 << endl;
    cout << endl  <<  "Quaternion Qr updated by another quaternion: " << endl;
    cout << "========================================" << endl;
```

```
    cout << "q*q_delta: use quaternion"  << endl << q_update << endl;

    return 0;
}
```

执行结果如下， 可以看出矩阵方式和四元数方式的更新结果非常接近；

```

Matrix R updated by rotationMatrix:
---------------------------------------
R*w_exp01: use  AngleAxiesd
  0.685368  -0.727891  0.0211022
  0.727926   0.685616 0.00738758
-0.0198454  0.0102976    0.99975
---------------------------------------
R*w_exp02: use Taylor
  0.685368  -0.727891  0.0211022
  0.727926   0.685616 0.00738758
-0.0198454  0.0102976    0.99975
---------------------------------------
R*w_exp03: use Rodrigues
  0.685368  -0.727891  0.0211022
  0.727926   0.685616 0.00738758
-0.0198454  0.0102976    0.99975
---------------------------------------
R*w_exp04: use Sophus
  0.685368  -0.727891  0.0211022
  0.727926   0.685616 0.00738758
-0.0198454  0.0102976    0.99975

Quaternion Qr updated by another quaternion:
=======================================
q*q_delta: use quaternion
  0.685371  -0.727888  0.0210998
  0.727924   0.685618 0.00738668
-0.0198431  0.0102964    0.99975
```

3.其他导数

使用右乘 $\mathfrak{so}(3)$,推导以下导数：

$$\frac{\mathrm{d}\left(\mathbf{R}^{-1}\mathbf{p}\right)}{\mathrm{d}\mathbf{R}}$$

$$\frac{\mathrm{d}\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\mathrm{d}\mathbf{R}_2}$$

回答:

使 用 性 质 $(\exp(\phi^{\wedge}))^{-1}=\exp(-\phi)^{\wedge}$, 具 体 过 程 如 下 ：

$$
\begin{aligned}
\frac{\mathrm{d}\left(\mathbf{R}^{-1}\mathbf{p}\right)}{\mathrm{d}\mathbf{R}} &= \lim_{\phi\to 0}\frac{\left(\mathbf{R}\exp(\phi^{\wedge})\right)^{-1}\mathbf{p}-\mathbf{R}^{-1}\mathbf{p}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\left(\exp(\phi^{\wedge})\right)^{-1}\mathbf{R}^{-1}\mathbf{p}-\mathbf{R}^{-1}\mathbf{p}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\exp(-\phi)^{\wedge}\mathbf{R}^{-1}\mathbf{p}-\mathbf{R}^{-1}\mathbf{p}}{\phi} \\
&\approx \lim_{\phi\to 0}\frac{\left(\mathbf{I}+(-\phi)^{\wedge}\right)\mathbf{R}^{-1}\mathbf{p}-\mathbf{R}^{-1}\mathbf{p}}{\phi} \\
&= \lim_{\phi\to 0}\frac{(-\phi)^{\wedge}\mathbf{R}^{-1}\mathbf{p}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\left(\mathbf{R}^{-1}\mathbf{p}\right)^{\wedge}\phi}{\phi} \\
&= \left(\mathbf{R}^{-1}\mathbf{p}\right)^{\wedge}
\end{aligned}
$$

以下方式针对 $\exp()\mathbf{R}_2^{-1}$ 使用伴随性质，得到的 J_r 雅可比，具体过程如下：

$$
\begin{aligned}
\frac{\mathrm{d}\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\mathrm{d}\mathbf{R}_2} &= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\left(\mathbf{R}_2\exp(\phi^{\wedge})\right)^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\left(\exp(\phi^{\wedge})\right)^{-1}\mathbf{R}_2^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\exp(-\phi^{\wedge})\mathbf{R}_2^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\exp\left(\mathbf{R}_2\left(-\phi\right)\right)^{\wedge}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}-\mathbf{J}_r^{-1}\left(\ln(\mathbf{R}_1\mathbf{R}_2^{-1})\right)\mathbf{R}_2\phi-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= -\mathbf{J}_r^{-1}\left(\ln(\mathbf{R}_1\mathbf{R}_2^{-1})^{\vee}\right)\mathbf{R}_2
\end{aligned}
$$

以下方式针对 $\mathbf{R}_1\exp()$ 使用伴随性质，得到的 J_l 雅可比，具体过程如下：

$$
\begin{aligned}
\frac{\mathrm{d}\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\mathrm{d}\mathbf{R}_2} &= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\left(\mathbf{R}_2\exp(\phi^{\wedge})\right)^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\left(\exp(\phi^{\wedge})\right)^{-1}\mathbf{R}_2^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\mathbf{R}_1\exp(-\phi^{\wedge})\mathbf{R}_2^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{\ln\left(\exp\left(-\mathbf{R}_1\phi\right)^{\wedge}\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}-\ln\left(\mathbf{R}_1\mathbf{R}_2^{-1}\right)^{\vee}}{\phi} \\
&= \lim_{\phi\to 0}\frac{-\mathbf{J}_l^{-1}(\ln(\mathbf{R}_1\mathbf{R}_2^{-1})^{\vee})\mathbf{R}_1\phi}{\phi} \\
&= -\mathbf{J}_l^{-1}(\ln(\mathbf{R}_1\mathbf{R}_2^{-1})^{\vee})\mathbf{R}_1
\end{aligned}
$$