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:

- 视觉与 IMU 进行融合之后有何优势?
  - 单一传感器不能适用所有场景, 例如视觉在玻璃、白墙等特征较少的场景无法工作；IMU 的特点是长时间使用有很大的累计误差，但是在短时间内，其相对位移数据又有很高的精度，所有当视觉传感器失效时，融合 IMU 数据能够提高定位的精度；视觉和惯性测量存在互补特性；而且在大多数智能手机中这两种传感器也都有；具体来说， 有如下一些优点：
    - 视觉与 IMU 的融合，可以借助 IMU 较高的采样频率， 进而提高系统的输出频率；
    - 视觉与 IMU 的融合， 可以有效的消除 IMU 的暂时积分漂移和交阵 IMU 的 Bias；
    - 视觉与 IMU 的融合， 可以提高视觉的鲁棒性，例如视觉 SLAM 因为某些运动或场景出现错误时；
    - 单目与 IMU 的融合，可以有效解决单目尺度不可观测的问题，且轻便小巧限制少。相比之下， 激光测距器太重；红外传感对太阳光敏感；声纳测量距离受限；双目相机极限太短对远处距离无法准确测量；
- 有哪些常见的视觉+IMU 融合方案? 有没有工业界应用的例子?
  - VINS Mono：紧耦合；前端使用 Harris+光流，有回环检测；它是基于非线性优化器优化一个滑窗内的关键帧， 帧间通过鲁棒的角点关联。在系统初始化阶段，通过松耦合的方式融合多种传感器；在重定位阶段，通过紧耦合的方式融合传感器；在优化之前，通过 IMU 预积分减少计算量；还提供了基于 4DoF 的位子图优化和回环检测； https://github.com/HKUST-Aerial-Robotics/VINS-Mono
  - OKVIS：紧耦合；前端使用多尺度 Harris 提取特征点，并用 BRISK 作为描述子，后端利用 Ceres 通过非线性优化完成状态估计。视觉误差是重投影，无回环检测；它利用非线性优化一个滑窗内的关键帧，其损失函数包括带权重的投影差和带权重的惯导误差。 http://ethz-asl.github.io/okvis/
  - MSCKF: 紧耦合；前端使用 fast+光流，后端采用 EKF 滤波，视觉误差是重投影，无回环检测；核心心思路使用帧间的几何多视约束，且不需要保存 3D 点的状态信息。 https://github.com/daniilidis-group/msckf_mono
  - ROVIO：紧耦合； 前端使用 fast+光度，后端使用 IEKF 滤波进行状态估计，无回环检测；使用 FAST 提取角点，所有角点通过图像块进行描述，并通过视频流获取了多层次表达，利用 IMU 估计的位子来计算特征投影后的光度误差，并将其用于后续优化。它是基于单目开发的 VIO 系统。https://github.com/ethz-asl/rovio
  - SVO+MSF: 松耦合；无回环检测； MSF 是基于 EKF 的多传感器融合算法。SVO 是一个轻量的直接法 VO。SVO 的前端是利用 FAST 提取角点，并通过最小化照度误差进行状态优化。MSF 将 SVO 的状态信息与 IMU 进行融合，从而得到更优的状态估计结果。

    - SVO+GTSAM：松耦合；无回环检测；和 SVO+MSF 类似，后端是通过因子图优化 iSAM2 的方式融合在一起。SVO 和 GTSAM 分别由开源代码，暂时还没有公开的整合系统。
    - VI ORB SLAM2：紧耦合，前端使用 ORB，后端是非线性优化，视觉误差采用重投影，有回环检测。https://github.com/jingpang/LearnVIORB
    - VIO 在 AR/VR 领域的应用: 例如 Apple 推出的 ARKit 实现了虚拟场景与摄像头真实场景的叠加效果，SLAM 在其中用于固定虚拟物体相对真实环境中的位置，通过 VIO 实现移动设别在空间中的精确定位，保证视角变化时，虚拟物体和真实环境的相对位置不变化；
- 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?
    - ICE-BA：前端是 FAST+光流，在后端采用一种增量式 BA，尝试利用先前的优化结果来减少新计算量，提出通过测量矩阵的 QR 分解来解决优化问题，每个新的优化迭代仅更新分解结果的一小部分而不是分解整个图，因为即使重新计算，得到的 Jacobian 也没什么变化；提出逐步恢复估计和协方差。H. Liu. ICE-BA: Incremental, consisten and efficient bundle adjustment for visual-inertial slam. 2018. CVPR.
    - 神经网络学习方法: 2019年,Visual-Inertial Mapping with Non-Linear Factor Recovery, 通过重建非线因子图，将回环约束加入到因子图中，进行全局非线性优化, 并不直接进行imu预积分，而是将高帧率的视觉惯性信息包含从非线优化边缘化中所提取的因子中. https://link.zhihu.com/?target=https%3A//gitlab.com/VladyslavUsenko/basalt
    - 神经网络学习方法: Shamwell, E. Jared, et al. "Unsupervised Deep Visual-Inertial Odometry with Online Error Correction for RGB-D Imagery." IEEE transactions on pattern analysis and machine intelligence (2019). 学会了在没有惯性测量单元（IMU）内在参数或IMU和摄像机之间的外部校准的情况下执行视觉惯性里程计（VIO）,网络学习整合IMU测量并生成假设轨迹，然后根据相对于像素坐标的空间网格的缩放图像投影误差的雅可比行列式在线校正。
    - 神经网络学习方法: Chen, Changhao, et al. "Selective Sensor Fusion for Neural Visual-Inertial Odometry." arXiv preprint arXiv:1903.01534 (2019). CVPR2019已经接收, 在如何学习多传感器融合策略上。提出了一种针对单目VIO的端到端的多传感器选择融合策略, 提出了两种基于不同掩蔽策略(masking strategies)的融合模态：确定性软融合和随机硬融合，并与先前提出的直接融合baseline进行比较.
    - Apple神经网络与 VIO 的结合： Apple 专利“利用机器学习计算系统来映射和追踪位置信息”。通常的 SLAM 技术对于真实环境的理解是通过传感器的数据建立真实环境的缩放几何模型，无需预先理解环境；Apple 专利声称能够通过机器学习的方式对图像帧进行测量，确定位置/方向等信息，并生成多个特征模型以估计 VIO 系统的健康值，每个特征模型包括至少一个与 VIO 特征测量值有关的可调特征模型参数，然后该方法将特征健康值与该组图像帧相关联的 ground truth 健康分数作比较，以确定特征健康值与 ground truth 健康分数之间的误差。基于这些误差，该方法能够更新一个或多个特征模型参数；https://patent.yivian.com/3892.html

---

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>
#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) {
    // 备注: 此处也可以直接调用Eigen::Vector3d 的norm和normalized来求模和归一化
    // 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();
    // 备注：此处也可以直接调用Eigen::Vector3d 的norm和normalized来求模和归一化
    // 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：

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

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：

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