深蓝学院 VIO 课程作业

温焕宇

2019.6.12

1 题一

1.1 视觉与 IMU 进行融合之后有何优势？

答：a) 从特性方面：i.IMU 适合短时间快速运动，视觉适合长时间慢速运动，两者之间互补。同时，可利用视觉定位信息估计 IMU 的零偏，减少 IMU 由零偏导致的发散和误差累计。ii.IMU 和单目视觉融合时可以提供绝对尺度信息

b) 从成本方面：i. 摄像头 + 单目（基本满足精度前提）成本都相对激光雷达深度相机等传感器低很多

c) 从用途方面：i. 相对于深度相机 or 激光雷达等，单目 +IMU 可以用于手机等手持设备或者 AR, 也可以用在无人机等机器人

1.2 有哪些常见的视觉 +IMU 融合的方案？有没有工业界应用的例子？

答：a) 常见的视觉 +IMU 融合方案

名称	耦合方式	后端及库	前端	视觉误差	是否回环
MSCKF	紧耦合	EKF	Fast+ 光流	重投影	否
ROVIO	紧耦合	IEKF	Fast+ 光流	光度	否
SVO+MSF	松耦合	EKF Ceres	FAST	光度	否
VINS	紧耦合	优化 g2o	Harris+ 光流	重投影	是
VIORB	紧耦合	优化	ORB	重投影	是
OKVIS	紧耦合	优化 Ceres	Harris+BRISK	重投影	否
SVO+iSAM2		优化 GTSAM		光度	否

b) 工业界应用的例子

i. 谷歌 Project Tango：MSCKF

ii. 苹果的 ARkit

iii. 微软的 HoloLens：IMU 感应 HoloLens 的方向，环境感知摄像头感应 HoloLens 相对位置的偏移，深度摄像头感知 HoloLens 周围环境。Kinect Fusion+IMU+ 单目 VO

1.3 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？

答：a) 学术界 VIO 进展

i. 目前最流行的框架莫过于基于港科大的 VINS-Fusion。融合了单目双目 +IMU+GPS，有一统多传感器融合框架的趋势。但经过实际测试，发现系统时间越长，存在延迟的现象。

ii. 慕尼黑工大 2019 年发表一篇文章，基于 BA 优化的双目 +IMU，但是没有开源，在 EuRoCs 数据集显示结果比 VINS-Fusion 好。(Visual-Inertial Mapping with Non-Linear Factor Recovery)

iii. 深度相机融合 IMU

iiii. 还有新鲜出炉的 2019 年新的框架：

->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

->Stereo Visual Inertial LiDAR Simultaneous Localization and Mapping：紧耦合双目 VIO 和 LiDAR 建图。在隧道中比 LOAM 好

b) 学习方法用到 VIO 中的例子

i.Unsupervised Deep Visual-Inertial Odometry with Online Error Correction for RGB-D Imagery：

学习在无 IMU 内参或 IMU 和 Camera 外参情况下执行 VIO，学习 IMU 测量生成假设轨迹，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

ii.Selective Sensor Fusion for Neural Visual-Inertial Odometry：

针对单目 VIO 的端到端多传感器融合策略

iii.Visual-Inertial Odometry for Unmanned Aerial Vehicle using Deep Learning：

提出一种网络，针对 VIO 不需要进行标定

1.4 对自己感兴趣的方向进行文献调研，阐述自己观点

答：‘A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots ‘

本文对比了四种不同运算平台上的 VIO 性能，各有优缺点。但其实基于 ARM 的芯片算力都比较强悍，在定位方面 ARM 应该足以。但是针对稠密建图来说，大部分都依赖高性能的 GPU，该如何设计策略来用 ARM 等一些轻量的嵌入式设备支持定位和稠密建图？

2 题二

2.1 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　对某旋转更新时, 有两种不同方式:

$$R \leftarrow R exp(\omega^{\wedge}) \tag{1}$$

$$q \leftarrow q \otimes [1, 1/2\omega]^{\mathrm{T}} \tag{2}$$

请编程验证对于小量　= [0.01, 0.02, 0.03]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 q 还是 R, 也不区分其更新方式为上式的哪一种。

答：a）主程序如下：

```cpp
#include <iostream>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <Eigen/Dense>
using namespace std;
/* 1、定义R 及 q
 * 2、利用计算出来的w对R q更新.增量为w=[0.01,0.02,0.03]T
 * R <- R*exp(w^)
 * q <- q*[1,1/2*w]T
*/
int main()
{
    // 设置R q,假设初始旋转为绕z轴旋转90度
    Eigen::Matrix3d R = Eigen::AngleAxisd(M_PI/4, Eigen::Vector3d(0,0,1)).toRotationMatrix();
    Eigen::Quaterniond q = Eigen::Quaterniond(R);
    cout << "R is = " << endl << R << endl;
```

```cpp
    // 实部在后，虚部在前
    cout << "q is = " << q.coeffs().transpose() << endl;

    // w是计算出来的增量.轴角形式=(v，theta),v为单位向量,theta是向量的模
    Eigen::Vector3d w(0.01, 0.02, 0.03);
    // w的模=m
    double m = sqrt( w(0)*w(0)+w(1)*w(1)+w(2)*w(2) );
    // 旋转向量转换成旋转矩阵
    // 此处等价于【w的指数映射】【罗德里格公式（旋转向量->旋转矩阵）】
    Eigen::Matrix3d w_ = Eigen::AngleAxisd( m, Eigen::Vector3d(0.01/m, 0.02/m, 0.03/m) ).toRotationMatrix();

    /**************** 此处用指数映射计算(不推介但可以用)：  *************/
    // 【注】Eigen里边的exp()函数是对每个元素求exp，不可用在此处，此处通过matlab的expm函数算出w_hat的指数映射
    Eigen::Matrix3d w_hat;
    w_hat << 0, -w(2), w(1),
             w(2), 0, -w(0),
             -w(1), w(0), 0;
    Eigen::Matrix3d w_hat_exp;
    w_hat_exp <<      0.9994,    -0.0299,     0.0201,
                      0.0301,     0.9995,    -0.0097,
                     -0.0198,     0.0103,     0.9998;
    Eigen::Matrix3d R_exp = R * w_hat_exp;
    cout << "R_update with exp is =" << endl << R_exp << endl;
    /***********************************************************/
    // 使用R方式存储，更新R
    R = R * w_;
    cout << "R_update with Rodrigues's Formula is =" << endl << R << endl;

    // 使用q方式储存
    Eigen::Quaterniond q_ = Eigen::Quaterniond (1,0.5*w(0),0.5*w(1),0.5*w(2));
    // 更新q
    q = q * q_;
    q.normalize(); // 归一化

    // 四元数转化为旋转矩阵
    Eigen::Matrix3d q2R = q.toRotationMatrix();
    cout << "R_update form q_update is = " << endl << q2R << endl;

      // 作差比较精度
    Eigen::Matrix3d diff_exp = R_exp - q2R;
    cout << endl << "diff_ between R & q with exp is = " << endl << diff_exp << endl;

```

```
    Eigen::Matrix3d diff_rod = R - q2R;
    cout << endl << "diff_ between R & q with Rodrigues's Formula is = " <<
     endl << diff_rod << endl;
    return 0;
}
```

b) CMakeList.txt:

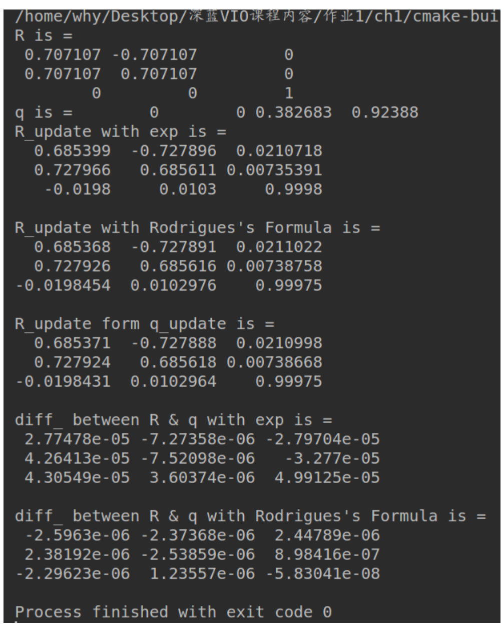
```
cmake_minimum_required(VERSION 3.12)
project(ch1)

set(CMAKE_CXX_STANDARD 11)

include_directories( "/usr/include/eigen3" )
add_executable(ch1 Eigen.cpp)
```

c）运行结果：

根据运行结果显示，罗德里格公式转换的旋转矩阵最后结果要比 exp() 低阶泰勒展开结果更精确一点



```
/home/why/Desktop/深蓝VIO课程内容/作业1/ch1/cmake-bui
R is =
 0.707107 -0.707107         0
 0.707107  0.707107         0
        0         0         1
q is =        0        0 0.382683  0.92388
R_update with exp is =
  0.685399  -0.727896  0.0210718
  0.727966   0.685611 0.00735391
   -0.0198     0.0103     0.9998

R_update with Rodrigues's Formula is =
  0.685368  -0.727891  0.0211022
  0.727926   0.685616 0.00738758
-0.0198454  0.0102976    0.99975

R_update form q_update is =
  0.685371  -0.727888  0.0210998
  0.727924   0.685618 0.00738668
-0.0198431  0.0102964    0.99975

diff_ between R & q with exp is =
 2.77478e-05 -7.27358e-06 -2.79704e-05
 4.26413e-05 -7.52098e-06   -3.277e-05
 4.30549e-05  3.60374e-06  4.99125e-05

diff_ between R & q with Rodrigues's Formula is =
 -2.5963e-06 -2.37368e-06  2.44789e-06
 2.38192e-06 -2.53859e-06  8.98416e-07
-2.29623e-06  1.23557e-06 -5.83041e-08

Process finished with exit code 0
```

3 题三

3.1 使用右乘推导以下导数

答：用到的一些性质：

$R^{-1} = R^T$

$R^T exp(\phi^\wedge) R = exp((R^T \phi)^\wedge$

$\ln(Rexp(\phi^\wedge) = \ln(R) + J_r^{-1}\phi$

为了与课件格式保持一致，推导如下：

a) $\frac{\mathrm{d}(R^{-1}p)}{\mathrm{d}R}$

$$\frac{\mathrm{d}(R^{-1}p)}{\mathrm{d}R} \tag{3}$$

$$= \lim_{\phi\to 0} \frac{(Rexp(\phi^\wedge))^{-1} - R^{-1}p}{\phi} \tag{4}$$

$$= \lim_{\phi\to 0} \frac{exp(\phi^\wedge)^{-1} R^{-1}p - R^{-1}p}{\phi} \tag{5}$$

$$= \lim_{\phi\to 0} \frac{(I - \phi^\wedge) R^{-1}p - R^{-1}p}{\phi} \tag{6}$$

$$= \lim_{\phi\to 0} \frac{(-\phi^\wedge) R^{-1}p}{\phi} \tag{7}$$

$$= \lim_{\phi\to 0} \frac{(R^{-1}p)^\wedge \phi}{\phi} \tag{8}$$

$$= (R^{-1}p)^\wedge \tag{9}$$

b) $\frac{\mathrm{d}\ln(R_1R_2^{-1})^\vee}{\mathrm{d}R_2}$

$$\frac{\mathrm{d}\ln(R_1R_2^{-1})}{\mathrm{d}R_2} \tag{10}$$
$$= \lim_{\phi\to 0}\frac{\ln R_1(R_2exp(\phi^\wedge))^{-1} - \ln R_1R_2^{-1}}{\phi} \tag{11}$$
$$= \lim_{\phi\to 0}\frac{\ln R_1exp(\phi^\wedge)^{-1}R_2^{-1} - \ln R_1R_2^{-1}}{\phi} \tag{12}$$
$$= \lim_{\phi\to 0}\frac{\ln R_1R_2^{-1}exp((R_2^{-1}\phi)^\wedge)^{-1} - \ln R_1R_2^{-1}}{\phi} \tag{13}$$
$$= \lim_{\phi\to 0}\frac{\ln R_1R_2^{-1}exp((-R_2^{-1}\phi)^\wedge) - \ln R_1R_2^{-1}}{\phi} \tag{14}$$
$$= \lim_{\phi\to 0}\frac{\ln R_1R_2^{-1} - J_r^{-1}R_2^{-1}\phi - \ln R_1R_2^{-1}}{\phi} \tag{15}$$
$$= -J_r^{-1}(\ln(R_1R_2^{-1})^\vee)R_2^{-1} \tag{16}$$

此前是由于: $(Rexp(\phi^\wedge)^TR^T)^T = R^Texp(\phi^\wedge)R$ 此部分求错，导致错误

现根据：$Rexp(\phi^\wedge)R^T = exp((R\phi)^\wedge)$
$(Rexp(\phi^\wedge)^TR^T)^T = Rexp(\phi^\wedge)R^T$ 修改如下:

$$\frac{\mathrm{d}\ln(R_1R_2^{-1})^\vee}{\mathrm{d}R_2} \tag{17}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1(R_2exp(\phi^\wedge))^{-1})^\vee - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{18}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1exp(\phi^\wedge)^{-1}R_2^{-1})^\vee - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{19}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1R_2^{-1}(R_2exp(\phi^\wedge)^{-1}R_2^{-1}))^\vee - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{20}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1R_2^{-1}(R_2exp(-\phi^\wedge)R_2^{-1}))^\vee - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{21}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1R_2^{-1}exp((R_2(-\phi))^\wedge))^\vee - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{22}$$
$$= \lim_{\phi\to 0}\frac{\ln(R_1R_2^{-1})^\vee - J_r^{-1}R_2\phi - \ln(R_1R_2^{-1})^\vee}{\phi} \tag{23}$$
$$= -J_r^{-1}(\ln(R_1R_2^{-1})^\vee)R_2 \tag{24}$$

1

[1]【注】部分内容参考崔华坤 VIO 公开课程 ppt，及数理之家公众号推送