第一题 文献阅读

我看的是这篇综述：

> A review of visual inertial odometry from filtering and optimisation perspectives

文章首先提出了基于滤波和基于优化的两种处理SLAM的手段。贝叶斯法则是这两种方法的桥梁。

基于优化的方法，可以看作是最大似然估计。 基于滤波的slam可以看做是最大后验估计。

然后文章具体分析了两种方法在VIO中的应用。并对来两种方法的联系做了详细阐述。其中涉及到VIO各个方面的基础知识，值得一读。

回答作业中提出的问题：

> 1. 视觉与 IMU 进行融合之后有何优势？

这个问题，我认为高博的课件已经说的很明白了。

IMU 与视觉定位方案优势与劣势对比：

方案	IMU	视觉
优势	快速响应 不受成像质量影响 角速度普遍比较准确 可估计绝对尺度	不产生漂移 直接测量旋转与平移
劣势	存在零偏 低精度 IMU 积分位姿发散 高精度价格昂贵	受图像遮挡、运动物体干扰 单目视觉无法测量尺度 单目纯旋转运动无法估计 快速运动时易丢失

整体上，视觉和 IMU 定位方案存在一定互补性质：

- IMU 适合计算短时间、快速的运动；
- 视觉适合计算长时间、慢速的运动。

同时，可利用视觉定位信息来估计 IMU 的零偏，减少 IMU 由零偏导致的发散和累积误差；反之，IMU 可以为视觉提供快速运动时的定位。

> 2. 有哪些常见的视觉 +IMU 融合方案？有没有工业界应用的例子？

- 基于滤波： MSCKF ROVIO

- 基于优化： VINS、 VI-ORB、 ICE-BA
- 工业界： 苹果的 ARKit 和谷歌的 ARCore 都是 VIO 的典型应用

> 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？

见以下两篇论文

> Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight

> A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots

第二题 编程验证

仅仅使用eigen：

```
#include <iostream>
#include <Eigen/Jacobi>
#include <Eigen/Dense>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <Eigen/Householder>
#include <cmath>
using namespace std;
int main(int argc, char** argv){
    Eigen::AngleAxisd rotation_vector(M_PI/3, Eigen::Vector3d(0,0,1));
    cout.precision(3);
    Eigen::Matrix3d R = rotation_vector.toRotationMatrix();
    Eigen::Quaterniond q = Eigen::Quaterniond(rotation_vector);
    // 旋转小量w
    Eigen::Vector3d w(0.01, 0.02, 0.03);
    Eigen::Matrix3d w_hat;
    w_hat << 0,-0.03,0.02,
            0.03,0,-0.01,
            -0.02,0.01,0;
    cout << "变换前的R:" << endl << R << endl;
// 这里要求第一种更新方式
    Eigen::Matrix3d I = Eigen::Matrix3d::Identity();
    R = R *(I + w_hat);
    cout << "第一种更新后的旋转矩阵R:" << endl << R << endl;
    // 第二种变换方式
    Eigen::Quaterniond Q1(1, 0.005, 0.01, 0.015);
    q = q*Q1;
    q.normalize();
    Eigen::Matrix3d R2(q);
    cout << "第二种更新方法后的R："<< endl << R2 << endl;
}
```

程序输出结果：

```
变换前的R:
   0.5 -0.866      0
 0.866    0.5      0
     0      0      1
第一种更新后的旋转矩阵R:
 0.474 -0.881 0.0187
 0.881  0.474 0.0123
 -0.02   0.01      1
第二种更新方法后的R：
  0.474   -0.881   0.0185
  0.881    0.474   0.0126
-0.0198   0.0103        1
```

使用Sophus:

```
//
// Created by allen on 19-6-15.
//
#include <iostream>
#include <cmath>
using namespace std;
#include <Eigen/Core>
#include <Eigen/Geometry>
#include "sophus/so3.h"
using namespace std;
int main(int argc, char** argv){
    // 根据角轴创建旋转矩阵
    Eigen::Matrix3d R = Eigen::AngleAxisd(M_PI/3, Eigen::Vector3d(0,0,1)).toRotationM
    // 旋转矩阵直接构造李群
    Sophus::SO3 SO3_R(R);
    Eigen::Quaterniond q(R);
    cout << "李代数初始模样：" << endl << SO3_R << endl;
    // 增量扰动模型
    Eigen::Vector3d v_updated(0.01, 0.02, 0.03);
    Sophus::SO3 SO3_updated = SO3_R * Sophus::SO3::exp(v_updated);
    cout<< "第一种更新方式：" << endl << SO3_updated << endl;
    Eigen::Quaterniond q_updated(1, 0.005, 0.01, 0.015);
    q = q * q_updated;
    // normalize直接在原始的q上更改  normalized返回一个归一化后的q
    q.normalize();
    Sophus::SO3 SO3_q(q);
    cout << "第二种更新方式" << endl << SO3_q << endl;
    return 0;
}
```

输出结果：

```
李代数初始模样：
      0       0 1.0472

第一种更新方式：
-0.00140673   0.0234364     1.07715

第二种更新方式
-0.00140656   0.0234337     1.07715
```

可以看出，两种更新方法只在小数点第三位有微小的区别，所以以后做姿态更新，两个方法都没问题。

第三题 公式推导

①求导：

$$\frac{d(R^{-1}p)}{dR} = \lim_{\varphi \to 0} \frac{R^{-1}\exp(\varphi^\wedge)p - R^{-1}p}{\varphi} = \lim_{\varphi \to 0} \frac{R^{-1}(I+\varphi^\wedge)p - R^{-1}p}{\varphi}$$

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

②求导：

$$\frac{d\ln(R_1R_2^{-1})^\vee}{dR_2} = \lim_{\varphi \to 0} \frac{\ln(R_1R_2^{-1}\exp(\varphi^\wedge))^\vee - \ln(R_1R_2^{-1})^\vee}{\varphi}$$

$$= \lim_{\varphi \to 0} \frac{J_r^{-1}(\ln(R_1R_2^{-1})^\vee)\varphi^\wedge + (\ln(R_1R_2^{-1}))^\vee - (\ln(R_1R_2^{-1}))^\vee}{\varphi^\wedge}$$

$$= J_r^{-1}((\ln(R_1R_2^{-1}))^\vee)$$

其中J_r^{-1}是右乘BCH近似雅可比J_r的逆：具体如下所示：

$$J_r^{-1} = \frac{\theta}{2}\cot\frac{\theta}{2} I + \left(1-\frac{\theta}{2}\cot\frac{\theta}{2}\right)\vec{w}\vec{w}^T - \frac{\theta}{2}\vec{w}^\wedge$$，其中$\varphi = \theta\vec{w}$