1.VIO 文献阅读

阅读 VIO 相关综述文献如 *a*，回答以下问题：

1. 视觉与 IMU 进行融合之后有何优势？
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？
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？
   你也可以对自己感兴趣的方向进行文献调研，阐述你的观点。

解：

1.

案优势与劣势对比：

方案	IMU	相机
优势	快速响应 不受成像质量的影响 角速度普遍准确 可估计绝对尺度	不产生偏漂移（静态时） 直接旋转和平移
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 快速运动时容易丢失

整体上，视觉和 IMU 融合存在一定的互补性质：

总结为：Low drift、high rate 、Robustness、**物理尺度状态估计（位置、速度、姿态）**

详解为：

1) MU 适合计算短时间、快速的运动；
2) 视觉适合计算短时间、慢速的运动；
3) 同时，可以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计误差。反之，IMU 可以为视觉提供快速运动的定位。
4) IMU 可以提供尺度信息，避免单目无法测尺度

2.那些的视觉 +IMU 融合方案？有没有工业界应用的例子？

MSCKF、OKVIS、RBVIO、VINS

4．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？

a.VIO 结合直线和点特征

如：2018 rifo-VIO: Robust and Efficient Stereo Visual Inertial Odometry using Points and Lines

Stereo Visual-Inertial SLAM With Points and Lines

b.时间在线标定

如：Online Temporal Calibration for Monocular Visual-Inertial Systems

c.多机器人协作

如 CVI-SLAM – Collaborative Visual-Inertial SLAM

d.利用深度学习实现深度估计（机器学习）

如; Embedding Temporally Consistent Depth Recovery for Real-time Dense Mapping in Visual-inertial Odometry

e. 港科大多传感器融合

A General Optimization-based Framework for Global Pose Estimation with Multiple Sensors

2.四元数或和李代数更新

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 $\boldsymbol{\omega}$ 对某旋转更新时，有两种不同方式：

$$\mathbf{R} \leftarrow \mathbf{R} \exp(\boldsymbol{\omega}^\wedge)$$

$$\text{或} \quad \mathbf{q} \leftarrow \mathbf{q} \otimes [1, \tfrac{1}{2}\boldsymbol{\omega}]^T$$

请编程验证对于小量 $\boldsymbol{\omega} = [0.01, 0.02, 0.03]^T$，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 $\mathbf{q}$ 还是 $\mathbf{R}$，也不区分其更新方式为上式的哪一种。

解：程序如下：

```cpp
#include <iostream>

//using eigen3
#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Geometry>

using namespace std;


int main() {
    //随机生成随机旋转矩阵
    Eigen::Vector3d   rot_axis=Eigen::Vector3d::Random();
    rot_axis.normalize();

    Eigen::AngleAxisd  rot_angle_axis(M_PI/4,rot_axis);
    Eigen::Matrix3d  R=rot_angle_axis.toRotationMatrix();
    cout<<"rotation  matrix  R before  update"<<endl<<R<<endl;

    Eigen::Vector3d w(0.01,0.02,0.03);
    double  robot_angle=w.norm();
    Eigen::AngleAxisd  w_rot_angle_axis(robot_angle,w.normalized());

    //update R using w.toRotationMatrix()
    Eigen::Matrix3d     w_rot_matrix=w_rot_angle_axis.toRotationMatrix();
    R=R*w_rot_matrix;
    cout<<"rotation  matrix R after  update using matrix"<<endl<<R<<endl;

    //update R using q
    Eigen::Quaterniond  q(rot_angle_axis);
    Eigen::Quaterniond  q_w(1,0.5*w(0),0.5*w(1),0.5*w(2));
    q=q*q_w;

    q.normalize();
    cout<<"rotation  matrix R  after  update using q"<<endl<<q.toRotationMatrix()<<endl;

    return 0;
}
```

运行结果为：

```
vslam@vslam:~/VIO_Tutorial/第1节概述与课程介绍/L1-2/build$ make
Scanning dependencies of target main
[ 50%] Building CXX object CMakeFiles/main.dir/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
vslam@vslam:~/VIO_Tutorial/第1节概述与课程介绍/L1-2/build$ ./main
rotation  matrix  R before  update
  0.870833  -0.490788 -0.0278859
  0.389125   0.722888  -0.570977
  0.300387   0.486374   0.820492
rotation  matrix R after  update using matrix
   0.856051   -0.516861 -0.00557622
   0.421957    0.705015   -0.570005
   0.298545    0.485601    0.821622
rotation  matrix R  after  update using q
   0.856053   -0.516858 -0.00557882
   0.421953    0.705017   -0.570006
   0.298545    0.485601    0.821622
vslam@vslam:~/VIO_Tutorial/第1节概述与课程介绍/L1-2/build$
```

3.其他导数

使用右乘 $\mathfrak{so}(3)$，推导以下导数：

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

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

解：

1）

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

$$=\lim_{\varphi\to 0}\frac{\left(\mathrm{Re}xp\left(\varphi^{\wedge}\right)\right)^{-1}p\text{-}R^{-1}p}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\mathrm{e}xp\left(\varphi^{\wedge}\right)^{-1}R^{-1}p\text{-}R^{-1}p}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\mathrm{e}xp\left((-\varphi)^{\wedge}\right)R^{-1}p\text{-}R^{-1}p}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\left(I+(-\varphi)^{\wedge}\right)R^{-1}p\text{-}R^{-1}p}{\varphi}$$

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

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

$$=\left(R^{-1}p\right)^{\wedge}$$

$$\frac{d(\ln(R_1R_2^{-1}))}{dR_2}$$

$$=\lim_{\varphi\to 0}\frac{\ln\left(\mathrm{R}_1\left(\mathrm{R}_2\mathrm{e}xp\left(\varphi^{\wedge}\right)\right)^{-1}\right)-\ln(R_1R_2^{-1})}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\ln\left(\mathrm{R}_1\left(\mathrm{e}xp\left(\varphi^{\wedge}\right)\right)^{-1}\mathrm{R}_2^{-1}\right)-\ln(R_1R_2^{-1})}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\ln\left(\mathrm{R}_1\left(\mathrm{e}xp\left(-\varphi^{\wedge}\right)\right)\mathrm{R}_2^{-1}\right)-\ln(R_1R_2^{-1})}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\ln\left(\mathrm{R}_1\mathrm{R}_2^{-1}\mathrm{R}_2\left(\mathrm{e}xp\left(-\varphi^{\wedge}\right)\right)\mathrm{R}_2^{-T}\right)-\ln(R_1R_2^{-1})}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{\ln\left(\mathrm{R}_1\mathrm{R}_2^{-1}\left(\mathrm{e}xp\left((-\mathrm{R}_2\varphi)^{\wedge}\right)\right)\right)-\ln(R_1R_2^{-1})}{\varphi}$$

$$=\lim_{\varphi\to 0}\frac{J_r(\ln(\mathrm{R}_1\mathrm{R}_2^{-1}))(-\mathrm{R}_2\varphi)+\ln(R_1R_2^{-1})-\ln(R_1R_2^{-1})}{\varphi}$$

$$=-J_r(\ln(\mathrm{R}_1\mathrm{R}_2^{-1}))\mathrm{R}_2$$

这里用到了 SO(3) 的伴随性质：

$$\mathrm{R}_2\left(\mathrm{e}xp\left(-\varphi^{\wedge}\right)\right)\mathrm{R}_2^{-T}=\mathrm{e}xp\left((-\mathrm{R}_2\varphi)^{\wedge}\right)$$