# 习题一 VIO 文献阅读

1. 阅读 VIO 相关综述文献，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。

答案

1) IMU 是测量自身的（proprioceptive），测量结果不受成像质量影响，视觉是感知外界的（exteroceptive），不会产生漂移，两者之间可以优势互补：
   - 视觉可以解决纯 IMU 下产生角度和位置漂移（零偏）导致的累计误差和积分发散的问题。
   - IMU 可以解决在在单目相机下尺度不确定的问题，同时 IMU 测量的频率（>100Hz）比视觉高，可以解决视觉在快速运动下图像模糊和特征匹配出现错误的问题。
2) 常见的融合方案：
   I. 松耦合：IMU 和视觉分别输出位姿，融合过程为对二者后处理，典型方案有卡尔曼滤波器
   II. 紧耦合：同时考虑视觉输入的图像特征和 IMU 的 15 维信息（速度，位置，方向，偏移）来进行位姿优化，维度高计算量非常大（因此首先可以排除粒子滤波器），但由于是同时构建运动方程和观测方程来进行状态估计，不容易像松耦合那样陷入局部最优解，典型方案是基于非线性优化 VINS，OKVIS 和基于滤波 MSCKF，ROVIO

   总得来说，松耦合通常计算简单一些，但精度较差，而紧耦合则相反。

   工业界的应用有很多方向，主流的是在 AR/VR 上的应用，如苹果的 ARKit 和谷歌的 ARCore，微软的 HoloLens 等。

3) 
   I. 关于 VIO 的新进展：
      - 常见的 VIO 选取的关键帧间隔较大来提高效率，但是这样 IMU 在多次预积分后能提供的信息很少，此论文提出利用 2 级结构， 第 1 层为 VIO 系统采用稀疏光流下 KLT 跟踪，结合金字塔层级来实现大位移下的鲁棒性，第 2 层用来建图，仅在第 1 层下使用 IMU 的预积分来进行运动估计，第 2 层建图则在第 1 层的基础上提取非线性因子来近似，减小了优化问题的规模同时采用了回环检测来提高建图质量[1]
      - S-MSCKF 基于卡尔曼滤波的开源方案，实现了可以在笔记本（无需 GPU）的算力下实现了较好的实时性 [2]
      - 其它的新进展包括将 deep learning 在 VIO 中的应用

[1] Vladyslav, U., Nikolaus D., et al. Visual-Inertial Mapping with Non-Linear Factor Recovery, IEEE Robotics and Automation Letters (Volume: 5, Issue: 2, April 2020)
[2] Ke, S., Kartik M., et al. Stereo Visual Inertial Odometry for Fast Autonomous Flight, IEEE Robotics and Automation Letters (Volume: 3, Issue: 2, April 2018)

II. 将学习的方法应用到 VIO 中的：

- VINet 提出将 VIO 系统描述为一个完全的端到端的训练模型, 用 CNN 来处理图像，LSTM 来处理 IMU 信息，对于平移的运动估计较好（因为可以通过学习来估计尺度），主要优势还是在能学习鲁棒的策略来处理标定误差 [1]
- 采用 deep learning 实现一个鲁棒的传感器融合策略，来处理数据损坏以及 IMU 和视觉同步较差时的融合问题 [2]

[1] Clark, R., Wang, S., et al. (2017). VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem. Proceedings of the AAAI Conference on Artificial Intelligence
[2] Changhao Chen, Stefano Rosa, et al. Selective Sensor Fusion for Neural Visual-Inertial Odometry, Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 10542-10551

## 习题二 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 $\omega$ 对某旋转更新时，有两种不同方式：

$$\begin{aligned} & \mathbf{R} \leftarrow \mathbf{R}\exp\left(\boldsymbol{\omega}^\wedge\right) \\ \text{或} \quad & \mathbf{q} \leftarrow \mathbf{q} \otimes \left[1, \tfrac{1}{2}\boldsymbol{\omega}\right]^\top \end{aligned} \tag{20}$$

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。

```cpp
// using Sophus for update
void update_Matrix_Sophus(const Vector3d & update, Matrix3d & matrix3D){
    Sophus::SO3d SO3d_R(matrix3D);
    matrix3D = (SO3d_R * Sophus::SO3d::exp(update)).matrix();

}
// using Rodrigues' formula for exp(w^) without Sophus
void update_Matrix(const Vector3d & update, Matrix3d & matrix3D){
    double theta = update.norm();
    Vector3d norm = update.normalized();
    Matrix3d skewMatrix = Matrix3d::Zero();
    skewMatrix( row: 0, col: 1) = -norm.z();
    skewMatrix( row: 0, col: 2) = norm.y();
    skewMatrix( row: 1, col: 0) = norm.z();
    skewMatrix( row: 1, col: 2) = -norm.x();
    skewMatrix( row: 2, col: 0) = -norm.y();
    skewMatrix( row: 2, col: 1) = norm.x();
    //Rodrigues' formula
    Matrix3d R_update = cos(theta) * Matrix3d::Identity()
                        + (1 - cos(theta)) * norm * norm.transpose()
                        + sin(theta) * skewMatrix;

    matrix3D *= R_update;
}
void update_Quaternion(const Vector3d & update, Quaterniond & q){
    Quaterniond q_update = { w: 1, x: update.x()/2, y: update.y()/2, z: update.z()/2};
    q *= q_update;
    q.normalize();
}
```

```cpp
int main() {

    Vector3d update = { x: 0.01, y: 0.02, z: 0.03};
    // original rotation matrix
    Matrix3d R = AngleAxisd(M_PI_2, axis: Vector3d{ x: 0, y: 0, z: 1}).toRotationMatrix();
    Quaterniond q(R);

    // updated rotation matrix
//    update_Matrix_Sophus(update, R);
    update_Matrix(update, &: R);
    cout << "use Rotation Matrix to update :\n" << R << endl;

    update_Quaternion(update, &: q);
    cout << "use Quaternion to update :\n" << q.toRotationMatrix() << endl;

    cout << "\ndifference:\n" << q.toRotationMatrix() - R << endl;

    ////conclusion: order of magnitude at least -6, so the difference can be neglected

    return 0;
```

关于旋转矩阵的更新分别提供了由 Sophus 库和利用罗德里格斯公式更新的两种实现。

分别用四元数和旋转矩阵对绕 Z 轴转动 90 度的旋转进行更新，在更新量为[0.01, 0.02, 0.03]时得到的结果如下，可见二者差值数量级为 10 的-6 次方级别，因此当更新量很小时二者区别可以忽略。

```
use Rotation Matrix to update :
 -0.030093    -0.9995  0.0096977
   0.99935  -0.029893  0.0201453
-0.0198454  0.0102976    0.99975
use Quaternion to update :
-0.0300895    -0.9995 0.00969661
   0.99935 -0.0298895  0.0201429
-0.0198431  0.0102964    0.99975

difference:
 3.52014e-06 -1.16608e-07 -1.09564e-06
 1.51591e-07  3.47349e-06 -2.36619e-06
 2.29623e-06 -1.23557e-06  5.83041e-08
```

## 习题三 导数推导

**使用右乘 $\mathfrak{so}(3)$，推导以下导数：**

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

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

1.

$$\frac{\mathrm{d}R^{-1}\cdot p}{\mathrm{d}R}=\frac{\mathrm{d}R^{\top}\cdot p}{\mathrm{d}R}=\lim_{\varphi\to 0}\frac{(R\,exp(\varphi^{\wedge}))^{\top}\cdot p-R^{\top}\cdot p}{\varphi}=\lim_{\varphi\to 0}\frac{exp(-\varphi^{\wedge})R^{\top}\cdot p-R^{\top}\cdot p}{\varphi}$$
$$=\lim_{\varphi\to 0}\frac{(I-\varphi^{\wedge})R^{\top}\cdot p-R^{\top}\cdot p}{\varphi}=\lim_{\varphi\to 0}\frac{-\varphi^{\wedge}R^{\top}\cdot p}{\varphi}=\lim_{\varphi\to 0}\frac{(R^{\top}p)^{\wedge}\varphi}{\varphi}$$
$$=(R^{\top}p)^{\wedge}$$

易错点在于更新量 $exp(\varphi^{\wedge})$ 是右乘在$R$上而不是$R^{\top}$上

2.

$$\frac{\mathrm{d}\ ln(R_1 R_2^{-1})^\vee}{\mathrm{d}R_2} = \frac{\mathrm{d}\ ln(R_1 R_2^\top)^\vee}{\mathrm{d}R_2} = \lim_{\varphi \to 0} \frac{ln(R_1(R_2\ exp(\varphi^\wedge))^\top)^\vee - ln(R_1 R_2^\top)^\vee}{\varphi}$$

$$= \lim_{\varphi \to 0} \frac{ln(R_1\ exp(-\varphi^\wedge) R_2^\top)^\vee - ln(R_1 R_2^\top)^\vee}{\varphi}$$

$$= \lim_{\varphi \to 0} \frac{ln(R_1 R_2^\top\ exp(-(R_2\varphi)^\wedge))^\vee - ln(R_1 R_2^\top)^\vee}{\varphi}$$

$$= \lim_{\varphi \to 0} \frac{ln(R_1 R_2^\top)^\vee - J_r^{-1}(ln(R_1 R_2^\top)^\vee) \cdot (R_2\varphi) - ln(R_1 R_2^\top)^\vee}{\varphi}$$

$$= -J_r^{-1}(ln(R_1 R_2^\top)^\vee) \cdot R_2$$

第二行到第三行用到了 SO(3)上伴随的性质

第三行到第四行是 BCH 右乘小量的近似, 其中 $J_r^{-1}(ln(R_1 R_2^\top)^\vee)$ 表示右乘时雅可比矩阵的逆。