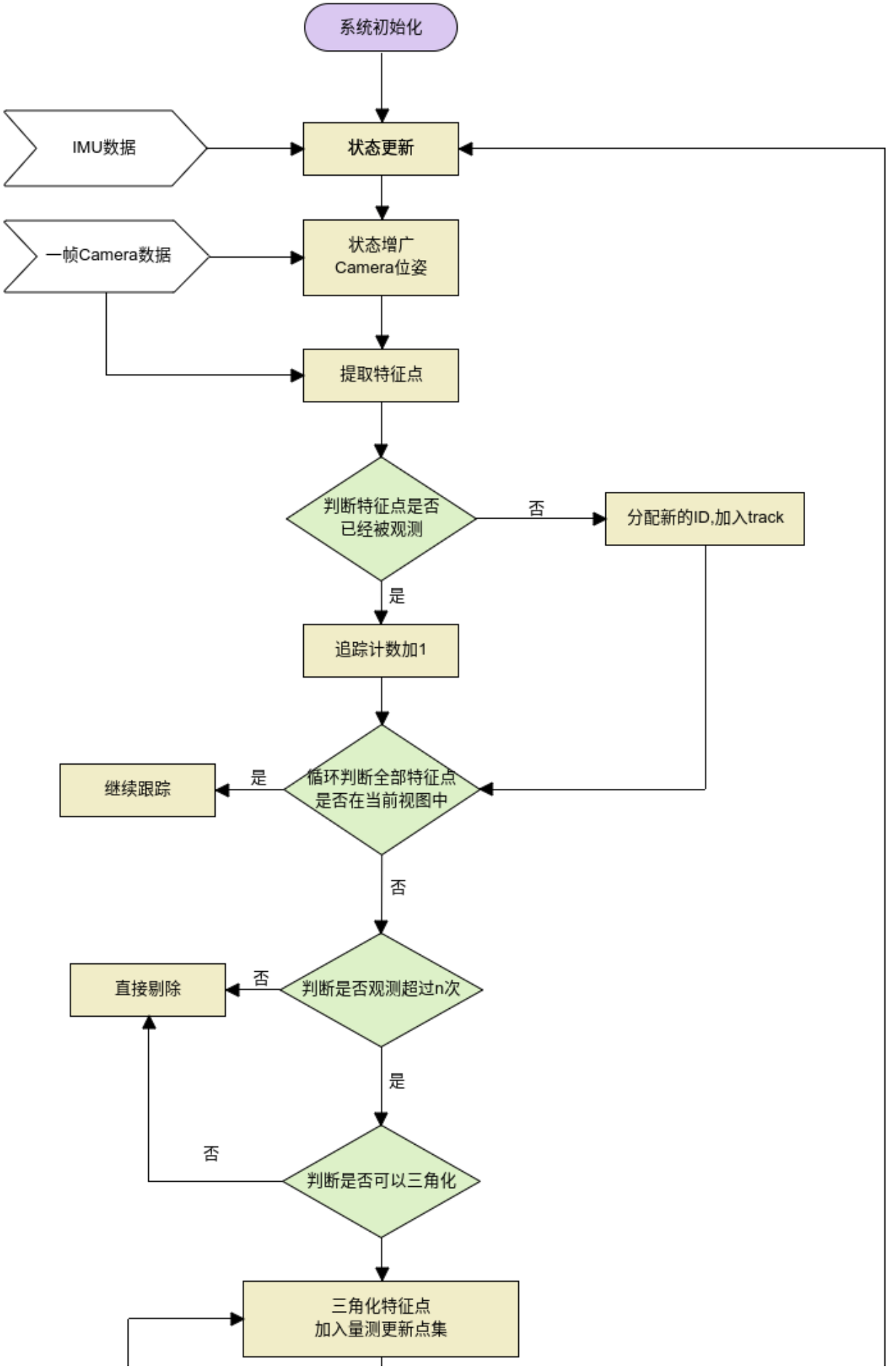
MSCKF 思路梳理

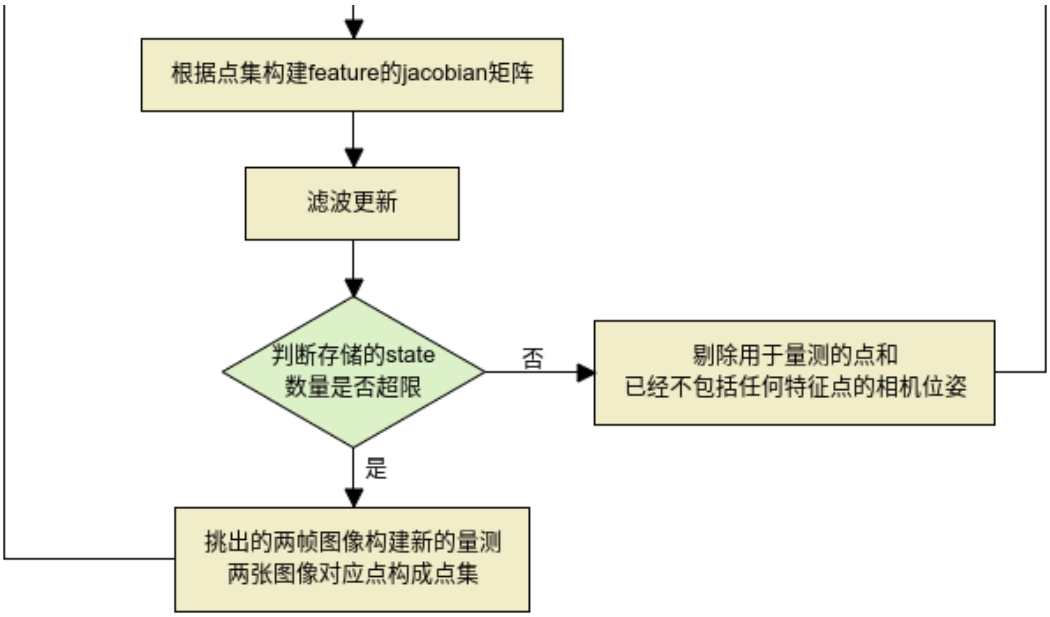
简介

惯性导航利用惯性测量单元(IMU)测量得到的角速度、加速度信息进行惯性导航解算得到运载体的位置、速度、姿态(含航向)等信息，具有实时性好、动态性能好等优点；但是由于其积分式特点，使得传感器和算法解算的误差会持续累积，导致长时间精度很低，特别是对于低端IMU。即使是军用战略性武器上使用的高精度惯导系统也在努力与别的信息进行融合，提高实际导航精度。视觉信息包含了丰富的三维场景信息，通过视觉跟踪可以对运载体的运动进行测量，视觉测量可以给出运动增量。惯性与视觉融合算法利用视觉观测与IMU预测联合得到重投影误差作为观测量来进行系统状态估计。分为滑动窗SWF与多状态约束卡尔曼滤波MSCKF两种技术方案。

MSCKF (Multi-State Constraint Kalman Filter),从2007年提出至今,一直是filter-based SLAM比较经典的实现。在传统的EKF-SLAM框架中，特征点的信息会加入到特征向量和协方差矩阵里,这种方法的缺点是特征点的信息会给一个初始深度和初始协方差，如果不正确的话，极容易导致后面不收敛，出现inconsistent的情况。MSCKF维护一个pose的FIFO，按照时间顺序排列，可以称为滑动窗口，一个特征点在滑动窗口的几个位姿都被观察到的话，就会在这几个位姿间建立约束，从而进行KF的更新。

流程图


系统初始化
IMU数据
状态更新
一帧Camera数据
状态增广
Camera位姿
提取特征点
判断特征点是否
已经被观测
否
分配新的ID,加入track
是
追踪计数加1
继续跟踪
是
循环判断全部特征点
是否在当前视图中
否
直接剔除
否
判断是否观测超过n次
是
否
判断是否可以三角化
三角化特征点
加入量测更新点集



流程图注释

- **系统初始化**
  - 相机参数设置,初始方差设定,噪声方差(包括图像噪声,IMU噪声,IMU的bias),IMU和Camera相关参数设定(估计或者给定)
  - MSCKF相关参数: 状态向量里滑动窗口大小的范围、空间点三角化误差阈值、是否做零空间矩阵构造和QR分解等
  - 构造基本的MSCKF状态量, 包括基本的IMU15维等
  - 初始位置和姿态确定[自主确定一个局部的坐标系或者GPS信息给定一个全局的初始信息]
  - IMU状态量构造如下：

$$X = \begin{Bmatrix} \phi_i & v_i & p_i & b_g & b_a \end{Bmatrix}$$

- **状态更新**
  - 利用IMU数据进行状态方差和状态量(IMU的状态)的更新
    对应IMU状态方差更新：

$$P_{II_{k+1|k}} = \Phi_k P_{II_{k|k}} {\Phi_k}^T + Q_k$$

    对应全状态方差的表达如下：

$$P_{k|k} = \begin{Bmatrix} P_{II_{k|k}} & P_{IC_{k|k}} \\ P^T_{IC_{k|k}} & P_{CC_{k|k}} \end{Bmatrix}$$

时间更新全状态量方差：

$$P_{k+1|k} = \begin{Bmatrix} P_{II_{k+1|k}} & \Phi_k P_{IC_{k|k}} \\ P_{IC_{k|k}}^T \Phi_k^T & P_{CC_{k|k}} \end{Bmatrix}$$

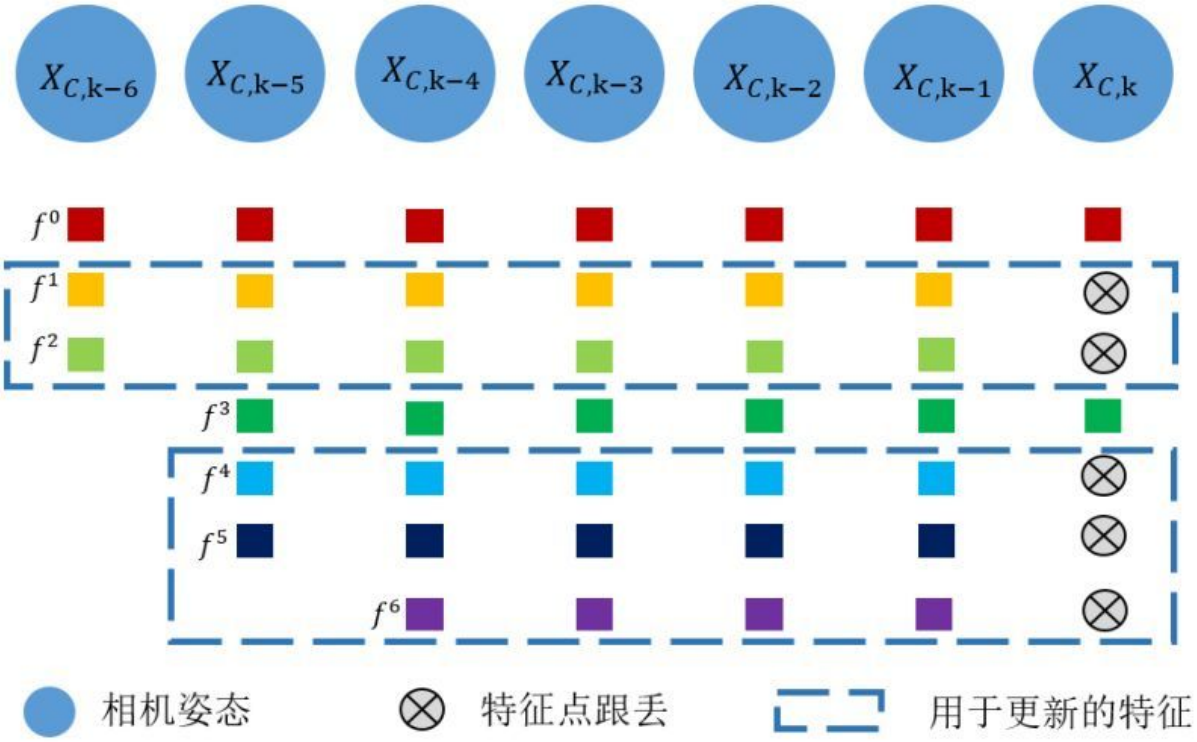
- **状态增广**
  - 状态增广是将给进来的Camera时刻的位置和姿态增广到状态向量和状态方差中
    具体公式如下:

$$P_{k|k} = \begin{Bmatrix} I_{15+6N} \\ J \end{Bmatrix} P_{k|k} \begin{Bmatrix} I_{15+6N} \\ J \end{Bmatrix}^T$$

$$J = \begin{pmatrix} J_I & 0_{6N\times 6} \end{pmatrix}$$

$$J_I = \begin{Bmatrix} C_i^c & 0 & 0 & 0_{3\times 6} \\ (C_i^n * l_{ic}^i) & 0 & I_3 & 0_{3\times 6} \end{Bmatrix}$$

    和算法组的给的note有差异,参考了论文[3][4].
- **特征点信息使用**
  - 提取特征点,剔除一些外点.
  - 判断点是否在已经观测到的点中,
    - 如果在,追踪计数加1, 用来计算和已有的特征点的重复率,后面可以用来做状态量超限时挑选该剔除的帧数据.
    - 如果不在,分配全新的ID,加入track表.
  - 对全部特征点进行循环判断,判断是否在当前View中可观测.
    - 如果可观测,不做处理.
    - 如果不可观测, 需要对其进行一系列处理.
  - 不可观测特征点处理
    - 是否观测次数超过一定次数(如3次),否则剔除
    - 是否可以三角化,满足三角化的指标,否则剔除
    - 满足全部条件点,构成用来测量更新的点集



[来自: http://www.sohu.com/a/271370131_715754]

- **三角化**
  - [5]中feature.hpp文件中的initializePosition函数
  - 逆深度法加LM优化
  - 算法组程序中feature_msckf.hpp中的initializePosition_LM函数
- **feature的Jacoian**
  - 根据量测更新使用的特征点,计算对应的雅各比矩阵
  - 相关公式表达如下:
    量测方程

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_f^c = P_f^c = (C_c^n)^T (P_f^n - l_{nc}^n) \tag{1}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{Z_f^c} \begin{bmatrix} X \\ Y \end{bmatrix}_f^c \tag{2}$$

其中 $\begin{bmatrix} u \\ v \end{bmatrix}$ 表示归一化后且已经进行过畸变矫正的像素点坐标,本文档中暂时不考虑估计外参和内参,直接使用标定值.对式(2)求Jacobian有:

$$J_1 = \frac{\partial(u,v)}{\partial(X,Y,Z)} = \begin{bmatrix} \frac{1}{Z_f^c} & 0 & -\frac{X_f^c}{(Z_f^c)^2} \\ 0 & \frac{1}{Z_f^c} & -\frac{Y_f^c}{(Z_f^c)^2} \end{bmatrix}$$

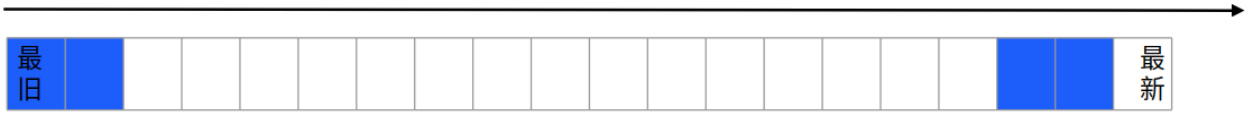
对(1)式求Jacobian有:

$$J_2 = \frac{\partial P_f^c}{\partial(C_c^n, P_f^n, l_{nc}^n)} = \begin{bmatrix} (C_c^n)^T(P_f^n - l_{nc}^n)\times & (C_c^n)^T & -(C_c^n)^T \end{bmatrix} \cdot \begin{bmatrix} \theta_c & p_f & t_c \end{bmatrix}^T$$

θ_c 和 t_c 为camera位姿对应的误差改正数,p_f 为特征点位置改正数.
此处还可以进行observability constrain 可以参考论文"Observability-constrained vision-aided inertial navigation" 和[5]

- 卡方检验 对每个点计算的残差和H进行卡方检验 [5]中msckf_vio.cpp中的gatingTest()函数,通过则加入当前量测更新矩阵.
- 完成全部量测点的方程构建后,计算H_f的左零空间,消除feature点的量测矩阵 此处可做最大行数限制,以保证矩阵不至于过大.
- [5]中的msckf_vio.cpp的featureJacobian()函数

- **滤波更新**
  - 在矩阵维度较高的条件下,可以采用QR分解,减小矩阵维度,然后在进行滤波处理
  - 普通EKF的update过程
- **超限状态剔除问题**
  - 在实际数据处理过程中,由于可能存在相机状态保持了很多个历元,但特征点没有更新和丢失(如静止一段时间),因此需要设定保持状态量的最大数目(如30,[5]中设定的),即滑动窗口的大小.
  - 筛选剔除状态的策略
    - 选择序列末尾的2个和比较新的2个(距离最新camera数据1帧的2帧),比较这四帧的信息



选择这4帧中包含信息最少的两个, 具体细节策略参考[5]msckf_vio.cpp中的findRedundantCamStates()函数

```
void MsckfVio::findRedundantCamStates(vector<StateIDType> &rm_cam_state_ids)
{
    // Move the iterator to the key position.
    auto key_cam_state_iter = state_server.cam_states.end();
    for (int i = 0; i < 4; ++i)
        --key_cam_state_iter;
    auto cam_state_iter = key_cam_state_iter;
    ++cam_state_iter;
    auto first_cam_state_iter = state_server.cam_states.begin();
    // Pose of the key camera state.
    const Vector3d key_position = key_cam_state_iter->second.position;
    const Matrix3d key_rotation = quaternionToRotation(key_cam_state_iter->secon
    // Mark the camera states to be removed based on the
    // motion between states.
    for (int i = 0; i < 2; ++i)
    {
        const Vector3d position =cam_state_iter->second.position;
        const Matrix3d rotation = quaternionToRotation(cam_state_iter->second.or

        double distance = (position - key_position).norm();
        double angle = AngleAxisd(rotation * key_rotation.transpose()).angle();
        if (angle < 0.2618 && distance < 0.4 && tracking_rate > 0.5)
        {
            rm_cam_state_ids.push_back(cam_state_iter->first);
            ++cam_state_iter;
        }
        else
        {
            rm_cam_state_ids.push_back(first_cam_state_iter->first);
            ++first_cam_state_iter;
        }
    }
}
```

    - 利用筛选出来的两帧图像都能观测到的特征点再进行一次量测更新,更新过程如上.只能被一个图像观测到的直接剔除
- **全部状态和点的更新** 将使用过的量测点和已经不包括任何点的camera state全部剔除.

参考文献/程序

[1] http://www.sohu.com/a/271224863_715754

[2] https://github.com/UMiNS/MSCKF_VIO_MONO

[3] Sun, Ke, et al. "Robust stereo visual inertial odometry for fast autonomous flight." IEEE Robotics and Automation Letters 3.2 (2018): 965-972.

[4] Mourikis, Anastasios I., and Stergios I. Roumeliotis. "A multi-state constraint Kalman filter for

vision-aided inertial navigation." Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, 2007.
[5] https://github.com/KumarRobotics/msckf_vio
[6] http://www.sohu.com/a/271370131_715754