# 一步步深入了解**S-MSCKF**

作者:钟心亮

联系方式:xinliangzhong@foxmail.com



## 0.前言

## 1.简介

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.据说这也是谷歌tango里面的算法，这要感觉Mingyang Li博士在MSCKF的不懈工作。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。

## 2.前端

本文主要针对2017年Kumar实验室开源的S-MSCKF进行详细分析,其实这篇文章整体思路与07年提出的基本上是一脉相承的.作为一个VIO的前端,MSCKF采用的是光流跟踪特征点的方法,特征点使用的是FAST特征,另外这是MSCKF双目的一个实现,双目之间的特征点匹配采用的也是光流,这与传统的基于descriptor匹配的方法不同.前端部分其实相对简单,整个前端部分基本在 **image_processor.cpp**中实现.

### 2.1 基本数据结构说明

**A.**特征点检测和跟踪的参数

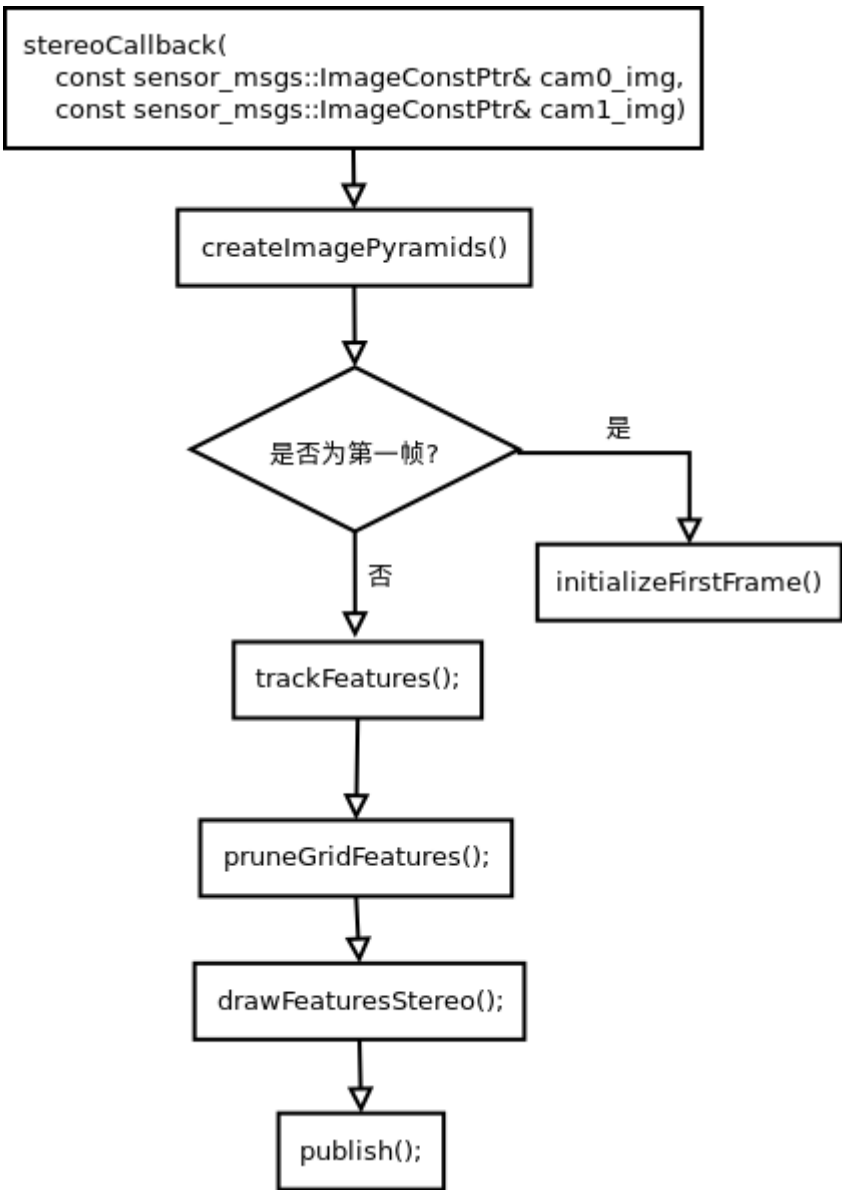
```
struct ProcessorConfig {
  int grid_row;               //划分图像网格的行数
  int grid_col;               //划分图像网格的列数
  int grid_min_feature_num;   //每个网格特征点的最小个数
  int grid_max_feature_num;   //每个网格特征点的最大个数

  int pyramid_levels;         //金字塔层数
  int patch_size;
  int fast_threshold;
  int max_iteration;
  double track_precision;
  double ransac_threshold;
  double stereo_threshold;
};
```

**B.** 特征点数据

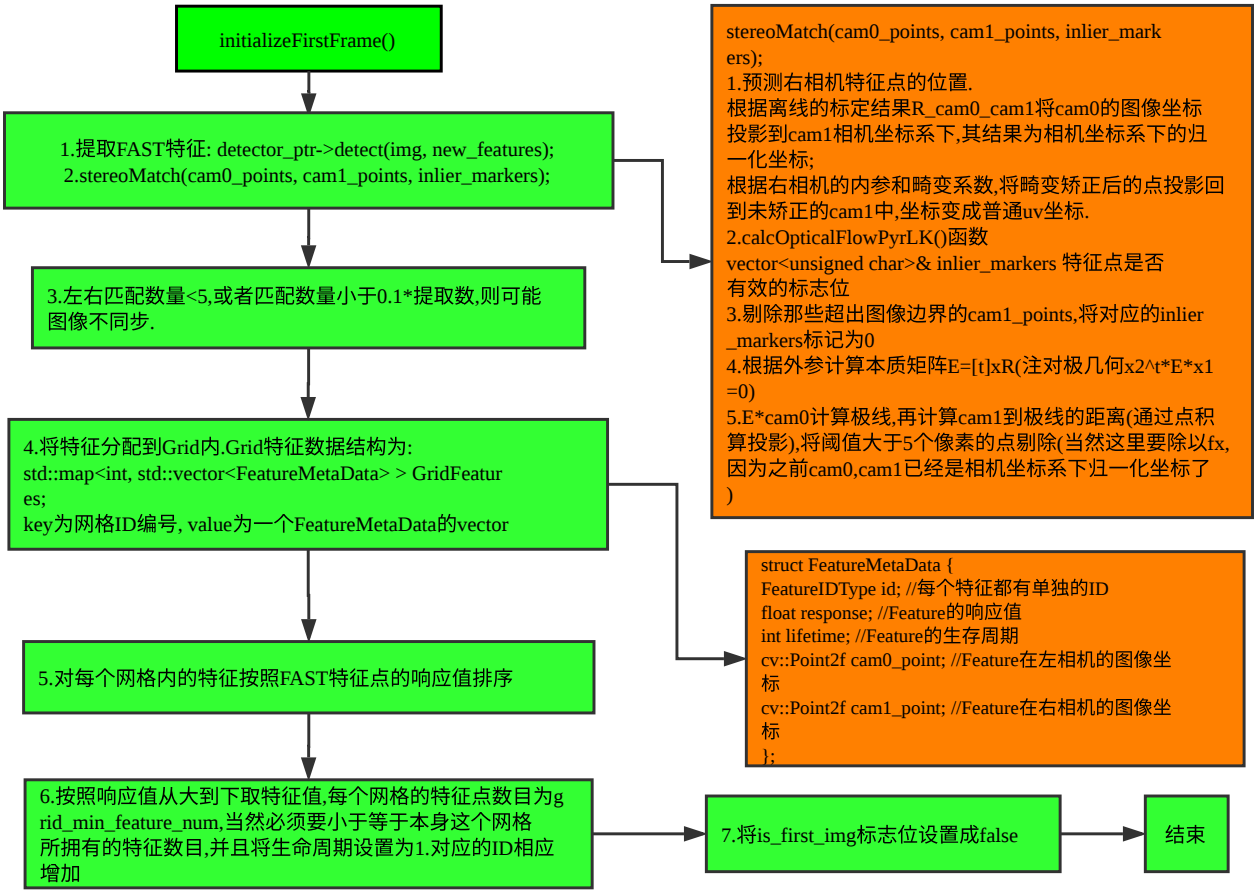
```
struct FeatureMetaData {
  FeatureIDType id;       //unsigned long long int 每个特征都有单独的ID
  float response;         //Feature的响应值
  int lifetime;           //Feature的生存周期
  cv::Point2f cam0_point; //Feature在左相机的图像坐标
  cv::Point2f cam1_point; //Feature在右相机的图像坐标
};
```

## 2.2 跟踪流程

整体框架如下面的流程图所示:

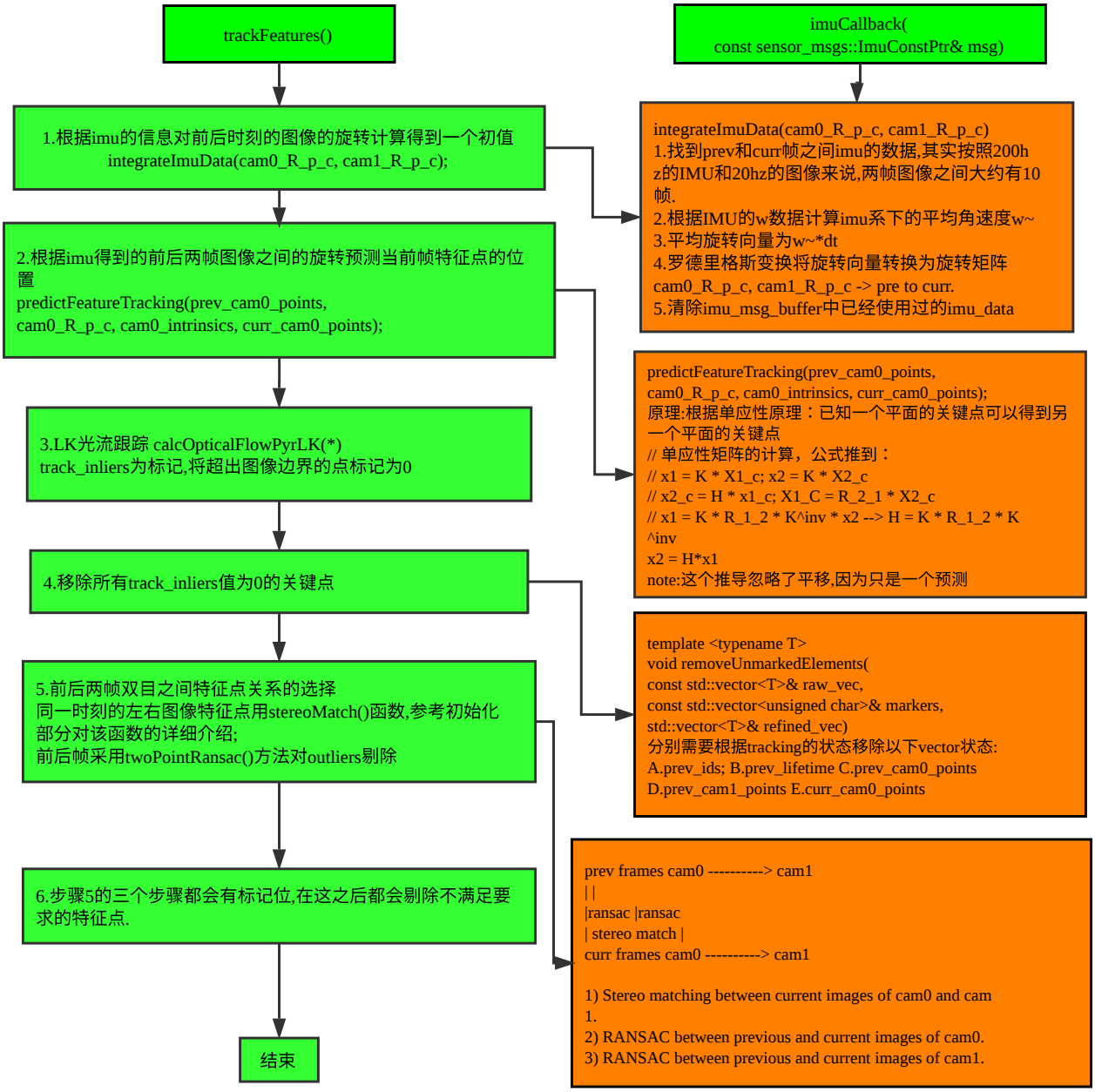



### 2.2.1 Initialization




### 2.2.2 trackFeatures

当第一帧初始化完成之后,后面帧则只需要进行跟踪第一帧的特征点,并且提取新的特征点,整个流程如下:

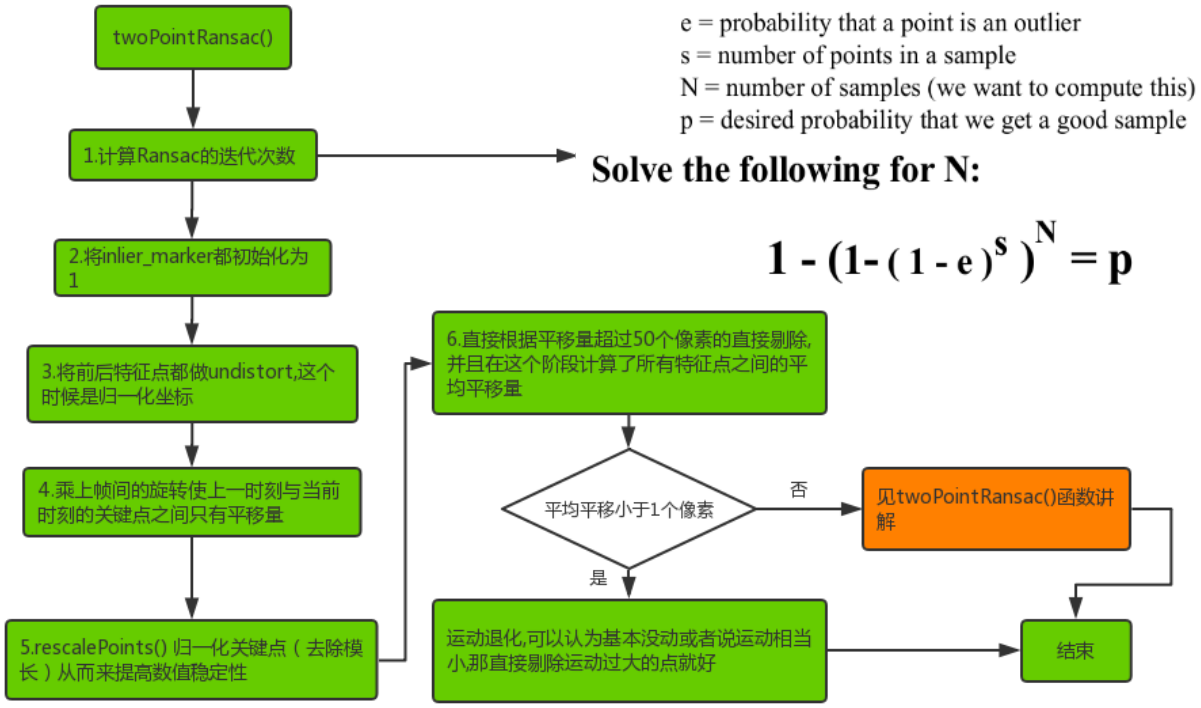



整个流程还算比较清晰,但是有一个步骤需要单独说明一下,也就是作者在论文中提到的twoPointRansac的方法.我们先来看一下函数原型:

```
/**
 * @brief 计算原图像帧关键点对应的矫正位置
 * @param pts1：上一时刻的关键点位置
 * @param pts2:当前时刻跟踪匹配到的关键点位置
 * @param R_p_c:根据imu信息计算得到的两个时刻相机的相对旋转信息
 * @param distortion_model,intrinsics：相机内参和畸变模型
 * @param inlier_error：内点可接受的阈值（关键点距离差）
 * @param success_probability：成功的概率
 * @return inlier_markers：内点标志位
 */
void ImageProcessor::twoPointRansac(
    const vector<Point2f>& pts1, const vector<Point2f>& pts2,
    const cv::Matx33f& R_p_c, const cv::Vec4d& intrinsics,
```

```
    const std::string& distortion_model,
    const cv::Vec4d& distortion_coeffs,
    const double& inlier_error,
    const double& success_probability,
    vector<int>& inlier_markers)
```

整个函数的基本流程如下:




下面我们来详细讲解一下RANSAC模型及原理依据: 我们由对极几何可以知道有以下约束:

$$p_2^T \cdot [t]_x \cdot R \cdot p_1 = 0 \tag{2.1}$$

我们假设前后帧对应点归一化坐标分别为,

$$R \cdot p_1 = [x1 \quad y1 \quad 1]^T, p_2 = [x2 \quad y2 \quad 1]^T \tag{2.2}$$

其中R为根据IMU的平均角速度得到的,此时坐标系都统一到一个坐标系下.

$$[x_2 \quad y_2 \quad 1] \cdot \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = 0 \tag{2.3}$$

将式子(2.3)展开之后我们可以得到:

$$[y_1 - y_2 \quad -(x_1 - x_2) \quad x_1 y_2 - x_2 y_2] \cdot \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = 0 \tag{2.4}$$

其中绿色部分在代码中对应这一块:

```
vector<Point2d> pts_diff(pts1_undistorted.size());
  for (int i = 0; i < pts1_undistorted.size(); ++i)
    pts_diff[i] = pts1_undistorted[i] - pts2_undistorted[i];
...
...
MatrixXd coeff_t(pts_diff.size(), 3);
  for (int i = 0; i < pts_diff.size(); ++i) {
    coeff_t(i, 0) = pts_diff[i].y;
    coeff_t(i, 1) = -pts_diff[i].x;
    coeff_t(i, 2) = pts1_undistorted[i].x*pts2_undistorted[i].y -
      pts1_undistorted[i].y*pts2_undistorted[i].x;
  }
```

至于这个模型是怎么选出来的呢? 假设一共有N个inliers点对,那么根据式(2.4)势必会得到一个N*3 * 3*1 = N(0)的等式.但事实上由于误差和outliers的存在,最终结果不可能为零,我们取两个点将式子分块,并且只考虑两个点的情况,那么将会有:

$$\begin{bmatrix} y_1 - y_2 & -(x_1 - x_2) & x_1 y_2 - x_2 y_2 \\ y_3 - y_4 & -(x_3 - x_4) & x_3 y_4 - x_4 y_3 \end{bmatrix} \cdot \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}^T \cdot \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.5}$$

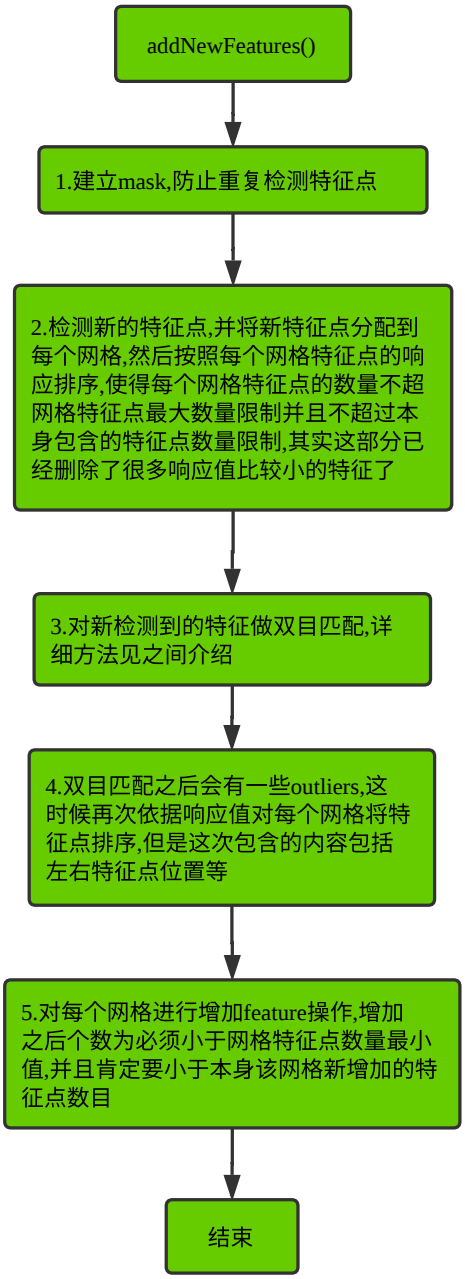
那我们可以分别得到以下三个式子:

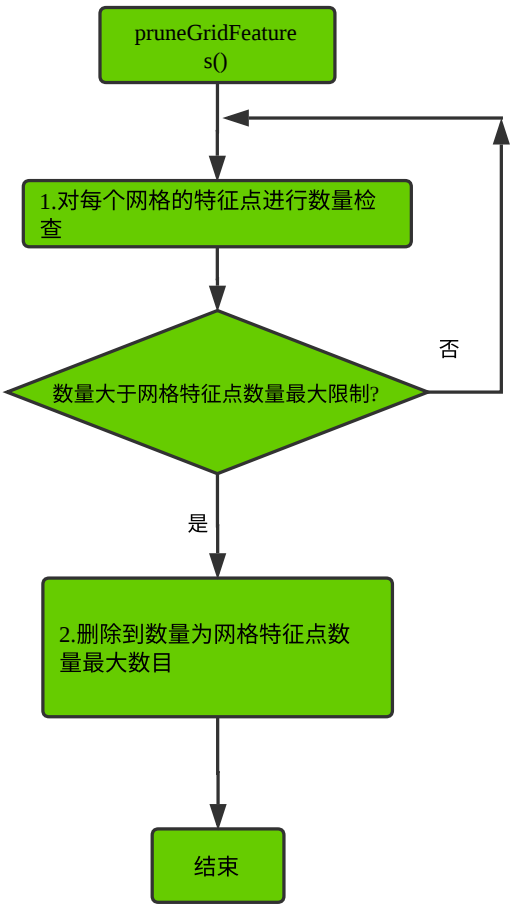
$$\begin{aligned} \begin{bmatrix} A_x \\ A_y \end{bmatrix}^T \cdot \begin{bmatrix} t_x \\ t_y \end{bmatrix} &\approx A_z \cdot t_z \\ \begin{bmatrix} A_x \\ A_z \end{bmatrix}^T \cdot \begin{bmatrix} t_x \\ t_z \end{bmatrix} &\approx A_y \cdot t_y \\ \begin{bmatrix} A_y \\ A_z \end{bmatrix}^T \cdot \begin{bmatrix} t_y \\ t_z \end{bmatrix} &\approx A_x \cdot t_x \end{aligned} \tag{2.6}$$

我们的目标当然是使得误差最小,所以作者的做法是比较式子(2.6)绿色部分的大小,取最小的并令模型的平移为1,进而直接求逆然后得到最初的模型假设,之后要做的步骤跟常规RANSAC就十分接近了,找出适应当前模型的所有inliers,然后计算误差并不断迭代找到最好的模型. 至此我们已经完成了整个feature的tracking过程!

### 2.2.3 addNewFeatures & pruneGridFeatures

如果一直tracking的话那么随着时间流逝,有些特征会消失,有些可能也会有累计误差,所以我们势必要添加一些新的特征,这个步骤正是在跟踪上一帧特征之后要做的,因为stereoMatching 和twoPointRansac都会剔除一些outliers,那我们需要提取新的特征保证能一直运行下去.

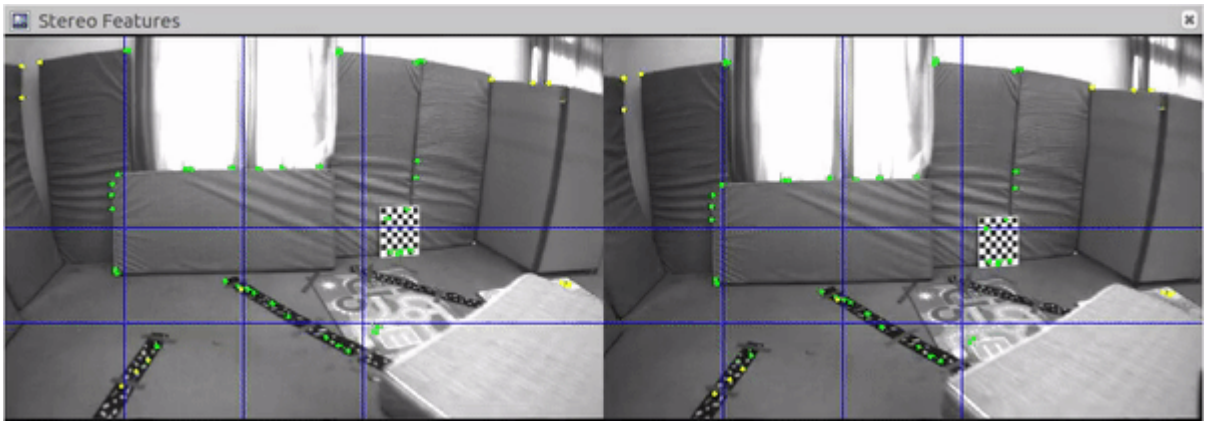



#### 2.2.4 publish

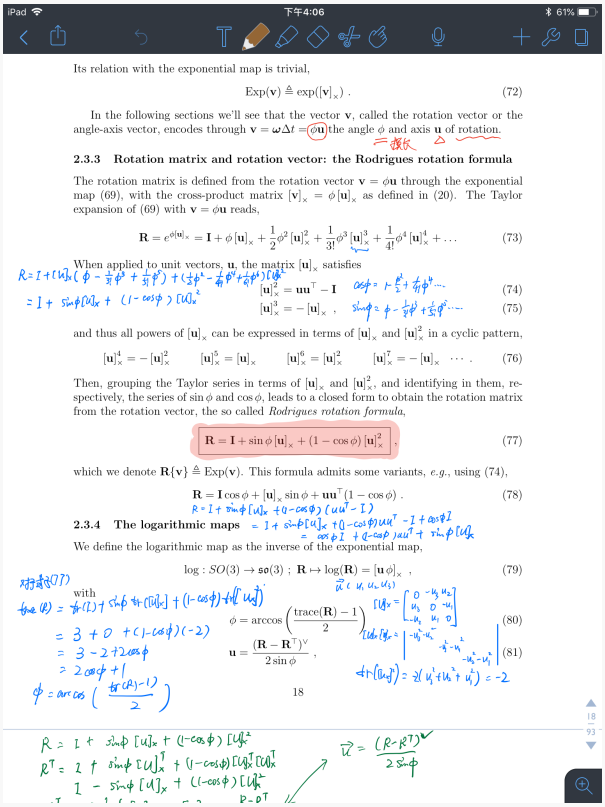
## 2.3 显示

其实前端基本上可以说是非常简单了,也没有太多的trick,最后我们来看一下前端的跟踪效果的动图:




## 3.基于四元数的 error-state Kalman Filter

其实原理部分相当重要,包括你对error-state Kalman Filter的理解,但是如果要从头讲起的话可以说篇幅太长,考虑到做SLAM的同学们基本上都应该知道这本书 Quaternion kinematics for the error-state Kalman filter, 这本书会让你在四元数,SO3,IMU的模型以及基于IMU的ESKF(error-state Kalman Filter)都会有比较深刻的理解,对应这本书我也做了很多注释,关于基于四元数原理部分由于篇幅限制,我这里不想做太多的说明,但是接下来在讲解S-MSCKF原理部分我们会将参考公式附上.下面是一些我在这本参考资料中的注释图.



Its relation with the exponential map is trivial,

$$\mathrm{Exp}(\mathbf{v}) \triangleq \exp([\mathbf{v}]_\times) \ . \tag{72}$$

In the following sections we'll see that the vector $\mathbf{v}$, called the rotation vector or the angle-axis vector, encodes through $\mathbf{v} = \boldsymbol{\omega}\Delta t = \phi\mathbf{u}$ the angle $\phi$ and axis $\mathbf{u}$ of rotation.

### 2.3.3 Rotation matrix and rotation vector: the Rodrigues rotation formula

The rotation matrix is defined from the rotation vector $\mathbf{v} = \phi\mathbf{u}$ through the exponential map (69), with the cross-product matrix $[\mathbf{v}]_\times = \phi[\mathbf{u}]_\times$ as defined in (20). The Taylor expansion of (69) with $\mathbf{v} = \phi\mathbf{u}$ reads,

$$\mathbf{R} = e^{\phi[\mathbf{u}]_\times} = \mathbf{I} + \phi[\mathbf{u}]_\times + \frac{1}{2}\phi^2[\mathbf{u}]_\times^2 + \frac{1}{3!}\phi^3[\mathbf{u}]_\times^3 + \frac{1}{4!}\phi^4[\mathbf{u}]_\times^4 + \dots \tag{73}$$

When applied to unit vectors, $\mathbf{u}$, the matrix $[\mathbf{u}]_\times$ satisfies

$$[\mathbf{u}]_\times^2 = \mathbf{u}\mathbf{u}^\top - \mathbf{I} \tag{74}$$

$$[\mathbf{u}]_\times^3 = -[\mathbf{u}]_\times \ , \tag{75}$$

and thus all powers of $[\mathbf{u}]_\times$ can be expressed in terms of $[\mathbf{u}]_\times$ and $[\mathbf{u}]_\times^2$ in a cyclic pattern,

$$[\mathbf{u}]_\times^4 = -[\mathbf{u}]_\times^2 \qquad [\mathbf{u}]_\times^5 = [\mathbf{u}]_\times \qquad [\mathbf{u}]_\times^6 = [\mathbf{u}]_\times^2 \qquad [\mathbf{u}]_\times^7 = -[\mathbf{u}]_\times \quad \cdots \ . \tag{76}$$

Then, grouping the Taylor series in terms of $[\mathbf{u}]_\times$ and $[\mathbf{u}]_\times^2$, and identifying in them, respectively, the series of $\sin\phi$ and $\cos\phi$, leads to a closed form to obtain the rotation matrix from the rotation vector, the so called *Rodrigues rotation formula*,

$$\boxed{\mathbf{R} = \mathbf{I} + \sin\phi\,[\mathbf{u}]_\times + (1 - \cos\phi)\,[\mathbf{u}]_\times^2} \ , \tag{77}$$

which we denote $\mathbf{R}\{\mathbf{v}\} \triangleq \mathrm{Exp}(\mathbf{v})$. This formula admits some variants, *e.g.*, using (74),

$$\mathbf{R} = \mathbf{I}\cos\phi + [\mathbf{u}]_\times \sin\phi + \mathbf{u}\mathbf{u}^\top(1 - \cos\phi) \ . \tag{78}$$

### 2.3.4 The logarithmic maps

We define the logarithmic map as the inverse of the exponential map,

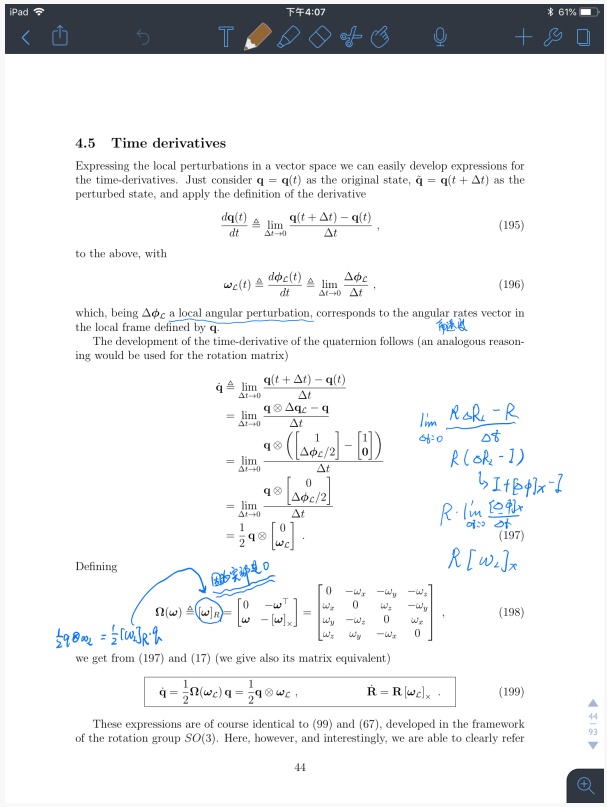
$$\log : SO(3) \to \mathfrak{so}(3) \ ; \ \mathbf{R} \mapsto \log(\mathbf{R}) = [\mathbf{u}\phi]_\times \ , \tag{79}$$

with

$$\phi = \arccos\left(\frac{\mathrm{trace}(\mathbf{R}) - 1}{2}\right) \tag{80}$$

$$\mathbf{u} = \frac{(\mathbf{R} - \mathbf{R}^\top)^\vee}{2\sin\phi} \ , \tag{81}$$





### 4.5 Time derivatives

Expressing the local perturbations in a vector space we can easily develop expressions for the time-derivatives. Just consider $\mathbf{q} = \mathbf{q}(t)$ as the original state, $\tilde{\mathbf{q}} = \mathbf{q}(t + \Delta t)$ as the perturbed state, and apply the definition of the derivative

$$\frac{d\mathbf{q}(t)}{dt} \triangleq \lim_{\Delta t \to 0} \frac{\mathbf{q}(t + \Delta t) - \mathbf{q}(t)}{\Delta t} \ , \tag{195}$$

to the above, with

$$\boldsymbol{\omega}_\mathcal{L}(t) \triangleq \frac{d\boldsymbol{\phi}_\mathcal{L}(t)}{dt} \triangleq \lim_{\Delta t \to 0} \frac{\Delta\boldsymbol{\phi}_\mathcal{L}}{\Delta t} \ , \tag{196}$$

which, being $\Delta\boldsymbol{\phi}_\mathcal{L}$ a local angular perturbation, corresponds to the angular rates vector in the local frame defined by $\mathbf{q}$.

The development of the time-derivative of the quaternion follows (an analogous reasoning would be used for the rotation matrix)

$$\begin{aligned}
\dot{\mathbf{q}} &\triangleq \lim_{\Delta t \to 0} \frac{\mathbf{q}(t + \Delta t) - \mathbf{q}(t)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\mathbf{q} \otimes \Delta\mathbf{q}_\mathcal{L} - \mathbf{q}}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\mathbf{q} \otimes \left(\begin{bmatrix} 1 \\ \Delta\boldsymbol{\phi}_\mathcal{L}/2 \end{bmatrix} - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}\right)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\mathbf{q} \otimes \begin{bmatrix} 0 \\ \Delta\boldsymbol{\phi}_\mathcal{L}/2 \end{bmatrix}}{\Delta t} \\
&= \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_\mathcal{L} \end{bmatrix} \ .
\end{aligned} \tag{197}$$

Defining

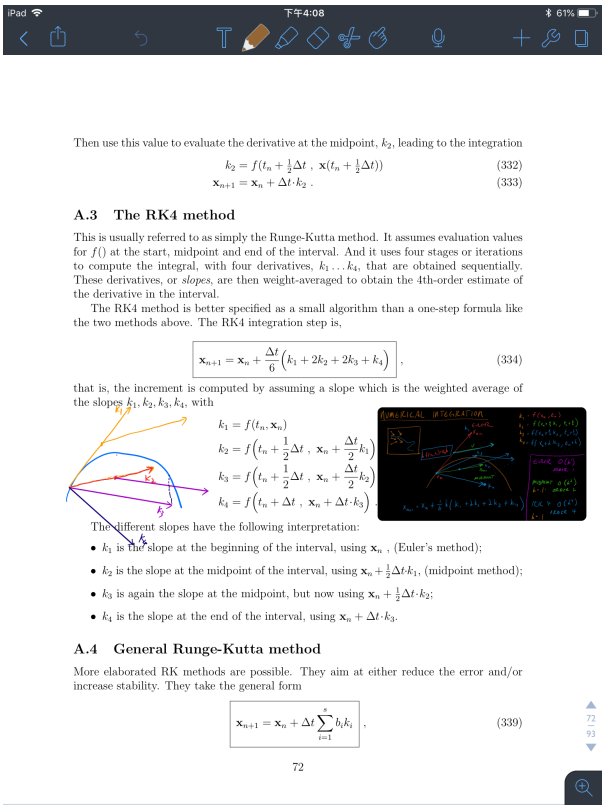
$$\boldsymbol{\Omega}(\boldsymbol{\omega}) \triangleq \begin{bmatrix} 0 & -\boldsymbol{\omega}^\top \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}]_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \ , \tag{198}$$

we get from (197) and (17) (we give also its matrix equivalent)

$$\boxed{\dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_\mathcal{L})\,\mathbf{q} = \frac{1}{2}\mathbf{q} \otimes \boldsymbol{\omega}_\mathcal{L} \ , \qquad \dot{\mathbf{R}} = \mathbf{R}\,[\boldsymbol{\omega}_\mathcal{L}]_\times \ .} \tag{199}$$

These expressions are of course identical to (99) and (67), developed in the framework of the rotation group $SO(3)$. Here, however, and interestingly, we are able to clearly refer





Then use this value to evaluate the derivative at the midpoint, $k_2$, leading to the integration

$$k_2 = f(t_n + \tfrac{1}{2}\Delta t \ , \ \mathbf{x}(t_n + \tfrac{1}{2}\Delta t)) \tag{332}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \cdot k_2 \ . \tag{333}$$

## A.3 The RK4 method

This is usually referred to as simply the Runge-Kutta method. It assumes evaluation values for $f()$ at the start, midpoint and end of the interval. And it uses four stages or iterations to compute the integral, with four derivatives, $k_1 \dots k_4$, that are obtained sequentially. These derivatives, or *slopes*, are then weight-averaged to obtain the 4th-order estimate of the derivative in the interval.

The RK4 method is better specified as a small algorithm than a one-step formula like the two methods above. The RK4 integration step is,

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)} \ , \tag{334}$$

that is, the increment is computed by assuming a slope which is the weighted average of the slopes $k_1, k_2, k_3, k_4$, with



$$k_1 = f(t_n, \mathbf{x}_n)$$

$$k_2 = f\left(t_n + \frac{1}{2}\Delta t \ , \ \mathbf{x}_n + \frac{\Delta t}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{1}{2}\Delta t \ , \ \mathbf{x}_n + \frac{\Delta t}{2}k_2\right)$$

$$k_4 = f\left(t_n + \Delta t \ , \ \mathbf{x}_n + \Delta t \cdot k_3\right)$$



The different slopes have the following interpretation:

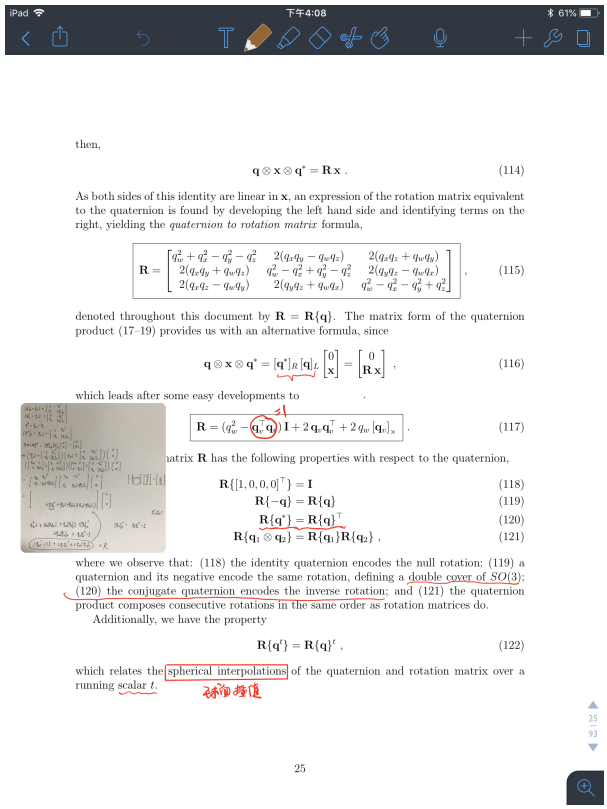
- $k_1$ is the slope at the beginning of the interval, using $\mathbf{x}_n$ , (Euler's method);
- $k_2$ is the slope at the midpoint of the interval, using $\mathbf{x}_n + \frac{1}{2}\Delta t \cdot k_1$, (midpoint method);
- $k_3$ is again the slope at the midpoint, but now using $\mathbf{x}_n + \frac{1}{2}\Delta t \cdot k_2$;
- $k_4$ is the slope at the end of the interval, using $\mathbf{x}_n + \Delta t \cdot k_3$.

## A.4 General Runge-Kutta method

More elaborated RK methods are possible. They aim at either reduce the error and/or increase stability. They take the general form

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \sum_{i=1}^{s} b_i k_i} \ , \tag{339}$$





then,

$$\mathbf{q} \otimes \mathbf{x} \otimes \mathbf{q}^* = \mathbf{R}\,\mathbf{x} \ . \tag{114}$$

As both sides of this identity are linear in $\mathbf{x}$, an expression of the rotation matrix equivalent to the quaternion is found by developing the left hand side and identifying terms on the right, yielding the *quaternion to rotation matrix* formula,

$$\boxed{\mathbf{R} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}} \ , \tag{115}$$

denoted throughout this document by $\mathbf{R} = \mathbf{R}\{\mathbf{q}\}$. The matrix form of the quaternion product (17–19) provides us with an alternative formula, since

$$\mathbf{q} \otimes \mathbf{x} \otimes \mathbf{q}^* = [\mathbf{q}^*]_R\,[\mathbf{q}]_L \begin{bmatrix} 0 \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{R}\,\mathbf{x} \end{bmatrix} \ , \tag{116}$$

which leads after some easy developments to

$$\boxed{\mathbf{R} = (q_w^2 - \mathbf{q}_v^\top\mathbf{q}_v)\,\mathbf{I} + 2\,\mathbf{q}_v\mathbf{q}_v^\top + 2\,q_w\,[\mathbf{q}_v]_\times} \ . \tag{117}$$



matrix $\mathbf{R}$ has the following properties with respect to the quaternion,

$$\mathbf{R}\{[1, 0, 0, 0]^\top\} = \mathbf{I} \tag{118}$$

$$\mathbf{R}\{-\mathbf{q}\} = \mathbf{R}\{\mathbf{q}\} \tag{119}$$

$$\mathbf{R}\{\mathbf{q}^*\} = \mathbf{R}\{\mathbf{q}\}^\top \tag{120}$$

$$\mathbf{R}\{\mathbf{q}_1 \otimes \mathbf{q}_2\} = \mathbf{R}\{\mathbf{q}_1\}\mathbf{R}\{\mathbf{q}_2\} \ , \tag{121}$$

where we observe that: (118) the identity quaternion encodes the null rotation; (119) a quaternion and its negative encode the same rotation, defining a double cover of $SO(3)$; (120) the conjugate quaternion encodes the inverse rotation; and (121) the quaternion product composes consecutive rotations in the same order as rotation matrices do.

Additionally, we have the property

$$\mathbf{R}\{\mathbf{q}^t\} = \mathbf{R}\{\mathbf{q}\}^t \ , \tag{122}$$

which relates the spherical interpolations of the quaternion and rotation matrix over a running scalar $t$.



当然重要的事情需要说三遍,那就是Quaternion kinematics for the error-state Kalman filter四元数是Hamilton表示方法,而MSCKF中采用的是JPL表示方法,关于这两者的不同,可以参考书中34页,Table2.

看过论文代码的同学可能会说,MSCKF这一部分代码参考的是MARS实验室Indirect Kalman Filter for 3D Attitude Estimation-A Tutorial for Quaternion Algebra,答案是肯定的.但是我的建议是以Hamilton那本书为基础,然后自行再去推导MARS出品那一本,那样你的体验会更加深刻.

# 4.S-MSCKF

巴拉巴拉这里放一点大致性的简介

## 4.1 基本原理

在讲解之前,我们先来定义一下坐标系,如下图所示:

作为一个滤波器,我们首先来看滤波器的状态向量,它包含两个部分,IMU状态和Camera状态:

$$\mathbf{x}_I = \begin{bmatrix} {}^I_G\textbf{q}^T & \end{bmatrix}$$

## 4.2三角化

## 4.2 能观测性分析

# 5.S-MSCKF代码流程

# 6.总结