代码解读 | VINS 视觉前端

vins前端概述

在搞清楚VINS前端之前，首先要搞清楚什么是SLAM前端?

SLAM的前端、后端系统本身没有特别明确的划分，但是在实际研究中根据处理的先后顺序一般认为特征点提取和跟踪为前端部分，然后利用前端获取的数据进行优化、回环检测等操作，从而将优化、回环检测等作为后端。

而在VINS_MONO中将视觉跟踪模块（feature_trackers）为其前端。在视觉跟踪模块中，首先，对于每一幅新图像，KLT稀疏光流算法对现有特征进行跟踪。然后，检测新的角点特征以保证每个图像特征的最小数目，并设置两个相邻特征之间像素的最小间隔来执行均匀的特征分布。接着，将二维特征点去畸变，然后在通过外点剔除后投影到一个单位球面上。最后，利用基本矩阵模型的RANSAC算法进行外点剔除。

VINS_MONO原文中还将关键帧的选取作为前端分，本文暂不讨论， 后续文章会详细介绍。

VINS-Mono将前端封装为一个ROS节点，该节点的实现在feature_tracker目录下的src中，src里共有3个头文件和3个源文件：

- feature_tracker_node.cpp构造了一个ROS节点feature_tracker_node，该节点订阅相机图像话题数据后，提取特征点，然后用KLT光流进行特征点跟踪。feature_tracker节点将跟踪的特征点作为话题进行发布，供后端ROS节点使用。同时feature_tracker_node还会发布标记了特征点的图片，可供Rviz显示以供调试。

- 如下表所示

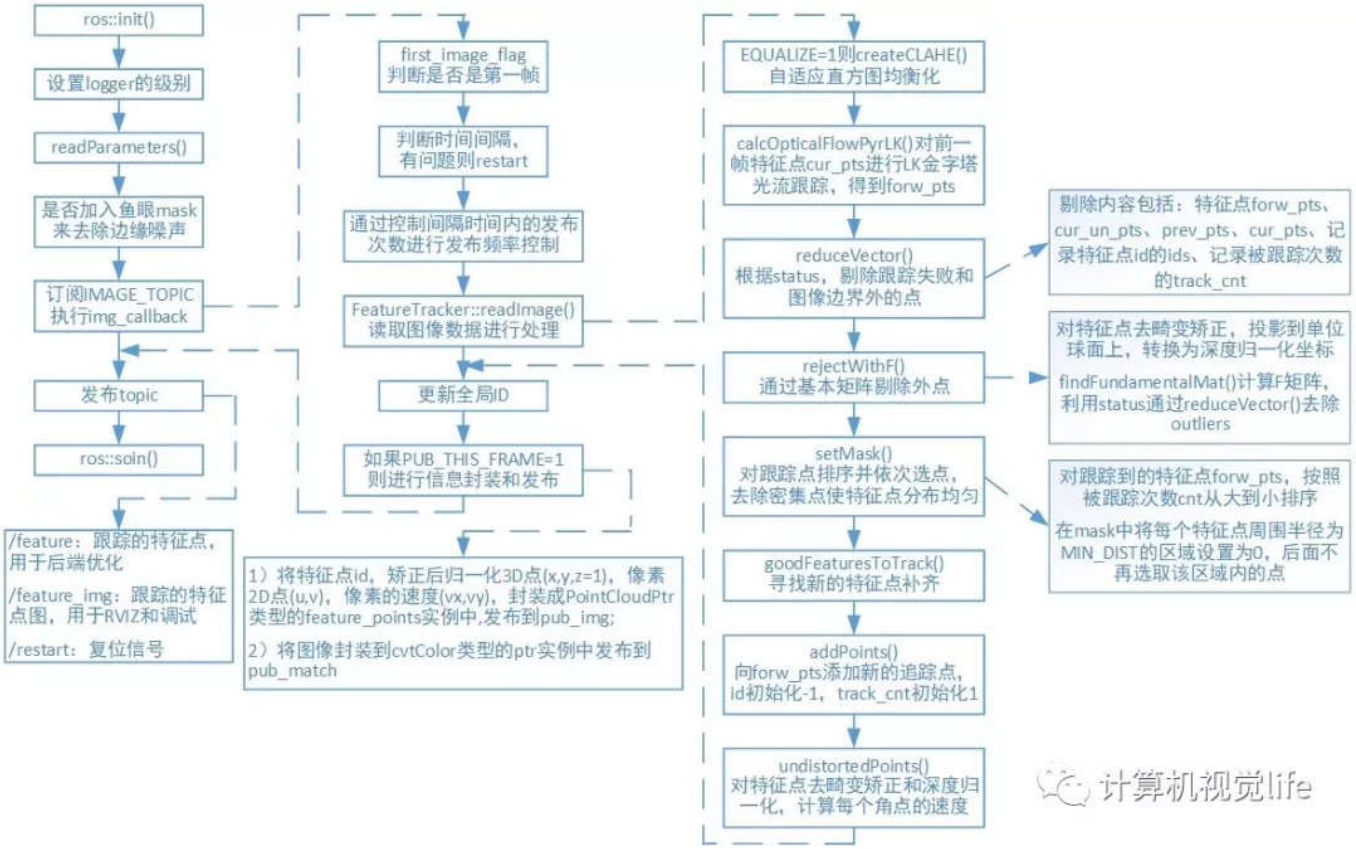
操作	话题	消息类型	功能
Subscribe	image	sensor_msgs::ImageConstPtr	订阅原始图像，传给回调函数
Publish	feature	sensor_msgs::PointCloud	跟踪的特征点，供后端优化使用
Publish	feature_img	sensor_msgs::Image	跟踪特征点图片，输出给RViz，调试用

feature_tracker.h和feature_tracker.cpp实现了一个类FeatureTracker，用来完成特征点提取和特征点跟踪等主要功能，该类中主要函数和实现的功能如下

函数	功能
bool **inBorder**()	*判断跟踪的特征点是否在图像边界内*
void **reduceVector**()	*去除无法跟踪的特征点*
void **FeatureTracker**::**setMask**()	*对跟踪点进行排序并去除密集点*
void **FeatureTracker**::**addPoints**()	*添将新检测到的特征点n_pts*
void **FeatureTracker**::**readImage**()	*对图像使用光流法进行特征点跟踪*
void **FeatureTracker**::**rejectWithF**()	*利用F矩阵剔除外点*
bool **FeatureTracker**::**updateID**()	*更新特征点id*
void **FeatureTracker**::**readIntrinsicParameter**()	*读取相机内参*
void **FeatureTracker**::**showUndistortion**()	*显示去畸变矫正后的特征点*
void **FeatureTracker**::**undistortedPoints**()	*对角点进行去畸变矫正，并计算每个角点的速*

- tic_toc.h中是作者自己封装的一个类TIC_TOC，用来计时；
- parameters.h和parameters.cpp处理前端中需要用到的一些参数；

流程图



代码解读

feature_tracker_node系统入口main() 函数：

1. ROS初始化和输出调试信息(可左右滑动)：

```
//ros初始化和设置句柄
ros::init(argc, argv, "feature_tracker");
ros::NodeHandle n("~");
//设置logger的级别。 只有级别大于或等于level的日志记录消息才会得到处理。
ros::console::set_logger_level(ROSCONSOLE_DEFAULT_NAME, ros::console::levels::Info)
```

2. 读取配置参数：

```
//读取config->euroc->euroc_config.yaml中的一些配置参数
readParameters(n);
```

3. 读取相机内参读取每个相机对应内参，单目时**NUM_OF_CAM=1**：

```
for (int i = 0; i < NUM_OF_CAM; i++)
   trackerData[i].readIntrinsicParameter(CAM_NAMES[i]);
```

4. 判断是否加入鱼眼mask来去除边缘噪声

5. 订阅话题IMAGE_TOPIC，当有图像进来的时候执行回调函数：

```
ros::Subscriber sub_img = n.subscribe(IMAGE_TOPIC, 100, img_callback);
```

6. 将处理完的图像信息用PointCloud实例feature_points和Image的实例ptr消息类型，发布到"feature"和"feature_img"的topic

```
pub_img = n.advertise<sensor_msgs::PointCloud>("feature", 1000);
pub_match = n.advertise<sensor_msgs::Image>("feature_img",1000);
pub_restart = n.advertise<std_msgs::Bool>("restart",1000);
```

回调函数imf_callback

1. 判断是否为第一帧，若为第一帧，将该帧的时间赋给 **first_image_time**和**last_image_time** ，然后返回

```
if(first_image_flag)
    {
        first_image_flag = false;
        first_image_time = img_msg->header.stamp.toSec();//记录图像帧的时间
        last_image_time = img_msg->header.stamp.toSec();
        return;
    }
```

2. 通过判断时间间隔，有问题则restart

```
if (img_msg->header.stamp.toSec() - last_image_time > 1.0 || img_msg->header.stamp.
```

3. 发布频率控制（不是每来一张图像都要发布，但是都要传入readImage()进行处理），保证每秒钟处理的图像不超过FREQ，此处为每秒10帧

```
if (round(1.0 * pub_count / (img_msg->header.stamp.toSec() - first_image_time)) <=
{
    PUB_THIS_FRAME = true;
    // 时间间隔内的发布频率十分接近设定频率时，更新时间间隔起始时刻，并将数据发布次数置0
   if (abs(1.0 * pub_count / (img_msg->header.stamp.toSec() - first_image_time) - FR
    {
    first_image_time = img_msg->header.stamp.toSec();
    pub_count = 0;
    }
}
else
    PUB_THIS_FRAME = false;
```

4. 将图像编码8UC1转换为mono8

5. 处理图片：**readImage**()

6. 判断是否显示去畸变矫正后的特征点

7. 更新全局ID，将新提取的特征点赋予全局id

```
for (unsigned int i = 0;; i++)
{
   bool completed = false;
   for (int j = 0; j < NUM_OF_CAM; j++)
       if (j != 1 || !STEREO_TRACK)
           completed |= trackerData[j].updateID(i);
   if (!completed)
       break;
}
```

8. 将特征点id，矫正后归一化平面的3D点(x,y,z=1)，像素2D点(u,v)，像素的速度(vx,vy)，封装成sensor_msgs::PointCloudPtr类型的feature_points实例中，发布到pub_img，将图像封装到cv_bridge::cvtColor类型的ptr实例中发布到pub_match

9. 发布消息的数据：

   pub_img.publish(feature_points)

   pub_match.publish(ptr->toImageMsg())

readimage()

1. 判断EQUALIZE的值，决定是否对图像进行直方图均衡化处理：**createCLAHE**()

2. 若为第一次读入图片，则：**prev_img = cur_img = forw_img = img**；若不是第一帧，则：**forw_img = img**，其中cur_img 和 forw_img 分别是光流跟踪的前后两帧，forw_img 才是真正的当前帧，cur_img 实际上是上一帧，prev_img 是上一次发布的帧。

```
prev_img = cur_img = forw_img = img;//避免后面使用到这些数据时，它们是空的
```

3. 调用 **cv::calcOpticalFlowPyrLK()**进行光流跟踪，跟踪前一帧的特征点 cur_pts 得到 forw_pts，根据 status 把跟踪失败的点剔除(注意 prev, cur, forw, ids, track_cnt都要剔除)，而且还需要将跟踪到图像边界外的点剔除。

```
cv::calcOpticalFlowPyrLK(cur_img, forw_img, cur_pts, forw_pts, status, err, cv::Siz
```

4. 判断是否需要发布该帧图像：

   否(**PUB_THIS_FRAME=0**)：当前帧 forw 的数据赋给上一帧 cur，然后在这一步就结束了。

   是(**PUB_THIS_FRAME=0**)：

   4.1、 调用**rejectWithF()**对prev_pts和forw_pts做RANSAC剔除outlier，函数里面主要是调用了**cv::findFundamentalMat()** 函数，然后将然后所有剩下的特征点的 track_cnt 加1，track_cnt数值越大，说明被追踪得越久.

   (可左右滑动)

```
void FeatureTracker::rejectWithF()
{
    if (forw_pts.size() >= 8)
    {
        ROS_DEBUG("FM ransac begins");
        TicToc t_f;

        vector<cv::Point2f> un_cur_pts(cur_pts.size()), un_forw_pts(forw_pts.size());
        for (unsigned int i = 0; i < cur_pts.size(); i++)
        {
            Eigen::Vector3d tmp_p;
            //根据不同的相机模型将二维坐标转换到三维坐标
            m_camera->liftProjective(Eigen::Vector2d(cur_pts[i].x, cur_pts[i].y), tmp_
            //转换为归一化像素坐标
            tmp_p.x() = FOCAL_LENGTH * tmp_p.x() / tmp_p.z() + COL / 2.0;
            tmp_p.y() = FOCAL_LENGTH * tmp_p.y() / tmp_p.z() + ROW / 2.0;
            un_cur_pts[i] = cv::Point2f(tmp_p.x(), tmp_p.y());

            m_camera->liftProjective(Eigen::Vector2d(forw_pts[i].x, forw_pts[i].y), tm
            tmp_p.x() = FOCAL_LENGTH * tmp_p.x() / tmp_p.z() + COL / 2.0;
            tmp_p.y() = FOCAL_LENGTH * tmp_p.y() / tmp_p.z() + ROW / 2.0;
            un_forw_pts[i] = cv::Point2f(tmp_p.x(), tmp_p.y());
        }

        vector<uchar> status;
        //调用cv::findFundamentalMat对un_cur_pts和un_forw_pts计算F矩阵
        cv::findFundamentalMat(un_cur_pts, un_forw_pts, cv::FM_RANSAC, F_THRESHOLD, 0.
        int size_a = cur_pts.size();
        reduceVector(prev_pts, status);
        reduceVector(cur_pts, status);
        reduceVector(forw_pts, status);
        reduceVector(cur_un_pts, status);
        reduceVector(ids, status);
        reduceVector(track_cnt, status);
        ROS_DEBUG("FM ransac: %d -> %lu: %f", size_a, forw_pts.size(), 1.0 * forw_pts.
        ROS_DEBUG("FM ransac costs: %fms", t_f.toc());
    }
}
```

4.2、 调用**setMask**()函数，先对跟踪到的特征点 forw_pts 按照跟踪次数降序排列(认为特征点被跟踪到的次数越多越好)，然后遍历这个降序排列，对于遍历的每一个特征点，在 mask中将该点周围半径为 MIN_DIST=30 的区域设置为 0，在后续的遍历过程中，不再选择该区域内的点。

4.3、 在mask中不为0的区域，调用**goodFeaturesToTrack**提取新的角点n_pts，通过addPoints()函数push到forw_pts中，id初始化-1，track_cnt初始化为1（由于跟踪过程中，上一帧特征点由于各种原因无法被跟踪，而且为了保证特征点均匀分布而剔除了一些特征点，如果不补充新的特征点，那么每一帧中特征点的数量会越来越少）。

```
cv::goodFeaturesToTrack(forw_img, n_pts, MAX_CNT - forw_pts.size(), 0.01, MIN_DIST, ma
```

5. 调用**undistortedPoints**() 函数根据不同的相机模型进行去畸变矫正和深度归一化，计算速度。

reference

1. https://github.com/QingSimon/VINS-Mono-code-annotation/blob/master/VINS-Mono%E8%AF%A6%E8%A7%A3.pdf

2. https://blog.csdn.net/wangshuailpp/article/details/78461171

3. https://blog.csdn.net/qq_41839222/article/details/85797156

4. https://qingsimon.github.io/post/