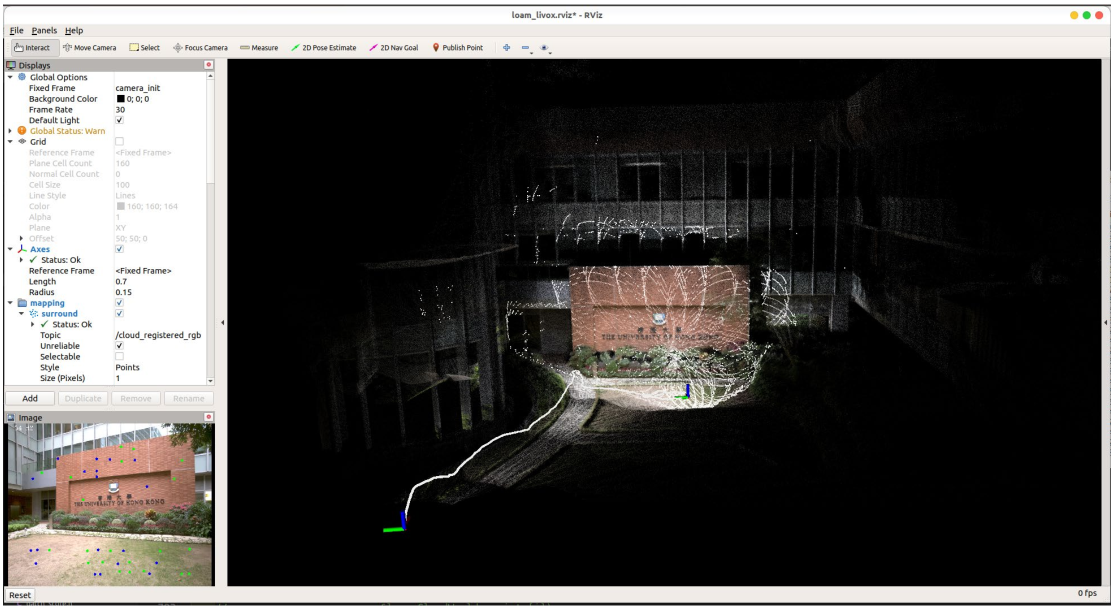
Fast-livo



Important parameters

Edit `config/xxx.yaml` to set the below parameters:

- `lid_topic`: The topic name of LiDAR data.
- `imu_topic`: The topic name of IMU data.
- `img_topic`: The topic name of camera data.
- `img_enable`: Enbale vio submodule.
- `lidar_enable`: Enbale lio submodule.
- `point_filter_num`: The sampling interval for a new scan. It is recommended that `3~4` for faster odometry, `1~2` for denser map.
- `outlier_threshold`: The outlier threshold value of photometric error (square) of a single pixel. It is recommended that `50~250` for the darker scenes, `500~1000` for the brighter scenes. The smaller the value is, the

faster the vio submodule is, but the weaker the anti-degradation ability is.

- `img_point_cov`: The covariance of photometric errors per pixel.
- `laser_point_cov`: The covariance of point-to-plane redisual per point.
- `filter_size_surf`: Downsample the points in a new scan. It is recommended that `0.05~0.15` for indoor scenes, `0.3~0.5` for outdoor scenes.
- `filter_size_map`: Downsample the points in LiDAR global map. It is recommended that `0.15~0.3` for indoor scenes, `0.4~0.5` for outdoor scenes.

先来看看NodeHandle类的主要成员函数：

发布话题，返回一个Publisher，负责广播topic

Publisher **advertise** (const std::string &topic, uint32_t queue_size, bool latch=false)

订阅一个话题，收到话题中的消息后触发回调函数

Subscriber **subscribe** (const std::string &topic, uint32_t queue_size, void(T::*fp)(M), T *obj, const **TransportHints** &transport_hints=**TransportHints**())

类似于发布话题，还可以发布服务

ServiceServer **advertiseService** (const std::string &service, bool(T::*srv_func)(MReq &, MRes &), T *obj)

客户端通过调用服务节点完成某项任务

ServiceClient **serviceClient** (const std::string &service_name, bool persistent=false, const M_string &header_values=M_string())

创建定时器，按一定周期执行指定的函数

Timer **createTimer** (Rate r, Handler h, Obj o, bool oneshot=false, bool autostart=true) const

从参数服务中获得某个参数

bool **getParam** (const std::string &key, std::string &s) const

对应的就是设置参数

void **setParam** (const std::string &key, const char *s) const

- **main()**

//初始化，节点名为 laserMapping，为基本名称（不能包含于命名空间）

ros::init(argc, argv, "laserMapping");

//通过 ros::NodeHandle，读取参数，否则传入默认值

nh.param<int>("dense_map_enable",dense_map_en,1);

其中subscribe有很多重定义。例如:

```
Subscriber ros::NodeHandle::subscribe    (    const std::string & topic,
                                               uint32_t     queue_size,
                                               void(*)(M)      fp,
                                               const TransportHints &      transport_hints = TransportHints()
                                          )
```

Parameters:

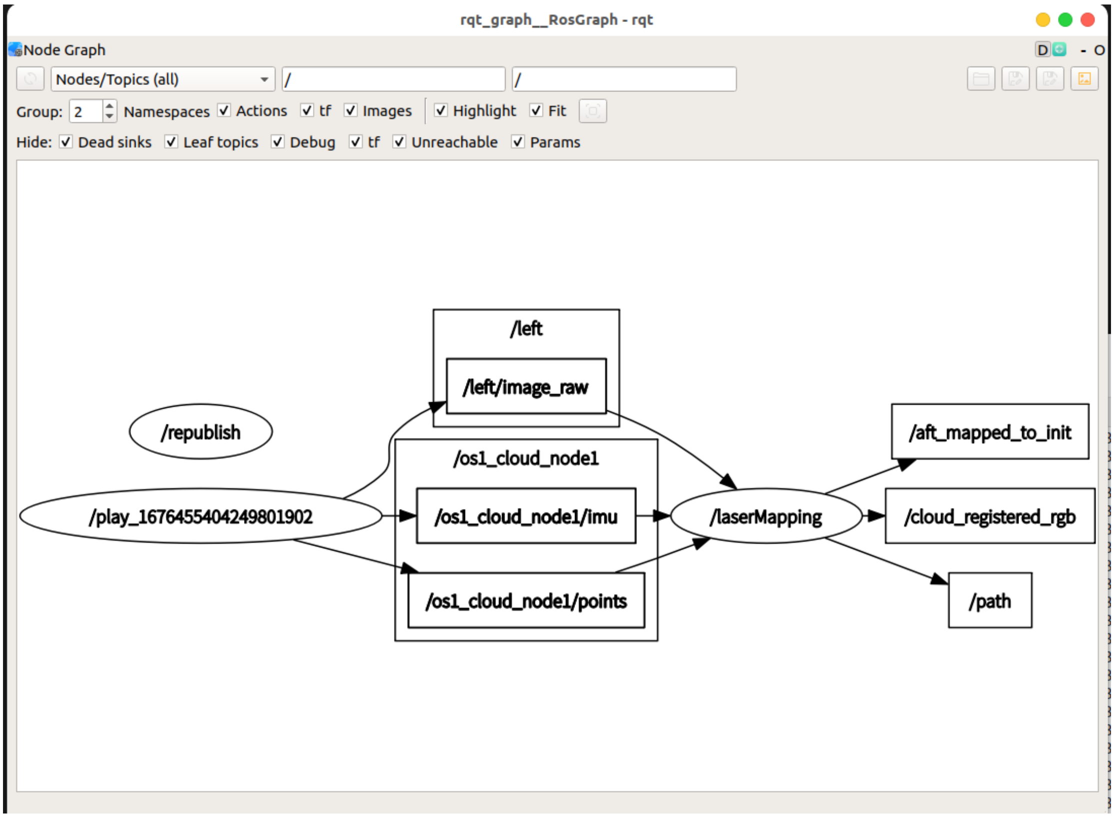
M	[template] M here is the callback parameter type (e.g. const boost::shared_ptr<M const>& or const M&), **not** the message type, and should almost always be deduced
topic	Topic to subscribe to
queue_size	Number of incoming messages to queue up for processing (messages in excess of this queue capacity will be discarded).
fp	Function pointer to call when a message has arrived
transport_hints	a TransportHints structure which defines various transport-related options

其中的参数:

topic 为订阅的节点名，字符串类型。

queue_size 为待处理信息队列大小。

fp 当消息传入时，可以调用的函数指针，即回调函数。



//preprocess.h 中定义的一些变量

```cpp
#define IS_VALID(a)  ((abs(a)>1e8) ? true : false)

typedef pcl::PointXYZINormal PointType;
typedef pcl::PointCloud<PointType> PointCloudXYZI;

enum LID_TYPE{AVIA = 1, VELO16, OUST64}; //{1, 2, 3}
enum Feature{Nor, Poss_Plane, Real_Plane, Edge_Jump, Edge_Plane, Wire, ZeroPoint};
enum Surround{Prev, Next};
enum E_jump{Nr_nor, Nr_zero, Nr_180, Nr_inf, Nr_blind};
```

//通过 lidar 类型判断激光回调函数

```cpp
void livox_pcl_cbk(const livox_ros_driver::CustomMsg::ConstPtr &msg)
void standard_pcl_cbk(const sensor_msgs::PointCloud2::ConstPtr &msg)
```

//图像和 imu 的回调函数仅一种

```cpp
ros::Subscriber sub_imu = nh.subscribe(imu_topic, 200000, imu_cbk);
ros::Subscriber sub_img = nh.subscribe(img_topic, 200000, img_cbk);
```

//然后发布一些点云，图像，轨迹的话题

```cpp
image_transport::Publisher img_pub = it.advertise("/rgb_img", 1);
ros::Publisher pubLaserCloudFullRes = nh.advertise<sensor_msgs::PointCloud2>
        ("/cloud_registered", 100);
ros::Publisher pubLaserCloudFullResRgb = nh.advertise<sensor_msgs::PointCloud2>
        ("/cloud_registered_rgb", 100);
ros::Publisher pubVisualCloud = nh.advertise<sensor_msgs::PointCloud2>
        ("/cloud_visual_map", 100);
ros::Publisher pubSubVisualCloud = nh.advertise<sensor_msgs::PointCloud2>
        ("/cloud_visual_sub_map", 100);
ros::Publisher pubLaserCloudEffect = nh.advertise<sensor_msgs::PointCloud2>
        ("/cloud_effected", 100);
ros::Publisher pubLaserCloudMap = nh.advertise<sensor_msgs::PointCloud2>
        ("/Laser_map", 100);
ros::Publisher pubOdomAftMapped = nh.advertise<nav_msgs::Odometry>
        ("/aft_mapped_to_init", 10);
ros::Publisher pubPath          = nh.advertise<nav_msgs::Path>
        ("/path", 10);
```

//变量定义（不使用 IKFOM 的情况）

```cpp
/*** variables definition ***/
#ifndef USE_IKFOM
VD(DIM_STATE) solution;
MD(DIM_STATE, DIM_STATE) G, H_T_H, I_STATE;
V3D rot_add, t_add;
StatesGroup state_propagat;
```

PointType pointOri, pointSel, coeff;
#endif

//设置点云降采样的体素分割

downSizeFilterSurf.setLeafSize(filter_size_surf_min, filter_size_surf_min, filter_size_surf_min);
downSizeFilterMap.setLeafSize(filter_size_map_min, filter_size_map_min, filter_size_map_min);

//循环处理，收集测量信息进入 LidarMeasureGroup 结构。首先判断雷达，无雷达忽略图像。有雷达时判断图像，若无图像，保留 IMU 信息，注意 IMU 的信息要比雷达大已完成完整的状态传播；有图像判断雷达和图像时戳判断处理哪个传感器帧

bool sync_packages(LidarMeasureGroup &meas)

//IMU 处理，先判断是否需要初始化

void ImuProcess::Process2(LidarMeasureGroup &lidar_meas, StatesGroup &stat, PointCloudXYZI::Ptr cur_pcl_un_)

IMU 迭代初始化

1. 初始化重力、陀螺偏置、加计和陀螺协方差

2. 将加速度测量值归一化为单位重力

$$\Sigma = \frac{N-1}{N}\Sigma + \frac{N-1}{N^2}\mathrm{dot}(m-\bar{m}, m-\bar{m})$$

[方差公式]

有时候在处理**流式数据**的时候，需要**实时更新数据**的统计值，如平均值和方差，如果通过传统求解方差或者平均值时，每到达一个新的数据就需要遍历来求解。在数据量比较少的时候，通过遍历和递推求解的时间消耗和空间消耗并不是很明显，但是在大数据或者流式数据的应用场景下，$O(n)$和$O(1)$的时间复杂度以及空间复杂度的区别还是很明显的。

均值公式：

$$A_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

均值递推公式：

$$A_n = A_{n-1} + \frac{(X_n - A_{n-1})}{n}$$

方差公式：

$$V_n = \frac{1}{n}\sum_{i=1}^{n}(X_i - A_n)$$

方差递推公式：

$$V_n = \frac{n-1}{n^2}(X_n - A_{n-1})^2 + \frac{n-1}{n}V_{n-1}$$

均值递推公式可以参考：https://blog.csdn.net/u014485485/article/details/77679669
方差递推公式可以参考：https://blog.csdn.net/wuqinlong/article/details/78432574

好像需要静止？陀螺均值当作零偏。

若不需初始化，则进行点云去畸变（传播也在此步骤中）

void ImuProcess::UndistortPcl(LidarMeasureGroup &lidar_meas, StatesGroup &state_inout, PointCloudXYZI &pcl_out)

is_lidar_end 来判断是 lidar 观测值还是图像观测值，每次对齐后

lidar_meas.measures 里仅有一类观测值（true: IMU+雷达，false: IMU+图像）

```
auto &&head = *(it_imu);
auto &&tail = *(it_imu + 1);
```

这属于万能引用，可接受左右值（能取地址的是左值，不能的是右值）

$$\left[\delta\boldsymbol{\theta}^T \quad {}^G\widetilde{\mathbf{p}}_I^T \quad {}^G\widetilde{\mathbf{v}}_I^T \quad \widetilde{\mathbf{b}}_{\boldsymbol{\omega}}^T \quad \widetilde{\mathbf{b}}_{\mathbf{a}}^T \quad {}^G\widetilde{\mathbf{g}}^T\right]^T$$

通过两个 IMU 帧得到平均线加速度和角速度

$$
\boldsymbol{F}=\begin{pmatrix}
-\boldsymbol{\omega}\times & & & -\boldsymbol{I} & & \\
 & & \boldsymbol{I} & & & \\
-{}^{G}\boldsymbol{R}_{I}\left(\boldsymbol{f}\times\right) & & & & -{}^{G}\boldsymbol{R}_{I} & \boldsymbol{I} \\
 & & & & & \\
 & & & & & \\
 & & & & &
\end{pmatrix}
\boldsymbol{\Phi}=\begin{pmatrix}
\boldsymbol{Exp}\left(-\boldsymbol{\omega}\Delta t\right) & & & -\boldsymbol{I}\Delta t & & \\
 & \boldsymbol{I} & \boldsymbol{I}\Delta t & & & \\
-{}^{G}\boldsymbol{R}_{I}\left(\boldsymbol{f}\times\right)\Delta t & & \boldsymbol{I} & & -{}^{G}\boldsymbol{R}_{I}\Delta t & \boldsymbol{I}\Delta t \\
 & & & \boldsymbol{I} & & \\
 & & & & \boldsymbol{I} & \\
 & & & & & \boldsymbol{I}
\end{pmatrix}
$$

$$
\boldsymbol{Q}=\begin{pmatrix}
\operatorname{diag}\left(\boldsymbol{ARW}\right)\Delta t^{2} & & & & & \\
 & & & & & \\
 & & {}^{G}\boldsymbol{R}_{I}\boldsymbol{VRW}\,{}^{G}\boldsymbol{R}_{I}^{T}\Delta t^{2} & & & \\
 & & & \operatorname{diag}\left(\boldsymbol{RW}_{bg}\right)\Delta t^{2} & & \\
 & & & & \operatorname{diag}\left(\boldsymbol{RW}_{ba}\right)\Delta t^{2} & \\
 & & & & &
\end{pmatrix}
$$

$$\boldsymbol{P}=\boldsymbol{\Phi}\boldsymbol{P}\boldsymbol{\Phi}^{T}+\boldsymbol{Q}$$

符号推导，导出其与 great 符号定义的关系，在 great 中姿态的扰动表达形式为

$$\dot{\boldsymbol{\alpha}}=-{}^{e}\boldsymbol{\omega}_{ie}\times\boldsymbol{\alpha}-{}^{e}\boldsymbol{R}_{b}\cdot\boldsymbol{b}_{g}$$

根据误差的左右扰动有，第一个表示 great 扰动 e 系（不一定是地球系，也表示 slam 的全局系），第二个表示扰动 b 系

$${}^{e}\boldsymbol{R}_{b}=\left(\boldsymbol{I}+\boldsymbol{\alpha}\times\right){}^{e}\hat{\boldsymbol{R}}_{b}={}^{e}\hat{\boldsymbol{R}}_{b}\left(\boldsymbol{I}-\boldsymbol{\theta}\times\right)$$

推导有

$$
\begin{aligned}
\boldsymbol{\alpha}&=-{}^{e}\boldsymbol{R}_{b}\boldsymbol{\theta}\\
{}^{e}\dot{\boldsymbol{R}}_{b}&=-\left({}^{e}\boldsymbol{\omega}_{be}\times\right){}^{e}\boldsymbol{R}_{b}={}^{e}\boldsymbol{R}_{b}\left({}^{b}\boldsymbol{\omega}_{eb}\times\right)\\
\dot{\boldsymbol{\alpha}}&=-{}^{e}\dot{\boldsymbol{R}}_{b}\boldsymbol{\theta}-{}^{e}\boldsymbol{R}_{b}\dot{\boldsymbol{\theta}}=\left({}^{e}\boldsymbol{\omega}_{be}\times\right){}^{e}\boldsymbol{R}_{b}\boldsymbol{\theta}-{}^{e}\boldsymbol{R}_{b}\dot{\boldsymbol{\theta}}\\
&=\left({}^{e}\boldsymbol{\omega}_{ie}\times\right){}^{e}\boldsymbol{R}_{b}\boldsymbol{\theta}-{}^{e}\boldsymbol{R}_{b}\cdot\boldsymbol{b}_{g}
\end{aligned}
$$

得到

$$\dot{\boldsymbol{\theta}}=-\left({}^{b}\boldsymbol{\omega}_{ib}\times\right)\boldsymbol{\theta}+\boldsymbol{b}_{g}$$

其与 fast-livo 中 F 矩阵的第一行一致，零偏猜测可能与 great 符号相反，补偿的时候需要用减号，后面验证。great 中速度的扰动表达形式为（考虑重力误差，此处可引入协方差导致重力可被优化）

$$
\begin{aligned}
{}^{e}\delta\dot{\boldsymbol{v}}&=\left({}^{e}\boldsymbol{R}_{b}\boldsymbol{f}\right)\times\boldsymbol{\alpha}-2\,{}^{e}\boldsymbol{\omega}_{ie}\times{}^{e}\delta\boldsymbol{v}+{}^{e}\boldsymbol{R}_{b}\boldsymbol{b}_{a}+{}^{e}\delta\boldsymbol{g}\\
&=-{}^{e}\boldsymbol{R}_{b}\left(\boldsymbol{f}\times\right)\boldsymbol{\theta}-2\,{}^{e}\boldsymbol{\omega}_{ie}\times{}^{e}\delta\boldsymbol{v}+{}^{e}\boldsymbol{R}_{b}\boldsymbol{b}_{a}+{}^{e}\delta\boldsymbol{g}
\end{aligned}
$$

这里全局系忽略地球自转，与 fast-livo 一致，注意加速度计零偏也和 great 反的！

```
cov_w.block<3,3>(0,0).diagonal()   = cov_gyr * dt * dt;
cov_w.block<3,3>(6,6)              = R_imu * cov_acc.asDiagonal() * R_imu.transpose() * dt * dt;
cov_w.block<3,3>(9,9).diagonal()   = cov_bias_gyr * dt * dt; // bias gyro covariance
cov_w.block<3,3>(12,12).diagonal() = cov_bias_acc * dt * dt; // bias acc covariance
```

谱密度传播这里感觉乘多了一个 dt，不过问题不大。

点云去畸变 fast-livo 写法和 r2live/r3live 一样

V3D T_ei(pos_imu + vel_imu * dt + 0.5 * acc_imu * dt * dt + R_i * Lid_offset_to_IMU - pos_liD_e);

V3D P_compensate = state_inout.rot_end.transpose() * (R_i * P_i + T_ei);

$$
\begin{aligned}
&{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{G}\boldsymbol{p}_{b_{meas}}+{}^{G}\boldsymbol{R}_{b_{meas}}{}^{b}\boldsymbol{p}_{L}-\left({}^{G}\boldsymbol{p}_{b_{scan-end}}+{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{b}\boldsymbol{p}_{L}\right)\right]\\
&={}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}-\left({}^{G}\boldsymbol{p}_{b_{scan-end}}+{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{b}\boldsymbol{p}_{L}\right)\right]\\
&={}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}-{}^{G}\boldsymbol{p}_{b_{scan-end}}\right]-{}^{b}\boldsymbol{p}_{L}
\end{aligned}
$$

而 fast-lio 的写法为

V3D P_i(it_pcl->x, it_pcl->y, it_pcl->z);

V3D T_ei(pos_imu + vel_imu * dt + 0.5 * acc_imu * dt * dt - imu_state.pos);

V3D P_compensate = imu_state.offset_R_L_I.conjugate() * (imu_state.rot.conjugate() * (R_i * (imu_state.offset_R_L_I * P_i + imu_state.offset_T_L_I) + T_ei) - imu_state.offset_T_L_I);// not accurate!

$$
{}^{b}\boldsymbol{R}_{L}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left({}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{b}\boldsymbol{R}_{L}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}\right)-{}^{b}\boldsymbol{p}_{L}\right]
$$

前一种没考虑外参的旋转，后一种不完整（缺了 meas 到 scan-end 的平移），个

人认为此处应该修改为：

$$
\begin{aligned}
&{}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{b}\boldsymbol{R}_{L}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}={}^{G}\boldsymbol{R}_{b_{scan-end}}\left({}^{b}\boldsymbol{R}_{L}{}^{L_{scan-end}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{scan-end}}\\
&{}^{L_{scan-end}}\boldsymbol{p}_{f}={}^{b}\boldsymbol{R}_{L}{}^{T}\left\{{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{b}\boldsymbol{R}_{L}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}-{}^{G}\boldsymbol{p}_{b_{scan-end}}\right]-{}^{b}\boldsymbol{p}_{L}\right\}\\
&={}^{b}\boldsymbol{R}_{L}{}^{T}\left\{{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}\left({}^{b}\boldsymbol{R}_{L}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{b}\boldsymbol{p}_{L}\right)+{}^{G}\boldsymbol{p}_{b_{meas}}-\left({}^{G}\boldsymbol{p}_{b_{scan-end}}+{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{b}\boldsymbol{p}_{L}\right)\right]\right\}\\
&={}^{b}\boldsymbol{R}_{L}{}^{T}\left\{{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{T}\left[{}^{G}\boldsymbol{R}_{b_{meas}}{}^{b}\boldsymbol{R}_{L}{}^{L_{meas}}\boldsymbol{p}_{f}+{}^{G}\boldsymbol{p}_{b_{meas}}+{}^{G}\boldsymbol{R}_{b_{meas}}{}^{b}\boldsymbol{p}_{L}-\left({}^{G}\boldsymbol{p}_{b_{scan-end}}+{}^{G}\boldsymbol{R}_{b_{scan-end}}{}^{b}\boldsymbol{p}_{L}\right)\right]\right\}
\end{aligned}
$$

feats_undistort 为 lidar 帧中矫正畸变后的点云

点云的上色问题，fast-lio2

3.4 PCD file save

Set `pcd_save_enable` in launchfile to `1`. All the scans (in global frame) will be accumulated and saved to the file `FAST_LIO/PCD/scans.pcd` after the FAST-LIO is terminated. `pcl_viewer scans.pcd` can visualize the point clouds.

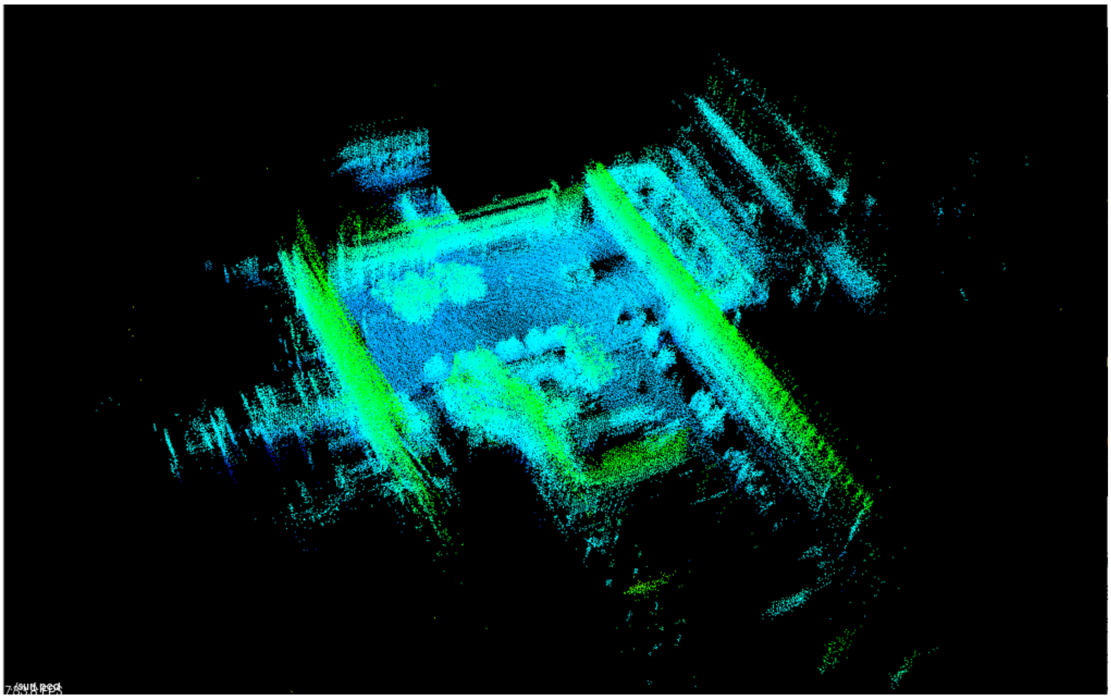
Tips for pcl_viewer:

- change what to visualize/color by pressing keyboard 1,2,3,4,5 when pcl_viewer is running.

```
1 is all random
2 is X values
3 is Y values
4 is Z values
5 is intensity
```

terminal: pcl_viewer *.pcd

然后按键 1 2 3 4 5



//判断程序是否准备完成，需要有点云，然后滤波器初始化完成要满足时间限制

```
if (feats_undistort->empty() || (feats_undistort == nullptr))
{
    // cout<<" No point!!!"<<endl;
    if (!fast_lio_is_ready)
    {
        first_lidar_time = LidarMeasures.lidar_beg_time;
        p_imu->first_lidar_time = first_lidar_time;
        LidarMeasures.measures.clear();
        cout<<"FAST-LIO not ready"<<endl;
        continue;      xuankuzcr, 3个月前 • [Release] release source code & dataset &
    }
}
else
{
    int size = feats_undistort->points.size();
}
fast_lio_is_ready = true;
flg_EKF_inited = (LidarMeasures.lidar_beg_time - first_lidar_time) < INIT_TIME ? \
                        false : true;
```

//处理 vio 子系统,first_lidar_time 是非常大的,此处应该是判断是否处理过 lidar

```
if (first_lidar_time<10)
{
    continue;
}
```

//

void LidarSelector::detect(cv::Mat img, PointCloudXYZI::Ptr pg)

这里很奇怪，参数只传递了相机内参，并没传递图像的长宽信息，而是按照默认值 800，600 的进行图像 resize

```
if(stage_ == STAGE_FIRST_FRAME && pg->size()>10)
{
    new_frame_->setKeyframe();
    stage_ = STAGE_DEFAULT_FRAME;
}
```

如果是首帧而且点云足够，就设为关键帧

```
void Frame::setKeyframe()
{
    is_keyframe_ = true;
    setKeyPoints();
}
```

关键帧中设置关键点，五个特征和关联的 3D 点，用于检测两个帧是否具有重叠的视野。

vector<FeaturePtr> key_pts_; //!< Five features and associated 3D points which are used to detect if two frames have overlapping field of view.

```
void Frame::setKeyPoints()
{
  for(size_t i = 0; i < 5; ++i)
    if(key_pts_[i] != nullptr)
      if(key_pts_[i]->point == nullptr)
        key_pts_[i] = nullptr;
  std::for_each(fts_.begin(), fts_.end(), [&](FeaturePtr ftr){ if(ftr->point != nullptr) checkKeyPoints(ftr); });
}
```

这其中用到了 LAMBDA 表达式和泛型算法 for_each

[&]	函数局部作用域里的所有变量都按引用捕获

相当于对 fts 里的每一个元素做 checkKeyPoints 操作，目的是找到最中间和 4 方最边缘的点，然后进入

void LidarSelector::addFromSparseMap(cv::Mat img, PointCloudXYZI::Ptr pg)

这一步的主要目的是从特征地图选择出视觉子地图

//特征体素地图的数据格式为

unordered_map<VOXEL_KEY, VOXEL_POINTS*> feat_map;

pcl_wait_pub 为全局帧的点云

将降采样后特征地图的点投影到像素坐标系，保留该像素对应的深度（负深度点被丢弃），同时去除一些图像边缘上的点，将对应子特征体素地图占位符设置为 1.0。然后对子地图的每个特征，查找特征地图中的对应体素的所有特征点，去畸变投影到像素坐标系，寻找到该体素的代表特征放入存储 map_dist 和 voxel_points_。在一个 patch(40×40 像素)内，轮询雷达得到的深度和上一步体素代表特征的深度（必须都存在），若不超限（1.5m）则可用。

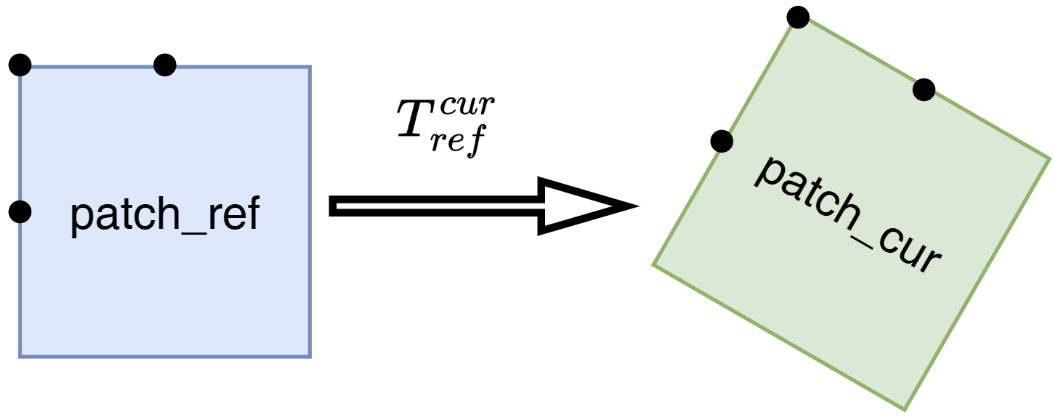
//然后找到体素代表特征视角最接近的帧

if(!pt->getCloseViewObs(new_frame_->pos(), ref_ftr, pc)) continue;

//计算矩阵 Ai

$$\mathbf{0} = \mathbf{r}_c(\mathbf{x}_k, {}^G\mathbf{p}_i) = \mathbf{I}_k(\boldsymbol{\pi}({}^I\mathbf{T}_C^{-1}\, {}^G\mathbf{T}_{I_k}^{-1}\, {}^G\mathbf{p}_i)) - \mathbf{A}_i \mathbf{Q}_i$$

getWarpMatrixAffine(*cam, ref_ftr->px, ref_ftr->f, (ref_ftr->pos() - pt->pos_).norm(),
new_frame_->T_f_w_ * ref_ftr->T_f_w_.inverse(), 0, 0, patch_size_half, A_cur_ref_zero);



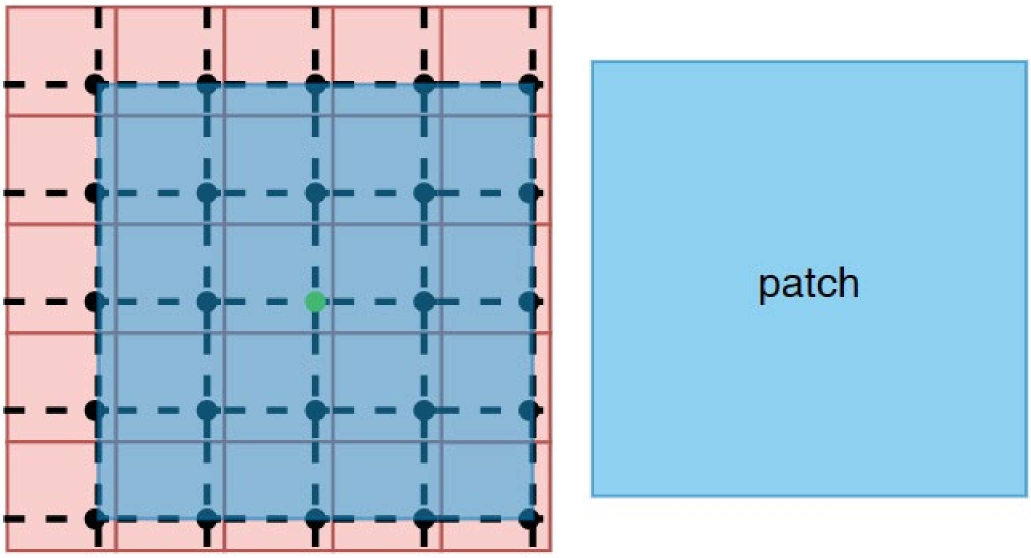
//从参考图像生成多层图像包络

```
for(int pyramid_level=0; pyramid_level<=0; pyramid_level++)
{
    warpAffine(A_cur_ref_zero, ref_ftr->img, ref_ftr->px, ref_ftr->level, search_level, pyramid_level, patch_size_half, patch_wrap);
}
```

search_level 不变，pyramid_level 改变

//在 0 层对当前图像做整数像素到浮点像素的插值得到 patch

getpatch(img, pc, patch_cache, 0);



//然后进入相关系数验证 NCC，若误差不超限（ncc_thre 和 outlier_threshold*patch_size*patchsize）,在当前帧特征 sub_map_cur_frame_加入该体素点,然后在子稀疏地图 sub_sparse_map 中添加该特征、预测的 patch_wrap。

到此，addFromSparseMap 函数结束，回到 detect 主流程，再进入 addSparseMap 函数，这个函数的目的是往 addFromSparseMap 函数中的 feat_map 中添加特征

```cpp
for (int i=0; i<pg->size(); i++)
{
    V3D pt(pg->points[i].x, pg->points[i].y, pg->points[i].z);
    V2D pc(new_frame_->w2c(pt));
    if(new_frame_->cam_->isInFrame(pc.cast<int>(), (patch_size_half+1)*8)) // 20px is the patch size in the matcher
    {
        int index = static_cast<int>(pc[0]/grid_size)*grid_n_height + static_cast<int>(pc[1]/grid_size);
        // float cur_value = CheckGoodPoints(img, pc);
        float cur_value = vk::shiTomasiScore(img, pc[0], pc[1]);

        if (cur_value > map_value[index]) //&& (grid_num[index] != TYPE_MAP || map_value[index]<=10)) //! only add in not occupied grid
        {
            map_value[index] = cur_value;
            add_voxel_points_[index] = pt;
            grid_num[index] = TYPE_POINTCLOUD;
        }
    }
}
```

先判断点云中有没有比之前体素中已经存在的点云更像角点的，然后对这些非常角点的点，计算 3 层的 patch，然后直接增加地图点，没什么质量控制。

//对来自 addFromSparseMap 的点，计算雅各比

```cpp
void LidarSelector::ComputeJ(cv::Mat img)
{
    int total_points = sub_sparse_map->index.size();
    if (total_points==0) return;
    float error = 1e10;
    float now_error = error;

    for (int level=2; level>=0; level--)
    {
        now_error = UpdateState(img, error, level);
    }
    if (now_error < error)
    {
        state->cov -= G*state->cov;
    }
    updateFrameState(*state);
}
```

level=0 最精细，level=2 最粗糙

$$\frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f}=\begin{bmatrix} \frac{f_x}{{}^{C}Z_f} & 0 & -\frac{f_x \cdot {}^{C}X_f}{\left({}^{C}Z_f\right)^2} \\ 0 & \frac{f_y}{{}^{C}Z_f} & -\frac{f_y \cdot {}^{C}Y_f}{\left({}^{C}Z_f\right)^2} \end{bmatrix}=\frac{1}{{}^{C}Z_f}\begin{bmatrix} f_x & 0 & -\frac{f_x \cdot {}^{C}X_f}{{}^{C}Z_f} \\ 0 & f_y & -\frac{f_y \cdot {}^{C}Y_f}{{}^{C}Z_f} \end{bmatrix}=\begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}\frac{1}{{}^{C}Z_f}\begin{bmatrix} 1 & 0 & -\frac{{}^{C}X_f}{{}^{C}Z_f} \\ 0 & 1 & -\frac{{}^{C}Y_f}{{}^{C}Z_f} \end{bmatrix}$$

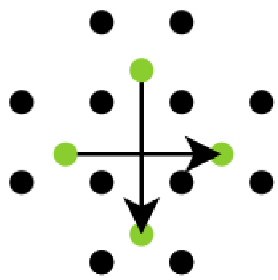
$$=\frac{\partial \boldsymbol{h}_d(\cdot)}{\partial \boldsymbol{z}_{n,k}}\frac{\partial \boldsymbol{h}_p(\cdot)}{\partial {}^{C}\boldsymbol{p}_f}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} := \mathbf{z}_k = \mathbf{h}_d(\mathbf{z}_{n,k},\ \zeta) = \begin{bmatrix} f_x * x + c_x \\ f_y * y + c_y \end{bmatrix}$$

$$\mathbf{z}_{n,k} = \mathbf{h}_p({}^{C_k}\mathbf{p}_f) = \begin{bmatrix} {}^C x / {}^C z \\ {}^C y / {}^C z \end{bmatrix}$$

$$\text{where} \quad {}^{C_k}\mathbf{p}_f = \begin{bmatrix} {}^C x \\ {}^C y \\ {}^C z \end{bmatrix}$$

```
float du = 0.5f * ((w_ref_tl*img_ptr[scale] + w_ref_tr*img_ptr[scale*2] + w_ref_bl*img_ptr[scale*width+scale] + w_ref_br*img_ptr[scale*width+scale*2])
            -(w_ref_tl*img_ptr[-scale] + w_ref_tr*img_ptr[0] + w_ref_bl*img_ptr[scale*width-scale] + w_ref_br*img_ptr[scale*width]));
float dv = 0.5f * ((w_ref_tl*img_ptr[scale*width] + w_ref_tr*img_ptr[scale+scale*width] + w_ref_bl*img_ptr[width*scale*2] + w_ref_br*img_ptr[width*scale*2+scale])
            -(w_ref_tl*img_ptr[-scale*width] + w_ref_tr*img_ptr[-scale*width+scale] + w_ref_bl*img_ptr[0] + w_ref_br*img_ptr[scale]));
```



$$\mathbf{0} = \mathbf{r}_c(\mathbf{x}_k, {}^G\mathbf{p}_i) = \mathbf{I}_k(\boldsymbol{\pi}({}^I\mathbf{T}_C^{-1}\, {}^G\mathbf{T}_{I_k}^{-1}\, {}^G\mathbf{p}_i)) - \mathbf{A}_i\mathbf{Q}_i$$

$$r_C = I\left(\pi\left({}^C R_G\left({}^G p_f - {}^G p_C\right)\right)\right) - AQ$$

```
Jimg << du, dv;
Jimg = Jimg * (1.0/scale);
Jdphi = Jimg * Jdpi * p_hat;
Jdp = -Jimg * Jdpi;
JdR = Jdphi * Jdphi_dR + Jdp * Jdp_dR;
Jdt = Jdp * Jdp_dt;
```

$$
\begin{aligned}
{}^{C}\boldsymbol{p}_f &= {}^{G}\boldsymbol{R}_C^{-1}\left({}^{G}\boldsymbol{p}_f - {}^{G}\boldsymbol{p}_C\right) \\
\frac{\partial I}{\partial \boldsymbol{\pi}} &= \begin{bmatrix} e_{du} & e_{dv} \end{bmatrix}^T \\
\frac{\partial I}{\partial {}^{G}\boldsymbol{R}_C} &= \frac{\partial I}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f} \frac{\partial {}^{C}\boldsymbol{p}_f}{\partial {}^{G}\boldsymbol{R}_C} = \frac{\partial I}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f} \cdot \left[-{}^{G}\boldsymbol{R}_C^{-1}\left({}^{G}\boldsymbol{p}_f - {}^{G}\boldsymbol{p}_C\right)\times\right] \\
&= \frac{\partial I}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f} \cdot \left(-{}^{C}\boldsymbol{p}_f \times\right) \\
\frac{\partial I}{\partial {}^{G}\boldsymbol{p}_C} &= \frac{\partial I}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f} \frac{\partial {}^{C}\boldsymbol{p}_f}{\partial {}^{G}\boldsymbol{p}_C} \\
&\quad \frac{\partial I}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial {}^{C}\boldsymbol{p}_f} \cdot {}^{G}\boldsymbol{R}_C^{-1}
\end{aligned}
$$

//dphi 是相机相对全局系旋转的扰动，dR 是惯导相对全局系旋转的扰动

Jdphi_dR = Rci;

$$
\begin{aligned}
{}^{G}\boldsymbol{R}_C &= {}^{G}\hat{\boldsymbol{R}}_C\left(\boldsymbol{I} - \boldsymbol{\theta}_C \times\right) \\
{}^{G}\boldsymbol{R}_I &= {}^{G}\hat{\boldsymbol{R}}_I\left(\boldsymbol{I} - \boldsymbol{\theta}_I \times\right) \\
\frac{\partial {}^{G}\boldsymbol{R}_C}{\partial {}^{G}\boldsymbol{R}_I} &= {}^{C}\boldsymbol{R}_I
\end{aligned}
$$

tmp << SKEW_SYM_MATRX(Pic);
Jdp_dR = -Rci * tmp;
Jdp_dt = Rci * Rwi.transpose();

$$
\begin{aligned}
{}^{G}\boldsymbol{p}_C &= {}^{G}\boldsymbol{p}_I + {}^{G}\boldsymbol{R}_I\, {}^{I}\boldsymbol{p}_C \\
\frac{\partial {}^{G}\boldsymbol{p}_C}{\partial {}^{G}\boldsymbol{R}_I} &= -{}^{G}\boldsymbol{R}_I\left({}^{I}\boldsymbol{p}_C \times\right) \\
\frac{\partial {}^{G}\boldsymbol{p}_C}{\partial {}^{G}\boldsymbol{p}_I} &= \boldsymbol{I}
\end{aligned}
$$

公式为我自行推导，虽然与代码中间计算步骤不同，但最终归结到 IMU 位姿的

雅各比都能对上，均只差负号（移项）。

```
if (error <= last_error)
{
    old_state = (*state);
    last_error = error;

    // K = (H.transpose() / img_point_cov * H + state->cov.inverse()).inverse() * H.transpose() / img_point_cov;
    // auto vec = (*state_propagat) - (*state);
    // G = K*H;
    // (*state) += (-K*z + vec - G*vec);

    auto &&H_sub_T = H_sub.transpose();
    H_T_H.block<6,6>(0,0) = H_sub_T * H_sub;
    MD(DIM_STATE, DIM_STATE) &&K_1 = (H_T_H + (state->cov / img_point_cov).inverse()).inverse();
    auto &&HTz = H_sub_T * z;
    // K = K_1.block<DIM_STATE,6>(0,0) * H_sub_T;
    auto vec = (*state_propagat) - (*state);
    G.block<DIM_STATE,6>(0,0) = K_1.block<DIM_STATE,6>(0,0) * H_T_H.block<6,6>(0,0);
    auto solution = - K_1.block<DIM_STATE,6>(0,0) * HTz + vec - G.block<DIM_STATE,6>(0,0) * vec.block<6,1>(0,0);
    (*state) += solution;
    auto &&rot_add = solution.block<3,1>(0,0);
    auto &&t_add   = solution.block<3,1>(3,0);

    if ((rot_add.norm() * 57.3f < 0.001f) && (t_add.norm() * 100.0f < 0.001f))
    {
        EKF_end = true;
    }
}
else
{
    (*state) = old_state;
    EKF_end = true;
}    xuankuzcr, 3个月前 • [Release] release source code & dataset & hardwar…
```

采用迭代更新的策略，当迭代到达次数，或位姿改正量很小，或误差不再下降，退出迭代更新。

auto solution = - K_1.block<DIM_STATE,6>(0,0) * HTz + vec - G.block<DIM_STATE,6>(0,0) * vec.block<6,1>(0,0);

这里的 solution 增益项是加了个负号算的，然后补偿用+号，相当于 solution 增益项不加负号，补偿用减号。所以前面自行推导的公式仅仅是没有移项，所以差负号。

关于迭代卡尔曼滤波量测更新（测量方差阵对角时约等于成立）

$$
\begin{aligned}
\boldsymbol{K} &= \left(\boldsymbol{H}^T\boldsymbol{R}^{-1}\boldsymbol{H}+\boldsymbol{P}^{-1}\right)^{-1}\boldsymbol{H}^T\boldsymbol{R}^{-1} \\
&\approx \left(\boldsymbol{H}^T\boldsymbol{H}+\left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^T \\
\boldsymbol{X} &= \boldsymbol{X}+\boldsymbol{K}\left(\boldsymbol{Z}-\boldsymbol{H}\boldsymbol{X}\right) \\
&= \boldsymbol{X}+\left(\boldsymbol{H}^T\boldsymbol{H}+\left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^T\boldsymbol{Z}-\left(\boldsymbol{H}^T\boldsymbol{H}+\left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^T\boldsymbol{H}\boldsymbol{X}
\end{aligned}
$$

所以 solution 中非增益项的尾巴，来自迭代过程！

```
if (now_error < error)
{
    state->cov -= G*state->cov;
}
```

方差的更新过程，但是迭代中只更新状态，不更新方差，方差只最后更新一次。

//往观测到点的帧里面添加当前帧

void LidarSelector::addObservation(cv::Mat img)

$$
\begin{aligned}
\operatorname{tr}(\boldsymbol{R}) &= \cos\theta \operatorname{tr}(\boldsymbol{I}) + (1-\cos\theta)\operatorname{tr}\left(\boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\right) + \sin\theta \operatorname{tr}(\boldsymbol{n}^{\wedge}) \\
&= 3\cos\theta + (1-\cos\theta) \\
&= 1 + 2\cos\theta.
\end{aligned}
\tag{3.16}
$$

```
double delta_p = delta_pose.translation().norm();        xuankuzcr, 3个月前 • [Release] release source code & dataset & hardwar…
double delta_theta = (delta_pose.rotation_matrix().trace() > 3.0 - 1e-6) ? 0.0 : std::acos(0.5 * (delta_pose.rotation_matrix().trace() - 1));
if(delta_p > 0.5 || delta_theta > 10) add_flag = true;
```

判断旋转和平移的大小，个人认为这里关于旋转的判断有问题，应该是 10 度，

原作者回应，设置为 0.3

```
// Step 3: pixel distance
Vector2d last_px = last_feature->px;
double pixel_dist = (pc-last_px).norm();
if(pixel_dist > 40) add_flag = true;
```

判断像素距离，也可以改变添加的 flag

```
// Maintain the size of 3D Point observation features.
if(pt->obs_.size()>=20)
{
    FeaturePtr ref_ftr;
    pt->getFurthestViewObs(new_frame_->pos(), ref_ftr);
    pt->deleteFeatureRef(ref_ftr);
    // ROS_WARN("ref_ftr->id_ is %d", ref_ftr->id_);
}
```

帧不能无限多，慢的时候丢掉最远的帧

```
void LidarSelector::display_keypatch(double time)
{
    int total_points = sub_sparse_map->index.size();
    if (total_points==0) return;
    for(int i=0; i<total_points; i++)
    {
        PointPtr pt = sub_sparse_map->voxel_points[i];
        V2D pc(new_frame_->w2c(pt->pos_));
        cv::Point2f pf;
        pf = cv::Point2f(pc[0], pc[1]);
        if (sub_sparse_map->errors[i]<8000) // 5.5
            cv::circle(img_cp, pf, 4, cv::Scalar(0, 255, 0), -1, 8); // Green Sparse Align tracked
        else
            cv::circle(img_cp, pf, 4, cv::Scalar(255, 0, 0), -1, 8); // Blue Sparse Align tracked
    }
    std::string text = std::to_string(int(1/time))+" HZ";
    cv::Point2f origin;
    origin.x = 20;
    origin.y = 20;
    cv::putText(img_cp, text, origin, cv::FONT_HERSHEY_COMPLEX, 0.6, cv::Scalar(255, 255, 255), 1, 8, 0);
}
```

这个函数的效果暂时没看见（需要关闭硬件加速），至此，detect 函数结束。

然后当前帧特征添加到 sub_map_cur_frame_point，发布上色的点云，暂时也没看到（需要关闭硬件加速）。

```
export LIBGL_ALWAYS_SOFTWARE=1
```

[rviz/Troubleshooting - ROS Wiki]

lasermap_fov_segment();

//这个函数大概意思就是将当前位置（IMU）作为局部地图的中心，删除过远 box 中的点，涉及到很多 ikd-Tree 的知识，后面再补

```
if(ikdtree.Root_Node == nullptr)        xuankuzcr, 3个月前 • [Re
{
    if(feats_down_body->points.size() > 5)
    {
        ikdtree.set_downsample_param(filter_size_map_min);
        ikdtree.Build(feats_down_body->points);
    }
    continue;
}
```

ikd 树在此处被初始化，还支持下采样，我认为这里初始化应该遵循增量地图的全局点云，而 feats_down_body 来自 feats_undistort，是 lidar 帧的局部点云，我认为此处应改为如下，注意先必须要 resize 才能坐标转换!!

```
if(ikdtree.Root_Node == nullptr)
{
    if(feats_down_body->points.size() > 5)
    {
        feats_down_size = feats_down_body->points.size();
        feats_down_world->resize(feats_down_size);
        for (int it_down = 0; it_down < feats_down_body->points.size(); it_down++)
        {
            /* transform to world frame */
            pointBodyToWorld(&(feats_down_body->points[it_down]), &(feats_down_world->points[it_down]));
        }

        ikdtree.set_downsample_param(filter_size_map_min);
        // ikdtree.Build(feats_down_body->points);
        ikdtree.Build(feats_down_world->points);
    }
    continue;
}
```

然后进入雷达测量值的处理

//将指定 Node（即 kdtree 结构中的节点）下的点云另存为线性化排列的点云;仅在需要可视化 ikdtree 地图时，在算法循环中被调用。

ikdtree.flatten(ikdtree.Root_Node, ikdtree.PCL_Storage,

NOT_RECORD);

看了一点 ikd Tree 构建的知识

```
// Select the longest dimension as division axis
float min_value[3] = {INFINITY, INFINITY, INFINITY};
float max_value[3] = {-INFINITY, -INFINITY, -INFINITY};
float dim_range[3] = {0,0,0};
for (i=l;i<=r;i++){
    min_value[0] = min(min_value[0], Storage[i].x);
    min_value[1] = min(min_value[1], Storage[i].y);
    min_value[2] = min(min_value[2], Storage[i].z);
    max_value[0] = max(max_value[0], Storage[i].x);
    max_value[1] = max(max_value[1], Storage[i].y);
    max_value[2] = max(max_value[2], Storage[i].z);
}
for (i=0;i<3;i++) dim_range[i] = max_value[i] - min_value[i];
for (i=1;i<3;i++) if (dim_range[i] > dim_range[div_axis]) div_axis = i;
// Divide by the division axis and recursively build.

(*root)->division_axis = div_axis;
```

选择最长的维度作为区分轴

```
switch (div_axis)
{
case 0:
    nth_element(begin(Storage)+l, begin(Storage)+mid, begin(Storage)+r+1, point_cmp_x);
    break;
case 1:
    nth_element(begin(Storage)+l, begin(Storage)+mid, begin(Storage)+r+1, point_cmp_y);
    break;
case 2:
    nth_element(begin(Storage)+l, begin(Storage)+mid, begin(Storage)+r+1, point_cmp_z);
    break;
default:
    nth_element(begin(Storage)+l, begin(Storage)+mid, begin(Storage)+r+1, point_cmp_x);
    break;
}
(*root)->point = Storage[mid];
```

根据轴找到对应第 mid 小的元素，作为当前节点的 point，然后再分为左右子树递归构建，再回到 flatten，其就是将指定 Node（即 kdtree 结构中的节点）下的点云另存为线性化排列的点云。

//对每个点寻找最近的 K 个点，此处为 5，也就是最近的平面

```
/** Find the closest surfaces in the map **/
#ifdef USE_ikdtree
    #ifdef USE_ikdforest
        search_flag = ikdforest.Nearest_Search(point_world, NUM_MATCH_POINTS, points_near, pointSearchSqDis, first_lidar_time, 5);
    #else
        ikdtree.Nearest_Search(point_world, NUM_MATCH_POINTS, points_near, pointSearchSqDis);
    #endif
#else
    kdtreeSurfFromMap->nearestKSearch(point_world, NUM_MATCH_POINTS, points_near, pointSearchSqDis);
#endif
```

6) Nearest_Search

```
void KD_TREE<PointType>::Nearest_Search(PointType point, int k_nearest,
PointVector& Nearest_Points, vector<float> & Point_Distance, double max_dist)
```

Description: Search k nearest neighbors of the target point on the ikd-Tree.

point: The target point to find nearest neighbors of.

k_nearest: The number of nearest neighbors to search.

Nearest_Points: Return the nearest neighbor points of the target point.

Point_Distance: Return the distance from the nearest neighbor points to the target point (squared distance, Unit: m^2).

max_dist: The range limitation to find nearest neighbor (Unit: meter).

if (esti_plane(pabcd, points_near, 0.1f)) //(planeValid)

进行平面的估计

```
template<typename T>
bool esti_plane(Matrix<T, 4, 1> &pca_result, const PointVector &point, const T &threshold)
{
    Matrix<T, NUM_MATCH_POINTS, 3> A;
    Matrix<T, NUM_MATCH_POINTS, 1> b;
    b.setOnes();
    b *= -1.0f;

    for (int j = 0; j < NUM_MATCH_POINTS; j++)
    {
        A(j,0) = point[j].x;
        A(j,1) = point[j].y;
        A(j,2) = point[j].z;
    }
      xuankuzcr, 3个月前 • [Release] release source code & dataset & hardwar…
    Matrix<T, 3, 1> normvec = A.colPivHouseholderQr().solve(b);

    T n = normvec.norm();
    pca_result(0) = normvec(0) / n;
    pca_result(1) = normvec(1) / n;
    pca_result(2) = normvec(2) / n;
    pca_result(3) = 1.0 / n;

    for (int j = 0; j < NUM_MATCH_POINTS; j++)
    {
        if (fabs(pca_result(0) * point[j].x + pca_result(1) * point[j].y + pca_result(2) * point[j].z + pca_result(3)) > threshold)
        {
            return false;
        }
    }
```

$$Ax + By + Cz + 1 = 0$$

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_5 & y_5 & z_5 \end{pmatrix}\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} -1 \\ \vdots \\ -1 \end{pmatrix}$$

colPivHouseholderQr 就是求解 Ax=b 的 x，这里相当于 5 个点列 5 个直线方程，求解。然后单位化，直接求取这 5 点到该平面的距离，如果不超限制则内符合，平面可估，此处阈值为 0.1。然后利用 lidar 降采样的特征判断

float s = 1 - 0.9 * fabs(pd2) / sqrt(p_body.norm());

这个式子没太看懂，应该来自于 LOAM，当 pd2（特征点到平面的距离）很小或 p_body.norm()（特征点离 lidar 的距离）很大时，s 接近 1；反之 s 会很小。可以理解为对平面的置信度，远处的平面对位姿求解起主要作用，这也符合我们的常规认知，远处的平面对位姿约束会好一些。后续的判断挑选 s>0.9 的

```
if (s > 0.9)
{
    point_selected_surf[i] = true;
    normvec->points[i].x = pabcd(0);
    normvec->points[i].y = pabcd(1);
    normvec->points[i].z = pabcd(2);
    normvec->points[i].intensity = pd2;
    res_last[i] = abs(pd2);
}
```

所以 s 越大越好，说明需要选出离 lidar 远的、拟合的又很好的平面。

```
point_this += Lidar_offset_to_IMU;
```

我认为这句有问题，应该是求特征在 IMU 帧中的位置。

考虑改写为（Lid_rot_to_IMU 变量需要自己添加）：

```
point_this = Lid_rot_to_IMU * point_this + Lidar_offset_to_IMU;
```

但这样写好像有点问题，不知道为什么，输出这步结果没问题，但是最终结果有问题，可能是 M3D 和 V3D 的乘法有点问题，考虑 MatrixXd 接受结果，再赋值给 V3D，暂且这样吧。

```
MatrixXd point_this_m=Lid_rot_to_IMU * point_this;
point_this(0)=point_this_m(0,0);
point_this(1)=point_this_m(1,0);
point_this(2)=point_this_m(2,0);
point_this += Lidar_offset_to_IMU;
```

雅可比推导

$$
\begin{aligned}
&{}^{G}\boldsymbol{u}_{plane}{}^{T} \cdot {}^{G}\boldsymbol{p}_{f} + D = 0 \\
&{}^{G}\boldsymbol{p}_{f} = {}^{G}\boldsymbol{R}_{I}\,{}^{I}\boldsymbol{p}_{f} + {}^{G}\boldsymbol{p}_{I} \\
&{}^{G}\boldsymbol{R}_{I} = {}^{G}\hat{\boldsymbol{R}}_{I}\,{}^{\hat{I}}\boldsymbol{R}_{I} = {}^{G}\hat{\boldsymbol{R}}_{I}\left(\boldsymbol{I} - \boldsymbol{\theta}_{I}\times\right) \\
&{}^{G}\boldsymbol{p}_{I} = {}^{G}\hat{\boldsymbol{p}}_{I} - \delta\,{}^{G}\boldsymbol{p}_{I}
\end{aligned}
$$

$$
\begin{aligned}
{}^{G}\boldsymbol{p}_{f} &= {}^{G}\hat{\boldsymbol{R}}_{I}\left(\boldsymbol{I}-\boldsymbol{\theta}_{I}\times\right){}^{I}\boldsymbol{p}_{f} + {}^{G}\hat{\boldsymbol{p}}_{I} - \delta\,{}^{G}\boldsymbol{p}_{I} \\
&= {}^{G}\hat{\boldsymbol{R}}_{I}\,{}^{I}\boldsymbol{p}_{f} + {}^{G}\hat{\boldsymbol{R}}_{I}\left({}^{I}\boldsymbol{p}_{f}\times\right)\boldsymbol{\theta}_{I} + {}^{G}\hat{\boldsymbol{p}}_{I} - \delta\,{}^{G}\boldsymbol{p}_{I} \\
&= {}^{G}\hat{\boldsymbol{p}}_{f} + {}^{G}\hat{\boldsymbol{R}}_{I}\left({}^{I}\boldsymbol{p}_{f}\times\right)\boldsymbol{\theta}_{I} - \delta\,{}^{G}\boldsymbol{p}_{I}
\end{aligned}
$$

$$
\begin{aligned}
&{}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot{}^{G}\boldsymbol{p}_{f} + D = 0 \\
&{}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot\left({}^{G}\hat{\boldsymbol{p}}_{f} + {}^{G}\hat{\boldsymbol{R}}_{I}\left({}^{I}\boldsymbol{p}_{f}\times\right)\boldsymbol{\theta}_{I} - \delta\,{}^{G}\boldsymbol{p}_{I}\right) = 0 \\
&{}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot{}^{G}\hat{\boldsymbol{R}}_{I}\left({}^{I}\boldsymbol{p}_{f}\times\right)\boldsymbol{\theta}_{I} - {}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot\delta\,{}^{G}\boldsymbol{p}_{I} = -\left({}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot{}^{G}\hat{\boldsymbol{p}}_{f} + D\right) \\
&\left[-\left({}^{I}\boldsymbol{p}_{f}\times\right)\cdot{}^{G}\hat{\boldsymbol{R}}_{I}{}^{T}\cdot{}^{G}\boldsymbol{u}_{plane}\right]^{T}\cdot\boldsymbol{\theta}_{I} - {}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot\delta\,{}^{G}\boldsymbol{p}_{I} = -\left({}^{G}\boldsymbol{u}_{plane}{}^{T}\cdot{}^{G}\hat{\boldsymbol{p}}_{f} + D\right)
\end{aligned}
$$

```
/*** calculate the Measuremnt Jacobian matrix H ***/
V3D A(point_crossmat * state.rot_end.transpose() * norm_vec);
Hsub.row(i) << VEC_FROM_ARRAY(A), norm_p.x, norm_p.y, norm_p.z;

/*** Measuremnt: distance to the closest surface/corner ***/
meas_vec(i) = - norm_p.intensity;
```

我的推导和代码的设计矩阵相差一个负号

```
auto &&Hsub_T = Hsub.transpose();
auto &&HTz = Hsub_T * meas_vec;
H_T_H.block<6,6>(0,0) = Hsub_T * Hsub;
// EigenSolver<Matrix<double, 6, 6>> es(H_T_H.block<6,6>(0,0));
MD(DIM_STATE, DIM_STATE) &&K_1 = \
        (H_T_H + (state.cov / LASER_POINT_COV).inverse()).inverse();
G.block<DIM_STATE,6>(0,0) = K_1.block<DIM_STATE,6>(0,0) * H_T_H.block<6,6>(0,0);
auto vec = state_propagat - state;      xuankuzcr, 3个月前 • [Release] release source code & dataset & 
solution = K_1.block<DIM_STATE,6>(0,0) * HTz + vec - G.block<DIM_STATE,6>(0,0) * vec.block<6,1>(0,0);

int minRow, minCol;
if(0)//if(V.minCoeff(&minRow, &minCol) < 1.0f)
{
    VD(6) V = H_T_H.block<6,6>(0,0).eigenvalues().real();
    cout<<"!!!!!! Degeneration Happend, eigen values: "<<V.transpose()<<endl;
    EKF_stop_flg = true;
    solution.block<6,1>(9,0).setZero();
}

state += solution;
```

注意此处 solution 增益项的符号，与视觉不同，这里没有负号，而补偿用加号。因此，我的公式推导对应于 great 中补偿用减号的情形。

关于迭代卡尔曼滤波量测更新（测量方差阵对角时约等于成立）

$$
\begin{aligned}
\boldsymbol{K} &= \left(\boldsymbol{H}^{T}\boldsymbol{R}^{-1}\boldsymbol{H} + \boldsymbol{P}^{-1}\right)^{-1}\boldsymbol{H}^{T}\boldsymbol{R}^{-1} \\
&\approx \left(\boldsymbol{H}^{T}\boldsymbol{H} + \left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^{T} \\
\boldsymbol{X} &= \boldsymbol{X} + \boldsymbol{K}\left(\boldsymbol{Z} - \boldsymbol{H}\boldsymbol{X}\right) \\
&= \boldsymbol{X} + \left(\boldsymbol{H}^{T}\boldsymbol{H} + \left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^{T}\boldsymbol{Z} - \left(\boldsymbol{H}^{T}\boldsymbol{H} + \left(\frac{\boldsymbol{P}}{R}\right)^{-1}\right)^{-1}\boldsymbol{H}^{T}\boldsymbol{H}\boldsymbol{X}
\end{aligned}
$$

所以 solution 中非增益项的尾巴，来自迭代过程！

```
/*** Rematch Judgement ***/
nearest_search_en = false;
if (flg_EKF_converged || ((rematch_num == 0) && (iterCount == (NUM_MAX_ITERATIONS - 2))))
{
    nearest_search_en = true;
    rematch_num ++;
}
```

lidar 测量每滤波更新一次，就令算法不再找最近平面，除非①滤波收敛或②未重匹配过且当前为倒数第二次迭代。当重匹配过或最后一次迭代，更新方差。

```
/*** Convergence Judgements and Covariance Update ***/
if (!EKF_stop_flg && (rematch_num >= 2 || (iterCount == NUM_MAX_ITERATIONS - 1)))
{
    if (flg_EKF_inited)
    {
        /*** Covariance Update ***/
        // G.setZero();
        // G.block<DIM_STATE,6>(0,0) = K * Hsub;
        state.cov = (I_STATE - G) * state.cov;
        total_distance += (state.pos_end - position_last).norm();
        position_last = state.pos_end;
        geoQuat = tf::createQuaternionMsgFromRollPitchYaw
                    (euler_cur(0), euler_cur(1), euler_cur(2));

        VD(DIM_STATE) K_sum  = K.rowwise().sum();
        VD(DIM_STATE) P_diag = state.cov.diagonal();
        // cout<<"K: "<<K_sum.transpose()<<endl;
        // cout<<"P: "<<P_diag.transpose()<<endl;
        // cout<<"position: "<<state.pos_end.transpose()<<" total distance: "<<total_distance<<endl;
    }
    EKF_stop_flg = true;        xuankuzcr, 3个月前 • [Release] release source code & dataset & hardwar…
}
solve_time += omp_get_wtime() - solve_start;
```

//将特征点增加到增量地图 kd 树中

map_incremental();

```
publish_frame_world(pubLaserCloudFullRes);
// publish_visual_world_map(pubVisualCloud);
publish_effect_world(pubLaserCloudEffect);
// publish_map(pubLaserCloudMap);
publish_path(pubPath);
```

//发布全局帧的点云(整体/下采样)、使用到的有效点云、轨迹

最后输出一个 PCD

至此，FAST-LIVO 阅读完成（2023.2.23）