cmake:

cmake\_minimum\_required(VERSION 2.8)

project(aaa)

#使用c++11

add\_compile\_options(-std=c++11)

add\_definitions(-std=c++11)

#编译模式

set(CMAKE\_BUILD\_TYPE “Release”)

set(CMAKE\_CXX\_FLAGS\_RELEASE “$ENV{CXXFLAGS} -O3 -Wall”)

set(CMAKE\_CXX\_FLAGS\_DEBUG “$ENV{CXXFLAGS} -O0 -Wall -g -ggdb”)

#ROS\_CATKIN

find\_package(catkin REQUIRED COMPONENTS

rospy

roscpp

std\_msgs

sensor\_msgs

geometry\_msgs

pcl\_ros

tf

std\_srvs

eigen\_conversions

message\_generation

)

add\_service\_files(

FILES

saveMap.srv

)

ros的东西后面再学，参考链接：

<http://wiki.ros.org/cn/ROS/Tutorials/CreatingMsgAndSrv>

glog:

include\_directories(${GLOG\_INCLUDE\_DIRS})

list(APPEND ALL\_TARGET\_LIBRARIES ${GLOG\_LIBRARIES})

geographic\_lib:

add\_subdirectory(${CMAKE\_SOURCE\_DIR}/third\_party/GeographicLib/include/)

学会使用file：

file(GLOB\_RECURSE ALL\_SRCS “\*.cpp”)

file(GLOB\_RECURSE NODE\_SRCS “\*\_node.cpp”)

list(REMOVE\_ITEM ALL\_SRCS ${NODE\_SRCS})

add\_executable(aaa ${ALL\_SRCS})

cmake阶段打草完毕

先看subscriber

类怎么写？

成员变量是什么？

成员变量是一个ros::Subscriber，一个ros::NodeHandle，还有一个就是接收容器deque<DataType>

成员函数是什么？

既然是订阅，就一定有一个msg\_callback，它的参数是ros消息的ConstPtr

还留了一个接口，可以把成员变量即缓冲区里面的deque传递出去，取名叫ParseData

每一个消息对应的自定义数据类型是什么？

成员变量一一对应即可，比如imu就是加速度、角速度以及姿态，注意要包含时间戳

再看publisher

成员变量是什么？

一个ros::Publisher，一个ros::NodeHandle，再加一个ros消息类型，比如nav\_msgs::Odometry，或者sensor\_msgs::PointCloud2

成员函数是什么？

主要是Publish函数，它接收自定义的消息类型，最后调用ros::Publisher对象

接下来是TF的使用

TF是干什么的呢？在这里是监听lidar到imu的外参的

我们从里到外地来研究一下这个东西，哈哈哈

自定义类TFListener，成员变量：一个nodehandle，两个frame的id，以及一个tf::transformListener listener\_。

成员函数有两个：

1. 将tf的transform转换成eigen的transform。

这里面涉及到tf里面的内置函数，具体不展开了，用的时候

1. 查询函数

函数返回一个bool值，表示是否查到。

参数返回值是一个Eigen::Matrix4f

这里的关键是，tf::transformListener这个类有一个lookupTransform的成员函数，它返回void，有4个参数，分别是：

* target frame的id
* source frame的id
* ros::Time
* tf类型的transform

具体使用如下：

if (lidar\_to\_imu\_ptr->LookupData(lidar\_to\_imu)) {

transform\_received = true;

}

这里有一个利用global\_definiton的小细节，可以拿来学习一下

它想解决什么问题呢？解决程序的可移植性问题！即不同的用户把程序放到任意的目录内，都绝对不会影响程序的编译运行。

然后是怎么做的呢？

1. 单独建里一个global\_defination.cmake文件
2. 在文件里面set一个变量

set(WORK\_SPACE\_PATH ${PROJECT\_SOURCE\_DIR})

这个不管你把程序放在任何目录下，都可以被变量WORK\_SPACE\_PATH保存下来

3、关键：cmake里面可以写一个configure\_file函数，它有两个参数，一个是输入，一个是输出，我们习惯上把输出文件放在binary\_dir里面：  
configure\_file(${PROJECT\_SOURCE\_DIR}/include/${PROJECT\_NAME}/global\_defination.h.in

${PROJECT\_BINARY\_DIR}/include/${PROJECT\_NAME}/global\_defination.h)

)

1. 然后把这个${PROJECT\_BINARY\_DIR}/include/${PROJECT\_NAME}给include进来
2. include\_directories(${PROJECT\_BINARY\_DIR}/include)
3. 在global\_defination.h.in里面定义变量，利用@xxx@运算符

const std::string WORK\_SPACE\_PATH = “@WORK\_SPACE\_PATH@”

1. 在需要用到该变量的地方include头文件

搞定。

现在只剩models和front\_end了。之所以有models，是因为匹配和体素滤波两个环节可多次复用，故将其提取出来

好，那我们先看匹配，匹配需要写成多态。所以：

1. 在registration\_interface.hpp里面写接口类：

class RegistrationInterface。里面有三个public的成员，其中一个是析构函数，virtual ~X() = default; 这样写可以防止析构不完全

第二个是setInputTarget函数

第三个就是ScanMatch函数，这个函数为了写的严谨使用，可以这么搞：返回值bool类型，表示是否能匹配上；参数有四个：source点云、预测位姿、矫正后的点云、变换矩阵。其中前两个是const类型，是输入；后面两个是输出。

1. 在ndt\_registration.hpp写具体的派生类

class NDTRegistration: public RegistrationInterface

成员变量是什么呢？类需要什么组件？

pcl::NormalDistributionsTransform<CloudData::POINT, CloudData::POINT>::Ptr

成员函数又是什么呢？

首先是重载两个纯虚函数，设置target和scan\_match。还有两个构造函数，分别是用参数初始化，以及用YAML初始化。

我们先看参数初始化，后面再看YAML初始化

参数初始化，分别设置NDT的分辨率、步长、Epsilon以及最大迭代次数

接下来是非常需要学习的YAML初始化

总结一下，如果想通过YAML进行初始化，需要进行哪些操作呢？

1. 在cmake里引入YAML
2. 编写yaml文件
3. 程序文件引入头文件：#include <yaml-cpp/yaml.h>
4. 编写处理函数，参数是const YAML::Node& node。  
   调用方式是：float A = node[“A”].as<float>()
5. 最外层，YAML node

std::string config\_file\_path = WORK\_SPACE\_PATH + "/config/front\_end/config.yaml";

YAML::Node config\_node = YAML::LoadFile(config\_file\_path);

nice！！！继续炖

接下来是体素滤波部分

先是接口，其实接口设计很简单，体素滤波不就是输入一个原始点云，输出一个滤波后的点云嘛！

程序部分也跟前面大同小异，只不过它要控制的参数是leaf\_size的xyz

好了，主体功能模块基本都分析完了，接下来我们看一看node程序是怎么写的，又是怎么封装到flow里面的。

先看test\_frame\_node.cpp

1、ros程序上来两板斧：

ros::Init(argc, argv, “node\_name”);

ros::NodeHandle nh;

2、接下来通过类的智能指针来创建订阅与发布

std::shared\_ptr<GNSSSubscriber> gnss\_sub\_ptr = std::make\_shared<GNSSSubscriber>(nh, 消息名, buffer\_size);

这样类似的方式搞定sub、pub以及tf

1. 用deque来搞imu、gnss、lidar的缓冲区

std::deque<GNSSData> gnss\_data\_buff;

1. 设定外参

Eigen::Matrix4f lidar\_to\_imu = Eigen::Matrix4f::Identity();

1. ros循环  
   设定一个ros::Rate，参数是频率，然后while(ros::OK())，循环里面先进行一次ros::spinOnce()，即运行订阅信息的回调函数
2. 循环里面进行的主要内容是什么？

设立站心

得到局部笛卡尔

得到里程计（4\*4矩阵表示）

用里程计来注册点云

点云累积显示

结束

ok，到现在还没有管前端呢。

前端类的设计思路是什么呢？类大概分为两种，一种是数据型的类，一种是功能型的类。对于数据型的类，首先是要考虑其成员变量，再给他们赋予一些基于成员变量的函数；而对于功能型的类，则要从功能入手，看实现这个功能需要哪些组件或者子功能。

子功能体素滤波和匹配已经单独拿出来了，还有的子功能是关键帧、滑动窗口以及位姿初始化

关键帧就是一帧点云，加上它对应的位姿

滑动窗口就是用一个deque把关键帧存起来，取名叫local\_map，当长度超出时，就把最前面的点云扔出去

位姿预测就是利用运动模型导出的位姿

分析完了子功能，下面就是接口，包括初始位姿、地图输出、位姿输出。

下面具体分析代码

先定义内置的Frame类型：

struct Frame {

Eigen::Matrix4f pose = Eigen::Matrix4f::Identity();

CloudData cloud\_data;

}

接下来是private类型的一些成员变量：

体素滤波相关的：

std::shared\_ptr<CloudFilterInterface> frame\_filter\_ptr\_;

std::shared\_ptr<CloudFilterInterface> local\_map\_filter\_ptr\_;

std::shared\_ptr<CloudFilterInterface> display\_filter\_ptr\_;

匹配相关的：

std::shared\_ptr<RegistrationInterface> registration\_ptr\_;

注意在这种多态类中，我们总是定义基类的指针，这样可以实现多态

滑动窗口：

std::deque<Frame> local\_map\_frames\_;

std::deque<Frame> global\_map\_frames\_;

点云：

CloudData::CLOUD\_PTR local\_map\_ptr\_;

CloudData::CLOUD\_PTR global\_map\_ptr\_;

CloudData::CLOUD\_PTR result\_cloud\_ptr\_;

当前帧：

初始位姿：

Eigen::Matrix4f init\_pose\_ = Eigen::Matrix4f::Identity();

接下来看一些方法