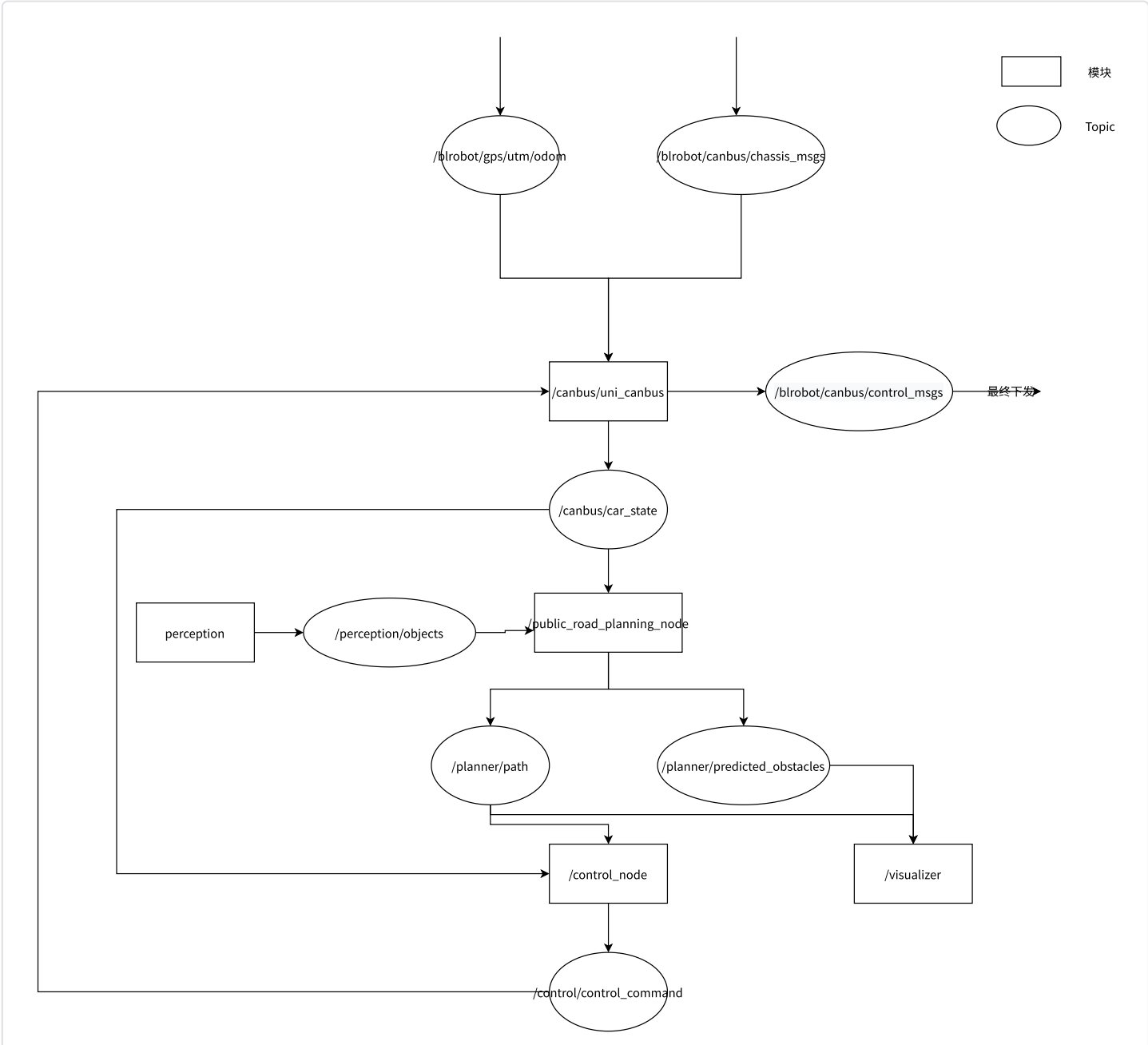
功能模块说明和调试文档V1.3

版本变更记录

版本名	说明
V1.0	各模块流程图和相应说明
V1.1	更新Planner模块限速配置说明和Control模块参数说明
V1.2	更新canbus模块详细说明
V1.3	补充信号流Topic及其对应proto类型

模块的运行流程图



Topic类型说明

Topic Name	类型
/blrobot/gps/utm/odom	nav_msgs::Odometry

/blrobot/canbus/chassis_msgs	chassisInfo.msg
/blrobot/canbus/control_msgs	busControl.msg
/canbus/car_state	common-protocol/proto/common/vehicle_state/vehicle_state.proto
/perception/objects	common-protocol/proto/perception/perception_obstacle.proto
/planner/predicted_obstacles	common-protocol/proto/perception/perception_obstacle.proto
/planner/path	planner/planner_common/common/proto/path.proto
/control/control_command	control/control_common/common/proto/control_command.proto

各个模块说明

地图文件

拷贝map文件夹到/opt/uni/下

Canbus 模块

转发车辆状态信息，包括从gnss接收到的定位信息和从底盘读取的车辆速度和转角等当前状态信息

```
启动命令
拷贝control文件夹到/opt/uni/下
source /opt/uni/control/setup.bash
roslaunch canbus beiligong.launch
```

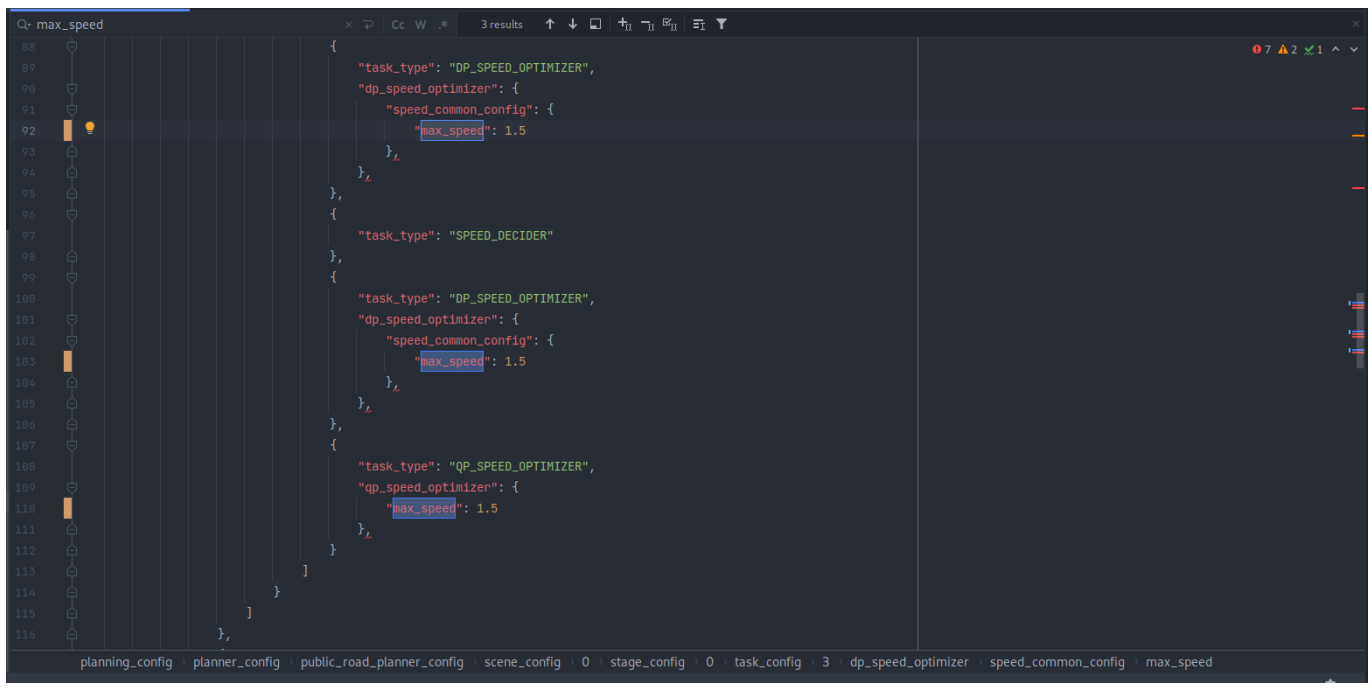
> **1113更新**
>
> Canbus模块代码详细说明
>
> 1. beiligong_message.h（deprecated）最早用做can信号通信的消息结构体，后来有了chassisinfo之后就废弃了，直接接chassisinfo
> 2. VehicleControllerBeiLiGong 继承自 VehicleController
>   a. runController : 创建订阅的callback，主要订阅控制指令(control_command)和车辆位置(/blrobot/gps/utm/odom)，同时创建了定频发送器保证CarState定频发送50hz
>   b. carStatePublisher：发送CarState的函数，本来考虑了插值的后来考虑到并没有多源头的信号所以改成了直接发送
>   c. repeatCmd：重复发送bus_control_msg
>   d. cmdHandlerV2：把从控制模块接收到的控制指令(control_command)转换为bus control msg进行下发

Planner模块

接收car_state车辆状态信息，同时载入地图计算车辆行驶轨迹

```
启动命令
拷贝planner文件夹到/opt/uni/下
source /opt/uni/planner/setup.zsh
roslaunch planner planner_liangjiang.launch
```

> 1109 新增调整全局限速的功能
>
> 1. 打开/opt/uni/planner/share/planner/json/planner_liangjiang.json
> 2.



修改max_speed这个参数，总共有三个，全部修改为期望的数值（单位为m/s）比如我这里设置成了1.5m/s。同时注意一下这个**会与地图限速取小值**，也就是target_speed_limit = min(lane_speed_limit,config_speed_limit)

Visualizer可视化模块

显示当前车辆位置，显示地图车道线，显示车辆规划出来的轨迹

```
启动命令
拷贝visualizer文件夹到/opt/uni/下
source /opt/uni/visualizer/setup.zsh
roslaunch visualizer visualizer_liangjiang.launch
```

Control 控制模块

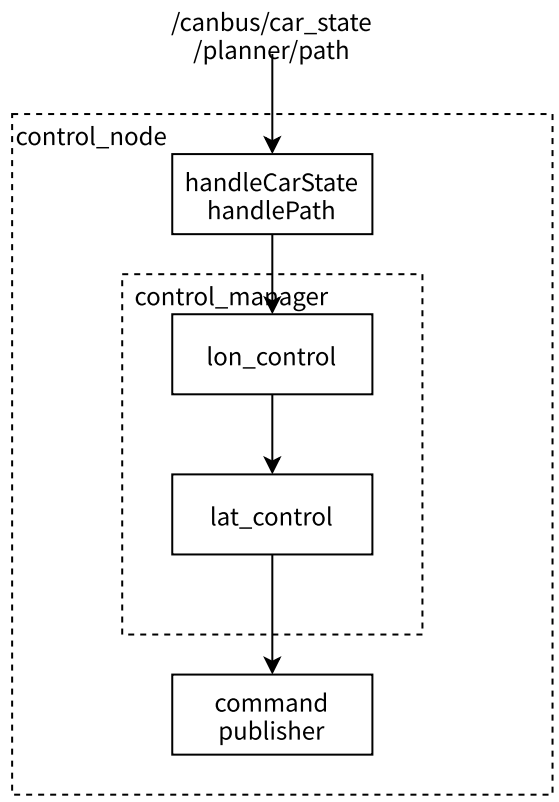
根据车辆当前位置和规划轨迹计算出控制指令

```
启动命令
拷贝control文件夹到/opt/uni/下（canbus那一步如果拷贝了就跳过这步）
source /opt/uni/control/setup.zsh
roslaunch control control.launch
```

> 1109更新控制模块参数说明
>
> control.cfg
> 纵向：
> 最大加速度acceleration_limit
> 最大减速度deceleration_limit

> 横向：
>
> angle_slew_rate 单位 读每秒

代码架构说明



control_node.cpp

```
// 接收car state信号，运行控制器，计算控制量。
void ControlNode::handleCarState(const std_msgs::StringConstPtr &msg)
// 接收path信号
void ControlNode::handlePath(const control_common::Bytes::ConstPtr &msg)
// 发布控制命令
void ControlNode::publishCommand(const proto::ControlCommand &control_command)
```

control_manager.cpp

```
// 构造函数，根据配置文件生成横向和纵向控制器
ControlManager::ControlManager(const proto::ConfigControlManager &config)
// 运行纵向控制器，计算纵向控制量
// 运行横向控制器，计算横向控制量
bool ControlManager::control(const uni::common::VehicleState &car_state,
                             const proto::Path &path,
                             proto::ControlCommand *control_command)
```

controller_lon

lon_control.cpp

纵向控制器基类，提供接口函数和纵向控制器生成函数

lon_control_speed.cpp

```
// 速度控制器，输出速度控制量
bool LonControlSpeed::processControl(
    const uni::common::VehicleState &car_state, const proto::Path &path,
```

```
    proto::ControlCommand * /*control_command*/) {
 // 根据当前位置在path找最近点，以该点速度为速度控制量
 if (path_analyzer_ptr_->findNearestPoint(car_state, path, &iter)) {
     speed_cmd = iter->speed();
 }
 // 根据设定的最大加减速度对速度控制量限幅
   target_speed_ += clamp(speed_cmd - target_speed_,
                          -dec_limit_ / config_car_.control_frequency(),
                          acc_limit_ / config_car_.control_frequency());
}
```

lon_control_cruise.cpp

```
// 加速度控制器，输出加速度控制量
bool LonControlCruise::processControl(
    const uni::common::VehicleState &car_state, const proto::Path &path,
    proto::ControlCommand * /*control_command*/) {
     // 根据当前位置在path找最近点，以该点速度为速度控制量
     if (path_analyzer_ptr_->findNearestPoint(car_state, path, &iter)) {
         speed_cmd = iter->speed();
     }
     // 根据设定的最大加减速度对速度控制量限幅
     target_speed_ += clamp(speed_cmd - target_speed_,
                          -dec_limit_ / config_car_.control_frequency(),
                          acc_limit_ / config_car_.control_frequency());
     // 计算加速度前馈量（表征当前的期望加速度）
     acc_ff = acc_ff * factor * config_.acc_ff_gain();
     // 计算加速度反馈量（表征期望状态和当前状态的差）
     acc_fb =
      controller_lon_pid_ptr_->setControl(target_speed_, car_state.speed());
     // 最终控制量
     double acc_cmd = clamp(acc_ff + acc_fb, -dec_limit_, acc_limit_);

}
```

controller_lat

lat_control.cpp

横向控制器基类，提供接口函数和纵向控制器生成函数

lat_control_pid.cpp

```
// 基于pure pursuit的横向控制器，输出前轮转角。可参考
// https://blog.csdn.net/weixin_42301220/article/details/124882144
// https://blog.csdn.net/qq_35635374/article/details/120704494
bool LatControlPid::control(const uni::common::VehicleState &car_state,
                            const proto::Path &path,
                            proto::ControlCommand *control_command){
   // 根据当前车速计算前馈前视距离（前馈表征目标点的曲率）
   getFeedForwardAheadDistance(speed_stamp, &feedforward_ahead_distances)
   // 根据当前车速计算反馈前视距离（反馈表征车辆当前朝向和目标点朝向的偏差）
   getFeedbackCurvature(car_state, &feedback_curvature)
   // 根据前视距离在path上在对应目标点
   path_analyzer_ptr_->updateMatchedPoints ()
   // 得到反馈曲率 2.0 * 横向误差 / sqr(当前车辆到目标点距离)
   getFeedbackCurvature(car_state, &feedback_curvature)
```

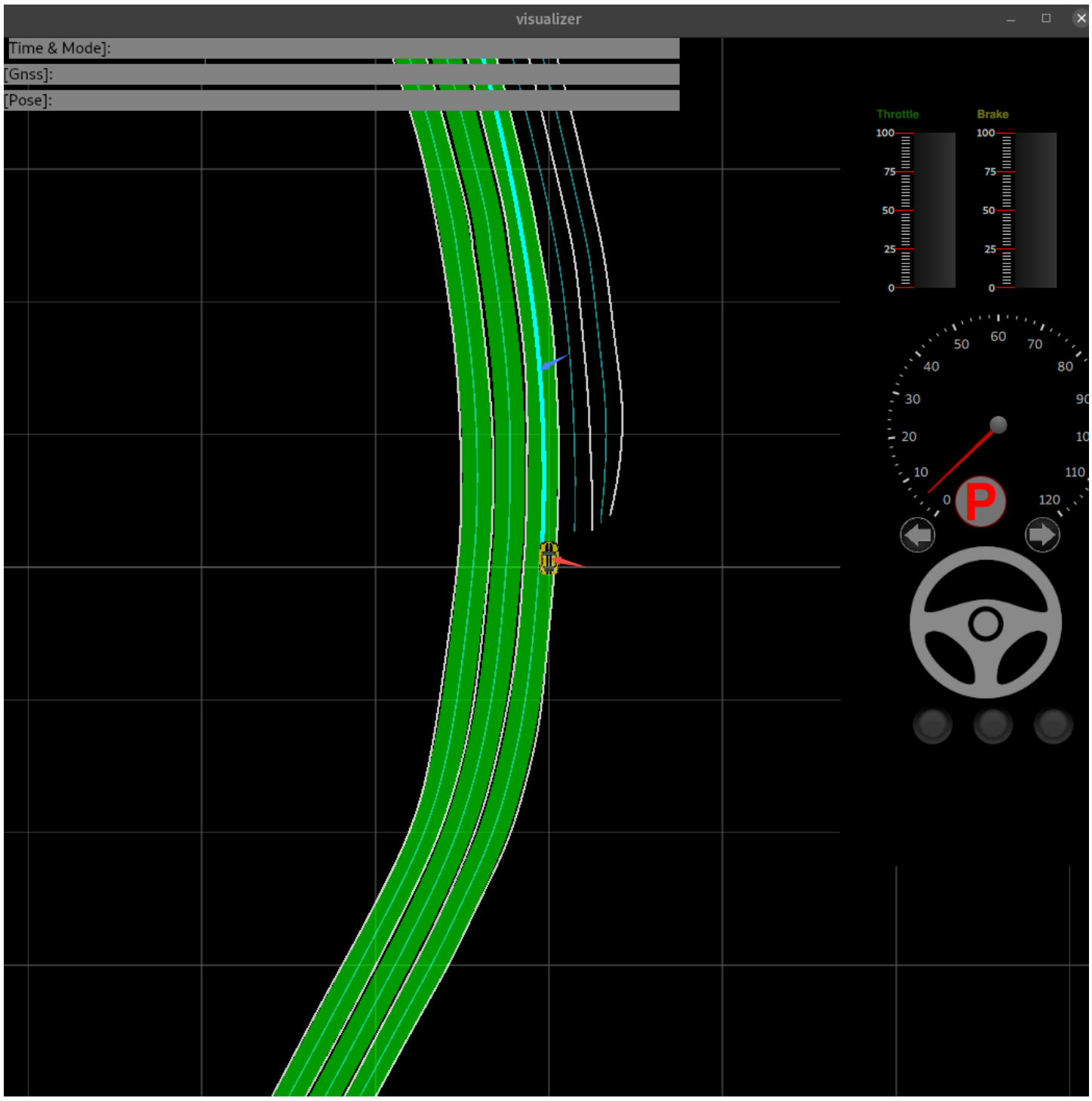
```
    // 得到前馈曲率， 目标点的曲率
    getFeedforwardCurvature ()
    // 根据atan(曲率/轮距) 得到前轮转角，并通过转角限速限幅
    updateSteeringAngleSetPoint ()
}
```

Debug工具

1. 信号查看工具

```
python vehicle_test.py -h 查看帮助
python vehicle_test.py -c 0
0: /canbus/car_state \
1: /planner/path \
2: /control/control_command"
```

最终可视化的效果



蓝色箭头所示为车辆规划出的轨迹，红色箭头所示为自车当前位置

Debug流程

1. 按上面步骤启动所有模块，并启动发送bus_msgs/chassisInfo和/blrobot/gps/utm/odom的程序，还有接收/blrobot/canbus/control_msgs的程序
2. rqt_graph查看所有模块和Topic流转是否与最上面图示一致
3. 将车辆手动开到图上，**不闭环查看车辆是否在地图上**（首先保证不会差距很远）
4. 录制除了点云外的所有topic（便于Debug），开启自动模式看车辆是否能闭环（注意安全随时接管），同时我们这边做了地图限速的保护（暂定速度为10km/h）