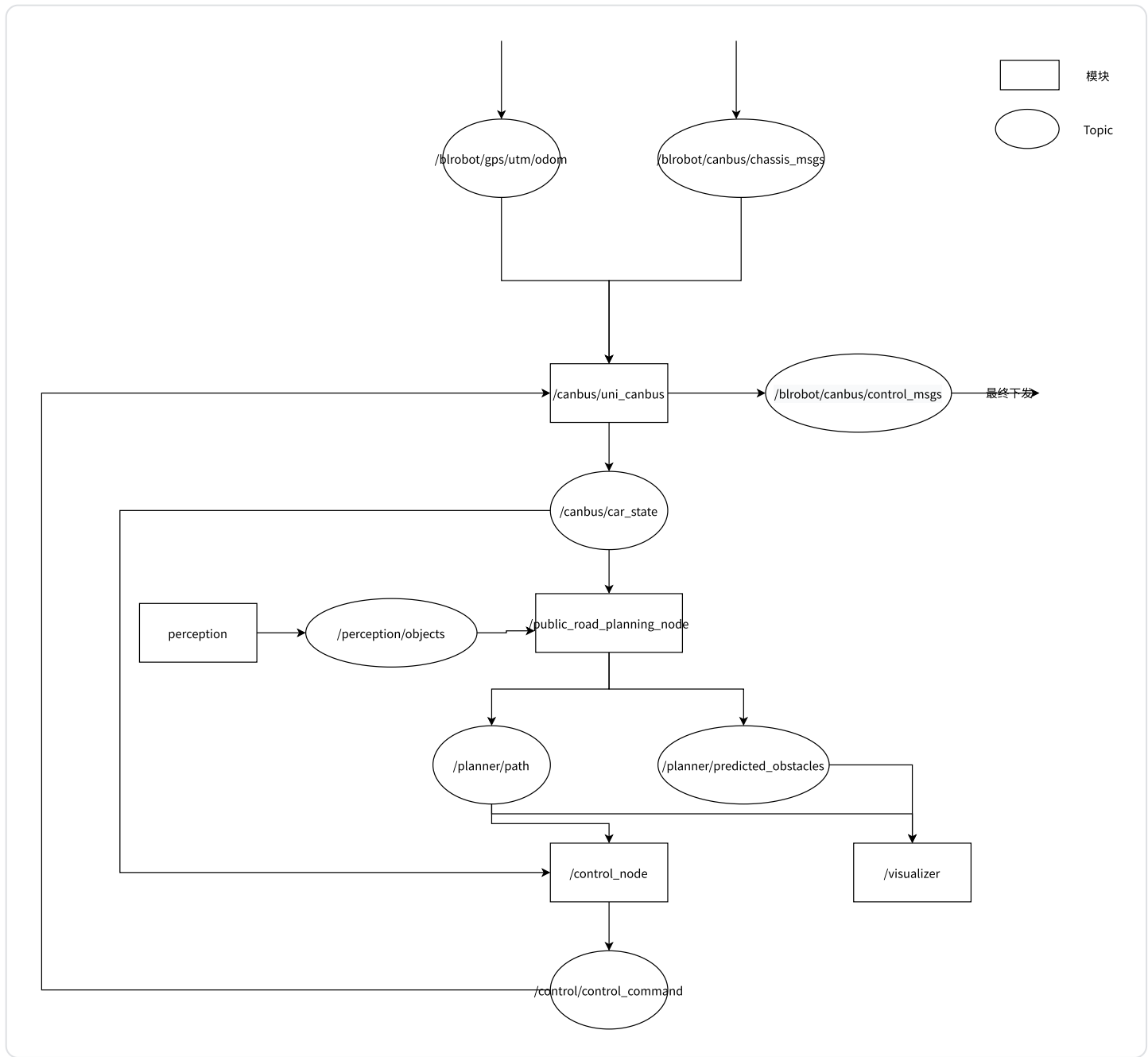


功能模块说明和调试文档V1.7

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
V1.4	增加control、visualizer、simulator代码及使用说明
V1.5	交付新版地图，新routing配置和routing模块说明
V1.6	更新高精度地图信息说明，修改simulator模块说明的颜色
V1.7	对应control_cruise.cfg更新控制参数说明

模块的运行流程图



Topic类型说明

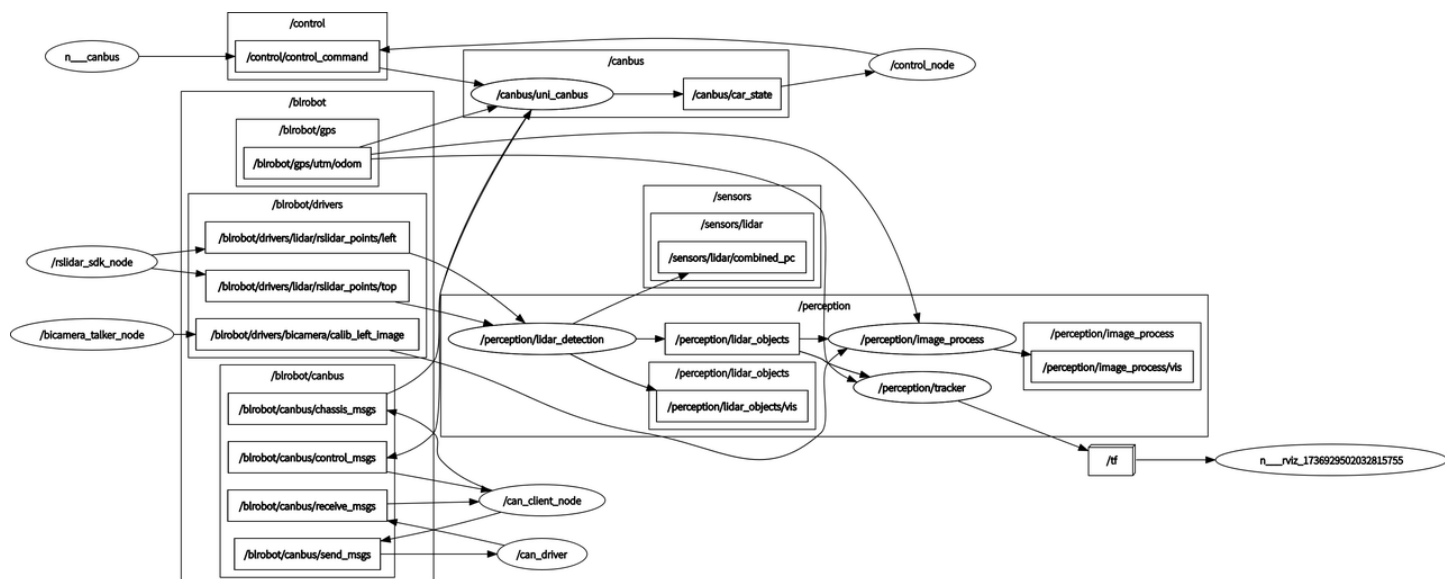
Topic Name	类型
/blrobot/gps/utm/odom	nav_msgs::Odometry
/blrobot/canbus/chassis_msgs	chassisInfo.msg
/blrobot/canbus/control_msgs	busControl.msg
/canbus/car_state	common-protocol/proto/common/vehicle_state/vehicle_state.proto
/perception/objects	common-protocol/proto/perception/perception_obstacle.proto

/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

底盘通信模块说明

1. EMUC_B202 V3.2版本文件夹中打开终端,启动底盘can驱动程序，
bash start_2_4_sxxx.sh
 2. 启动can接口程序，在Document/base-master里面 roslaunch can_driver can_driver.launch
roslaunch can_client can_client_node
 3. 启动
 - a. source /opt/beiligong/control/setup.bash
 - b. roslaunch canbus beiligong.launch
 4. 手动验证信号通信（可跳过）
 - a. 再打开另一个Terminal，进入 /opt/beiligong/control/lib/canbus 目录
执行./key_operator （注意此处报错时，执行 source ../../setup.bash）
按键说明：
Key "w": increase throttle value
Key "b": brake
Numpad "=": Increase steer angle
Numpad "-": Decrease steer angle
- 注意4.2前此处需要切换遥控器旋钮到自驾状态
- 启动完成后，与底盘相关的模块在ros graph里张这样



地图文件

拷贝map文件夹到/opt/uni/下

高精地图信息说明

主要proto定义文件在这里

https://github.com/Uni321/common_protocol_release/blob/main/proto/semantic_map/roadmap/road_map.proto

```
1 message RoadMap {
2   optional MapProjection projection = 1;
3   repeated Lane lanes = 2;
4   repeated LaneBoundary lane_boundaries = 3;
5   repeated ClearZone clear_zones = 4;
6   repeated CrossWalk cross_walks = 5;
7   repeated Junction junctions = 6;
8   repeated StopLine stop_lines = 7;
9   repeated StopSign stop_signs = 8;
10  repeated TrafficLight traffic_lights = 9;
11  repeated YieldSign yield_signs = 10;
12 }
```

主要有Lane, LaneBoundary, CrossWalk, Junction, StopLine, Traffic Light

具体的说明在每个字段的定义里面有，看字段名应该能得出相关信息

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

- 1 启动命令
- 2 拷贝control文件夹到/opt/uni/下
- 3 `source /opt/uni/control/setup.bash`
- 4 `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

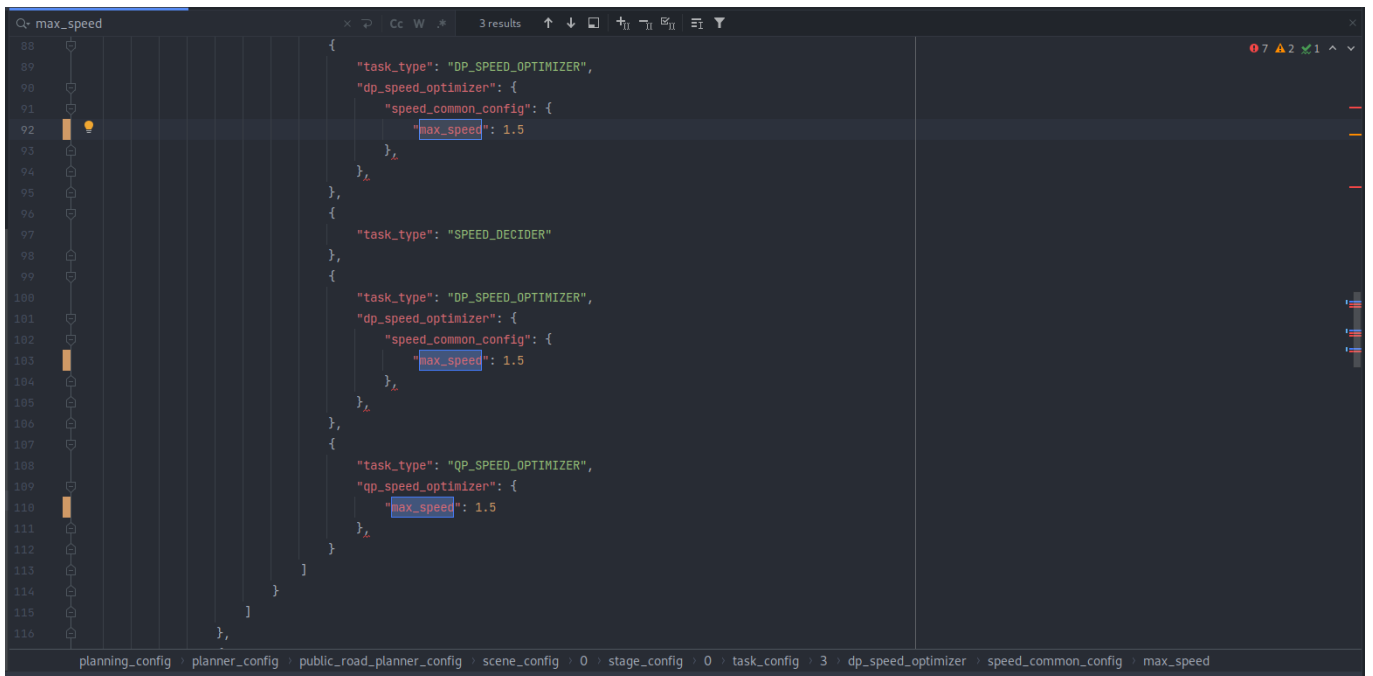
Planner模块

接收car_state车辆状态信息，同时载入地图计算车辆行驶轨迹

- 1 启动命令
- 2 拷贝planner文件夹到/opt/uni/下
- 3 `source /opt/uni/planner/setup.zsh`
- 4 `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.



```
88 {
89     "task_type": "DP_SPEED_OPTIMIZER",
90     "dp_speed_optimizer": {
91         "speed_common_config": {
92             "max_speed": 1.5
93         }
94     },
95 },
96 {
97     "task_type": "SPEED_DECIDER"
98 },
99 {
100     "task_type": "QP_SPEED_OPTIMIZER",
101     "qp_speed_optimizer": {
102         "speed_common_config": {
103             "max_speed": 1.5
104         }
105     },
106 },
107 {
108     "task_type": "QP_SPEED_OPTIMIZER",
109     "qp_speed_optimizer": {
110         "max_speed": 1.5
111     }
112 }
113 }
114 }
115 },
116 }
```

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 $\text{target_speed_limit} = \min(\text{lane_speed_limit}, \text{config_speed_limit})$

Routing模块说明


该模块属于planner的子模块，也就是我们平时所说的全局导航，根据起点和终点搜索出lane级别的导航路线

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

- 1 启动命令
 - 2 拷贝visualizer文件夹到/opt/uni/下
 - 3 source /opt/uni/visualizer/setup.zsh
 - 4 roslaunch visualizer visualizer_liangjiang.launch

操作说明：

-  A： 俯视图

S: 45°视图

D: 前视图

G: 显示/隐藏栅格

P: 显示/隐藏栅格route

B: 显示/隐藏障碍物polygon

W: 显示/隐藏人行道

E: 显示/隐藏预测线

O: 切换障碍物显示模式（不显示，显示polygon，显示 boundingbox，显示 polygon&&boundingbox）

Q: 窗口化/全屏

ESC: 关闭软件

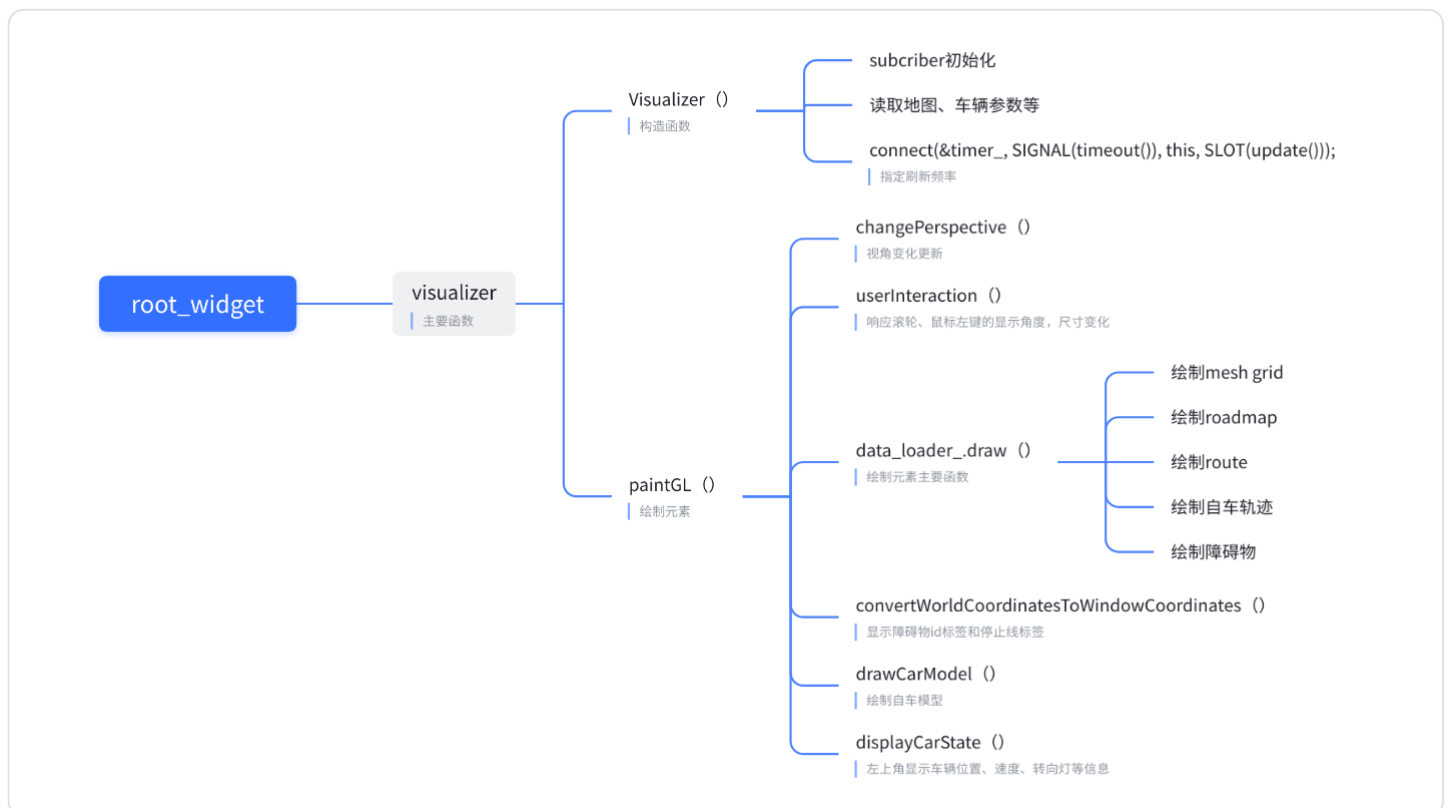
N: 显示/隐藏障碍物id标签

滑轮滚动: 放大缩小

滑轮按压: 平移

鼠标左键: 旋转

代码框架



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

- 1 启动命令
- 2 拷贝control文件夹到/opt/uni/下（canbus那一步如果拷贝了就跳过这步）

```
3 source /opt/uni/control/setup.zsh
4 roslaunch control control.launch
```

1109更新控制模块参数说明

control.cfg

纵向：

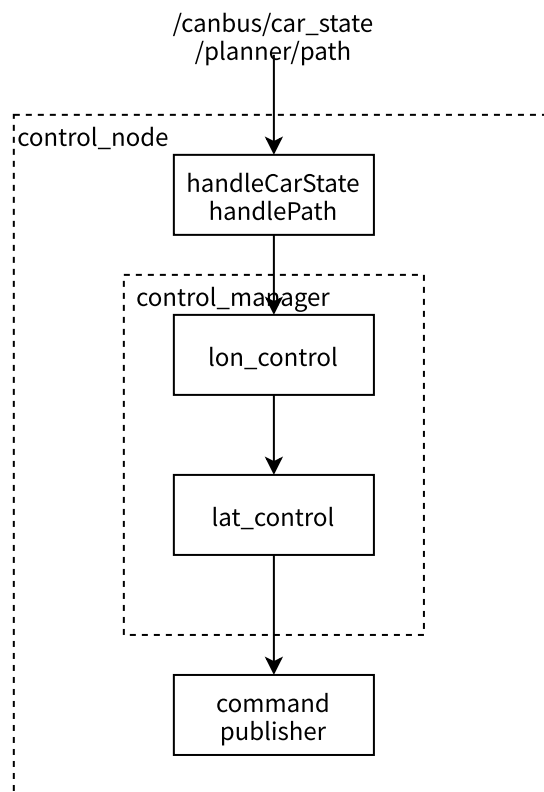
最大加速度acceleration_limit

最大减速度deceleration_limit

横向：

angle_slew_rate 单位 度每秒

代码架构说明



control_node.cpp

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


control_manager.cpp

```
1 // 构造函数，根据配置文件生成横向和纵向控制器
2 ControlManager::ControlManager(const proto::ConfigControlManager &config)
3 // 运行纵向控制器，计算纵向控制量
4 // 运行横向控制器，计算横向控制量
5 bool ControlManager::control(const uni::common::VehicleState &car_state,
6                               const proto::Path &path,
7                               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

```
1 // 速度控制器，输出速度控制量
2 bool LonControlSpeed::processControl(
3     const uni::common::VehicleState &car_state, const proto::Path &path,
4     proto::ControlCommand * /*control_command*/) {
5     // 根据当前位置在path找最近点，以该点速度为速度控制量
6     if (path_analyzer_ptr->findNearestPoint(car_state, path, &iter)) {
7         speed_cmd = iter->speed();
8     }
9     // 根据设定的最大加减速度对速度控制量限幅
10    target_speed_ += clamp(speed_cmd - target_speed_,
11                           -dec_limit_ / config_car_.control_frequency(),
12                           acc_limit_ / config_car_.control_frequency());
13 }
```

lon_control_cruise.cpp

```
1 // 加速度控制器，输出加速度控制量
2 bool LonControlCruise::processControl(
3     const uni::common::VehicleState &car_state, const proto::Path &path,
4     proto::ControlCommand * /*control_command*/) {
5     // 根据当前位置在path找最近点，以该点速度为速度控制量
6     if (path_analyzer_ptr->findNearestPoint(car_state, path, &iter)) {
7         speed_cmd = iter->speed();
8     }
9     // 根据设定的最大加减速度对速度控制量限幅
```

```

10     target_speed_ += clamp(speed_cmd - target_speed_,
11                             -dec_limit_ / config_car_.control_frequency(),
12                             acc_limit_ / config_car_.control_frequency());
13     // 计算加速度前馈量 (表征当前的期望加速度)
14     acc_ff = acc_ff * factor * config_.acc_ff_gain();
15     // 计算加速度反馈量 (表征期望状态和当前状态的差)
16     acc_fb =
17         controller_lon_pid_ptr_>setControl(target_speed_, car_state.speed());
18     // 最终控制量
19     double acc_cmd = clamp(acc_ff + acc_fb, -dec_limit_, acc_limit_);
20
21 }

```

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

```

1 // 基于pure pursuit的横向控制器，输出前轮转角。可参考
2 // https://blog.csdn.net/weixin_42301220/article/details/124882144
3 // https://blog.csdn.net/qq_35635374/article/details/120704494
4 bool LatControlPid::control(const uni::common::VehicleState &car_state,
5                             const proto::Path &path,
6                             proto::ControlCommand *control_command){
7     // 根据当前车速计算前馈前视距离 (前馈表征目标点的曲率)
8     getFeedForwardAheadDistance(speed_stamp, &feedforward_ahead_distances)
9     // 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)
10    getFeedbackCurvature(car_state, &feedback_curvature)
11    // 根据前视距离在path上在对应目标点
12    path_analyzer_ptr_>updateMatchedPoints ()
13    // 得到反馈曲率  $2.0 * \text{横向误差} / \text{sqr(当前车辆到目标点距离)}$ 
14    getFeedbackCurvature(car_state, &feedback_curvature)
15    // 得到前馈曲率, 目标点的曲率
16    getFeedforwardCurvature ()
17    // 根据atan(曲率/轮距) 得到前轮转角, 并通过转角限速限幅
18    updateSteeringAngleSetPoint ()
19 }

```

```

1 config_longitudinal_control {
2     cruise {
3         vel_pid_param {

```

```

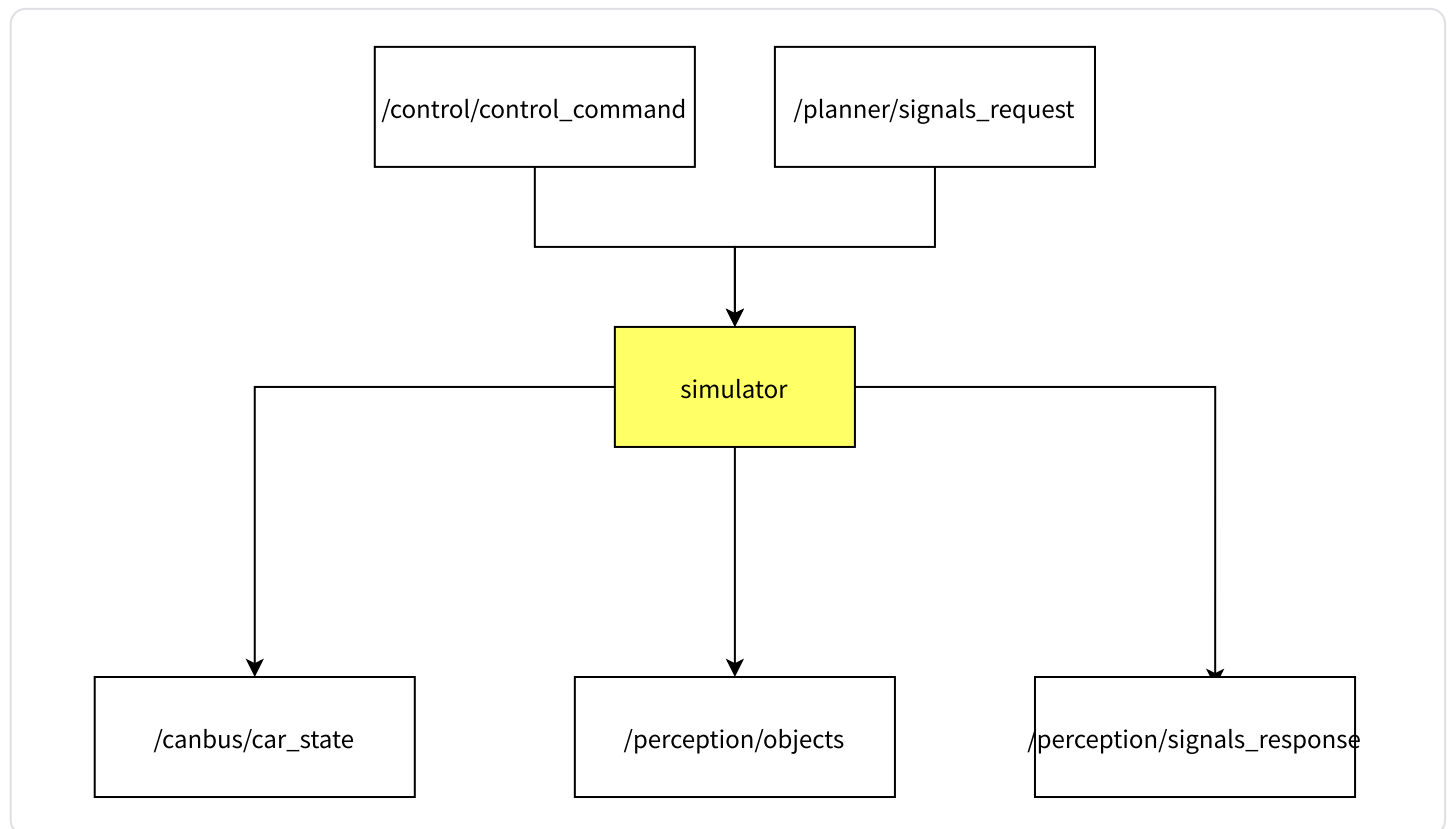
4      kp: 0.2
5      ki: 0.04
6      kd: 0.0
7      kt: 0.0
8      wp: 1.0
9      tau: 0.2
10     saturation: 1.0
11   }
12
13     acc_ff_gain: 0.6
14   gravity_ratio : 0.5
15   stop_jerk_ratio: 1.0
16 }

```

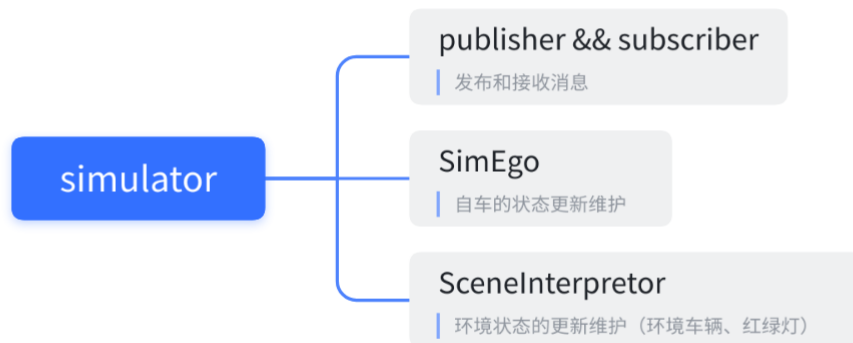
控制参数说明，以上部分来自文件control_cruise.cfg，vel_pid_param就是常规的PID控制参数，acc_ff_gain为前馈加速度的gain，stop_jerk_ratio为停车过程中jerk的比例

Simulator

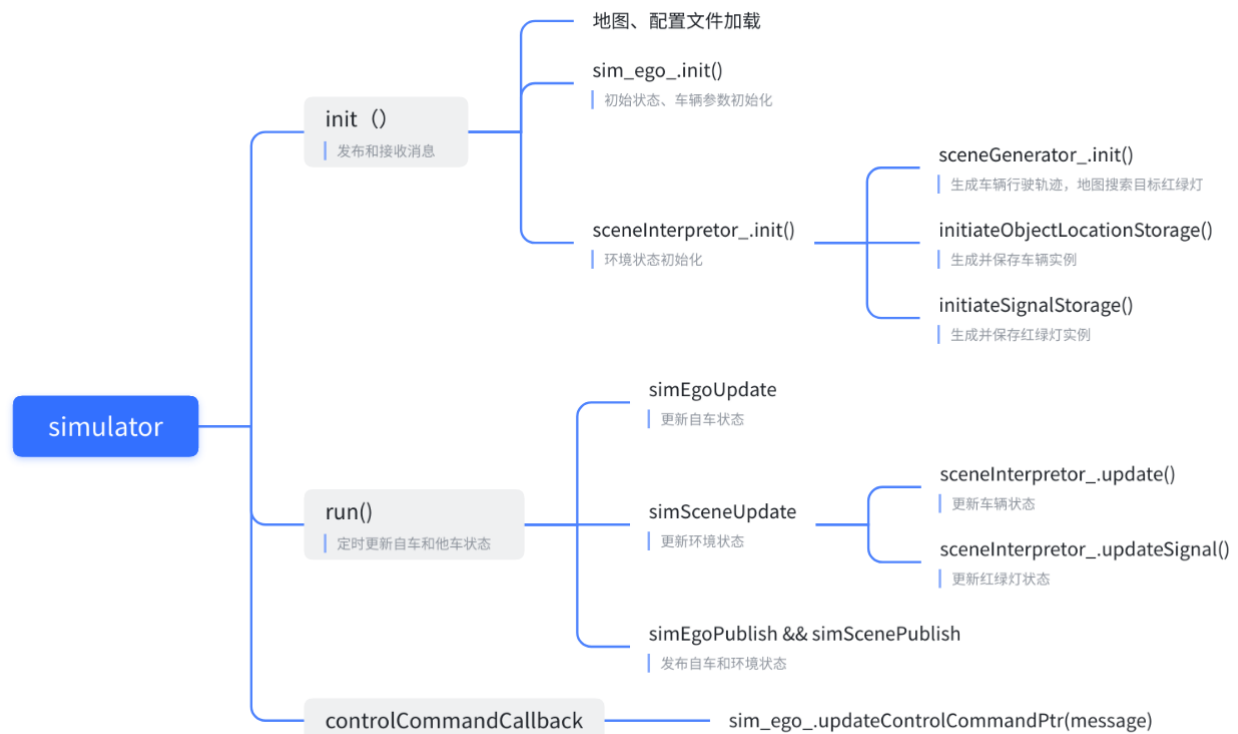
数据流



代码层级



pipeline



配置说明

Launch

```

1 launch>
2
  
```

```

3 <node name="simulator" pkg="simulator" type="simulator" output="screen">
4   <!-- 地图路径 -->
5   <param name="roadmap_filename" value="/opt/uni/map/map1120.txt"/>
6   <!-- 仿真场景配置文件路径 -->
7   <param name="simulation_scene_filename"
8     value="/opt/uni/simulator/share/simulator/simulation_liangjiang.cfg"/>
9   <!-- 自车参数文件路径 -->
10  <param name="config_car_filename"
11    value="/opt/uni/simulator/share/simulator/car.cfg"/>
12  <!-- 是否发布环境障碍物和红绿灯数据 -->
13  <param name="simulate_perception" value="true"/>
14  <!-- 环境障碍物预测轨迹点时间间隔 -->
15  <param name="prediction_step_t" value="0.1"/>
16  <!-- 环境障碍物预测轨迹时长 -->
17  <param name="prediction_duration" value="5.0"/>
18
19 </node>
20
21 </launch>

```

cfg 见 simulator/simulator_common/common/proto/simulator_scene.proto

```

1 sim_car {
2   # 车辆起始状态（位置，速度，朝向，加速度）
3   init_position {
4     x: 7120.63
5     y: 12506.2
6     z: 0.0
7   }
8   init_heading: -3.14
9   init_velocity: 4
10  init_acceleration: 0.0
11  init_gear: DRIVE
12 }
13
14
15 sim_objects {
16   id: 10
17   type: VEHICLE
18   dimension {
19     x: 4.8
20     y: 2.0
21     z: 2.3
22   }
23   # 障碍物路线设置 route config 可选 [xy_points] 或 [lane_id, lane_s] 两种模式

```

```

24 route_config {
25     xy_points {
26         points {
27             x: 7075.48
28             y: 12505.7
29         }
30         points {
31             x: 7029.79
32             y: 12395.6
33         }
34     }
35 }
36 # route_config {
37 #     lane_id: 122
38 #     lane_s: 1
39 #     lane_id: 118
40 #     lane_s: 20.0
41 #}
42
43 acceleration_max: 2.0
44 deceleration_max: 2.0
45 speed: 5.0
46 # 障碍物到达终点后是否从起点重新运动
47 reapeation : false
48 # 是否在stop sign前停止
49 stop_by_sign : true
50 # 在stop sign等待时间
51 stop_line_wait_time: 3.0
52 # 在仿真开始几秒后开始更新该障碍物状态
53 time_delay ; 2.0
54 # 距离自车小于该值开始更新该障碍物状态
55 start_distance_to_ego : 10.0
56 }
57
58 sim_objects {
59     id: 11
60     type: VEHICLE
61     dimension {
62         x: 4.8
63         y: 2.0
64         z: 2.3
65     }
66     # 也可直接输入path指定障碍物路径, 指定每个轨迹点的位置速度和纵向值
67     path{
68         path_points{
69             position{
70                 x: 8062.27

```

```
71     y: 13548.1
72     z: 0
73 }
74 speed: 1.0
75 s : 0.0
76 }
77 path_points{
78     position{
79         x: 8072.44
80         y: 13544.6
81         z: 0
82     }
83     speed: 1.0
84     s : 11
85 }
86 }
87 speed: 1.0
88 reapeation : false
89 stop_by_sign : false
90 time_delay: 2.0
91 }
92
93
94
95 sim_signals {
96     # 仿真route所覆盖的所有红绿灯
97     route_config{
98         xy_points {
99             points {
100                 x: 7075.48
101                 y: 12505.7
102             }
103             points {
104                 x: 6942.96
105                 y: 12506.2
106             }
107         }
108     }
109
110     #也可直接指定红绿灯id 仿真对应红绿灯
111     #traffic_light_id: 12
112
113     # 绿灯持续时间
114     green_duration: 10.0
115     # 黄灯持续时间
116     yellow_duration: 3.0
117     # 红灯持续时间
```

```
118   red_duration: 20.0
119   # 交通灯以红->黄->绿变化, start_time为交通灯开始时的时间, 根据在total time =
    (green_duration +
120   # yellow_duration + red_duration) 的位置决定初始灯的颜色, 取值必须小于total time
121   start_time: 10.0
122 }
```

运行方式

- 1 启动命令)
- 2 source /opt/uni/simulator/setup.zsh
- 3 roslaunch simulator simulator_xxx.launch

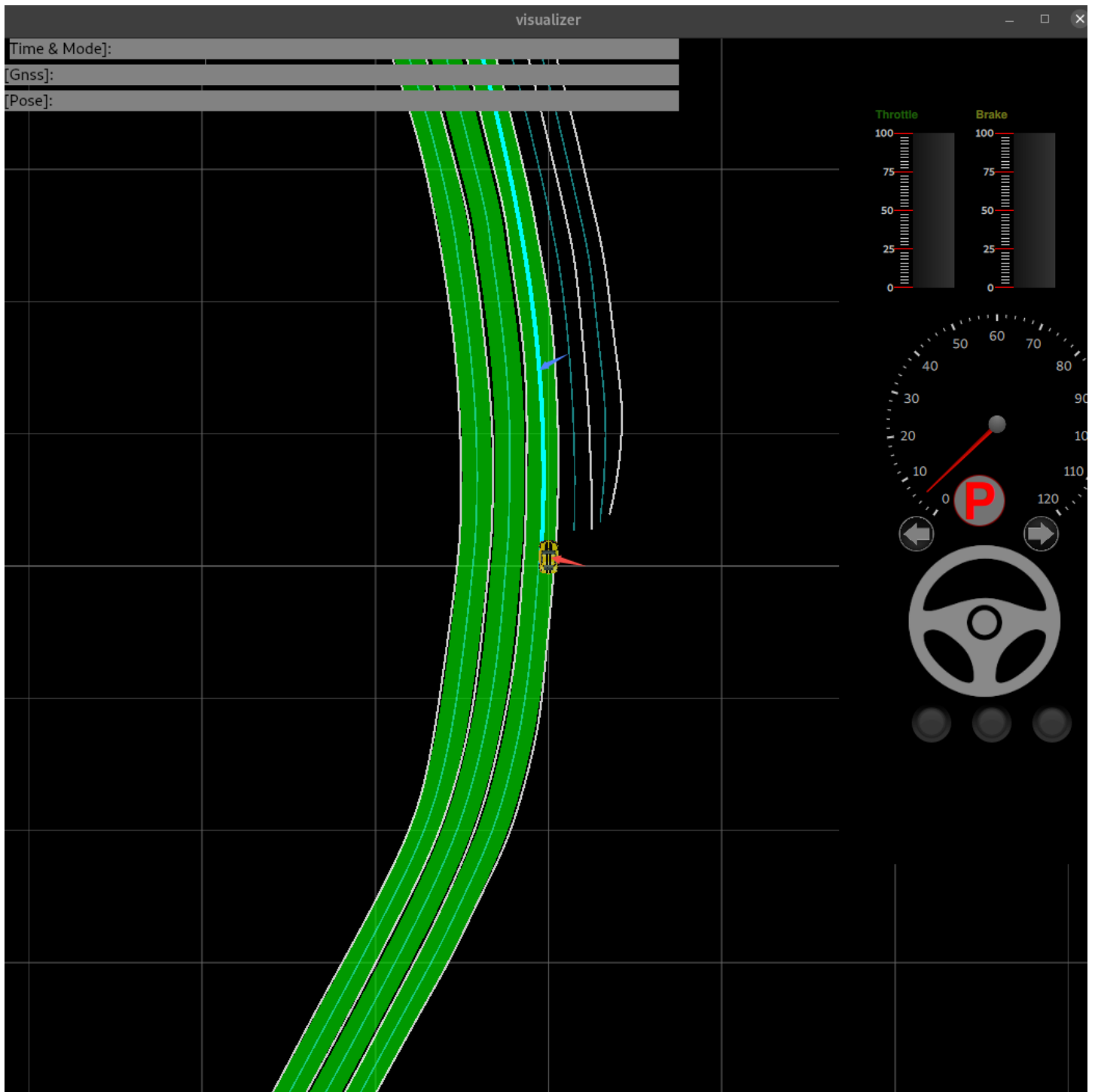
Debug工具

1. 信号查看工具

- ```
1 python vehicle_test.py -h 查看帮助
2 python vehicle_test.py -c 0
3 0: /canbus/car_state \
4 1: /planner/path \
5 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，启动CAN模块，EMUC下面3.1下面脚本start\_\*.sh。启动can模块，可能需要多启动几次，确保can在can0和can1。启动roslaunch can\_client 和 roslaunch can\_driver（参照上方的“底盘通信模块说明”），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）

## 新路线部署说明

**地图部署：**将 new\_map0204.txt(在map.zip里) 拷贝到/opt/uni/map/

**可视化部署：**将 visualizer\_beiligong.launch 拷贝到  
/opt/uni/map/opt/uni/visualizer/share/visualizer

**Planner Routing配置部署：**

1.将planner\_beiligong.json文件拷贝到/opt/uni/planner/share/planner/json/，将  
planner\_beiligong.launch文件拷贝到/opt/uni/planner/share/planner/

2.新路线启动命令则为roslaunch planner planner\_beiligong.launch

**Simulation部署：(如果仅为上车调试，那这一步可暂时跳过)**

将simulation\_beiligong.cfg文件拷贝到/opt/uni/simulator/share/simulator，将  
simulation\_beiligong.launch文件拷贝到/opt/uni/simulator/share/simulator

PS：各模块部署完成后启动命令依旧为ros启动，记得换launch文件就好