技术文档丨 Can总线与定位模块

CAN（Controller Area Network ）总线在整个无人驾驶系统中有着十分重要的作用。除了在VCU信号需要通过CAN总线进行传输外，无人车上的某些传感器（如雷达、Mobileye）的信号传递也是通过CAN实现的。

定位是让无人车知道自身确切位置的方法，这是一个美妙但是十分艰难的任务，同时也对无人驾驶车十分重要。定位不仅仅是找出自身的大概方位，而是要以**10cm**级别，将车感信息与高精地图信息进行比较来精确的位置寻找。

以下，ENJOY

Can总线

简 介

Can总线接收和执行来自**控制模块**的命令，收集汽车底盘的状态信息并反馈给控制模块。

输 入

- 控制命令

- 底盘状态
- 底盘各部件消息状态

Canbus模块主要包括以下几个部分：

- 车辆：车辆本身，包括车辆控制器和消息管理器
- CAN 客户端— CAN 客户端已被移动到`/modules/drivers/canbus` 目录下，因为它已经被使用canbus 协议的不同传感器共享。

通过继承`CanClient`类的方式，您可以在文件夹`can_client`中实现自己的CAN 客户端。

注意:

不要忘了在`'CanClientFactory'`（Can客户端工厂）类中注册您自己的CAN 客户端。

在文件夹`vehicle`下，您也可以通过继承`VehicleController` `MessageManager`类的方式实现自己的车辆控制器和消息管理器。

注意:

不要忘了在`` `VehicleFactory` ``类中注册您的车辆。

定位模块

简介

该模块提供定位服务。定位模块中提供了以下两种方法：

- 基于 **RTK(Real Time Kinematic，实时动态定位)**的方法，该方法结合GPS和IMU（惯性测量单元）信息。
- **多传感器融合**方法，该方法采用GPS，IMU和激光雷达的信息。

输入

在基于**RTK** 的定位方法中，有两个输入：

- GPS – 全球地理信息定位系统
- IMU – 惯性测量单元

在**多传感器融合**的定位方法中，由如下三个输入：

- GPS – 全球地理信息定位系统
- IMU – 惯性测量单元
- LiDAR – 激光雷达传感器

更多信息，请参阅以下论文：

作者：Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Yao Zhou, Hao Wang, Shiyu Song.

题目：《基于多传感器融合的不同城市场景下的坚固、精确的车辆定位》

"Robust and Precise Vehicle Localization Based on Multi-Sensor Fusion in Diverse City Scenes," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 4670-4677. doi: 10.1109/ICRA.2018.8461224. link

输出为一个由Protobuf格式定义的对象实例LocalizationEstimate，该实例可在`localization/proto/localization.proto`文件中查找到。

当前，基于RTK的定位方法由**RTKLocalization类**实现。如果您正在实现一个新的定位算法，例如**FooLocalization**，您可能需要执行以下步骤：

1. 在`proto/localization_config.proto`文件中，在`LocalizationType`中添加一个值为FOO的枚举类型。

2. 切换到`modules/localization`目录，然后创建一个名为`foo`的文件夹。在`foo`文件夹中，参照`rtk`目录下`RTKLocalization`类的代码实现自己的定位算法类`FooLocalization`。`FooLocalization`必须是`LocalizationBase`的子类。另外，参照`rtk/BUILD`文件创建一个名为`Also foo/BUILD`的文件。

3. 您需要在函数`Localization::RegisterLocalizationMethods()`中注册您的新类`FooLocalization`，该函数在文件`localization.cc`中实现。您可以在该函数的最后插入以下代码来注册自己添加的类：

```
localization_factory_.Register(LocalizationConfig::FOO, []()->LocalizationBase* {
```

4.确保您的代码编译时包含定义`FooLocalization` 类的头文件`FooLocalization`。

5. 现在您可以回到`apollo`的根目录，使用`bash apollo.sh build`命令编译您的代码。