

进阶课程④② | Apollo实战——车辆与循迹驾驶能力实战

循迹自动驾驶是指让车辆按照录制好的轨迹线进行自动驾驶，其涉及到自动驾驶中最基本的**底盘线控能力、定位能力、控制能力**，是自动驾驶系统的一个**最小子集**。

以下，ENJOY

在搭建完自动驾驶车辆的软、硬件环境以后，通常采用循迹测试进行验证，如图1所示。

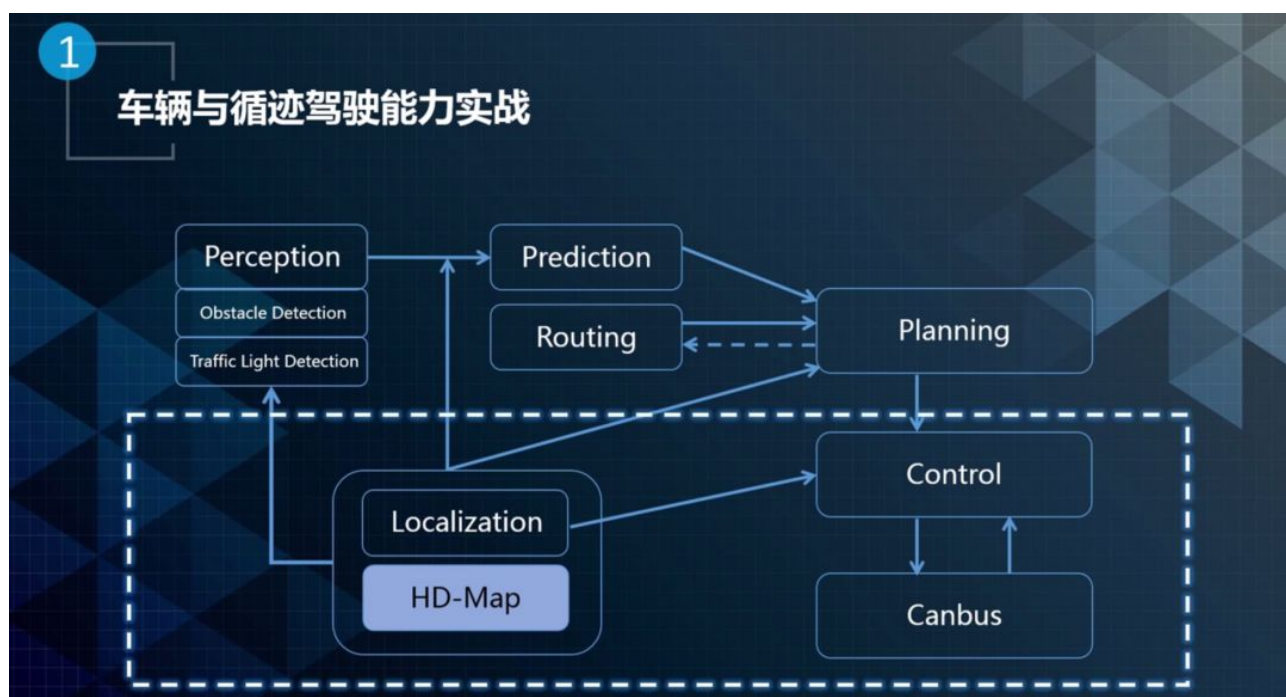


图1 循迹测试涉及的Apollo模块

循迹测试涉及最底下的几个模块，只需要定位、控制以及Canbus这三个模块，是Apollo的最小子集，通过循迹可以验证车的线控能力以及模块的整体集成能力。

那么如何做循迹测试呢？首先在硬件上，我们需要一辆线控车辆、一个工控机以及惯导系统GPS和IMU，如图2所示。

1

Apollo1.0循迹参考车辆和硬件



Lincoln MKZ Hybrid

- 电力支持
- 制动控制
- 动力控制
- 转向控制
- 信息交互
- 设备结构



Nuvo-5095GC

- CPU计算
- GPU计算
- CAN卡
- 串口/USB
- 网络接口
- 交互设备



NovAtel SPAN-Igm

- GNSS
- IMU
- 时钟同步

图2 Apollo循迹测试硬件需求

如果大家使用的是参考硬件搭建的车辆，不需要进行适配，可以直接进行验证。

如果你不是用参考车辆来做这件事，需要做以下几步：

首先是**要实现一个适配层**。通过代码里的Vehicle模块添加一个新Vehicle，其实就是从代码逻辑上添加一个车辆的设备层。具体包括添加一个新车控制器，再实现一个新消息管理器，然后在工厂类中注册这个新车和更新配置文件，如图3所示。

1

Vehicle

- 实现新车控制器
- 实现新消息管理器
- 在工厂类中注册新车
- 更新配置文件:
canbus/conf/canbus_conf.pb.txt

```
/**
 * @class NewVehicleFactory
 *
 * @brief this class is inherited from AbstractVehicleFactory. It can be used to
 * create controller and message manager for lincoln vehicle.
 */
class NewVehicleFactory : public AbstractVehicleFactory {
public:
    /**
     * @brief destructor
     */
    virtual ~NewVehicleFactory() = default;

    /**
     * @brief create lincoln vehicle controller
     * @returns a unique_ptr that points to the created controller
     */
    std::unique_ptr<VehicleController> CreateVehicleController() override;

    /**
     * @brief create lincoln message manager
     * @returns a unique_ptr that points to the created message manager
     */
    std::unique_ptr<MessageManager> CreateMessageManager() override;
};
```

https://github.com/ApolloAuto/apollo/blob/master/docs/howto/how_to_add_a_new_vehicle.md

图3 添加一个新的Vehicle

第二步是**Can卡的管理**。Canbus通过CAN Card硬件去完成沟通，默认有一个叫ESD的CAN Card。如果我们新添不同的厂家，且它的驱动数据格式不一样，需要按照以下方式去集成一个新CAN Card。

1 CAN Card

- 实现新CAN卡类 `CanClient`
- 在工厂类 `CanClientFactory` 中注册新CAN卡
- 更新配置文件: canbus/proto/can_card_parameter.proto

```
/**
 * @brief Send messages
 * @param frames The messages to send.
 * @param frame_num The amount of messages to send.
 * @return The status of the sending action which is defined by
 *         apollo::common::ErrorCode.
 */
apollo::common::ErrorCode Send(const std::vector<CanFrame>& frames,
                               int32_t* const frame_num) override;

/**
 * @brief Receive messages
 * @param frames The messages to receive.
 * @param frame_num The amount of messages to receive.
 * @return The status of the receiving action which is defined by
 *         apollo::common::ErrorCode.
 */
apollo::common::ErrorCode Receive(std::vector<CanFrame>* const frames,
                                  int32_t* const frame_num) override;

/**
 * @brief Get the error string.
 * @param status The status to get the error string.
 */
std::string GetErrorString(const int32_t status) override;
```

图4 Can卡的集成方法

最后是**控制模块**。控制模块是一个非常开放的模块，大部分情况下，不需要定制。当然如果整个车辆的控制，底层的车辆有很大的差别，在个别情况下需要用自己的控制算法。新增控制算法的流程如图5所示。

1 Control Algorithm

- 创建一个控制器
- 在`control_config`文件，为新控制器添加配置
- 注册新控制器

```
namespace apollo {
namespace control {

class NewController : public Controller {
public:
    NewController();
    virtual ~NewController();
    Status Init(const ControlConf* control_conf) override;
    Status ComputeControlCommand(
        const localization::LocalizationEstimate* localization,
        const canbus::Chassis* chassis, const planning::ADCTrajectory* trajectory,
        ControlCommand* cmd) override;
    Status Reset() override;
    void Stop() override;
    std::string Name() const override;
};

} // namespace control
} // namespace apollo
```

图5 新增控制算法的流程

接下来就是定位，它是非常核心的一部分，图6是目前定位整体的逻辑框架。

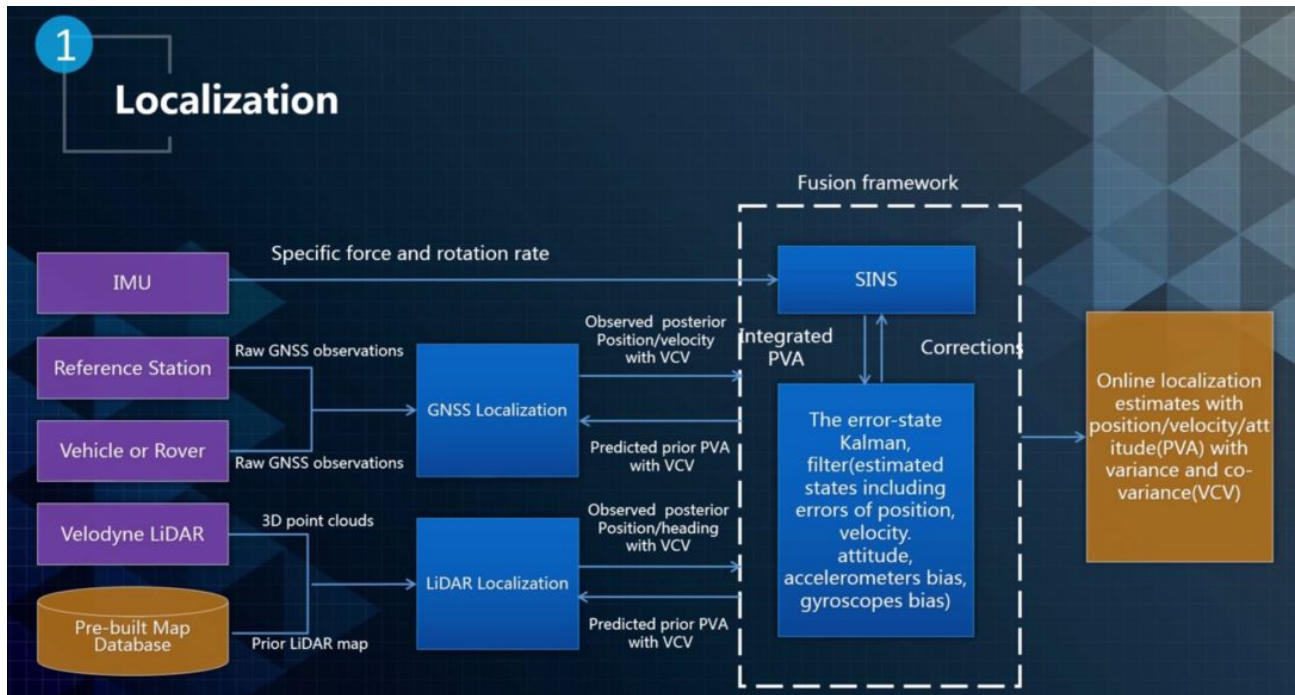


图6 Apollo中定位模块的整体逻辑

目前Apollo提供两种定位方式，一种是**RTK**定位方式，基于基站的方式，需要GPS和IMU惯导。另一种方案是**MSF（多传感器融合）**的定位方式，除了GPS和IMU之外，还通过Lidar的3D点云来做认证匹配定位。循迹测试的定位只需要一个GPS即可。如果使用的不是Apollo推荐的参考硬件，就需要新添加，具体过程就是新建一个GPS解析类去解析GPS的数据格式，然后再配置就可以实现。

设备搭建完成之后，可以通过HMI界面启动循迹测试，如图7所示，包括录制和执行两步。在录制之前，需要确认已经启动了所有相关的GPS，CAN Card模块。

1

启动循迹自动驾驶

- 录制

- 在Quick Record下，单击Setup以启动所有模块并执行硬件运行状况检查。
- 如果硬件健康检查通过，单击 Start 按钮开始记录驱动程序轨迹。
- 到达目的地后，点击Stop 按钮停止录制。

- 执行

- 在Quick Play下，单击 Setup 启动所有模块并执行硬件运行状况检查。
- 确保驾驶员准备好了！ 点击 Start按钮开始自动驾驶。
- 到达目的地后，点击 Stop 按钮停止重放录制的轨迹。

图7 启动循迹自动驾驶的步骤

