Apollo开发套件入门手册

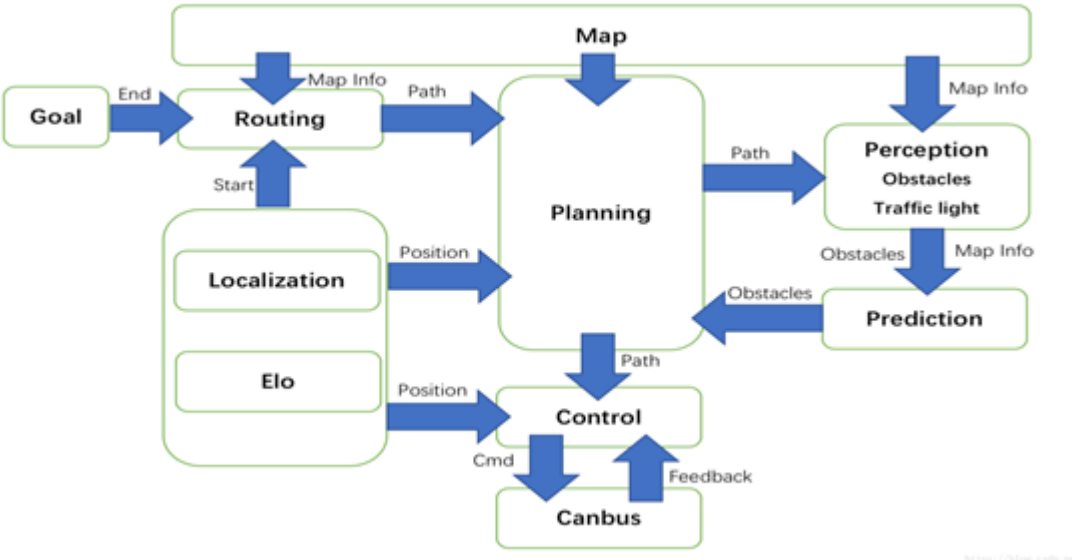
1. Apollo开发平台构成

1.1 平台总体概述

阿波罗是百度面向汽车行业及自动驾驶领域合作伙伴发布的，一个开放、完整、安全的自动驾驶平台。通过这个平台，开发者可整合车辆和硬件系统，搭建完整的自动驾驶系统。阿波罗平台提供的软硬件和服务，包括车辆平台、硬件平台、软件平台、云端数据服务等四大部分。 阿波罗目前已经开放Apollo1.0封闭场景循迹自动驾驶、Apollo1.5昼夜定车道自动驾驶、Apollo2.0简单城市道路自动驾驶、Apollo2.5限定区域视觉高速自动驾驶、Apollo3.0量产园区自动驾驶和Apollo3.5带来的Apollo Enterprise五个版本，完成平台四大模块全部开源，提供更多场景、更低成本、更高性能的能力支持，为自动驾驶车辆量产提供软件、硬件、安全、多模人机交互方面的全面平台服务支持。

1.1.1 系统架构及组成

Apollo软件架构由以下几个模块组成：车辆CAN总线控制模块、控制模块、决策模块、可视化模块、驱动模块、人机交互模块、定位模块、监控模块、感知模块、预测模块以及通用监控与可视化工具，其关系如图3.16所示。自动驾驶系统先通过起点终点规划出全局路径（routing）；然后在行驶过程中感知（perception）当前环境（识别车辆行人路况标志等），并预测下一步发展；然后将采集到信息传入规划模块（planning），规划车辆行驶轨迹；控制模块（control）将轨迹数据转换成对车辆的控制信号，通过汽车交互模块（canbus）控制汽车。



1.2 硬件构成及介绍

1.2.1 感知系统

1.2.2 定位模块

GPS/IMU用来对车辆进行精确定位. Apollo开发者套件中，采用的是星网宇达Newton-M2组合惯导系统。

星网宇达Newton-M2是转为自动驾驶应用设计的一款低成本车载卫星与惯性组合导航系统。卫星定位采用北斗导航、GPS、GLONASS等多模多频卫星定位系统，能够实现全天候、全球覆盖的高精度定位。在惯性导航方面，Newton-M2内置高精度MEMS陀螺仪与高精度加速度计，使用精确标定技术与多传感器数据融合技术，为车辆提供可靠的、精确的定位、航向、姿态等信息。

1.2.3 计算平台

Apollo开发者套件采用宸曜科技Nuvo-6108GC工控机.

Nuvo-6108GC是一款支持GPU高端显卡的工业级宽温型车载工控机，可搭载32GB ECC/非ECC DDR4内存的Celeron赛扬®E3 V5或第六代Intel因特尔®SkyLake酷睿™ i7/i5中央处理器. 该工控机还支持250W NVIDIA® GPU，来为人工智能、虚拟现实和自动驾驶提供最佳解决方案.

Nuvo-6108GC集成了通用计算机的I/O，包括千兆以太网、USB3.0通用串行总线与串行通讯端口. 此外，除了用于安装GPU的x16 PCIe总线端口之外，还提供两个x8 PCIe插槽，方便对工控机进行扩展，可用来为工控机添加CAN卡.

Nuvo-6108GC配备了复杂的电源设计，可处理250W的重功耗和电源瞬态. 此外，为了保证在工业环境中有着可靠的GPU性能，Nuvo-6108GC工控机采用了特别设计的散热系统，可以控制冷气进气的风流，有效散发GPU产生的热能. Nuvo-6108GC工控机散热系统能够保证在60°C的环境下GPU仍然有100%的加载能力.

接口	
以太网	1 x intel I219-LM千兆网卡端口
	1 x intel I219-IT千兆网卡端口
图像输出口	2 x DVI-D 端口，最高支持1920x1080分辨率
串行通讯端口	2 x 软件可编程RS232/422/485串口
USB	4 x USB3.0端口
音频	1 x 喇叭

扩展总线接口	
PCIe接口	1 x PCIe x16插槽@Gen3, 16-lanes PCIe信号，为提供GPU接口
	1 x PCTe x8插槽@Gen3, 4-lanes PCIe信号
M.2接口	1 x M.2 B Key插槽，用于3G/4G SIM卡
mini-PCIe接口	1 x 全尺寸mini PCIe总线插槽
远程控制 & 状态输出	1 x 2x6-pin 2.0mm pin-header端子接口，用来远程开启和状态LED显示

存储器接口	
SATA	4 x SATA插槽，用于2.5" HDD/SSD硬盘安装，支持RAID 0/1/5/10

电源接口	
DC输入	1 x 3-pin可拔插端子，用于24V DC输入
远程控制 & 状态输出	1 x 3-pin可拔插端子，用于远程控制

工作环境	
工作温度(CPU/GPU全功率运行)	-25℃ ~ 60℃
存储器工作温度	-40℃ ~ 85℃*
湿度	10% ~ 90%, 非凝露
震动	运行, 1Grms, 5-500Hz, 3轴(采用GPU, fan和HDD硬盘)，参考IEC60068-2-64
EMC	CE/FCC Class A, 参考EN 55022, EN 55024 & EN 55032

> - 在0℃下运行，需要采用宽温HDD或SSD


DVI/HDMI
COM2
COM1
V+ GND
GPU


DVI/HDMI
COM3
V+ GND
GPU

更多有关Nuvo-6108GC产品的信息请登录https://www.neousys-tech.com.cn/cn/product/application/rugged-embedded/nuvo-6108gc-gpu-computing

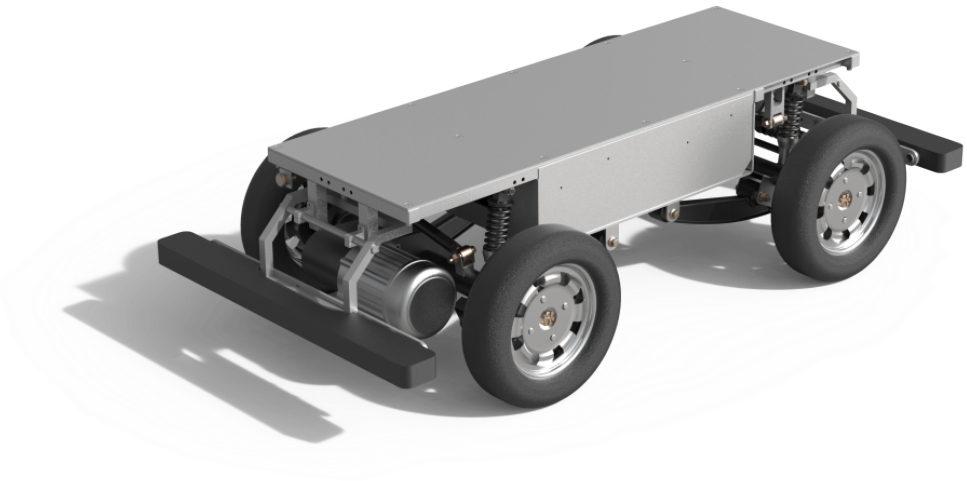
在自动驾驶中，常常需要大量的图像处理与人工智能计算，因此强烈建议在工控机中选配一块高性能NVIDIA®GPU.

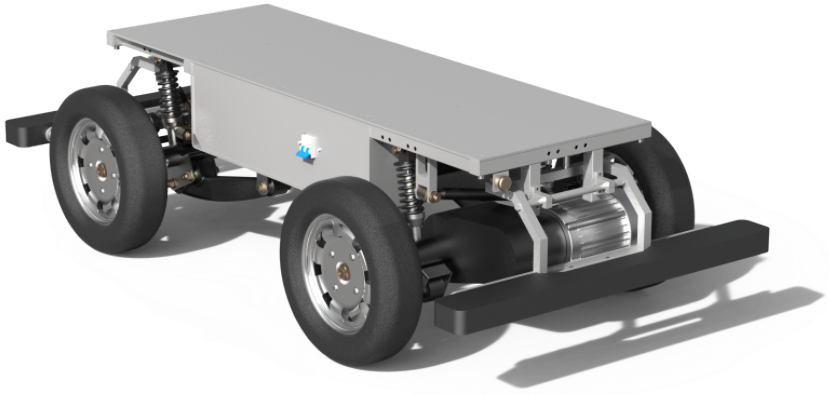
综合成本、性能与稳定性考虑，我们推荐使用NVIDIA® GEFORCE GTX1080系列显卡.

1.2.4 底盘

车辆底盘部分由电池组、传动系统、转向系统组成。底盘结构为桁架式+一体式设备舱，悬架形式采用了整体桥+拖曳臂的形式。最大制动减速度为0.66G。电池组容量为1.7kWh，在综合工况下续航里程为60km，续航时间为6小时.

小车的底盘模型如图所示：





底盘控制器接口支持CAN/RS485/RS232，能够提供车速反馈、线控驱动，完美适配Apollo平台.

1.3 Apollo软件平台介绍

1.3.1 软件平台架构

1.3.2 模块功能

2. Apollo开发平台配置与标定

2.1 底盘与供电配置

2.1.2 遥控器配置

(1) 遥控器简介

Apollo开发者套件在非自动驾驶状态下需要通过遥控器进行控制。遥控器是Apollo开发者套件中的重要部分。由于本套件转为自动驾驶技术而设计，整机无方向盘、制动踏板、油门踏板等执行机构，因此在非自动驾驶状态下，需要通过遥控器来控制车辆，实现车辆的行进、转向和制动，以及自动数据采集、录制数据包等非自动驾驶模式下的操作。

Apollo开发者套件使用RadioLink AT9S遥控器。本遥控器有一个电源开关，8个控制开关，编号为A~H，A~D在正面，拨向操作员一侧是控制开关的**关闭**位置，E~H在天线所在的面上，拨向地面是控制开关的**关闭**位置。如图所示：


急停
电琐
急停
AT9S
RadioLink
POWER
左右
上电
前后
FHSS
DSSS
HYBRID DUAL SPREAD SPECTRUM

遥控器左右摇杆分别控制主机的转向与前进。

(2)遥控器功能介绍

以下是遥控器的控制键功能介绍：

编号	功能	
A	急停	
D	急停	A、D为两个串联急停开关，分别位于遥控器操作面的左上角和右上角，开启这两个急停开关中的任意一个都将触发急停。当触发急停时，底盘前进后退控制无效。有且只有这两个急停开关同时处于关闭位置时，才会释放刹车，使操作人员可以控制车辆
B	电锁开关	电锁开启时底盘才接收遥控器的控制
左侧摇杆	转向控制	左右拨动控制方向，摇杆松开后，底盘方向自动回正
右侧摇杆	行进控制	上下拨动方向盘控制前进和后退，摇杆松开后，电机停止工作

(3)开启和关闭

出于安全考虑，请在使用遥控器之前确保开关均处于关闭状态！

开启遥控器并控制车辆使用如下的步骤：

1. 向上推动电源键，给遥控器上电
2. 向前拨动B开关，开启电锁

在停车后，通过以下步骤来关闭遥控器：

1. 向后拨动B开关，关闭电锁
2. 向下推动电源键，关闭遥控器

> **ATTENTION** 主机底盘具有检测遥控器信号是否失联的自我保护功能，当底盘发现遥控器在没有关闭电锁的情况下失去信号，底盘会自动刹车

> **ATTENTION** 如果关闭遥控器电源先于关闭电锁，会让底盘误以为是遥控器信号失联，因此会触发自动刹车

> **ATTENTION** 线控底盘采用油压碟刹，长期停车时应避免处于制动状态

> **ATTENTION** 在关闭遥控器之前，确保所有控制键处于关闭状态

(4)遥控器安全机制

在进入自动驾驶模式前，请**务必**提前熟悉遥控器的各项操作，尤其关于急停功能的使用，请在每次进入自动驾驶模式前进行遥控操作确认。我们已在工控机与ECU协议层定义了遥控器的绝对优先权，即平台移动的任何状态下（请在每次运行前确保遥控器电量充足），只要遥控器上电且电锁推起，平台即进入人工接管模式。

(5)控制器附加功能

2.2 Apollo平台配置

2.2.1 Linux系统安装

Apollo平台基于Linux系统，因此，我们需要在工控机上安装Ubuntu16.04系统。

(1) 创建引导盘

准备Ubuntu iso文件。在其他计算机上进入Ubuntu官网下载桌面版iso镜像文件至本地，这里推荐使用Ubuntu16.04版本。

下载软件制作U盘启动盘。可以用来制作启动盘的软件有很多，这里以常用的UltraISO为例介绍如何制作启动盘。在下图所示UltraISO软件界面，依次点击“文件”、“打开”，选择下载好的Ubuntu16.04 iso文件，再点击“启动”、“写入硬盘镜像”、“写入”即可。“写入”操作会先格式化U盘，建议先保存其中的文件，几分钟后刻录结束，U盘启动盘制作成功。

(2) 安装Ubuntu系统

将U盘插入工控机，按照以下步骤启动工控机：

1. 将IPC开机后按F2进入BIOS设置菜单，建议禁用BIOS中的快速启动和静默启动，以便捕捉引导启动过程中的问题。
2. 进入选择Boot，设置电脑从U盘启动，之后系统进入Ubuntu安装界面。如下图所示：

之后会出现系统分区界面，由于在IPC只需安装Ubuntu系统，故用户可按系统默认分区，也可自定义分区大小。分区结束后点击“继续”按照屏幕提示完成系统安装。 3. 安装完成后，重启进入Ubuntu16.04系统。执行软件更新器(Software Updater)更新最新软件包，或在终端执行以下命令完成更新：

```
sudo apt-get update
sudo apt-get upgrade
```

4. 打开终端，输入以下命令，安装Linux 4.4 内核: sudo apt-get install linux-generic-lts-xenial IPC必须接入网络以便更新与安装软件，所以请确认网线插入并连接，如果连接网络没有使用动态分配（DHCP），需要更改网络配置。

获取更多Ubuntu信息，可访问Ubuntu桌面站点: https://www.ubuntu.com/desktop

2.2.2 Apollo安装

(1) Apollo内核

为了适应自动驾驶任务和特定的硬件设备，我们需要对Linux内核进行一些改造。Apollo团队发布了一个适用于自动驾驶任务的内核包。

Apollo内核包基于Linux 4.4.32版本内核，为Apollo软件平台的运行提供了必要的内核级支持。相比原生内核，Apollo内核包增加以下布丁：

1. Realtime Patch(为Linux提供实时性的特性)
2. 最新e1000e intel以太网卡驱动
3. 一些cve安全补丁

此外Apollo内核：

- 修复了Nvidia显卡驱动在Realtime补丁下的Bug
- 修复了inet_csk_clone_lock的重复释放的Bug

Apollo内核开源在ApolloAuto/apollo-kernel中，开发者可在GitHub网站上查看(https://github.com/ApolloAuto/apollo-kernel)。另外，Apollo还打包了已经编译好的内核，开发者可前往以下网址下载：

```
https://github.com/ApolloAuto/apollo-kernel/releases
```

下载之后，在命令行终端中使用以下语句来安装内核：

```
tar zxvf linux-4.4.32-apollo-1.0.0.tar.gz
cd install
sudo ./install_kernel.sh
```

开发者也可克隆GitHub中的源码，自己编译内核安装包:

```
git clone https://github.com/ApolloAuto/apollo-kernel.git
cd linux
./build.sh
```

编译完成后，生成一个.tar.gz压缩包，在linux/install/rt文件夹下。将压缩包解压，运行./install_kernel.sh脚本进行安装。

> **ATTENTION** 我们建议配合下一节中的内容，将ESDCAN卡驱动添加高Apollo Kernel源码中，编译一份新的安装包，以方便移植。

(2)ESD-CAN卡驱动安装

CAN卡是工控机与底盘之间通信的必要组件。Apollo采用ESDCAN卡实现上下通信。原生Ubuntu系统与Apollo kernel中都没有ESDCAN卡驱动，需要将该驱动安装到内核中。

ESDCAN卡驱动可从ApolloAuto/apollo-contrib仓库的分支中找到。

安装ESDCAN卡有两种方法，一种是将ESDCAN卡驱动与Apollo内核直接编译为一个安装包(推荐)，另一种是在已经安装好的内核中添加ESDCAN驱动。

我们先下载一份Linux 4.4.32的内核：

```
https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.32.tar.gz
```

同时拉取一份Apollo Kernel的源码文件并进入其子文件夹中：

```
git clone https://github.com/ApolloAuto/apollo-kernel.git
cd linux
```

解压Linux 4.4.32内核压缩包，将解压后linux-4.4.32文件下的所有内容复制到apollo-kernel/linux/ 文件夹下。之后，返回apollo-kernel/linux/ 文件夹下，运行:

```
./build.sh
```

当界面中出现以下内容时，程序的运行:

```
patching file kernel/locking/rtmutex.c
patching file Makefile
patching file scripts/setlocalversion
patching file drivers/net/ethernet/intel/e1000e/defines.h
patching file drivers/net/ethernet/intel/e1000e/e1000.h
patching file drivers/net/ethernet/intel/e1000e/hw.h
patching file drivers/net/ethernet/intel/e1000e/ich8lan.c
```

```
patching file drivers/net/ethernet/intel/e1000e/netdev.c
patching file drivers/net/ethernet/intel/e1000e/ptp.c
patching file drivers/net/ethernet/intel/e1000e/regs.h
patching file net/ipv4/inet_connection_sock.c
patching file net/ipv4/tcp_minisocks.c
patching file crypto/ahash.c
patching file drivers/gpu/drm/vmwgfx/vmwgfx_surface.c
patching file drivers/scsi/sg.c
patching file include/crypto/internal/hash.h
patching file net/dccp/input.c
patching file net/packet/af_packet.c
patching file net/sctp/socket.c
patching file net/xfrm/xfrm_user.c
============================
[ OK ] build Non-RT kernel
[INFO] Took .048 seconds
============================
To support ESD CAN, ESD CAN driver supplied by ESD Electronics is required
, but not found.
Please refer to ESDCAN-README.md for more information.
Build will continue after 6 seconds, but ESD CAN support will not be built-
in;
type ctrl+c to interrupt if that's not what you want.
```

由于没有ESDCAN驱动的文件，因此此时无法编译ESDCAN驱动。此时，将apollo-contrib/esd/src文件夹下除Makefile之外的所有文件复制到apollo-kernel/linux/drivers/esdcan中，在该目录下运行以下命令：

```
rm Makefile Kconfig
ln -s Makefile.esd Makefile
ln -s Kconfig.esd Kconfig
```

回到apollo-kernel/linux文件夹下，执行脚本：

```
./build.sh
```

等待编译完成。

编译完成之后，会在apollo-kernel/linux/install/rt/ 文件夹下产生一个.tar.gz压缩文件包。将该压缩包解压，在解压后的目录中执行：

```
cd linux
sudo ./install_kernel.sh
```

程序运行结束之后，Apollo内核就已经安装完毕。

(3) Apollo安装

打开Ubuntu16.04的终端，按照以下步骤安装Apollo：

安装docker环境

1. 更新apt安装包

```
sudo apt-get update
```

2. 对下列安装包进行安装

```
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
```

3. 添加docker官方GPG密钥：

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
```

4. 验证是否成功添加密钥：

```
sudo apt-key fingerprint 0EBFCD88
```

5. 对稳定版repository进行设置

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

6. 更新apt安装包：

```
sudo apt-get update
```

7. 安装最新版docker：

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

或者指定特定版本docker进行安装：

```
sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=
<VERSION_STRING> containerd.io
```

8. 验证是否成功安装docker：

```
sudo docker run hello-world
```

以上步骤参考网址为：https://docs.docker.com/install/linux/docker-ce/ubuntu/

安装Git LFS

在终端中，执行以下语句，安装Git LFS:

```
curl -s https://packagecloud.io/install/repositories/github/git-
lfs/script.deb.sh | sudo bash
sudo apt-get install -y git-lfs
```

下载Apollo代码

在终端中，执行以下语句，从GitHub上拉取3.0版本的Apollo:

```
git clone -b r3.0.0 https://github.com/ApolloAuto/apollo.git
```

执行完毕之后，进入下载好的apollo文件夹中，开始安装Apollo.

编译Apollo

按照如下的步骤编译和安装Apollo:

1. 启动容器

```
bash docker/scripts/dev_start.sh
```

2. 进入容器

```
bash docker/scripts/dev_into.sh
```

3. 如果用户电脑中没有安装GPU，则运行下述命令：

```
bash apollo.sh build_cpu
```

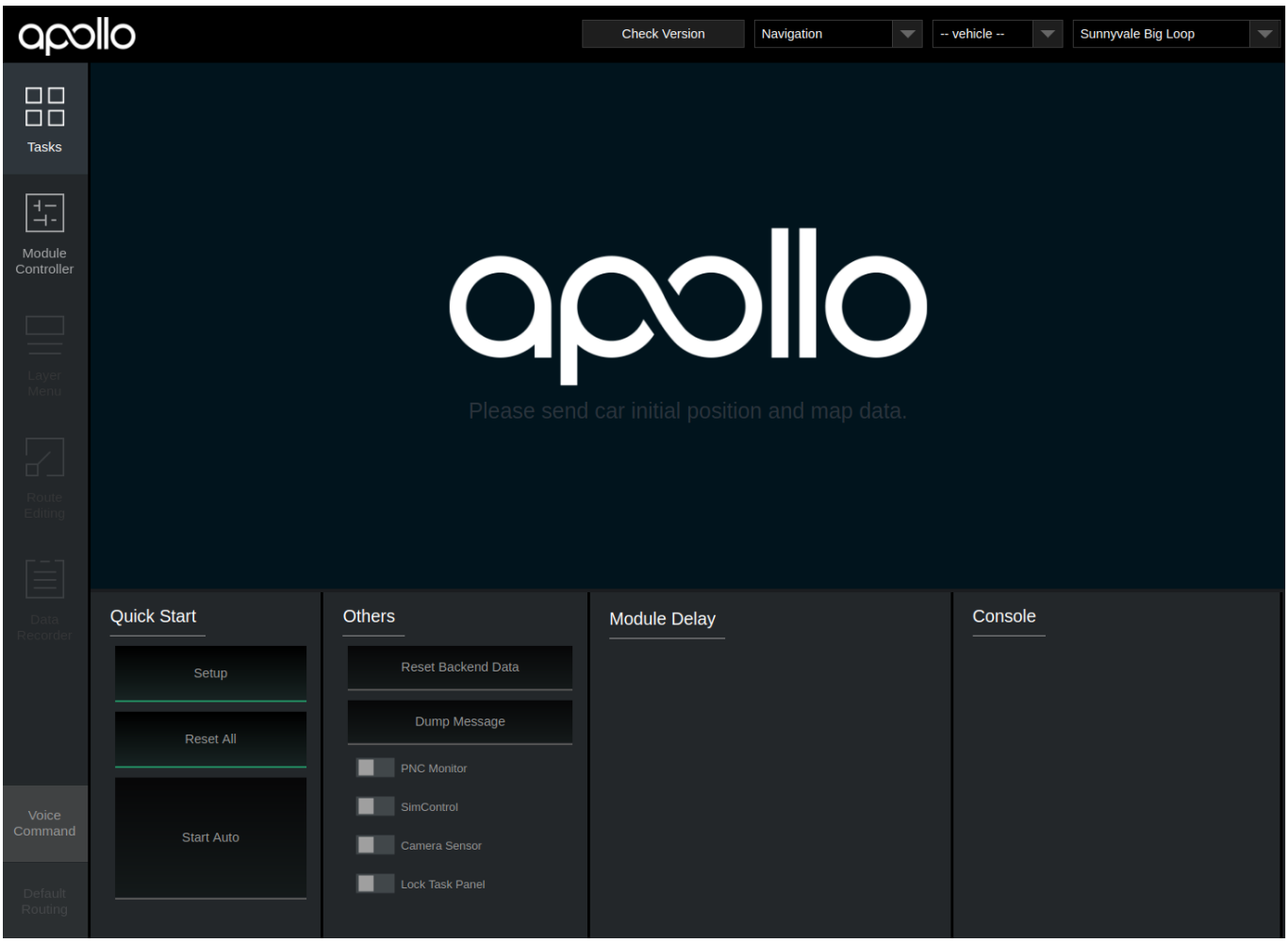
由于配置的IPC包含GPU，用户可直接运行下述命令：

```
bash apollo.sh build
```

4. 编译完成后，启动apollo

```
bash scripts/bootstrap.sh
```

5. 在浏览器中输入网址 http://localhost:8888 查看dreamview页面如下：



2.3 感知系统配置

2.3.1 激光雷达安装与配置(没有激光雷达)

（1）安装与配置步骤（2）效果验证

2.3.2 摄像头安装与配置(没有摄像头)

（1）安装与配置步骤（2）效果验证 …

2.4 传感器联合标定(没有传感器联合标定)

2.4.1 激光雷达和摄像头联合标定

（1）标定基本原理（2）标定过程（3）效果验证

2.4.2 摄像头和毫米波雷达联合标定

……

2.5 油门刹车标定

2.5.1 标定基本原理

在Apollo系统中,控制模块会请求加速度量值。通过车辆标定表,控制模块便能找到准确产生所需加速度量值对应的油门、刹车踏板开合度控制命令,之后下发给车辆底盘。车辆标定表提供一个描述车辆速度、油门/刹车踏板开合度、加速度量之间关系的映射表。油门刹车标定过程便是生成车辆标定表的过程。

Apollo系统为教学小车提供了一份默认的标定表。如用户期望自己重新标定车辆,可以参考以下车辆标定流程说明。

2.5.2 标定步骤

油门刹车标定之前，需要进行以下的准备工作:

- 改变驾驶模式
- 选择测试地点

(1) 改变驾驶模式

在modules/canbus/conf/canbus_conf.pb.txt中,设置驾驶模式为AUTO_SPEED_ONLY.

(2) 选择测试地点

理想的测试地点是平坦的长直路，因此，选择一段开阔的长直路来进行标定.

以上准备工作完成后, 在modules/tools/calibration文件夹中按顺序完成如下工作

- 采集数据
- 处理数据
- 绘制结果
- 转换结果为Protobuf格式

(1)采集数据

1. 运行 modules/tools/calibration/ 下的python data_collector.py，之后输入参数x y z. 其中，x代表加速踏板开合度(百分比正值), y代表了速度限值(米/秒), z代表刹车踏板开合度(百分比负值). 输入参数后,车辆即开始以x加速踏板值加速至y速度限值,之后再以z刹车踏板值减速直至车辆停止。
2. 根据车辆反应情况，调整命令参数。如加速踏板值过小无法启动车辆时，增加加速踏板开度。
3. 对应产生对应x y z 参数的csv文件。比如输出指令15 5.2 -10 ,将会生成名为 t15b-10r0.csv 的文件。

对x,y,z，我们建议建议:

- x: 过低可能无法启动,通常20以上。但是不能过大否则安全风险
- y: 单位米/秒,根据车型和实际试验确定
- z: 刹车踏板,-35已经是急刹车

对于x y z参数的组合，我们建议:

- 一般先固定速度限值y,之后固定加速踏板命令x,并尝试采集不同减速指令z: (22 2 -23) (22 2 -25) (22 2 -27)(22 2 -29)
- 之后再逐一更改加速踏板命令x: (24 2 -23) (24 2 -25) (24 2 -27)(24 2 -29) (26 2 -23) (26 2 -25) (26 2 -27)(26 2 -29)
- 之后再尝试更改速度限值y

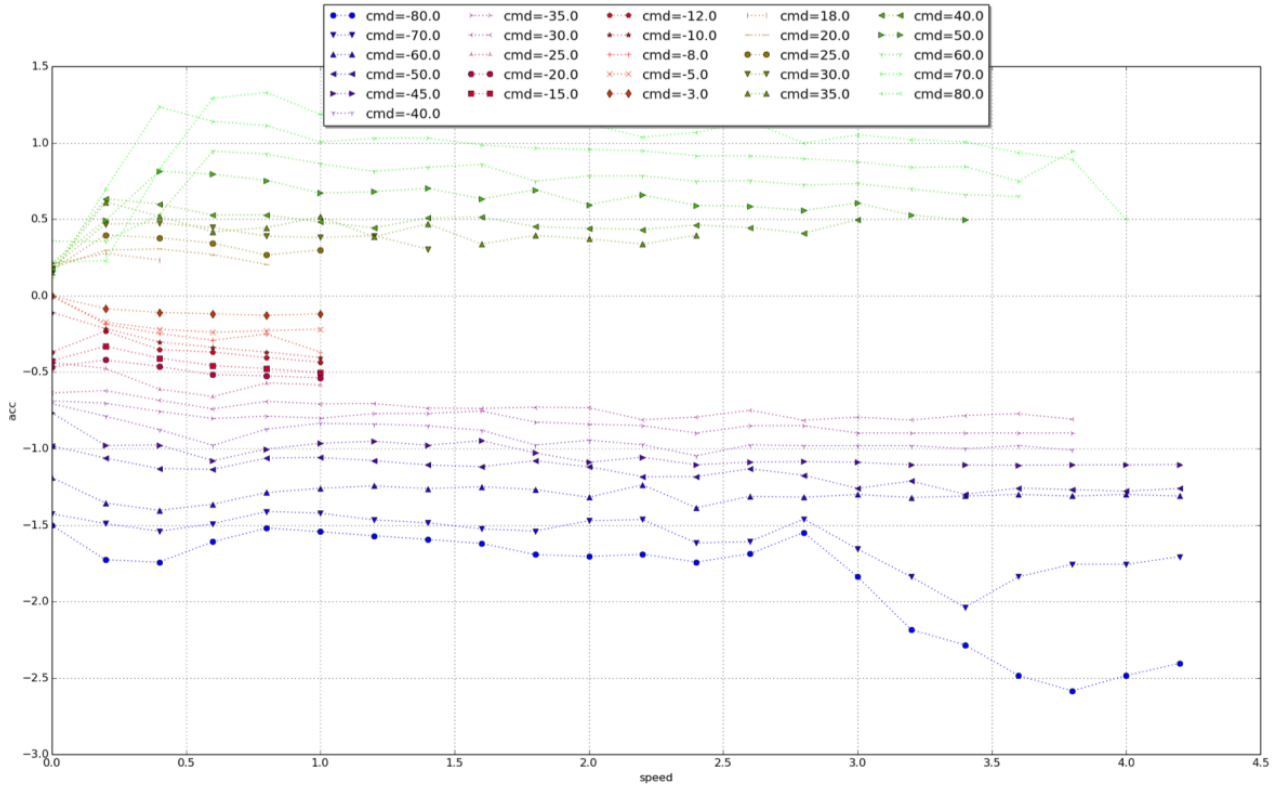
(2) 处理数据

对每个记录的日志分别运行 process_data.sh {dir}，其中dir为t15b-10r0.csv所在的目录。每个数据都被处理成 t15b-10r0.csv.result 。

(3) 绘制结果

通过运行python plot_results.py t15b-10r0.csv得到可视化最终结果,检查是否有异常。

正确的标定结果应如下图所示：



(4) 转换结果为 Protobuf 格式

运行 result2pb.sh，把校准结果result.csv转换成控制模块定义的 Protobuf 格式。运行命令后生成control_conf_pb.txt文件。该文件里面的calibration_table段落是我们需要的,我们将该calibration_table段替换放到文件/modules/control/conf/lincoln.pb.txt下对应的alibration_table段。

3. 运行循迹例程(暂时做不到)

3.1 循迹例程介绍

3.1.1 Apollo基本运行模式介绍

3.1.2 循迹例程算法介绍

3.2 启动Localization模块

3.2.1 定位准确度效果验证

3.2.2 常见问题调试

3.3 启动Perception模块

3.3.1 感知结果输出验证

3.3.2 常见问题调试

3.4 启动Planning模块

3.4.1 规划路径输出验证

3.4.2 常见问题调试

3.5 启动循迹例程

3.5.1 Dreamview界面配置

3.5.2 录制轨迹数据

（1）回放数据包 （2）循迹例程输出效果（Control模块）调试

4. 注意事项与常见问题

4.1 开发平台操作注意事项

4.2 开发平台调试常见问题