- 卫星天线（2个）：测量型卫星天线，信号接口为TNC母口

- 射频连接线（2根）：射频线两端分别为TNC公头和SMA母头

- 数据/电源线缆（1根）：一端是和主机相连的接口，另一端含有一个串口母头，一根RJ-45网线母头，一根电源接口和一个USB接口

挂载

使用者可以将主机安装在载体上的任何位置，但我们强烈建议使用者采用下述的建议方案：

- 将M2主机安装在载体的后轴的中心位置上，使主机铭牌上标示的坐标系XOY面尽量与载体坐标系平行并保持各

轴向一致，建议将Y轴的正向保持与载体的前进方向一致，M2主机单元必须与被测载体固连。

- GNSS双天线应尽量与载体坐标系Y轴平行并且前天线（Secondary）应在Y轴正方向上，GNSS天线要尽可能的将其安置于测试载体的最高处以保证能够接收到良好的GNSS信号。

配线

执行如下步骤将主机和天线连接到Apollo系统：

- 将两根射频线的TNC公头连接上卫星天线的TNC母口
- 将射频线的SMA母口连接上主机的SMA公口
- 将数据/电源线缆的公口和主机的母口连接
- 将数据/电源线缆的串口母头连接上IPC的串口公口
- 将数据/电源线缆的网线母头和有线网的水晶头相连接
- 将数据/电源线缆的USB接口连接上IPC的USB接口
- 将数据/电源线缆的电源线接口和载体的电源接口连接起来

请参考下述图片：

量取杆臂值

参考下述步骤：

- 当主机和天线处在正确位置时量取天线到主机的距离。主机的中点和天线的中点标记在设备的外部。
- 距离被测量并记录为X轴偏移、Y轴偏移和Z轴偏移。坐标轴由主机的位置确定。偏移量的误差应该被控制在一厘米以内以获取高精度的定位信息。

配置GPS和主机

下面展现了配置GPS和主机的方法。当设备正确接入系统后，在/dev/下面有名为ttyACM0的设备，即表示M2已经被正确的加载了。配置设备时，需要将设备的串口线连接上电脑的串口才可以对设备进行配置，也就是说，用来配置设备的电脑主机需要拥有串口。Windows下可以通过串口助手、串口猎人或者COMCenter等工具进行配置，Linux下可以通过Minicom、Cutecom等工具进行配置。

杆臂配置

车尾天线（后天线，通常是主天线，也就是Primary）杆臂配置：

```
$cmd,set,leverarm,gnss,0,0.359,0.248*ff
```

杆臂值请以自己使用的实际情况为主

GNSS航向配置

天线车头车尾前后安装

```
$cmd,set,headoffset,0*ff
```

导航模式配置

```
$cmd,set,navmode,FineAlign,off*ff
$cmd,set,navmode,coarsealign,off*ff
$cmd,set,navmode,dynamicalign,on*ff
$cmd,set,navmode,gnss,double*ff
$cmd,set,navmode,carmode,on*ff
$cmd,set,navmode,zupt,on*ff
$cmd,set,navmode,firmwareindec,0*ff
```

USB接口输出设置

```
$cmd,output,usb0,rawimub,0.010*ff
$cmd,output,usb0,inspvab,0.010*ff
$cmd,through,usb0,bestposb,1.000*ff
$cmd,through,usb0,rangeb,1.000*ff
$cmd,through,usb0,gpsephemb,1.000*ff
$cmd,through,usb0,gloephemerisb,1.000*ff
$cmd,through,usb0,bdsephemerisb,1.000*ff
$cmd,through,usb0,headingb,1.000*ff
```

网口配置

```
$cmd,set,localip,192,168,1,2*ff
$cmd,set,localmask,255,255,255,0*ff
$cmd,set,localgate,192,168,1,1*ff
$cmd,set,netipport,111,112,113,114,8000*ff
$cmd,set,netuser,username:password*ff
$cmd,set,mountpoint,XMJL*ff
```

网络配置依据自己的实际情况自行更改为相应的配置。

PPS授时接口输出

```
ppscontrol enable positive 1.0 10000
```

```
log com3 gprmc ontime 1 0.25
```

通过 `$cmd,get,navmode*ff` 指令可以获取设备当前的导航模式配置。将所有配置逐条或一起发送给设备，得到设备返回 `$cmd,config,ok*ff` 字段，说明配置成功，配置成功后要进行配置保存，发送 `$cmd,save,config*ff` 指令。根据设备安装情况，要进行设备坐标轴配置，可以发送： `$cmd,get,coordinate,x,y,z*ff` 指令获取当前坐标轴配置。坐标轴默认配置为 `$cmd,set,coordinate,x,y,z*ff` ，即与设备外壳标注一致。杆臂是指后天线（Primary）的几何中心位置相对于主机几何中心在直角坐标系内x,y,z三方向的位置差。通过如下指令进行补偿设置： `$cmd,set,leverarm,gnss,x_offset,y_offset,z_offset*ff` 。
x_offset:X方向的杆臂误差，单位为米，以此类推。注：上述坐标XYZ为设备坐标轴配置后的实际坐标，一般应与载体坐标系一致，注意补偿的是后天线（Primary）杆臂值。当进行导航模式配置后必须对设备进行重新上电启动。

Apollo代码配置

此适配过程是基于Apollo的v3.1_dev分支的代码进行的，将其clone至本地后编译通过即可。

修改配置

- 修改/apollo/modules/drivers/gnss/proto文件夹下面的配置文件config.proto，将NEWTONM2_TEXT = 30;NEWTONM2_BINARY = 31;添加进message Stream的后面;
- 修改/apollo/modules/drivers/gnss/conf文件夹下面的配置文件gnss_conf.pb.txt，将gnss_conf_newton.pb.txt的内容全部拷贝覆盖gnss_conf.pb.txt的内容即可。

关闭雷达

在apollo/modules/localization/conf/localization.conf文件中将：-enable_lidar_localization=true 修改为：-enable_lidar_localization=false.

修改定位模式

在apollo/modules/localization/conf/localization_config.pb.txt文件中这个配置应为localization_type: MSF，M2不支持RTK模式。

修改脚本

在apollo/scripts/docker_adduser.sh文件中将#setup GPS device下面添加如下代码

```
if [ -e /dev/ttyACM0 ]; then
  chmod a+rw /dev/ttyACM0
fi
```

添加航向

在apollo/modules/localization/conf/msf_adapter.conf文件中添加如下内容：

```
config
{
    type: GNSS_HEADING
    mode: RECEIVE_ONLY
    message_history_limit: 10
}
```

摄像头安装与配置

摄像头型号：LI-USB30-AR023ZWDR（leopard 摄像头）

更多详细参数可参考：https://leopardimaging.com/product/li-usb30-ar023zwdrb/

安装要求：牢固安装在小车结构架前端横梁处，水平安装，俯仰角向下0-2度（向下倾斜小于2度，不能上仰），翻滚角误差±1度（左右两侧的平齐程度），航向角误差±2度，镜头保持清洁，避免影响图像采集。安装位置如下图所示

设备连接说明：直接用数据线将设备连接在IPC的USB3.0接口。安插要牢固，不能有外力影响，避免脱落或损伤。设备的输出口和连接线如下图所示：

配置检查：将摄像头连接到IPC的USB接口后，在terminal下使用cheese命令查看摄像头能否正常采集图像，或者进入/dev 路径查看是否有video*节点。如果可以采集到图像或者可以看到video*节点，摄像头安装成功。
apollo默认对应的摄像头设备为video0。如果摄像头不是video0，执行如下修改操作：

```
$ cd ~/apollo/modules/drivers/usb_cam/launch/

$ vim start_one_leopard.launch
```

修改

```
“video_device” value=”/dev/video0” #将video0改为你对应的video*。

$ vim launch/usb_cam-test.launch
```

修改

```
“video_device” value=”/dev/video0” #将video0改为你对应的video*。

$ vim start_obstacle_camera.launch
```

修改（如果需要）

```
“video_device” value=”/dev/video0” #将video0改为你对应的video*。
```

编译Apollo自带的usb_cam

```
$ cd /apollo

$ bash apollo.sh build_usbcam
```

编译完成后，根据摄像头自身特点，进行设备固化，操作如下：

编辑99-webcam.rules 文件，具体内容如下：

```
SUBSYSTEM==”video4linux”, SUBSYSTEMS==”usb”, ATTR{name}==”AR023ZWDR(Rev[0-9][0-9][0-9]s)”, MODE=”0666”, SYMLINK+=”camera/obstacle”, OWNER=”apollo”, GROUP=”apollo”
SUBSYSTEM==”video4linux”, SUBSYSTEMS==”usb”, ATTR{name}==”AR023ZWDR(Rev[0-9][0-9][0-9])”, MODE=”0666”, SYMLINK+=”camera/trafficlights”, OWNER=”apollo”, GROUP=”apollo”
SUBSYSTEM==”video4linux”, SUBSYSTEMS==”usb”, ATTR{name}==”AR023ZWDR(Rev[0-9][0-9][0-9]F12)”, MODE=”0666”, SYMLINK+=”camera/lanemark”, OWNER=”apollo”, GROUP=”apollo”
```

将文件拷贝到/etc/udev/rules.d/路径下

```
$ sudo cp ~/Downloads/99-webcam.rules /etc/udev/rules.d/

$ sudo service udev restart
```

重新插拔摄像头，在找到/dev/camera/lanemark节点，将video*固化到节点上：

```
sudo ln -s /dev/video0 /dev/camera/lanemark
```

完成固化后，配置perception_lowcost.conf文件进行调试：

```
$ vim modules/perception/conf/perception_lowcost.conf
```

配置dag_config_path:
```
- dag_config_path=conf/dag_camera_obstacle_lane_motion_vis.config
```

```
$ ./apollo.sh build_opt_gpu
```

编译完成后，执行:

```
$ cd /apollo
```

```
$ ./scripts/bootstrap.sh
```

```
$ rostopic list
```

如果出现：/apollo/sensor/camera/obstacle/front_6mm 摄像头配置完成，在dreamview界面可以看到摄像头采集画面。

毫米波雷达安装与配置

毫米波雷达型号：continental AS 408-21

更多详细参数可参考：https://www.conti-engineering.com/en-US/Industrial-Sensors/Industrial-Sensors

安装要求：毫米波雷达要牢靠固定在车身上，连接到毫米波雷达的接头要牢靠接插。离地面高0.5米，不能向下倾斜，向上仰0~2度以内，高度误差±0.2米，俯仰角误差0~2度（向上仰小于2度，不能向下倾斜），翻滚角误差±2度（radar左右两侧的平齐程度），航向角误差±2度（radar是否正对前方），安装位置如下：

设备连接说明：毫米波雷达数据接口与CAN线接口连接。各针脚信号定义如下：

配置检查：将毫米波雷达接入CAN1口，供电后即可收到CAN信息。

激光雷达安装与配置

激光雷达型号：80-VLP-16（velodyne 16线激光雷达）

更多详细参数可参考：https://velodynelidar.com/vlp-16.html

安装要求：16线激光雷达及控制盒要牢靠固定在车身上，连接到控制盒上的接头要牢靠接插。水平安装在车顶部，对地高度1.5米，水平放置，精度在2度以内。连接线缆带屏蔽，接口镀金，可参考京东线缆购买网址：https://item.jd.com/1342004.html 网线两头需贴上标签注明设备名称。安装位置如下图

设备连接说明：数据输出端口通过以太网连接到工控机以太网口。

控制盒接口如图所示：

GPS授时接口改造：裸线对应的PPS信号，GPRMC信号，地，分别压入黑色控制盒PCB板上面对应的三个螺丝下面。信号对应关系和实际连接图如下所示：

配置检查：设备连接完成后，给激光雷达供电，修改激光雷达的默认ip地址，使激光雷达和IPC在一个网段内，velodyne 16线激光雷达的出厂默认IP地址为：192.168.1.201。修改成192.168.20.13即可。

油门刹车标定

油门刹车标定是车辆纵向精准控制的前提。用户可以使用系统预先标定好的参数，也可以按照手册说明重新进行标定。

标定原理介绍

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。

标定流程说明

按如下顺序完成准备工作:
- 改变驾驶模式
- 选择测试地点

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` .

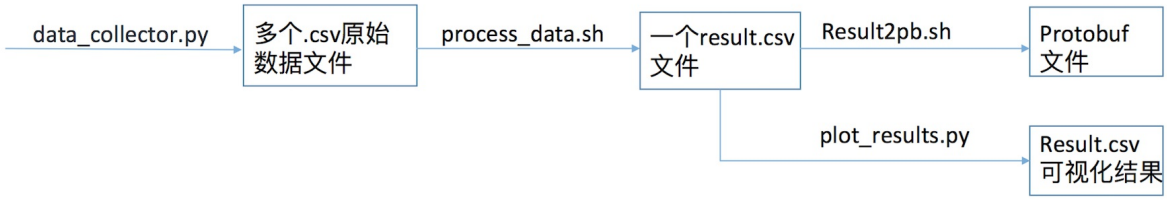
选择测试地点

理想的测试地点是平坦的长直路

以上准备工作完成后，在 `modules/tools/calibration` 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



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，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，如加速踏板过小不能启动或者刹车踏板过小不能停车，需要相应调整命令参数。
3. 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。

A1 | time

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	time	io	ctlmode	ctlbrake	ctlthrottle	ctlgear_loca	vehicle_spe	engine_rpm	driving_mo	throttle_per	brake_perce	gear_locatio	imu
2	1541406731	0	1	0	0	0	0	0	4	15.9472036	14.6059361	0	-0.0021377
3	1541406731	0	1	0	0	0	0	0	4	15.9472036	14.6532383	0	0.00304671
4	1541406731	0	1	0	0	0	0	0	4	15.9594107	14.6532383	0	-0.0012254
5	1541406731	0	1	0	0	0	0	0	4	15.9594107	14.6242466	0	-0.0079573
6	1541406731	0	1	0	0	1	0	0	4	15.9594107	14.6242466	0	-0.0075942
7	1541406731	0	1	0	0	1	0	0	4	15.9227896	14.6791792	0	0.00799229

x, y, z取值建议：
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。

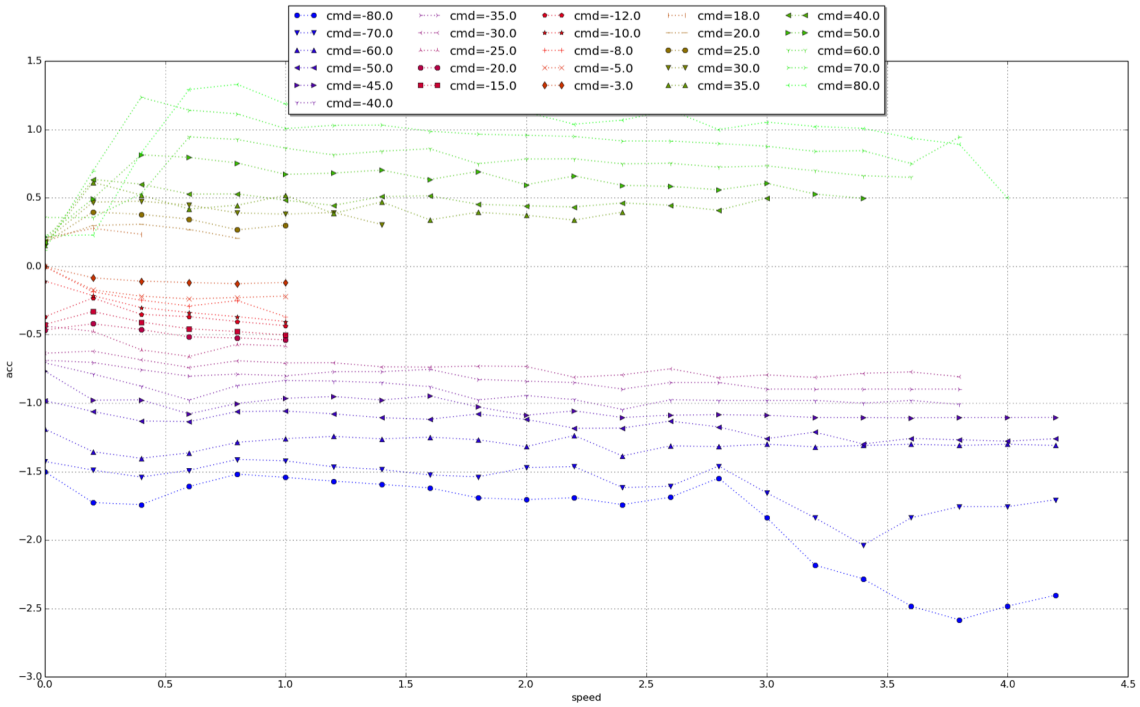
x y z参数组合取值建议：
一般先固定速度限值y，之后固定加速踏板命令x，并尝试采集不同减速指令z：
（22 2 -23） （22 2 -25） （22 2 -27）（22 2 -29）
之后再逐一更改加速踏板命令x：
（24 2 -23） （24 2 -25） （24 2 -27）（24 2 -29）
（26 2 -23） （26 2 -25） （26 2 -27）（26 2 -29）
之后再尝试更改速度限值y。

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。每个数据日志被处理成 `t15b-10r0.csv.result` 。

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。



转换结果为 `Protobuf` 格式

如果一切正常，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calibration_table段。

注：Calibration_table 片段示例

```
calibration_table {
    calibration {
      speed: 0.0
      acceleration: -1.43
      command: -35.0
    }
    calibration {
      speed: 0.0
      acceleration: -1.28
      command: -27.0
    }
    calibration {
      speed: 0.0
      acceleration: -1.17
      command: -25.0
    }
    calibration {
      speed: 0.0
      acceleration: -1.02
      command: -30.0
    }
 }
```

启动循迹

在完成以上软硬件安装，标定以及系统文件配置后，用户可以通过Dreamview界面录制车辆轨迹并回放，完成第一个循迹演示。本文档分成两个主要的部分，系统文件配置和循迹操作说明。

系统文件配置

系统文档配置主要包括两个部分，GNSS配置和车辆选型。

GNSS配置

在/modules/drivers/gnss/conf/gnss_conf.pb.txt文件中，
修改如下一行 +zone=50, （ 北京 ）

```
proj4_text: "+proj=utm +zone=10 +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

修改如下内容配置基站信息

```
rtk_from {
    format: RTCM_V3
    ntrip {
        address: "IP"
        port: 8000
        mount_point: "MOUNTPOINT"
        user: "USERNAME"
        password: "PASSWORD"
        timeout_s: 5
    }
    push_location: true
}
```

此时，配置完成，应该可以运行循迹了。
但是我们在dreamview 中GPS ERROR, 日志报错

```
E1015 11:25:08.626899 6541 file.h:144] Failed to parse file
/apollo/modules/drivers/gnss/conf/gnss_conf.pb.txt as binary proto.
E1015 11:25:08.627151 6541 apollo_app.cc:59] gnss Init failed: DRIVER_ERROR_GNSS: Unable to load
gnss conf file: /apollo/modules/drivers/gnss/conf/gnss_conf.pb.txt
```

从报错字面意义上来看是文件格式问题，把配置文件/apollo/modules/drivers/gnss/conf/gnss_conf.pb.txt尾部的注释去掉以后，成功运行。如下一行以后的信息都删除

```
wheel_parameters: "SETWHEELPARAMETERS 100 1 1\r\n"
```

另外，在程序运行的过程中，有可能会把
modules/calibration/data/vehicle_name/gnss_params/gnss_conf.pb.txt拷贝到
modules/drivers/gnss/conf/gnss_conf.pb.txt，那么我们也需要修改
modules/calibration/data/vehicle_name/gnss_params/gnss_conf.pb.txt里面的基站配置信息和+zone=50才能保证gnss配置正确。

常见问题：
a 系统无法生成驱动设备ttyACM0
/apollo/data/log/gnss.ERROR 里面会有类似报错提示：

```
open device /dev/ttyACM0 failed,  error: no such file or directory
gnss driver connect failed, stream init failed
```

docker 内和docker外的/dev/下都没有ttyACM0设备
解决办法：

退出docker，
关闭docker，

```
cd /apollo/docker/setup_host
bash setup_host.sh
```

重启工控机，
然后在/docker/外，dev/下，就有ttyACM0，
再进docker， 再试gps，可以了。

b MSF修改为RTK
lidar_extrin: …
lidar_height: …
umm zone id:
ant imu lever arm file: ….
以上是localization.INFO的日志，并且 `rostopic echo /apollo/localization/pose` 没有信号，按正常通过命令配置了Newton-M2杆臂值后，杆臂值不应该来自文件，说明有问题，该问题是由于采用MSF方案导致的，MSF方案会采用雷达数据并处理。
方法：
修改modules/localization/conf/localization.conf下

```
--enable_lidar_localization=true   ( 改为false )
```

修改modules/localization/conf/localization_config.pb.txt下

```
localization_type: MSF   ( 改为RTK  )
```

车辆选型

如何让新车辆在dreamview中可以选择（ 即可以通过下拉菜单选择新车型 ），具体操作如下：
查看/apollo/modules/calibration/data/目录下，有一个mkz_example和README.md，那么以酷黑小车ch为例，如果我们希望dreamview中多出一个ch的选项，那么就需要在这个目录下新建一个ch目录并放入相关配置文件。
mkdir ch //新建ge3目录
里面该放入什么文件呢，那么我们先看看mkz_example下面是什么文件，ls命令，我们可以看到该目录下有如下文件夹及文件

- camera_params 目录
- gnss_params 目录
- init_params目录
- radar_params 目录
- vehicle_params目录
- velodyne_params目录
- calibration_table.pb.txt 文件
- navigation_ch.pb.txt 文件
- start_velodyne.launch 文件
- vehicle_info.pb.txt 文件
- vehicle_params.pb.txt 文件

gnss_params目录主要是关于gps基站和IMU的相关配置，其中gnss_conf.pb.txt是关于基站的配置，其中内容在gps配置部分已经作了说明，必须保证信息正确。
vehicle_params目录下是关于车辆translation和rotation相关设置，不用修改，
vehicle_info.pb.txt是车辆型号配置，根据新车型作相应修改。

因此如果只做循迹方案，只需要gnss_params目录，vehicle_params目录，vehicle_info.pb.txt文件，其他文件并不需要。
保证这些文件正确，重新编译，启动bootstrap， 就可以在dreamview下选择酷黑小车ch车辆，进行下面的循迹操作了。

循迹操作说明

当gps安装配置好以后，就可以进行循迹测试，所谓的循迹测试是对车辆底盘线控改造的一个综合测试（ 对油门刹车档位等的控制 ），以及对apollo控制算法和传感器集成的综合测试。下面按时间先后对循迹测试的步骤做说明。
以下所说的操作都是在apollo docker内部。

循迹前检查和准备

在命令行中执行

```
1. bash bootstrap.sh
```

启动dreamview，浏览器中输入http://：8888进入dreamview界面，;在dreamview顶部下拉菜单中选择ch车辆（表示选择酷黑车型），rtk_record_play表示选择循迹。
点击左侧的task标签，点击页面中的reset all ，接着点击setup，将会启动apollo中相应的模块。点击页面左侧的modules controllers，然后在右侧查看各个模块的打开情况，蓝色表示快开启，灰色表示模块处于关闭状态，确保GPS，CAN Bus， Control，Localization模块是开启的，其他的Gardian和Record Bag处于关闭状态。
执行如下三个命令可以看到gps， imu， localization信号和状态。

```
1. rostopic echo /apollo/sensor/gnss/best_pose
2. rostopic echo /apollo/sensor/gnss/imu
3. rostopic echo /apollo/localization/pose
```

都正常，就可以开始进行循迹测试(有时候localization信号没有，可能需要等待一会儿，如果还是没有那么可以尝试让车辆跑一下)。

录制循迹数据包

车辆开到循迹起始点，保持N档，线控开启

```
1. cd /apollo/scripts
2. bash rtk_record.sh setup
3. bash rtk_record.sh start （ 命令输入后，开始开车走一段轨迹 ）
4. bash rtk_record.sh stop( 如果无法输入就按Ctrl + C结束 )
```

Ctrl + C结束后，apollo将会录制一个轨迹数据包garage.csv，放在data/log/下（主要记录了位置、刹车、油门、方向、速度、等信息）。

回放数据包

N档，线控开启，

```
1. bash scripts/rtk_player.sh setup
2. bash scripts/rtk_player.sh start （ 这个命令敲完，车还不会反应 ）
```

发现命令行中record数据被重放，log不停刷屏，dreamview会显示大致轨迹，清浏览轨迹是否大致相符，如果相差很大，比如本来录制的时候是直行，而轨迹显示大幅度转弯，请小心测试，谨防危险。
点击start auto，这时候车子会出现反应，并且是大反应（司机注意接管）。bash scripts/rtk_player.sh start 这一步只是把record数据重新放出来，或者对路径进行规划，即完成planning的过程。

注意事项：一定要在宽阔的场地进行测试，确保周围没有人，循迹测试之前做好应急接管准备，有问题随时遥控器接管。不要轻易尝试速度大于20km/h（ 6m/s ）的循迹

结束播放

bash rtk_player.sh stop(如果无法输入就按Ctrl + C或者Ctrl + D 结束)

循迹回放时录制数据包

为了能观察循迹效果，可以在循迹回放的时候录制数据包，命令如下

```
bash record_bag.sh start && bash rtk_player.sh start （ 车辆开始启动并开始录）
Ctrl + C （ 车辆循迹到终点时输入 Ctrl + C结束循迹 ）
bash record_bag.sh stop （ 停止录制 ）
```

该数据包都放在/apollo/data/bag/下，并以时间命名。该数据包可以用来分析和评估车辆线控适配后的循迹效果。
录制数据包也可以用dreamview界面中的按钮来完成，有兴趣的开发者可以尝试。

分析评估车辆线控循迹效果

apollo准备了一个工具用于评估循迹效果，拷贝循迹过程中录制的包到/apollo下的一个目录（ 自己新建目录），拷贝分析工具到该目录下，工具名称:
control_planning_evaluation.py
然后执行

```
python control_planning_evaluation.py --bag xxx.bag
```

注意这个命令是在docker内部执行，xxx.bag为自己循迹过程中录制的数据包名字。
通常第一次执行的时候会报错提示要安装xlwt， 请按照要求输入命令安装xlwt.
执行完后将生成一个曲线图和一个评分表格。可以根据图表来分析优化自己的车辆线控控制，从而改进循迹效果。

调试与常见问题

本小节讲述小车Apollo 循迹过程的实际调试方法和操作，在此之前必须保证环境已经准备好，即Ubuntu 14.4已经装好kernel，docker，canbus，硬件和传感器已经连接好。

底盘连接是否正常

硬件连接：确保CAN硬件连接线CAN0和车辆线控底盘连接正常，CAN卡跳线帽设置正确，线控底盘开关处于打开状态
进入Ubuntu Docker环境，输入以下命令，

```
cd /apollo/scripts
bash roscore.sh  启动ros
bash canbus.sh  启动canbus模块
bash diagnostics.sh 启动diagnostics工具
```

在diagnostics界面中应该能看到如下的模块

Module	Topic	Period	Max	Min
CHASSIS	/apollo/canbus/chassis	9.45	10.96	9.18
CHASSIS_DETAIL	/apollo/canbus/chassis_detail	0.00	0.00	0.00

用键盘上下箭头移动光标选择CHASSIS或者CHASSIS_DETAIL，选中后按右箭头可以进入查看详细信息，这些信息即是车辆底盘信息，
CHASSIS消息如下：

```
Chassis
  engine_started: True
  speed_mps: 0.00000
  throttle_percentage: 0.00000
  brake_percentage: 0.00000
  steering_percentage: 0.00000
  driving_mode: COMPLETE_MANUAL
  error_code: NO_ERROR
  gear_location: GEAR_REVERSE
  header:
    timestamp_sec: 1551682297.83584
    module_name: chassis
    sequence_num: 10234
  engage_advice:
    advice: READY_TO_ENGAGE
```

Chassis detail消息如下：

```
: GEAR_STS_REVERSE
  check_response:
    is_eps_online: False
    is_esp_online: False
    is_vcu_online: False
  ch:
    brake_status__511:
      brake_pedal_en_sts: BRAKE_PEDAL_EN_STS_DISABLE
      brake_pedal_sts: 0
    throttle_status__510:
      throttle_pedal_en_sts: THROTTLE_PEDAL_EN_STS_DIS
      throttle_pedal_sts: 0
    turnsignal_status__513:
      turn_signal_sts: TURN_SIGNAL_STS_NONE
    steer_status__512:
      steer_angle_en_sts: STEER_ANGLE_EN_STS_DISABLE
      steer_angle_sts: 0.00000
    ecu_status_1_515:
      speed: 0.00000
      acc_speed: 0.00000
      ctrl_sts: CTRL_STS_OUT_OF_CONTROL
      chassis_sts: 0
      chassis_err: 8
    gear_status_514:
```

如果这些信息得到正确显示。说明CANBUS模块工作正常。如果不能显示底盘信息，很大可能是CANBUS有问题，或者底盘有问题，没有向上反馈底盘数据。

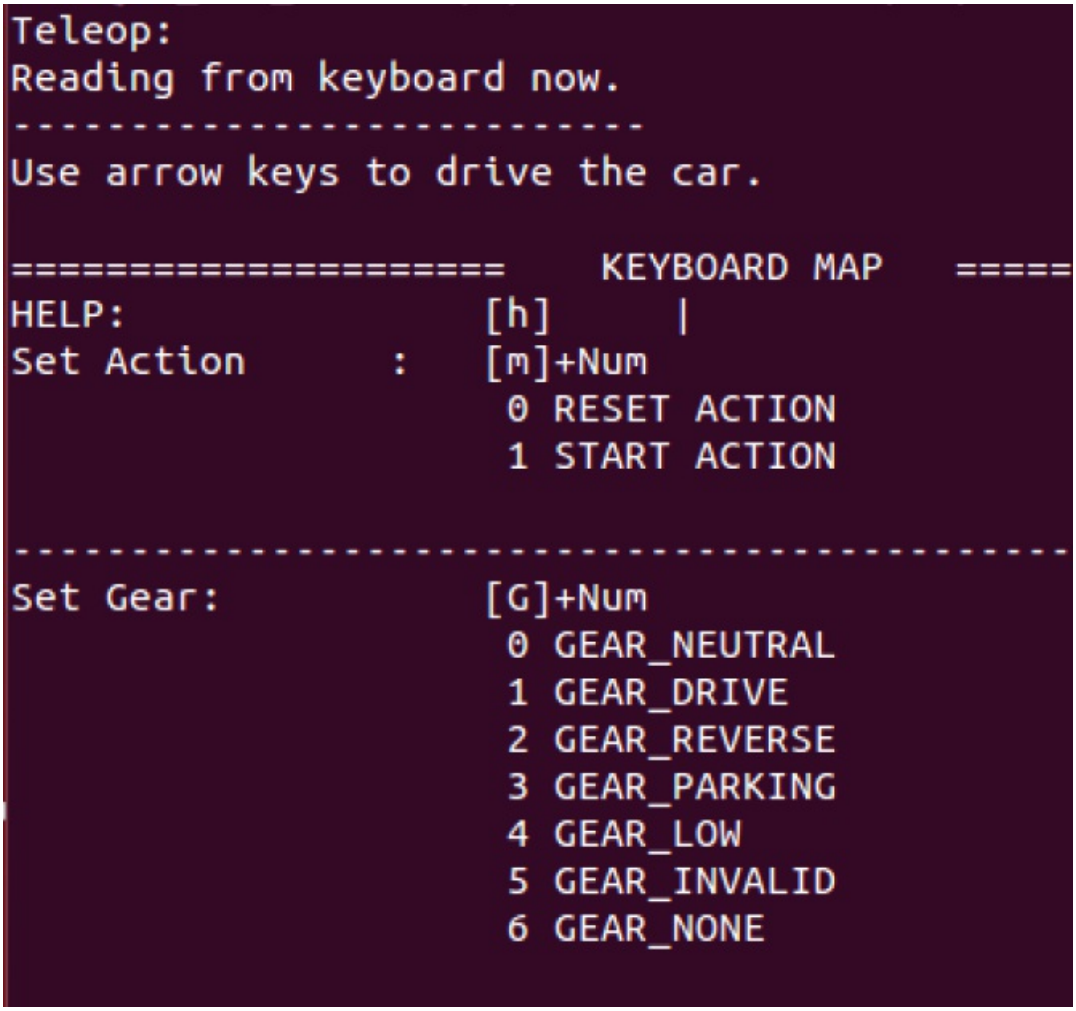
保证CANBUS的连接正确性，才能确定工控机计算单元可以控制车辆底盘，才能继续其它部分的调试。

canbus_teleop的使用

CANBUS能够连接底盘后，我们可以尝试控制底盘的运动。如果控制正常，则说明工控机计算单元下发命令后得到正确执行。
进入Ubuntu Docker环境，输入以下命令，

```
cd /apollo/scripts
bash canbus_teleop.sh
```

弹出界面如下：



a 根据提示按M0重置系统，按 M1是开始控制底盘，
b 按几次A或者D，看看车轮是否转动。
c 按G1，挂前进档，按几次W，看车辆是否前进，按几次S看车辆是否停下来。
（ 请小心测试，不要轻易长时间连续按键，以防车辆突然高速动作发生事故 ）

星网宇达M2 GPS调试

M2硬件设备的配置

M2 GPS首先需要保证M2硬件设备配置好，参考《Newton-M1使用维护说明书》（ M1和M2配置方法都相同 ）。默认出厂的M2已经配置好，可以给百度apollo系统使用，只需要小部分改动，例如杆臂值的配置。

硬件连接：需要把M2的串口线连接到工控机的串口。M2的USB口连接到工控机的USB口。M2接上电源并上电。
软件部分：Ubuntu 14.4 系统安装串口调试工具cutecom（ sudo apt-get install cutecom ）
安装好以后应该可以在/dev/下看到ttyACM0的设备，这就是我们的M2设备名。修改设备权限sudo chmod 777 /dev/ttyACM0,为了避免每次开机修改，可以修改apollo启动相关文件/apollo/scripts/docker_adduser.sh 在41行后面加上如下程序段

```
    if [ -e /dev/ttyACM0 ]; then
       chmod a+rw /dev/ttyACM0
    fi
```

apollo GPS模块相关配置

根据apollo星网宇达GPS配置相关说明文档，修改/apollo/modules/drivers/gnss/conf文件夹下面的配置文件

M2 GPS状态查看

```
cd /apollo/scripts
bash roscore.sh            启动ros
bash gps.sh                启动gps
bash localization.sh       启动localization
```

rostopic list 列出topic，应该可以得出以下结果

```
budaoshi@in_dev_docker:/apollo/scripts$ rostopic list
/apollo/canbus/chassis
/apollo/canbus/chassis_detail
/apollo/control
/apollo/guardian
/apollo/monitor
/apollo/sensor/gnss/best_pose
/apollo/sensor/gnss/corrected_imu
/apollo/sensor/gnss/gnss_status
/apollo/sensor/gnss/heading
/apollo/sensor/gnss/imu
/apollo/sensor/gnss/ins_stat
/apollo/sensor/gnss/ins_status
/apollo/sensor/gnss/odometry
/apollo/sensor/gnss/raw_data
/apollo/sensor/gnss/rtcm_data
/apollo/sensor/gnss/rtk_eph
/apollo/sensor/gnss/rtk_obs
/apollo/sensor/gnss/stream_status
/rosout
/rosout_agg
/tf
```

我们在双天线M2 GPS方案中，主要用到如下两个topic：best_pose和imu。
输入命令

```
rostopic echo /apollo/sensor/gnss/best_pose
```

```
measurement_time: 1235717903.0
sol_status: SOL_COMPUTED
sol_type: NARROW_INT
latitude: 40.0883592504
longitude: 116.107416244
height_msl: 76.7926985528
undulation: -9.80000019073
datum_id: WGS84
latitude_std_dev: 0.011541582644
longitude_std_dev: 0.00924708787352
height_std_dev: 0.0277880299836
base_station_id: "2"
differential_age: 1.0
solution_age: 0.0
num_sats_tracked: 32
num_sats_in_solution: 30
num_sats_l1: 30
num_sats_multi: 24
extended_solution_status: 33
galileo_beidou_used_mask: 48
gps_glonass_used_mask: 51

---
```

输入命令

```
1. rostopic echo /apollo/sensor/gnss/imu
```

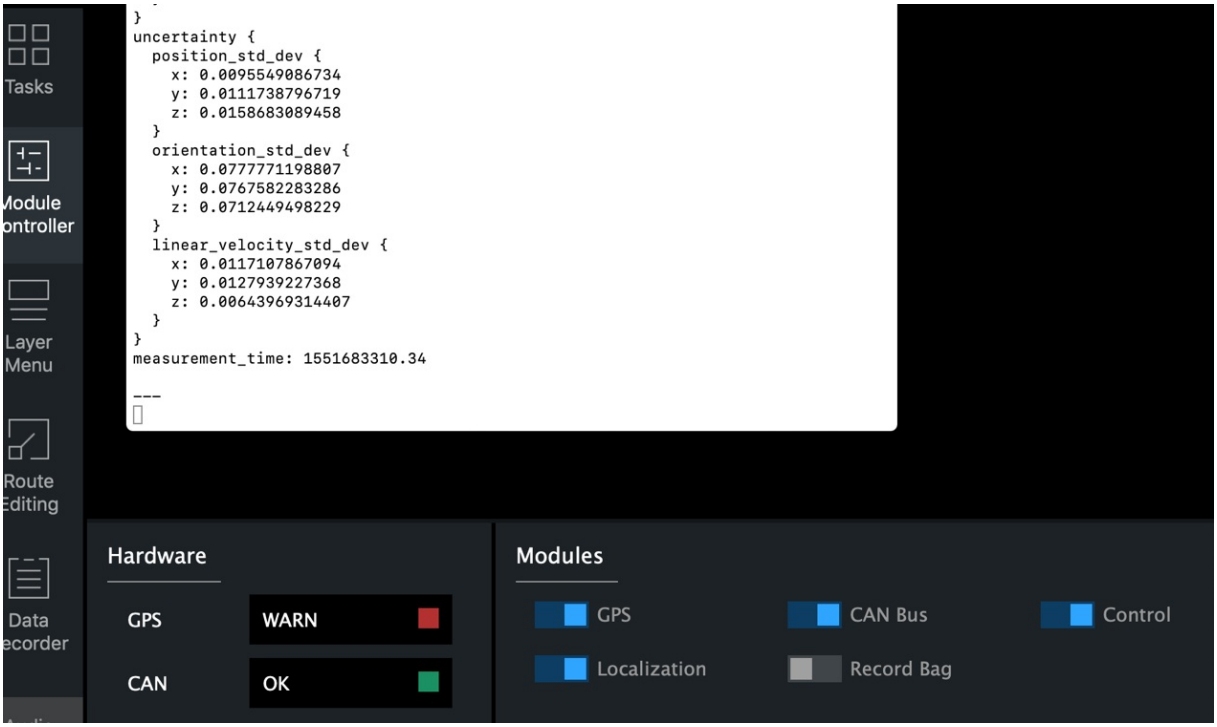
```
---
header {
  timestamp_sec: 1551682715.57
  module_name: "gnss"
  sequence_num: 4556
}
measurement_time: 1235717933.55
measurement_span: 0.00999999977648
linear_acceleration {
  x: -0.062255859375
  y: 0.0277709960938
  z: 9.78591918945
}
angular_velocity {
  x: -0.000282118055556
  y: 0.00399305555556
  z: 0.000423177083333
}

---
```

接下来我们通常需要打开localization模块，鉴于后续我们都会用到dreamview模块，所以我们先打开dreamview，然后在dreamview界面中打开localization模块。
输入命令：

```
1.  bash bootstrap.sh start
```

然后在网页中输入http://IP_ADDRESS:8888(如果在本地机器可以用http://localhost:8888)即可以打开dreamview界面，该界面可以控制和监测车辆自动驾驶动作和效果。选择左侧的modules conttroller，可以出现很多模块，我们点击localization使它变成蓝色即可以打开localization模块，接着在命令行，就可以用命令rostopic echo /apollo/localization/pose观察localization的效果，dreamview和localization topic的效果如下图所示



（ 在GPS定位方案中，localization模块和GPS模块相关，如果GPS模块没有打开并正常工作那么localization模块也无法正常工作 ）

如果无法得出以上结果，请到/apollo/data/log/下查看日志gnss.ERROR，并根据报错做相应的排查。Apollo所有的日志都在/apollo/data/log/下，要多看这些log才能知道问题和运行过程，更快地完成调试。

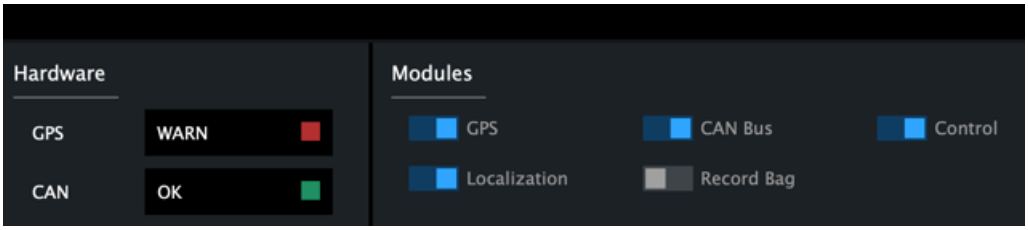
rtk循迹测试的调试

配置dreamview

在dreamview界面中确保选择自己的车型和RTK_Record/Replay,如下2幅图所示

打开传感器

确保canbus,roscore,control,gps,localization这几个模块是打开的，如下图



录制轨迹包

车辆保持N档，线控权限交给apollo，然后在命令行中，输入如下命令。
bash rtk_recorder.sh setup
bash rtk_recorder.sh start （ 输入这条命令后，命令行界面会看到消息不停刷屏，记录车辆运动轨迹，开始用遥控控制车辆走一段距离 ，让车辆停止后，在命令行中ctrl + c按钮结束数据录制 ）
这些数据会记录到/apollo/data/log/garage.csv文件夹中，如下图所示

```
[RtkRecord][DEBUG] 2019-03-04 15:17:23,974 rtk_recorder.py:152 started moving and write data at
[RtkRecord][DEBUG] 2019-03-04 15:17:23,984 rtk_recorder.py:152 started moving and write data at
[RtkRecord][DEBUG] 2019-03-04 15:17:23,994 rtk_recorder.py:152 started moving and write data at
^C[RtkRecord][INFO] 2019-03-04 15:17:24,167 rtk_recorder.py:163 Shutting Down...
[RtkRecord][INFO] 2019-03-04 15:17:24,167 rtk_recorder.py:164 file is written into /apollo/modul
../../data/log//garage.csv
budaoshi@in_dev_docker:/apollo/scripts$ ls -l ./../data/log/ga*
-rw-r--r-- 1 budaoshi budaoshi 129584 Mar  4 15:17 ./../data/log/garage.csv
```

回放数据包

Dreamview界面要准备好，需要命令行和dreamview界面配合操作
在命令行界面中输入如下命令，

```
1. bash rtk_player.sh setup
2. bash rtk_player.sh start
```

如下图所示

```
budaoshi@in_dev_docker:/apollo/scripts$ bash rtk_player.sh setup
Module canbus is already running - skipping.
Module gnss is already running - skipping.
Module localization is already running - skipping.
Module control is already running - skipping.
budaoshi@in_dev_docker:/apollo/scripts$ bash rtk_player.sh start
```

这时候会有很多关于轨迹的数据播放并刷屏，dreamview界面中也会看到有一条浅蓝色的线，代表车辆即将要走的轨迹。接下来需要在dreamview 界面中点击Start Auto，如下图所示，车辆开始运动，观察其循迹效果