**底盘 && Radar数据采集**

1. **底盘 && radar数据采集**

|  |
| --- |
| 参考：[车机系统](https://vqgjrby0o97.feishu.cn/wiki/wikcnrzJiPz9QYl9ZgO4WGU2Pug?table=tblVdbBHI1kLudV2&view=vewUIIwxDi) 文档若无法查看，请向@欧阳希修申请权限。 |

1.1 **采集步骤**

1.1.1 **can口硬件连接**

以车牌：**粤B97E4为例：**

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| * 域控制器的can0连接到底盘的can口 * 域控制器的can1连接到radar的can口 * 上图中，左边为连接示意图，右边为域控制器主板CAN接线定义。按照此定义，左图中C2\_D2接头为CAN0，E2\_F2接头为CAN1。 |

车牌：**粤B97E4**上的

* 底盘的DBC文件：

**[MPC\_20220902.dbc]**

* 毫米波雷达的DBC文件：

**[06.3\_ARS410OD\_ObjectCAN\_v191016\_CANFD版\_私can.dbc]**

1.1.2 **底盘 && RADAR 数据的DBC报文**

* 轮速：

|  |
| --- |
| SQL BO\_ 496 IPB\_0x1F0: 8 Vector\_\_XXX  SG\_ Wheel\_Speed\_FL\_1F0\_S : 0|12@1+ (0.06875,0) [0|281.4625] "km/h" MPC  SG\_ Wheel\_Speed\_FR\_Stats\_1F0\_S : 12|1@1+ (1,0) [0|1] "" MPC  SG\_ Wheel\_Speed\_FL\_Stats\_1F0\_S : 13|1@1+ (1,0) [0|1] "" MPC  SG\_ Wheel\_Speed\_RR\_Stats\_1F0\_S : 14|1@1+ (1,0) [0|1] "" MPC  SG\_ WheelSpeed\_RL\_Status\_1F0\_S : 15|1@1+ (1,0) [0|1] "" MPC  SG\_ Wheel\_Speed\_FR\_1F0\_S : 16|12@1+ (0.06875,0) [0|281.4625] "km/h" MPC  SG\_ Wheel\_Speed\_RL\_1F0\_S : 28|12@1+ (0.06875,0) [0|281.4625] "km/h" MPC  SG\_ Wheel\_Speed\_RR\_1F0\_S : 40|12@1+ (0.06875,0) [0|281.4625] "km/h" MPC  SG\_ Message\_Counter\_1F0\_S : 52|4@1+ (1,0) [0|15] "" MPC  SG\_ Checksum\_1F0\_S : 56|8@1+ (1,0) [0|255] "" MPC |

* 车速：

|  |
| --- |
| SQL BO\_ 289 IPB\_0x121: 8 Vector\_\_XXX  SG\_ IPB\_Vehicle\_Speed\_S : 0|12@1+ (0.06875,0) [0|281.4625] "km/h" MPC  SG\_ Vehicle\_Speed\_Stats\_S : 15|1@1+ (1,0) [0|1] "" MPC  SG\_ AWB\_Available\_S : 28|1@1+ (1,0) [0|1] "" MPC  SG\_ AWB\_Active\_Brake\_Warning\_S : 29|1@1+ (1,0) [0|1] "" MPC  SG\_ AEB\_Dec\_Active\_S : 30|1@1+ (1,0) [0|1] "" MPC  SG\_ AEB\_not\_Available\_S : 31|1@1+ (1,0) [0|1] "" MPC  SG\_ Prefill\_Available\_S : 32|1@1+ (1,0) [0|1] "" MPC  SG\_ Prefill\_Active\_S : 33|1@1+ (1,0) [0|1] "" MPC  SG\_ ABAavailable\_S : 34|1@1+ (1,0) [0|1] "" MPC  SG\_ ABA\_Active\_S : 35|1@1+ (1,0) [0|1] "" MPC  SG\_ Alive\_Counter\_121\_S : 48|4@1+ (1,0) [0|15] "" MPC  SG\_ Checknum121\_S : 56|8@1+ (1,0) [0|255] "" MPC |

* yawRate：

|  |
| --- |
| SQL BO\_ 546 IPB\_0x222: 8 Vector\_\_XXX  SG\_ Yaw\_Rate\_Signal\_S : 0|12@1+ (0.002132603,-2.0943) [-2.0943|2.0943] "rad/s" MPC  SG\_ Yaw\_Rate\_Offset\_S : 12|12@1+ (0.002132603,-0.13) [-0.13|0.13] "rad/s" MPC  SG\_ EBD\_Active\_222\_S : 24|1@1+ (1,0) [0|1] "" MPC  SG\_ ABS\_Active\_222\_S : 25|1@1+ (1,0) [0|1] "" MPC  SG\_ EBD\_Fault\_222\_S : 26|1@1+ (1,0) [0|1] "" MPC  SG\_ ABS\_Fault\_222\_S : 27|1@1+ (1,0) [0|1] "" MPC  SG\_ TCS\_Active\_222\_S : 28|1@1+ (1,0) [0|1] "" MPC  SG\_ DWT\_B\_Active\_222\_S : 29|1@1+ (1,0) [0|1] "" MPC  SG\_ VDC\_Active\_222\_S : 30|1@1+ (1,0) [0|1] "" MPC  SG\_ Yaw\_Rate\_Status\_S : 31|1@1+ (1,0) [0|1] "" MPC  SG\_ Message\_Counter222\_S : 48|4@1+ (1,0) [0|15] "" MPC  SG\_ Checksum222\_S : 56|8@1+ (1,0) [0|255] "" MPC |

* radar：

**粤B97E4**车上的毫米波雷达的报文包含了40个目标的报文，比如：

|  |
| --- |
| SQL BO\_ 32 FRS\_Obj\_00\_Part2of2: 8 FRS  SG\_ FRS\_P2\_00\_Obj\_YVelRel : 47|11@0+ (0.1,-102.4) [-102.4|102.3] "m/s" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Obj\_YPos : 12|13@0+ (0.015625,-64) [-64|63.984375] "m" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Obj\_XVelRel : 7|11@0+ (0.1,-102.4) [-102.4|102.3] "m/s" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Obj\_XPos : 31|14@0+ (0.015625,0) [0|255.984375] "m" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Obj\_Type : 33|2@0+ (1,0) [0|3] "n/a" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Obj\_MeasFlag : 52|1@0+ (1,0) [0|1] "n/a" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Msg\_AliveCounter : 51|4@0+ (1,0) [0|15] "n/a" Vector\_\_XXX  SG\_ FRS\_P2\_00\_Msg\_CheckSum : 63|8@0+ (1,0) [0|255] "n/a" Vector\_\_XXX  BO\_ 80 FRS\_Obj\_00\_Part1of2: 8 FRS  SG\_ FRS\_P1\_00\_Msg\_AliveCounter : 51|4@0+ (1,0) [0|15] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Msg\_CheckSum : 63|8@0+ (1,0) [0|255] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_MotionPattern : 31|3@0+ (1,0) [0|7] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_YPos\_Stdev : 22|7@0+ (0.1,0) [0|12.7] "m" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_XVelRel\_Stdev : 32|7@0+ (0.05,0) [0|6.35] "m/s" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_XPos\_Stdev : 14|7@0+ (0.1,0) [0|12.7] "m" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_XAccRel : 39|7@0+ (0.15,-9.6) [-9.6|9.45] "m/s2" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_ValidFlag : 23|1@0+ (1,0) [0|1] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_UpdateFlag : 15|1@0+ (1,0) [0|1] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_ObstacleProb : 28|5@0+ (3.2258,0) [0|99.9998] "%" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_ID : 7|8@0+ (1,0) [0|255] "n/a" Vector\_\_XXX  SG\_ FRS\_P1\_00\_Obj\_ExstProb : 41|6@0+ (1.5873,0) [0|99.9999] "%" Vector\_\_XXX |

BO\_ 32 FRS\_Obj\_00\_Part2of2和BO\_ 80 FRS\_Obj\_00\_Part1of2为FRS\_Obj\_00的part2与part1，两者组合在一起代表了FRS\_Obj\_00的信息。

1.1.3 **自启动CAN口**

* 创建 enable\_CAN.sh 以实现系统启动时自动启动CAN口：

|  |
| --- |
| Bash touch /enable\_CAN.sh chmod 777 /enable\_CAN.sh |

* enable\_CAN.sh具体内容：

|  |
| --- |
| Bash #!/bin/bash #设置 can0 为 canfd 模式，波特率 500Kbps，数据段波特率 2Mbps sudo ip link set can0 type can bitrate 500000 dbitrate 2000000 fd on #打开CAN0 sudo ip link set up can0  #设置 can1 为 canfd 模式，波特率 500Kbps，数据段波特率 2Mbps sudo ip link set can1 type can bitrate 500000 dbitrate 2000000 fd on #打开CAN1 sudo ip link set up can1  exit 0 |

* 创建自启动文件/etc/rc.local：

|  |
| --- |
| Bash printf '%s\n' '#!/bin/bash' 'exit 0' | sudo tee -a /etc/rc.local sudo chmod +x /etc/rc.local |

* 在/etc/rc.local中添加：

|  |
| --- |
| Bash sh /enable\_CAN.sh & |

现在系统启动时能够自动运行 enable\_CAN.sh 脚本并配置 CAN 控制器了。可以重新启动 Jetson Xavier，在终端中运行 ifconfig 命令，检查是否看到 can0 和 can1 设备。

1.1.4 **自启动底盘 && radar数据发布节点**

|  |
| --- |
| **注意：底盘 && radar数据的发布会使用ros开展，因此需要具备ros开发经验。** |

|  |
| --- |
| 车牌：**粤B97E4**底盘 && radar的数据发布的代码位于工作空间**/home/nvidia/test\_can/ws\_pubCanData中。**  如若需要在其他车辆上部署数据发布节点，则需要：   1. 创建工作空间   mkdir -p ws\_pubCanData/src  cd ws\_pubCanData/src  catkin\_init\_workspace  cd ws\_pubCanData  catkin\_make   1. 创建功能包   cd ws\_pubCanData/src  catkin\_create\_pkg pkg\_pubCanData roscpp sensor\_msgs std\_msgs |

将如下的功能包解压，将其中的内容放于上述步骤2创建的功能包中。

**[pkg\_pubCanData.zip]**

然后，返回到文件夹ws\_pubCanData中：

cd ws\_pubCanData

并执行编译命令：

catkin\_make

* 创建启动底盘数据发布节点的脚本pubChassis.sh：

|  |
| --- |
| Bash touch /pubChassis.sh chmod 777 /pubChassis.sh |

pubChassis.sh具体内容：

|  |
| --- |
| Bash #具体的路径取决于工作目录所在的路径 cd /home/nvidia/test\_can/ws\_pubCanData  source devel/setup.bash roslaunch pkg\_pubCanData pubChassis.launch |

* 创建启动radar数据发布节点的脚本pubRadar.sh：

|  |
| --- |
| Bash touch /pubRadar.sh chmod 777 /pubRadar.sh |

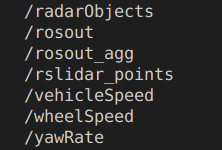
pubRadar.sh具体内容：

|  |
| --- |
| Bash #具体的路径取决于工作目录所在的路径 cd /home/nvidia/test\_can/ws\_pubCanData  source devel/setup.bash roslaunch pkg\_pubCanData pubRadar.launch |

* 在/etc/rc.local中添加：

|  |
| --- |
| Bash bash /pubChassis.sh & bash /pubRadar.sh & |

现在系统启动时能够自动运行 pubChassis.sh 与 pubRadar.sh脚本并发布底盘与radar的数据话题了。可以重新启动 Jetson Xavier，在终端中运行rostopic list命令，检查是否看到底盘与radar的数据话题了。



话题名具体为：

* 底盘相关：  
  yawrate数据：  
  yawRate  
  车速数据：  
  vehicleSpeed  
  轮速信息：  
  wheelSpeed
* radar毫米波雷达的数据  
  radarObjects

1.1.5 **底盘 && radar 节点数据话题的订阅**

通过自启动CAN口 && 自启动底盘 && radar数据发布节点两个步骤，能够实现开机自动发布对应的数据节点话题。

现在具体介绍底盘以及radar的话题名以及对应的话题订阅。

由于底盘以及radar的数据均是自定义的消息结构，因此在订阅节点的功能包中需要创建对应的msg文件夹，并在其中创建:

yawRate.msg、wheelSpeed.msg、vehicleSpeed.msg、radarObjects.msg、radarObject.msg

|  |
| --- |
| 在创建订阅节点的时候，记得在订阅节点所在功能包的CMakeLists.txt与package.xml中添加自定义消息的相关配置： |

* CMakeLists.txt

|  |
| --- |
| CMake find\_package(catkin REQUIRED COMPONENTS  roscpp  sensor\_msgs  std\_msgs  message\_generation )  add\_message\_files(  FILES  wheelSpeed.msg  yawRate.msg  vehicleSpeed.msg  radarObject.msg  radarObjects.msg )  generate\_messages(  DEPENDENCIES  sensor\_msgs  std\_msgs )  catkin\_package( # INCLUDE\_DIRS include # LIBRARIES pkg\_pubChassisData  CATKIN\_DEPENDS roscpp sensor\_msgs std\_msgs message\_runtime # DEPENDS system\_lib ) |

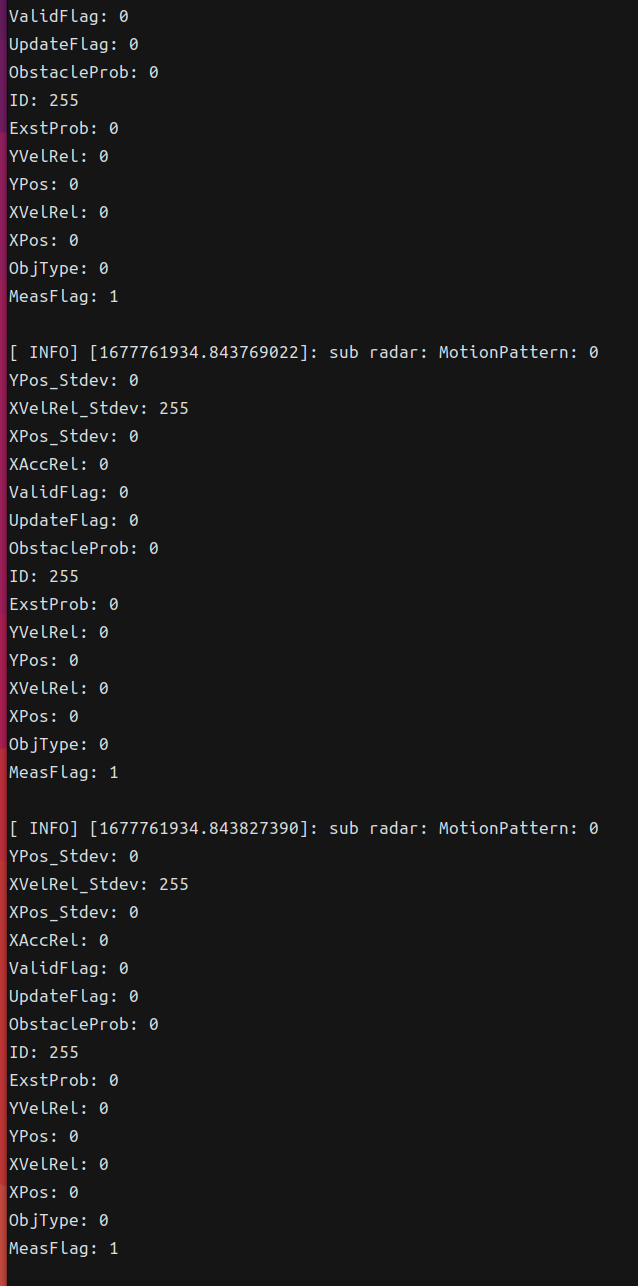
* package.xml

|  |
| --- |
| XML <build\_depend>message\_generation</build\_depend> <exec\_depend>message\_runtime</exec\_depend> |

|  |
| --- |
| 在创建订阅节点的时候，记得在订阅节点所在功能包的src文件夹中的main函数所在的C++文件中include底盘和radra的头文件：   * 比如，如果创建的功能包名为pkg\_subCanData，那么需要include的头文件的示例为： |

|  |
| --- |
| C++ #include "pkg\_subCanData/wheelSpeed.h" #include "pkg\_subCanData/vehicleSpeed.h" #include "pkg\_subCanData/yawRate.h"  #include "pkg\_subCanData/radarObject.h" #include "pkg\_subCanData/radarObjects.h" |

示例，订阅发布出来的毫米波雷达的话题内容：



1.1.5.1 **底盘**

* yawRate话题

yawRate话题的具体ros消息的定义为yawRate.msg：

**[yawRate.msg]**

其具体内容：

|  |
| --- |
| Go std\_msgs/Header header float32 yawRate |

* vehicleSpeed话题

vehicleSpeed话题的具体ros消息的定义为vehicleSpeed.msg：

**[vehicleSpeed.msg]**

其具体内容为：

|  |
| --- |
| Plain Text std\_msgs/Header header float32 vehicleSpeed |

* wheelSpeed话题

wheelSpeed话题的具体ros消息的定义为wheelSpeed.msg：

**[wheelSpeed.msg]**

其具体内容为：

|  |
| --- |
| Plain Text std\_msgs/Header header #左前轮速 float32 FLSpeed #右前轮速 float32 FRSpeed #左后轮速 float32 RLSpeed #右后轮速 float32 RRSpeed |

1.1.5.2 **radar**

* radarObjects话题

毫米波话题的具体ros消息的定义为radarObjects.msg：

**[radarObjects.msg]**

其具体内容为：

|  |
| --- |
| Plain Text std\_msgs/Header header radarObject[] radarObjects |

其中radarObject类型的数据，定义在radarObject.msg：

**[radarObject.msg]**

其具体内容为：

|  |
| --- |
| Plain Text # std\_msgs/Header header # 7 "RESERVED" 6 "RESERVED" 5 "RESERVED" 4 "CROSSING" 3 "MOVING" 2 "STOPPED" 1 "STATIONARY" 0 "UNKNOWN" uint8 MotionPattern float32 YPos\_Stdev float32 XVelRel\_Stdev float32 XPos\_Stdev float32 XAccRel # 1 "OBJECT\_VALID" 0 "OBJECT\_INVALID" uint8 ValidFlag # 1 "OBJECT\_EXISTED\_IN\_PREVIOUS\_CYCLE" 0 "NEW\_OBJECT\_INTHE\_CYCLE" uint8 UpdateFlag float32 ObstacleProb # 255 "INVALID" uint16 ID float32 ExstProb int16 YVelRel int16 YPos int16 XVelRel int16 XPos # 3 "PEDESTRAIN" 2 "TWO\_WHEELER" 1 "FOUR\_WHEELER" 0 "UNKNOWN" uint8 ObjType # 1 "OBJECT\_EXTRAPOLATED\_THIS\_CYCLE" 0 "OBJECT\_MEASURED\_INTHIS\_CYCLE" uint8 MeasFlag |

2. **补充信息**

2.1 **CAN操作相关**

|  |
| --- |
| Bash #查看CAN相关信息 ifconfig -a  #关闭CAN0 sudo ip link set down can0  #设置 can0 为 canfd 模式，波特率 500Kbps，数据段波特率 2Mbps sudo ip link set can0 type can bitrate 500000 dbitrate 2000000 fd on  #打开CAN0 sudo ip link set up can0 |

2.2 **CAN相关学习资料**

* [Linux下can总线dbc载入通信详解](https://blog.csdn.net/hhlenergystory/article/details/81976696)
* [DBC文件解析](https://blog.csdn.net/u010808702/article/details/104152745)
* [Can/CANFD设置脚本](https://blog.csdn.net/weixin_68294039/article/details/125967638)
* [C++的CAN通信demo](https://zhuanlan.zhihu.com/p/623504336?utm_id=0)
* [从DBC和Simulink模型生成CAN报文C++解析代码](https://blog.csdn.net/weixin_39199083/article/details/120402694?spm=1001.2101.3001.6650.6&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-6-120402694-blog-128432066.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7ERate-6-120402694-blog-128432066.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&utm_relevant_index=12)
* [Linux+QT+SocketCAN：使用信号槽机制实现数据收发](https://www.guyuehome.com/39979)
* [TX2平台下can总线收发功能的实现（二）——Linux下can总线编程模型和源码解读](https://blog.csdn.net/hhlenergystory/article/details/79534422?spm=1001.2101.3001.6650.18&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-18-79534422-blog-81976696.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-18-79534422-blog-81976696.235%5Ev35%5Epc_relevant_increate_t0_download_v2_base&utm_relevant_index=22)
* [在 Nvidia Jetson Xavier 开发者套件上启用 CAN 总线](https://steinslab.io/archives/1712)
* [Linux SocketCAN 编程（C++，启用多线程接收）](https://blog.csdn.net/Flag_ing/article/details/126387114)