

开发者说 | 无人驾驶中的CAN消息解析

知
识
点

敲黑板，本文需要学习的知识点有

普通帧

扩展帧

CAN通信

曲率

VCU

负载

在自动驾驶调试的领域，使用CAN接口的应用非常广泛。首先，普及一下相关的知识点，普通的IPC或者PC是不具有CAN口，也就是不能直接和CAN设备通信，所以他们之间需要用到转换设备，将CAN协议转换成适配IPC的协议，例如PCIE、USB等。

像Kvaser PCIEcan 4×HS就是一个CAN转PCIE设备，俗称CAN卡，后面的4xHS代表其可以“一拖四”的带有四个CAN口，它需要插在主板上的PCIE接口。而kvaser leaf light HS v2是一个CAN转USB设备，其只有一个CAN口，是插在USB口即可。

以下，ENJOY



到目前为止，无人驾驶技术入门硬件篇的分享已经介绍完了。接下来我将会分享更多和软件相关的内容。这一期的主要内容是——无人驾驶中的CAN（Controller Area Network）总线。

CAN总线在整个无人驾驶系统中有着十分重要的作用。除了在《无人驾驶技术入门（九）——与生俱来的VCU信号》中提到的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。

我在《无人驾驶，个人如何研究？》中提到过

实现一个无人驾驶系统，会有几个层级：

感知层 → 融合层 → 规划层 → 控制层。

更具体一点为：

传感器层 → 驱动层 → 信息融合层 → 决策规划层 → 底层控制层

本次主要介绍的是“驱动层”相关的内容。

正文

apollo 开发者社区

CAN通信是一套高性能、高可靠性的通信机制，目前已广泛应用在汽车电子领域。有关CAN的总线的原理及特性并不是本次分享的重点。本文的重点在无人驾驶系统获取到CAN消息后，如何根据CAN协议，解析出想要的信息。从CAN总线中解析出传感器的信息，可以说是每个自动驾驶工程师，甚至每一个汽车电子工程师必备的技能。

认识CAN消息

apollo 开发者社区

以百度推出的Apollo开源的代码为例做CAN消息的讲解，我们先看到每一帧的CAN消息是如何被定义的。

```

/**
 * @class CanFrame
 * @brief The class which defines the information to send and receive.
 */
struct CanFrame {
    /// Message id
    uint32_t id;
    /// Message length
    uint8_t len;
    /// Message content
    uint8_t data[8];
    /// Time stamp
    struct timeval timestamp;

    /**
     * @brief Constructor
     */
    CanFrame() : id(0), len(0), timestamp{0} {
        std::memset(data, 0, sizeof(data));
    }
}

```

知乎 @陈光

可以看到这个名为CanFrame的消息结构中包含4个关键信息，分别是：




uint32_t id

CAN消息的ID号。

由于CAN总线上传播着大量CAN消息，因此两个节点进行通信时，会先看ID号，以确保这是节点想要的CAN消息。最初的CAN消息ID号的范围是000-7FF（16进制数），但随着汽车电控信号的增多，需要传递的消息变多，信息不太够用了。工程师在CAN消息基础上，扩展了ID号的范围，大大增加了ID号的上限，并将改进后的CAN消息称为“扩展帧”，旧版CAN消息称为“普通帧”。

如果拿写信做比较，这个ID就有点类似写在信件封面上的名字。



uint8_t len

CAN消息的有效长度。

每一帧CAN消息能够传递最多8个无符号整形数据，或者说能够传递8*8的bool类型的数据。这里的len最大值为8，如果该帧CAN消息中有些位没有数据，这里的len就会小于8。



uint8_t data[8]

CAN消息的实际数据。

正如刚才提到的，每一帧CAN消息都包含至多8*8个bool类型的数据，因此可以通过8*8个方格，可视化CAN消息中的data。如下图所示：

	7	6	5	4	3	2	1	0
data[0]	0	0	1	1	0	0	1	0
data[1]	0	1	1	0	1	0	0	0
data[2]	0	1	0	0	1	1	0	0
data[3]	0	1	1	0	1	0	1	0
data[4]	1	0	1	0	1	0	0	0
data[5]	0	1	1	0	1	0	0	0
data[6]	0	1	0	0	1	0	0	0
data[7]	0	1	1	1	1	0	0	0

CAN 数据示例

知乎 @陈光

在没有CAN协议帮助我们解析的情况下，这里的数据无异于乱码，根本无法得到有用的消息，这也是CAN消息难以破解的原因之一。

Timestamp

CAN消息的时间戳。

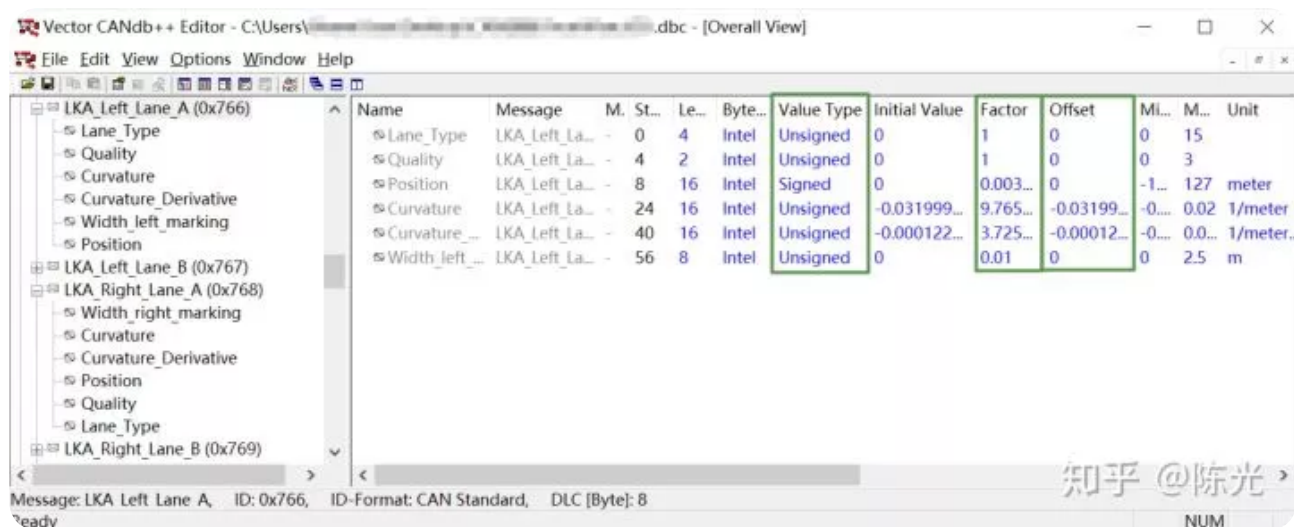
时间戳表示的是收到该CAN消息的时刻。通过连续多帧的时间戳，可以计算出CAN消息的发送周期，也可以用于判断CAN消息是否被持续收到。

综上，每帧CAN消息中最重要的部分其实是Data，即8*8的bool值。所谓解析CAN消息，其实就是解析这8*8个bool类型的值。



目前业界的CAN协议，都是以后缀名为.dbc的文件进行存储的。德国Vector公司提供CANDb++ Editor是一款专门用于阅读.dbc文件的软件。

如下图所示，为Mobileye提供的车道线的.dbc文件。（文末提供CANDb++ Editor安装包和Mobileye车道线的.dbc文件的获取方法）



以ID号为0x766的LKA_Left_Lane_A为例，这是Mobileye检测无人车左侧车道线的部分信息，包括了左侧车道线的偏移量，曲率等。该帧CAN消息（Message）中的五个信号（Signal），分别是Lane_Type、Quality、Curvature、Curvature_Derivative、Width_left_marking、Position。

每个信号的具体描述显示在软件右侧，其中与解析直接相关的三个要素已用绿色框选中。

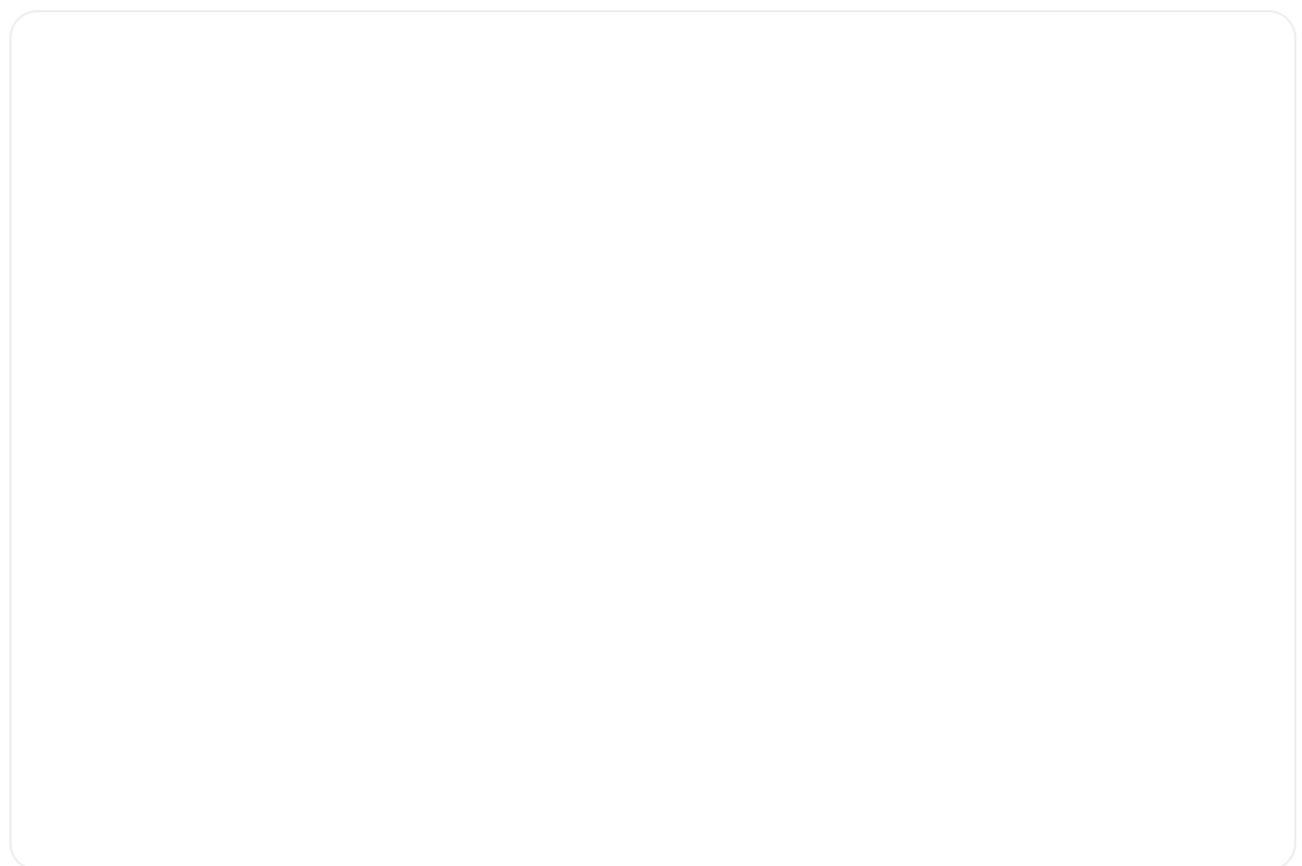
Value Type(Undsigned或Signed)

某些物理量在描述时是有符号的，比如温度。而描述另外一些量时，是没有符号的，即均为正数，比如曲率。

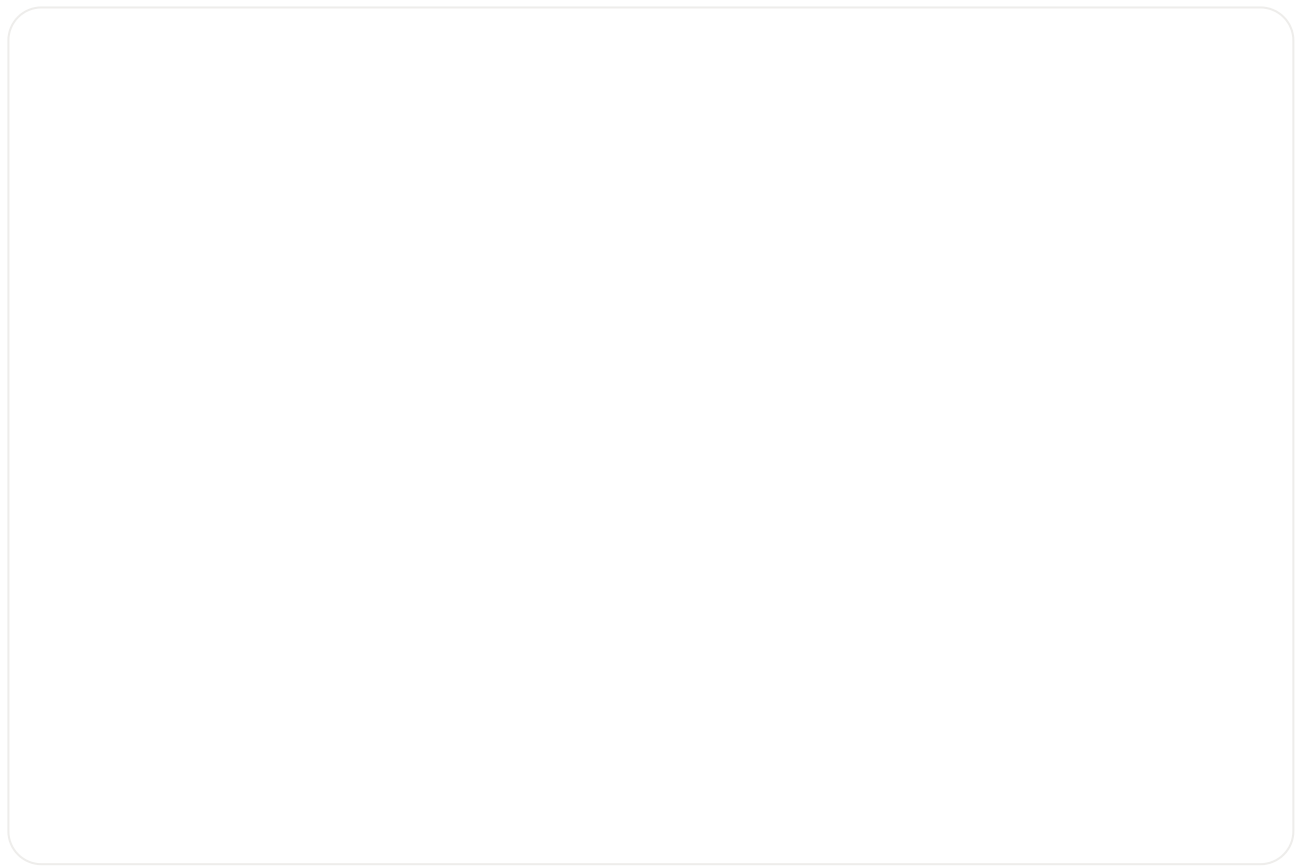
Factor和Offset

这两个参数需要参与实际的物理量运算，Factor是倍率，Offset是偏移量。例如Lane_Type和Quality信号的Factor为1，Offset为0，而其他信号的Factor均为小数。具体的计算方法请往下看。

双击LKA_Left_Lane_A，打开Layout页，会发现很熟悉的方块阵列，如下图所示。



工程师真正关心的恰好是这块彩色图，因为该图上的每个小方块和Data中的每一个bool量一一对应。这就是CAN协议的真面目。



由于彩色方块图与Data是一一对应的，我们将两个图叠加，将得到如下图所示的data图。

	7	6	5	4	3	2	1	0
data[0]	0	0	Quality 1	1	Lane_Type 0	0	1	0
data[1]	0	1	1	0	1	0	0	0
data[2]	Position 0	1	0	0	1	1	0	0
data[3]	0	1	1	0	1	0	1	0
data[4]	Curvature 1	0	1	0	1	0	0	0
data[5]	0	1	1	0	1	0	0	0
data[6]	Curvature_Derivative 0	1	0	0	1	0	0	0
data[7]	Width_left_marking 0	1	1	1	1	0	0	0

CAN 数据示例

知乎 @陈光

每个信号物理量的计算公式为：

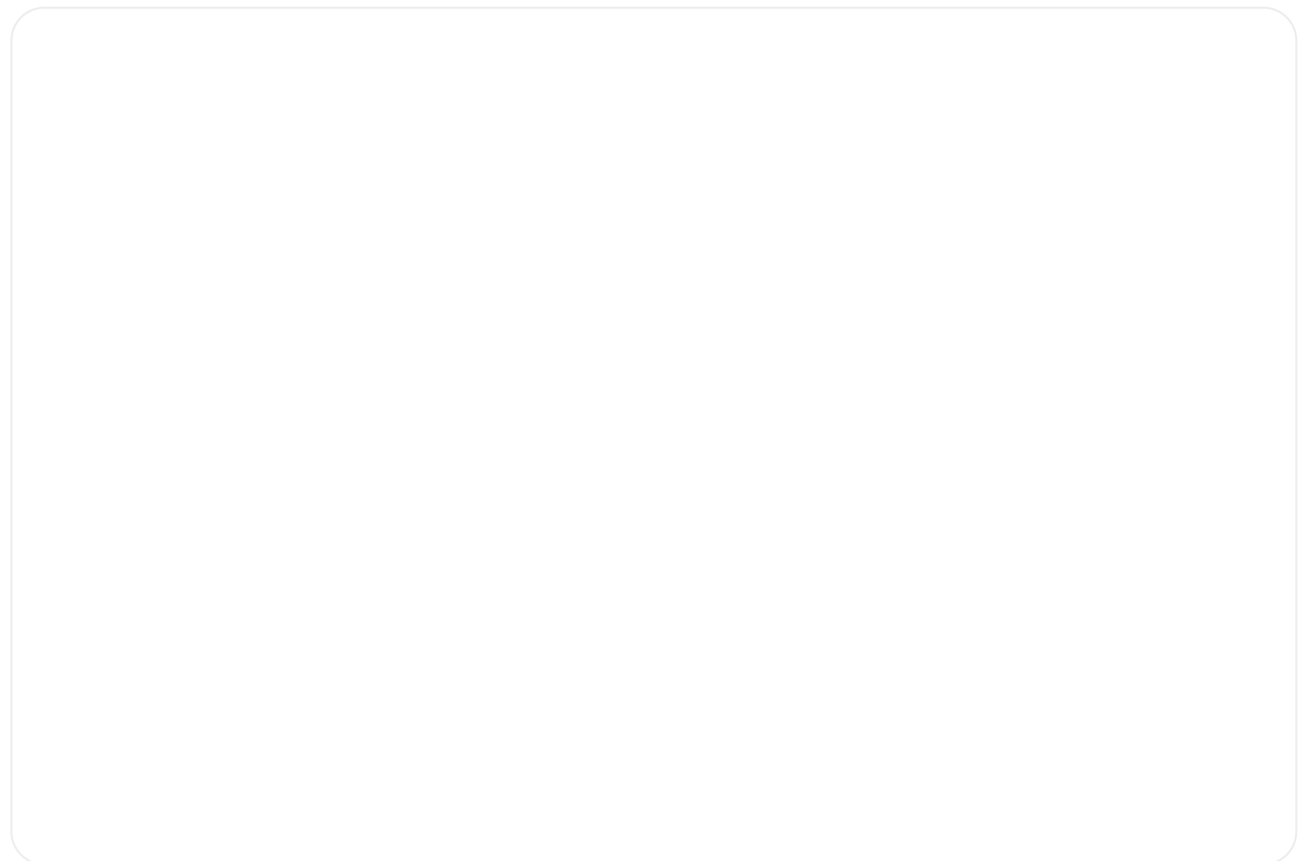
$$\text{实际值} = (\text{十进制值} * \text{Factor}) + \text{Offset}$$

Factor为1的物理量

由于Lane_Type和Quality的Factor为1，Offset为0，因此十进制值为多少，实际物理量即为多少。

从图中就能直接看出Quality这个信号占据两个位，二进制数11，换算为十进制是3（ $1*2 + 1*1$ ）；Lane_Type占据四个位，二进制数为0010，换算为十进制是2（ $0*8 + 0*4 + 1*2 + 0*1$ ）。

所以这一帧信号表示此时的左车道线Lane_Type值为2，Quality值为3。对于整数值，通信双方可以约定规则，比如Mobileye就规定了，Quality为0或者1时表示车道线的置信度较低，不推荐使用此时的值；2表示置信度中等，3表示置信度较高，请放心使用。



对于Factor不为1的物理量，比如Position，需要使用移位的方法进行解析，但解析公式保持不变。以百度 Apollo提供的源码为例进行讲解。

这里的bytes即为CAN消息中的Data，首先将Position信号所在的行取出来，将第1行的8个bool值存储在变量t1中，将第二行的8个bool值存储在变量t0中。由于在这条CAN消息中，Position同时占据了高8位和低8位，因此需要将第一行和第二行的所有bool位拿来计算，高8位存储在32位的变量x中，低8位存储在32位的变量t中。

现在需要将高8位和低8位拼接，将高8位左移8位，然后与低8位求或运算，即可得到Position的二进制值。随后进行的左移16位，再右移16位的操作是为了将32位的变量x的高16位全部初始化为0。之后将x乘以Factor再加上Offset即可得到真实的Position值，给真实值加上单位meter，即可获取实际的物理量。

VCU、雷达等通过CAN总线传递信号，随着CAN的负载越来越高，很多传感器选择了其他通信方式。比如激光雷达的点云数据量太过庞大，使用的是局域网的方式进行传递；再比如GPS和惯导使用的是串口进行通信。

虽然通信方式和通信协议千差万别，但解析的方法都是一样的。

结语

apollo 开发者社区

这篇分享的内容基本上讲清楚了CAN总线消息的解析过程。这是无人驾驶系统传感器驱动层的基本理论。

由于不同ID的CAN消息的结构不一样，因此在写解析代码时，需要十分仔细，否则会给后续处理带来想不到的Bug。

本文部分内容参考链接：

* 《无人驾驶技术入门（九）| 与生俱来的VCU信号》

【<https://zhuanlan.zhihu.com/p/36654874>】

* 《无人驾驶，个人如何研究？》

【<https://www.zhihu.com/question/20210846/answer/215490332>】