无人驾驶技术入门（十一）
无人驾驶中的 CAN 消息解析

陈 光

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

“传感器层”在前十期的技术入门分享中已经介绍过了，这次主要介绍的是“驱动层”相关的内容。

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，解析出想要的数据。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。

1 认识 CAN 消息

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));
  }
```


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：

1. 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。

2. 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。

3. 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

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。

4. **Timestamp**

CAN 消息的时间戳。

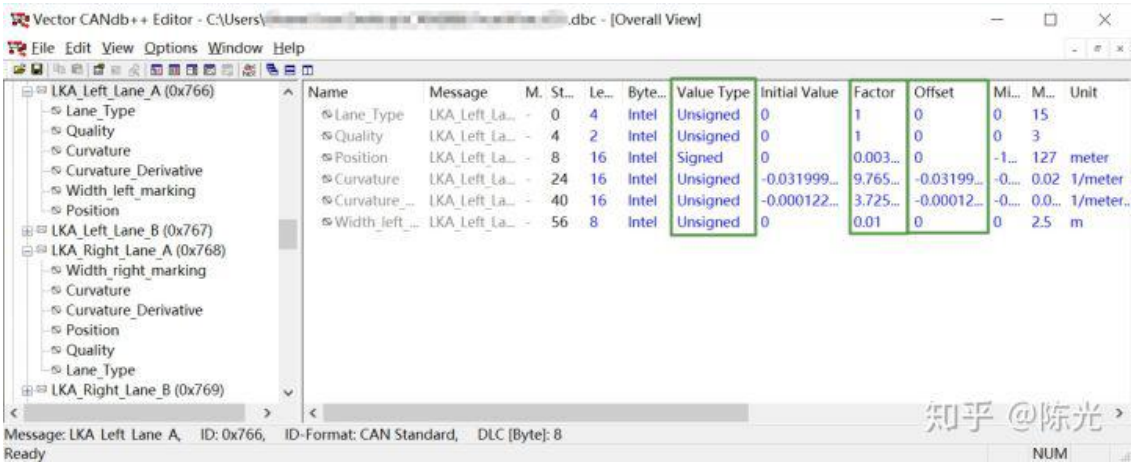
时间戳表示的是收到该 CAN 消息的时刻。通过连续多帧的时间戳，可以计算出 CAN 消息的发送周期，也可以用于判断 CAN 消息是否被持续收到。

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。

2 认识 CAN 协议

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。

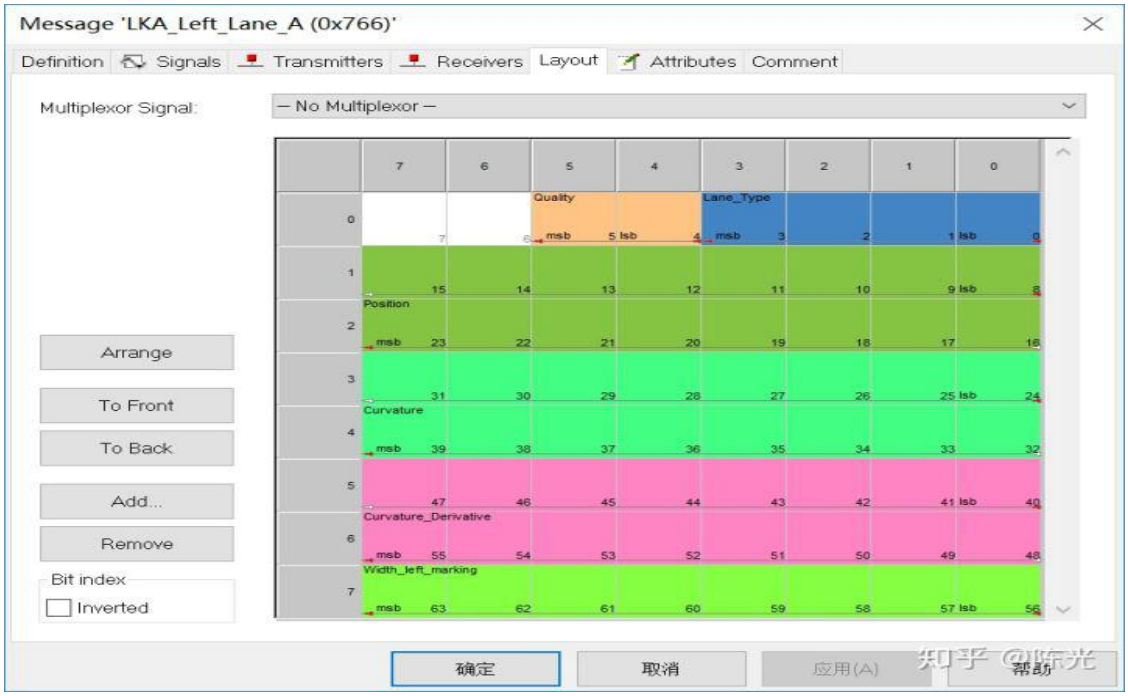
1. Value Type（Unsigned 或 Signed）

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，比如说曲率。

2. 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。

3 解析 CAN 信号

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

每个信号物理量的计算公式为：

实际值 = （十进制值 * Factor） + Offset

1. 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。

2. Factor 为小数的物理量

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。

```
// config detail: {'name': 'position', 'offset': 0.0, 'precision': 0.00390625,
// 'len': 16, 'f_type': 'value', 'is_signed_var': True, 'physical_range':
// '[-128|127]', 'bit': 8, 'type': 'double', 'order': 'intel', 'physical_unit':
// '"meter"'}
double Lka766::position(const uint8_t* bytes, int32_t length) const {
  Byte t0(bytes + 2);
  int32_t x = t0.get_byte(0, 8);

  Byte t1(bytes + 1);
  int32_t t = t1.get_byte(0, 8);
  x <<= 8;
  x |= t;

  x <<= 16;
  x >>= 16;

  return x * 0.003906;
}
```

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。

4 与 CAN 类似的通信协议

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。

5 结语

好了\(^o^)/~，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。

文章中使用的软件 CANdb++ Editor 下载+安装方法及 Mobileye 车道线的原始 dbc 文件，可以关注我的公众号：自动驾驶干货铺，在后台回复关键字"CAN"获取。如果你想获取更多 Mobileye 的信息或技术资料，可以在值乎向我提问。

本文原载：知乎号“陈光”，作者授权转载。

临菲信息技术港

临菲信息技术港公众号

临菲学堂