第五课 STM32 和 ROS 之间的通信建立

1. 通信内容和原理

底盘驱动板和 ROS 的通信是基于串口通信的，也可以理解为 STM32 和树莓派之间的通信，甚至也可以认为是 STM32 和 ubuntu 之间的通信。这样的表达似乎能够提升我们的理解，只是这份通信代码需要使用 ROS 软件框架来封装，那么又如何理解 ROS 软件框架呢？其实就是在 C++、python 上使用一些开源的机器人库而已，后面的课程会详细的学习 ROS 的相关课程。本节课程需要学习如何将机器人的底层里程计、电池状态、IMU 姿态信息等数据提交到 ROS 系统当中。如图 1-1-0 所示：

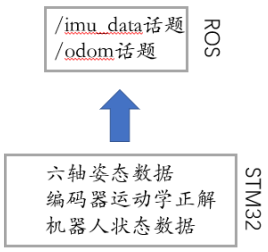

图 1-1-0 通信框架

如何定义 STM32 和 ROS 之间的通信协议呢？需要考虑传输的速度、大小、数据内容等问题，然后进行统一的封装，最后调用串口发送函数将封装好的数据按照字节序发送出去。在数据通信协议的定义上，使用一个 union 类型来定义两个相同大小的域，一个是数据收发的缓存区数据 buffer，另一个是数据域结构体。我们知道 union 类型只会分配最大域大小的内存空间。当两个域大小相同时（同时约定内存对齐为 1 个字节），这时只需要通过串口去操作收发缓存数据，即可完成数据的透传过程。

```
#pragma pack(1)                 //数据按照1字节对齐方式存取
typedef struct __Mpu6050_Str_   //IMU的数据格式
{
    short X_data;
    short Y_data;
    short Z_data;
}Mpu6050_Str;

typedef union _Upload_Data_          //通信协议的定义
{
    unsigned char buffer[PROTOCL_DATA_SIZE];    //数据收发缓存区
    struct _Sensor_Str_
    {
        unsigned int Header;                //通信协议的数据校验头部
        float X_speed;                      //基于右手坐标系的正解速度信息，  X Y Z
        float Y_speed;
        float Z_speed;
        float Source_Voltage;               //机器人的电池电压

        Mpu6050_Str Link_Accelerometer;     //IMU的三轴加速的原始数据
        Mpu6050_Str Link_Gyroscope;         // IMU的三轴角速度的原始数据

        unsigned char End_flag;             //通信协议的数据校验尾部
    }Sensor_Str;
}Upload_Data;
#pragma pack(4)          //数据按照4字节对齐方式存取
```

图 1-1-0 通信协议

2. STM32 的串口通信

串行接口是一种可以将接受来自 CPU 的并行数据字符转换为连续的串行数据流发送出去，同时可将接受的串行数据流转换为并行的数据字符供给 CPU 的器件。一般完成这种功能的电路，我们称为串行接口电路。

串口通讯的数据包由发送设备通过自身的 TXD 接口传输到接收设备的 RXD 接口，通讯双方的数据包格式要规约一致才能正常收发数据。STM32 中串口异步通信需要定

义的参数：起始位、数据位、奇偶校验位、停止位、波特率设置。UART 串口通信的数据包以帧为单位，常用的帧结构为：1 位起始位+8 位数据位+1 位奇偶校验位（可选）+1 位停止位。如下图 2-1-0 所示：对于两个进行通信的端口，这些参数必须匹配。

图249　字长设置

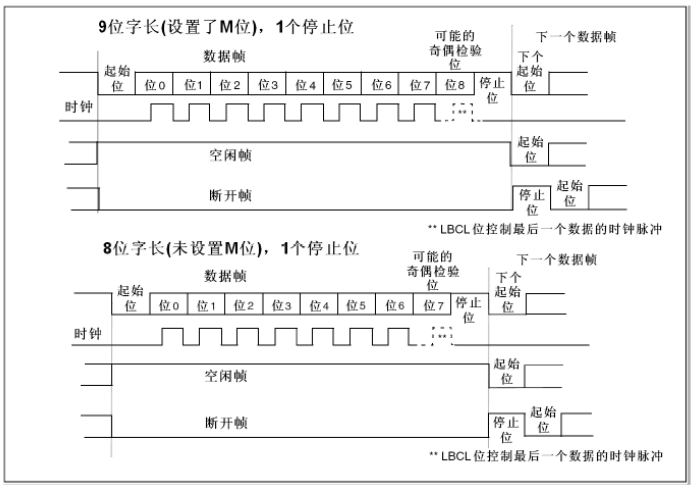

图 2-1-0 串口通信字长设置

串口通信（Serial Communications）的概念非常简单，串口按位（bit）发送和接收字节。尽管比按字节（byte）的并行通信慢，但是串口可以在使用一根线发送数据的同时用另一根线接收数据。它很简单并且能够实现远距离通信。通信使用 3 根线完成，分别是地线(GND)、发送(TX)、接收(RX)。由于串口通信是异步的，端口能够在一根线上发送数据同时在另一根线上接收数据。

在底盘驱动板中，初始化了 USART1 的串口接收和发送，用于透传数据到树莓派的 ROS 系统当中，在源码的 Huanyu_usart.c 文件中可以看到串口的初始化配置。

```
/*
 @ describetion: USART1 initialization function
 @ param: bound->115200
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void Huanyu_Usart1_Init(u32 bound)
*/
void Huanyu_Usart1_Init(u32 bound)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);

    GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource9 ,GPIO_AF_USART1);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
```

```
    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

图 2-1-0 通信通信配置函数

配置好串口的收发，停止位、数据位、校验位、数据位后，我们便可以在任何时刻通过配置好的串口发送数据,也可以在任何时候传力串口接收中断中已经成功接收到的数据。

```
/*
 @ describetion:usart send a char data
 @ param: b:data
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void USART1_SendChar(unsigned char b)
*/
void USART1_SendChar(unsigned char b)
{
    while (USART_GetFlagStatus(USART1,USART_FLAG_TC) == RESET);
           USART_SendData(USART1,b);
}
```

图 2-1-1 串口发送一个字节函数

```
/*
 @ describetion:Send Data To ubuntu pc
 @ param: b:data
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void Huanyu_SendTo_UbuntuPC()
*/
void Huanyu_SendTo_UbuntuPC()
{
    unsigned char i = 0;
    Send_Data.Sensor_Str.Header   = PROTOCOL_HEADER;
    Send_Data.Sensor_Str.End_flag = PROTOCOL_END;

    Send_Data.Sensor_Str.X_speed = (Left_moto.Current_Speed + Right_moto.Current_Speed)/2.0f;
    Send_Data.Sensor_Str.Y_speed = 0.0;
    Send_Data.Sensor_Str.Z_speed = (Left_moto.Current_Speed - Right_moto.Current_Speed)/Base_Width;

    Send_Data.Sensor_Str.Source_Voltage = Source_Valtage;

    for(i=0; i<PROTOCL_DATA_SIZE; i++)
    {
        USART1_SendChar(Send_Data.buffer[i]);
    }
}
```

图 2-1-2 封装和发送数据函数

3. ROS 串口通信

在 ROS 平台下，设计一个串口节点，该节点订阅控制节点发来的命令主题，将命令通过串口设备发送到移动底座也可以是飞控设备;同时串口节点实时接收移动底座通过串口发送过来的传感器实时数据，并将该数据封装后以 *sensor* 主题的模式进行发布，*listenner* 节点可以实现订阅该主题。这样就实现了 ROS 与移动底座的串口通信过程。

在 ROS 开源的软件包中，有很多关于串口通信封装库，有些过于臃肿和复杂。在经过使用和对比后发现,使用 serial 串口库,既高效又稳定。下面需要在树莓派的 Ubuntu

系统中安装 serial 库，安装很简单，在树莓派上打开一个新的终端，输入以下命令：

```
sudo apt-get install ros-kinetic-serial
```

安装完成后，同样也需要在 ROS 上定义相同的通信协议。

```
#pragma pack(1)
typedef struct __Mpu6050_Str_
{
    short X_data;
    short Y_data;
    short Z_data;
}Mpu6050_Str;

typedef union _Upload_Data_
{
    unsigned char buffer[PROTOBUF_SIZE];

    struct _Sensor_Str_
    {
        unsigned int Header;
        float X_speed;
        float Y_speed;
        float Z_speed;

        float Source_Voltage;

        Mpu6050_Str Link_Accelerometer;
        Mpu6050_Str Link_Gyroscope;

        unsigned char End_flag;
    }Sensor_Str;
}Upload_Data;
#pragma pack(4)
```

在代码中配置和打开一个串口设备，但是值得注意的时，配置的所有参数都要和 STM32 的串口配置一样，包括数据位、起始位、停止位、校验位等。创建一个 cpp 文件，在代码的合适位置（或者类的构造函数中）初始化串口配置，并且打开串口。

```
try{
    Robot_Serial.setPort(this->usart_port);
    Robot_Serial.setBaudrate(this->baud_data);
    serial::Timeout to = serial::Timeout::simpleTimeout(2000);
    Robot_Serial.setTimeout(to);
    Robot_Serial.open();
}
catch (serial::IOException& e){
    ROS_ERROR_STREAM("[ZHOUXUEWEI] Unable to open port ");
}
if(Robot_Serial.isOpen()){
    ROS_INFO_STREAM("[ZHOUXUEWEI] Serial Port opened");
}else{
}
```

初始化串口后就可以通过以下函数实现和 STM32 的串口通信了：

```
Robot_Serial.write(Send_Str.buffer, sizeof(Send_Str.buffer));
```

```
Robot_Serial.read(Reciver_Str.buffer,sizeof(Reciver_Str.buffer));
```

4. 总结

本节课程我们学习了 STM32 的串口通信原理和串口通讯基本配置，也清楚了如何定义一个高效的通信协议来收发数据。在树莓派上安装了串口节点库，依赖于 serial 库实现了串口的基本配置和收发方式。