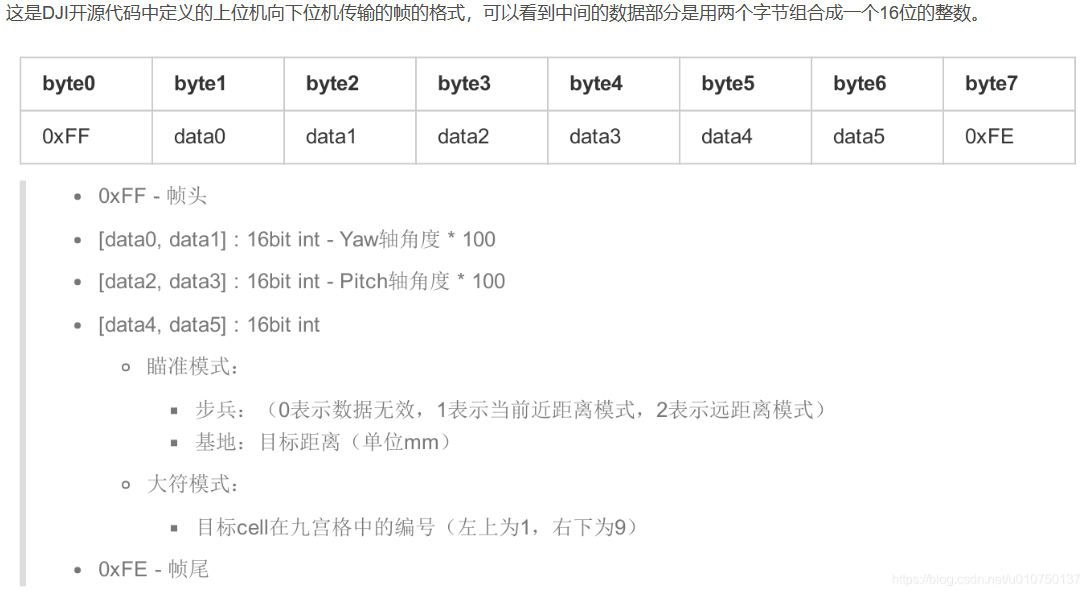
**初步想法**  
我参考了一下东南大学的哨兵通讯开源内容，准备以下方式来传输上下位机数据

各位可以参考一下，看是否有所欠缺。

采用设置指定帧头帧尾的方式来传输数据，会定义四个不同的帧头以及一个帧尾。

例如下图。

**注意：下方的byte是字节不是bit，别搞混了**



我觉得可以这样：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| 帧头 | Data1 | Data2 | Data3 | Data4 | Data5 | Data6 | 帧尾 |

第一段来传送云台控制数值，四个bytes发送两个轴角度，一个byte为发射方式和发射速度另一个byte为是否发射。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| 帧头 | Pitch轴角度 | Pitch轴角度 | Yaw轴角度 | Yaw轴角度 | 发射方式(停火/点射/连射) | 发射速度(低速/高速) | 帧尾 |

第二段来控制底盘，暂时先设定为两个bytes控制小车前后左右(小车坐标系)一个byte控制底盘摇摆或跟随，一个byte控制小车旋转。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| 帧头 | 小车纵向速度 | 小车横向速度 | 小车旋转(向左旋转/向右旋转) | 底盘跟随或底盘摇摆 | 空字节 | 空字节 | 帧尾 |

第三段为下位机返回给上位机的数据。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| 帧头 | 发射方式 | 子弹射速 | 被击打的装甲板ID | 剩余HP | 剩余HP | 暂时空余 | 帧尾 |

第四个段暂时空余，可以放入其他内容：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Byte0 | Byte1 | Byte2 | Byte3 | Byte4 | Byte5 | Byte6 | Byte7 |
| 帧头 | Data1 | Data2 | Data3 | Data4 | Data5 | Data6 | 帧尾 |

我现在大概有点明白了。

在下位机那边用枚举等玩意写一个发送和解码的代码

然后在上位机那边用ROS写一个开启/关闭串口的节点

注:

1. 我暂时不明白为什么东南大学每次只传8个字节，也许是更方便管理吧。
2. 各位可以提出一些看法，我看CSDN上说只要波特率设置得当(40000左右)，每秒通讯次数可以达到一千余次，对于100余帧的摄像头而言，这种沟通速度是足够的。
3. 他们的头文件中是这样的

struct ControlData//控制段

{

    uint8\_t    frame\_seq;           //Corresponding to the record对应于记录

    uint16\_t   shoot\_mode;          //高八位发射方式 低八位发射速度等级  (0xFFFF-记录当前角度  0xEEEE-红方  0xDDDD-蓝方  0xCCCC-通信建立  0xBBBB-Jetson为了接收数据发送帧)

    float      pitch\_dev;           //Pitch目标角度

    float      yaw\_dev;             //Yaw目标角度

    /\*  哨兵专用   \*/

    int16\_t    rail\_speed;          //目标轨道速度（哨兵用）

    uint8\_t    gimbal\_mode;         //哨兵云台攻击模式

};

struct FeedBackData

{

    uint8\_t    task\_mode;           //所需控制模式

    uint8\_t    bullet\_speed;        //射速

    /\*  哨兵专用   \*/

    uint8\_t    rail\_pos;            //所处轨道标号

    uint8\_t    shot\_armor;          //被打击装甲板标识

    uint16\_t   remain\_HP;           //剩余血量

};

/\*

 \* @Brief:  Serial communication protocol and inplement

 \*/

class Serial

{

public:

    /\*

     \* @Brief: 比赛红蓝方

     \*/

    enum TeamName

    {

        BLUE\_TEAM       =   (uint16\_t)0xDDDD,

        RED\_TEAM        =   (uint16\_t)0xEEEE

    };

    /\*

     \* @Brief: control frame mode

     \*/

    enum ControlMode

    {

        SET\_UP          =   (uint16\_t)0xCCCC,

        RECORD\_ANGLE    =   (uint16\_t)0xFFFF,

        REQUEST\_TRANS   =   (uint16\_t)0xBBBB

    };

    /\*

     \* @Brief: 发射方式

     \*/

    enum ShootMode

    {

        NO\_FIRE         =   (uint16\_t)(0x00<<8),//不发射

        SINGLE\_FIRE     =   (uint16\_t)(0x01<<8),//点射

        BURST\_FIRE      =   (uint16\_t)(0x02<<8) //连发

    };

    /\*

     \* @Brief: 发射速度

     \*/

    enum BulletSpeed

    {

        HIGH\_SPEED      =   (uint16\_t)(0x01),   //高速

        LOW\_SPEED       =   (uint16\_t)(0x02)    //低速

    };

    /\*

     \* @Breif:所需控制模式

     \*/

    enum TaskMode

    {

        NO\_TASK         =   (uint8\_t)(0x00),    //手动控制

        SMALL\_BUFF      =   (uint8\_t)(0x01),    //小符模式

        BIG\_BUFF        =   (uint8\_t)(0x02),    //大符模式

        AUTO\_SHOOT      =   (uint8\_t)(0x03)     //自动射击

    };

    /\*

     \* @Brief: 哨兵云台工作模式

     \*/

    enum GimbalMode

    {

        PATROL\_AROUND   =   (uint8\_t)(0x01),    //旋转巡逻

        PATROL\_ARMOR\_0  =   (uint8\_t)(0x02),    //巡逻装甲板0

        PATROL\_ARMOR\_1  =   (uint8\_t)(0x03),    //巡逻装甲板1

        SERVO\_MODE      =   (uint8\_t)(0x04)     //伺服打击

    };

    /\* @Brief:

     \*      SYSTEM\_ERROR:   System error catched. May be caused by wrong port number,

     \*                      fragile connection between Jetson and STM, STM shutting

     \*                      down during communicating or the sockets being suddenly

     \*                      plugged out.

     \*      OJBK:         Everything all right

     \*      PORT\_OCCUPIED:  Fail to close the serial port

     \*      READ\_WRITE\_ERROR: Fail to write to or read from the port

     \*      CORRUPTED\_FRAME: Wrong frame format

     \*      TIME\_OUT:       Receiving time out

     \*/

    enum ErrorCode

    {

        SYSTEM\_ERROR    = 1,

        OJBK            = 0,

        PORT\_OCCUPIED   = -1,

        READ\_WRITE\_ERROR= -2,

        CORRUPTED\_FRAME = -3,

        TIME\_OUT        = -4

    };