

目 录

1 通信协议格式.....	2
1.1 FrameHeader 格式.....	2
1.1.1 帧头结构.....	2
1.2 命令码 ID 说明	2
1.2.1 比赛进程信息.....	2
1.2.2 实时伤害信息.....	4
1.2.3 实时射击信息.....	4
2.附件-CRC 校验代码.....	6

1 通信协议格式

FrameHeader(4-Byte)	CmdID(2-Byte)	Data(n-Byte)	FrameTail(2-Byte, CRC16)
---------------------	---------------	--------------	-----------------------------

1.1 FrameHeader 格式

1.1.1 帧头结构

表 1.1-1 帧头格式

SOF	DataLength	CRC8
1-Byte	2-Byte	1-Byte

表 1.1-2 帧头详细定义

域	偏移位置	大小（字节）	详细描述
SOF	0	1	数据帧起始字节，固定值为 0xA5。
DataLength	1	2	数据帧内 Data 长度。
CRC8	3	1	帧头 CRC 校验

1.2 命令码 ID 说明

命令码	功能介绍
0x0001	比赛进程信息
0x0002	实时血量变化数据
0x0003	实时射击数据

1.2.1 比赛进程信息（0x0001）

数据	字节偏移	大小	说明
	0	4	比赛剩余时间（从倒计时三分钟开始计算，单位 s）
	4	2	机器人剩余血量
	6	4	实时底盘输出电压（单位 V）
	10	4	实时底盘输出电流（单位 A）
	14	4	四个小符点状态，可能是： 0x0 = 无效 0x01 = 防御符 0x04 = 加血符 0x08 = 加弹符
	18	1	大符点 1 状态： 0x0 = 无效

裁判系统协议（学生接口）

		0x01 = 有效，无机器人占领 0x02 = 机器人正在占领 0x03 = 已占领
19	1	大符点 2. 其值意义同大符点 1
20	1	传送带以及停机坪状态： 0: 1bits: 0 号传送带状态： 0x0 = 传送带停止 0x1 = 传送带正转 0x2 = 传送带反转 2: 3bits: 1 号传送带状态，其值意义同传送带 0 4bits: 0 号停机坪状态： 0: 未检测到飞行器 1: 检测到飞行器 5bits: 1 号停机坪状态 6bits: 2 号停机坪状态 7bits: 3 号停机坪状态
21	17	GPS 状态，见 tGpsData 结构体定义

```

typedef enum{
    BUFF_TYPE_NONE,                //无效
    BUFF_TYPE_ARMOR = 0x01,        //防御符
    BUFF_TYPE_SUPPLY = 0x04,       //加血符
    BUFF_TYPE_BULLFETS= 0x08,     //加弹符
}eBuffType;

typedef __packed struct
{
    Uint8_t flag;                  //0 无效， 1 有效
    uint32_t x;
    uint32_t y;
    uint32_t z;
    uint32_t compass;
}tGpsData;

typedef __packed struct
{
    Uint32_t remainTime;
    uint16_t remainLifeValue;
    float  realChassisOutV;
    float  realChassisOutA;
    uint8_t runeStatus[4];
    uint8_t bigRune0Status;
    uint8_t bigRune1status;
    uint8_t conveyorBelts0:2;
    
```

```

uint8_t conveyorBelts1:2;
uint8_t parkingApron0:1;
uint8_t parkingApron1:1;
uint8_t parkingApron2:1;
uint8_t parkingApron3:1;
tGpsData gpsData;
}tGameInfo;
    
```

注：此数据包的发送频率为 50Hz

1.2.2 实时血量变化信息(0x0002)

字节偏移 大小 说明

数据	字节偏移	大小	说明
	0	1	0-3bits: 若变化类型为装甲伤害时：标识装甲 ID 0x00: 0号装甲面（前） 0x01: 1号装甲面（左） 0x02: 2号装甲面（后） 0x03: 3号装甲面（右） 0x04: 4号装甲面（上1） 0x05: 5号装甲面（上2） 其它暂作保留 4-7bits: 血量变化类型 0x0: 装甲伤害（受到攻击） 0x1: 子弹超速扣血 0x2: 子弹超频扣血 0x3: 功率超限 0x4: 模块离线扣血 0x6: 普通犯规扣血 0xa: 获取加血神符
	1	2	血量变化值

```

typedef __packed struct
{
    uint8_t weakId:4;
    uint8_t way:4;
    uint16_t value;
}tRealBloodChangedData;
    
```

1.2.3 实时射击信息(0x0003)

字节偏移 大小 说明

数据	字节偏移	大小	说明
	0	4	子弹实时射速（m/s）
	4	4	子弹实时射频（发/s）
	8	4	高尔夫实时射速(m/s 英雄机器人)
	12	4	高尔夫实时射频(发/s 英雄机器人)

裁判系统协议（学生接口）

```
typedef __packed struct
{
    float realBulletShootSpeed;
    float realBulletShootFreq;
    float realGolfShootSpeed;
    float realGolfShootFreq;
}tRealShootData;
```

2.附件-CRC 校验代码

```
//crc8 generator polynomial:G(x)=x8+x5+x4+1
const unsigned char CRC8_INIT = 0xff;
const unsigned char CRC8_TAB[256] =
{
    0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83, 0xc2, 0x9c, 0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41,
    0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e, 0x5f, 0x01, 0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc,
    0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0, 0xe1, 0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62,
    0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63, 0x3d, 0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff,
    0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79, 0x9b, 0xc5, 0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07,
    0xdb, 0x85, 0x67, 0x39, 0xba, 0xe4, 0x06, 0x58, 0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a,
    0x65, 0x3b, 0xd9, 0x87, 0x04, 0x5a, 0xb8, 0xe6, 0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24,
    0xf8, 0xa6, 0x44, 0x1a, 0x99, 0xc7, 0x25, 0x7b, 0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9,
    0x8c, 0xd2, 0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f, 0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd,
    0x11, 0x4f, 0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92, 0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50,
    0xaf, 0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c, 0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
    0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1, 0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d, 0x73,
    0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49, 0x08, 0x56, 0xb4, 0xea, 0x69, 0x37, 0xd5, 0x8b,
    0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4, 0x95, 0xcb, 0x29, 0x77, 0xf4, 0xaa, 0x48, 0x16,
    0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a, 0x2b, 0x75, 0x97, 0xc9, 0x4a, 0x14, 0xf6, 0xa8,
    0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7, 0xb6, 0xe8, 0x0a, 0x54, 0xd7, 0x89, 0x6b, 0x35,
};

unsigned char Get_CRC8_Check_Sum(unsigned char *pchMessage,unsigned int
dwLength,unsigned char ucCRC8)
{
    unsigned char ucIndex;

    while (dwLength--)
    {
        ucIndex = ucCRC8^(*pchMessage++);
        ucCRC8 = CRC8_TAB[ucIndex];
    }

    return(ucCRC8);
}

/*
** Descriptions: CRC8 Verify function
** Input:        Data to Verify,Stream length = Data + checksum
** Output:       True or False (CRC Verify Result)
*/
```

```

*/
unsigned int Verify_CRC8_Check_Sum(unsigned char *pchMessage, unsigned int dwLength)
{
    unsigned char ucExpected = 0;

    if ((pchMessage == 0) || (dwLength <= 2)) return 0;

    ucExpected = Get_CRC8_Check_Sum (pchMessage, dwLength-1, CRC8_INIT);

    return ( ucExpected == pchMessage[dwLength-1] );
}

/*
** Descriptions: append CRC8 to the end of data
** Input:          Data to CRC and append,Stream length = Data + checksum
** Output:         True or False (CRC Verify Result)
*/
void Append_CRC8_Check_Sum(unsigned char *pchMessage, unsigned int dwLength)
{
    unsigned char ucCRC = 0;

    if ((pchMessage == 0) || (dwLength <= 2)) return;

    ucCRC = Get_CRC8_Check_Sum ( (unsigned char *)pchMessage, dwLength-1, CRC8_INIT);

    pchMessage[dwLength-1] = ucCRC;
}

uint16_t CRC_INIT = 0xffff;
const uint16_t wCRC_Table[256] =
{
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
    0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdced, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,

```

```

0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xdd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};

/*
** Descriptions: CRC16 checksum function
** Input:          Data to check,Stream length, initialized checksum
** Output:         CRC checksum
*/
uint16_t  Get_CRC16_Check_Sum(uint8_t  *pchMessage,uint32_t  dwLength,uint16_t
wCRC)
{
    Uint8_t  chData;
    if (pchMessage == NULL)
    {
        return 0xFFFF;
    }

    while(dwLength--)
    {
        chData = *pchMessage++;
        (wCRC) = ((uint16_t)(wCRC) >> 8) ^ wCRC_Table[((uint16_t)(wCRC) ^
(uint16_t)(chData)) & 0x00ff];
    }

    return wCRC;
}

```



```

}

/*
** Descriptions: CRC16 Verify function
** Input:      Data to Verify,Stream length = Data + checksum
** Output:     True or False (CRC Verify Result)
*/
uint32_t Verify_CRC16_Check_Sum(uint8_t *pchMessage, uint32_t dwLength)
{
    uint16_t wExpected = 0;

    if ((pchMessage == NULL) || (dwLength <= 2))
    {
        return __FALSE;
    }
    wExpected = Get_CRC16_Check_Sum ( pchMessage, dwLength - 2, CRC_INIT);

    return ((wExpected & 0xff) == pchMessage[dwLength - 2] && ((wExpected >> 8) & 0xff)
== pchMessage[dwLength - 1]);
}

/*
** Descriptions: append CRC16 to the end of data
** Input:      Data to CRC and append,Stream length = Data + checksum
** Output:     True or False (CRC Verify Result)
*/
void Append_CRC16_Check_Sum(uint8_t * pchMessage,uint32_t dwLength)
{
    uint16_t wCRC = 0;

    if ((pchMessage == NULL) || (dwLength <= 2))
    {
        return;
    }
    wCRC = Get_CRC16_Check_Sum ( (U8 *)pchMessage, dwLength-2, CRC_INIT );

    pchMessage[dwLength-2] = (U8)(wCRC & 0x00ff);
    pchMessage[dwLength-1] = (U8)((wCRC >> 8)& 0x00ff);
}

```

注：uart4 通信配置，波特率 115200,数据位 8,停止位 1，检验位无，流控制无