



Q1 Series

产品手册

Revision 0.50

2019/01

COPYRIGHT

THE INFORMATION CONTAINED HERE IS PROPRIETARY TECHNICAL INFORMATION OF FOHEART CO., LTD. TRANSMITTING, REPRODUCTION, DISSEMINATION AND EDITING OF THIS DOCUMENT AS WELL AS UTILIZATION OF THE CONTENT ARE FORBIDDEN WITHOUT PERMISSION. OFFENDERS WILL BE HELD LIABLE FOR PAYMENT OF DAMAGES. ALL RIGHTS ARE RESERVED IN THE EVENT OF A PATENT GRANT OR REGISTRATION OF A UTILITY MODEL OR DESIGN.

GENERAL NOTES

FOHEART OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. FOHEART MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. FOHEART DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

For technical support, please visit:

<http://www.foheart.com/support.html> or

<http://www.foheart.com/question.html>

北京总部

Tel: (+86)010-56106165

Email: contact@foheart.com



地址：北京市海淀区黑山扈路红山口 8 号 D2-南-3 号

邮编：100091

Copyright © FOHEART Co., Ltd. 2015-2019. All rights reserved.

Revision History

Revision No.	Date	Description	Author(s)
0.10	Feb. 15, 2017	First Draft	HMX
0.20	Oct. 27, 2017	修改了 2.4.1 配置文件的存放顺序。 修改了 2.4.2 算法配置区为 512 字节。	HMX
0.40	Jan. 30, 2018	添加加速度计、磁力计校准等新指令。	HMX
0.50	Jan. 2019	添加了安装尺寸、订货信息、更改了部分指令格式	HMX

简介

MOTIONMARS™ Q1 系列是孚心科技推出的 IMU/VRU，外壳使用高强度铝合金 CNC 加工，可有效抵御外部粉尘、液体泼溅等对传感器的伤害。数据接口方面提供 USB/UART 接口以便连接用户现有硬件，UART 接口数据速率最高可达 3Mbps 以减小接收端的数据采集延迟。

通过配套客户端软件 MOTIONMARS™ 2，用户可通过图形化界面对模组性能进行整体评估，通过指令可设置模组的数据发送类型、速率等多种参数。

每个模组出厂时均进行了标定，具有唯一的标定参数，包含加速度计、陀螺仪的零偏误差、轴间误差等多种信息，免去了用户在使用之前的标定等工作。

对于产品尺寸敏感的用户我们还提供 PLCC44 标准封装的核心板模组，功能与铝合金外壳包装模组完全一致。

提供 Arduino 参考程序^{注1} 帮助用户在数分钟内快速完成对模组的评估，方便快速熟悉与接入模组到现有嵌入式系统中。对于专业人员我们也提供 ANSI-C 语言的 SDK 即指令集参考源码文件（包含.c 与.h），只需映射相应的串口读/写函数即可完成模组的数据集成。

注 1：使用 Arduino MEGA2560 开发板，操作步骤与参考源码见第 9 章。

名词解释：

四元数：模值为 1.0，发送顺序按照 WXYZ 发送，坐标系定义见附录。

欧拉角：单位为 degree，发送顺序按照 XYZ 发送，坐标系定义见附录。

加速度值：单位为 g，发送顺序按照 XYZ 发送，坐标系定义见附录。

角速度值：单位为 degree/s，发送顺序按照 XYZ 发送，坐标系定义见附录。

磁力计值：单位为 mGauss，发送顺序按照 XYZ 发送，坐标系定义见附录。

PUBLIC

简介	3
名词解释：	4
一、 TTL 基本命令	8
1.1 查询模块固件信息.....	8
1.1.1 发送端数据帧格式.....	8
1.1.2 接收端返回数据帧格式.....	8
1.2 查询/设置加速度计参数	9
1.2.1 发送端数据帧格式.....	9
1.2.2 接收端返回数据帧格式.....	9
1.3 查询/设置陀螺仪参数	10
1.3.1 发送端数据帧格式.....	10
1.3.2 接收端返回数据帧格式.....	11
1.4 查询/设置磁力计参数	11
1.4.1 发送端数据帧格式.....	12
1.4.2 接收端返回数据帧格式.....	12
1.5 重新启动/复位	13
1.5.1 发送端数据帧格式.....	13
1.5.2 接收端返回数据帧格式.....	13
1.6 置零当前旋转.....	14
1.6.1 发送端数据帧格式.....	14
1.6.2 接收端返回数据帧格式.....	14
1.7 发送原始数据.....	15
1.7.1 发送端数据帧格式.....	15
1.7.2 接收端返回数据帧格式.....	16
1.8 恢复出厂设置.....	16
1.8.1 发送端数据帧格式.....	17
1.8.2 接收端返回数据帧格式.....	17

二、TTL 传输命令	19
2.1 设置传输参数.....	19
2.1.1 发送端数据帧格式.....	19
2.1.2 接收端返回数据帧格式.....	22
2.2 开始实时传输.....	23
2.2.1 发送端数据帧格式.....	23
2.2.2 接收端返回数据帧格式.....	23
三、硬件特性.....	27
3.1 陀螺仪参数.....	27
3.2 加速度计参数.....	27
3.3 磁力计参数.....	27
3.4 姿态解算.....	27
3.5 电气/接口参数	28
四、硬件接口.....	29
4.1 硬件接口图.....	29
4.2 USB 连接方式.....	29
4.2.1 Win10 系统驱动安装.....	29
4.2.2 Win7 系统驱动安装.....	35
4.3 TTL 连接方式	37
五、安装尺寸图.....	38
5.1 外观尺寸.....	38
六、订货信息.....	39
6.1 评估套件.....	39
七、工作流程图.....	40
八、使用流程.....	41
8.1 配置流程.....	41
8.2 姿态数据包解析流程.....	41
九、使用 Arduino 评估模组	47

附录 1 帧格式.....	58
附录 2 CRC16 校验源码.....	58
附录 3 坐标系定义.....	62

PUBLIC

一、TTL 基本命令

1.1 查询模块固件信息

命令字/Code	指令	描述
0x01 (Write)	REQFW	查询节点固件信息
0x02 (Read)		

1.1.1 发送端数据帧格式

数据帧格式
<0x01>+<1Byte:Code>+<CRC16>
C/C++结构体参考
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; 5. uint16_t crc16; 6. }ReqFw_UART_PC2ND; </pre>

1.1.2 接收端返回数据帧格式

数据帧格式
<地址 0x01>+<Code: 0x02>+[4x3Bytes: CPUID]+[4Bytes: Serial Number]+[3Bytes: 固件版本号]+[2Bytes: CPU 频率, 单位 MHz]+[6Bytes: 保留信息]+<CRC16>
C/C++结构体参考
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; 5. uint32_t cpuid[3]; 6. uint32_t sn; 7. uint8_t fwversion[3]; 8. uint16_t cpuClkMHz; 9. uint8_t resv[6]; 10. uint16_t crc16; 11. }ReqFw_UART_ND2PC; </pre>

1.2 查询/设置加速度计参数

命令字	指令	描述
0x03 (Write)	ACCELINFO	查询加速度计参数
0x04 (Read)		

1.2.1 发送端数据帧格式

数据帧格式
<地址 0x01>+<0x03>+<CRC16>
C/C++结构体参考
<pre> 1. typedef struct { 2. uint8_t addr; 3. uint8_t code; 4. /*0x01:请求读取零偏; 5. 0x02: 请求写入零偏; 6. 0x03: 请求写入零偏, 并保存到外置存储器, 永久使用*/ 7. uint8_t cmd; 8. /*倍率 1<<16*/ 9. int32_t bias[3]; 10. /*倍率 1<<24*/ 11. int32_t scall[3][3]; 12. uint16_t crc16; 13. }AccelInfo_UART_PC2ND; </pre> 

1.2.2 接收端返回数据帧格式

数据帧格式
<地址 0x01>+<0x04>+<1Byte: status>+<3x4Bytes: 零偏>+<3x3x4Bytes: 比例因子>+<CRC16>
C/C++结构体参考
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; </pre>

```

5.      /*
6.       1、若上位机发送 0x01:
7.           (1) 返回 0x01 代表读取零偏成功。
8.           (2) 返回 0xFF 代表读取零偏失败。
9.       2、若上位机发送 0x02:
10.      (1) 返回 0x01 写入零偏成功。
11.      (2) 返回 0xFF 代表写入零偏失败。
12.      3、若上位机发送 0x03:
13.          (1) 返回 0x01 写入零偏，并保存到外置存储器成功。
14.          (2) 返回 0xFF 写入零偏失败。
15.      */
16.     uint8_t status;
17.     /*倍率 1<<16*/
18.     int32_t bias[3];
19.     /*倍率 1<<24*/
20.     int32_t      scall[3][3];
21.     uint16_t      crc16;
22. }AccelInfo_UART_ND2PC;

```

1.3 查询/设置陀螺仪计参数

命令字	指令	描述
0x05 (Write)	GYROINFO	查询陀螺仪参数
0x06 (Read)		

1.3.1 发送端数据帧格式

数据帧格式
<地址 0x01>+<0x05>+<CRC16>
C/C++结构体参考
<pre> 1. typedef struct { 2. uint8_t addr; 3. uint8_t code; 4. /*0x01:请求读取零偏; 5. 0x02: 请求写入零偏; 6. 0x03: 请求写入零偏，并保存到外置存储器，永久使用*/ 7. uint8_t cmd; </pre>

```

8.     /*倍率 1<<16*/
9.     int32_t bias[3];
10.    /*倍率 1<<24*/
11.    int32_t      scall[3][3];
12.    uint16_t     crc16;
13. }GyroInfo_UART_PC2ND;

```

1.3.2 接收端返回数据帧格式

数据帧格式

<地址 0x01>+<0x06>+<1Byte: status>+<3x4Bytes: 零偏>+<3x3x4Bytes: 比例因子>+<CRC16>

C/C++结构体参考

```

1.  typedef struct
2.  {
3.     uint8_t addr;
4.     uint8_t code;
5.     /*
6.      1、若上位机发送 0x01:
7.          (1) 返回 0x01 代表读取零偏成功。
8.          (2) 返回 0xFF 代表读取零偏失败。
9.      2、若上位机发送 0x02:
10.         (1) 返回 0x01 写入零偏成功。
11.         (2) 返回 0xFF 代表写入零偏失败。
12.      3、若上位机发送 0x03:
13.          (1) 返回 0x01 写入零偏，并保存到外置存储器成功。
14.          (2) 返回 0xFF 写入零偏失败。
15.     */
16.     uint8_t status;
17.     /*倍率 1<<16*/
18.     int32_t bias[3];
19.     /*倍率 1<<24*/
20.     int32_t      scall[3][3];
21.     uint16_t     crc16;
22. }GyroInfo_UART_ND2PC;

```

1.4 查询/设置磁力计参数

命令字	指令	描述
-----	----	----

0x07 (Write)	MAGINFO	查询磁力计参数
0x08 (Read)		

1.4.1 发送端数据帧格式

数据帧格式

<地址 0x01>+<0x07>+<CRC16>

C/C++结构体参考

```

1. typedef struct {
2.     uint8_t addr;
3.     uint8_t code;
4.     /*0x01:请求读取零偏;
5.      0x02: 请求写入零偏;
6.      0x03: 请求写入零偏, 并保存到外置存储器, 永久使用*/
7.     uint8_t cmd;
8.     /*倍率 1<<16*/
9.     int32_t bias[3];
10.    /*倍率 1<<24*/
11.    int32_t scall[3][3];
12.    uint16_t crc16;
13. }MagInfo_UART_PC2ND;

```

1.4.2 接收端返回数据帧格式

数据帧格式

<地址 0x01>+<0x08>+<1Byte: status>+<3x4Bytes: 零偏>+<3x3x4Bytes: 比例因子>+<CRC16>

C/C++结构体参考

```

1. typedef struct
2. {
3.     uint8_t addr;
4.     uint8_t code;
5.     /*
6.      1、若上位机发送 0x01:
7.          (1) 返回 0x01 代表读取零偏成功。
8.          (2) 返回 0xFF 代表读取零偏失败。
9.      2、若上位机发送 0x02:
10.         (1) 返回 0x01 写入零偏成功。
11.         (2) 返回 0xFF 代表写入零偏失败。

```

```

12.      3、若上位机发送 0x03:
13.          (1) 返回 0x01 写入零偏，并保存到外置存储器成功。
14.          (2) 返回 0xFF 写入零偏失败。
15.      */
16.      uint8_t status;
17.      /*倍率 1<<16*/
18.      int32_t bias[3];
19.      /*倍率 1<<24*/
20.      int32_t    scall[3][3];
21.      uint16_t   crc16;
22. }MagInfo_UART_ND2PC;

```

1.5 重新启动/复位

此条命令控制模块重启，模块收到复位指令后，首先返回数据包，然后自动重启，重启后模块返回上一次开机时的运行状态。

命令字	指令	描述
0x09 (Write)	REBOOT	
0x0A (Read)		模块重启

1.5.1 发送端数据帧格式

数据帧格式

<地址 0x01>+<0x09>+<CRC16>

C/C++结构体参考

```

1.  typedef struct
2.  {
3.      uint8_t      addr;
4.      uint8_t      code;
5.      uint16_t     crc16;
6.  }Reboot_UART_PC2ND;

```

1.5.2 接收端返回数据帧格式

数据帧格式

<地址 0x01>+<0x0A>+<CRC16>

C/C++结构体参考

```

1.  typedef struct

```

```

2. {
3.     uint8_t      addr;
4.     uint8_t      code;
5.     uint16_t     crc16;
6. }Reboot_UART_ND2PC;

```

1.6 置零当前旋转

无论当前旋转值为多少，以当前值为 0 旋转，传输求差值之后的旋转量，即在当前旋转值的基础上旋转了多少。

命令字	指令	描述
0x0B (Write)	SETZERO	置零当前旋转
0x0C (Read)		

1.6.1 发送端数据帧格式

数据帧格式

<地址 0x01>+<0x0B>+<1Byte:Command>+<CRC16>

Command	含义
0x01	开启置零当前旋转
0x02	退出置零当前旋转（开机默认为此状态）

C/C++结构体参考

```

1. typedef struct
2. {
3.     uint8_t      addr;
4.     uint8_t      code;
5.     /*cmd :
6.      0x01:set current quat to 1
7.      0x02:reset current quat
8.     */
9.     uint8_t      cmd;
10.    uint16_t     crc16;
11. }SetZero_UART_PC2ND;

```

1.6.2 接收端返回数据帧格式

数据帧格式

<地址 0x01>+<0x0C>+<1Byte: Status>+<CRC16>

Status	说明
0x01	执行成功
0xFF	执行失败

C/C++结构体参考

```

1. typedef struct
2. {
3.     uint8_t addr;
4.     uint8_t code;
5.     /*status:
6.      0x01:execute success.
7.      */
8.     uint8_t status;
9.     uint16_t crc16;
10. }SetZero_UART_ND2PC;

```

1.7 发送原始数据

请求是否发送原始数据，

设置加速度计、陀螺仪、磁力计是否输出未经校准的原始值。

6Dof 及 9Dof 融合产生姿态四元数，使用的是经过校准的原始数据，不受此条指令的影响。

什么是原始值：芯片的原始输出，未经过校准的数据，一般偏移较大。

命令字	指令	描述
0x0D (Write)	RAWDATA	置零当前旋转
0x0E (Read)		

1.7.1 发送端数据帧格式

数据帧格式

<地址 0x01>+<1Byte: Code>+<1Byte: Command>+<CRC16>

Command	含义
0x01	设置为发送未经校准的原始数据
0x02	设置为发送经过校准的数据（开机默认为此状态）

C/C++结构体参考

```

1. typedef struct
2. {
3.     uint8_t      addr;
4.     uint8_t      code;
5.     uint8_t      cmd;
6.     uint16_t     crc16;
7. }RawData_UART_PC2ND;

```

用例**1.7.2 接收端返回数据帧格式****数据帧格式**

<地址 0x01>+<0x0E>+<1Byte: Status>+<CRC16>

Status	说明
0x01	执行成功
0xFF	执行失败

C/C++结构体参考

```

1. typedef struct
2. {
3.     uint8_t      addr;
4.     uint8_t      code;
5.     uint8_t      status;
6.     uint16_t     crc16;
7. }RawData_UART_ND2PC;

```

1.8 恢复出厂设置

恢复模组的出厂设置， 默认出厂设置如下：

波特率： 115200bps

发送数据频率： 200Hz

发送数据类型：四元数+欧拉角+标定后的加速度值+标定后的角速度值+标定后的磁力计值。

命令字	指令	描述
0x0F (Write)	FACTORYRESTORE	恢复模块的出厂设置
0x10 (Read)		

1.8.1 发送端数据帧格式

数据帧格式	
<地址 0x01>+<1Byte: Code>+<1Byte: Command>+<CRC16>	
Command	含义
0x01	设置恢复出厂设置，Q1 硬件回复命令后，自动重启后生效。
0x02	设置恢复出厂设置，Q1 硬件回复命令后，暂不重启，在手动重启或断电重启后生效。
C/C++结构体参考	
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; 5. uint8_t cmd; 6. uint16_t crc16; 7. }FactoryRestore_UART_PC2ND; </pre>	

1.8.2 接收端返回数据帧格式

数据帧格式	
<地址 0x01>+<0x0E>+<1Byte: Status>+<CRC16>	
Status	说明
0x01	执行成功
0xFF	执行失败
C/C++结构体参考	
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; 5. 6. /*status: </pre>	

```
7.          0x01:命令成功执行。  
8.          0xFF: 未知错误,  
9.          */  
10.         uint8_t      status;  
11.         uint16_t     crc16;  
12. }FactoryRestore_UART_ND2PC;
```

PUBLIC

二、TTL 传输命令

2.1 设置传输参数

要求节点以指定频率实时上传姿态等数据，节点确认传输条件是否具备。

若设置参数均正确，模组回复指令的时间大约需要 80ms。

若有错误参数，模组在 5ms 内返回错误信息。

命令字	指令	描述
0x21 (Write)	RTTRANS	设置数据传输参数
0x22 (Read)		

2.1.1 发送端数据帧格式

数据帧格式

<地址 0x01>+<0x21>+<2Bytes: dataSendFlag 数据发送标志>+<2Bytes: freq 发送频率>+<4Bytes: baudRate 波特率>+<CRC16>

2Bytes: dataSendFlag

用位表示是否包含数据项，全部发送则设置为 0x001F。

Bit15-Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Reserved	1: 上传磁力值 0: 不上传磁力值	1: 上传角速度 0: 不上传角速度	1: 上传加速度 0: 不上传加速度	1: 上传欧拉角 0: 不上传欧拉角	1: 上传四元数 0: 不上传四元数

2Bytes: freq (单位 Hz)

上传数据频率，实时生效，手动重启或者断电重启后保持新设置值。

0 代表停止发送。任何不在上表中的值均无法设置。

可以设置的值^{#1} (单位 Hz) 如下：

设置值 (十进制表示)	设置值 (十六进制表示)
-------------	--------------

200 (默认)	C8
100	64
50	32
25	19
20	14
10	0A
5	05
4	04
2	02
1	01
0	00

注 1：较低的上传数据速率并不能明显降低模组的功耗，因为模组内传感器采样与姿态解算速度固定为 200Hz。

4Bytes: BaudRate 波特率^{注2} (单位 bps)

波特率，手动重启或者断电重启后生效。

关于波特率的说明：波特率的最小值为 115200bps。

取值如下 (单位 bps)：

设置值（十进制表示）	设置值（十六进制表示）
115200 (默认)	0x01C200
128000	0x01F400
230400	0x038400
460800	0x070800
921600	0x0E1000
1500000	0x16E360
2000000	0x1E8480
3000000	0x2DC6C0

注 2：若忘记了上次波特率的设置值，可以逐个尝试上个列表中的值，或者通过 4.2 节中 USB 线连接模组，发送恢复出厂设置指令，恢复至 115200bps 波特率。

C/C++结构体参考

```

1.  typedef struct
2.  {
3.      uint8_t addr;
4.      uint8_t code;
5.
6.      /*用位表示是否包含数据项，全部发送则设置为 0x001F*/
7.      union {
8.          struct {
9.              uint16_t quatFlag : 1;    /*Bit:0*/
10.             uint16_t eulerFlag : 1;  /*Bit:1*/
11.             uint16_t accelFlag : 1; /*Bit:2*/
12.             uint16_t gyroFlag : 1;  /*Bit:3*/
13.             uint16_t magFlag : 1;   /*Bit:4*/
14.             uint16_t resvFlag : 11; /*Bit:5~11*/
15.         };
16.         uint16_t dataSendFlag;
17.     };
18.
19.     /*
20.      上传数据频率，实时生效，手动重启或者断电重启后保持新设置值。
21.      可以设置的值（单位 Hz）如下：
22.      -----
23.      DEC|200|100|50|25|20|10|5 |4 |2 |1 |0 |
24.      HEX|C8 |64 |32|19|14|0A|05|04|02|01|00|
25.      -----
26.      0 代表停止发送。任何不在上表中的值均无法设置。
27.      */
28.      uint16_t freq;
29.      /*
30.        波特率，手动重启或者断电重启后生效。
31.        取值如下（单位 bps）：
32.        -----
33.        DEC|115200|128000|230400|460800|921600|1500000|2000000|3000000|
34.        HEX|01C200|01F400|038400|070800|0E1000|16E360 |1E8480 |2DC6C0 |
35.        -----
36.        */
37.      uint32_t baudRate;
38.      uint16_t crc16;
39. }RtTrans_UART_PC2ND;

```

用例

发送/回复指令	说明
发送指令: 01 21 1F 00 64 00 00 10 0E 00 FF FF 回复指令: 01 22 01 F9 60	上传四元数+欧拉角+加速度+角速度+磁力值; 100Hz 上传; 921600bps 波特率;
发送指令: 01 21 1F 00 01 00 00 10 0E 00 FF FF 回复指令: 01 22 01 F9 60	上传四元数+欧拉角+加速度+角速度+磁力值; 1Hz 上传; 921600bps 波特率;
发送指令: 01 21 1F 00 00 00 00 10 0E 00 FF FF 回复指令: 01 22 01 F9 60	上传四元数+欧拉角+加速度+角速度+磁力值; 0Hz 上传 (停止上传); 921600bps 波特率;
发送指令: 01 21 01 00 C8 00 00 10 0E 00 FF FF 回复指令: 01 22 01 F9 60	上传四元数; 200Hz 上传; 921600bps 波特率;
01 21 1F 00 C8 00 00 C2 01 00 FF FF	上传四元数+欧拉角+加速度+角速度+磁力值; 200Hz 上传; 115200bps 波特率;

2.1.2 接收端返回数据帧格式

数据帧格式	
<地址 0x01>+<0x22>+<1Byte: Status>+<CRC16>	
Status	说明
0x01	执行成功

0xFF	执行失败
C/C++结构体参考	
<pre> 1. typedef struct 2. { 3. uint8_t addr; 4. uint8_t code; 5. /*status: 6. 0x01: 设置成功. 7. 0xFF: 设置中有一个或多个参数不正确。. 8. */ 9. uint8_t status; 10. uint16_t crc16; 11. }RtTrans_UART_ND2PC; </pre>	

2.2 开始实时传输

节点收到 2.1 节 RTTRANS 指令后，按照要求的频率上传姿态及其他数据，其中波特率的设置需要重启生效，其它的设置为实时生效。

命令字	指令	描述
0xXX (No Write)		
0x24 (Read)	UDIMU	上传 IMU 数据

2.2.1 发送端数据帧格式

无需向模组发送指令，模组根据 2.1 节的设置自动上传数据帧。

2.2.2 接收端返回数据帧格式

数据帧格式
<地址 0x01>+<0x22>+<2Bytes: dataSendFlag 数据发送标志>+[8Bytes: 四元数]+[6Bytes: 欧拉角]+[6Bytes: 加速度]+ [6Bytes: 角速度]+ [6Bytes: 磁力值]+<CRC16>
2Bytes: dataSendFlag
根据 dataSendFlag 的不同可至多发送 16 种不同数据。

Bit15-Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Reserved	1: 包含磁	1: 包含角	1: 包含加	1: 包含欧	1: 包含四

	力值 0: 不包含 磁力值	速度 0: 包含传 角速度	速度 0: 不包含 加速度	拉角 0: 不包含 欧拉角	元数 0: 不包含 四元数
8Bytes: 四元数					
Byte7	Byte6	Byte5	Byte4	Byte3	Byte2
z 高字节	z 低字节	y 高字节	y 低字节	x 高字节	x 低字节
两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据
将上述所得数据 除以 (1<<14) 获 得浮点数值表示 的 z	将上述所得数据 除以 (1<<14) 获 得浮点数值表示 的 y	将上述所得数据 除以 (1<<14) 获 得浮点数值表示 的 x	将上述所得数据 除以 (1<<14) 获 得浮点数值表示 的 w		
取 值 范 围 为 [-1.0,+1.0]	取 值 范 围 为 [-1.0,+1.0]	取 值 范 围 为 [-1.0,+1.0]			
6Bytes: 欧拉角					
Byte5	Byte4	Byte3	Byte2	Byte1	Byte0
z 高字节	z 低字节	y 高字节	y 低字节	x 高字节	x 低字节
两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据	两字节合为一个 int16_t 类型数据
将上述所得数据除以 (1<<7) 获得浮点数值 表示的 z	将上述所得数据除以 (1<<7) 获得浮点数值表 示的 y	将上述所得数据除以 (1<<7) 获得浮点数值表 示的 x			
取值范围为[0,360.0]	取 值 范 围 为 [-180.0,+180.0]	取 值 范 围 为 [-180.0,+180.0]	取 值 范 围 为 [-180.0,+180.0]	取 值 范 围 为 [-180.0,+180.0]	取 值 范 围 为 [-180.0,+180.0]

6Bytes: 加速度					
Byte5	Byte4	Byte3	Byte2	Byte1	Byte0
z 高字节	z 低字节	y 高字节	y 低字节	x 高字节	x 低字节
两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据	
将上述所得数据除以 2048.0 获得浮点数值表 示的 z		将上述所得数据除以 2048.0 获得浮点数值表 示的 y		将上述所得数据除以 2048.0 获得浮点数值表 示的 x	
取值范围为[-16.0,+16.0]		取值范围为[-16.0,+16.0]		取值范围为[-16.0,+16.0]	

6Bytes: 角速度					
Byte5	Byte4	Byte3	Byte2	Byte1	Byte0
z 高字节	z 低字节	y 高字节	y 低字节	x 高字节	x 低字节
两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据	
将上述所得数据除以 16.4 获得浮点数值表 示的 z		将上述所得数据除以 16.4 获得浮点数值表 示的 y		将上述所得数据除以 16.4 获得浮点数值表 示的 x	
取 值 范 围 为 [-2000.0,+2000.0]		取 值 范 围 为 [-2000.0,+2000.0]		取 值 范 围 为 [-2000.0,+2000.0]	

6Bytes: 磁力值					
Byte5	Byte4	Byte3	Byte2	Byte1	Byte0
z 高字节	z 低字节	y 高字节	y 低字节	x 高字节	x 低字节
两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据		两字节合为一个 <code>int16_t</code> 类型数据	
将上述所得数据除以 1.0 获得浮点数值表示的 z		将上述所得数据除以 1.0 获得浮点数值表示的 y		将上述所得数据除以 1.0 获得浮点数值表示的 x	

C/C++结构体参考

```
1.  typedef struct
2.  {
3.      uint8_t addr;
4.      uint8_t code;
5.      /*[0, 4294967295]之间增加, 达到最大值后从0开始重新计数,
6.       注意若以200Hz上传数据, 此值约248天溢出一次*/
7.      uint32_t frameNum;
8.
9.      /*用位表示是否包含数据项*/
10.     union {
11.         struct {
12.             uint16_t quatFlag : 1; /*Bit:0*/
13.             uint16_t accelFlag : 1; /*Bit:1*/
14.             uint16_t gyroFlag : 1; /*Bit:2*/
15.             uint16_t magFlag : 1; /*Bit:3*/
16.             uint16_t eulerFlag : 1; /*Bit:4*/
17.             uint16_t resvFlag : 11; /*Bit:5~11*/
18.         };
19.         uint8_t dataSendFlag;
20.     };
21.
22.     int16_t quat[4];
23.     int16_t euler[3];
24.     int16_t accel[3];
25.     int16_t gyro[3];
26.     int16_t mag[3];
27.     uint16_t crc16;
28. }UDIMU_UART_ND2PC;
```

三、硬件特性

3.1 陀螺仪参数

Full-Scale Range	$\pm 2000^\circ/\text{s}$
Cross-Axis Sensitivity	$\pm 2\%$
Nonlinearity	$\pm 0.1\%$
Noise Power Spectral Density	$0.01^\circ/\text{s}/\sqrt{\text{Hz}}$
Low pass filter	250Hz

3.2 加速度计参数

Full-Scale Range	$\pm 16g$
Cross-Axis Sensitivity	$\pm 2\%$
Nonlinearity	$\pm 0.5\%$
Noise Power Spectral Density	$300\mu\text{g}/\sqrt{\text{Hz}}$
Low pass filter	460Hz

3.3 磁力计参数

Full-Scale Range	$\pm 2\text{Gauss}$
Cross-Axis Sensitivity	$0.1\%/\text{G}(\text{Cross field} = 1 \text{ Gauss, H applied} = \pm 2 \text{ Gauss})$
Linearity (Best fit linear curve)	$0.1\%\text{FS}(\text{Field Range} = \pm 2\text{G})$
X-Y-Z Orthogonality	$90 \pm 1 \text{ degree}$

3.4 姿态解算

解算精度	
俯仰 Pitch/翻滚 Roll	0.5° RMS
航向 Yaw	1° RMS

运行时间	
解算时间（数据发出时刻-数据采集时刻 ^{注1)}	3.94ms
解算频率	200Hz
解算周期	5ms

注 1：解算时间定义的是从“开始数据采集时刻”到“开始发送数据帧”的时刻，由于使用的串口波特率不同，所以接收端收到一帧完整数据包的时间也不同，建议若硬件条件允许，应使用尽量高的通信波特率：（1）在 3Mbps 波特率下，接收一帧完整数据包需 139uS。而（2）在 115200bps 波特率下，接收一帧完整数据包需要 3.59ms。

3.5 电气/接口参数

电气参数	
电流	25mA@5V
功耗	125mW
供电电源	5V/1A DC
接口参数	
TTL 串口	Baudrate(Default):115200bps(Up to 3Mbps) Data Bits:8 Stop Bits:1 Parity:None Flow Control:None
USB 接口	USB 2.0 Full-Speed

四、硬件接口

4.1 硬件接口图

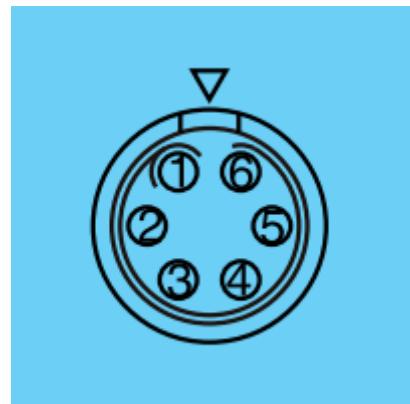


图 4.1 数据接口 Front View/前视图

表 4.1 引脚功能

Pin Number	Function
1	USB DP
2	5V DC
3	GND
4	TTL TX
5	TTL RX
6	USB DM

4.2 USB 连接方式

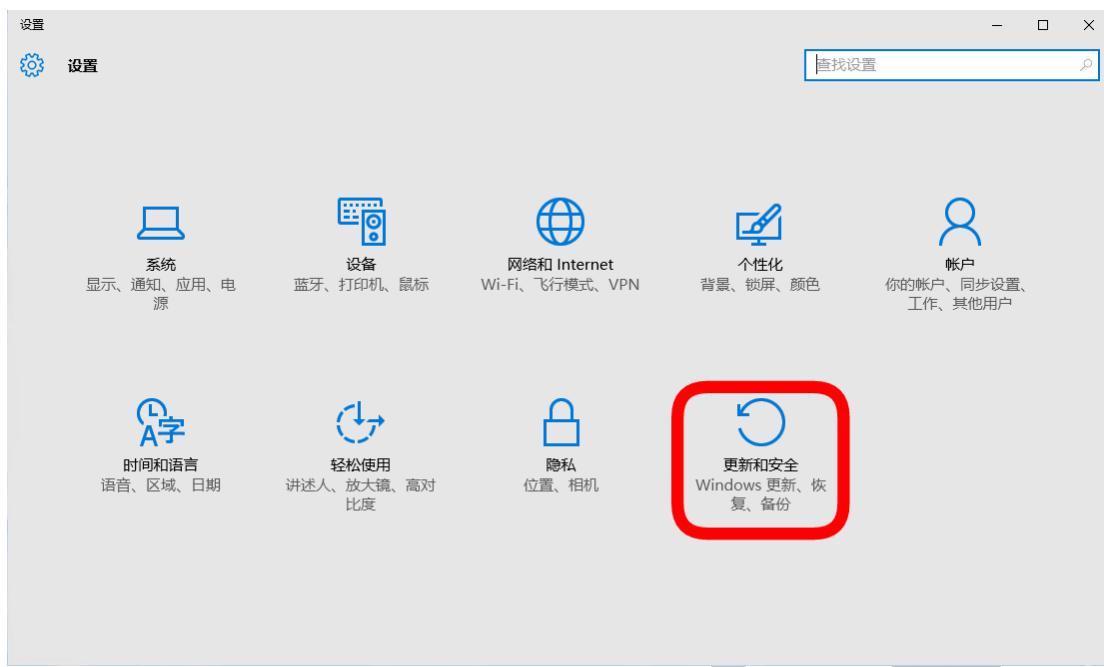
使用 USB 连接方式评估模组的性能。

使用 USB 航空插头线，连接模组与 PC 机，并按照下面教程或视频教程安装所需硬件驱动，下载安装 **MotionMars™ 2** 软件客户端评估模组性能。

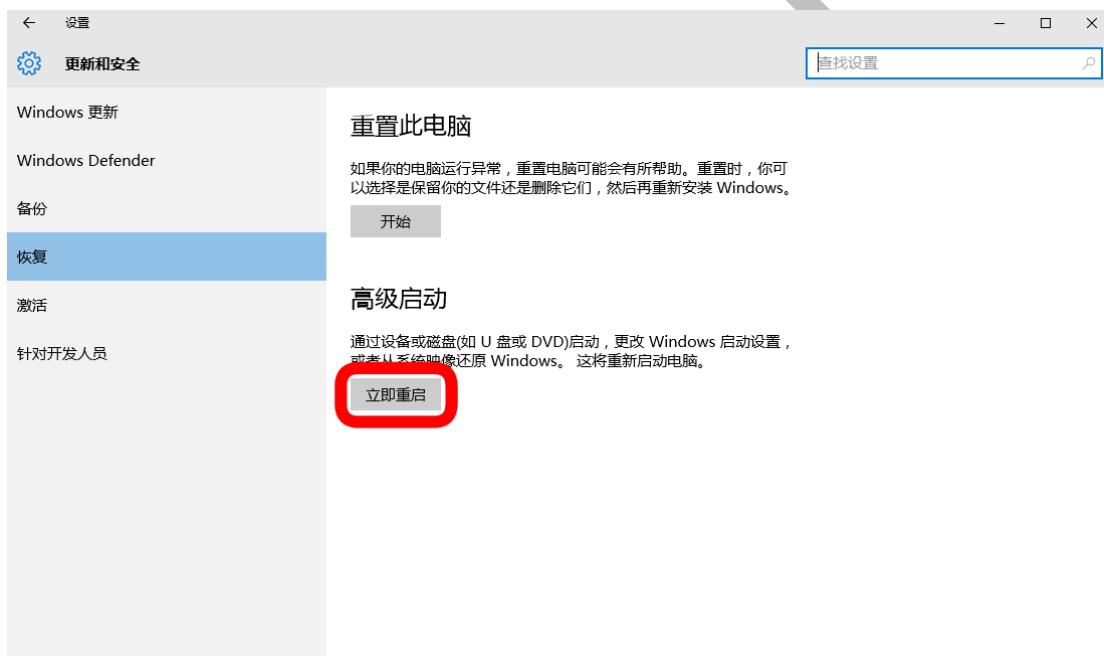
4.2.1 Win10 系统驱动安装

(1) 打开所有设置

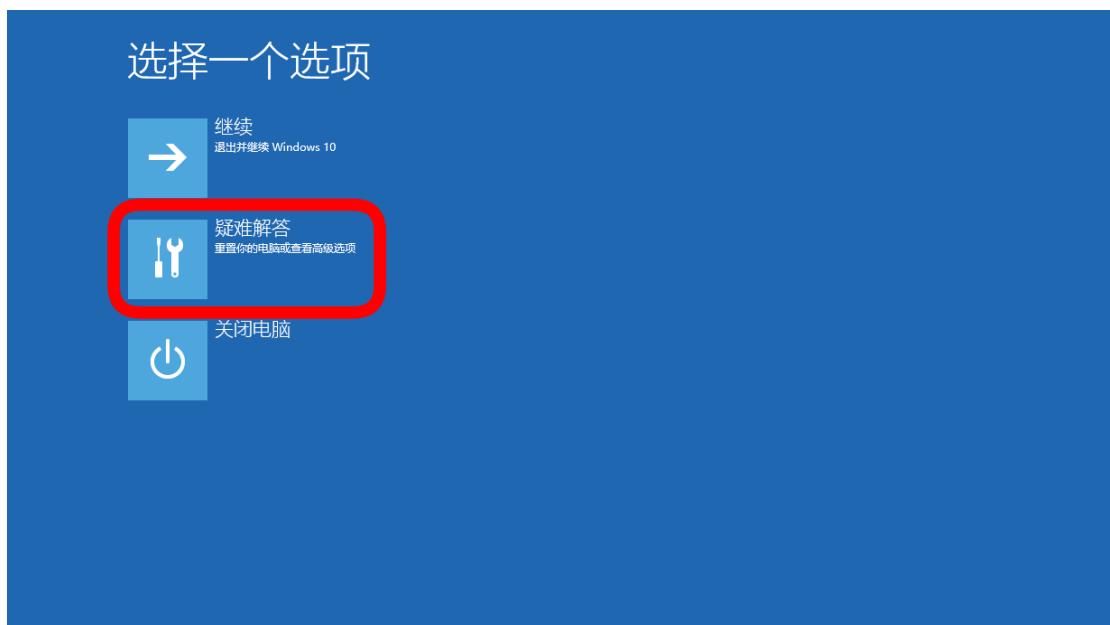
选择更新和安全。



(2) 选择高级启动



(3) 选择疑难解答

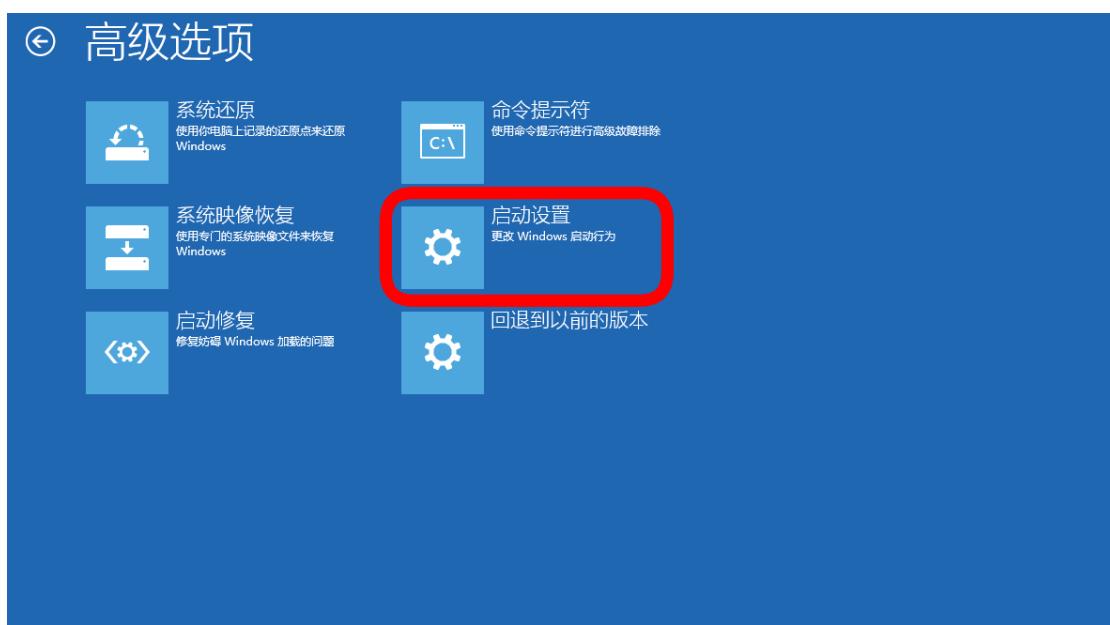


(4) 选择高级选项



(5) 选择启动设置

④ 高级选项



(6) 选择重启

⑤ 启动设置

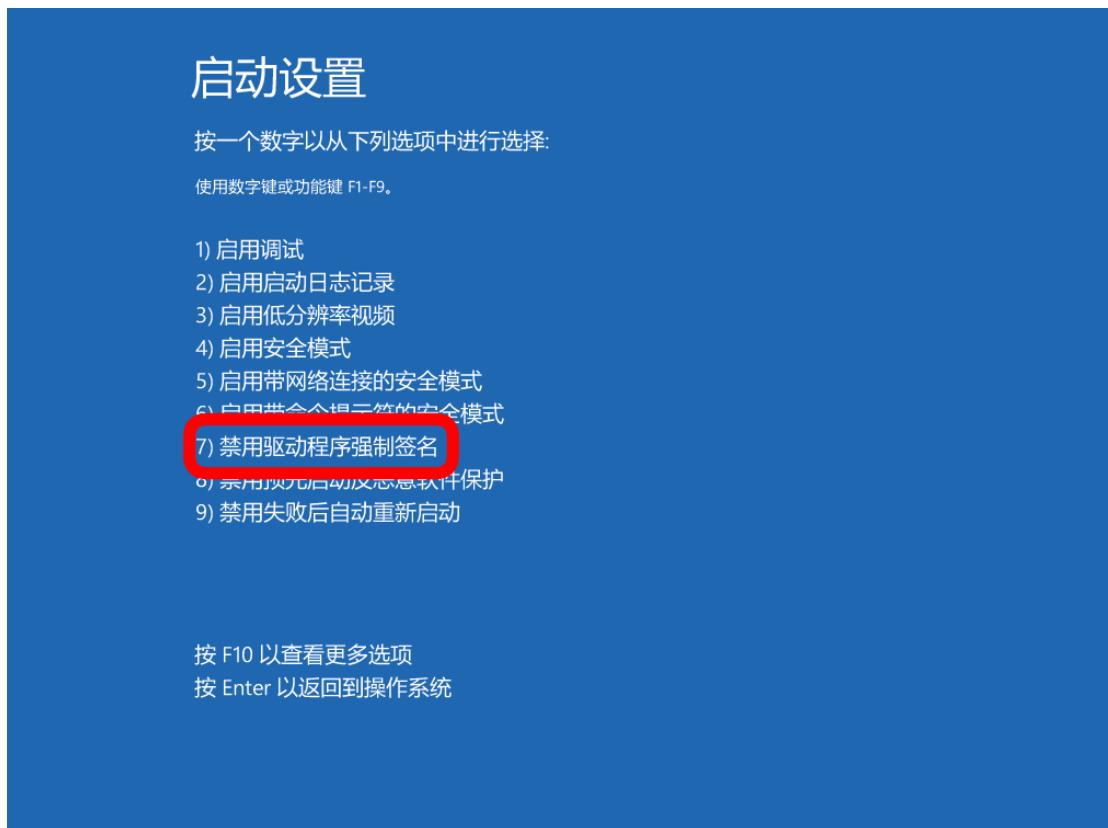
重启以更改 Windows 选项，例如：

- 启用低分辨率视频模式
- 启用调试模式
- 启用启动日志
- 启用安全模式
- 禁用驱动程序强制签名
- 禁用预先启动反恶意软件保护
- 禁用系统失败时自动重启

重启

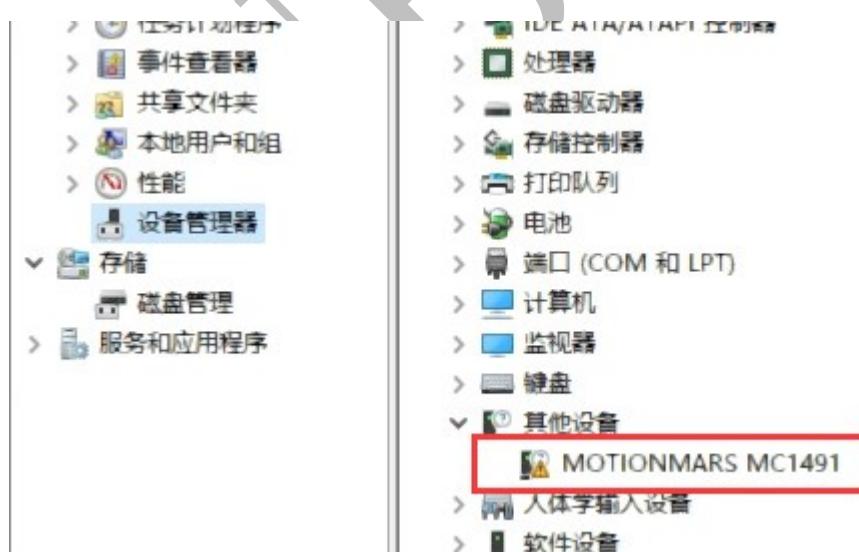
重启之后，进入启动设置界面。

(7) 按键盘 7 选择禁用驱动程序强制签名。



(8) 连接模组

连接 MOTIONMARS Q1 设备到电脑，打开计算机管理，出现 MOTIONMARS MC1491 设备。



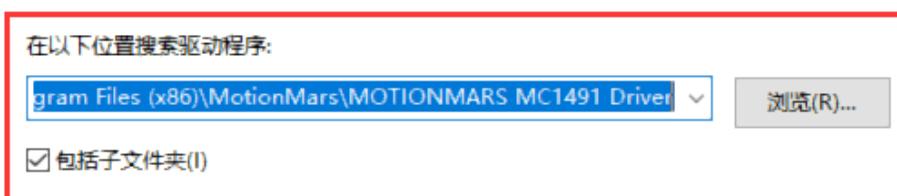
右键更新驱动程序软件，选择浏览计算机以查找驱动程序软件。

选择 MotionMars™ 2 软件安装目录中的《MOTIONMARS MC1491 Driver》文件夹。

X

← ■ 更新驱动程序 - MOTIONMARS MC1491

浏览计算机上的驱动程序



→ 让我从计算机上的可用驱动程序列表中选取(L)

此列表将显示与该设备兼容的可用驱动程序，以及与该设备属于同一类别的所有驱动程序。

单击下一步，选择始终安装驱动



正在安装驱动程序软件...



(9) 断电重启模组，驱动安装完成

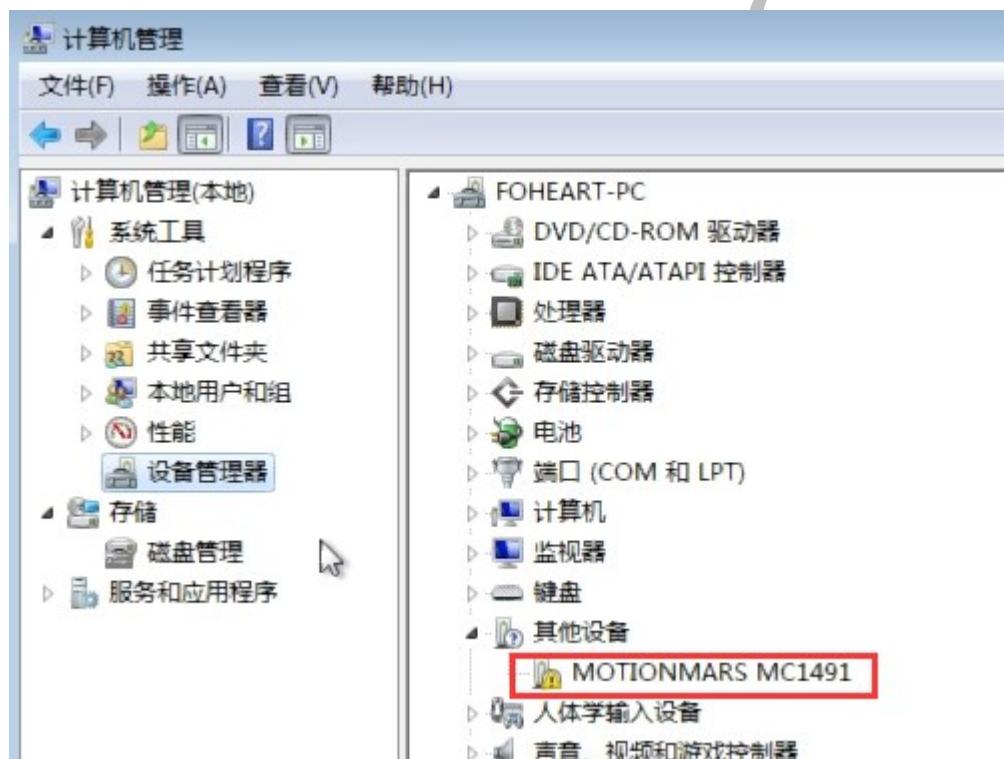
在设备管理器中出现 MOTIONMARS MC1491 设备，驱动成功安装。



4.2.2 Win7 系统驱动安装

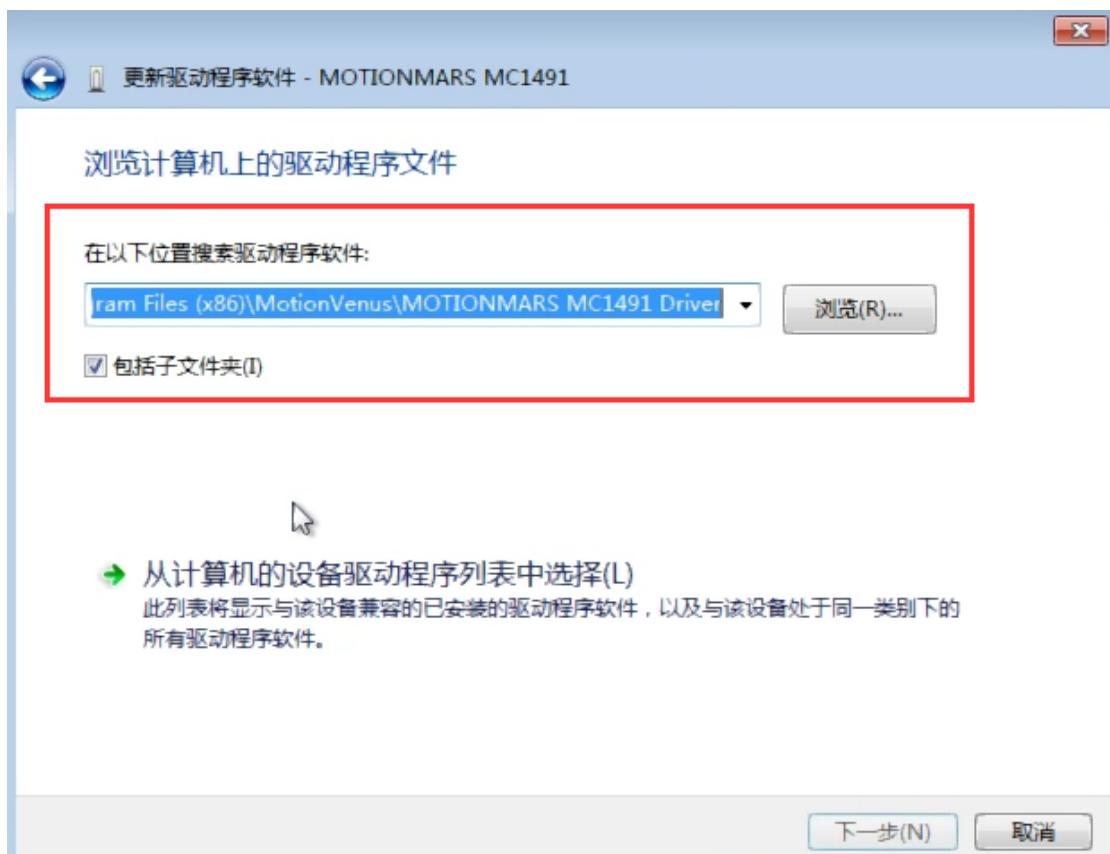
(1) 连接模组

连接 MOTIONMARS Q1 设备到电脑，打开计算机管理，出现 MOTIONMARS MC1491 设备。

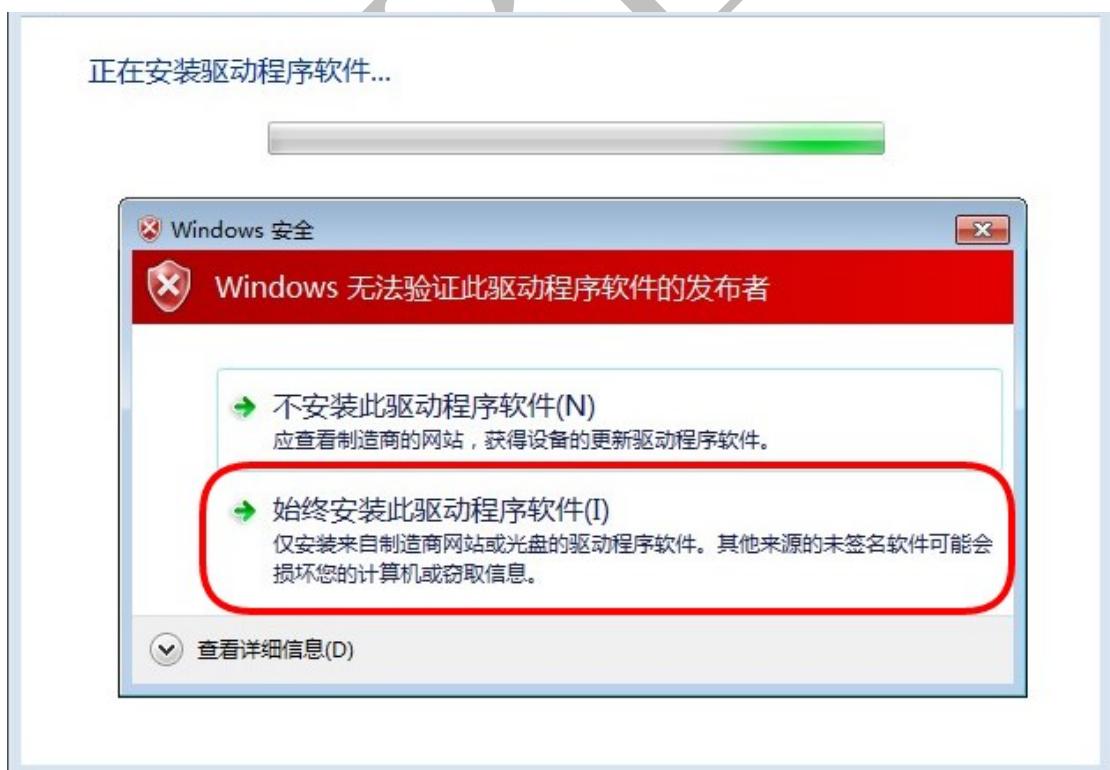


右键更新驱动程序软件，选择浏览计算机以查找驱动程序软件。

选择 MotionMars™ 2 软件安装目录中的《MOTIONMARS MC1491 Driver》文件夹。

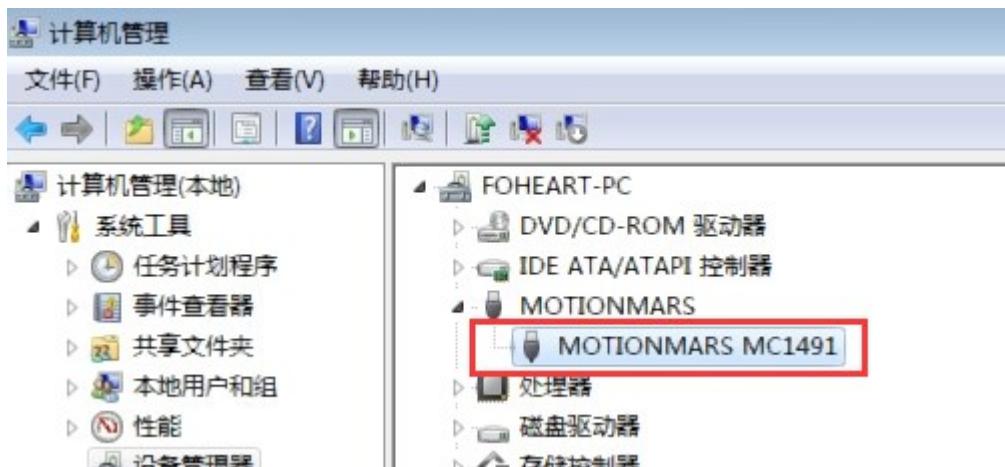


单击下一步，选择始终安装驱动。

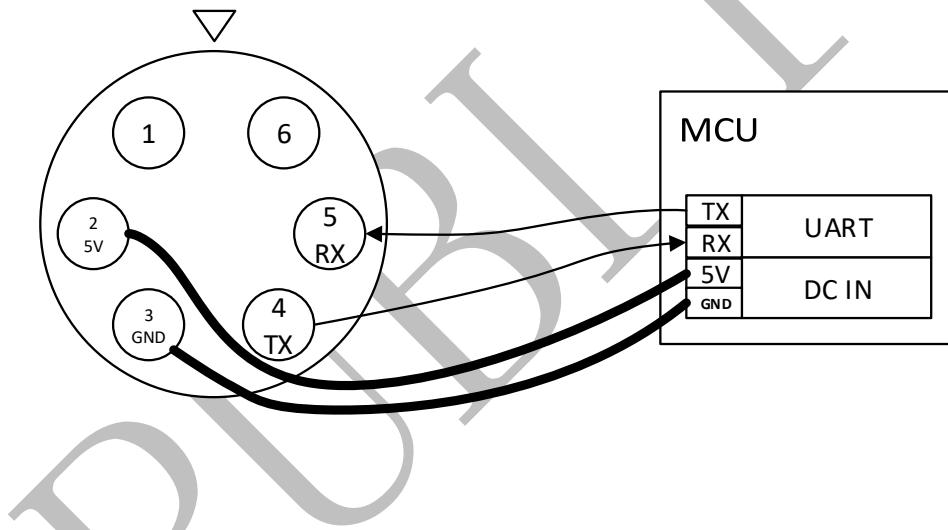


(9) 断电重启模组，驱动安装完成

在设备管理器中出现 MOTIONMARS MC1491 设备，驱动成功安装。



4.3 TTL 连接方式



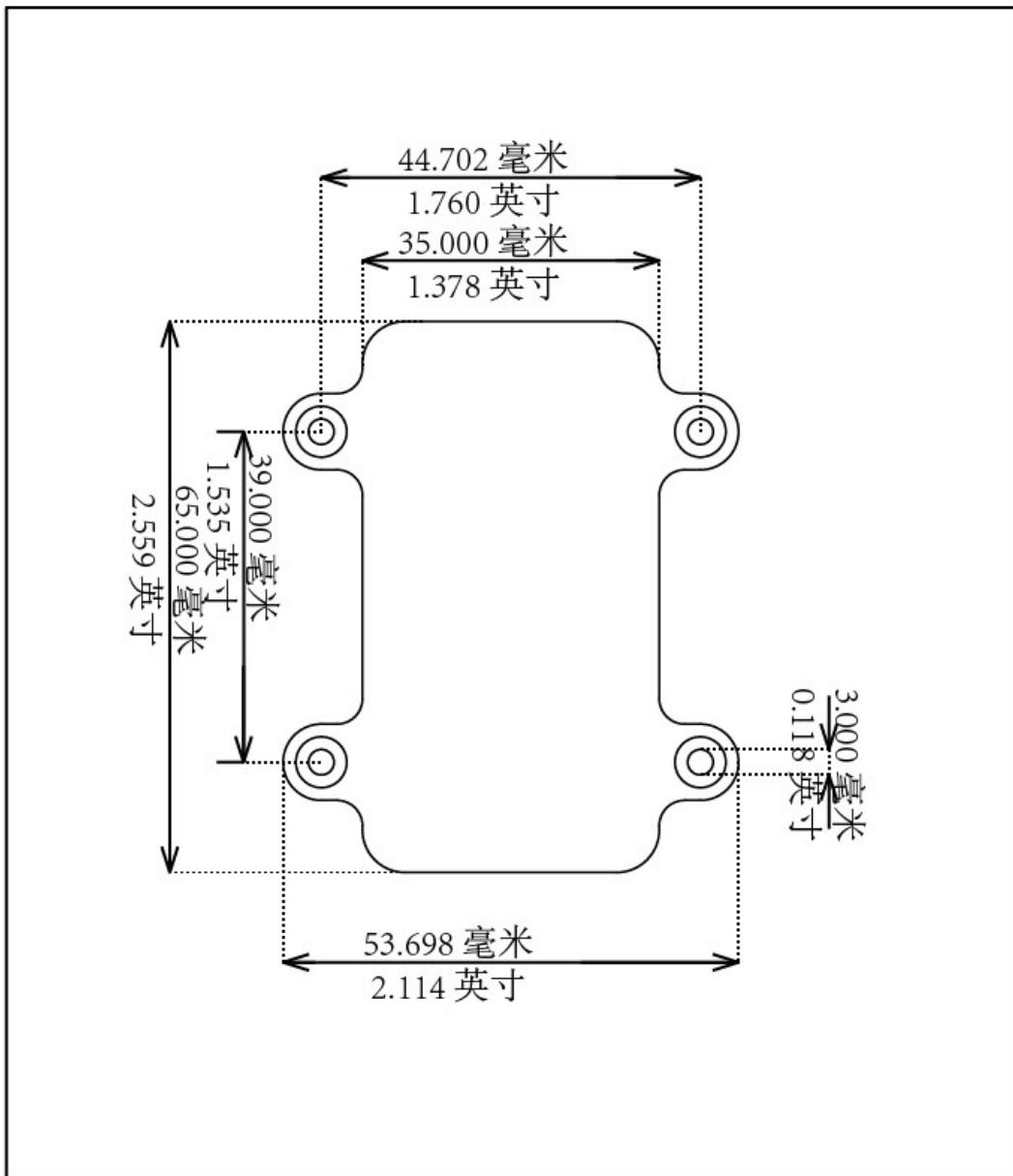
TTL 方式通过交叉线与单片机端口连接，通过单片机发送指令与模组进行交互通信，适用于最终产品。

使用 TTL 线无法连接客户端 MotionMars™ 2 进行评估。MotionMars™ 2 使用 USB 线连接模组。

TTL 与 USB 在模组上为并行运行模式，即可在 USB 连接电脑端的情况下同时 TTL 连接单片机端口进行指令通信，在任意一端进行的配置，会同步到另一端口。

五、安装尺寸图

5.1 外观尺寸



△ 更新信息: 1	(1) 壳体高度18毫米; (2) 安装螺丝Φ3x0.5mm螺距。	标题: Q1外观尺寸图
△ 更新信息: 2		日期: Jan 5, 2019
△ 更新信息: 3		绘制者: HMX
△ 更新信息: 4		页数: 1
		比例: 1

图 5.1 安装尺寸图

六、订货信息

全系列均包含 IMU/VRU 的所有功能，输出加速度、角速度、磁力计原始数据/校准后的数据(通过串口指令切换)，以及经过内置姿态融合算法计算的俯仰、翻滚、横滚角等数据。

型号	功能	封装	品牌
MC1491-C	IMU/VRU	6061 铝合金（外壳）	MOTIONMARS™
MC1491-D	IMU/VRU	PLCC44	MOTIONMARS™

6.1 评估套件

MC1491-C 型号模组 x1

防水外包装盒 x1

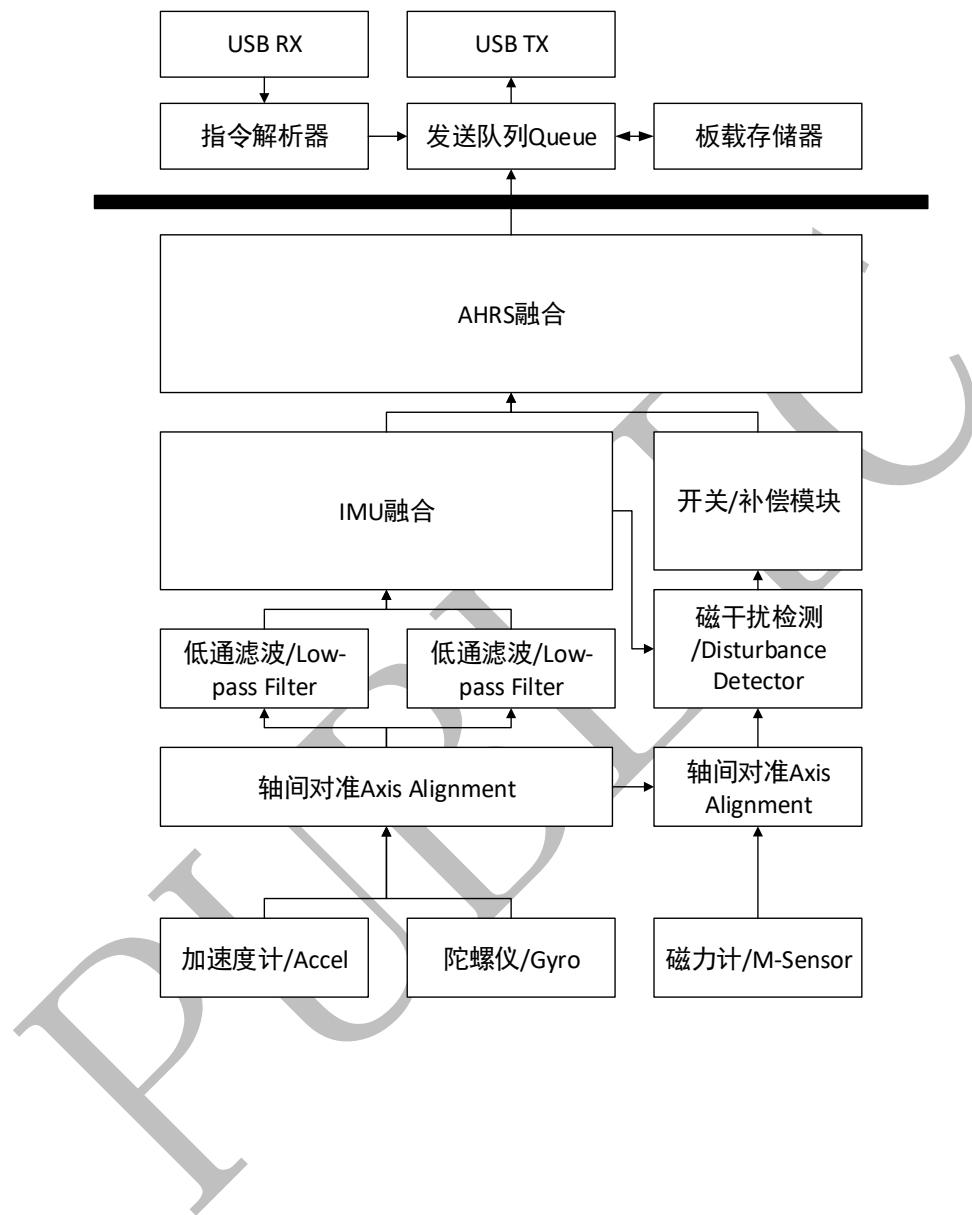
2 米 USB 转航空插头线 x1

1 米 TTL 转航空插头线 x1

说明书 x1

七、工作流程图

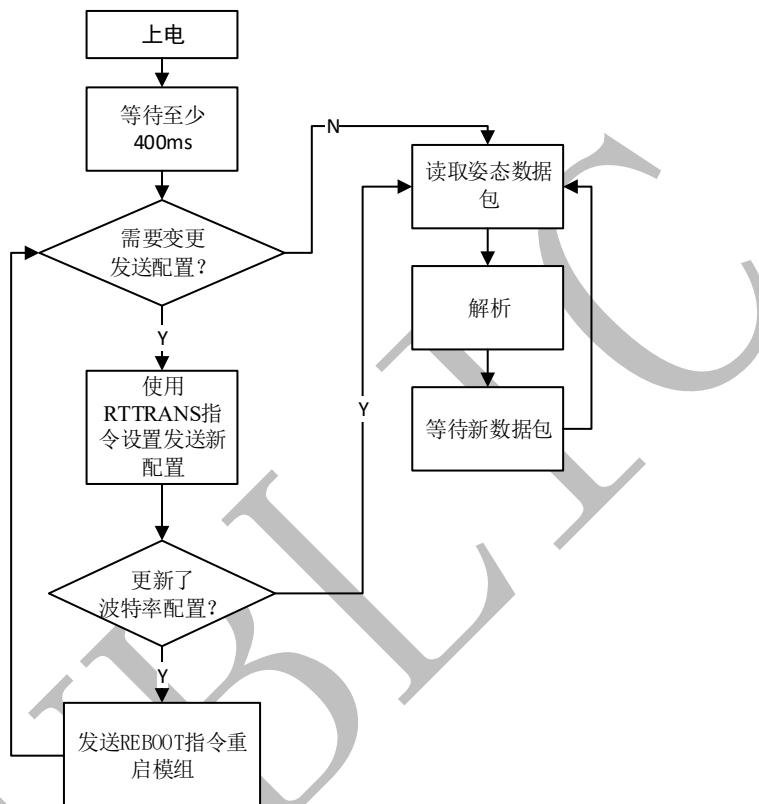
本产品可实现最大 200Hz 的 6-DoF 及 9-DoF 追踪，其工作流程如下：



八、使用流程

8.1 配置流程

以 TTL 方式连接单片机，在单片机端按照以下流程进行模块操作：



8.2 姿态数据包解析流程

根据 RTTRANS 指令设置的不同，模组上传到数据包内容也不相同，即模组的数据包长度可变。

收到一帧完整数据后，首先需计算 CRC16 是否正确（源码见附录 2），再截取数据包头部信息，根据头部信息顺序向下截取四元数、欧拉角、加速度、角速度、磁力值等信息。

头部信息的结构体如下：

```

1. #pragma pack(1)
2. typedef struct {
3.     uint8_t addr;
4.     uint8_t code;
  
```

```

5.
6.
7.     /*[0, 4294967295]之间增加, 达到最大值后从 0 开始重新计数,
8.         注意若以 200Hz 上传数据, 此值约 248 天溢出一次*/
9.     uint32_t frameNum;
10.
11.    /*用位表示是否包含数据项*/
12.    union {
13.        struct {
14.            uint16_t quatFlag : 1; /*Bit:0*/
15.            uint16_t eulerFlag : 1; /*Bit:1*/
16.            uint16_t accelFlag : 1; /*Bit:2*/
17.            uint16_t gyroFlag : 1; /*Bit:3*/
18.            uint16_t magFlag : 1; /*Bit:4*/
19.            uint16_t resvFlag : 11; /*Bit:5~11*/
20.        };
21.        uint16_t dataSendFlag;
22.    };
23. }UDIMU_UART_HDR_ND2PC;
24. #pragma pack()

```

数据解析参考代码如下:

```

1. ****
2. * Function: ParsePckage
3. * Description: 姿态数据包解析参考函数
4. * Input: buff:一帧完整数据包的首字节地址 size:一帧完整数据包长度
5. * Output:NULL
6. * Return:
7. * Others: (1) 如果 crc 校验失败会中途退出 (2) 需要引入<stdint.h>头文件
8. ****
9. void ParsePckage( uint8_t* buff, uint16_t size )
10. {
11.     uint16_t      crc16          = buff[size - 2] | ( buff[size - 1] << 8 );
12.     uint16_t      crc16Expect   = CRC16( buff, size - 2 );
13.
14.     if( crc16 != crc16Expect )
15.     {
16.         /*CRC16 校验错误, 不执行后续代码*/
17.         return;
18.     }
19.

```

```
20.     int                     frontptr = 0;
21.     UDIMU_UART_HDR_ND2PC     hdr;
22.     for( int i = 0; i < sizeof( hdr ); i++ )
23.     {
24.         *( (uint8_t * )( &hdr ) + i ) = buff[ i ];
25.     }
26.     /*打印帧序号，如不需要可去掉*/
27.     printf( "[%d]Frame Num:%d\r\n", HAL_GetTick( ), hdr.frameNum );
28.     frontptr += sizeof( hdr );
29.
30.     /*! 最终数据！
31.      定义浮点数数据，由整型格式转换出浮点数表示的数据*/
32.     float    quat_f[4] = { 0.0f, 0.0f, 0.0f, 0.0f } ;/*w,x,y,z*/
33.     float    euler_f[3] = { 0.0f, 0.0f, 0.0f } ;
34.     float    accel_f[3] = { 0.0f, 0.0f, 0.0f } ;
35.     float    gyro_f[3] = { 0.0f, 0.0f, 0.0f } ;
36.     float    mag_f[3] = { 0.0f, 0.0f, 0.0f } ;
37.
38.     if( hdr.quatFlag )
39.     {
40.         int16_t quat[4] = { 0, 0, 0, 0 };
41.         memcpy( quat, &buff[frontptr], sizeof( quat ) );
42.         frontptr += sizeof( quat );
43.         quat_f[0] = (float)quat[0] / ( 1 << 14 );
44.         quat_f[1] = (float)quat[1] / ( 1 << 14 );
45.         quat_f[2] = (float)quat[2] / ( 1 << 14 );
46.         quat_f[3] = (float)quat[3] / ( 1 << 14 );
47.         /*打印四元数，如不需要可去掉*/
48.         printf( "Quat:%f %f %f %f\r\n",
49.                 quat_f[0],
50.                 quat_f[1],
51.                 quat_f[2],
52.                 quat_f[3]
53.             );
54.     }
55.
56.     if( hdr.eulerFlag )
57.     {
58.         int16_t euler[3] = { 0, 0, 0 };
59.         memcpy( euler, &buff[frontptr], sizeof( euler ) );
60.         frontptr += sizeof( euler );
61.         euler_f[0] = (float)euler[0] / ( 1 << 7 );
```

```
62.         euler_f[1] = (float)euler[1] / ( 1 << 7 );
63.         euler_f[2] = (float)euler[2] / ( 1 << 7 );
64.         /*打印欧拉角, 如不需要可去掉*/
65.         printf( "Euler:%f %f %f\r\n",
66.                 euler_f[0],
67.                 euler_f[1],
68.                 euler_f[2]
69.             );
70.     }
71.
72.     if( hdr.accelFlag )
73.     {
74.         int16_t accel[3] = { 0, 0, 0 };
75.         memcpy( accel, &buff[frontptr], sizeof( accel ) );
76.         frontptr += sizeof( accel );
77.         accel_f[0] = (float)accel[0] / 2048.0f;
78.         accel_f[1] = (float)accel[1] / 2048.0f;
79.         accel_f[2] = (float)accel[2] / 2048.0f;
80.         /*打印加速度, 如不需要可去掉*/
81.         printf( "Accel:%f %f %f\r\n",
82.                 accel_f[0],
83.                 accel_f[1],
84.                 accel_f[2]
85.             );
86.     }
87.
88.     if( hdr.gyroFlag )
89.     {
90.         int16_t gyro[3] = { 0, 0, 0 };
91.         memcpy( gyro, &buff[frontptr], sizeof( gyro ) );
92.         frontptr += sizeof( gyro );
93.         gyro_f[0] = (float)gyro[0] / 16.4f;
94.         gyro_f[1] = (float)gyro[1] / 16.4f;
95.         gyro_f[2] = (float)gyro[2] / 16.4f;
96.         /*打印角速度, 如不需要可去掉*/
97.         printf( "Gyro:%f %f %f\r\n",
98.                 gyro_f[0],
99.                 gyro_f[1],
100.                gyro_f[2]
101.            );
102.    }
103.
```

```

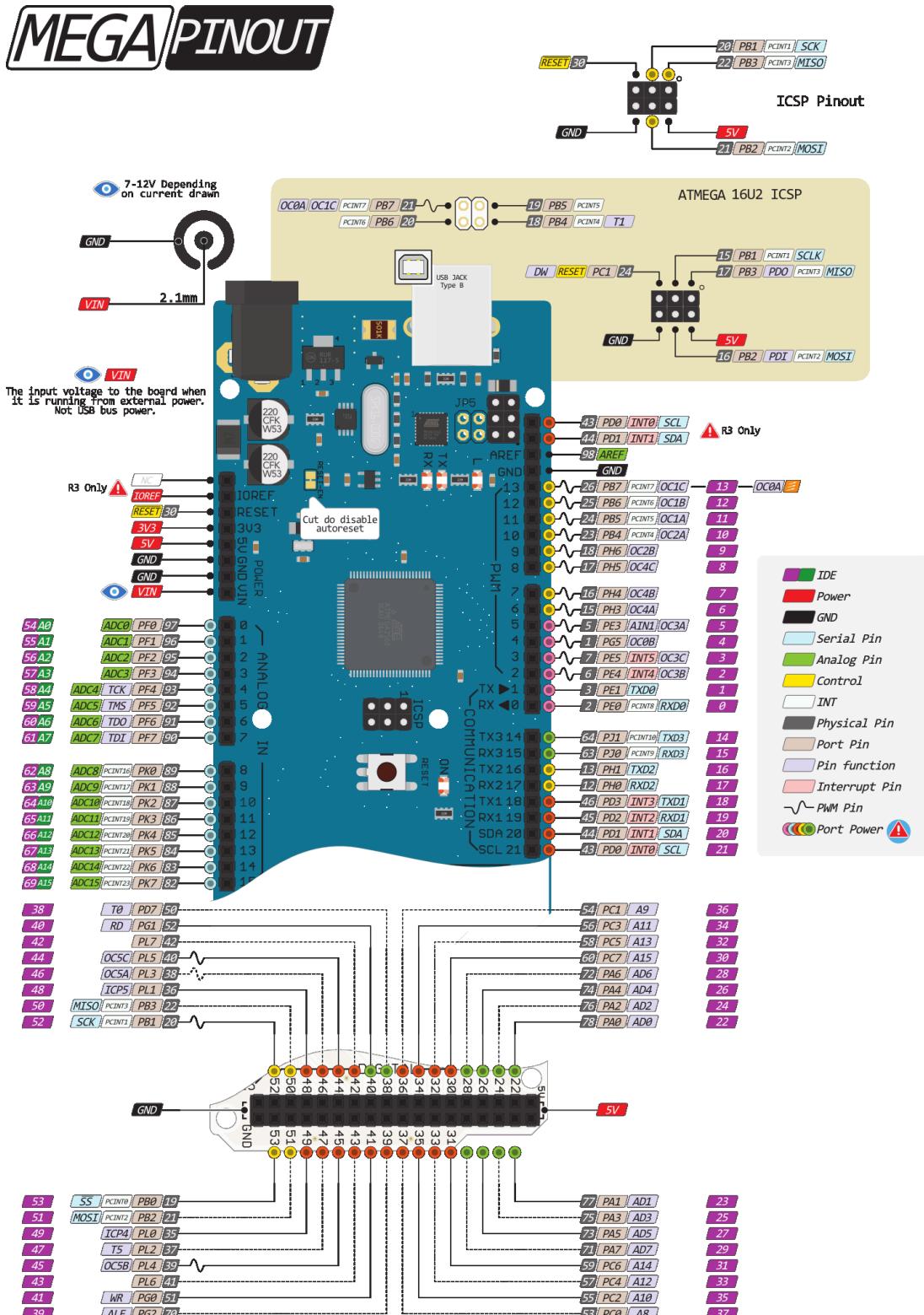
104.     if( hdr.magFlag )
105.     {
106.         int16_t mag[3] = { 0, 0, 0 };
107.         memcpy( mag, &buff[frontptr], sizeof( mag ) );
108.         frontptr += sizeof( mag );
109.         mag_f[0] = (float)mag[0] / 120.0f;
110.         mag_f[1] = (float)mag[1] / 120.0f;
111.         mag_f[2] = (float)mag[2] / 120.0f;
112.         /*打印磁力值, 如不需要可去掉*/
113.         printf( "Mag:%f %f %f\r\n",
114.                 mag_f[0],
115.                 mag_f[1],
116.                 mag_f[2]
117.             );
118.     }
119.     /*在此可插入回调函数, 将解析完成的单项或多项数据分发出去。*/
120. }
```

样例数据	
接收数据（16 进制，低字节在前）	解析后的浮点数
01 24 02 00 00 00 1F 00 FE 3F 53 00 56 00 8C 00 4B 00 4C 00 7E 00 15 00 0B 00 80 07 FD FF 02 00 00 00 A3 FB 9F 15 E5 E8 0A 0B	[1396]Frame Num:2 Quat:0.999878 0.005066 0.005249 0.008545 Euler:0.585938 0.593750 0.984375 Accel:0.010254 0.005371 0.937500 Gyro:-0.182927 0.121951 0.000000 Mag:-9.308333 46.125000 -49.291668
01 24 03 00 00 00 1F 00 FF 3F 4E 00 05 00 8D 00 46 00 04 00 7E 00 14 00 1B 00 91 07 00 00 FB FF 00 00 BC FB A4 15 FE E8 A0 7E	[2396]Frame Num:3 Quat:0.999939 0.004761 0.000305 0.008606 Euler:0.546875 0.031250 0.984375 Accel:0.009766 0.013184 0.945801 Gyro:0.000000 -0.304878 0.000000 Mag:-9.100000 46.166668 -49.083332
01 24 04 00 00 00 1F 00 FF 3F 4E 00 ED	[3396]Frame Num:4

FF 8C 00 45 00 EE FF 7D 00 0C 00 18 00 8C 07 FE FF 03 00 FF FF 87 FB D3 15 E0 E8 89 61	Quat:0.999939 0.004761 -0.001160 0.008545 Euler:0.539063 -0.140625 0.976563 Accel:0.005859 0.011719 0.943359 Gyro:-0.121951 0.182927 -0.060976 Mag:-9.541667 46.558334 -49.333332
01 24 05 00 00 00 1F 00 FF 3F 4D 00 E1 FF 8D 00 44 00 E4 FF 7E 00 15 00 0F 00 85 07 FD FF 01 00 01 00 9B FB BF 15 EA E8 61 B7	[4396]Frame Num:5 Quat:0.999939 0.004700 -0.001892 0.008606 Euler:0.531250 -0.218750 0.984375 Accel:0.010254 0.007324 0.939941 Gyro:-0.182927 0.060976 0.060976 Mag:-9.375000 46.391666 -49.250000
01 24 06 00 00 00 1F 00 FF 3F 4D 00 DA FF 8D 00 44 00 DE FF 7E 00 0A 00 15 00 89 07 FD FF 00 00 00 00 B7 FB BF 15 A7 E8 BA C5	[5396]Frame Num:6 Quat:0.999939 0.004700 -0.002319 0.008606 Euler:0.531250 -0.265625 0.984375 Accel:0.004883 0.010254 0.941895 Gyro:-0.182927 0.000000 0.000000 Mag:-9.141666 46.391666 -49.808334

九、使用 Arduino 评估模组

使用 Arduino MEGA 2560 接收并解析来自 MotionMars Q1 的姿态数据包。



Q1 使用出厂默认配置：

波特率： 115200bps

发送数据频率: 200Hz

发送数据类型: 四元数+欧拉角+标定后的加速度值+标定后的角速度值+标定后的磁力计值。

硬件连接方式:

- (1) 连接 Q1 的 TTL TX 线到 MEGA2560 的 RX1 19 脚。
- (2) 连接 Q1 的 TTL RX 线到 MEGA2560 的 TX1 18 脚。
- (3) 连接 Q1 的 5V 电源与 GND 到 MEGA2560 的 5V 电源与 GND。
- (4) 打开 Arduino IDE 的工具->串口监视器查看解析完成的数据包。

参考源码:

```

1.  ****
2.  * Copyright (C), 2015-2019, BEIJING FOHEART Tech. Co., Ltd.
3.  * FileName:MotionMars_Q1_Arduino_MEGA2560.ino
4.  * Author:MaoxiaoHu
5.  * Date:01/09/2019
6.  * Description:MotionMars Q1
7.  Q1 使用出厂默认配置:
8.  波特率: 115200bps
9.  发送数据频率: 200Hz
10. 发送数据类型: 四元数+欧拉角+标定后的加速度值+标定后的角速度值+标定后的磁力计值。
11.
12. 使用 Arduino MEGA 2560 接收并解析来自 MotionMars Q1 的姿态数据包。
13. (1) 连接 Q1 的 TTL TX 线到 MEGA2560 的 RX1 19 脚。
14. (2) 连接 Q1 的 TTL RX 线到 MEGA2560 的 TX1 18 脚。
15. (3) 连接 Q1 的 5V 电源与 GND 到 MEGA2560 的 5V 电源与 GND。
16. (4) 打开 Arduino IDE 的工具->串口监视器查看解析完成的数据包。
17. * Version:
18. * Function List:
19. *   1. -----
20. * History:
21. *   <author>  <time>  <version>  <desc>
22. *   MaoxiaoHu 19/01/10    1.0    build this moudle
23. ****
24. #define CIRCLEBUFF_LEN ( 42 )
25.
```

```
26. #pragma pack(1)
27. typedef struct {
28.     uint8_t addr;
29.     uint8_t code;
30.
31.     /*[0, 4294967295]之间增加, 达到最大值后从 0 开始重新计数,
32.      注意若以 200Hz 上传数据, 此值约 248 天溢出一次*/
33.     uint32_t frameNum;
34.
35.     /*用位表示是否包含数据项*/
36.     union {
37.         struct {
38.             uint16_t quatFlag : 1; /*Bit:0*/
39.             uint16_t eulerFlag : 1; /*Bit:1*/
40.             uint16_t accelFlag : 1; /*Bit:2*/
41.             uint16_t gyroFlag : 1; /*Bit:3*/
42.             uint16_t magFlag : 1; /*Bit:4*/
43.             uint16_t resvFlag : 11; /*Bit:5~11*/
44.         };
45.         uint16_t dataSendFlag;
46.     };
47. }UDIMU_UART_HDR_ND2PC;
48.
49. /*根据 Flag 的不同可至多发送 16 种不同类型*/
50. typedef struct
51. {
52.     uint8_t addr;
53.     uint8_t code;
54.
55.
56.     /*[0, 4294967295]之间增加, 达到最大值后从 0 开始重新计数,
57.      注意若以 200Hz 上传数据, 此值约 248 天溢出一次*/
58.     uint32_t frameNum;
59.
60.     /*用位表示是否包含数据项*/
61.     union {
62.         struct {
63.             uint16_t quatFlag : 1; /*Bit:0*/
64.             uint16_t eulerFlag : 1; /*Bit:1*/
65.             uint16_t accelFlag : 1; /*Bit:2*/
66.             uint16_t gyroFlag : 1; /*Bit:3*/
67.             uint16_t magFlag : 1; /*Bit:4*/
```

```
68.             uint16_t    resvFlag : 11; /*Bit:5~11*/
69.         };
70.         uint16_t dataSendFlag;
71.     };
72.
73.     int16_t    quat[4];
74.     int16_t    euler[3];
75.     int16_t    accel[3];
76.     int16_t    gyro[3];
77.     int16_t    mag[3];
78.     uint16_t   crc16;
79. }UDIMU_UART_ND2PC;
80.
81. #pragma pack()
82.
83. UDIMU_UART_ND2PC    nd2pc;
84. uint8_t            globalBuff[CIRCLEBUFF_LEN];
85.
86.
87. ****
88. * Function:      Function Name
89. * Description:
90. * Input:
91. * Input:
92. * Output:
93. * Return:
94. * Others:        Other Description.
95. ****
96. void setup( )
97. {
98.     // put your setup code here, to run once:
99.     Serial.begin( 115200 );
100.    Serial1.begin( 115200 );
101. }
102.
103. ****
104. * Function:      loop
105. * Description: Arduino 程序主循环
106. * Input:
107. * Input:
108. * Output:
109. * Return:
```

```
110. * Others:  
111.     (1) 监听串口 1，将收到的数据放入循环缓冲区。  
112.     (2) 寻找姿态数据包的头部 也就是连续的两个字节 0x01 0x24。  
113. ****  
114. void loop( )  
115. {  
116.     if( Serial1.available( ) > 0 )  
117.     {  
118.         int numdata = Serial1.available( );  
119.         if( numdata > 1 )  
120.         {  
121.             byte b[64];  
122.             Serial1.readBytes( b, numdata );  
123.             //Serial.println( numdata );  
124.         }else  
125.         {  
126.             byte b[64];  
127.             Serial1.readBytes( b, numdata );  
128.  
129.             for( int i = 1; i < CIRCLEBUFF_LEN; i++ )  
130.             {  
131.                 globalBuff[i - 1] = globalBuff[i];  
132.             }  
133.             globalBuff[CIRCLEBUFF_LEN - 1] = b[0];  
134.  
135.             if( globalBuff[0] == 0x01 &&  
136.                 globalBuff[1] == 0x24  
137.                 )  
138.             {  
139.                 ParsePckage( globalBuff, CIRCLEBUFF_LEN );  
140.             }  
141.         }  
142.     }  
143. }  
144.  
145. ****  
146. * Function:          Function Name  
147. * Description:  
148. * Input:  
149. * Input:  
150. * Output:  
151. * Return:
```

```

152. * Others:这里的打印函数输出非常占用 Arduino 时间，打印的数据
153.          要比实际收到的数据少，实际收到的数据速率为 200Hz!
154.          可适当减少打印内容。
155. ****
156. void ParsePckage( uint8_t* buff, uint16_t size )
157. {
158.     uint16_t      crc16           = buff[size - 2] | ( buff[size - 1] << 8 );
159.     uint16_t      crc16Expect    = CRC16( buff, size - 2 );
160.
161.     if( crc16 != crc16Expect )
162.     {
163.         /*CRC16 校验错误，不执行后续代码*/
164.         return;
165.     }
166.
167.     int             frontptr = 0;
168.     UDIMU_UART_HDR_ND2PC   hdr;
169.     for( int i = 0; i < sizeof( hdr ); i++ )
170.     {
171.         *( (uint8_t *)( &hdr ) + i ) = buff[i];
172.     }
173.     /*打印帧序号，如不需要可去掉*/
174.     Serial.print( hdr.frameNum ); Serial.print( " " );
175.     frontptr += sizeof( hdr );
176.
177.
178.     /*! 最终数据！
179.      定义浮点数数据，由整型格式转换出浮点数表示的数据*/
180.     float   quat_f[4]  = { 0.0f, 0.0f, 0.0f, 0.0f }; /*w,x,y,z*/
181.     float   euler_f[3] = { 0.0f, 0.0f, 0.0f };
182.     float   accel_f[3] = { 0.0f, 0.0f, 0.0f };
183.     float   gyro_f[3]  = { 0.0f, 0.0f, 0.0f };
184.     float   mag_f[3]   = { 0.0f, 0.0f, 0.0f };
185.
186.     if( hdr.quatFlag )
187.     {
188.         int16_t quat[4] = { 0, 0, 0, 0 };
189.         memcpy( quat, &buff[frontptr], sizeof( quat ) );
190.         frontptr += sizeof( quat );
191.         quat_f[0]  = (float)quat[0] / ( 1 << 14 );
192.         quat_f[1]  = (float)quat[1] / ( 1 << 14 );

```

```
193.     quat_f[2] = (float)quat[2] / ( 1 << 14 );
194.     quat_f[3] = (float)quat[3] / ( 1 << 14 );
195.     /*打印四元数, 如不需要可去掉*/
196.     Serial.print( "Quat:" ); Serial.print( quat_f[0], 6 ); Serial.print
   ( " " ); Serial.print( quat_f[1], 6 ); Serial.print( " " );
197.     Serial.print( quat_f[2], 6 ); Serial.print( " " ); Serial.print( qu
   at_f[3], 6 ); Serial.print( " " );
198. }
199.
200. if( hdr.eulerFlag )
201. {
202.     int16_t euler[3] = { 0, 0, 0 };
203.     memcpy( euler, &buff[frontptr], sizeof( euler ) );
204.     frontptr += sizeof( euler );
205.     euler_f[0] = (float)euler[0] / ( 1 << 7 );
206.     euler_f[1] = (float)euler[1] / ( 1 << 7 );
207.     euler_f[2] = (float)euler[2] / ( 1 << 7 );
208.
209.     /*打印欧拉角, 如不需要可去掉*/
210.     Serial.print( "Euler:" ); Serial.print( euler_f[0], 6 ); Serial.pr
   int( " " );
211.     Serial.print( euler_f[1], 6 ); Serial.print( " " ); Serial.print( e
   uler_f[2], 6 ); Serial.print( " " );
212. }
213.
214. if( hdr.accelFlag )
215. {
216.     int16_t accel[3] = { 0, 0, 0 };
217.     memcpy( accel, &buff[frontptr], sizeof( accel ) );
218.     frontptr += sizeof( accel );
219.     accel_f[0] = (float)accel[0] / 2048.0f;
220.     accel_f[1] = (float)accel[1] / 2048.0f;
221.     accel_f[2] = (float)accel[2] / 2048.0f;
222.     /*打印加速度, 如不需要可去掉*/
223.     Serial.print( "Accel:" ); Serial.print( accel_f[0], 6 ); Serial.pr
   int( " " );
224.     Serial.print( accel_f[1], 6 ); Serial.print( " " ); Serial.print( a
   ccel_f[2], 6 ); Serial.print( " " );
225. }
226.
227. if( hdr.gyroFlag )
228. {
```

```

229.         int16_t gyro[3] = { 0, 0, 0 };
230.         memcpy( gyro, &buff[frontptr], sizeof( gyro ) );
231.         frontptr += sizeof( gyro );
232.         gyro_f[0] = (float)gyro[0] / 16.4f;
233.         gyro_f[1] = (float)gyro[1] / 16.4f;
234.         gyro_f[2] = (float)gyro[2] / 16.4f;
235.         /*打印角速度, 如不需要可去掉*/
236.         Serial.print( "Gyro:" ); Serial.print( gyro_f[0], 6 ); Serial.print
237.             ( " " );
238.         Serial.print( gyro_f[1], 6 ); Serial.print( " " ); Serial.print( gy
239.             ro_f[2], 6 ); Serial.print( " " );
240.     }
241.     if( hdr.magFlag )
242.     {
243.         int16_t mag[3] = { 0, 0, 0 };
244.         memcpy( mag, &buff[frontptr], sizeof( mag ) );
245.         frontptr += sizeof( mag );
246.         mag_f[0] = (float)mag[0] / 120.0f;
247.         mag_f[1] = (float)mag[1] / 120.0f;
248.         mag_f[2] = (float)mag[2] / 120.0f;
249.         /*打印磁力值, 如不需要可去掉*/
250.         Serial.print( "Mag:" ); Serial.print( mag_f[0], 6 ); Serial.print(
251.             " " );
252.         Serial.print( mag_f[1], 6 ); Serial.print( " " ); Serial.print( mag
253.             _f[2], 6 ); Serial.print( " " );
254.     }
255.     /*在此可插入回调函数, 将解析完成的单项或多项数据分发出去。*/
256.     static const uint8_t    CRCHi[] = {
257.         0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
258.         0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
259.         0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
260.         0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
261.         0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,

```

```
262.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
263.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
264.    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
265.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
266.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
267.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
268.    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
269.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
270.    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
271.    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
272.    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
273.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
274.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
275.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
276.    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
277.    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
278.    0x00, 0xC1, 0x81, 0x40  
279. };  
280.  
281. static const uint8_t    CRCLo[] = {  
282.    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,  
283.    0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E,  
284.    0xA, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9,
```

```
285.    0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
286.    0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
287.    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,
288.    0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,
289.    0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
290.    0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF,
291.    0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
292.    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,
293.    0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
294.    0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB,
295.    0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,
296.    0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
297.    0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
298.    0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97,
299.    0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E,
300.    0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89,
301.    0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
302.    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,
303.    0x41, 0x81, 0x80, 0x40
304. };
305.
306.
307. ****
308. * Function:          CRC16
```

```
309. * Description:    Return the CRC16 result of data stream
310. * Input:          pucFrame:Data stream repre by hex16
311. * Input:          usLen: Total length of data stream
312. * Output:
313. * Return:         CRC16 Results
314. * Others:        Other Description.
315. ****
316. uint16_t CRC16( uint8_t * pucFrame, uint16_t usLen )
317. {
318.     uint8_t ucCRCHi      = 0xFF;
319.     uint8_t ucCRCLo      = 0xFF;
320.     int      iIndex;
321.
322.     while( usLen-- )
323.     {
324.         iIndex      = ucCRCLo ^ *( pucFrame++ );
325.         ucCRCLo      = ( uint8_t )( ucCRCHi ^ CRCHi[iIndex] );
326.         ucCRCHi      = CRCLo[iIndex];
327.     }
328.     return ( uint16_t )( ucCRCHi << 8 | ucCRCLo );
329. }
330.
331. **** The End Of File ****
*****
```



附录 1 帧格式

帧格式说明：

<>：表示必须包含的部分。

[]： 表示可选的部分。

一般格式为

<帧头>+<命令字>+<>+[].....[]+<CRC16>

其中：

- (1) 帧头：为主/从设备在通信局域网中的标号，在这里由于结构为单主单从，帧头设置为 0x01 即可。
- (2) 命令字：标识当前包的功能。
- (3) CRC16：整个帧的 CRC16 校验码，低八位在前，高八位在后。

附录 2 CRC16 校验源码

由于数据传输需要，所以数据校验必不可少；在这里约定一种统一的 CRC16 校验方法，可以移植到任何支持 ANSIC 标准编译器的平台使用。

```
#include <stdint.h>

static const uint8_t CRCHi[] = {
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
```

```
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,  
0x00, 0xC1, 0x81, 0x40  
};  
  
static const uint8_t CRCLo[] = {  
    0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7,  
    0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0xF, 0xCF, 0xCE, 0xE,  
    0xA, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9,  
    0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,  
    0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,  
    0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,  
    0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,  
    0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,  
    0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF,  
    0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,  
    0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1,  
    0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,  
    0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB,  
    0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,
```

```

0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97,
0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E,
0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89,
0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,
0x41, 0x81, 0x80, 0x40
};

*****
```

* Function: CRC16
 * Description: Return the CRC16 result of data stream
 * Input: pucFrame:Data stream repre by hex16
 * Input: usLen: Total length of data stream
 * Output:
 * Return: CRC16 Results
 * Others: Other Description.

~~*****~~

```
uint16_t CRC16( uint8_t * pucFrame, uint16_t usLen )
```

```
{
    uint8_t ucCRCHi       = 0xFF;
    uint8_t ucCRCLo       = 0xFF;
    int      iIndex;
```

```
while( usLen-- )
{
    iIndex     = ucCRCLo ^ *( pucFrame++ );
```

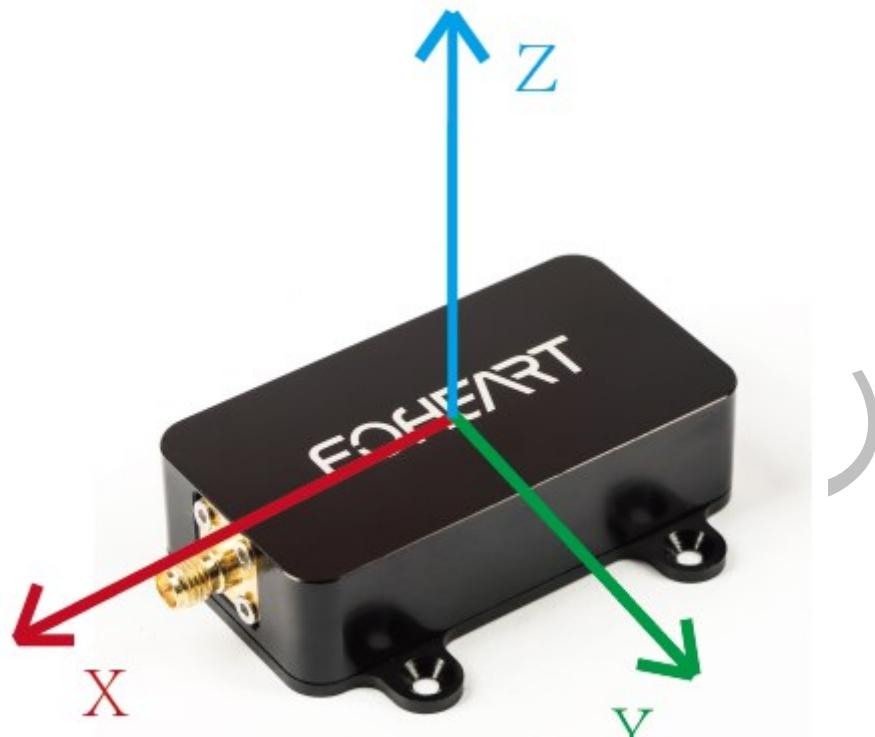
```
    ucCRCLo      = ( uint8_t )( ucCRCHi ^ CRCHi[iIndex] );
    ucCRCHi      = CRCLo[iIndex];
}

return ( uint16_t )( ucCRCHi << 8 | ucCRCLo );
}
```

PUBLIC

附录 3 坐标系定义

惯性单元 (IMU Sensor)，其坐标系定义如下：



符合该坐标系的输出值：

- (1) 加速度值
- (2) 角速度值
- (3) 磁力值
- (4) VRU 下的四元数欧拉角。

九轴融合后的四元数、欧拉角坐标系符合东(X)北(Y)天(Z)坐标系定义。

📞 (+86)010-56106165

✉ contact@foheart.com

🌐 www.foheart.com

📍 北京市海淀区黑山扈路红山口8号D2-南-3号

