用户手册

TCM3 & TCM5

1 版权警告信息

Revised June 2008; For most recent version visit our website at www.pnicorp.com
PNI Corporation
133 Aviation Blvd, Suite 101
Santa Rosa, CA 95403, USA
Tel: (707) 566-2260
Fax: (707) 566-2261

Warranty and Limitation of Liability. PNI Corporation ("PNI") manufactures its TCM products (“Products”) from parts and components that are new or equivalent to new in performance. PNI warrants that each Product to be delivered hereunder, if properly used, will, for one year following the date of shipment unless a different warranty time period for such Product is specified: (i) in PNI’s Price List in effect at time of order acceptance; or (ii) on PNI’s web site (www.pnicorp.com) at time of order acceptance, be free from defects in material and workmanship and will operate in accordance with PNI’s published specifications and documentation for the Product in effect at time of order. PNI will make no changes to the specifications or manufacturing processes that affect form, fit, or function of the Product without written notice to the OEM, however, PNI may at any time, without such notice, make minor changes to specifications or manufacturing processes that do not affect the form, fit, or function of the Product. This warranty will be void if the Products' serial number, or other identification marks have been defaced, damaged, or removed. This warranty does not cover wear and tear due to normal use, or damage to the Product as the result of improper usage, neglect of care, alteration, accident, or unauthorized repair.
THE ABOVE WARRANTY IS IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE. PNI NEITHER ASSUMES NOR AUTHORIZES ANY PERSON TO ASSUME FOR IT ANY OTHER LIABILITY.

If any Product furnished hereunder fails to conform to the above warranty, OEM’s sole and exclusive remedy and PNI’s sole and exclusive liability will be, at PNI’s option, to repair, replace, or credit OEM’s account with an amount equal to the price paid for any such Product which fails during the applicable warranty period provided that (i) OEM promptly notifies PNI in writing that such Product is defective and furnishes an explanation of the deficiency; (ii) such Product is returned to PNI’s service facility at OEM’s risk and expense; and (iii) PNI is satisfied that claimed deficiencies exist and were not caused by accident, misuse, neglect, alteration, repair, improper installation, or improper testing. If a Product is defective, transportation charges for the return of the Product to OEM within the United States and Canada will be paid by PNI. For all other locations, the warranty excludes all costs of shipping, customs clearance, and other related charges. PNI will have a reasonable time to make repairs or to replace the Product or to credit OEM’s account. PNI warrants any such repaired or replacement Product to be free from defects in material and workmanship on the same terms as the Product originally purchased.

Except for the breach of warranty remedies set forth herein, or for personal injury, PNI shall have no liability for any indirect or speculative damages (including, but not limited to, consequential, incidental, punitive and special damages) relating to the use of or inability to use this Product, whether arising out of contract, negligence, tort, or under any warranty theory, or for infringement of any other party’s intellectual property rights, irrespective of whether PNI had advance notice of the possibility of any such damages, including, but not limited to, loss of use, revenue or profit. In no event shall PNI’s total liability for all claims regarding a Product exceed the price paid for the Product. PNI neither assumes nor authorizes any person to assume for it any other liabilities.

Some states and provinces do not allow limitations on how long an implied warranty lasts or the exclusion or limitation of incidental or consequential damages, so the above limitations or exclusions may not apply to you. This warranty gives you specific legal rights and you may have other rights that vary by state or province.

2 PNI 公司的 TCM3&TCM5

感谢使用 TCM3/TCM5。您选择了一款代表今后若干年指南针发展方向的产品。TCM3/TCM5/5L 是一个代表现代科技水平的，低功耗的，高性能的带倾斜补偿的电子指南针传感器模组。

TCM3/TCM5 采用了先进的算法对模组进行了软磁和硬磁校正，提供了精确的方位信息。模组的输出信息表示了模组当前的姿态位置，可以用于需要 360°旋转的系统当中(仅 TCM5)。TCM 是目前市场上是体积最小的将三轴磁传感，三轴倾斜传感以及指南针集成到一起模组。由于小巧的体积，TCM3/TCM5 非常适合现代空间要求高的系统。这些优点都使 PNI 公司的 TCM3/TCM5 成为众多高精度，高性能应用的选择。

TCM3/TCM5 结合了 PNI 公司专利的磁感应传感器和三轴加速度计测量电路技术，具有极佳的性价比。磁传感器和加速度计在-40 到 85℃的环境下进行标定。因此在全温度范围里测量结果也是非常稳定的。

TCM3/5/5L 的优点使其非常适用于许多应用，包括：

- 高性能固态导航仪器
- 高性能姿态测量
- 惯性测量装置(IMU Inertial Measurement Unit)集成
- 机器人系统
- 激光测距仪
- 钻井应用

由于有许多潜在的应用，TCM3/5/5L 提供了灵活方便的命令设置。许多参数都是用户可编程的，包括单位，大范围的采样配置等等。我们希望 TCM3/5/5L 能帮助您的目标系统实现最好的性能。感谢您选用 TCM3/5/5L。

2.1 性能规格

方位规格

参数	TCM3	TCM5	单位
倾斜<70°的精度	0.5°	0.3°	度 RMS
倾斜>70°的精度	0.8°	0.5°	度 RMS
分辨率	0.1°	0.1°	度 RMS
重复性[1]	0.05°	0.05°	度 RMS
最大倾角	85°	85°	度

[1] 重复性是基于±3 sigma 限制的统计数据.

磁力计规格

参数	TCM3	TCM5	单位
标定测量范围	±80	±80	μT
磁场强度分辨率	±0.05	±0.05	μT
磁场强度重复性	±0.1	±0.1	μT

倾斜规格

参数	TCM3	TCM5	单位
倾斜精度	0.2°	0.2°	Deg RMS
翻转精度	0.2° for pitch <65° 0.5° for pitch <80°	0.2° for pitch <65° 0.5° for pitch <80° 1.0 for pitch <86°	Deg RMS
倾斜范围	±80°	±90° pitch ±180° roll	Deg
倾斜分辨率	<0.01°	<0.01°	Deg
倾斜重复性[1]	0.05°	0.05°	Deg RMA

[1] Repeatability is based on statistical data at ±3 sigma limit about the mean.

校正

参数	TCM3	TCM5
硬磁校正	有	有
软磁校正	有	有

机构规格

参数	TCM3	TCM5	单位
体积 (LxWxH)	3.5 x 4.3 x 1.3	3.5 x 4.3 x 1.3	cm
重量	12	12	克
安装方式	螺钉安装/支座 水平	螺钉安装/支座 水平或垂直	
RS232连接器	9-pin	9-pin	

IO 口规格

参数	TCM3	TCM5	单位
上电延迟	<50	<50	毫秒
休眠延迟	<1	<1	毫秒
最小采样速率	20	20	采样点/秒
RS-232 通信速率	300 to 115200	300 to 115200	baud
输出格式	二进制协议		

电源规格

Parameter	TCM3	TCM5	Units
供电电压	3.6 to 5 V	3.6 to 5 V	VDC
电流* **Max:** **Typ:**	22 <20	22 <20	mA
空闲模式	14 - 18	14 – 18	mA
休眠模式	0.6	0.6	mA

*连续输出

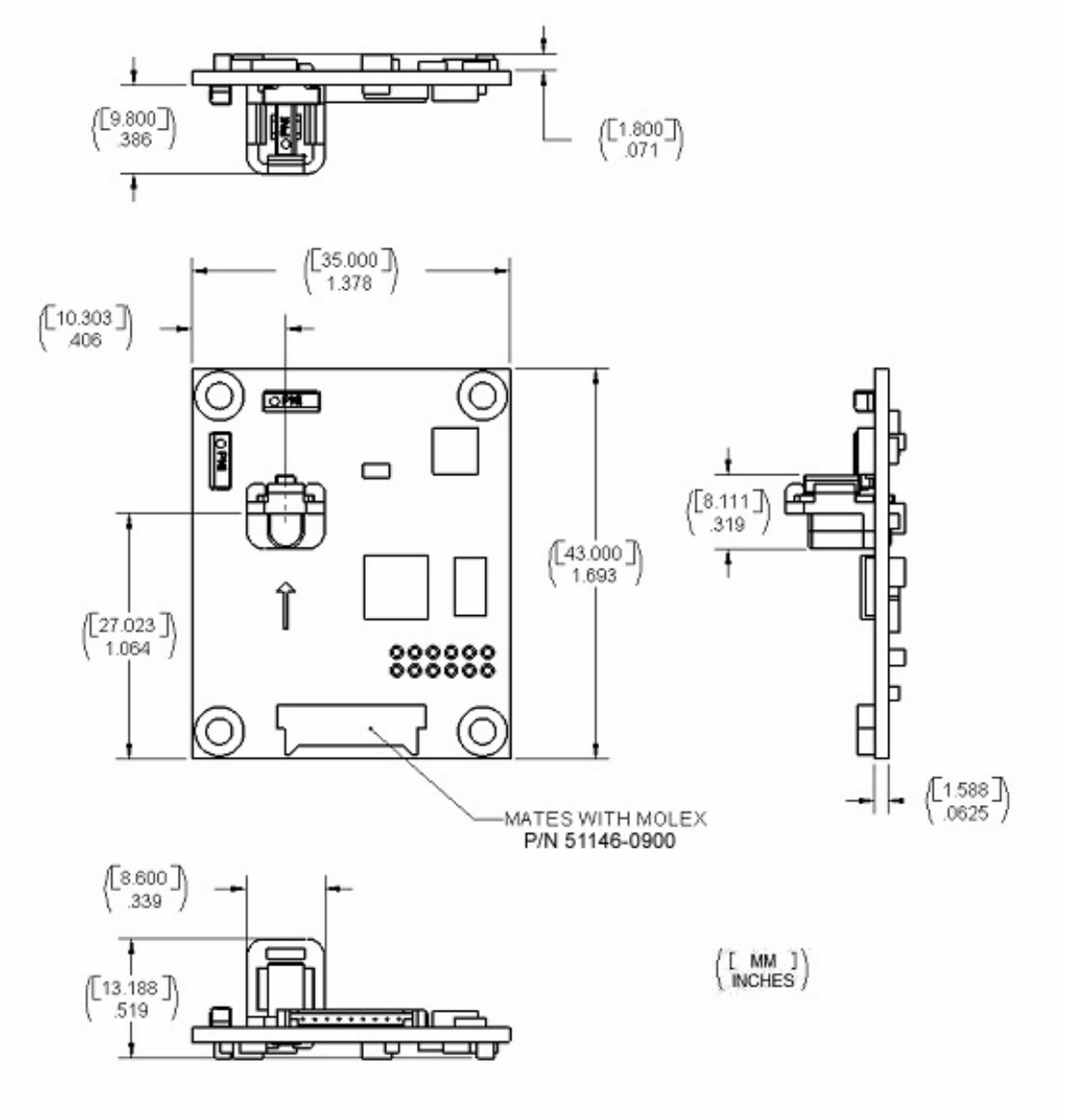
环境规格

参数	TCM3	TCM5	单位
工作温度	-40º to 85º	-40º to 85º	℃
存储温度	-40º to 125º	-40º to 125º	℃
震动	Up to 2500 G's per MIL-STD-810F		
抖动	Qualified to MIL-STD-810F		
湿度	Non-condensing/Qualified to MIL-STD-810F		

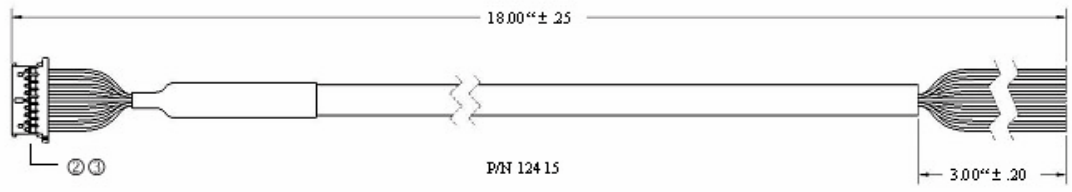
2.2 机构

2.2.1 TCM3/TCM5

TCM3/TCM5 默认丝印箭头指向前方，也就是把放置 Molex 连接器的反向作为板子的前端。



2.2.2 18 英寸线缆装配



①Molex p/n 51146-0900

②Molex p/n 50641-8141

TCM3/TCM5 管脚定义

管脚	引线颜色	描述
1	黑	电源地
2	灰	NC
3	绿	R2-232 地
4	橙色	NC
5	紫罗兰	NC
6	棕	NC
7	黄	TxD
8	蓝	RxD
9	红	5 VDC

3 TCM3&TCM5 的安装

本节将介绍如果在您的系统中配置，编程和控制 TCM3/TCM5。在您的系统中安装 TCM3/TCM5，需按以下步骤：

- 给 TCM3/TCM5 建立电气连接
- 利用 TCM Studio 程序评估 TCM3/TCM5
- 选择安装位置
- 安装 TCM3/TCM5
- 用户校正

在安装模组之前，可以通过 TCM Studio 在您的系统以外进行评估，请参考 4.1 节。

3.1 电气连接

TCM3/TCM5 的接口配件是一条线缆，用于将模块连接到您的系统。线缆的末端需要能和 TCM3/TCM5 匹配，线缆中的彩色引线的定义如下。

PNI 同时也提供了一个带 DB9 接头的线缆，请联系 PNI 获取购买信息。

TCM3/TCM5 管脚定义

管脚	引线颜色	描述
1	黑	电源地
2	灰	NC
3	绿	R2-232 地
4	橙色	NC
5	紫罗兰	NC
6	棕	NC
7	黄	TxD
8	蓝	RxD
9	红	5 VDC

3.2 安装在什么地方

TCM3/TCM5 的磁力计具有大的动态的范围，同时尖端的校正算法使模组能够适用于各种复杂的环境。但为了保证达到最佳的性能，您在安装 TCM3/TCM5 的时候应该遵循以下几点意见：

TCM3/TCM5 的磁力计不能饱和

TCM3/TCM5 可以通过用户校正来补偿主机系统所产生的静态磁场。TCM3/TCM5 磁力计的每个轴的最大动态范围是±80uT。如果任何一个轴的磁场强度超过这个值，那么 TCM3/TCM5 将不能得到精确的方位信息。在安装 TCM3/TCM5 的时候，要综合考虑每个磁场源的影响，在加上地球磁场的情况下可能使 TCM3/TCM5 磁传感器饱和。举个例子来说，大型的含铁设备例如变压器和汽车底盘，大电流，恒磁体例如电动马达等等。

TCM3/TCM5 的安装位置应远离变化的磁场源

目前为止还不能对变化异常的磁场进行标定，因此，为了达到良好的精度，应保持 TCM3/TCM5 尽量远离系统中变化的磁场源。举例来说，电气设备的开关或者附件的含铁物体位置发生变化，确保 TCM3/TCM5 不会被放在离可能大的磁场的物体附近。

TCM3/TCM5 应该被安装在物理环境比较稳定的位置

TCM3/TCM5 的安装位置应该与剧烈的震动，振荡和抖动隔离开。

测试

测试应该安排在开发的早期阶段，用于了解失真区域的范围和瞬变情况，进而考虑元件的安装位置。

为了确定失真区域的范围，将指南针放置在一个固定的位置，然后移动/给有影响的元件上电，同时观察指南针的输出来确定什么时候有影响。

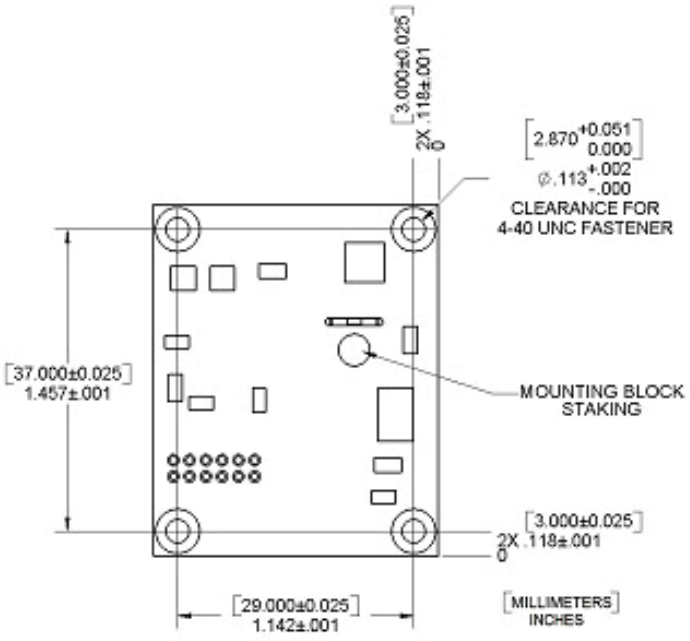
为了确定安装位置的磁场强度是否在指南针的动态范围之内，需要进行以下的测试：

在安装指南针的情况下，尽可能多的旋转和倾斜系统，与此同时，监控磁力计的输出，观察是否超出最大的动态范围，我们建议尽可能的使模组的动态范围留有余量。

3.3 机械安装 TCM3/TCM5

有关 TCM3/TCM5 的尺寸规格在本手册中的随后部分有为 TCM3/TCM5 电路板尺寸和方位的介绍。

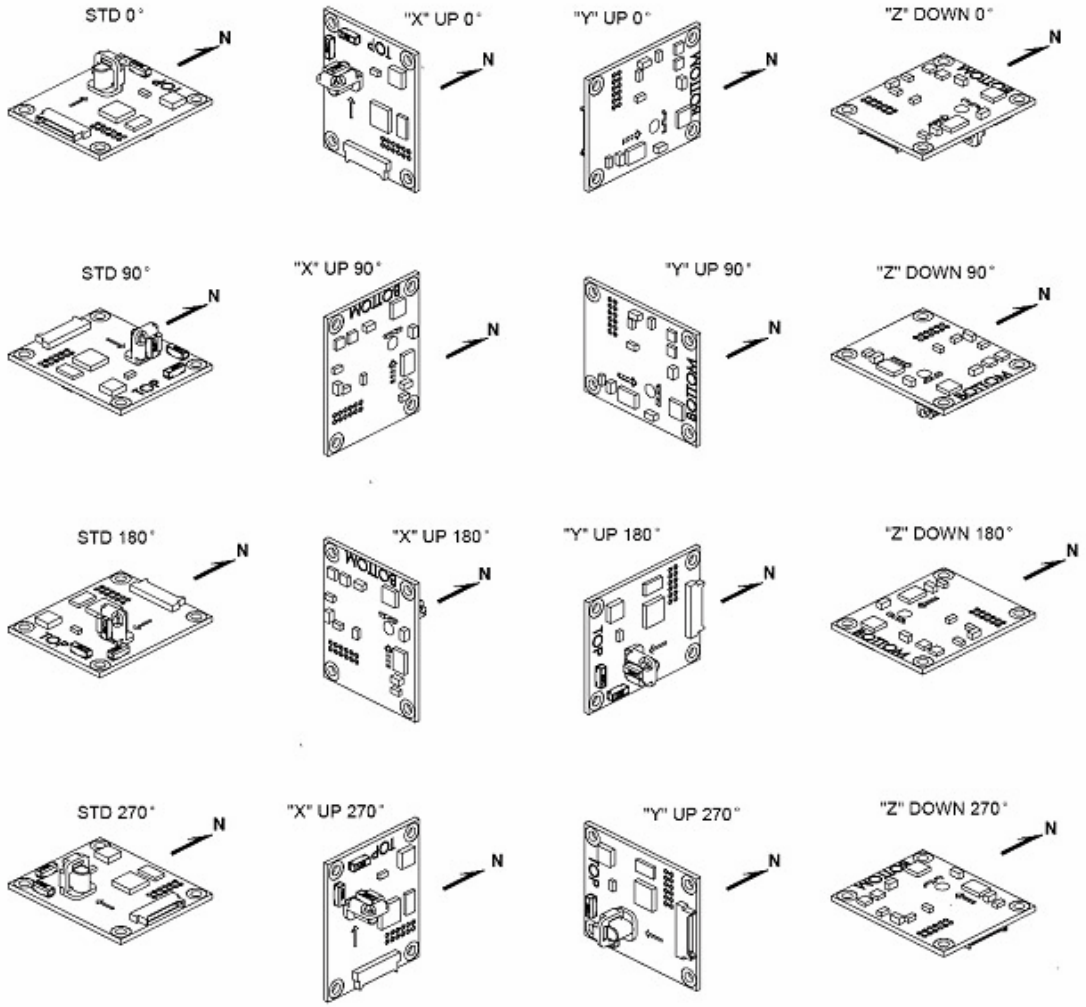
TCM3/TCM5 的工厂标定都是相对于定位孔的，示意图如下，因此在系统中应该以定位为基准来对齐，而不能以电路板的边缘为基准。



安装选项

TCM3/TCM5 的安装具有极大的灵活性，可以安装在不同的位置上。所有的参考点都是基于电路板顶端白色的丝印箭头。

注意：下面图中的电路板只是作为指示用，并不是真实的 TCM3/TCM5 电路板。



4 TCM3&TCM5 的使用

- TCM Studio
- 用户标定
- 二进制协议
- 示例代码

4.1 TCM Studio

TCM3/TCM5 软件通过 PC 的 COM 口实现与 TCM3/TCM5 模块之间的通信，它提供了一个简单易用的二进制语言的接口，用户可以使用按钮，对话框等来替代手动发送二进制的命令。TCM3/TCM5 评估软件读取 TCM3/TCM5 的输出字符串并将其格式转化为易读的数据形式。该程序同时具有将采集到的 TCM3/TCM5 的数据以文件的形式保存。这些都将使您在刚开始学习 TCM3/TCM5 的时候获得非常友好的 TCM Studio 程序接口。可以到 PNI 的网站 www.pnicorp.com 查阅最新的版本。

18 英寸线缆装配

1. 拖动”TCM Studio.exe”到您的电脑的工作目录下。

2. 移动 Quesa.dll 到 windows system 或者 system32 文件夹下，Quesa 是一个 OpenGL 演示引擎，没有它 TCMStudio 的 3D 模型将不能工作。
    - Windows2000/NT 拷贝至： /WinNT/System32 文件夹
    - Windows XP 拷贝至：/Windows/System32 文件夹

将 TCM Studio 安装到 Mac OSX 系统中：

1. 拖动”TCM Studio.exe”到您的电脑的工作目录下。

2. 将 Quesa 插件移动到：/Library/CFMSupport

连接页面

初始连接：

1. 选择 38400 作为波特率
2. 选择您连接的串口号
3. 点击<Connection>按钮
4. 一旦“连接”指示灯变绿，模组的固件版本，序列号都会显示出来。

更改波特率：

1. 为模组选择新的波特率
2. 点击<Power Down>按钮
3. 在电脑上设置同样的波特率
4. 点击<Power Up>按钮

更换模组：

一旦连接被建立起来，TCM Stduio 会记录上一次的设置。随时将模组拔掉，换上一个新的模组点击<Connecti>按钮，软件会自动根据前一次的波特率进行连接。

4.1.3 配置页面

注意：一定要点击<SAVE>按钮设置才会生效。

Mounting options：

注意：如果选项是灰色的或者没有列出来，表示不支持该项功能。同时也是“机械安装”部分“安装选项”的图片的文字说明。

Standard：选择该项时，主板上的传感单元被水平安装(Z 轴的磁传感器垂直)。

Standard 90 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方顺时针旋转 90 度的方向。

Standard 180 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方旋转 180 度的方向。

Standard 270 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方顺时针旋转 270 度的方向。

X Sensor Up：当选择该项时，主板上的传感单元被垂直安装(X 轴传感器垂直)。

X Sensor Up Plus 90 Degrees：当选择该项时，主板上的传感单元被垂直安装(X 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 90 度。

X Sensor Up Plus 180 Degrees：当选择该项时，主板上的传感单元被垂直安装(X 轴传感器垂直)，同时沿主机系统的前方旋转 180 度。

X Sensor Up Plus 270 Degrees：当选择该项时，主板上的传感单元被垂直安装(X 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 270 度。

Y Sensor Up：当选择该项时，主板上的传感单元被垂直安装(Y 轴传感器垂直)。

Y Sensor Up Plus 90 Degrees：当选择该项时，主板上的传感单元被垂直安装(Y 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 90 度。

Y Sensor Up Plus 180 Degrees：当选择该项时，主板上的传感单元被垂直安装(Y 轴传感器垂直)，同时沿主机系统的前方旋转 180 度。

Y Sensor Up Plus 270 Degrees：当选择该项时，主板上的传感单元被垂直安装(Y 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 270 度。

Z Sensor Down：当选择该项时，主板上的传感单元被垂直安装(Z 轴传感器垂直)。

Z Sensor Down Plus 90 Degrees：当选择该项时，主板上的传感单元被垂直安装(Z 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 90 度。

Z Sensor Down Plus 180 Degrees：当选择该项时，主板上的传感单元被垂直安装(Z 轴传感器垂直)，同时沿主机系统的前方旋转 180 度。

Z Sensor Down Plus 270 Degrees：当选择该项时，主板上的传感单元被垂直安装(Z 轴传感器垂直)，同时沿主机系统的前方顺时针旋转 270 度。

North Reference:

Magnetic: 当“Magnetic”被选中时，指南针方位会相对于磁北。

Ture: 当”True”被选中时，指南针方位会相对于真北。使用”True”模式时，需要在”Declination”窗口中填入磁偏角。请参考”使用 TCM 磁偏角值”的章节以获取更多的信息。

Endianess：选择数据的大端模式还是小端模式，默认为大端模式。

Filter Setting: 选择 0(无滤波)，4，8，16 或者 32 个采样点，并利用这些数据在计算方位之前进行 FIR 滤波。进行滤波可以使读数更加稳定，但是会使得数据刷新变慢，默认采样点为 32 个。

Acquisition Parameter：

Mode：

- 当”Poll”被选中时，TCM Studio 会从传感器模组中请求数据，一旦接受到来自模组的数据，TCM Studio 程序会以”Poll Time”框中设置值为间隔时间再次请求数据。如果被设置成 0，则 TCM Studio 会在接收到数据后立刻请求下一次数据。

- 当”Push”被选中时，传感器模组将进入间隔模式。一旦模组被设置成为间隔模式，在”Interval Time”设置框中设置间隔时间，模组将会以期望的间隔时间发送当前的数据，如果间隔时间被设置成为 0，模组将会在发送完上一组数据之后立刻发送新的数据。

Acquire Time：

“Acquire Time”设置框用于设置模组的采样时间，这个是用于进行模组内部设置的，与模组传送数据给程序或主机的时间无关。

Flush Filter：

该滤波设置只是用于更新滤波器中的最后一个采样值，例如一旦初始的 32 个采样值都采样完成了，那么在最后加入新的采样值的同时，第一个采样值将会被放弃。在这种情况下，”Acquire Time”被设置为某个值，在计算方位之前清除滤波器。这种清除方式要求模组每次在计算的时候都用新的 32 个采样值。

注意：如果”Flush Filter”被选中，那么数据更新将会花费更多的时间。

用户标定设置:

Stability Checking:

在默认的情况下，模组在标定模式下在保存用于标定的采样数据之前将优先等待读取 3 个连续稳定输出值。这就是模组在标定过程中在某个标定点需要保持稳定的原因。这个稳定的过程有助于确保得到一个合适的方位值，同时达到更好的精度，但是这样会花费更多的时间。如果用户不选择该项，模组在标定的时候将没有稳定时间，只要输出值大于标定点之间的最小变化值，就将读取该数据作为标定值。

Automatic Sampling：

如果选择自动采样，模组将在满足最小变化和稳定性通过的情况下，自动进行采样。如果用户需要对模组有更多的控制，那么请取消选中该项。一旦取消选中，在 **Calibration** 页面的**<Take sample>**按钮将会被激活。点击**<Take sample>**按钮在满足条件的情况下表示进行一次标定采样。

Calibration Points：

用户可以自由选择标定的点数。可以成功标定的最小的点数为 12 个。作为需要标定的 12 个点，模组在水平面内需要被转动最少 180 度，最少一次正向和负向倾斜，最少一次正向和负向翻滚。

Enable 3D Model：

一些计算机系统可能不具备图形处理能力来演示 3D 模型，所以这类计算机必须关闭这一功能。

Default：

这个按钮将把 TCM Studio 程序恢复为出厂时的默认设置。

Revert：

这个按钮将使 TCM Studio 程序读取模组的设置，并显示在屏幕上。

4.1.4 标定页面

注意：模组的默认设置推荐为最好的精度及最佳的标定质量。

Samples：

1、点击<Start>按钮开始标定

2、要对一个点采样，模组需要稳定一小段时间。一旦窗口显示下一个数，模组需要移动一定距离并保持稳定以进行下一次采样。方位或倾斜至少需要 30 度的改变量才能进行采样。两个采样点间的距离越大越好。标定时的纵横摇量将决定模组在使用中能补偿的纵横摇量。一旦达到预先设置的采样数目，则标定完成。

注意：模组至少需要 12 个点才能进行一次成功的标定。在这 12 个点中，模组需要在水平面内至少旋转 180 度，最少一次正向和负向的纵横摇。

Results：

1、完成标定后，“Coverage”窗口将会显示标定量。X，Y，Z 的值显示了在标定中占每个轴向上的比例。唯一的一种使 Z 值大于 50%的方法是使得一些点随模组上下翻转。以 uT 为单位显示的值表示了这些采样点测量值的标准差，这个值越小越好。如果需要一个更好的评分，则点击<Start>开始一次新的标定。

注意：以 uT 为单位显示的值仅仅表示标定的效果而不是表示准确的方位。如果模组在使用中所处的场和标定时的场不一致的话，可能出现标定“好”但是精度很差的情况。

2、如果标定满足要求，则点击<Save>按钮保存标定。如果没有选择这个按钮，则模组需要在上电之后需要重新进行标定。

Current Configuration:

Stability Checking: 表示 Stability Checking 选项是否被勾选。

Automatic Sampling: 表示 Automatic Sampling 选项是否被勾选。

Number of samples is: 表示当前标定中所用的采样数。

Options:

Audible Feedback: 如果选择 TCM Studio，在采样时将给出一个发声信号。

Clear:

该按钮将清除用户对模组的标定。一旦选定，模组将设置成原厂的标定。

4.1.5 测试页面

Current Reading:

当<GO>按钮被选中时，模组将开始输出方位，倾斜和翻滚数据。选中<Stop>按钮或切换标签，将中止模组的输出。

Contrast:

反转当前读数窗口的背景颜色。

Acquisition Settings:

这一窗口显示了相关的设置信息。

3D Model:

直升机将会随附加的模块运动并且给出该模块确切的运动方向。

4.1.6 数据记录器界面

1、选择数据登入“Data”窗口。
2、使用 Shift-Ctrl-Click 和 Ctrl-Click 来选中多个选项。
3、点击<GO>按钮开始录入数据，点击<Stop>按钮停止数据录入。
4、点击<Export>按钮将数据保存到一个文件中。
5、点击<Clear>按钮清除窗口中的数据。

注意：数据记录器使用标记作为时间基准，一个标记为 1/60 秒。

4.1.7 系统记录界面

Export:

选择<Export>按钮将系统记录保存到一个文件中。

Graph

图表提供了一个2轴(X,Y)的被测场强的点迹，该图表能够比较直观的显示利用TCM模组测量到的硬磁和软磁的影响，同时也可以看到用户标定后的校正数据的输出情况。

4.2 用户标定

所有的罗盘都能在受控的环境下有良好的性能，这个环境周围的磁场完全由地球磁场组成。然而在绝大多数情况下，电子罗盘将会安装在例如像汽车这样的自身有较大局部磁场源的主机系统中：含铁的金属底盘，变压器铁心，电流以及电动机中的永磁体。

通过执行用户标定程序，您可以使用TCM3/TCM5来识别这些局部磁场异常的磁场源，随后在测量地球磁场以计算罗盘方位的过程中消除它们的影响。当您执行用户标定程序时，TCM3/TCM5将进行一系列的磁场测量。它会分析全部的磁场测量，从那些由局部磁场产生的部分（我们希望扣除的信号）中识别出由地球磁场产生的部分（即所需信号）。

以上程序的最终目的是使TCM3/TCM5在其安装位置能够测量精测量出主机系统所产生的静态三维磁场矢量，随后这个矢量将从实时的磁场测量中扣除来得到地球磁场矢量。

TCM3/TCM5的三维磁力计和三维加速度计的一个重要作用是它能用倾斜幅度来补偿所有方位的失真。前面我们提到过，为了进行准确的标定，罗盘必须测量出它在主机系统的当前位置上所产生的局部磁场矢量。由于TCM3/TCM5的磁力计是接连或者固定于主机系统之上，当主机系统的方位改变时，这个局部的磁场矢量不变，使得TCM3/TCM5在任何倾斜和翻滚方向都能得到精确校正。而对于万向磁选通器这样的系统，在非水平的方向上不能进行精确的标定。这是由于它的磁力计是万向的，会随着主机方位的变化而改变位置，在这种情况下会出现不同于标定时情况的局部变形场。

关键点

- 模组能进行成功标定的最少点数为12。
- 模组需要在水平面上旋转至少180度，包括至少一次正向和一次负向的倾斜和翻滚运动。
  1. 在标定的过程中尽可能的倾斜，这使得罗盘能充分利用3轴磁力计的优势。
- 您可以尽量的尝试对尽可能多的方位和偏向的磁场进行均匀采样，可能的话包括倒扣。
- 注意coverage的比例，比例越低则罗盘的精确度越低。

4.2.1 标定原理

准确的标定方法取决于实际的标定参数的设置。不同的设置及其效果举例可参看TCM Studio –Evaluation Software部分的介绍。

标定的主要目标是使TCM3/TCM5校正由主机系统产生的变形磁场。为此TCM3/TCM5需要安装在主机系统当中，在标定时整个系统的运动被看成是一个模组。标定的运动包括至少180度的水平转动，但是为了达到最佳的精度，需要360度的水平转动，同时在校正中应包括尽可能多的不同倾斜角度。

为了TCM3/TCM5在整个的倾角范围内达到最好的精度，模组需要在整个量程内倾斜。例如，如果一个模组纵横摇的范围为40度，那么TCM3/TCM5的方位信息也将只会在40度纵横摇范围内准确。

4.2.2 硬磁和软磁影响

硬磁失真是由恒磁体或者带磁性的钢或者铁靠近传感器造成的。这种类型的失真对于相对位置固定的传感器在所有方向上的失真影响使恒定的。硬磁失真会在传感器的每个轴向上都叠加上一个固定的磁场分量，所以硬磁失真可以利用简单的算法进行补偿。

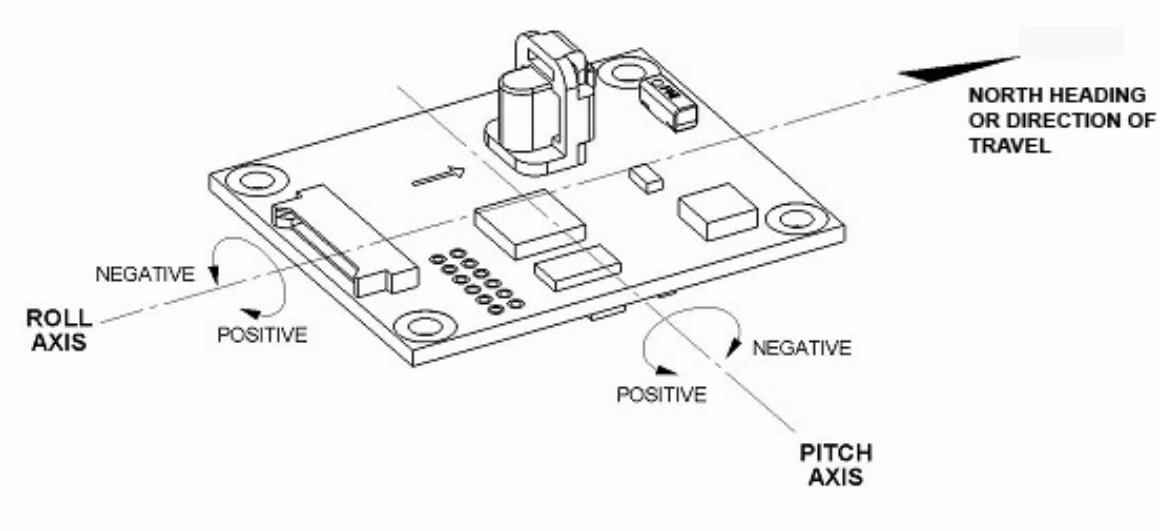
软磁失真是由地球磁场和软性磁材料接近传感器共同作用的结果。从技术上来说，软磁材料具有高的可磁导率。已知材料的磁导率可以作为磁力线穿过能力的度量，相对于大气（大气的磁导率为1）。

4.2.3 倾斜和翻转

TCM3/TCM5利用加速度计来测量相对于万有引力的指南针方向。既然指南针测量完整的磁场强度，TCM3/TCM5能够补偿指南针的倾斜，使指南针达到最佳的精度。

TCM3/TCM5利用欧拉算法来确定准确的方位。该方法与飞机方位表示方法一致，输出包括航向，纵横摇。当使用欧拉角的时候，纵横摇被定义为沿机身中心轴旋转的角度。倾斜式沿通过机翼中心的轴旋转。因为另一个旋转以飞机机体为轴进行旋转，所以这两种旋转是相互独立的。

对于TCM3/TCM5而言，当电路板前沿向上旋转的时候为正向的倾斜，当电路板的右边缘向下旋转的时候为正向翻滚。



TCM3/TCM5标准安装方式

4.2.4 最小采样点数的标定过程

利用TCMStudio应用程序和演示板的线缆可以对TCM3/5/5L系列模组进行标定，提供标定的指令。TCMStudio应用程序的所有功能都可以通过TCM模组的二进制协议来实现，该过程可以被移植到用户的嵌入式方案中。标定的次序给出了一个最小点数采样的分配方式，另外为了获得更好的标定效果，可以增加更多的采样点。

1）TCM模组连接好，并与TCMStudio建立通信连接，打开configuration页面。
2）配置模组的步骤如下：
在Filter Settings窗口中设置为32。
Calibration Setting：选择 Stability Checking check box
选择 Automatic Sampling
选择标定点数为12
3）点击Save按钮
4）打开标定页面

注意：一旦开始进行采集标定点，在预定的采样点之间停留，会被自动采样程序认为是标定点而采集。
在标定的过程中，应该移动模组到指定的位置，这些位置并不是需要绝对的方位，而是大致的相对于您标定的第一个点方位的一个相对位置。您不需要知道真北在什么位置，请按以下方式对12点进行采样：

稍微倾斜模组

- 0度+小的正向翻滚
- 90度+小的负向翻滚
- 180+小的正向翻滚
- 270度+小的负向翻滚

模组倾斜50度

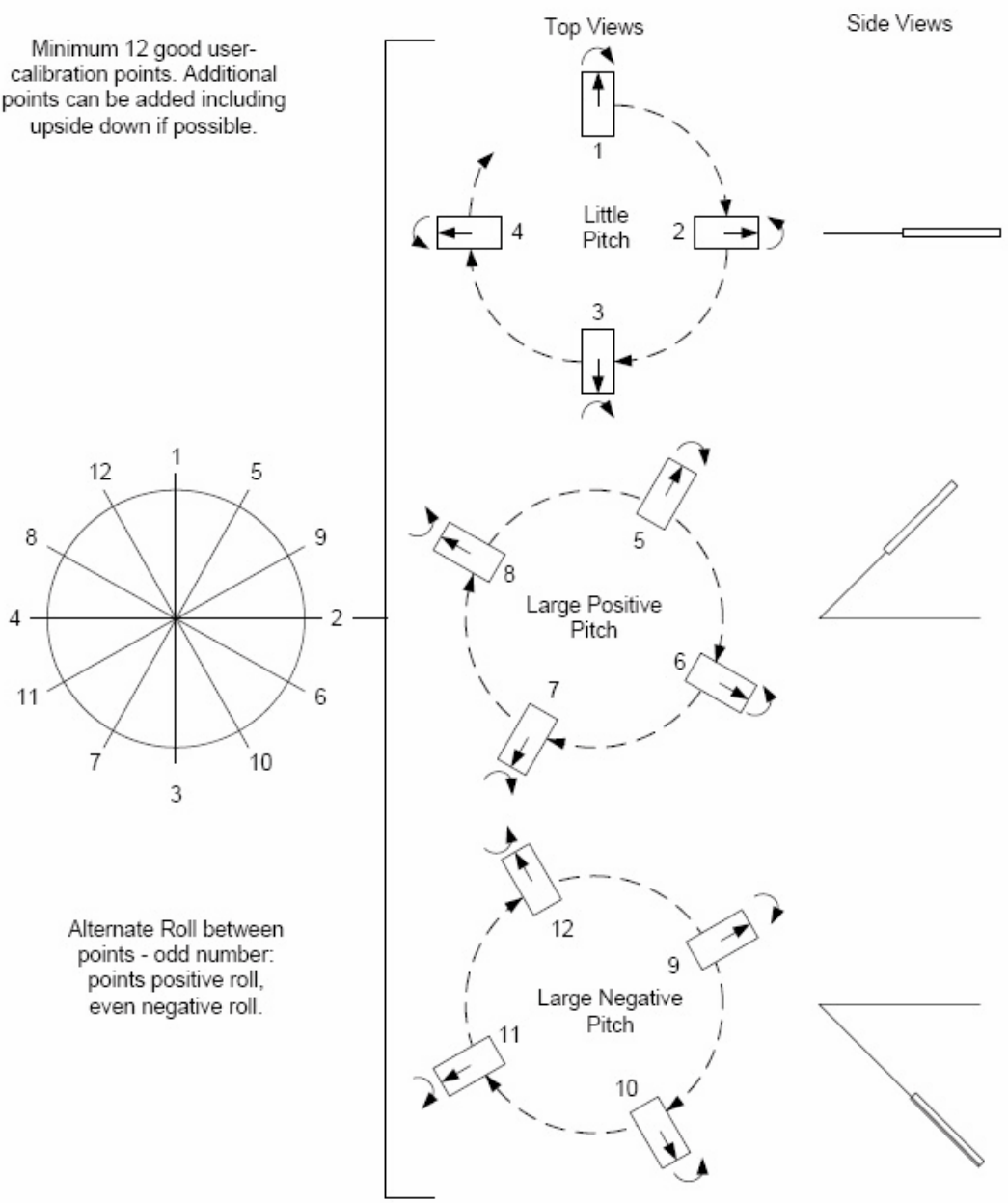
- 0度+小的负向翻滚
- 90度+小的正向翻滚
- 180+小的负向翻滚
- 270度+小的正向翻滚

模组倾斜-50度

- 0度+小的正向翻滚
- 90度+小的负向翻滚
- 180+小的正向翻滚
- 270度+小的负向翻滚

5）保持模组稳定
6） 点击Start按钮，等待自动采样
7） 旋转模组到下一个方位，大约为90度，保持模组稳定直到该点被采样
8） 重复上述步骤直到12点全部采样完毕
9） 点击Save按钮保存
10） 标定结果将显示在Results窗口中，以X，Y，Z的覆盖范围(%为单位)以及磁场强度的标准差来表示(uT为单位)。覆盖范围得分是以百分比来表示每个轴的传感器的失真被补偿的效果。理想的效果是X，Y的得分在85%以上，使用以上的方法Z轴的得分会在50%以下。标准差得分应该好于0.1uT。标准差得分表示失真

能够被补偿到什么程度。如果在标定的过程中，引起失真磁场相对于模组发生变化的话，那么补偿结果的得分将会很差，另外比较大的磁场环境噪声也可能使得分比较差。



4.2.5 磁偏角

磁偏角，也被称为磁差，与真北和磁北有所不同，相对于地球的某个点。它可以通过真北偏东或者偏西多少度来表示。磁偏角的修正通过存储准确的磁偏角度值，然后将方位参考由磁北变为真北。世界上各地的磁偏角角度都不一样，同时随着时间的变化缓慢的变化。为了指南针能够达到最好的精度，去国家地理数据中心网站上查询您所在经纬度位置的磁偏角值：

http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp

4.2.6 其他限制

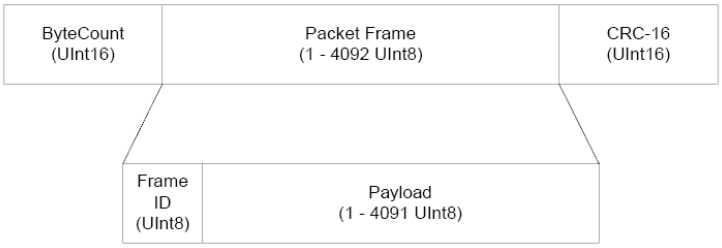
像先前讨论的一样，TCM3/TCM5 模组受到的局部干扰被作为静态的磁场矢量叠加到地球的磁场上。任何的局部磁场，只要不是静态的，就会造成错误。您不可能对不规则变化的磁场进行标定。例如，您可能指挥刀 TCM3/TCM5 会用于非常靠近车辆的地方，但是您不能对相当于 TCM3/TCM5 位置变化的车辆进行标定。这样的限制对于目前所有的指南针都一样。因此，TCM3/TCM5 的位置相对于任何潜在的磁场源将不会是静态的：磁性货物或者负载可能会放置到附近，风扇或者其他电气设备的开或关，等等。

TCM3/TCM5 能够对所有没有超出其传感器动态范围的环境磁场进行标定。

4.3 二进制协议—RS232 接口

4.3.1 数据结构

RS232 通信传输层：



注意：

1. ByteCount 是 Packet 中包括 CRC-16 在内的所有字节的总和。
2. CRC-16 计算从 ByteCount 开始到 Packet 帧的最后一个字节（见后面的 C 代码）
3. ByteCount 和 CRC-16 都是以大端模式发送的。

4.3.2 参数格式

浮点

浮点参数的表示为 IEEE 标准格式，ANSI/IEEE Std 754-1985

64-Bit(双精度浮点)

下面显示的是以大端模式表示的 64-Bit 浮点，小端模式中 4 字节的顺序和大端模式相反。(例如：大端模式：ABCDEFGH 小端模式：DCBAHGFE)

值 v 被定义为(0<Exponent<2047)：$v= (-1)^S \times 2^{(Exponent-1023)} \times 1.Mantissa$

32-Bit(单精度浮点)

下面显示的是以大端模式表示的 32-Bit 浮点，小端模式中 4 字节的顺序和大端模式相反。

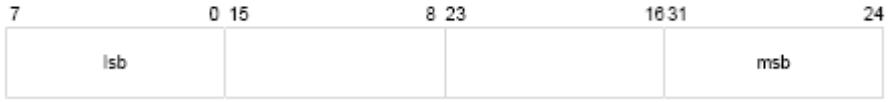
值 v 被定义为(0<Exponent<255)：$v= (-1)^S \times 2^{(Exponent-127)} \times 1.Mantissa$

注意：请参考 ANSI/IEEE Std 754-1985 以获取更多的信息。建议您同时参考您使用的编译器的浮点格式。

有符号 32-Bit 整数(SInt32)

SInt32 格式基于 32bit 数字(2 进制补码)。Bit 31 代表值的符号(0=负，1=正)

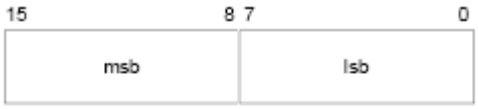
Big Endian



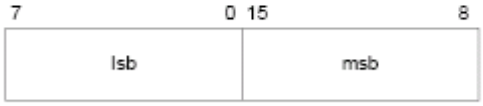
Little Endian

有符号 16-Bit 整数(SInt16)

SInt16 格式基于 16bit 数字(2 进制补码)。Bit 15 代表值的符号(0=负，1=正)



Big Endian



Little Endian

有符号 8-Bit 整数(SInt8)

SInt8 格式基于 8bit 数字(2 进制补码)。Bit 7 代表值的符号(0=负，1=正)

无符号 32-Bit 整数(UInt32)

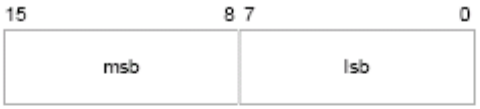
UInt32 格式基于 32bit 数字。

Big Endian

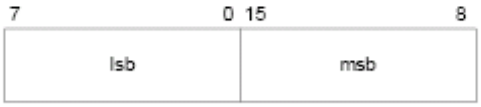
Little Endian

无符号 16-Bit 整数(UInt16)

UInt16 格式基于 16bit 数字。



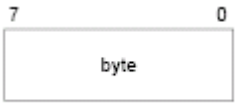
Big Endian



Little Endian

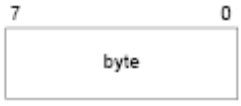
无符号 8-Bit 整数(UInt8)

UInt8 格式基于 8bit 数字。



布尔型

布尔型的数值其一个字节只能为 0（假）或者 1（真）



4.3.3 指令和通信帧

Frame ID	命令	描述
1	kGetModInfo	查询模组的类型及固件版本.
2	kModInfoResp	响应kGetModInfo
3	kSetDataComponents	设置输出的数据组成部分
4	kGetData	获取模组数据
5	kDataResp	响应kGetData
6	kSetConfig	模组内部参数设置
7	kGetConfig	查询当前模组内部参数设置值
8	kConfigResp	响应kGetConfig
9	kSave	保存内部参数和用户标定数据命令
10	kStartCal	用户标定开始命令
11	kStopCal	用户标定结束命令
12	kSetParam	设置指南针和加速度计的FIR滤波参数
13	kGetParam	查询指南针和加速度计的FIR滤波参数
14	kParamResp	包含指南针和加速度计的FIR滤波参数，响应kGetParam
15	kPowerDown	进入低功耗模式
16	kSaveDone	响应 kSave
17	kUserCalSampCount	完成一个采样点采样后由模组发送
18	kUserCalScore	标定得分
19	kSetConfigDone	响应kSetConfig
20	kSetParamDone	响应kSetParam
21	kStartIntervalMode	模组输出数据时间间隔指令
22	kStopIntervalMode	模组停止输出时间间隔指令
23	kPowerUp	由低功耗模式被唤醒后由模组发送
24	kSetAcqParams	设置传感器采集参数
25	kGetAcqParams	查询传感器采集参数
26	kAcqParamsDone	响应kSetAcqParams
27	kAcqParamsResp	响应kGetAcqParams
28	kPowerDownDone	响应 kPowerDown
29	kFactoryUserCal	清楚用户标定系数
30	kFactorUserCalDone	响应kFactoryUserCal
31	kTakeUserCalSample	在标定过程中采集一个标定点数据

KgetModInfo(frame ID1)

该帧命令用于查询模组的类型及固件版本号。该命令没有 payload 部分，完整的 kGetModInfo 命令包如下：

0005	01	EFD4

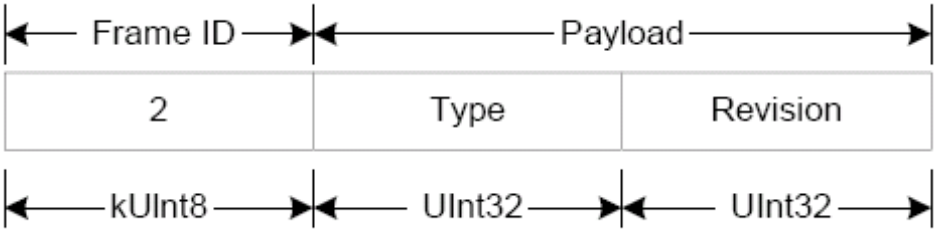
00 05　字节计数

01　kGetModInfo 命令

EFD4　CRC-16 校验码

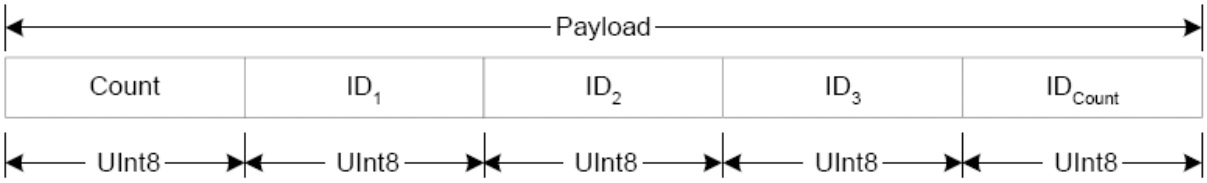
kModInfoResp(frame ID2)

该帧命令用于响应 kGetModInfo 帧。Payload 部分包括固件的版本号以及模组的标识符。



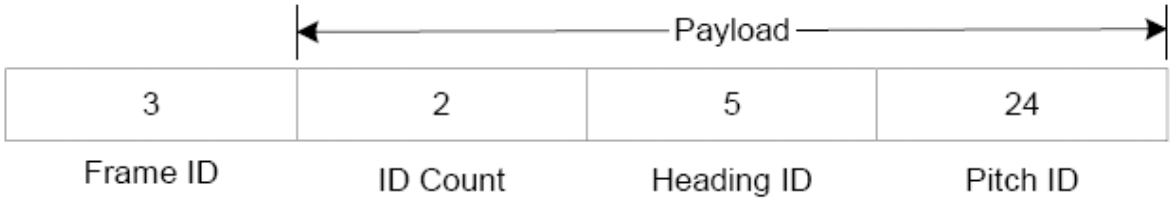
kSetDataComponents(frame ID3)

该帧命令用于设置模组输出数据的组成部分，这个命令不是用于查询模组的数据(见 kGetData)。Payload 部分的第一个字节表示输出的数据由多少个不同的 ID 组成。



例子：

查询方位和俯仰角，payload 部分应该包括：



当查询数据时(kGetData frame)，数据组成部分输出的次序将遵循上面 Payload 部分 ID 的次序输出。

组成部分标识符

组成部分	标识符 (decimal)	格式	单位	范围
kHeading	5	Float32	degrees	0.0˚ to 359.9˚
kTemperature	7	Float32	˚ Celsius	-40˚ to 85˚
kDistortion	8	Boolean	True or False	False (Default) = no distortion
kCalStatus	9	Boolean	True or False	False (Default) = not calibrated
kPCalibrated	21	Float32	G	-1.0 to 1.0
kRCalibrated	22	Float32	G	-1.0 to 1.0
kIZCalibrated	23	Float32	G	-1.0 to 1.0
kPAngle	24	Float32	degrees	-90.0˚ to 90.0˚
kRAngle	25	Float32	degrees	-180.0˚ to 180.0˚
KXAligned	27	Float32	μT	
KYAligned	28	Float32	μT	
KZAligned	29	Float32	μT	

kSetDataComponents&kDataResp 命令组成部分的类型

kHeading 指南针方位输出

kTemperature 从内部温度传感器进行采样，输出值的单位是摄氏度，精度±3℃。

kDistortion 只读标志位，表示至少有一个轴的磁场强度已经超过了±80uT。

kCalStatus 只读标志位，表示用户标定状态。False(默认)= 没有标定。

kPCalibrated,kRCalibrated&kIZCalibrated 工厂标定地球加速度矢量输出。

kPAngle,kRAngle纵横摇(俯仰和翻滚)角度输出，俯仰角等于-90°到 90°，翻滚角等于-180°到 180°。

kXAligned,kYAligned,kZAligned 用户标定地球磁场强度矢量输出。

kGetData (frame ID4)

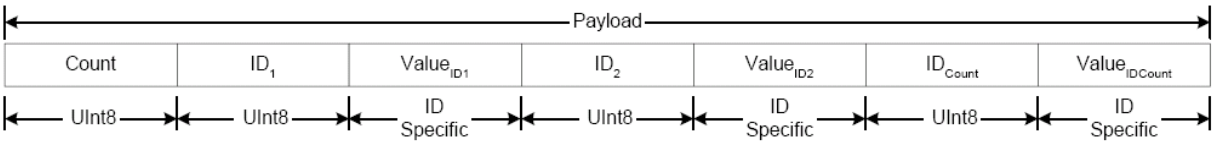
该命名从模组中读取数据，命令中无 payload。完整的 kGetData 命令应包括：

00 05	04	BF71

00 05　字节计数
04　kGetData 命令
BF71　CRC-16 校验码

kDataResp(frame ID5)

该命令帧是对 kGetData 命令帧的响应。Payload 的第一个字节表示了该命令帧由多少个 ID 和数据对组成，这些 ID 的次序由 kSetDataComponents 帧设置。



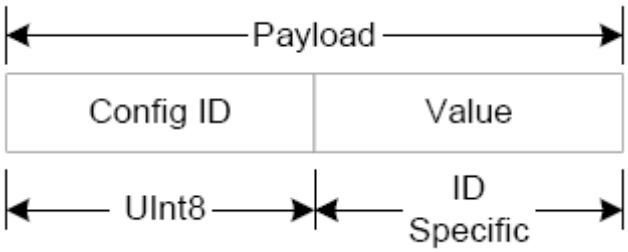
例子：

如果响应帧包括方位和俯仰角的输出，则其格式应为

2	5	359.9	24	10.5
ID Count	Heading ID	Heading Output (Float32)	Pitch ID	Pitch Output (Float32)

kSetConfig(frame ID6)

该命令帧对模组的内部参数进行配置。Payload 的第一个字节为配置的 ID 号，后面为一个具体的值，一次只能配置一个值。



例子：

配置磁偏角值，payload 将会是以下的格式：

1	10.0
Declination ID	Declination Angle (Float32)

配置标识符

设置	配置ID	格式	单位/范围	默认值
kDeclination	1	Float32	-180˚ to 180˚	0˚
kTrueNorth	2	Boolean	True or False	False
kBigEndian	6	Boolean	True or False	True
kMountingRef	10	UInt8	1 = Standard 2 = X axis up 3 = Y axis up 4 = -90° heading offset 5 = -180° heading offset 6 = -270° heading offset 7 = Z down 8 = X + 90° 9 = X + 180° 10 = X + 270° 11 = Y + 90° 12 = Y + 180° 13 = Y + 270° 14 = Z down + 90°	1
kUserCalStableCheck	11	Boolean	True or False	True
kUserCalNumPoints	12	UInt32	12 – 50	50
kUserCalAutoSampling	13	Boolean	True or False	True
kBaudRate	14	UInt8	0 – 300 1 – 600 2 – 1200 3 – 1800 4 – 2400 5 – 3600 6 – 4800 7 – 7200 8 – 9600 9 – 14400 10 – 19200 11 – 28800	12

kDeclination 设置磁偏角，确定真北方位。正磁偏角向东的磁偏角，负磁偏角是向西的磁偏角。TrueNorth 为真时，设置生效。

kTrueNorth 指南针输出为真北方位的标志，将磁偏角叠加到磁北方位上。

kBigEndian　　设置数据模式的标志位。

kMountingRef　　设置模组的参考方位

Standard: 模组安装时主板将水平安装(Z 轴磁传感器垂直)
X Sensor Up：当选择该项时，主板上的传感单元被垂直安装(X 轴传感器垂直)。
Y Sensor Up：当选择该项时，主板上的传感单元被垂直安装(Y 轴传感器垂直)。
Standard 90 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方顺时针旋转 90 度的方向。
Standard 180 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方旋转 180 度的方向。
Standard 270 Degrees：选择该项时，主板上的传感单元被水平安装，但是丝印箭头指向主机系统正前方顺时针旋转 270 度的方向。

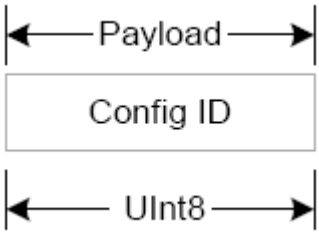
kUserCalStableCheck 在用户标定过程中使用的标志，如果被设置成为 FALSE，在标定时模组的任意一个轴的磁场变化超过 23uT 的时候，模组将进行采样。如果设置成为 TRUE，磁场每个方向上都有 30uT 的稳定值，且比前一个采样点磁场变化超过 5uT，加速度矢量变化在 2mg 之内，模组将进行标定采样。

kUserCalNumPoints　用户标定过程中的最大采样点数。

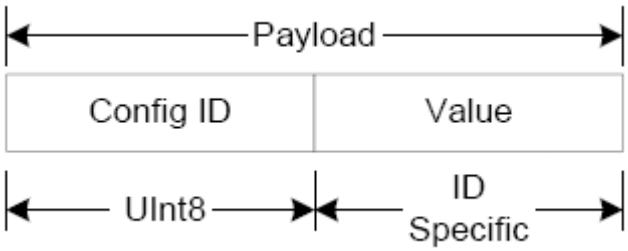
kUserCalAutoSampling　在用户标定过程中使用的标志，如果被设置成为 TRUE，模组将连续采样直到达到设置的采样点数。如果被设置成为 FLASE，模组将等待 kTakeUserCalSample 命令，在当前磁场矢量相对于前一个采样点有 5uT 的变化时进行采样。

kBaudRate　波特率值。更换波特率设置需要断电后，重新上电才能生效。

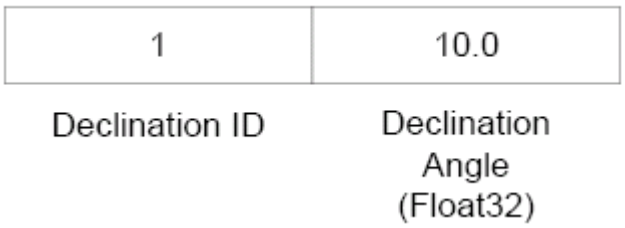
kGetConfig(frame ID7)
该命令帧用户获得模组内部参数的配置值。Payload 包括需要获得配置信息对应的 ID 号



kConfigResp(frame ID8)
该命令帧是对 kGetConfig 帧的响应。Payload 包括配置信息对应的 ID 以及对应的参数。



例子：
如果需要获得设置的磁偏角值，payload 将会是

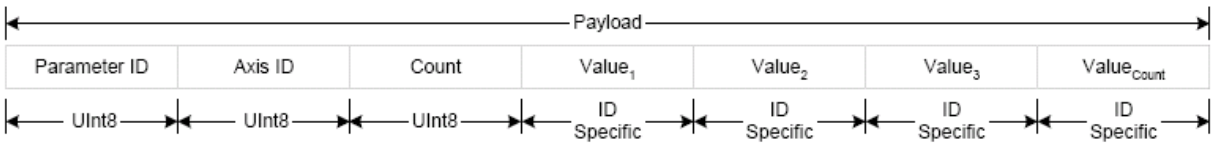


kSave(frame ID9)
该命令帧将内部配置的参数和用户标定的结果保存在内存中，内部配置参数和用户的标定结果都是在上电的时候保存。该命令帧没有 payload。这是唯一一个使模组保存信息的到内存中的命令。

kStartCal(frame ID10)
该命令帧使模组开始用户标定，标定过程中使用当前磁传感器的参数，内部配置信息及 FIR 滤波设置。

kStopCal(frame ID11)
该命令帧使模组停止标定点采样，并计算标定的得分和系数。

kSetParam(frame ID12)
该命令帧用户进行指南针和加速度计的 FIR 滤波设置。Payload 的第二个字节表示磁力计或者加速度计 x 方向的矢量，这是为了区别是否将滤波设置应用到磁力计或者加速度计上。Payload 的第三个字节表示 FIR 滤波器的传递函数的阶数，每个阶数由 Float64 来表示。最大可以设置数了 32，最小为 0。



参数标识符

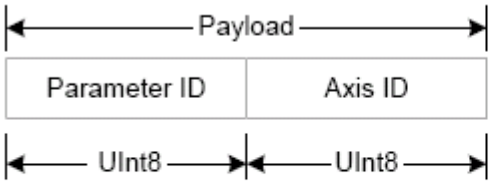
Settings	Parameter ID	Format
KFIRConfig*	3	AxisID (UInt8) + Count (UInt8) + Value (Float64) + Value (Float64) + …

推荐 FIR 滤波传递函数阶数值

2 Count	4 Tap Filter	8 Tap Filter	16 Tap Filter	32 Tap Filter
1	04.6708657655334e-2	01.9875512449729e-2	07.9724971069144e-3	01.4823725958818e-3
2	04.5329134234467e-1	06.4500864832660e-2	01.2710056429342e-2	02.0737124095482e-3
3	04.5329134234467e-1	01.6637325898141e-1	02.5971390034516e-2	03.2757326624196e-3
4	04.6708657655334e-2	02.4925036373620e-1	04.6451949792704e-2	05.3097803863757e-3
5		02.4925036373620e-1	07.1024151197772e-2	08.3414139286254e-3
6		01.6637325898141e-1	09.5354386848804e-2	01.2456836057785e-2
7		06.4500864832660e-2	01.1484431942626e-1	01.7646051430536e-2
8		01.9875512449729e-2	01.2567124916369e-1	02.3794805168613e-2
9			01.2567124916369e-1	03.0686505921968e-2
10			01.1484431942626e-1	03.8014333463472e-2
11			09.5354386848804e-2	04.5402682509802e-2
12			07.1024151197772e-2	05.2436112653103e-2
13			04.6451949792704e-2	05.8693165018301e-2
14			02.5971390034516e-2	06.3781858267530e-2
15			01.2710056429342e-2	06.7373451424187e-2
16			07.9724971069144e-3	06.9231186101853e-2
17				06.9231186101853e-2
18				06.7373451424187e-2
19				06.3781858267530e-2
20				05.8693165018301e-2
21				05.2436112653103e-2
22				04.5402682509802e-2
23				03.8014333463472e-2
24				03.0686505921968e-2
25				02.3794805168613e-2
26				01.7646051430536e-2

kGetParam(frame ID13)

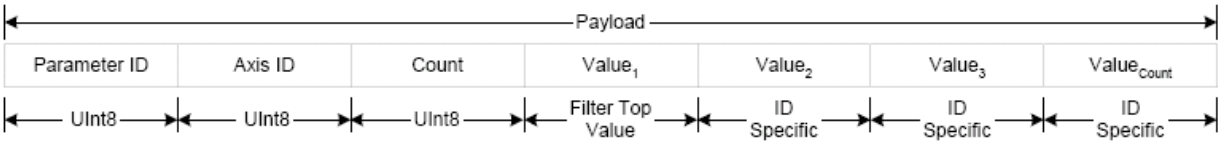
该命令帧用于查询磁力计和加速度计传感器的 FIR 滤波设置。Payload 的第一个字节为 kFIRConfig ID，随后字节为轴矢量的 ID。



Axis IDs
kXAxis = 1
kYAxis = 2
kZAxis = 3
kPAxis = 4
kRAxis = 5
KIZAxis = 6

kParamResp(frame ID14)

该帧包含了当前磁力计或加速度计传感器的 FIR 滤波设置。Payload 的第二个字节为轴矢量 ID，第三个字节为表示有多少滤波器阶数的数量，后面的字节对应每阶的滤波系数，都为 Float64。

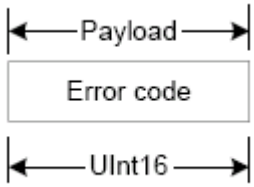


kPowerDown(frame ID15)

该命令帧用于使模组完全进入低功耗模式。该帧没有 payload，模组的外围设备将完全断电，包括 RS232 的驱动器部分，但是保持 Rx 能够接受命令。任何发送给模组的字符都能使模组退出低功耗模式，推荐发送 0xFFh。

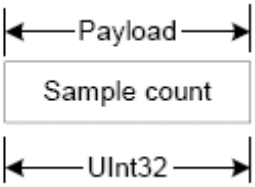
kSaveDone(frame ID16)

该帧用于响应 kSave 帧。Payload 包括一个 UInt16 错误代码，0000h 表示没有错误，0001h 表示在存储数据到内存中时产生了错误。



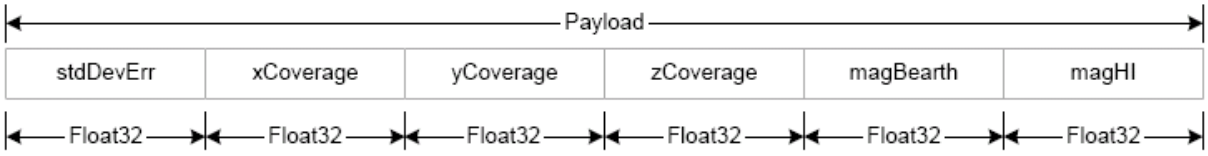
kUserCalSampCount(frame ID17)

该命令帧是模组采集了一个标定点后发送的。Payload 包括了采样计数值，范围为 1 到 50。



kUserCalScore(frame ID18)

该帧的 payload 包括了标定的得分，该得分以一系列的 Float32 值来表示：stdDevErr，xCoverage，yCoverage，zCoverage，magBearth，magHI。



StdDevErr：指南针采样点的磁场强度标准差错误。
XCoverage：X 轴磁力计被采样覆盖的百分比。
YCoverage：Y 轴磁力计被采样覆盖的百分比。
ZCoverage：Z 轴磁力计被采样覆盖的百分比。
MagBearth：从标定的采样点钟计算出的地球磁场强度。
MagHI：反向值，一直为 0。

kSetConfigDone(frame ID19)

该帧用于响应 kSetConfig 帧，该帧没有 payload 部分。

kSetParamDone(frame ID20)

该帧用于响应 kSetParam 帧，该帧没有 payload 部分。

kStartIntervalMode(frame ID21)

该命令帧用于设置模组按固定的时间间隔输出数据(见 kSetAcqParams)。该帧没有 payload 部分。

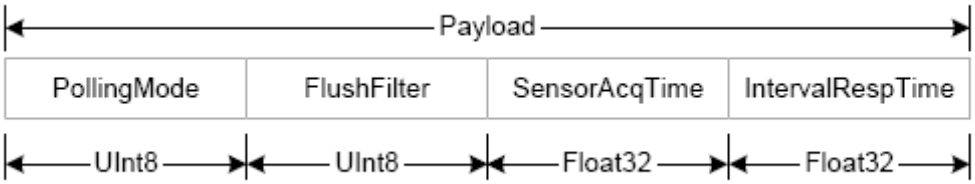
kStopIntervalMode(frame ID22)

该命令帧用于设置在一定的时间间隔后停止数据输出。该帧没有 payload 部分。

kPowerUp(frame ID23)

该帧是模组被从低功耗模式唤醒的时候发送的。该帧没有 payload，因为模组事先处于低功耗模式下，RS232 驱动的 TX 线处于低电平状态，所以建议忽略接收到的第一个字节。

kSetAcqParams(frame ID24)
该命令帧用于设置模组中的传感器采样参数。Payload 应包含：



PollingMode：设置问答/循环数据输出方式的标志位。默认为 TRUE(循环模式)

FlushFilter：设置 FIR filter 对每个采样点进行 flushing 的标志位，默认为 FLASE（没有 flushing）

SensorAcqTime：传感器采样的内部时间间隔。默认时间为 0.0 秒，这意味着模组在完成上一次采样后马上进行下一次的采样。

IntervalRespTime：在问答模式下模组输出数据的时间间隔。默认为 0.0 秒，这意味着模组在完成采样周期后立即发送数据。

kGetAcqParams(frame ID25)
该帧用于查询模组的采样参数。该帧没有 payload 部分。

kAcqParamsDone(frame ID26)
该帧用于响应 kSetAcqParams 帧，该帧没有 payload 部分。

kAcqParamsResp(frame ID27)
该命令帧用于响应 kGetAcqParams 帧。Payload 部分应该包含与 kSetAcqParams 帧相同的 payload。

kPowerDownDone(frame ID28)
该帧用于响应 kPowerDown 帧。表示模组已经成功接收到 kPowerDown 帧，正在进入低功耗模式。该帧没有 payload 部分。

kFactroyUserCal(frame ID29)
该帧用于清除用户标定的系数。该帧没有 payload 部分，该命令帧应在 kSave 帧之后，用于清除内存中的标定数据。

kFactroyUserCalDone(frame ID30)
该帧用于响应 kFactoryUserCal 帧。该帧没有 payload 部分。

kTakeUserCalSample(frame ID31)
该命令帧用于使模组在用户标定过程中采集标定点数据。该帧没有 payload 部分。

4.4 示例代码

4.4.1 Binary TCM High Performance Protocol C Header File & CRC-16 Function

```
// type declarations
typedef struct
{
    UInt8 pollingMode, flushFilter;
    Float32 sensorAcqTime, intervalRespTime;
}    attribute    ((packed)) AcqParams;

typedef struct
{
    Float32 stdDevErr;
    Float32 xCoverage;
    Float32 yCoverage;
    Float32 zCoverage;
    Float32 magBearth;
    Float32 reserve1;
} __attribute    ((packed)) CalScore;

enum
{

     kGetData,            // 4
     kDataResp,           // 5
     kSetConfig,          // 6
     kGetConfig,          // 7
     kConfigResp,         // 8
     kSave,               // 9
     kStartCal,           // 10
     kStopCal,            // 11
     kSetParam,           // 12
     kGetParam,           // 13
     kParamResp,          // 14
     kPowerDown,          // 15
     kSaveDone,           // 16
     kUserCalSampCount,   // 17
     kUserCalScore,       // 18
     kSetConfigDone,      // 19
     kSetParamDone,       // 20

    // Frame IDs (Commands)
    kGetModInfo = 1,     // 1
    kModInfoResp,// 2
    kSetDataComponents,     // 3
```

```
kStartIntervalMode,          //   21
kStopIntervalMode,           //   22
kPowerUp,                    //   23
kSetAcqParams,               //   24
kGetAcqParams,               //   25
kAcqParamsDone,              //   26
kAcqParamsResp,              //   27
kPowerDoneDown,              //   28
kFactoryUserCal,             //   29
kFactoryUserCalDone,         //   30
kTakeUserCalSample,          //   31

// Param IDs
kFIRConfig = 1,              //   3-AxisID(UInt8)+Count(UInt8)+Value(Float64)+...


 // Data Component IDs

 kHeading = 5,                      // 5 - type Float32
 kTemperature = 7,                  // 7 - type Float32
 kDistortion = 8,                 // 8 - type Boolean
 kPCalibrated = 21,                // 21 - type Float32
 kRCalibrated,                     // 22 - type Float32
 kIZCalibrated,                   // 23 - type Float32
 kPAngle,                           // 24 - type Float32
 kRAngle,                          // 25 - type Float32
 kXAligned = 27,                  // 27 - type Float32
 kYAligned,                        // 28 - type Float32
 kZAligned,                       // 29 - type Float32

 // Configuration Parameter IDs
 kDeclination = 1,              // 1 - type Float32
 kTrueNorth,                       // 2 - type Boolean
 kMountingRef = 10,           // 10 - type UInt8
 kUserCalStableCheck,      // 11 - type boolean
 kUserCalNumPoints,             // 12 - type UInt32
 kUserCalAutoSampling,     // 13 – type boolean
 kBaudRate,                       // 14 – UInt8

 // Mounting Reference IDs

  kMountedStandard = 1, // 1
  kMountedXUp,          // 2
  kMountedYUp,          // 3
  kMountedStdPlus90,    // 4
  kMountedStdPlus180,   // 5
  kMountedStdPlus270,   // 6
```

```
    // Result IDs
    kErrNone = 0,                   // 0
    kErrSave,                       // 1
};



// function to calculate CRC-16
UInt16 CRC(void * data, UInt32 len)
{
    UInt8 * dataPtr = (UInt8 *)data;
    UInt32 index = 0;
    // Update the CRC for transmitted and received data using
    // the CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
    UInt16 crc = 0;
    while(len--)
    {
        crc = (unsigned char)(crc >> 8) | (crc << 8);
        crc ^= dataPtr[index++];
        crc ^= (unsigned char)(crc & 0xff) >> 4;
        crc ^= (crc << 8) << 4;
        crc ^= ((crc & 0xff) << 4) << 1;
    }
    return crc;
```

4.4.2 Binary TCM Protocol C++ Communication Examples

The following 4 example files, CommProtocol.h, CommProtocol.cp, TCM5.h and TCM5.cp would be used together for proper communication with a TCM3, TCM5 or TCM5L module.

***NOTE**: The follwing files are not included in the samples code: SystemSenPort.h; Processes.h, TickGenerator.h.*

4.4.2.1 CommProtocol.h File:

```
#pragma once

#include "SystemSerPort.h"
#include "Processes.h"

//
//      This file contains objects used to handle the serial communication with the unit. Unfortunately, these files are not available as the program was written on a non-PC computer. The comments in the code should explain what is expected to be sent or received from these functions so that you can write this section for your specific platform. For example, with the TickGenerator.h, you would need to write a routing that generates 10msec ticks.
//



//
//  CommHandler is a base class that provides a callback for incoming messages.
//
class CommHandler
{
        public:
                // Call back to be implemented in derived class.
                virtual void HandleComm(UInt8 frameType, void * dataPtr = NULL, UInt16 dataLen = 0)
{}
};

//
//  CommProtocol handles the actual serial communication with the unit.
//   Process is a base class that provides CommProtocol with cooperative parallel processing.
  The Control method will be
//  called by a process manager on a continuous basis.
//
class CommProtocol : public Process
{
        public:
                enum
                {
                        // Frame IDs (Commands)
                        kGetModInfo = 1,     // 1
                        kModInfoResp,          // 2
                        kSetDataComponents,  // 3
                        kGetData,                      // 4
```

```
            kDataResp,          // 5

            // Data Component IDs

            kHeading = 5,               // 5 - type Float32
            kTemperature = 7,           // 7 - type Float32
            kPCalibrated = 21,          // 21 - type Float32
            kRCalibrated,               // 22 - type Float32
            kIZCalibrated,              // 23 - type Float32
            kPAngle,                    // 24 - type Float32
            kRAngle,                    // 25 - type Float32
        }

        enum
        {
            kBufferSize = 512,          // maximum size of our input buffer
            kPacketMinSize = 5          // minimum size of a serial packet
        }

        // SerPort is a serial communication object abstracting the hardware implementation
        CommProtocol(CommHandler * handler = NULL, SerPort * serPort = NULL);

        void Init(UInt32 baud = 38400);

        void SendData(UInt8 frame, void * dataPtr = NULL, UInt32 len = 0);
        void SetBaud(UInt32 baud);

    protected:
        CommHandler * mHandler;
        SerPort * mSerialPort;

        UInt8 mOutData[kBufferSize], mInData[kBufferSize]; UInt16
        mExpectedLen;
        UInt32 mOutLen, mOldInLen, mTime, mStep;

        UInt16 CRC(void * data, UInt32 len);
        void Control();
};
```

4.4.2.2 CommProtocol.cp File:

```
#include "CommProtocol.h"

// import an object that will provide a 10mSec tick count through a function called Ticks()
#include "TickGenerator.h"


//
//   SerPort is an object that controls the physical serial interface.   It handles sending out
//   the characters, and buffers the characters read in until we are ready for them.
//
CommProtocol::CommProtocol(CommHandler* handler, SerPort * serPort)
  : Process("CommProtocol")
{
        mHandler = handler;                                     // store the object that will parse the data
when it is fully received mSerialPort =
        serPort; Init();
}

//
//   Initialize the serial port and variables that will control this process
//
void CommProtocol::Init(UInt32 baud)
{
        SetBaud(baud);
        mOldInLen = 0;                  // no data previously received

        mStep = 1; // goto the first step of our process
}

//
//   Put together the frame to send to the unit
//
void CommProtocol::SendData(UInt8 frameType, void * dataPtr, UInt32 len)
{
        UInt8 * data = (UInt8 *)dataPtr;                        // the data to send
        UInt32 index = 0;                                       // our location in the frame we are putting together
        UInt16 crc;                                             // the CRC to add to the end of the packet
        UInt16 count;                                       // the total length the packet will be

        count = (UInt16)len + kPacketMinSize;

        // exit without sending if there is too much data to fit inside our packet if(len > kBufferSize -
        kPacketMinSize) return;

        // Store the total len of the packet including the len bytes (2), the frame ID (1),
        // the data (len), and the crc (2).   If no data is sent, the min len is 5 mOutData[index++] = count >> 8;
        mOutData[index++] = count & 0xFF;

        // store the frame ID
        mOutData[index++] = frameType ;

        // copy the data to be sent
```

```
        while(len--) mOutData[index++] = *data++;

        // compute and add the crc crc =
        CRC(mOutData, index);
        mOutData[index++] = crc >> 8 ;
        mOutData[index++] = crc & 0xFF ;

        // Write block will copy and send the data out the serial port
        mSerialPort->WriteBlock(mOutData,index);
}



//
//   Call the functions in serial port necessary to change the baud rate
//
void CommProtocol::SetBaud(UInt32 baud)
{
        mSerialPort->SetBaudRate(baud);
        mSerialPort->InClear();     // clear    any data that was  already  waiting  in the buffer
}

//
// Update the CRC for transmitted and received data using the CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
//
UInt16 CommProtocol::CRC(void * data, UInt32 len)
{
        UInt8 * dataPtr = (UInt8 *)data; UInt32 index =
        0;

        UInt16 crc = 0;
        while(len--)
        {
                crc = (unsigned char)(crc >> 8) | (crc << 8);
                crc ^= dataPtr[index++];
                crc ^= (unsigned char)(crc & 0xff) >> 4;
                crc ^= (crc << 8) << 4;
                crc ^= ((crc & 0xff) << 4) << 1;
        }
        return crc;
}

//
//   This is called each time this process gets a turn to execute.
//
void CommProtocol::Control()
{
        // InLen returns the number of bytes in the input buffer of the serial object that are available
        // for us to read.
        UInt32 inLen = mSerialPort->InLen();

        switch(mStep)
        {
```

```
            case 1:
            {
                // wait for length bytes to be received by the serial object if(inLen >= 2)
                {
                    // Read block will return the number of requested (or available) bytes
 that are in the
                    // serial objects input buffer.
                    // read the byte count
                    mSerialPort->ReadBlock(mInData, 2);

                    // byte count is ALWAYS transmitted in big endian, copy byte count to
mExpectedLen to
                    // native endianess
                    mExpectedLen = (mInData[0] << 8) | mInData[1];

                    // Ticks is a timer function.   1 tick = 10msec.
                    // wait up to 1/2s for the complete frame (mExpectedLen) to be received
                    mTime = Ticks() + 50 ;
                    mStep++ ;                           // goto the next step in the process
                }
                break ;
            }

            case 2:
            {
                // wait for msg complete or timeout if(inLen >=
                mExpectedLen - 2)
                {
                    UInt16 crc, crcReceived;            // calculated and received crcs.

                    // Read block will return the number of requested (or available) bytes
that are in the
                    // serial objects input buffer.
                    mSerialPort->ReadBlock(&mInData[2],mExpectedLen - 2);
                    // in CRC verification, don't include the CRC in the recalculation (-2)
                    crc = CRC(mInData, mExpectedLen - 2);
                    // CRC is also ALWAYS transmitted in big endian
                    crcReceived = (mInData[mExpectedLen - 2] << 8) | mInData[mExpectedLen - 1] ;

                    if(crc == crcReceived)
                    {
                        // the crc is correct, so pass the frame up for processing.
                        if(mHandler)        mHandler->HandleComm(mInData[2],        &mInData[3],
mExpectedLen - kPacketMinSize);
                    }
                    else
                    {
                        // crc's don't match so clear everything that is currently in the
input buffer since
                        // the data is not reliable.
                        mSerialPort->InClear();
                    }

                    // go back to looking for the length bytes.
```

```
                        mStep = 1 ;
                }
                else
                {
                        // Ticks is a timer function.   1 tick = 10msec.
                        if(Ticks() > mTime)
                        {
                                // Corrupted message. We did not get the length we were expecting within 1/2sec of receiving
                                 // the length bytes. Clear everything in the input buffer since the
data is unreliable
                                 mSerialPort->InClear();
                                 mStep = 1 ;       // Look for the next length bytes
                        }
                }
                break ;
        }

        default:
                break ;
        }
}
```

4.4.2.3 TCM5.h File (For TCM3, TCM5 and TCM5L Modules):

```
#pragma once

#include "Processes.h"
#include "CommProtocol.h"


//
// This file contains the object providing communication to the TCM3/TCM5.  It will set up the
unit and parse packets received
//  Process is a base class that provides TCM3/TCM5 with cooperative parallel processing. The
Control method will be
//  called by a process manager on a continuous basis.
//
class TCM3/TCM5 : public Process, public CommHandler
{
     public:
          TCM3/TCM5(SerPort * serPort);
          ~TCM3/TCM5();

     protected:
          CommProtocol * mComm;

          UInt32 mStep, mTime, mResponseTime;

          void HandleComm(UInt8 frameType, void * dataPtr = NULL, UInt16 dataLen = 0);
          void SendComm(UInt8 frameType, void * dataPtr = NULL, UInt16 dataLen = 0);

          void Control();
};
```

4.4.2.4 TCM5.cp File (For TCM3, TCM5 & TCM5L Modules):

```
#include "TCM3/TCM5.h"
#include "TickGenerator.h"

const UInt8 kDataCount = 4;        // We will be requesting 4 componets (Heading, pitch, roll,
temperature)
//
//    This object polls the TCM3/TCM5 unit once a second for heading, pitch, roll and temperature.
//

TCM3/TCM5::TCM3/TCM5(SerPort * serPort)
 : Process("TCM3/TCM5")

{
     // Let the CommProtocol know this object will handle any serial data returned by the unit
     mComm = new CommProtocol(this, serPort);

     mTime = 0;
     mStep = 1;
}

TCM3/TCM5::~TCM3/TCM5()
{
}

//
//  Called by the CommProtocol object when a frame is completely received
//
void TCM3/TCM5::HandleComm(UInt8 frameType, void * dataPtr, UInt16 dataLen)
{
     UInt8 * data = (UInt8 *)dataPtr;

     switch(frameType)
     {
          case CommProtocol::kDataResp:
          {
               // Parse the data response
               UInt8 count = data[0];        // The number of data elements returned
               UInt32 pntr = 1;              // Used to retrieve the returned elements

               // The data elements we requested
               Float32 heading, pitch, roll, temperature;

               if(count != kDataCount)
               {
                      // Message is a function that displays a C formatted string (similar to printf)
                      Message("Received %u data elements instead of the %u requested\r\n",
     (UInt16)count,
                      (UInt16)kDataCount);
                      return;
               }
```

```
          // loop through and collect the elements
          while(count)
          {
               // The elements are received as {type (ie. kHeading), data}
               switch(data[pntr++])   // read the type and go to the first byte of the
data
               {
                    // Only handling the 4 elements we are looking for case
                    CommProtocol::kHeading:
                    {
     // Move(source, destination, size (bytes)).  Move copies the specified number of
    // bytes from the source pointer to the destination pointer.
    // Store the heading.
                         Move(&(data[pntr]), &heading, sizeof(heading));

                         // increase the pointer to point to the next data element type
                         pntr += sizeof(heading);
                         break;
                    }

                    case CommProtocol::kPAngle:
                    {
                        // Move(source, destination, size (bytes)).    Move copies the
specified number of
                        // bytes from the source pointer to the destination pointer.
                        // Store the pitch.
                        Move(&(data[pntr]), &pitch, sizeof(pitch));
                        // increase the pointer to point to the next data element type
                        pntr += sizeof(pitch);
                        break;
                    }

                    case CommProtocol::kRAngle:
                    {
                        // Move(source, destination, size (bytes)).    Move copies the
specified number of
                        // bytes from the source pointer to the destination pointer.
                        // Store the roll.
                        Move(&(data[pntr]), &roll, sizeof(roll));

                        // increase the pointer to point to the next data element type
                        pntr += sizeof(roll);
                        break;
                    }

                    case CommProtocol::kTemperature:
                    {
                        // Move(source, destination, size (bytes)).  Move copies the
specified number of
                        // bytes from the source pointer to the destination pointer.
                         // Store the heading.
                         Move(&(data[pntr]), &temperature, sizeof(temperature));
```

```
                              // increase the pointer to point to the next data element type pntr
                              += sizeof(temperature);
                              break;
                         }

                         default:
                             // Message is a function that displays a formatted string
     (similar to printf)
                              Message("Unknown type: %02X\r\n", data[pntr - 1]);
                              // unknown data type, so size is unknown, so skip everything
                              return;
                              break;
                    }
               }

                    count--;                // One less element to read in
               }

               // Message is a function that displays a formatted string (similar to printf)
               Message("Heading: %f, Pitch: %f, Roll: %f, Temperature: %f\r\n", heading,
     pitch, roll,
               temperature);
               mStep--;  // send next data request
               break;
             }
          default:
          {
               // Message is a function that displays a formatted string (similar to printf)
               Message("Unknown frame %02X received\r\n", (UInt16)frameType);
               break;
          }
     }
}

//
//  Have the CommProtocol build and send the frame to the unit.
//
void TCM3/TCM5::SendComm(UInt8 frameType, void * dataPtr, UInt16 dataLen)
{
     if(mComm) mComm->SendData(frameType, dataPtr, dataLen);
     // Ticks is a timer function.  1 tick = 10msec.
     mResponseTime = Ticks() + 300;          // Expect a response within 3 seconds
}

//
//  This is called each time this process gets a turn to execute.
//
void TCM3/TCM5::Control()
{
     switch(mStep)
     {
          case 1:
          {
```

```
                UInt8 pkt[kDataCount + 1];    // the compents we are requesting, preceded by the
number of...
                                                    // ...components being requested

                pkt[0] = kDataCount;
                pkt[1] = CommProtocol::kHeading; pkt[2]
                = CommProtocol::kPAngle; pkt[3] =
                CommProtocol::kRAngle; pkt[4] =
                CommProtocol::kTemperature;

                SendComm(CommProtocol::kSetDataComponents, pkt, kDataCount + 1);

                // Ticks is a timer function.  1 tick = 10msec.
                mTime = Ticks() + 100;        // Taking a sample in 1s. mStep++;
                                              // go to next step of process
                break;
           }

           case 2:
           {
                // Ticks is a timer function.  1 tick = 10msec.
                if(Ticks() > mTime)
                {
                     // tell the unit to take a sample
                     SendComm(CommProtocol::kGetData);
                     mTime = Ticks() + 100;   // take a sample every second
                     mStep++;
                }
                break;
           }

           case 3:
           {
                // Ticks is a timer function.  1 tick = 10msec.
                if(Ticks() > mResponseTime)
                {
                      Message("No  response from the  unit.    Check    connection    and  try
again\r\n");
                      mStep = 0;
                }
                break;
           }

           default:
                break;
      }
}
```