



BH-MPU6050 六轴传感器 用户手册

修订历史

日期	版本	更新内容
2015/9/21	1.0.0	-





文档说明

本手册旨在帮助用户正确构建 BH-MPU6050 六轴传感器的使用环境，引导用户快速使用该模块。

关于模块的原理图、机械尺寸等说明请参考《MPU6050-黑白原理图》。

关于主控芯片 MPU6050 的硬件参数请参考官方固件库及资料里的文档。



目录

BH-MPU6050 六轴传感器.....	1
用户手册.....	1
文档说明.....	2
目录.....	3
1. MPU6050 简介.....	4
1.1 MPU6050 简介.....	4
1.2 特性参数.....	4
2. 硬件测试.....	5
2.1 硬件连接.....	5
2.2 基本测试.....	6
2.3 使用上位机查看姿态.....	8
3. 配套程序说明.....	10
3.1 基本驱动程序.....	10
3.2 原始数据说明.....	12
3.3 STM32-MPU6050_DMP 测试例程.....	13
3.4 STM32-MPU6050_DMP_python 上位机.....	16
4. 产品更新及售后支持.....	19



1. MPU6050 简介

1.1 MPU6050 简介

BH-MPU6050 是秉火科技推出的六轴传感器模块，它采用 InvenSense 公司的 MPU6050 作为主芯片，能同时检测三轴加速度、三轴陀螺仪(三轴角速度)的运动数据以及温度数据。利用 MPU6050 芯片内部的 DMP 模块（Digital Motion Processor 数字运动处理器），可对传感器数据进行滤波、融合处理，直接通过 IIC 接口向主控器输出姿态解算后的数据，降低主控器的运算量。其姿态解算频率最高可达 200Hz，即 5ms，非常适合用于对姿态控制实时要求较高的领域。常见应用于手机、智能手环、四轴飞行器、计步器等的姿态检测。

1.2 特性参数

参数	说明
供电	3.3V-5V
通讯接口	IIC 协议，支持的 IIC 时钟最高频率为 400KHz
测量维度	加速度：3 维 陀螺仪：3 维
ADC 分辨率	加速度：16 位 陀螺仪：16 位
加速度测量范围	±2g、±4g、±8g、±16g 其中 g 为重力加速度常数， $g=9.8m/s^2$
加速度最高分辨率	16384 LSB/g
加速度测量精度	0.1g
加速度输出频率	最高 1000Hz
陀螺仪测量范围	±250 °/s、±500 °/s、±1000 °/s、±2000 °/s、
陀螺仪最高分辨率	131 LSB/(°/s)
陀螺仪测量精度	0.1 °/s
陀螺仪输出频率	最高 8000Hz
DMP 姿态解算频率	最高 200Hz
温度传感器测量范围	-40~+85℃
温度传感器分辨率	340 LSB/℃
温度传感器精度	±1℃
工作温度	-40~+85℃
功耗	500uA~3.9mA (工作电压 3.3V)



2. 硬件测试

本模块配套 STM32 驱动程序，可直接使用秉火 ISO 及 ISO-Mini 开发板进行测试。
按要求使用杜邦线把模块连接到开发板，并下载程序即可。

2.1 硬件连接

BH-MPU6050 模块外观见图 2-1，模块引出了 6 个引脚，在其背面标有引脚名称及参考方向的丝印。

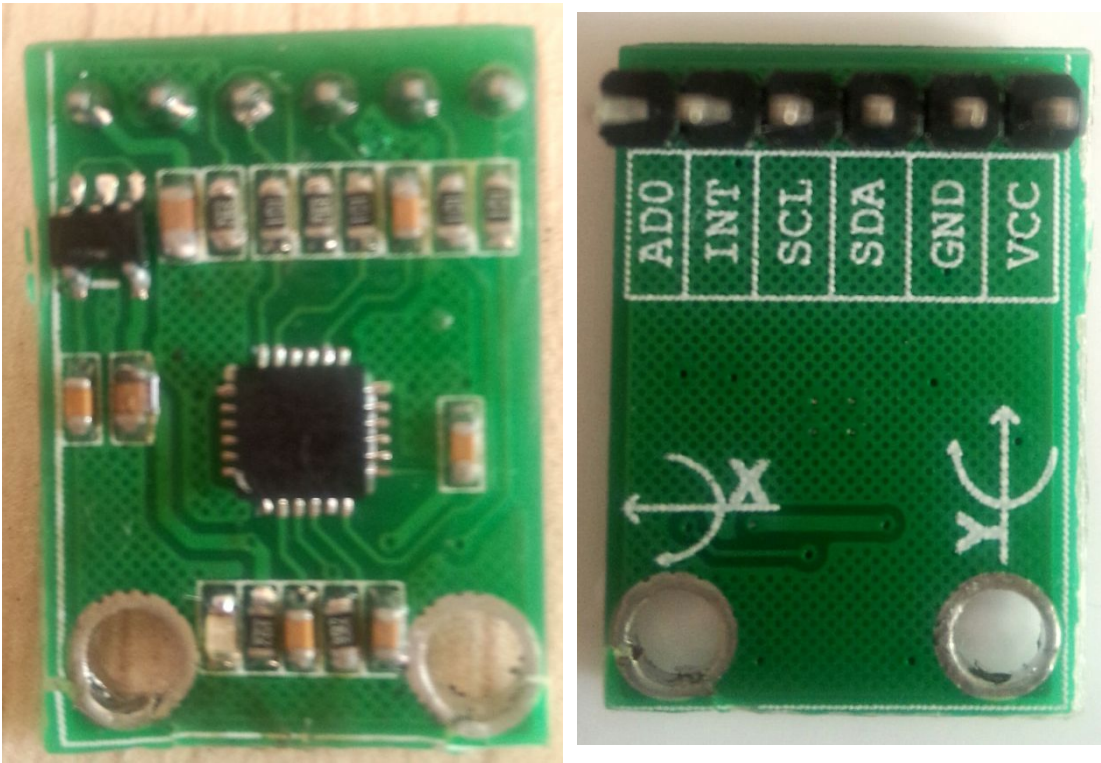


图 2-1 BH-MPU6050 模块外观

表 2-1 MPU6050 模块引脚说明

序号	引脚名称	说明	与 STM32 开发板连接
1	VCC	3.3/5V 电源输入	接 3.3V 或 5V
2	GND	地线	GND
3	SDA	IIC 数据信号线 SDA (模块上已接上拉电阻)	PB7
4	SCL	IIC 时钟信号线 SCL (模块上已接上拉电阻)	PB6
5	INT	中断输出引脚	PB10
6	ADO	从机地址设置引脚 <input type="checkbox"/> 接地或悬空时, 地址为: 0x68 <input type="checkbox"/> 接 VCC 时, 地址为: 0x69	悬空或接地

BH-MPU6050 模块引脚的具体说明见表 2-1，与 ISO 及 ISO-MINI STM32 开发板相连时，请按照“与 STM32 开发板的连接”一栏来连接。连接后图见图 2-2。



图 2-2 MPU6050 模块与 ISO 开发板连接图

2.2 基本测试

连接好模块后，找到配套资料里的例程“1.STM32-MPU6050_DMP 测试例程”，使用 MDK 编译并下载该程序到开发板，复位开发板让程序运行。

程序是 KEIL5 编译的，低于 KEI5 的版本打不开，必须保证开发环境是 KEIL5 以上。

工程所在目录：“开发板配套例程ISO 版本[或 ISO-MINI 版本]”。

1. 正常运行的实验现象

- ❑ 开发板的液晶屏会显示 Pitch、Roll、Yaw 姿态角以及温度，计步数，见图 2-3；
- ❑ 晃动陀螺仪时 Pitch、Roll、Yaw 数据会剧烈变化，这表明模块已经正常工作。
若晃动模块姿态角数据不更新，可能是晃动的过程中连接线松了，这时请重新固定引脚连线，并复位开发板，重新检测。
- ❑ 屏幕上的 Temperature 表示模块检测到的温度数据。(精度不高，而且模块运行久了温度会比气温高，所以有误差时不要怀疑模块不正常)；
- ❑ 屏幕的最后一项是计步数，拿起陀螺仪模仿摆臂运动，计步数会统计出 steps 数据。（计步模式匹配，需要持续多次重复模仿摆臂运动，请耐心等待）；



图 2-3 MPU6050 模块正常运行时的液晶显示

2. 不正常运行时故障排查

- ❑ 液晶屏以蓝色字显示 “No MPU6050 detected!”，见图 2-4。这时说明开发板检测不到 MPU6050 模块，请参照引脚连接表重新检查模块与开发板之间的连线。
- ❑ 若屏幕有显示姿态角等数据，但晃动模块姿态角数据不更新时，可能是晃动的过程中连接线松了，这时请重新固定引脚连线，并复位开发板，重新检测。



图 2-4 模块运行不正常时的液晶显示



2.3 使用上位机查看姿态

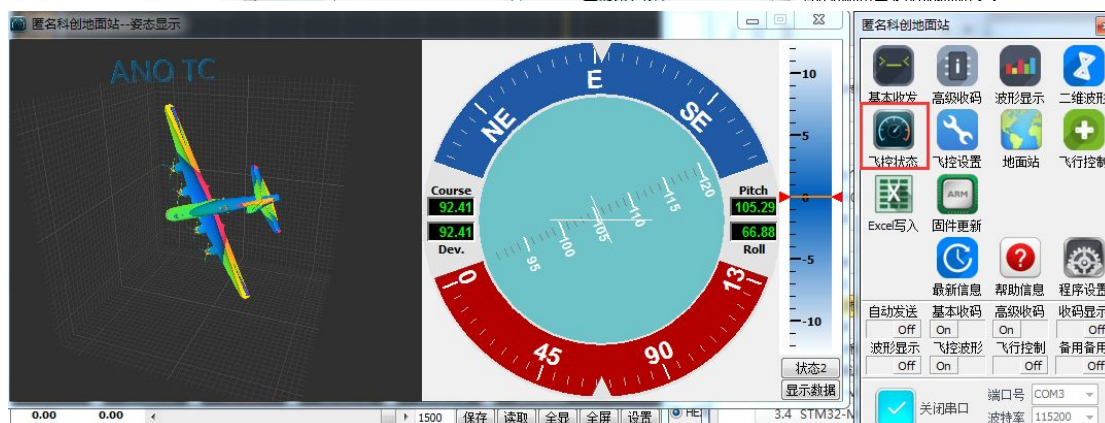
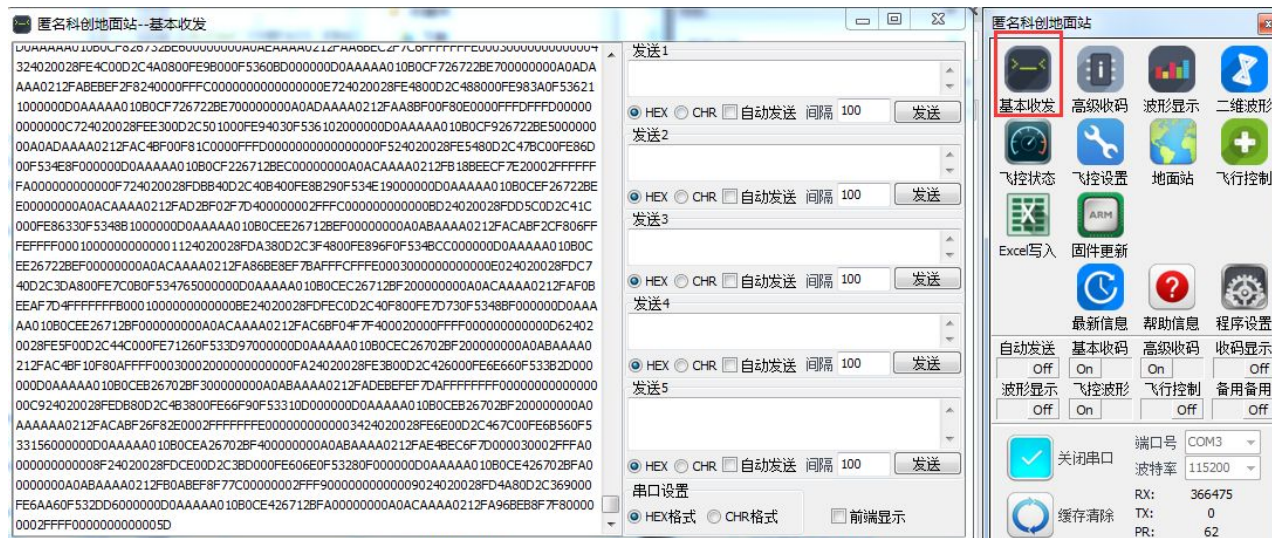
若通过液晶屏的信息了解到 MPU6050 模块已正常工作, 则可进一步在电脑上使用“ANO_TC 匿名飞控地面站-0512.exe”(以下简称“匿名上位机”)软件查看可视化数据。

软件所在目录: “配套软件”。

步骤如下:

- ❑ 确认开发板的 USB TO USART 接口已与电脑相连, 确认电脑端能查看到该串口设备。
- ❑ 打开配套资料里的“匿名上位机”软件, 在软件界面打开 开发板对应的串口 (波特率为 115200), 把“基本收码”、“高级收码”、“飞控波形”功能设置为 on 状态。点击上方图中的基本收发、波形显示、飞控状态图标, 会弹出窗口。具体见下文软件配置图。
- ❑ 在软件的“基本收发”、“波形显示”、“飞控状态”页面可看到滚动数据、随着模块晃动而变化的波形以及模块姿态的 3D 可视化图形。







3. 配套程序说明

BH-MPU6050 模块一共配套了四个例程，用户可根据需求选择相应的程序来学习。

例程所在目录：“开发板配套例程ISO 版本[或 ISO-MINI 版本]”。

程序	说明
1.STM32-MPU6050_DMP 测试例程	移植自官方驱动程序，使用了 DMP 功能，液晶屏会显示基本数据，支持匿名上位机可视数据。默认使用软件 IIC。
2.STM32-MPU6050_DMP_python 上位机	移植自官方驱动程序，使用了 DMP 功能，液晶屏会显示基本数据，可使用官方提供的 python 上位机控制，不支持匿名上位机可视数据。默认使用软件 IIC。
3.软件 STM32-MPU6050	软件 IIC，MPU6050 基本驱动程序，不包含 DMP 功能，没有移植官方驱动程序。本程序通过串口输出简单测量数据，没有驱动液晶显示。(不支持匿名上位机可视数据)。
4.硬件 STM32-MPU6050	硬件 IIC，MPU6050 基本驱动程序，不包含 DMP 功能，没有移植官方驱动程序。本程序通过串口输出简单测量数据，没有驱动液晶显示。(不支持匿名上位机可视数据)。 (使用硬件 IIC 时不能与液晶屏同时使用，因为 FSMC 的 NADV 与 IIC1 的 SDA 是同一个引脚，互相影响了)

3.1 基本驱动程序

学习 MPU6050 的代码时，先学习配套资料里的“3.软件 STM32-MPU6050”和“4.硬件 STM32-MPU6050”这两个工程，是驱动 MPU6050 的基本程序，相对简单。

MPU6050 模块使用 IIC 与 STM32 通讯，不了解 IIC 协议的用户请先学习下《零死角玩转 STM32》的 IIC 驱动 EEPROM 教程。

这里提到的硬件 IIC 和软件 IIC 两种驱动方式主要的区别在于是否使用了 STM32 的片上 IIC 外设，因为很多工程师反映 STM32 的硬件 IIC 存在硬件 bug，所以更喜欢使用软件模拟 IIC。而实际上，硬件 IIC 如果不是跟液晶屏同时使用(FSMC 的 NADV 与 IIC1 的 SDA 是同一个引脚)，是没有问题的。

这两个工程里最核心的就是 MPU6050 初始化及读取原始数据的过程。

在 main 函数初始化了 SysTick、Usart、IIC 等 STM32 外设后，调用了 MPU6050_Init 函数，其定义如下：

```
1 /**
2  * @brief    初始化 MPU6050 芯片
3  * @param
4  * @retval
5  */
6 void MPU6050_Init(void)
```



```
7 {
8     int i=0,j=0;
9     //在初始化之前要延时一段时间,若没有延时,则断电后再上电数据可能会出错
10    for (i=0; i<1000; i++) {
11        for (j=0; j<1000; j++) {
12            ;
13        }
14    }
15    MPU6050_WriteReg(MPU6050_RA_PWR_MGMT_1, 0x00);           //解除休眠状态
16    MPU6050_WriteReg(MPU6050_RA_SMPLRT_DIV, 0x07);          //陀螺仪采样率
17    MPU6050_WriteReg(MPU6050_RA_CONFIG, 0x06);
18    MPU6050_WriteReg(MPU6050_RA_ACCEL_CONFIG, 0x01);         //配置加速度传感器工
                                                                //作在 16G 模式
19    MPU6050_WriteReg(MPU6050_RA_GYRO_CONFIG, 0x18);         //陀螺仪自检及测量范
                                                                //围,典型值: 0x18 (
                                                                //不自检, 2000deg/s)
20
21 }
```

MPU6050_Init 函数调用了 MPU6050_WriteReg 使用 IIC 协议向模块的特定寄存器写入配置,主要配置了模块的电源模式、采样率、加速度计及陀螺仪的量程。关于寄存器的具体说明请阅读官方的 datasheet-配套资料里的《MPU6050 寄存器》。

初始化完成后,main 函数里就定时调用以下函数获取加速度、陀螺仪以及温度数据:

```
1 /**
2  * @brief   读取 MPU6050 的加速度数据
3  * @param
4  * @retval
5  */
6 void MPU6050ReadAcc(short *accData)
7 {
8     u8 buf[6];
9     MPU6050_ReadData(MPU6050_ACC_OUT, buf, 6);
10    accData[0] = (buf[0] << 8) | buf[1];
11    accData[1] = (buf[2] << 8) | buf[3];
12    accData[2] = (buf[4] << 8) | buf[5];
13 }
14
15 /**
16  * @brief   读取 MPU6050 的角加速度数据
17  * @param
18  * @retval
19  */
20 void MPU6050ReadGyro(short *gyroData)
21 {
22     u8 buf[6];
23     MPU6050_ReadData(MPU6050_GYRO_OUT, buf, 6);
24    gyroData[0] = (buf[0] << 8) | buf[1];
25    gyroData[1] = (buf[2] << 8) | buf[3];
26    gyroData[2] = (buf[4] << 8) | buf[5];
27 }
28
29
30 /**
31  * @brief   读取 MPU6050 的温度数据,转化成摄氏度
32  * @param
33  * @retval
34  */
35 void MPU6050_ReturnTemp(short*Temperature)
36 {
37     short temp3;
38     u8 buf[2];
39 }
```




```
40     MPU6050_ReadData(MPU6050_RA_TEMP_OUT_H,buf,2);           //读取温度值
41     temp3=(buf[0]<<8)|buf[1];
42     *Temperature=((double)(temp3+13200))/280-13;
43 }
```

其中返回的加速度和角速度都是 MPU6050 模块 ADC 输出的原始 16 位数据, 没有处理, 温度函数返回的则是以摄氏度为单位的值。

在电脑端使用串口接收开发板上传的数据, 实验现象见图 3-1。晃动模块, 数据会剧烈变化。



图 3-1 陀螺仪基本驱动实验现象

3.2 原始数据说明

加速度和角速度的原始数据如何转化成易于理解的以 m/s^2 和 $^\circ/\text{s}$ 为单位的数据?

转化时需要考虑初始化 MPU6050 时的量程配置:

```
1     MPU6050_WriteReg(MPU6050_RA_ACCEL_CONFIG, 0x00);
2           //配置加速度传感器工作在 2G 模式, 不自检
3     MPU6050_WriteReg(MPU6050_RA_GYRO_CONFIG, 0x18);
4           //陀螺仪自检及测量范围, 典型值: 0x1
5           8 (不自检, 2000deg/s)
```

根据寄存器手册可知配置的陀螺仪的量程是 $\pm 2000^\circ/\text{s}$, 其分辨率是 $16.4\text{LSB}(^\circ/\text{s})$ 。LSB 的意思是最小有效位, ADC 的分辨率一般用这种方式表示, $16.4\text{LSB}(^\circ/\text{s})$ 即说明原始数据的每 16.4 单位的值表示 $1^\circ/\text{s}$ 。若原始寄存器数据的值为 x , 那么转化为 $^\circ/\text{s}$ 即为:

$$x / 16.4 (^\circ/\text{s}).$$

MPU6050 输出的位数为 16 位(2 的 16 次方共 65536 个 LSB)对应满量程, 当量程为



$\pm 2000^{\circ}/s$ 时对应分辨率为 $65535/4000 = 16.4\text{LSB} (^{\circ}/s)$ 。因为 MPU6050 只能 16 位输出，所以测量范围越大，对应分辨率就越低。

FS_SEL selects the full scale range of the gyroscope outputs according to the following table.

FS_SEL	Full Scale Range
0	$\pm 250^{\circ}/s$
1	$\pm 500^{\circ}/s$
2	$\pm 1000^{\circ}/s$
3	$\pm 2000^{\circ}/s$

图 3-2 陀螺仪的几种量程配置

接下来是加速度的分析，同样根据英文寄存器手册可以知工程中设置的加速度量程是 $\pm 2g$ ，分辨率是 $16384\text{LSB}/g$ ，其中的 g 为重力常数 9.8 m/s^2 。若原始寄存器数据的值为 x ，那么转化为 m/s^2 即为：

$$(x / 16384) * g \text{ (m/s}^2\text{)}$$

实验时可以利用地球重力来测试，与重力加速度平行的那一轴，其加速度寄存器读取得的数值大小应约等于 16384。

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 2g$	$16384\text{ LSB}/g$
1	$\pm 4g$	$8192\text{ LSB}/g$
2	$\pm 8g$	$4096\text{ LSB}/g$
3	$\pm 16g$	$2048\text{ LSB}/g$

图 3-3 加速度配置跟量程的关系

实验中没有对这些原始数值进行转化，因为即使得到转化后的数据，也很难应用，在实际中一般配合 MPU6050 模块的 DMP 模块来处理数据，可直接得出姿态角，请参考下一节。

3.3 STM32-MPU6050_DMP 测试例程

“STM32-MPU6050_DMP 测试例程”即第 2 章中使用的 MPU6050 测试程序，实验现象已在第 2 章中说明。本程序移植自 InvenSense 公司提供的驱动，该驱动包含了控制 MPU6050 的所有功能，包括使用 DMP 进行数据融合，姿态解算，熟悉它的框架之后使用非常方便。驱动的源码包在配套资料里的 “motion_driver6.12”，该源码包里的 documentation 文件夹包含了关于驱动的详细说明。

固件库所在目录：“官方固件库及资料\motion_driver_6.12.zip”。

程序的简单说明：

在 main 函数里对 STM32 的基本外设以及 MPU6050 模块初始完成之后，进入了一个 while 循环，在 while 中持续检测由 MPU6050 中断设置的标志 `hal.new_gyro`，若数据有更

新(核心代码见图 3-4)，调用 `dmp_read_fifo` 读取加速度、陀螺仪原始数据，调用 `inv_build_gyro` `inv_build_accel` 生成处理后的加速度、陀螺仪数据，最后调用 `inv_build_quat` 计算出与姿态角相关的四元数(以上数据处理都由 MPU6050 的 DMP 完成)，STM32 获取了四元数之后，经过简单的计算，就可以得到欧拉角 Pitch、Roll、Yaw 数据了。(四元数和欧拉角都是用于表征姿态，四元数方便快速运算，欧拉角方便直接表达，它们可以进行数学转化。)

```

1138     hal.new_gyro = 0;
1139 } else if (hal.new_gyro && hal.dmp_on) {
1140     short gyro[3], accel_short[3], sensors;
1141     unsigned char more;
1142     long accel[3], quat[4], temperature;
1143     /* This function gets new data from the FIFO when the DMP is in
1144      * use. The FIFO can contain any combination of gyro, accel,
1145      * quaternion, and gesture data. The sensors parameter tells the
1146      * caller which data fields were actually populated with new data.
1147      * For example, if sensors == (INV_XYZ_GYRO | INV_WXYZ_QUAT), then
1148      * the FIFO isn't being filled with accel data.
1149      * The driver parses the gesture data to determine if a gesture
1150      * event has occurred; on an event, the application will be notified
1151      * via a callback (assuming that a callback function was properly
1152      * registered). The more parameter is non-zero if there are
1153      * leftover packets in the FIFO.
1154      */
1155     dmp_read_fifo(gyro, accel_short, quat, &sensor_timestamp, &sensors, &more);
1156     if (!more)
1157         hal.new_gyro = 0;
1158     if (sensors & INV_XYZ_GYRO) {
1159         /* Push the new data to the MPL. */
1160         inv_build_gyro(gyro, sensor_timestamp);
1161         new_data = 1;
1162         if (new_temp) {
1163             new_temp = 0;
1164             /* Temperature only used for gyro temp comp. */
1165             mpu_get_temperature(&temperature, &sensor_timestamp);
1166             inv_build_temp(temperature, sensor_timestamp);
1167         }
1168     }
1169     if (sensors & INV_XYZ_ACCEL) {
1170         accel[0] = (long)accel_short[0];
1171         accel[1] = (long)accel_short[1];
1172         accel[2] = (long)accel_short[2];
1173         inv_build_accel(accel, 0, sensor_timestamp);
1174         new_data = 1;
1175     }
1176     if (sensors & INV_WXYZ_QUAT) {
1177         inv_build_quat(quat, 0, sensor_timestamp);
1178         new_data = 1;
1179     }
1180 } else if (hal.new_gyro) {

```

图 3-4 程序中使用 DMP 获取四元数的核心代码

(具体可在源码查看，截图中的行号与源码行号一致，ISO 版本的 main 文件)

经过以上过程获取数据后，程序会进入 `inv_execute_on_data` 和 `read_from_mpl` 中运行。

```

1230
1231
1232
1233     if (new_data) {
1234         inv_execute_on_data();
1235
1236         /* This function reads bias-compensated sensor data and sensor
1237          * fusion outputs from the MPL. The outputs are formatted as seen
1238          * in eMPL_outputs.c. This function only needs to be called at the
1239          * rate requested by the host.
1240          */
1241         read_from_mpl();
1242     }
1243 }
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255 }
1256
1257
1258
1259 /*****END OF FILE*****/

```

图 3-5 向上位机发送数据



其中的 `inv_execute_on_data` 主要用于更新数据状态, `read_from_mpl` 则用于向上位机发送数据。它定义于 `main` 文件中, 见图 3-6。

```
271 static void read_from_mpl(void)
272 {
273     long msg, data[9];
274     int8_t accuracy;
275     unsigned long timestamp;
276     float float_data[3] = {0};
277
278     MPU_DEBUG_FUNC();
279     if (inv_get_sensor_type_quat(data, &accuracy, (inv_time_t*)&timestamp)) {
280         /* Sends a quaternion packet to the PC. Since this is used by the Python
281          * test app to visually represent a 3D quaternion, it's sent each time
282          * the MPL has new data.
283          */
284         eMPL_send_quat(data);
285
286         /* Specific data packets can be sent or suppressed using USB commands. */
287         if (hal.report & PRINT_QUAT)
288             eMPL_send_data(PACKET_DATA_QUAT, data);
289     }
290
291     if (hal.report & PRINT_ACCEL) {
292         if (inv_get_sensor_type_accel(data, &accuracy,
293             (inv_time_t*)&timestamp))
294             eMPL_send_data(PACKET_DATA_ACCEL, data);
295     }
296
297     if (hal.report & PRINT_GYRO) {
298         if (inv_get_sensor_type_gyro(data, &accuracy,
299             (inv_time_t*)&timestamp))
300             eMPL_send_data(PACKET_DATA_GYRO, data);
301     }
302 #ifdef COMPASS_ENABLED
303     if (hal.report & PRINT_COMPASS) {
304         if (inv_get_sensor_type_compass(data, &accuracy,
305             (inv_time_t*)&timestamp))
306             eMPL_send_data(PACKET_DATA_COMPASS, data);
307     }
308 #endif
309     if (hal.report & PRINT_EULER) {
310         if (inv_get_sensor_type_euler(data, &accuracy,
311             (inv_time_t*)&timestamp))
312             eMPL_send_data(PACKET_DATA_EULER, data);
313     }
314 }
315 /******发送数据到匿名四轴上位机******/
316 if (1)
317 {
318     char cStr [ 70 ];
319     unsigned long timestamp, step_count, walk_time;
320
321     /*获取欧拉角*/
322     if (inv_get_sensor_type_euler(data, &accuracy, (inv_time_t*)&timestamp))
323     {
324         float Pitch, Roll, Yaw;
325         Pitch = data[0]*1.0/(1<<16);
326         Roll = data[1]*1.0/(1<<16);
327         Yaw = data[2]*1.0/(1<<16);
328
329         /*向匿名上位机发送姿态*/
330         Data_Send_Status(Pitch, Roll, Yaw);
331         /*向匿名上位机发送原始数据*/
332         Send_Data((int16_t *)&sensors.gyro.raw, (int16_t *)&sensors.accel.raw);
333
334         sprintf ( cStr, "Pitch : %.4f ", Pitch ); //inv_get_sensor_type_euler读出的数据是Q16格式, 所以左移16位.
335         ILI9341_DispatchString_EN_CH(30, 90, (const uint8_t *)cStr, macBACKGROUND, macRED);
336
337         sprintf ( cStr, "Roll : %.4f ", Roll ); //inv_get_sensor_type_euler读出的数据是Q16格式, 所以左移16位.
338         ILI9341_DispatchString_EN_CH(30, 110, (const uint8_t *)cStr, macBACKGROUND, macRED);
339
340         sprintf ( cStr, "Yaw : %.4f ", Yaw ); //inv_get_sensor_type_euler读出的数据是Q16格式, 所以左移16位.
341         ILI9341_DispatchString_EN_CH(30, 130, (const uint8_t *)cStr, macBACKGROUND, macRED);
342
343         /*温度*/
344         mpu_get_temperature(data, (inv_time_t*)&timestamp);
345
346         sprintf ( cStr, "Temperature : %.2f ", data[0]*1.0/(1<<16) ); //inv_get_sensor_type_euler读出的数据是Q16格式, 所以左移16位.
347         ILI9341_DispatchString_EN_CH(30, 150, (const uint8_t *)cStr, macBACKGROUND, macRED);
348
349     }
350
351     /*获取步数*/
352     get_tick_count(&timestamp);
353     if (timestamp > hal.next_pedo_ms) {
354         hal.next_pedo_ms = timestamp + PEDO_READ_MS;
355         dmp_get_pedometer_step_count(&step_count);
356         dmp_get_pedometer_walk_time(&walk_time);
357
358         sprintf(cStr, "Walked steps : %ld steps over %ld milliseconds..", step_count, walk_time);
359         ILI9341_DispatchString_EN_CH(0, 180, (const uint8_t *)cStr, macBACKGROUND, macRED);
360     }
361 }
362
363
364
365
366
367
```

图 3-6 `read_from_mpl` 函数

(具体可在源码查看, 截图中的行号与源码行号一致, ISO 版本的 `main` 文件)

函数中根据标志位决定是否读取并向上位机发送四元数、加速度、陀螺仪及欧拉角等操作, 分别为 `inv_get_sensor_type_quat`、`inv_get_sensor_type_accel`、`inv_get_sensor_type_gyro`、`inv_get_sensor_type_euler` 这些函数。在这里我们忽略这些操作,



因为这是官方驱动程序配合数据发送到它的 python 上位机的(下一小节讲解),而我们使用“匿名上位机”来处理数据,这些数据上传由以上代码截图中的 `if(1){}` 结构内的操作完成。

这部分的代码是由我们在官方驱动程序的基础上添加的,它调用了 `inv_get_sensor_type_euler`、`dmp_get_pedometer_step_count` 函数获取了欧拉角、以及计步器测量到的步数,调用 `Data_Send_Status`、`Send_Data` 函数把欧拉角、加速度、陀螺仪数据以“匿名上位机”数据协议格式通过串口上传,同时把这些数据在液晶屏上显示出来。

所以用户使用 MPU6050 的 DMP 功能后,只需要按规定调用 `inv_get_sensor_type_euler` 函数即可获取最新的姿态数据。

3.4 STM32-MPU6050_DMP_python 上位机

“STM32-MPU6050_DMP_python 上位机”这个程序与上一章的“STM32-MPU6050_DMP 测试例程”类似,但它基本上是原汁原味的 MPU6050 官方驱动,我们只做了最小修改,因而它适用于官方提供的 python 上位机。上一章节的例程不适用于官方上位机,而本章的例程则不适用于“匿名上位机”,这主要是串口上传的数据格式不同导致的。**两个程序的主体是一样的,如果用户不懂 python 语言,直接用“匿名上位机”操作即可。以下说明只适用于有 python 基础,且对官方提供的 python 上位机感兴趣的用户。**

这个上位机的使用说明可在配套资料“motion_driver6.12”源码包 documentation 文件夹里的《Motion Driver 6.12 – Getting Started Guide》找到详细说明。
python 上位机的源码在“motion_driver6.12”源码包的 eMPL-pythonclient 文件夹,里边有三个 python 文件,见图 3-7。



图 3-7 源码包里的 python 上位机源码

该上位机支持 python2.7 环境(32 位),并且需要安装 Pyserial 库、Pygame 库。

可通过如下网址找到安装包。

Python: <https://www.python.org/downloads/>

Pyserial: <https://pypi.python.org/pypi/pyserial>



Pygame: <http://www.pygame.org/download.shtml>

使用步骤:

- ❑ 先把本 STM32 工程代码编译后下载到开发板上运行, 正常时开发板的液晶屏现象跟上一章例程的现象一样。
- ❑ 使用命令行切换到 python 上位机的目录, 执行如下命令:

```
python eMPL-client.py <COM PORT NUMBER>
```

其中<COM PORT NUMBER>参数是 STM32 开发板在电脑端的串口设备号, 运行命令后会弹出一个 3D 图形窗口, 显示陀螺仪的姿态, 见图 3-8。(图中的“python2_32”是本机的 python2.7-32 位 python 命令的名字, 用户默认用“python”命令即可。)

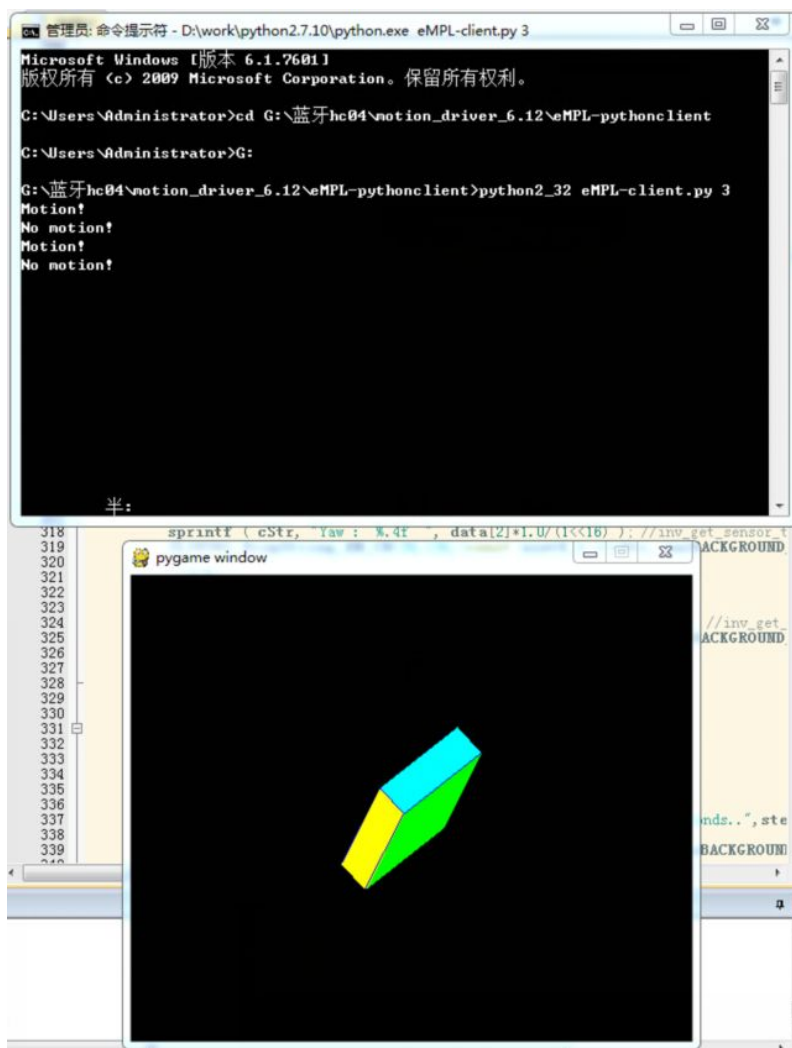
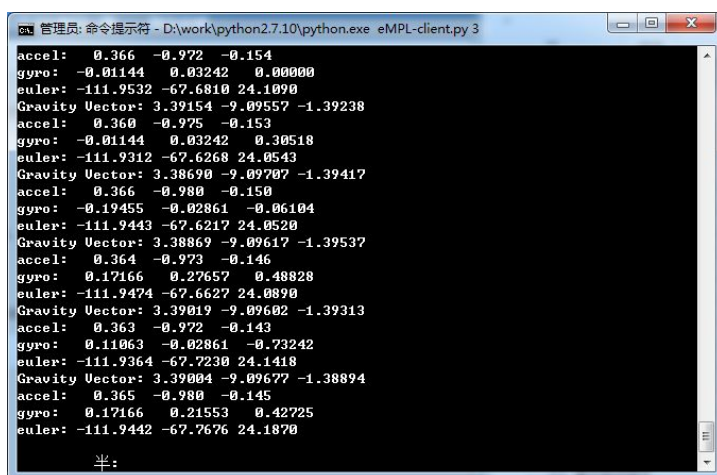


图 3-8 运行 python 上位机

- ❑ 这个上位机还可以接收命令来控制 STM32 进行数据输出, 选中图中的 pygame window 窗口(弹出来的 3D 图形窗口), 然后按下键盘的字母“a”键, 命令行窗口就会输出加速度信息, 按下“g”键, 就会输出陀螺仪信息。命令集说明如下:
 - ‘8’: Toggles Accel Sensor
 - ‘9’: Toggles Gyro Sensor



- '0' : Toggles Compass Sensor
- 'a' : Prints Accel Data
- 'g' : Prints Gyro Data
- 'c' : Prints Compass Data
- 'e' : Prints Euler Data in radius
- 'r' : Prints Rotational Matrix Data
- 'q' : Prints Quaternions
- 'h' : Prints Heading Data in degrees
- 'i' : Prints Linear Acceleration data
- 'o' : Prints Gravity Vector data
- 'w' : Get compass accuracy and status
- 'd' : Register Dump
- 'p' : Turn on Low Power Accel Mode at 20Hz sampling
- 'l' : Load calibration data from flash memory
- 's' : Save calibration data to flash memory
- 't' : run factory self test and calibration routine
- '1' : Change sensor output data rate to 10Hz
- '2' : Change sensor output data rate to 20Hz
- '3' : Change sensor output data rate to 40Hz
- '4' : Change sensor output data rate to 50Hz
- '5' : Change sensor output data rate to 100Hz
- ',' : set interrupts to DMP gestures only
- '.' : set interrupts to DMP data ready
- '6' : Print Pedometer data
- '7' : Reset Pedometer data
- 'f' : Toggle DMP on/off
- 'm' : Enter Low Power Interrupt Mode
- 'x' : Reset the MSP430
- 'v' : Toggle DMP Low Power Quaternion Generation



```
管理员: 命令提示符 - D:\work\python2.7.10\python.exe eMPL-client.py 3
accel: 0.366 -0.972 -0.154
gyro: -0.01144 0.03242 0.00000
euler: -111.9532 -67.6810 24.1090
Gravity Vector: 3.39154 -9.09557 -1.39238
accel: 0.360 -0.975 -0.153
gyro: -0.01144 0.03242 0.30518
euler: -111.9312 -67.6268 24.0543
Gravity Vector: 3.38690 -9.09707 -1.39417
accel: 0.366 -0.980 -0.150
gyro: -0.19455 -0.02861 -0.06104
euler: -111.9443 -67.6217 24.0520
Gravity Vector: 3.38869 -9.09617 -1.39537
accel: 0.364 -0.973 -0.146
gyro: 0.17166 0.27657 0.48828
euler: -111.9474 -67.6627 24.0890
Gravity Vector: 3.39019 -9.09602 -1.39313
accel: 0.363 -0.972 -0.143
gyro: 0.11063 -0.02861 -0.73242
euler: -111.9364 -67.7230 24.1418
Gravity Vector: 3.39004 -9.09677 -1.38894
accel: 0.365 -0.980 -0.145
gyro: 0.17166 0.21553 0.42725
euler: -111.9442 -67.7676 24.1870
半:
```

图 3-9 在 3D 窗口输入命令后的命令行窗口输出

在上一章中提到的 STM32 代码 `read_from_mpl` 函数, 根据标志位决定是否获取欧拉角、加速度、陀螺仪等数据并上传操作, 就是根据这个 python 上位机执行的。(在 `main` 函数里有个检测串口输入的代码, 若检测到串口输入, 则设置相应的标志位。)

有兴趣的读者可以根据这个官方的 python 上位机, 自己编写上位机控制程序。

4. 产品更新及售后支持

野火的产品资料更新会第一时间发布到论坛: <http://www.chuxue123.com>

购买野火产品请到野火官方淘宝店铺: <http://firestm32.taobao.com>

在学习或使用野火产品时遇到问题可在论坛发帖子与我们交流。