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	-
2016/11/19	1.1	新版配套程序使用不同的引脚 添加程序说明。

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 模块原理图》及《MPU6050 模块结构图》。

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

1. MPU6050 模块说明

1.1 MPU6050 简介

BH-MPU6050 是秉火科技推出的六轴传感器模块，见图 1-1，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，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。

图 1-1 MPU6050 模块外观

1.2 特性参数

MPU6050 模块的特性参数见表 1-1

表 1-1MPU6050 模块的特性参数。

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)

1.3 MPU6050 模块的引脚功能说明

该模块引出的 8 个引脚功能说明见表 1-2。

表 1-2 MPU6050 模块引脚说明

序号	引脚名称	说明
1	VCC	3.3/5V 电源输入
2	GND	地线
3	SCL	I2C 从时钟信号线 SCL (模块上已接上拉电阻)
4	SDA	I2C 从数据信号线 SDA (模块上已接上拉电阻)
5	XDA	I2C 主串行数据信号线，用于外接传感器(模块上已接上拉电阻)
6	XCL	I2C 主串行时钟信号线，用于外接传感器(模块上已接上拉电阻)
7	AD0	从机地址设置引脚 ❑ 接地或悬空时, 地址为: 0x68 ❑ 接 VCC 时，地址为:0x69
8	INT	中断输出引脚

其中的 SDA/SCL、XDA/XCL 通讯引脚分别为两组 I2C 信号线。当模块与外部主机通讯时，使用 SDA/SCL，如与 STM32 芯片通讯；而 XDA/XCL 则用于 MPU6050 芯片与其它 I2C 传感器通讯时使用，例如使用它与磁场传感器连接，MPU6050 模块可以把从主机 SDA/SCL 接收的数据或命令通过 XDA/XCL 引脚转发到磁场传感器中。但实际上这种功能比较鸡肋，控制麻烦且效率低，一般会直接把磁场传感器之类的 I2C 传感器直接与 MPU6050 挂载在同一条总线上（即都连接到 SDA/SCL），使用主机直接控制。

1.4 硬件原理图说明

MPU6050 模块的硬件原理图见图 1-2。

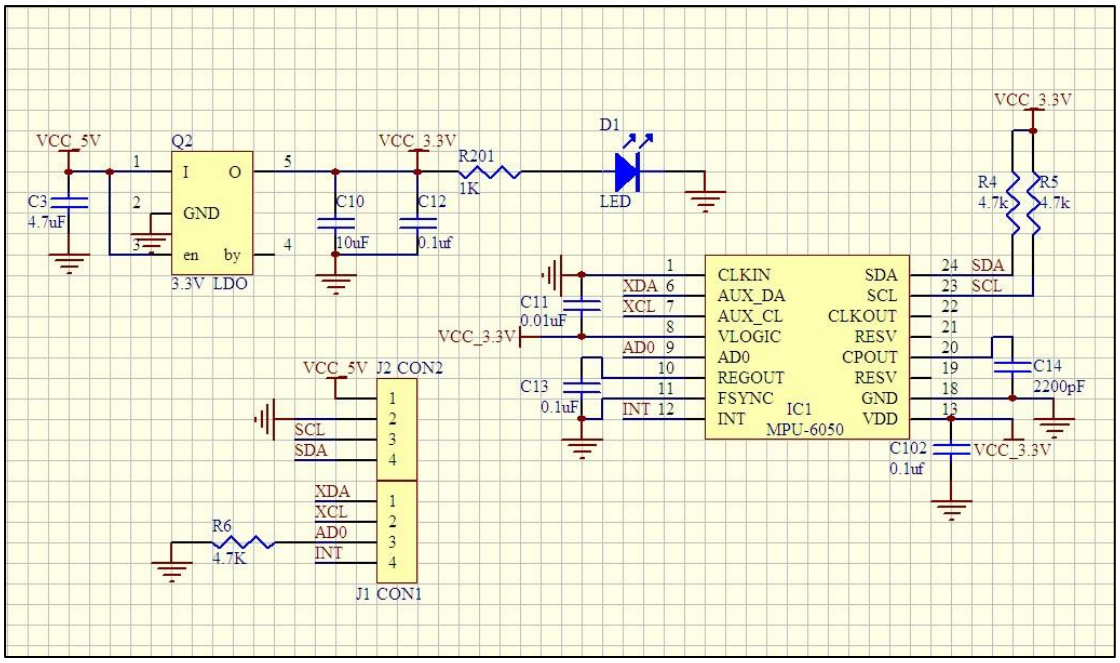

图 1-2 MPU6050 模块原理图

它的硬件非常简单，SDA 与 SCL 被引出方便与外部 I2C 主机连接，看图中的右上角，可知该模块的 I2C 通讯引脚 SDA 及 SCL 已经连接了上拉电阻，因此它与外部 I2C 通讯主机通讯时直接使用导线连接起来即可；而 MPU6050 模块与其它传感器通讯使用的 XDA、XCL 引脚没有接上拉电阻，要使用时需要注意。模块自身的 I2C 设备地址可通过 AD0 引脚的电平控制，当 AD0 接地时，设备地址为 0x68(七位地址)，当 AD0 接电源时，设备地址为 0x69(七位地址)。另外，当传感器有新数据的时候会通过 INT 引脚通知 STM32。

由于 MPU6050 检测时是基于自身中心坐标系的，见图 1-3，它表示的坐标系及旋转符号标出了 MPU6050 传感器的 XYZ 轴的加速度有角速度的正方向。所以在安装模块时，您需要考虑它与所在设备的坐标系统的关系。

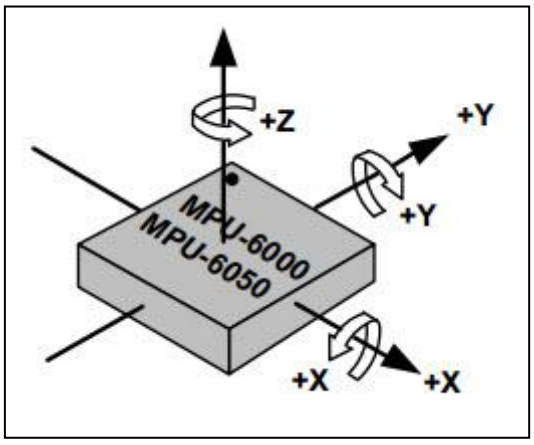

图 1-3 MPU6050 传感器的坐标及方向

2. 硬件测试

本模块配套 STM32 驱动程序，可直接使用秉火 F103 霸道、F103 指南者及 F429 挑战者开发板进行测试。按要求使用杜邦线把模块连接到开发板，并下载程序即可，其中秉火 F429 挑战者开发板板载了 MPU6050 芯片，不需要外接模块，此处直接列出其引脚关系供使用 STM32F4 的用户参考。

在测试前，先用杜邦线把 STM32 开发板与该 MPU6050 模块连接起来，见图 2-1 及表 2-1。

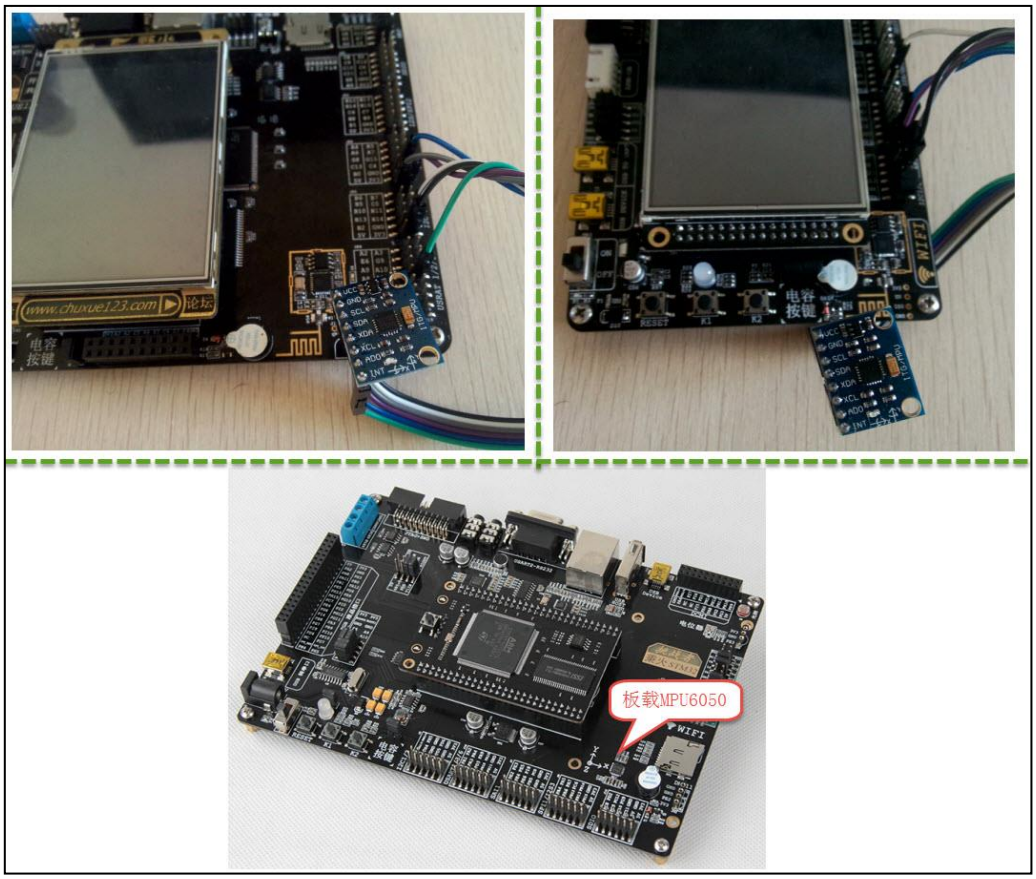

图 2-1 STM32 与 MPU6050 的硬件连接

表 2-1 MPU6050 模块引脚说明

序号	引脚名称	与 F103 霸道及 F103 指南者开发板连接	F429 挑战者板载的陀螺仪
1	VCC	接 3.3V 或 5V	接 3.3V 或 5V
2	GND	GND	GND
3	SCL	PB6	PB6
4	SDA	PB7	PB7
5	AD0	悬空或接地	悬空或接地
6	INT	PA11	PA8

F103 霸道、F103 指南者开发板与模块的连接关系一样，注意陀螺仪的 INT 引脚必须连接，否则程序不会更新数据。

2.1 基本测试

连接好模块后，找到配套资料里的例程“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。

测试工程所在目录：“开发板配套例程\F103 霸道版本[或 F103 指南者、F429 挑战者版本]”。

1. 正常运行的实验现象

由于测试程序中串口输出的是特定格式的数据包，不是字符串，直接用串口调试助手查看是一堆乱码，不便于确认程序是否正常运行，可通过板子配套的液晶屏来查看程序是否运行正常。

该测试程序下载到开发板并复位后，正常运行时液晶屏的实验现象如下：

- 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 2-2；
- 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，所以有误差时并不是模块有问题)；
- 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。（计步的模式匹配需要持续多次重复模仿摆臂运动，请耐心尝试）；

图 2-2 MPU6050 模块正常运行时的液晶显示

2. 不正常运行时故障排查

模块无法正常工作时，可能出现如下三种情况，请对照排查：

- 液晶屏以蓝色字显示“No MPU6050 detected!”，见图 2-3。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。

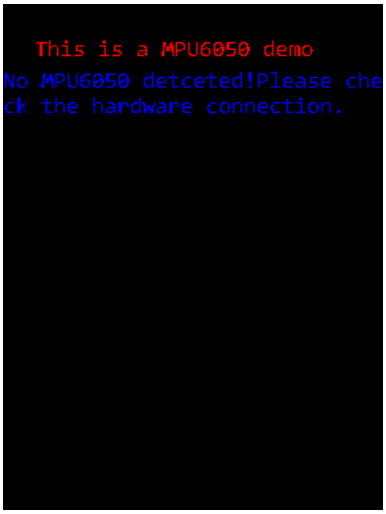

图 2-3 模块检测不正常时的液晶显示

- 若屏幕检测正常，但不显示各种姿态数据，见图 2-4。请检查模块的中断引脚 INT 是否与 STM32 开发板正常连接，连接好后复位开发板，重新检测。

图 2-4 模块检测正常但没有接 INT 中断引脚时的情况

- 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。

2.2 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\ANO_TC 匿名飞控地面站-0512”。

配合该软件查看陀螺仪模块姿态的步骤如下：

- 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。

图 2-5 上位机配置

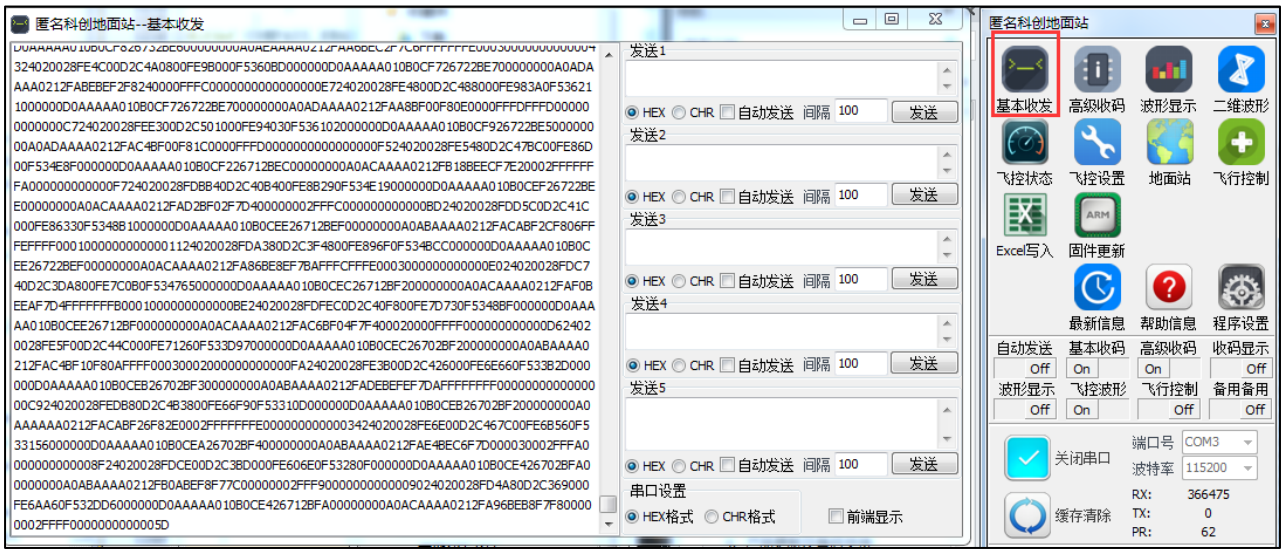

图 2-6 基本收发

图 2-7 波形显示

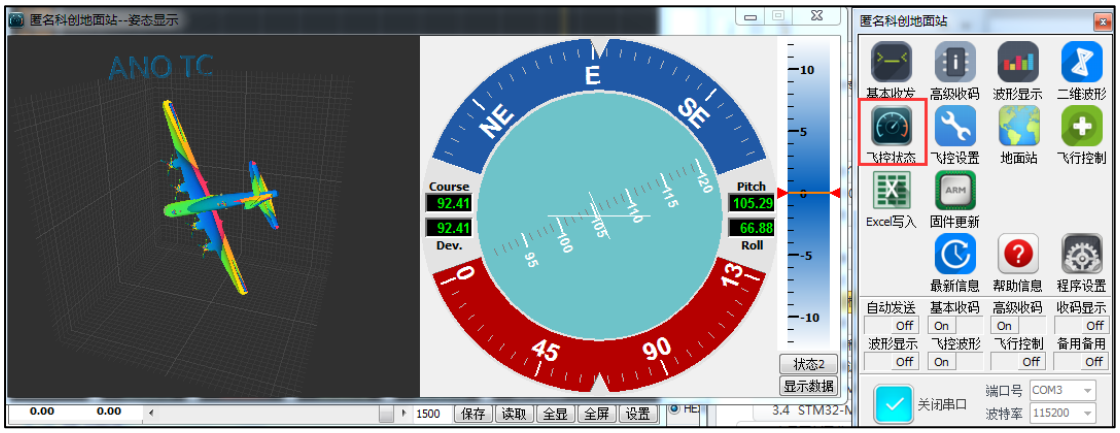

图 2-8 姿态显示

3. 配套程序说明

下面对 MPU6050 模块配套的几个实验做简单的说明。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，各个例程的基本功能介绍见表 3-1。

表 3-1 MPU6050 模块配套程序基本功能介绍

程序	说明
1.硬件 STM32-MPU6050	硬件 I2C，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 I2C 时不能与液晶屏同时使用，因为 FSMC 的 NADV 与 I2C1 的 SDA 是同一个引脚，互相影响了)
2.软件 STM32-MPU6050	软件 I2C，MPU6050 基本驱动程序，不包含 DMP 功能，没有移植官方驱动程序。本程序通过串口和液晶屏输出简单测量数据。(不支持匿名上位机可视数据)。

3.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 I2C。
4.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 I2C。

其中，例程 1 和 2 的功能类似，都是简单地获取数据，但由于硬件 I2C 不支持同时使用液晶屏，所以后面的 2、3、4 例程都默认使用软件 I2C 来驱动 MPU6050，底层的软件 I2C 驱动跟 EEPROM 基本本一致，本实验重点讲述上层的 MPU6050 应用及接口。

注意：本手册中介绍的程序包含有 F103 霸道、F103 指南者及 F429 挑战者开发板配套的版本，实验时请根据自己的实验平台来使用，各版本的程序控制原理类似，下文中不细述与平台相关的内容。（F103 霸道及指南者的程序仅液晶底层驱动不一样，F1 与 F4 的程序陀螺仪使用的引脚有差异，液晶显示函数也不同；F4 版本没有提供软件 I2C 基本读写的工程，直接使用 DMP 工程即可）。

3.1 硬件设计

在实验前，先用杜邦线把 STM32 开发板与该 MPU6050 模块连接起来，请参照前面“硬件测试”章节的说明连接模块。

3.2 MPU6050—获取原始数据实验

这一小节讲解如何使用 STM32 控制 MPU6050 传感器读取加速度、角速度及温度原始数据。在控制传感器时，使用到了 STM32 的 I2C 驱动，对 MPU6050 传感器的不同寄存器写入不同内容可以实现不同模式的控制，从特定的寄存器读取内容则可获取测量数据，这部分关于 MPU6050 具体寄存器的内容我们不再展开，请您查阅《MPU-60X0 寄存器》手册获知。

3.2.1 软件设计

本小节讲解的是“硬件 STM32-MPU6050”实验，请打开配套的代码工程阅读理解。为了方便展示及移植，我们把 STM32 的 I2C 驱动相关的代码都编写到“bsp_i2c.c”及“bsp_i2c.h”文件中，与 MPU6050 传感器相关的代码都写到“mpu6050.c”及“mpu6050.h”文件中。

1. 程序设计要点

(1) 初始化 STM32 的 I2C；

(2) 使用 I2C 向 MPU6050 写入控制参数；

(3) 定时读取加速度、角速度及温度数据。

2. 代码分析

I2C 的硬件定义

本实验中的 I2C 驱动与 MPU6050 驱动分开主要是考虑到扩展其它传感器时的通用性，如使用磁场传感器、气压传感器都可以使用同样一个 I2C 驱动，这个驱动只要给出针对不同传感器时的不同读写接口即可。关于 STM32 的 I2C 驱动原理请参考秉火 STM32 教程《零死角玩转 STM32》中读写 EEPROM 的章节，本章讲解的 I2C 驱动主要针对接口封装讲解，细节不再赘述。本实验中的 I2C 硬件定义见代码清单 3-1。

代码清单 3-1 I2C 的硬件定义(bsp_i2c.h 文件)

```
#define      SENSORS_I2Cx                          I2C1
#define      SENSORS_I2C_APBxClock_FUN             RCC_APB1PeriphClockCmd
#define      SENSORS_I2C_CLK                       RCC_APB1Periph_I2C1
#define      SENSORS_I2C_GPIO_APBxClock_FUN        RCC_APB2PeriphClockCmd
#define      SENSORS_I2C_GPIO_CLK                  RCC_APB2Periph_GPIOB
#define      SENSORS_I2C_SCL_PORT                  GPIOB
#define      SENSORS_I2C_SCL_PIN                   GPIO_Pin_6
#define      SENSORS_I2C_SDA_PORT                  GPIOB
#define      SENSORS_I2C_SDA_PIN                   GPIO_Pin_7
```

这些宏根据传感器使用的 I2C 硬件封装起来了。

初始化 I2C

接下来利用这些宏对 I2C 进行初始化，初始化过程与 I2C 读写 EEPROM 中的无异，见代码清单 3-2。

代码清单 3-2 初始化 I2C（bsp_i2c.c 文件）

```
1
2  /**
3    * @brief  I2C1 I/O 配置
4    * @param  无
5    * @retval 无
6    */
7  static void I2C_GPIO_Config(void)
8  {
9      GPIO_InitTypeDef  GPIO_InitStructure;
10     /* 使能与 I2C1 有关的时钟 */
11     SENSORS_I2C_APBxClock_FUN ( SENSORS_I2C_CLK, ENABLE );
12     SENSORS_I2C_GPIO_APBxClock_FUN ( SENSORS_I2C_GPIO_CLK, ENABLE );
13
14     /* PB6-I2C1_SCL、PB7-I2C1_SDA*/
15     GPIO_InitStructure.GPIO_Pin = SENSORS_I2C_SCL_PIN;
16     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
17     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;        // 开漏输出
18     GPIO_Init(SENSORS_I2C_SCL_PORT, &GPIO_InitStructure);
19
20     GPIO_InitStructure.GPIO_Pin = SENSORS_I2C_SDA_PIN;
21     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
22     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;        // 开漏输出
23     GPIO_Init(SENSORS_I2C_SDA_PORT, &GPIO_InitStructure);
24 }
25
26
27 /**
```

```
28   * @brief  I2C 工作模式配置
29   * @param  无
30   * @retval 无
31   */
32 static void I2C_Mode_Configu(void)
33 {
34     I2C_InitTypeDef  I2C_InitStructure;
35 
36     /* I2C 配置 */
37     I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
38 
39     /* 高电平数据稳定，低电平数据变化 SCL 时钟线的占空比 */
40     I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
41 
42     I2C_InitStructure.I2C_OwnAddress1 =I2Cx_OWN_ADDRESS7;
43     I2C_InitStructure.I2C_Ack = I2C_Ack_Enable ;
44 
45     /* I2C 的寻址模式 */
46     I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
47 
48     /* 通信速率 */
49     I2C_InitStructure.I2C_ClockSpeed = I2C_Speed;
50 
51     /* I2C1 初始化 */
52     I2C_Init(SENSORS_I2Cx, &I2C_InitStructure);
53 
54     /* 使能 I2C1 */
55     I2C_Cmd(SENSORS_I2Cx, ENABLE);
56 }
```

对读写函数的封装

初始化完成后就是编写 I2C 读写函数了，这部分跟 EERPOM 的一样，主要是调用 STM32 标准库函数读写数据寄存器及标志位，本实验的这部分被编写进 ST_Sensors_I2C_WriteRegister 及 ST_Sensors_I2C_ReadRegister 中了，在它们之上，再封装成了 Sensors_I2C_WriteRegister 及 Sensors_I2C_ReadRegister，见代码清单 3-3。

代码清单 3-3 对读写函数的封装(i2c.c 文件)

```
 1 
 2 /**
 3   * @brief  写寄存器(多次尝试)，这是提供给上层的接口
 4   * @param  slave_addr: 从机地址
 5   * @param  reg_addr:寄存器地址
 6   * @param len: 写入的长度
 7   * @param data_ptr:指向要写入的数据
 8   * @retval 正常为 0，不正常为非 0
 9   */
10 int Sensors_I2C_WriteRegister(unsigned char slave_addr,
11                                unsigned char reg_addr,
12                                unsigned short len,
13                                const unsigned char *data_ptr)
14 {
15     char retries=0;
16     int ret = 0;
17     unsigned short retry_in_mlsec = Get_I2C_Retry();
18 
19 tryWriteAgain:
20     ret = 0;
21     ret = ST_Sensors_I2C_WriteRegister( slave_addr, reg_addr, len, data_ptr);
22 
23     if (ret && retry_in_mlsec) {
24         if ( retries++ > 4 )
```

```
25              return ret;
26
27          Delay(retry_in_mlsec);
28          goto tryWriteAgain;
29      }
30      return ret;
31  }
32
33  /**
34    * @brief   读寄存器(多次尝试)，这是提供给上层的接口
35    * @param   slave_addr: 从机地址
36    * @param   reg_addr:寄存器地址
37    * @param len: 要读取的长度
38    * @param data_ptr:指向要存储数据的指针
39    * @retval 正常为0，不正常为非0
40    */
41  int Sensors_I2C_ReadRegister(unsigned char slave_addr,
42                               unsigned char reg_addr,
43                               unsigned short len,
44                               unsigned char *data_ptr)
45  {
46      char retries=0;
47      int ret = 0;
48      unsigned short retry_in_mlsec = Get_I2C_Retry();
49
50  tryReadAgain:
51      ret = 0;
52      ret = ST_Sensors_I2C_ReadRegister( slave_addr, reg_addr, len, data_ptr);
53
54      if (ret && retry_in_mlsec) {
55          if ( retries++ > 4 )
56              return ret;
57
58          Delay(retry_in_mlsec);
59          goto tryReadAgain;
60      }
61      return ret;
62  }
```

封装后的函数主要是增加了错误重试机制，若读写出现错误，则会进行多次尝试，多次尝试均失败后会返回错误代码。这个函数作为 I2C 驱动对外的接口，其它使用 I2C 的传感器调用这个函数进行读写寄存器。

MPU6050 的寄存器定义

MPU6050 有各种各样的寄存器用于控制工作模式，我们把这些寄存器的地址、寄存器位使用宏定义到了 mpu6050.h 文件中了，见代码清单 3-4。

代码清单 3-4MPU6050 的寄存器定义(mpu6050.h)

```
1  //模块的 A0 引脚接 GND，IIC 的 7 位地址为 0x68，若接到 VCC，需要改为 0x69
2  #define MPU6050_SLAVE_ADDRESS   (0x68<<1)       //MPU6050 器件读地址
3  #define MPU6050_WHO_AM_I         0x75
4  #define MPU6050_SMPLRT_DIV       0   //8000Hz
5  #define MPU6050_DLPF_CFG         0
6  #define MPU6050_GYRO_OUT         0x43      //MPU6050 陀螺仪数据寄存器地址
7  #define MPU6050_ACC_OUT          0x3B      //MPU6050 加速度数据寄存器地址
11 #define MPU6050_RA_XG_OFFS_TC       0x00 //[7] PWR_MODE, [6:1]
12 /*.........以下部分省略*/
```

初始化 MPU6050

根据 MPU6050 的寄存器功能定义，我们使用 I2C 往寄存器写入特定的控制参数，见代码清单 3-5。

代码清单 3-5 初始化 MPU6050(mpu6050.c)

```
1
2  /**
3    * @brief    写数据到 MPU6050 寄存器
4    * @param    reg_add:寄存器地址
5    * @param    reg_data:要写入的数据
6    * @retval
7    */
8  void MPU6050_WriteReg(u8 reg_add,u8 reg_dat)
9  {
10     Sensors_I2C_WriteRegister(MPU6050_ADDRESS,reg_add,1,&reg_dat);
11 }
12
13 /**
14   * @brief    从 MPU6050 寄存器读取数据
15   * @param    reg_add:寄存器地址
16   * @param    Read：存储数据的缓冲区
17   * @param    num：要读取的数据量
18   * @retval
19   */
20 void MPU6050_ReadData(u8 reg_add,unsigned char* Read,u8 num)
21 {
22     Sensors_I2C_ReadRegister(MPU6050_ADDRESS,reg_add,num,Read);
23 }
24
25
26 /**
27   * @brief    初始化 MPU6050 芯片
28   * @param
29   * @retval
30   */
31 void MPU6050_Init(void)
32 {
33     int i=0,j=0;
34     //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
35     for (i=0; i<1000; i++) {
36         for (j=0; j<1000; j++) {
37             ;
38         }
39     }
40     //解除休眠状态
41     MPU6050_WriteReg(MPU6050_RA_PWR_MGMT_1, 0x00);
42     //陀螺仪采样率
43     MPU6050_WriteReg(MPU6050_RA_SMPLRT_DIV , 0x07);
44     MPU6050_WriteReg(MPU6050_RA_CONFIG , 0x06);
45     //配置加速度传感器工作在 16G 模式
46     MPU6050_WriteReg(MPU6050_RA_ACCEL_CONFIG , 0x01);
47     //陀螺仪自检及测量范围，典型值：0x18(不自检，2000deg/s)
48     MPU6050_WriteReg(MPU6050_RA_GYRO_CONFIG, 0x18);
49 }
```

这段代码首先使用 MPU6050_ReadData 及 MPU6050_WriteRed 函数封装了 I2C 的底层读写驱动，接下来用它们在 MPU6050_Init 函数中向 MPU6050 寄存器写入控制参数，设置了 MPU6050 的采样率、量程(分辨率)。

读传感器 ID

初始化后，可通过读取它的“WHO AM I”寄存器内容来检测硬件是否正常，该寄存器存储了 ID 号 0x68，见代码清单 3-6。

代码清单 3-6 读取传感器 ID(mpu6050.c)

```
1
2  /**
3    * @brief    读取 MPU6050 的 ID
4    * @param
5    * @retval   正常返回 1，异常返回 0
6    */
7  uint8_t MPU6050ReadID(void)
8  {
9      unsigned char Re = 0;
10     MPU6050_ReadData(MPU6050_RA_WHO_AM_I,&Re,1);     //读器件地址
11     if (Re != 0x68) {
12         MPU_ERROR("检测不到 MPU6050 模块，请检查模块与开发板的接线");
13         return 0;
14     } else {
15         MPU_INFO("MPU6050 ID = %d\r\n",Re);
16         return 1;
17     }
18
19 }
```

读取原始数据

若传感器检测正常，就可以读取它数据寄存器获取采样数据了，见代码清单 3-7。

代码清单 3-7 读取传感器数据(mpu6050.c)

```
1
2  /**
3    * @brief    读取 MPU6050 的加速度数据
4    * @param
5    * @retval
6    */
7  void MPU6050ReadAcc(short *accData)
8  {
9      u8 buf[6];
10     MPU6050_ReadData(MPU6050_ACC_OUT, buf, 6);
11     accData[0] = (buf[0] << 8) | buf[1];
12     accData[1] = (buf[2] << 8) | buf[3];
13     accData[2] = (buf[4] << 8) | buf[5];
14 }
15
16 /**
17   * @brief    读取 MPU6050 的角加速度数据
18   * @param
19   * @retval
20   */
21 void MPU6050ReadGyro(short *gyroData)
22 {
23     u8 buf[6];
24     MPU6050_ReadData(MPU6050_GYRO_OUT,buf,6);
25     gyroData[0] = (buf[0] << 8) | buf[1];
26     gyroData[1] = (buf[2] << 8) | buf[3];
27     gyroData[2] = (buf[4] << 8) | buf[5];
28 }
29
30 /**
31   * @brief    读取 MPU6050 的原始温度数据
```

```
32    * @param
33    * @retval
34    */
35  void MPU6050ReadTemp(short *tempData)
36  {
37      u8 buf[2];
38      MPU6050_ReadData(MPU6050_RA_TEMP_OUT_H,buf,2);      //读取温度值
39      *tempData = (buf[0] << 8) | buf[1];
40  }
41
42  /**
43    * @brief    读取 MPU6050 的温度数据，转化成摄氏度
44    * @param
45    * @retval
46    */
47  void MPU6050_ReturnTemp(float*Temperature)
48  {
49      short temp3;
50      u8 buf[2];
51
52      MPU6050_ReadData(MPU6050_RA_TEMP_OUT_H,buf,2);      //读取温度值
53      temp3= (buf[0] << 8) | buf[1];
54      *Temperature=((double) (temp3 /340.0))+36.53;
55  }
```

其中前以上三个函数分别用于读取三轴加速度、角速度及温度值，这些都是原始的 ADC 数值(16 位长)，对于加速度和角速度，把读取得的 ADC 值除以分辨率，即可求得实际物理量数值。最后一个函数 MPU6050_ReturnTemp 展示了温度 ADC 值与实际温度值间的转换，它是根据 MPU6050 的说明给出的转换公式进行换算的，注意陀螺仪检测的温度会受自身芯片发热的影响，严格来说它测量的是自身芯片的温度，所以用它来测量气温是不太准确的。对于加速度和角速度值我们没有进行转换，在下一小节中我们直接利用这些数据交给 DMP 单元，求解出姿态角。

main 函数

最后我们来看看本实验的 main 函数，见代码清单 3-8。

代码清单 3-8 main 函数

```

 2  /**
 3    * @brief  主函数
 4    * @param  无
 5    * @retval 无
 6    */
 7  int main(void)
 8  {
 9      short Acel[3];
10      short Gyro[3];
11      float Temp;
12
13      SysTick_Init();
14      SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;
15      /* LED 端口初始化 */
16      LED_GPIO_Config();
17      /* 串口通信初始化 */
18      USART_Config();
19
20      //I2C 初始化
21      I2C_Bus_Init();
22      //MPU6050 初始化
```

```
23      MPU6050_Init();
24
25      //检测 MPU6050
26      if (MPU6050ReadID() == 1) {
27          while (1) {
28              if (Task_Delay[0]==TASK_ENABLE) {
29                  LED2_TOGGLE;
30                  Task_Delay[0]=1000;
31              }
32              if (Task_Delay[1]==0) {
33                  MPU6050ReadAcc(Acel);
34                  printf("加速度：%8d%8d%8d",Acel[0],Acel[1],Acel[2]);
35                  MPU6050ReadGyro(Gyro);
36                  printf("    陀螺仪%8d%8d%8d",Gyro[0],Gyro[1],Gyro[2]);
37                  MPU6050_ReturnTemp(&Temp);
38                  printf("    温度%8.2f\r\n",Temp);
39
40                  Task_Delay[1]=500;
41                 //更新一次数据，可根据自己的需求，提高采样频率，如 100ms 采样一次
42              }
43              //******************* 下面是增加任务的格式**************//
44              //    if(Task_Delay[i]==0)
45              //    {
46              //      Task(i);
47              //      Task_Delay[i]=;
48              //    }
49          }
50      } else {
51          printf("\r\n 没有检测到 MPU6050 传感器！ \r\n");
52          LED_RED;
53          while (1);
54      }
55  }
```

本实验中控制 MPU6050 并没有使用中断检测，我们是利用 Systick 定时器进行计时，隔一段时间读取 MPU6050 的数据寄存器获取采样数据的，代码中使用 Task_Delay 变量来控制定时时间，在 Systick 中断里会每隔 1ms 对该变量值减 1，所以当它的值为 0 时表示定时时间到。

在 main 函数里，调用 I2C_Bus_Init、MPU6050_Init 及 MPU6050ReadID 函数后，就在 whlie 循环里判断定时时间，定时时间到后就读取加速度、角速度及温度值，并使用串口打印信息到电脑端。

3.2.2 下载验证

使用杜邦线连接好开发板和模块，用 USB 线连接开发板“USB TO UART”接口跟电脑，在电脑端打开串口调试助手，把编译好的程序下载到开发板。在串口调试助手可看到 MPU6050 采样得到的调试信息。

3.3 MPU6050—利用 DMP 进行姿态解算

上一小节我们仅利用 MPU6050 采集了原始的数据，如果您对姿态解算的算法深有研究，可以自行编写姿态解算的算法，并利用这些数据，使用 STM32 进行姿态解算，解算后

输出姿态角。而由于 MPU6050 内部集成了 DMP，不需要 STM32 参与解算，可直接输出姿态角，也不需要对解算算法作深入研究，非常方便，本章讲解如何使用 DMP 进行解算。

实验中使用的代码主体是从 MPU6050 官方提供的驱动《motion_driver_6.12》移植过来的，该资料包里提供了基于 STM32F4 控制器的源代码（本工程正是利用该代码移植到 STM32F1 上的）及使用 python 语言编写的上位机，资料中还附带了说明文档，请您充分利用官方自带的资料学习。

3.3.1 硬件设计

硬件设计与上一小节实验中的完全一样，且软件中使用了 INT 引脚产生的中断信号，本小节中的代码默认使用软件 I2C。

3.3.2 软件设计

本小节讲解的是“3.MPU6050_python 上位机”实验，请打开配套的代码工程阅读理解。本工程是从官方代码移植过来的(IAR 工程移植至 MDK)，改动并不多，我们主要给读者讲解一下该驱动的设计思路，方便应用。由于本工程的代码十分庞大，在讲解到某些函数时，请善用 MDK 的搜索功能，从而在工程中查找出对应的代码。

1. 程序设计要点

(1) 提供 I2C 读写接口、定时服务及 INT 中断处理；
(2) 从陀螺仪中获取原始数据并处理；
(3) 更新数据并输出。

2. 代码分析

官方的驱动主要是了 MPL 软件库(Motion Processing Library)，要移植该软件库我们需要为它提供 I2C 读写接口、定时服务以及 MPU6050 的数据更新标志。若需要输出调试信息到上位机，还需要提供串口接口。

I2C 读写接口

MPL 库的内部对 I2C 读写时都使用 i2c_write 及 i2c_read 函数，在文件“inv_mpu.c”中给出了它们的接口格式，见代码清单 3-1。

代码清单 3-9 I2C 读写接口(inv_mpu.c 文件)

```
1 /* 以下函数需要定义成如下格式:
2  * i2c_write(unsigned char slave_addr, unsigned char reg_addr,
3  *       unsigned char length, unsigned char const *data)
4  * i2c_read(unsigned char slave_addr, unsigned char reg_addr,
5  *       unsigned char length, unsigned char *data)
6  */
7
8 #define i2c_write   Sensors_I2C_WriteRegister
9 #define i2c_read    Sensors_I2C_ReadRegister
```

这些接口的格式与我们上一小节写的 I2C 读写函数 Sensors_I2C_ReadRegister 及 Sensors_I2C_WriteRegister 一致，所以可直接使用宏替换。

提供定时服务

MPL 软件库中使用到了延时及时间戳功能，要求需要提供 delay_ms 函数实现毫秒级延时，提供 get_ms 获取毫秒级的时间戳，它们的接口格式也在“inv_mpu.c”文件中给出，见代码清单 3-2。

代码清单 3-10 定时服务接口（inv_mpu.c 文件）

```
/*
 * delay_ms(unsigned long num_ms)
 * get_ms(unsigned long *count)
 */

#define delay_ms    Delay_ms
#define get_ms      get_tick_count
```

我们为接口提供的 Delay_ms 及 get_tick_count 函数定义在 bsp_SysTick.c 文件，我们使用 SysTick 每毫秒产生一次中断，进行计时，见代码清单 3-11。

代码清单 3-11 使用 Systick 进行定时（bsp_SysTick.c）

```

static __IO u32 TimingDelay;
static __IO uint32_t g_ul_ms_ticks=0;
/**
  * @brief   us 延时程序,1ms 为一个单位
  * @param
  *   @arg nTime: Delay_ms( 1 ) 则实现的延时为 1 ms
  * @retval  无
  */
void Delay_ms(__IO u32 nTime)
{
    TimingDelay = nTime;

    while (TimingDelay != 0);
}

/**
  * @brief  获取节拍程序
  * @param  无
  * @retval 无
  * @attention  在 SysTick 中断函数 SysTick_Handler()调用
  */
void TimingDelay_Decrement(void)
{
    if (TimingDelay != 0x00) {
        TimingDelay--;
    }
}

/**
  * @brief  获取当前毫秒值
  * @param  存储最新毫秒值的变量
  * @retval 无
  */
int get_tick_count(unsigned long *count)
{
    count[0] = g_ul_ms_ticks;
```

```
38      return 0;
39  }
40
41  /**
42    * @brief  毫秒累加器，在中断里每毫秒加 1
43    * @param  无
44    * @retval 无
45    */
46  void TimeStamp_Increment (void)
47  {
48      g_ul_ms_ticks++;
49  }
```

上述代码中的 TimingDelay_Decrement 和 TimeStamp_Increment 函数是在 Systick 的中断服务函数中被调用的，见代码清单 3-12。systick 被配置为每毫秒产生一次中断，而每次中断中会对 TimingDelay 变量减 1，对 g_ul_ms_ticks 变量加 1。它们分别用于 Delay_ms 函数利用 TimingDelay 的值进行阻塞延迟，而 get_tick_count 函数获取的时间戳即 g_ul_ms_ticks 的值。

代码清单 3-12 Systick 的中断服务函数(stm32f10x_it.c 文件)

```
1  /**
2    * @brief  SysTick 中断服务函数.
3    * @param  None
4    * @retval None
5    */
6  void SysTick_Handler(void)
7  {
8      TimingDelay_Decrement();
9      TimeStamp_Increment();
10 }
```

提供串口调试接口

MPL 代码库的调试信息输出函数都集中到了 log_stm32.c 文件中，我们可以为这些函数提供串口输出接口，以便把这些信息输出到上位机，见代码清单 3-3。

代码清单 3-13 串口调试接口(log_stm32.c 文件)

```
1  /*串口输出接口*/
2  int fputcc(int ch)
3  {
4      /* 发送一个字节数据到 USART1 */
5      USART_SendData(DEBUG_USART, (uint8_t) ch);
6
7      /* 等待发送完毕 */
8      while (USART_GetFlagStatus(DEBUG_USART, USART_FLAG_TXE) == RESET);
9
10     return (ch);
11 }
12
13 /*输出四元数数据*/
14 void eMPL_send_quat(long *quat)
15 {
16     char out[PACKET_LENGTH];
17     int i;
18     if (!quat)
19         return;
20     memset(out, 0, PACKET_LENGTH);
21     out[0] = '$';
22     out[1] = PACKET_QUAT;
```

```
23      out[3] = (char)(quat[0] >> 24);
24      out[4] = (char)(quat[0] >> 16);
25      out[5] = (char)(quat[0] >> 8);
26      out[6] = (char)quat[0];
27      out[7] = (char)(quat[1] >> 24);
28      out[8] = (char)(quat[1] >> 16);
29      out[9] = (char)(quat[1] >> 8);
30      out[10] = (char)quat[1];
31      out[11] = (char)(quat[2] >> 24);
32      out[12] = (char)(quat[2] >> 16);
33      out[13] = (char)(quat[2] >> 8);
34      out[14] = (char)quat[2];
35      out[15] = (char)(quat[3] >> 24);
36      out[16] = (char)(quat[3] >> 16);
37      out[17] = (char)(quat[3] >> 8);
38      out[18] = (char)quat[3];
39      out[21] = '\r';
40      out[22] = '\n';
41
42      for (i=0; i<PACKET_LENGTH; i++) {
43          fputcc(out[i]);
44      }
45  }
```

上述代码中的 fputcc 函数是我们自己编写的串口输出接口，它与我们重定向 printf 函数定义的 fputc 函数功能很类似。下面的 eMPL_send_quat 函数是 MPL 库中的原函数，它用于打印“四元数信息”，在这个 log_stm32.c 文件中还有输出日志信息的_MLPrintLog 函数，输出原始信息到专用上位机的 eMPL_send_data 函数，它们都调用了 fputcc 进行输出。

MPU6050 的中断接口

与上一小节中的基础实验不同，为了高效处理采样数据，MPL 代码库使用了 MPU6050 的 INT 中断信号，为此我们要给提供中断接口，见代码清单 3-4。

代码清单 3-14 中断接口(stm32f10x_it.c 文件)

```
1
2  #define EXTI_INT_FUNCTION                    EXTI15_10_IRQHandler
3
4  void EXTI_INT_FUNCTION (void)
5  {
6
7   if (EXTI_GetITStatus(MPU_INT_EXTI_LINE) != RESET) { //确保是否产生了 EXTI Line 中断
8          /* 处理新的数据 */
9          gyro_data_ready_cb();
10
11         EXTI_ClearITPendingBit(MPU_INT_EXTI_LINE);      //清除中断标志位
12     }
13 }
```

在工程中我们把 MPU6050 与 STM32 相连的引脚配置成了中断模式，上述代码是该引脚的中断服务函数，在中断里调用了 MPL 代码库的 gyro_data_ready_cb 函数，它设置了标志变量 hal.new_gyro，以通知 MPL 库有新的数据，其函数定义见代码清单 3-15。

代码清单 3-15 设置标志变量(main.c 文件)

```
1  /* 每当有新的数据产生时，本函数会被中断服务函数调用，
2   * 在本工程中，它设置标志位用于指示及保护 FIFO 缓冲区
3
4   */
5  void gyro_data_ready_cb(void)
6  {
```

```
    hal.new_gyro = 1;
}
```

main 函数执行流程

了解 MPL 移植需要提供的接口后，我们直接看 main 函数了解如何利用 MPL 库获取姿态数据，见代码清单 3-5。

代码清单 3-16 使用 MPL 进行姿态解算的过程

```

/**
  * @brief  主函数
  * @param  无
  * @retval 无
  */
int main(void)
{
    inv_error_t result;
    unsigned char accel fsr,  new temp = 0;
    unsigned short gyro rate, gyro fsr;
    unsigned long timestamp;
    struct int_param_s int_param;

    SysTick_Init();
    SysTick->CTRL|=SysTick CTRL ENABLE Msk;
    /* LED 端口初始化 */
    LED GPIO Config();
    LED_BLUE;

    /* 串口通信初始化 */
    USART_Config();

    //MPU6050 中断引脚
    EXTI Pxy Config();
    //I2C 初始化
    I2C Bus Init();

    printf("mpu 6050 test start");

    result = mpu init(&int param);
    if (result) {
        LED RED;
        MPL_LOGE("Could not initialize gyro.result =  %d\n",result);
    } else {
        LED_GREEN;
    }

    /* If you're not using an MPU9150 AND you're not using DMP features, this
    * function will place all slaves on the primary bus.
    * mpu_set_bypass(1);
    */

    result = inv init mpl();
    if (result) {
        MPL_LOGE("Could not initialize MPL.\n");
    }

    /* 计算 6 轴和 9 轴传感器的四元数*/
    inv_enable_quaternion();
    inv enable 9x sensor fusion();

    /* 无运动状态时更新陀螺仪
     * WARNING: These algorithms are mutually exclusive.
     */
    inv_enable_fast_nomot();
    /* inv_enable_motion_no_motion(); */
    /* inv set no motion time(1000); */

    /* 当温度变化时更新 陀螺仪*/
    inv_enable_gyro_tc();

```

```
63      /* 允许 read_from_mpl 使用 MPL APIs. */
64      inv_enable_eMPL_outputs();
65
66      result = inv start mpl();
67      if (result == INV ERROR NOT AUTHORIZED) {
68          while (1) {
69              MPL_LOGE("Not authorized.\n");
70          }
71      }
72      if (result) {
73          MPL LOGE("Could not start the MPL.\n");
74      }
75
76      /* 设置寄存器，开启陀螺仪 */
77      /* 唤醒所有传感器 */
78
79      mpu_set_sensors(INV_XYZ_GYRO | INV_XYZ_ACCEL);
80      /* 把陀螺仪及加速度数据放进 FIFO */
81      mpu_configure_fifo(INV_XYZ_GYRO | INV_XYZ_ACCEL);
82      mpu set sample rate(DEFAULT MPU HZ);
83
84      /* 重新读取配置，确认前面的设置成功  */
85      mpu get sample rate(&gyro rate);
86      mpu_get_gyro_fsr(&gyro_fsr);
87      mpu_get_accel_fsr(&accel_fsr);
88
89      /*使用 MPL 同步配置 */
90      /* 设置每毫秒的采样率*/
91      inv_set_gyro_sample_rate(1000000L / gyro_rate);
92      inv set accel sample rate(1000000L / gyro rate);
93
94      /* 设置 chip-to-body 原点矩阵.
95       * 设置硬件单位为 dps/g's/degrees 因子.
96       */
97      inv_set_gyro_orientation_and_scale(
98          inv_orientation_matrix_to_scalar(gyro_pdata.orientation),
99          (long)gyro_fsr<<15);
100     inv_set_accel_orientation_and_scale(
101         inv orientation matrix to scalar(gyro pdata.orientation),
102         (long)accel_fsr<<15);
103
104     /* 初始化硬件状态相关的变量. */
105
106     hal.sensors = ACCEL_ON | GYRO_ON;
107
108     hal.dmp on = 0;
109     hal.report = 0;
110     hal.rx.cmd = 0;
111     hal.next_pedo_ms = 0;
112     hal.next_compass_ms = 0;
113     hal.next temp ms = 0;
114
115     /* 获取时间戳 */
116     get_tick_count(&timestamp);
117
118     /* 初始化 DMP 步骤:
119      * 1. 调用 dmp_load_motion_driver_firmware().
120           它会把 inv_mpu_dmp_motion_driver.h 文件中的 DMP 固件写入到 MPU 的存储空间
121      * 2. 把陀螺仪和加速度的原始数据矩阵送入 DMP.
122      * 3. 注册姿态回调函数. 除非相应的特性使能了，否则该回调函数不会被执行
123      * 4. 调用 dmp_enable_feature(mask) 使能不同的特性.
124      * 5. 调用 dmp_set_fifo_rate(freq) 设置 DMP 输出频率.
125      * 6. 调用特定的特性控制相关的函数.
126      *
127      * 调用 mpu_set_dmp_state(1)使能 DMP. 该函数可在 DMP 运行时被重复调用设置使能或关闭
128      *
129      * 以下是 inv mpu dmp motion driver.c 文件中的 DMP 固件提供的特性的简介:
130      * DMP_FEATURE_LP_QUAT: 使用 DMP 以 200Hz 的频率产生一个只包含陀螺仪的四元数数据
131        以高速的状态解算陀螺仪数据，减少错误(相对于使用 MCU 以一个低采样率的方式采样计算)
132      * DMP_FEATURE_6X_LP_QUAT: 使用 DMP 以 200Hz 的频率产生 陀螺仪/加速度 四元数 .
133        它不能与前面的 DMP_FEATURE_LP_QUAT 同时使用
134      * DMP_FEATURE_TAP: 检测 X, Y, 和 Z 轴.
135
```

```
136      * DMP_FEATURE_ANDROID_ORIENT: 谷歌屏幕翻转算法. 当屏幕翻转时，在四个方向产生一个事件
137      * DMP FEATURE GYRO CAL: 若 8s 内都没有运动，计算陀螺仪的数据
138      * DMP_FEATURE_SEND_RAW_ACCEL: 添加原始 加速度 数据到 FIFO.
139      * DMP_FEATURE_SEND_RAW_GYRO: 添加原始 陀螺仪 数据到 FIFO.
140      * DMP_FEATURE_SEND_CAL_GYRO: 添加 校准后的 陀螺仪 数据到 FIFO.
141        它不能与 DMP_FEATURE_SEND_RAW_GYRO.同时使用
142      */
143
144     dmp_load_motion_driver_firmware();
145     dmp_set_orientation(
146         inv_orientation_matrix_to_scalar(gyro_pdata.orientation));
147     dmp register tap cb(tap cb);
148
149     dmp_register_android_orient_cb(android_orient_cb);
150     /*
151      * DMP 传感器整合只能工作在陀螺仪+-2000dps 及加速度+-2G 的量程上。
152      */
153     hal.dmp features = DMP FEATURE 6X LP QUAT | DMP FEATURE TAP |
154        DMP FEATURE ANDROID ORIENT | DMP FEATURE SEND RAW ACCEL |
155             DMP FEATURE SEND CAL GYRO |DMP FEATURE GYRO CAL;
156     dmp_enable_feature(hal.dmp_features);
157     dmp_set_fifo_rate(DEFAULT_MPU_HZ);
158     mpu_set_dmp_state(1);
159     hal.dmp on = 1;
160
161
162     while (1) {
163
164         unsigned long sensor_timestamp;
165         int new data = 0;
166         if (USART_GetFlagStatus (DEBUG_USARTx, USART_FLAG_RXNE)) {
167             /* 已通过串口接收到数据. 使用 handle_input 来处理串口接收到的命令
168              * 这部分是处理 python 官方上位机串口发送的指令的。
169              */
170             USART_ClearFlag(DEBUG_USARTx, USART_FLAG_RXNE);
171
172             handle_input();
173         }
174         get tick count(&timestamp);
175
176         /* 温度数据不需要与陀螺仪数据那每次都采样，这里设置隔一段时间采样
177          */
178         if (timestamp > hal.next_temp_ms) {
179             hal.next temp ms = timestamp + TEMP READ MS;
180             new temp = 1;
181         }
182         if (hal.new_gyro && hal.dmp_on) {
183             short gyro[3], accel_short[3], sensors;
184             unsigned char more;
185             long accel[3], quat[4], temperature;
186             /* 当使用 DMP 时，本函数从 FIFO 读取新的数据
187             FIFO 中存储了陀螺仪、加速度、四元数及手势数据.
188             传感器参数可告知调用者哪种有新数据
189     * 例如, if sensors == (INV_XYZ_GYRO | INV_WXYZ_QUAT), 那么 FIFO 中就不包含加速度数据
190              * 手势数据的解算由是否产生了手势运动事件来触发
191              * 若产生了事件，应用函数会使用回调函数来通知
192              */
193       dmp_read_fifo(gyro, accel_short, quat, &sensor_timestamp, &sensors, &more);
194             if (!more)
195                 hal.new gyro = 0;
196             if (sensors & INV_XYZ_GYRO) {
197                 /* 把新数据送入 MPL. */
198                 inv_build_gyro(gyro, sensor_timestamp);
199                 new_data = 1;
200                 if (new_temp) {
201                     new temp = 0;
202                     /* 温度数据只用于陀螺仪的暂时计算 */
203                     mpu get temperature(&temperature, &sensor timestamp);
204                     inv build temp(temperature, sensor timestamp);
205                 }
206             }
207             if (sensors & INV XYZ ACCEL) {
208                 accel[0] = (long)accel short[0];
209                 accel[1] = (long)accel_short[1];
```

```
210                 accel[2] = (long)accel_short[2];
211                 inv_build_accel(accel, 0, sensor_timestamp);
212                 new_data = 1;
213             }
214             if (sensors & INV_WXYZ_QUAT) {
215                 inv_build_quat(quat, 0, sensor_timestamp);
216                 new_data = 1;
217             }
218         }
219         if (new_data) {
220             inv_execute_on_data();
221 
222             /* 本函数读取补偿的后的传感器数据和经过 MPL 传感器融合后输出的数据
223               输出的格式见 in_eMPL_outputs.c 文件.
224               这个函数在主机需要数据的时候调用即可，对调用频率无要求。
225              */
226             read_from_mpl();
227         }
228     }
229 
```

如您所见，main 函数非常长，而且我们只是摘抄了部分，在原工程代码中还有很多代码，以及不同模式下的条件判断分支，例如加入磁场数据使用 9 轴数据进行解算的功能(这是 MPU9150 的功能，MPU6050 不支持)以及其它工作模式相关的控制示例。上述 main 函数的主要执行流程概括如下：

(1) 初始化 STM32 的硬件，如 Systick、LED、调试串口、INT 中断引脚以及 I2C 外设的初始化；

(2) 调用 MPL 库函数 mpu_init 初始化传感器的基本工作模式(以下过程调用的大部分都是 MPL 库函数，不再强调)；

(3) 调用 inv_init_mpl 函数初始化 MPL 软件库，初始化后才能正常进行解算；

(4) 设置各种运算参数，如四元数运算(inv_enable_quaternion)、6 轴或 9 轴数据融合(inv_enable_9x_sensor_fusion)等等；

(5) 设置传感器的工作模式(mpu_set_sensors)、采样率(mpu_set_sample_rate)、分辨率(inv_set_gyro_orientation_and_scale)等等；

(6) 当 STM32 驱动、MPL 库、传感器工作模式、DMP 工作模式等所有初始化工作都完成后进行 while 循环；

(7) 在 while 循环中检测串口的输入，若串口有输入，则调用 handle_input 根据串口输入的字符(命令)，切换工作方式。这部分主要是为了支持上位机通过输入命令，根据进行不同的处理，如开、关加速度信息的采集或调试信息的输出等；

(8) 在 while 循环中检测是否有数据更新(`if (hal.new_gyro && hal.dmp_on)`)，当有数据更新的时候产生 INT 中断，会使 hal.new_gyro 置 1 的，从而执行 if 里的条件代码；

(9) 使用 dmp_read_fifo 把数据读取到 FIFO，这个 FIFO 是指 MPL 软件库定义的一个缓冲区，用来缓冲最新采集得的数据；

(10) 调用 inv_build_gyro、inv_build_temp、inv_build_accel 及 inv_build_quat 函数处理数据角速度、温度、加速度及四元数数据，并对标志变量 new_data 置 1；

(11) 在 while 循环中检测 new_data 标志位，当有新的数据时执行 if 里的条件代码；

(12) 调用 inv_execute_on_data 函数更新所有数据及状态；

(13) 调用 read_from_mpl 函数向主机输出最新的数据。

数据输出接口

在上面 main 中最后调用的 read_from_mpl 函数演示了如何调用 MPL 数据输出接口，通过这些接口我们可以获得想要的数据，其函数定义见代码清单 3-17。

代码清单 3-17 MPL 的数据输出接口(main.c)

```
/* 从 MOL 中获取数据 MPL.
 */
static void read from mpl(void)
{
    long msg, data[9];
    int8_t accuracy;
    unsigned long timestamp;
    float float data[3] = {0};

    MPU_DEBUG_FUNC();
    if (inv_get_sensor_type_quat(data, &accuracy, (inv_time_t*)&timestamp)) {
        /* 发送四元数数据包到 PC. 这些数据由 python 上位机构建 3D 模型姿态，
          因此每当有新数据就发送上去
         */
        eMPL_send_quat(data);

        /* 使用命令，可以控制它发送指定的数据包 */
        if (hal.report & PRINT QUAT)
            eMPL_send_data(PACKET_DATA_QUAT, data);
    }

    if (hal.report & PRINT_ACCEL) {
        if (inv_get_sensor_type_accel(data, &accuracy,
                                      (inv time t*)&timestamp))
            eMPL send data(PACKET DATA ACCEL, data);

    }
    if (hal.report & PRINT_GYRO) {
        if (inv_get_sensor_type_gyro(data, &accuracy,
                                     (inv time t*)&timestamp))
            eMPL_send_data(PACKET_DATA_GYRO, data);

    }

    if (hal.report & PRINT EULER) {
        if (inv_get_sensor_type_euler(data, &accuracy,
                                      (inv_time_t*)&timestamp))
            eMPL_send_data(PACKET_DATA_EULER, data);

    }

    /*******************使用液晶屏显示数据*************************/
#ifdef USE LCD DISPLAY

    if (1) {
        char cStr [ 70 ];
        unsigned long timestamp,step count,walk time;

        /*获取欧拉角*/
        if (inv_get_sensor_type_euler(data, &accuracy,(inv_time_t*)&timestamp)) {
            //inv get sensor type euler 读出的数据是 Q16 格式，所以左移 16 位.
            sprintf ( cStr, "Pitch :  %.4f  ", data[0]*1.0/(1<<16) );
            ILI9341_DispString_EN(30,90,cStr);

            //inv_get_sensor_type_euler 读出的数据是 Q16 格式，所以左移 16 位.
            sprintf ( cStr, "Roll :  %.4f  ", data[1]*1.0/(1<<16) );
            ILI9341 DispString EN(30,110,cStr);

            //inv get sensor type euler 读出的数据是 Q16 格式，所以左移 16 位.
            sprintf ( cStr, "Yaw :  %.4f  ", data[2]*1.0/(1<<16) );
            ILI9341_DispString_EN(30,130,cStr);

            /*温度*/
            mpu_get_temperature(data,(inv_time_t*)&timestamp);
```

```
65
66              //inv_get_sensor_type_euler 读出的数据是 Q16 格式，所以左移 16 位.
67              sprintf ( cStr, "Temperature :  %.2f  ", data[0]*1.0/(1<<16) );
68              ILI9341 DispString EN(30,150,cStr);
69
70
71          }
72          /*获取步数*/
73          get_tick_count(&timestamp);
74          if (timestamp > hal.next_pedo_ms) {
75
76              hal.next pedo ms = timestamp + PEDO READ MS;
77              dmp_get_pedometer_step_count(&step_count);
78              dmp_get_pedometer_walk_time(&walk_time);
79
80              sprintf(cStr, "Walked steps :  %ld  steps over  %ld  milliseconds..",
                            step count,walk time);
81
82              ILI9341_DispString_EN(0,180,cStr);
83
84          }
85      }
86
87  #endif
88      /*以下省略*/
89  }
```

上述代码展示了使用 inv_get_sensor_type_quat 、 inv_get_sensor_type_accel 、inv_get_sensor_type_gyro、inv_get_sensor_type_euler 及 dmp_get_pedometer_step_count 函数分别获取四元数、加速度、角速度、欧拉角及计步器数据。

代码中的 eMPL_send_data 函数是使用串口按照 PYTHON 上位机格式进行提交数据，上位机根据这些数据对三维模型作相应的旋转。

另外我们自己在代码中加入了液晶显示的代码(#ifdef USE_LCD_DISPLAY 宏内的代码)，它把这些数据输出到实验板上的液晶屏上。

您可根据自己的数据使用需求，参考这个 read_from_mpl 函数对数据输出接口的调用方式，编写自己的应用。

3.3.3 下载验证及 python 上位机的使用

直接下载本程序到开发板，在液晶屏上会观察到姿态角、温度、计步器数据，改变开发板的姿态，数据会更新(计步器数据要模拟走路才会更新)，若直接连接串口调试助手，会接收到一系列的乱码信息，这是正常的，这些数据需要使用官方的 Python 上位机解码。

本实验适用于官方提供的 Python 上位机，它可以把采样的数据传送到上位机，上位机会显示三维模式的姿态。

注意：以下内容仅针对有 Python 编程语言基础的用户，若您不会 Python，而又希望观察到三维模型的姿态，请参考下一小节的实验，它的使用更为简单。

1. Python 上位机源代码及说明

MPU6050 官方提供的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。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**错误!未找到引用源。**。

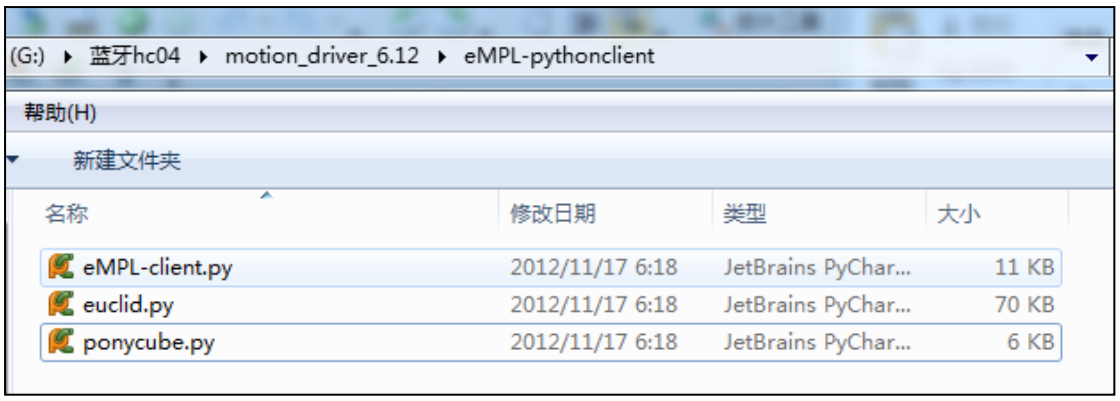

图 3-1 源码包里的 python 上位机源码

2. 安装 Python 环境

要利用上面的源码，需要先安装 Python 环境，该上位机支持 python2.7 环境(仅支持 32 位)，并且需要安装 Pyserial 库(仅支持 Pyserial2.6，不支持 3 版本)、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

3. Python 上位机的使用步骤

- 先把本 STM32 工程代码编译后下载到开发板上运行，确认开发板的 USB TO USART 接口已与电脑相连，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 3-2。(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。)

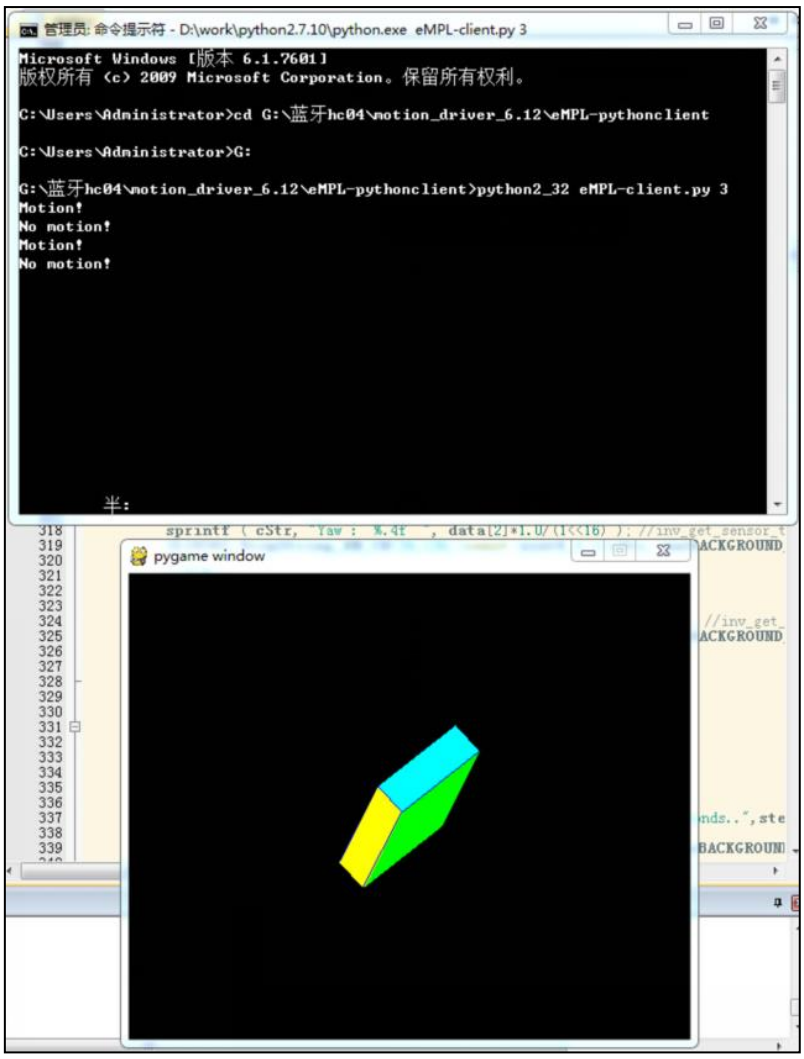

图 3-2 运行 python 上位机

- 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 Eular 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

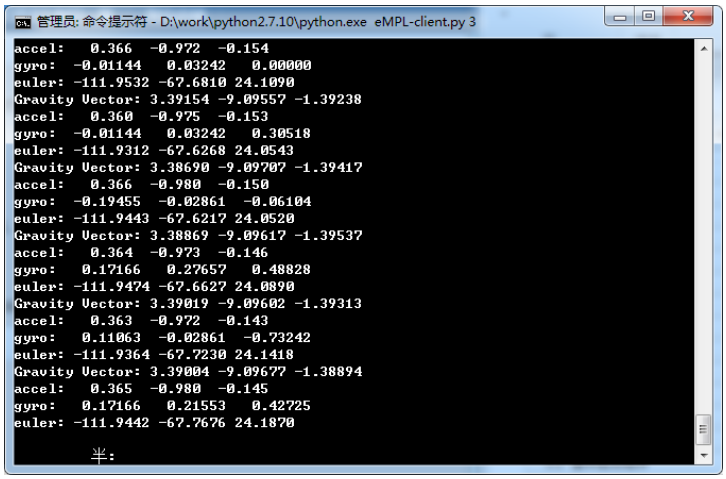

图 3-3 在 3D 窗口输入命令后的命令行窗口输出

在前面提到的 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，就是配合这个 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。

3.4 MPU6050—使用第三方上位机

上一小节中的实验必须配合使用官方提供的上位机才能看到三维模型，而且功能比较简单，所以在小节中我们演示如何把数据输出到第三方的上位机，直观地观察设备的姿态。

实验中我们使用的是“匿名飞控地面站 0512”版本的上位机，关于上位机的通讯协议可查阅《飞控通信协议》文档，或到他们的官方网站了解。

3.4.1 硬件设计

硬件设计与上一小节实验中的完全一样，同样使用了中断 INT 引脚获取数据状态，默认使用软件 I2C 通讯。

3.4.2 软件设计

本小节讲解的是“MPU6050_DMP 测试例程”实验，请打开配套的代码工程阅读理解。本小节的内容主体跟上一小节一样，区别主要是当获取得到数据后，本实验根据“匿名飞控”上位机的数据格式要求上传数据。

1. 程序设计要点

(1) 了解上位机的通讯协议；
(2) 根据协议格式上传数据到上位机；

2. 代码分析

通讯协议

要按照上位机的格式上传数据，首先要了解它的通讯协议，本实验中的上位机协议说明见表 3-2。

表 3-2 匿名上位机的通讯协议(部分)

帧	帧头	功能字	长度	数据	校验
STATUS	AAAA	01	LEN	int16 ROL*100 int16 PIT*100 int16 YAW*100 int32 ALT_USE u8 ARMED : A0 加锁 A1 解锁	SUM

SENSER	AAAA	02	LEN	int16 ACC_X int16 ACC_Y int16 ACC_Z int16 GYRO_X int16 GYRO_Y int16 GYRO_Z int16 MAG_X int16 MAG_Y int16 MAG_Z	SUM

表中说明了两种数据帧，分别是 STATUS 帧及 SENSER 帧，数据帧中包含帧头、功能字、长度、主体数据及校验和。“帧头”用于表示数据包的开始，均使用两个字节的 0xAA 表示；“功能字”用于区分数据帧的类型，0x01 表示 STATUS 帧，0x02 表示 SENSER 帧；“长度”表示后面主体数据内容的字节数；“校验和”用于校验，它是前面所有内容的和。

其中的 STATUS 帧用于向上位机传输横滚角、俯仰角及偏航角的值(100 倍)，SENSER 帧用于传输加速度、角速度及磁场强度的原始数据。

发送数据包

根据以上数据格式的要求，我们定义了两个函数，分别用于发送 STATUS 帧及 SENSER 帧，见代码清单 3-2。

代码清单 3-18 发送数据包（main.c 文件）

```
1
2  #define BYTE0(dwTemp)       (*(char *)(&dwTemp))
3  #define BYTE1(dwTemp)       (*((char *)(&dwTemp) + 1))
4  #define BYTE2(dwTemp)       (*((char *)(&dwTemp) + 2))
5  #define BYTE3(dwTemp)       (*((char *)(&dwTemp) + 3))
6
7  /**
8    * @brief  控制串口发送 1 个字符
9    * @param  c:要发送的字符
10   * @retval none
11   */
12 void usart_send_char(uint8_t c)
13 {  //循环发送,直到发送完毕
14     while (USART_GetFlagStatus(DEBUG_USART,USART_FLAG_TXE)==RESET);
15     USART_SendData(DEBUG_USART,c);
16 }
17
18
19
20 /*函数功能：根据匿名最新上位机协议写的显示姿态的程序（上位机 0512 版本）
21  *具体协议说明请查看上位机软件的帮助说明。
22  */
23 void Data_Send_Status(float Pitch,float Roll,float Yaw)
24 {
25     unsigned char i=0;
26     unsigned char _cnt=0,sum = 0;
27     unsigned int _temp;
28     u8 data_to_send[50];
29
30     data_to_send[_cnt++]=0xAA;
31     data_to_send[_cnt++]=0xAA;
32     data_to_send[_cnt++]=0x01;
33     data_to_send[_cnt++]=0;
```

```
34
35      _temp = (int)(Roll*100);
36      data_to_send[_cnt++]=BYTE1(_temp);
37      data_to_send[_cnt++]=BYTE0(_temp);
38      _temp = 0-(int)(Pitch*100);
39      data_to_send[_cnt++]=BYTE1(_temp);
40      data_to_send[_cnt++]=BYTE0(_temp);
41      _temp = (int)(Yaw*100);
42      data_to_send[_cnt++]=BYTE1(_temp);
43      data_to_send[_cnt++]=BYTE0(_temp);
44      _temp = 0;
45      data_to_send[_cnt++]=BYTE3(_temp);
46      data_to_send[_cnt++]=BYTE2(_temp);
47      data_to_send[_cnt++]=BYTE1(_temp);
48      data_to_send[_cnt++]=BYTE0(_temp);
49
50      data_to_send[_cnt++]=0xA0;
51
52      data_to_send[3] = _cnt-4;
53      //和校验
54      for (i=0; i<_cnt; i++)
55          sum+= data_to_send[i];
56      data_to_send[_cnt++]=sum;
57
58      //串口发送数据
59      for (i=0; i<_cnt; i++)
60          usart_send_char(data_to_send[i]);
61  }
62
63  /*函数功能：根据匿名最新上位机协议写的显示传感器数据（上位机 0512 版本）
64   *具体协议说明请查看上位机软件的帮助说明。
65   */
66  void Send_Data(int16_t *Gyro,int16_t *Accel)
67  {
68      unsigned char i=0;
69      unsigned char _cnt=0,sum = 0;
70  //  unsigned int _temp;
71      u8 data_to_send[50];
72
73      data_to_send[_cnt++]=0xAA;
74      data_to_send[_cnt++]=0xAA;
75      data_to_send[_cnt++]=0x02;
76      data_to_send[_cnt++]=0;
77
78
79      data_to_send[_cnt++]=BYTE1(Accel[0]);
80      data_to_send[_cnt++]=BYTE0(Accel[0]);
81      data_to_send[_cnt++]=BYTE1(Accel[1]);
82      data_to_send[_cnt++]=BYTE0(Accel[1]);
83      data_to_send[_cnt++]=BYTE1(Accel[2]);
84      data_to_send[_cnt++]=BYTE0(Accel[2]);
85
86      data_to_send[_cnt++]=BYTE1(Gyro[0]);
87      data_to_send[_cnt++]=BYTE0(Gyro[0]);
88      data_to_send[_cnt++]=BYTE1(Gyro[1]);
89      data_to_send[_cnt++]=BYTE0(Gyro[1]);
90      data_to_send[_cnt++]=BYTE1(Gyro[2]);
91      data_to_send[_cnt++]=BYTE0(Gyro[2]);
92      data_to_send[_cnt++]=0;
93      data_to_send[_cnt++]=0;
94      data_to_send[_cnt++]=0;
95      data_to_send[_cnt++]=0;
96      data_to_send[_cnt++]=0;
97      data_to_send[_cnt++]=0;
98
99      data_to_send[3] = _cnt-4;
100      //和校验
```

```
101     for (i=0; i<_cnt; i++)
102         sum+= data_to_send[i];
103     data_to_send[_cnt++]=sum;
104
105     //串口发送数据
106     for (i=0; i<_cnt; i++)
107         usart_send_char(data_to_send[i]);
108 }
```

函数比较简单，就是根据输入的内容，一字节一字节地按格式封装好，然后调用串口发送到上位机。

发送数据

与上一小节一样，我们使用 read_from_mpl 函数输出数据，由于使用了不同的上位机，所以我们修改了它的具体内容，见代码清单 3-3。

代码清单 3-19 read_from_mpl 函数(main.c 文件)

```
1
2  extern struct inv_sensor_cal_t sensors;
3
4  /* 从 MPL 中获取数据.
7   */
8  static void read_from_mpl(void)
9  {
10     float Pitch,Roll,Yaw;
11     int8_t accuracy;
12     unsigned long timestamp;
13     long  data[3];
14     /*获取欧拉角*/
15     inv_get_sensor_type_euler(data, &accuracy,(inv_time_t*)&timestamp);
16
17     //inv_get_sensor_type_euler 读出的数据是 Q16 格式，所以左移 16 位.
18     Pitch =data[0]*1.0/(1<<16) ;
19     Roll = data[1]*1.0/(1<<16);
20     Yaw = data[2]*1.0/(1<<16);
21
22     /*向匿名上位机发送姿态*/
23     Data_Send_Status(Pitch,Roll,Yaw);
24     /*向匿名上位机发送原始数据*/
25      Send_Data((int16_t *)&sensors.gyro.raw,(int16_t *)&sensors.accel.raw);
26 }
```

代码中调用 inv_get_sensor_type_euler 获取欧拉角，然后调用 Data_Send_Status 格式上传到上位机，而加速度及角速度的原始数据直接从 sensors 结构体变量即可获取，获取后调用 Send_Data 发送出去。

3.4.3 下载验证及匿名飞控地面站的使用

直接下载本程序到开发板，在液晶屏上会观察到姿态角、温度、计步器数据，改变开发板的姿态，数据会更新(计步器数据要模拟走路才会更新)，若直接连接串口调试助手，会接收到一系列的乱码信息，这是正常的，这些数据需要使用“匿名飞控地面站”上位机解码。

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。

实验步骤如下：

(1) 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。

(2) 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。

(3) 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。

图 3-4 上位机配置

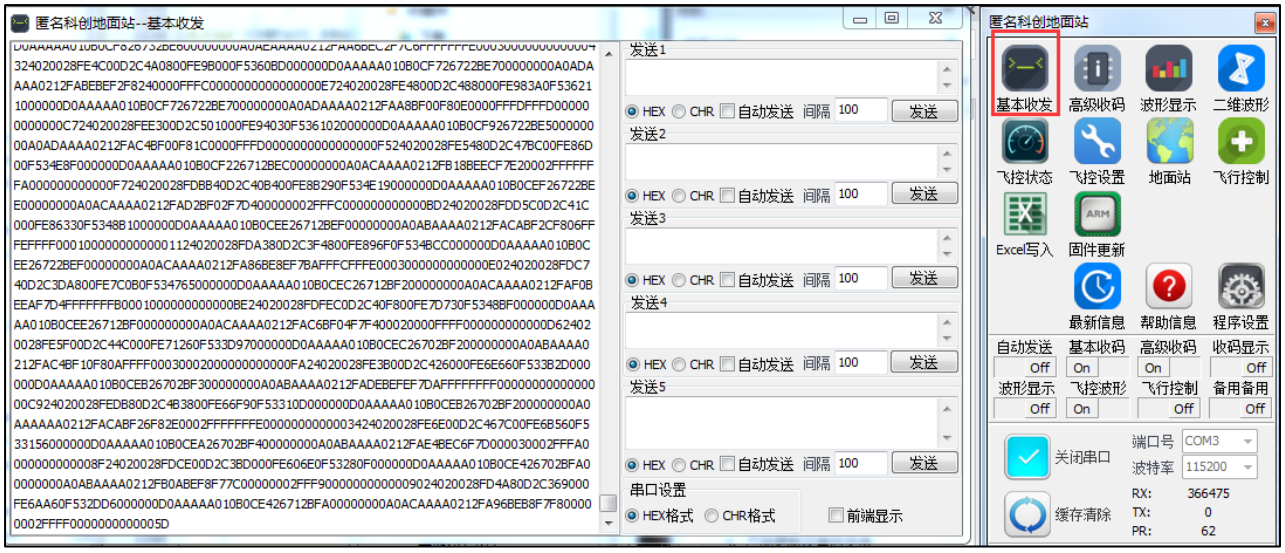

图 3-5 基本收发

图 3-6 波形显示

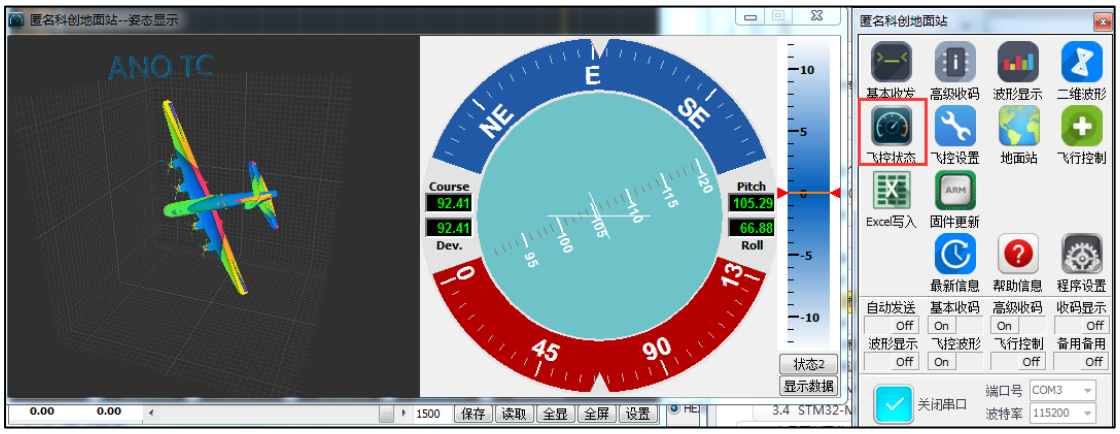

图 3-7 姿态显示

附录

姿态检测的基本概念

MPU6050 陀螺仪一般用于设备的姿态检测，它涉及各领域的知识非常多，感兴趣的可以查阅配套资料中“第三方资料”文件夹的内容来学习。下文附录也介绍了一些关于姿态检测的基本概念。

1. 基本认识

在飞行器中，飞行姿态是非常重要的参数，见图 0-1，以飞机自身的中心建立坐标系，当飞机绕坐标轴旋转的时候，会分别影响偏航角、横滚角及俯仰角。

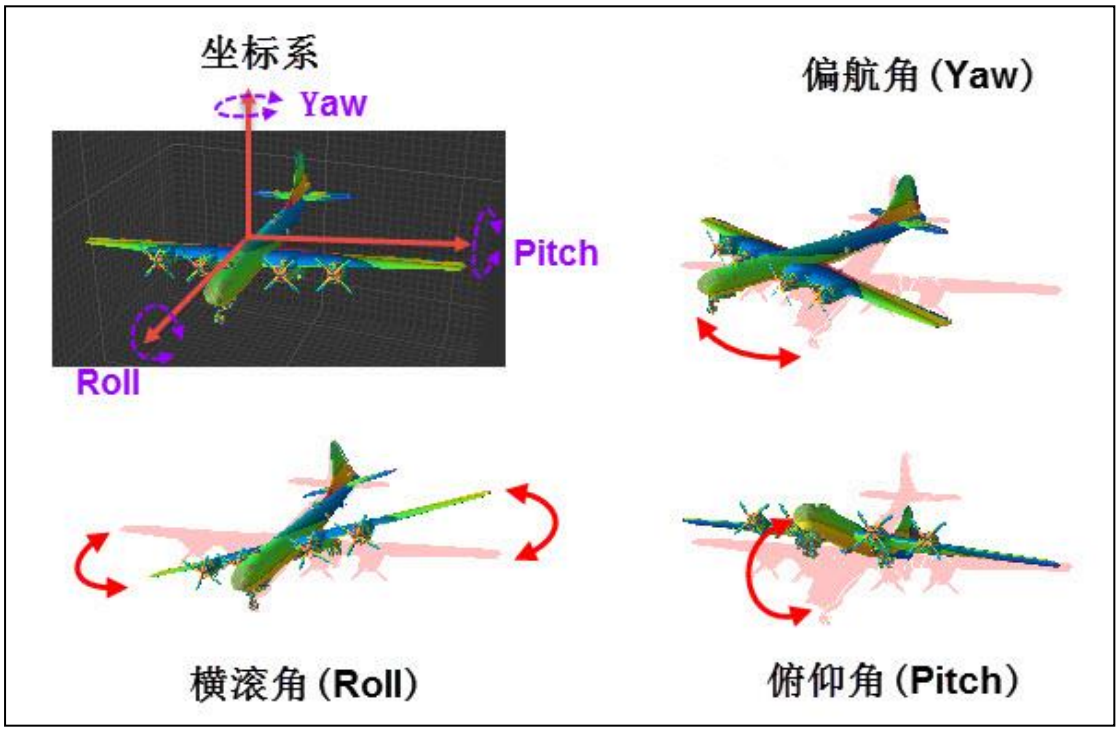

图 0-1 表示飞机姿态的偏航角、横滚角及俯仰角

假如我们知道飞机初始时是左上角的状态，只要想办法测量出基于原始状态的三个姿态角的变化量，再进行叠加，就可以获知它的实时姿态了。

2. 坐标系

抽象来说，姿态是“载体坐标系”与“地理坐标系”之间的转换关系。

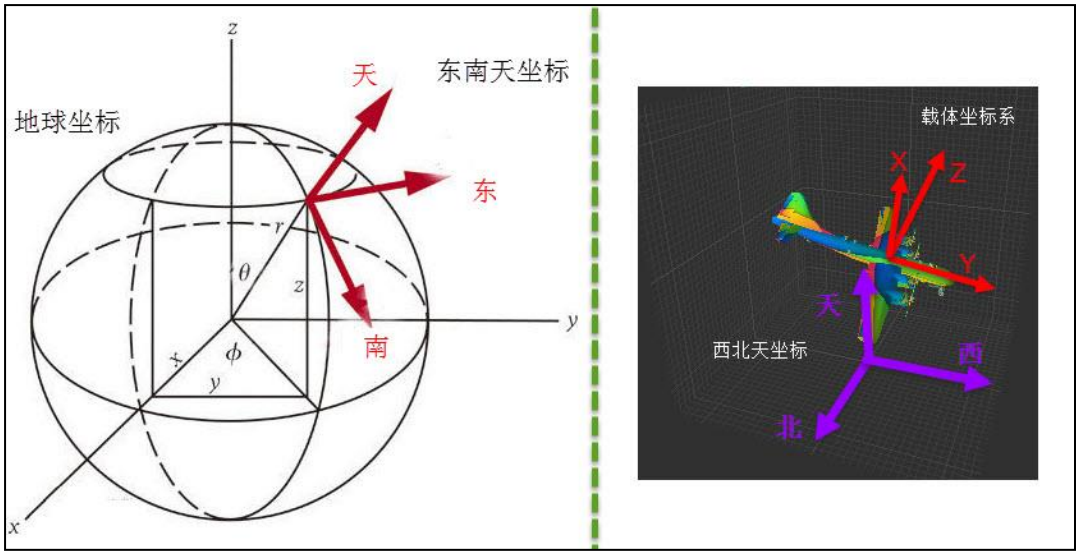

图 0-2 地球坐标系、地理坐标系与载体坐标系

我们先来了解三种常用的坐标系：

- 地球坐标系：以地球球心为原点，Z 轴沿地球自转轴方向，X、Y 轴在赤道平面内的坐标系。
- 地理坐标系：它的原点在地球表面(或运载体所在的点)，Z 轴沿当地地理垂线的方向(重力加速度方向)，XY 轴沿当地经纬线的切线方向。根据各个轴方向的不同，可选为“东北天”、“东南天”、“西北天”等坐标系。这是我们日常生活中使用的坐标系，平时说的东南西北方向与这个坐标系东南西北的概念一致。
- 载体坐标系：载体坐标系以运载体的质心为原点，一般根据运载体自身结构方向构成坐标系，如 Z 轴上由原点指向载体顶部，Y 轴指向载体头部，X 轴沿载体两侧方向。上面说基于飞机建立的坐标系就是一种载体坐标系，可类比到汽车、舰船、人体、动物或手机等各种物体。

地理坐标系与载体坐标系都以载体为原点，所以它们可以经过简单的旋转进行转换，载体的姿态角就是根据载体坐标系与地理坐标系的夹角来确定的。配合图 0-1，发挥您的空间想象力，假设初始状态中，飞机的 Z 轴、X 轴及 Y 轴分别与地理坐标系的天轴、北轴、东轴平行。如当飞机绕自身的“Z”轴旋转，它会使自身的“Y”轴方向与地理坐标系的“南北”方向偏离一定角度，该角度就称为偏航角(Yaw)；当载体绕自身的“X”轴旋转，它会使自身的“Z”轴方向与地理坐标系的“天地”方向偏离一定角度，该角度称为俯仰角(Pitch)；当载体绕自身的“Y”轴旋转，它会使自身的“X”轴方向与地理坐标系的“东西”方向偏离一定角度，该角度称为横滚角。

表 0-1 姿态角的关系

坐标系间的旋转角度	说明	载体自身旋转
偏航角(Yaw)	Y 轴与北轴的夹角	绕载体 Z 轴旋转可改变
俯仰角(Pitch)	Z 轴与天轴的夹角	绕载体 X 轴旋转可改变
横滚角(Roll)	X 轴与东轴的夹角	绕载体 Y 轴旋转可改变

这些角度也称欧拉角，是用于描述姿态的非常直观的角度。

3.4.4 利用陀螺仪检测角度

最直观的角度检测器就是陀螺仪了，见图 0-3，它可以检测物体绕坐标轴转动的“角速度”，如同将速度对时间积分可以求出路程一样，将角速度对时间积分就可以计算出旋转的“角度”。

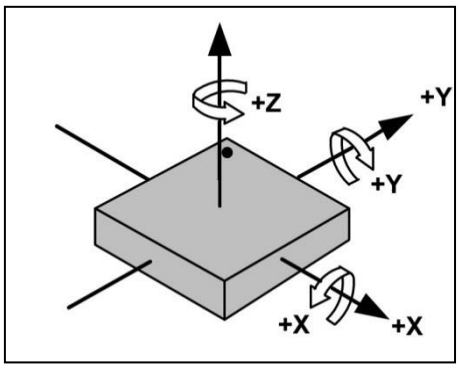

图 0-3 陀螺仪检测示意图

陀螺仪检测的缺陷

由于陀螺仪测量角度时使用积分，会存在积分误差，见图 0-4，若积分时间 Dt 越小，误差就越小。这十分容易理解，例如计算路程时，假设行车时间为 1 小时，我们随机选择行车过程某个时刻的速度 Vt 乘以 1 小时，求出的路程误差是极大的，因为行车的过程中并不是每个时刻都等于该时刻速度的，如果我们每 5 分钟检测一次车速，可得到 Vt1、Vt2、Vt3-Vt12 这 12 个时刻的车速，对各个时刻的速度乘以时间间隔(5 分钟)，并对这 12 个结果求和，就可得出一个相对精确的行车路程了，不断提高采样频率，就可以使积分时间 Dt 变小，降低误差。

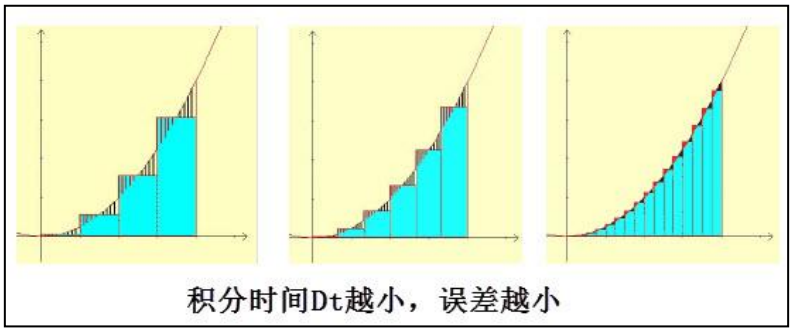

图 0-4 积分误差

同样地，提高陀螺仪传感器的采样频率，即可减少积分误差，目前非常普通的陀螺仪传感器的采样频率都可以达到 8KHz，已能满足大部分应用的精度要求。

更难以解决的是器件本身误差带来的问题。例如，某种陀螺仪的误差是 0.1 度/秒，当陀螺仪静止不动时，理想的角速度应为 0，无论它静止多久，对它进行积分测量得的旋转角度都是 0，这是理想的状态；而由于存在 0.1 度/秒的误差，当陀螺仪静止不动时，它采

样得的角速度一直为 0.1 度/秒，若静止了 1 分钟，对它进行积分测量得的旋转角度为 6 度，若静止了 1 小时，陀螺仪进行积分测量得的旋转角度就是 360 度，即转过了一整圈，这就变得无法忍受了。只有当正方向误差和负方向误差能正好互相抵消的时候，才能消除这种累计误差。

3.4.5 利用加速度计检测角度

由于直接用陀螺仪测量角度在长时间测量时会产生累计误差，因而我们又引入了检测倾角的传感器。

图 0-5 T 字型水平仪

测量倾角最常见的例子是建筑中使用的水平仪，在重力的影响下，水平仪内的气泡能大致反映水柱所在直线与重力方向的夹角关系，利用图 0-5 中的 T 字型水平仪，可以检测出图 0-1 中说明的横滚角与俯仰角，但是偏航角是无法以这样的方式检测的。

在电子设备中，一般使用加速度传感器来检测倾角，它通过检测器件在各个方向的形变情况而采样得到受力数据，根据 F=ma 转换，传感器直接输出加速度数据，因而被称为加速度传感器。由于地球存在重力场，所以重力在任何时刻都会作用于传感器，当传感器静止的时候(实际上加速度为 0)，传感器会在该方向检测出加速度 g，不能认为重力方向测出的加速度为 g，就表示传感器在该方向作加速度为 g 的运动。

当传感器的姿态不同时，它在自身各个坐标轴检测到的重力加速度是不一样的，利用各方向的测量结果，根据力的分解原理，可求出各个坐标轴与重力之间的夹角，见图 0-6。

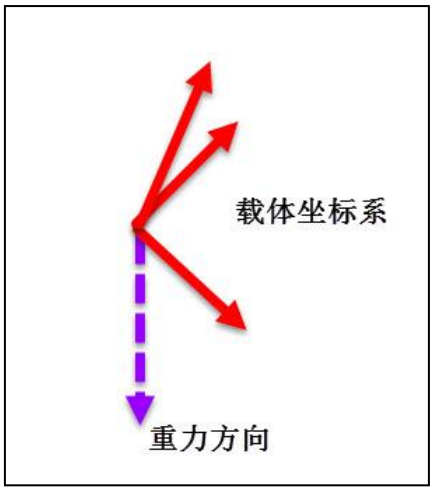

图 0-6 重力检测

因为重力方向是与地理坐标系的“天地”轴固连的，所以通过测量载体坐标系各轴与重力方向的夹角即可求得它与地理坐标系的角度旋转关系，从而获知载体姿态。

加速度传感器检测的缺陷

由于这种倾角检测方式是利用重力进行检测的，它无法检测到偏航角(Yaw)，原理跟 T 字型水平仪一样，无论如何设计水平仪，水泡都无法指示这样的角度。

另一个缺陷是加速度传感器并不会区分重力加速度与外力加速度，当物体运动的时候，它也会在运动的方向检测出加速度，特别在震动的状态下，传感器的数据会有非常大的数据变化，此时难以反应重力的实际值。

3.4.6 利用磁场检测角度

为了弥补加速度传感器无法检测偏航角(Yaw)的问题，我们再引入磁场检测传感器，它可以检测出各个方向上的磁场大小，通过检测地球磁场，它可实现指南针的功能，所以也被称为电子罗盘。由于地磁场与地理坐标系的“南北”轴固联，利用磁场检测传感器的指南针功能，就可以测量出偏航角(Yaw)了。

磁场检测器的缺陷

与指南针的缺陷一样，使用磁场传感器会受到外部磁场干扰，如载体本身的电磁场干扰，不同地理环境的磁铁矿干扰等等。

3.4.7 利用 GPS 检测角度

使用 GPS 可以直接检测出载体在地球上的坐标，假如载体在某时刻测得坐标为 A，另一时刻测得坐标为 B，利用两个坐标即可求出它的航向，即可以确定偏航角，且不受磁场的影响，但这种检测方式只有当载体产生大范围位移的时候才有效(GPS 民用精度大概为 10 米级)。

3.4.8 姿态融合与四元数

可以发现，使用陀螺仪检测角度时，在静止状态下存在缺陷，且受时间影响，而加速度传感器检测角度时，在运动状态下存在缺陷，且不受时间影响，刚好互补。假如我们同时使用这两种传感器，并设计一个滤波算法，当物体处于静止状态时，增大加速度数据的权重，当物体处于运动状时，增大陀螺仪数据的权重，从而获得更准确的姿态数据。同理，检测偏航角，当载体在静止状态时，可增大磁场检测器数据的权重，当载体在运动状态时，增大陀螺仪和 GPS 检测数据的权重。这些采用多种传感器数据来检测姿态的处理算法被称为姿态融合。

在姿态融合解算的时候常常使用“四元数”来表示姿态，它由三个实数及一个虚数组成，因而被称之为四元数。使用四元数表示姿态并不直观，但因为使用欧拉角(即前面说的偏航角、横滚角及俯仰角)表示姿态的时候会有“万向节死锁”问题，且运算比较复杂，所以一般在数据处理的时候会使用四元数，处理完毕后再把四元数转换成欧拉角。在这里我们只要了解四元数是姿态的另一种表示方式即可，感兴趣的话可自行查阅相关资料。

产品更新及售后支持

秉火的产品资料更新会第一时间发布到论坛：http://www.firebbs.cn
购买秉火产品请到秉火官方淘宝店铺：http://fire-stm32.taobao.com
在学习或使用秉火产品时遇到问题可在论坛发帖子与我们交流。