hmc5883 电子罗盘

在 GY85 中使用的是 Honeywell 公司的 HMC5883L，它是一个三轴的数码电子罗盘。该芯片被广泛用于数码指南针，用于检测与地磁北极的偏转角度。

将采集到的数据乘以数字分辨率即可得到以高斯为单位的数据：

```
data[0] *= 0.92;
data[1] *= 0.92;
data[2] *= 0.92;123
```

NOTE: 关于分辨率等参数的详细信息，请参考其数据手册。

转化为弧度(rad)：

```
Yaw = atan2(Y, X);1
```

由于存在磁偏角，所以当值为负时，需要进行校正：

```
if(Yaw < 0)
    Yaw += 2*PI;12
```

当存在正磁偏角时，同样需要校正：

```
if(Yaw > 2*PI)
    Yaw -= 2*PI;12
```

最后，将结果转化为角度(°)：

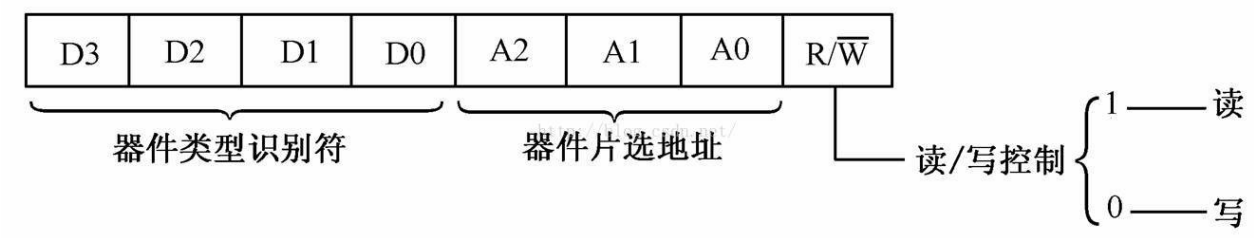
```
Yaw in ° = Yaw * 180/PI;
```

//==//

1、HMC5883L 通过 IIC 总线与主机进行通信，其 IIC 控制字节地址如下

7-bit 地址	0x1E	0001 1110
8-bit 读取地址	0x3D	0011 1101
8-bit 写入地址	0x3C	0011 1100

可以发现，HMC5883L 的 IIC 控制字节地址与 IIC 总线控制字节地址是一致的。其中高 4 位为器件类型标识符，接着 3 位为片选，最低位为读写控制位（1 为读操作，0 为写操作）。



HMC5883L IIC 接口特性说明：

HMC5883L 作为一个 IIC 兼容装置，该装置包含一个 7-bit 串行地址，并支持 IIC 协议。HMC5883L 可以支持标准和快速模式，分别为 100kHz 和 400kHz，但不支持高速模式（Hs）。

要求主机的活动（寄存器读取和写入）优先于内部活动，例如测量，这一优先次序的安排是为了不让主机等待，同时 IIC 总线占用的事件比必须的时间长。

2、HMC5883L 一些引脚说明

VCC	2.16V-3.6V
GND	
SCL	串行时钟——IIC 总线主/从时钟
SDA	串行数据——IIC 总线主/从数据
DRDY	数据准备，中断引脚，内部被拉高，选项为连接，当数据位于输出寄存器上时会在低电位上停 250us

3、HMC5883L 寄存器说明

地址	名称	访问	默认值
00	配置寄存器 A（Configuration Register A）	读/写	1111 0000（CRA）
01	配置寄存器 B（Configuration Register B）	读/写	0010 0000
02	模式寄存器（Mode Register）	读/写	1000 0001（默认为单一测量模
03	数据输出 X MSB 寄存器（Data Output X MSB Register）	读	0000 0000
04	数据输出 X LSB 寄存器（Data Output X LSB Register）	读	0000 0000
05	数据输出 Z MSB 寄存器（Data Output Z MSB Register）	读	0000 0000
06	数据输出 Z LSB 寄存器（Data Output Z LSB Register）	读	0000 0000
07	数据输出 Y MSB 寄存器（Data Output Y MSB Register）	读	0000 0000
08	数据输出 Y LSB 寄存器（Data Output Y LSB Register）	读	0000 0000
09	状态寄存器（Status Register）	读	0000 0000
10	识别寄存器 A（Identification Register A）	读	0100 1000（ASCII 值为 H）
11	识别寄存器 B（Identification Register B）	读	0011 0100（ASCII 值为 4）
12	识别寄存器 C（Identification Register C）	读	0011 0011（ASCII 值为 3）

4、HMC5883L 读写说明

HMC5883L 利用地址指针来说明读取或者写入的寄存器位置。这些指针由主机发往从机，并且跟在 7-bit 地址和 1-bit 读/写控制位之后。

下面以读数据为例说明 HMC5883L IIC 读取数据时应该进行的操作时序。

1、发送从器件控制字节 0x3C（写入操作）。注意，此操作之后主机再往从机发送的数据就是主机写入从机的数据

2、发送数据 3（由器件手册可知，3 为数据输出 X MSB 寄存器的地址）。此操作也就是写想要读取数据的寄存器地址。

3、发送从器件控制字节 0x3D（读取操作）。注意，此操作之后从机便向主机发送步骤 2 所发送的地址处的数据。

4、将从器件发送过来的数据写入相应的数组。

关于 HMC5883L 读数据需要作如下说明。

HMC5883L 为了尽可能减少主机与从机之间的通信，地址指针会在不需要主机干涉的情况下自动更新。这种自动更新的地址更新有两种附加的特性。

1、当访问 12 或者更高的地址时，指针会自动更新到 00。也就是返回首寄存器的地址。

2、当指针达到 08 时，指针回滚到地址 03。也就是说此时指针一直在 6 个数据寄存器之间滚动，从机一直滚动向主机发送数据。

To move the address pointer to a random register location, first issue a “write” to that register location with no data byte following the commend. For example, to move the address pointer to register 10, send 0x3C 0x0A.

为了将地址指针移动到随机的寄存器地址，首先应该发送写指令，之后再跟一个无数据字节（在这里我自己理解我所要移动到随机的寄存器的地址）。例如，要将地址指针移动到寄存器 10 ，就发送 0x3C 0x0A。

0x3C 为 HMC5883L 的写控制字节，告诉从机下一个发送的字节为写入到从机的字节。

0x0A 为 HMC5883L 的寄存器 10 的地址，此时地址指针就指向寄存器 10 了。如果要进行读操作，则发送 HMC5883L 的读控制字节 0x3D，然后从机将会发送相应的寄存器中的数据返回至主机。如果要进行写操作，则发送 HMC5883L 的写控制字节 0x3C，然后再发送的数据就会写入当前指针指向的寄存器。

//===//

二、配置 HMC5883L 模块

```
void HMC5883L_Init()
{
    _iic_Start();
    _iic_SendByte(0x3c );  //写操作
     _iic_SendByte( 0x00 );  //指针指向 00，配置寄存器 A
     _iic_SendByte( 0x78 );   //数据测量、输出速率 75hz
     _iic_Start();             //指针定位到 02，模式寄存器
     _iic_SendByte( 0x3c);
    _iic_SendByte(0x02);
    _iic_SendByte(0x00 );    //连续测量模式
```

```
    _iic_Stop();
 }
```

三、读取角度数据

接收三轴数据，处理 X,Y 轴的数据并计算角度：

```
int16_t HMC5883L_ReadAngle()
{
static uint8_t i;
static  uint8_t XYZ_Data[ 6 ];  //用来存储三个轴输出的数字量


_iic_Start();
_iic_SendByte(0x3c );  // 发送 HMC5883L 的器件地址 0x3c，写操作
 _iic_SendByte( 0x03 );  //指针指向 03，X msb 寄存器
_iic_Start();
_iic_SendByte(0x3d );  //改为读操作

//依次读取三个轴的数字量
for (i= 0 ;i< 5 ;i++)           //前 5 次读取发送应答信号
{
XYZ_Data[i] =_iic_ReadByte( 1);
}
XYZ_Data[5 ] =_iic_ReadByte( 0 );    //不应答
_iic_Stop();
return  atan2( ( double )((int16_t)((XYZ_Data[Ymsb]<< 8 )+XYZ_Data[Ylsb]) ),( double )
((int16_t)((XYZ_Data[Xmsb]<< 8 )+XYZ_Data[Xlsb])))*( 180 / 3.14159265 )+ 180 ;          //计
算角度，需要包含 math.h 头文件
}
```

//==//

四旋翼飞控项目日志

作者：WZY

前言

大四下学期完成主体程序，大四毕业暑假加入操作系统，希望大家共同学习，共同进步。

2013.8

时间 2013 年 2 月 2 日 11:57:37

项目进展了这么久第一次写日志，总结一下这一段时间的经验。首先，我通过李想的 stm32 教学视频对 stm32 进行了简单的了解，发现 stm32 用起来比 51 都方便，原来一直被“arm 难学”误导了，单片机只是工具，真正编算法才是难点，所以以后在做项目时不应该过分考虑单片机是否好用这个问题，只要适合项目的单片机就应该拿来用。

李想视频里编译环境是 MDK，所以在编程时一直使用 MDK 编译，感觉和学 51 时用的 keil 一模一样，比较好上手。由于项目所要做的第一步是要做姿态解算，所以在初步了解 stm32 原理以后，就开始从 MPU6050 传感器读数据。MPU6050 是一个集成了加速度计和陀螺仪的传感器，使用 I2C 和单片机进行通信，总体来说读取数据还是比较简单的。但是也遇到一些比较烦人的问题，卡了好久。一个是变量类型搞错了

这是一段读取 MPU6050 陀螺仪 x 轴数据的程序，先通过 I2C 函数把陀螺仪 X 轴数据的低 8 位和高 8 位读取出来，分别放入 BUF[0]和 BUF[1]中，在通过移位把两个数据合成一个数据传入 G_X 变量，G_X 是一个 short 型变量，而 imu_measure.gx 是一个 double 型变量，编译器编译是自动把 G_X 转成 double 型除以 16.4 赋值给 imu_measure.gx。曾经犯过一个错误就是把 BUF 数组数据相合后直接除以 16.4 然后赋值给 imu_measure.gx，结果通过串口传回来的数据就出现了错误。以后要避免这种错误。

在新添加文件后要记着把文件添加进工程，否则编译会出错。
至于为什么除以 16.4，参考《MPU-6050 寄存器映射》第 32 页

在 MPU6050 初始化时把 FullScale Range 设置为±2000°/s，所以灵敏度为 16.4 LSB/°/s，
所以在读出寄存器值的基础上除以 16.4 就是角速度。
而对于加速度，设置为±8g，灵敏度为 4096，所以在读出值的基础上除以 4096 就是加速度，单位为 g，再乘以 9.8 单位就是 m/s^2，所以 8192/9.8=835.066，除以这个值就是加速度（m/s^2）。同理，在理解 MPU6050 初始化的设置时，参考这本手册即可。

要设置为+-4g 即 0000100=0x08
还有一个就是 extern 变量的使用。正确的用法是在初始化的地方定义（函数外定义，如果在函数内则会是局部变量），然后在 External_Variable.h 文件中用 extern 关键字再定义一遍（注意，这里不要赋值），告诉编译器，这个变量在其他地方已经定义过，直接使用即可。然后需要用到全局变量的文件开头进行#Include “External_Variable.h”，就可以了。
MPU6050 传感器初始化时，使用 I2C 总线函数即可，例如：
Single_Write(MPU6050_Addr,PWR_MGMT_1, 0x00);　　//解除休眠状态
分别写入 MPU6050 地址，寄存器地址，和写入的值即可。出现一个问题就是 MPU6050 的 I2C 地址在手册上写的是 7 位固定，为 0x68（1101000），但有的例程上给的是直接写入 0x68，实际在使用过程中写入 0xD0（11010000）才有反应，即 7 为固定位加一位硬件设置的地址位（BIT0=0）。
在 MPU6050 初始化后就可以不停的读取传感器值进行姿态解算了。在姿态解算前，我先对四元数进行了初始化，即飞控板在静止状态下通过加速度计解算欧拉角，再通过四元数公式计算（即函数 quat_init()）。其实感觉这里不初始化也可以，因为姿态解算速度很快，第一次的值对以后影响并不大。

随后就开始真正的姿态解算了，首先要对读取的传感器值进行移动均值滤波。定义了 Filter_ACC 结构体，先把读取值加到这个结构体变量里，每加一次，积分标记位加 1。当定时器中断触发。姿态解算周期到来时，进入 if(IMUupdate_flag == TRUE){}里面求平均值并对变量清零，完成移动均值滤波。

滤波过后，就开始用 IMUupdate（）函数进行解算，这个函数是参考 http://www.amobbs.com/forum.php?mod=viewthread&tid=5492189&highlight=%E6%8D%B7%E8%81%94%E6%83%AF%E5%AF%BC
在实际编程中发现，在输入 IMUupdata()函数前如果角速度不除以一个比例系数，解算出的姿态角就会出现超调或者，但是如果系数过大就会出现调不够的现象，所以实验的几次以后找到了一个经验值。

进行使用的，基本原理是把加速度计解算出的姿态和上次解算出的姿态（认为是正确的）进行向量叉乘，得到两个向量的差，然后把这个差乘以比例系数和角速度数据融合得到认为是正确的角速度里，将正确的角速度代入到龙格库塔方程里推算出当前正确的姿态。方程里所需的半周期 halfT 是由 TIM2 中断产生的（TIM2 每 2ms 中断一次，即 T=2ms，halfT=1ms）。在这个过程里还引入里积分环节来消除静差。

比如陀螺 X 轴读取时始终有个 -32 °/s 的误差存在，在姿态解算后，滚转角始终不是平的，会有一个误差，但是会发现这个误差会逐渐减小，这就是积分环节在起作用。
其实在一开始，通过上位机看 stm32 姿态解算的状态并不是很好，最后发现是 uart1 传输函数里面有个 Delayms（1），晕死，本来对速度的要求就很高，结果每传输一次就延时 1ms，不出问题才怪。结果吧这个延时函数去掉以后干脆不出数据了，卡了好久，发现是自己思维进入死胡同了！忘记串口传输完毕是要等待传输结束的：

结果加上这句话后形式一片大好~顺利的解算出了姿态，而且效果很好。

这里使用的上位机软件是阿莫论坛上一位大虾提供的，很好用，不足的地方就是传送速率过快会卡，不能迅速显示，不过功能还是挺强大的，可以数据保存截图什么的。
下一步的任务就是使用 HMC5883 对偏航角进行修正，由于加速度计不能读出偏航角的数据，所以偏航角由于陀螺仪误差的缘故会一直向一个方向偏。再然后就是读取气压高度。

2013 年 2 月 3 日 23:14:52

今天主要解决的是全姿态罗盘。说来很惭愧，大部分时间卡在一个很细小的问题上，就是 HMC5883L 的 I2C 总线地址上。一开始翻看 MWC 的程序，磁阻仪的地址上 0x1E，于是用 I2C 向磁阻仪写入 0x1E，结果一点反应都木有啊~试了各种方法，一度以为是 MPU6050 的 I2C 辅助位没有设置好（其实设置是没问题的，让 MPU6050 的 I2C_Master 功能 disable，ByPass 功能打开，主芯片可以直接控制 HMC5883L，这两个寄存器分别在 0x37 和 0x6A 两个寄存器里），在这里设置了好久。后来突然想到上篇日志里写的地址移位的问题，查看 MWC 飞控的 I2C 写入函数，果然地址进去以后先要左移一位，顿时泪奔，0x1E 左移一位是 0x3C，按这个地址写入瞬间各种正常~
但是在磁阻仪数据读出后，立马又发现了一个问题。飞控板转动 90°，上位机显示实际的转动是小于 90°的。然后就各种逛论坛啊，各种没思路。于是决定先睡一会儿，困死了，结果刚躺下立马就有思路了，果然过于专注一个问题思维容易进入死胡同。其实磁阻仪的每个轴的数据并不是关于 0 对称的，这是一个误差。另一个误差是每个轴的变化量是不同的，以 X,Y 轴为例，如果以想两轴的数据确定一个点（X,Y），那么如果把飞控板转一圈得到的数据画出的圆其实是一个椭圆，而且和原点存在偏移。如图：

蓝色点是飞控板在 XY 平面旋转一圈采集到的数据，红色圆圈是用 matlab 拟合得到的圆。
通过 matlab 拟合，我们得到了下图数据：
其中数据从左到右分别为椭圆圆心的坐标（X,Y），然后是椭圆长半轴和短半轴的长度。
所以我们可以利用这些数据对磁阻仪进行校正。

其中椭圆圆心的偏移量分别为磁阻仪 XY 轴数据的偏移量，用原始数据减去这个偏移量即可。然后由于 Y 轴比 X 轴长，所以把 X 轴数据乘以长半轴和短半轴的商来校正 X 轴数据，这样 XY 轴数据就变成一个没有偏移的圆，通过 atan2 函数就可以求方向角了。

如图，我把飞控板旋转 90°，可以看到上位机读到的数据从 180°左右变到了 270°左右。

但是接下来又存在一个问题，就是这个方法只能在水平面内使用，如果有滚转或俯仰角，通过磁阻仪计算得到的偏航角就会有很大幅度的变化。然后又是一顿查资料，找到几篇关于全姿态罗盘的论文，参考上面的把磁阻仪三轴数据变化到平面的公式编入程序中发现，根本没有什么变化。可能是这个飞控板的传感器安装和论文里所用的安装方位不一样的缘故吧。然后又是一阵挠头，突然想到 MWC 飞控里应该有相应的程序。于是翻 MWC 的程序，结果还真找到了。

```
heading = _atan2f(EstG.V.X * EstM.V.Z -EstG.V.Z * EstM.V.X, EstG.V.Z * EstM.V.Y - EstG.V.Y * EstM.V.Z)
```

上面是 MWC 全姿态罗盘的程序，Est.V.axis 分别是重力向量在机体坐标系下在机体坐标系各轴的分量（即机体坐标系下重力向量），EstM.V.axis 同理为机体坐标系下磁场的向量。用这个坐标变换并用 atan2 函数就可算出偏航角。

按照这个思路把程序写出来并实验了一下。

下图中红色为偏航角，黄色为滚转角。可以看到当滚转角发生变化时，偏航角只发生了小幅的变动，可以看出全姿态罗盘算法起作用了。而偏航角的小幅变动我推测是由于 Z 轴没校正好的缘故。Z 轴的校正没有用 matlab 拟合，只是简单的找了最大值和最小值算偏移量（由于只是简单的手动旋转，可能没有找到真正的最大值和最小值），然后用 Z 轴数据的量程和 Y 轴数据的量程做比较得出比例系数。其实如果把三轴数据统计会得到一个椭球体，然后用 matlab 拟合得到椭球体的原点偏移和各轴的长度，相信可以得到更准确的磁阻仪校正。

至于如何用 matlab 拟合，是从论坛上找到的现成的程序，改天把它改成椭球体的拟合。数据处理是通过串口助手读取数据存到 word 里，然后将

文本准换成表格，以空格分列，然后把各列数据存到 Excel 里，最后用 matlab 读取数据。由于我的 matlab 貌似存在问题，得用主程序图标以管理员方式运行，进入后把目录设置成拟合程序的目录，然后用 XY=xlsread('file')函数读取数据。

在程序里，今天做的最大的修改就是把均值滤波去掉了。因为发现在姿态解算前积分量只积分了一次，也就是说就是当前读取传感器的值，所以干脆去掉了。而且传感器最快的读取速度也就 1KHz，姿态解算的速率是 500Hz，没有必要滤波了。如果以后装到飞机上发现噪声严重的话再滤波和降低姿态解算速率。

2013 年 2 月 4 日 22:27:33

今天做的主要一件事就是把陀螺仪和磁力计的数据进行融合。因为陀螺仪在很微小的转动时几乎察觉不不到飞控板的转动，所以需要磁力计的数据进行校正。而磁力计虽然昨天做了全姿态罗盘的变换，但是在存在俯仰和滚转角时，偏航角任然有扰动，所以又需要陀螺仪来校正，所以我要把这两个传感器的数据相互融合来取长补短。

其实一开始的想法是通过磁力计解算偏航角，然后把加计和陀螺仪解算的俯仰滚转角和磁力计解算的偏航角一并转成四元数，用于下次四元数解算。但是实际效果并不是很好。纠结了一会儿想到一个办法就是把加计和陀螺仪解算的偏航角和磁力计解算的偏航角融合，然后引入一个积分环节，每次积分的值是陀螺仪解算的偏航角和最终解算偏航角的差，用这个积分环节去校正由于加计和陀螺仪解算的偏航角的静差（这个偏航角有累积误差）。

这个方法试验了一下效果不错，但是就是存在一个问题。比如由 180°转到 360°（0°），由于当磁力计的偏航角到达 360°时，加计和陀螺仪的偏航角未必到达 360°，所以就会出现一个现象，再稍微顺时针转一下，磁力计的偏航角又变成几度，而加计和陀螺仪的偏航角还在 300 多度。这样积分环节就会把最终解算的偏航角逆时针转到磁力计的偏航角（大概几度那里）。这是一个很不好的现象，所以我在这个基础上对算法又进行

了改进。由于我们需要的是陀螺仪数据，所以我们可以直接从角速度入手，如下图：

把角速度计和磁力计数据融合，gz 为角速度数据，乘以负数是和定义的转动方向有关，我定义的是顺时针转动角度从 0°变到 360°，而 gz 数据正好相反，所以要乘负数。由于每 2ms 调用一次 IMUupdata()，所以 gz*0.002 就是 2ms 转过的角度，对此积分就是实时的角度把磁力计解算角度和上次解算的偏航角做差得到偏航角的误差，乘以一个系数 0.002（经验值）再积分用来修正角速度计的累积误差实验表明，角速度计再快速转动时误差较小，但是转动速度较慢时通过积分几乎得不到偏航角的变化，此时就需要用磁力计校正通过数据融合，即避免了角速度计的累积误差又可以用角速度计解算的角度校正滚转俯仰时磁力计产生的误差（从±15°减小到±3°）

这样就很好的避免了上述现象。效果如下：

上图中黄色是滚转角，蓝色是俯仰角（左面图例有误）。当我把飞控板做了三次滚转和三次俯仰动作，可以看到偏航角变化幅度很小。滚转时偏航角的变化是由于用手滚转时不可避免的会影响偏航角。基本上误差在±3°以内，而原来未用陀螺仪校正时可以达到±15°，所以，数据融合起到了很好的效果。

程序中变化较大的就是原来提到的陀螺数据除掉的那个系数，如下图：

这句代码被删掉了，这个系数也被直接放在 MPU6050 的宏定义里：

16.4*35=574 16.4 是陀螺仪灵敏度，35 是系数（本来应该是 57.3，即角度值转化为弧度制，但实验发现用 35 效果较好）。
原来的除以一个系数后来发现是由于我没把角度值转化成弧度制，看来以后还要细心。

然后我给 BMP085 写了驱动，其实 BMP085 一开始并不需要对其寄存器写一些值进行初始化，而是一开始从 BMP085 的 EEPROM 中读取一些值用来以后的数据校正。在使用 BMP085 时直接向寄存器写值读取即可。但是 BMP085 有个比较麻烦的地方就是，当你给寄存器写入值准备读取

后，它会有个转换时间，如果你在这里加延时等待，就会大大减缓飞控的主频率。所以我还是用了原来使用超声波计的方法，用一个标记位，先写入值准备读取然后跳过去执行其他的命令，当循环几次过来发现标记位触发然后接着读取，这样就会节约很多的时间。

上面是一段读取代码，BMP085_Read_flag 就是标记位，从 1 到 4 中间隔了三段，每段 2ms（IMUupdata 函数的调用周期），这样从 BMP085_Read_flag=1 到 BMP085_Read_flag=4 中间就隔了 6ms，而手册上写的读取时间是 4.5ms，所以可以读取。
不过问题就是，读取后的结果有点诡异。

不知道哪里出错了，等明天用普通的延时函数读取一下，看是不是这种方法不能用。

2013 年 2 月 5 日 13:08:50

OK，忙了一天继续写日志，感觉快养成一个习惯了。
首先今天上午在实验飞控的时候发现飞控在 0°的时候仍然存在当磁力计偏航角先从 0°多变到 350°多时，融合后的偏航角会从 0°多顺时针转到 350°多，也就是它绕了一个大圈去校正误差。这个问题上文提到过，但是当时以为已经解决了，现在看来还是没有，只是原来逻辑出错了。
所以我又设计了新的代码解决这个问题，那就是如果误差大于 180°就把积分值反方向加，这样就不会绕大圈校正误差了。代码如下：

把程序拷进去以后发现这个问题被很好的解决了，当我在 0°左右变化时，比如从 0°多变到 350°多，偏航角不会再绕一个大圈去校正了，效果如下：

接下来是卡了一天的 BMP085 数据读取问题，接着昨天的那个两个大气压，各种改程序啊，然后参考 MWC 的还用外部中断判断是否转换完毕，各种调不通啊。。。还读出过 5000 多 Pa 的气压，海拔-2000 米什么的。
后来灵感一来发现，我由于在把别人程序拷过来的时候嫌麻烦（别人程序和自己的定义不一样），把 uint_32 什么的强制转换直接去掉了，因为之前程序里也有好多强制转换直接忽略也没出什么问题，于是我把这些强

制转换都加上，uint_32 对应的是 unsigned long。发现数据就对了，看来调 bug 还是得靠憋灵感。。。
不过气压即使读出来发现气压计精度真的很差，静止不动的情况下从一开始的 0m 能飘到 20 多 m，无语，看来以后当普通的海拔高度计用吧。定高用更精确的气压计或者超声波什么的。

2013 年 2 月 6 日 23:12:01

今天最大的改动是先把所有的主循环任务放到一个新的函数 loop()里，以便以后添加新的任务

今天发现一个容易出错的问题就是 MDK 编程里往函数里输入数组直接输入数组名字即可，不需要加[]，加上反而会报错。

还有一个卡了好久的问题就是把滑动均值滤波函数和数据处理函数放到 ARHS 函数里上位机就没有三个角度的信息，好奇怪，调了好久也没调通，但是只要放在 READ_MPU6050()函数里就没问题，索性就放到这个函数里，反正读取数据就要连带滤波和数据处理。

今天最大的加入就是把 PWM 输出模块加进去了，虽然是直接拷贝别人的，不过大概看了一下发现也不是很难。基本原理是用 TIM4 定时器先预分频，72M 降到 24M，然后设置一个计数上限 1000，即 PWM 的频率是 24M/1000=24k Hz，然后 TIM4 有 4 个通道，每个通道有寄存器，用通道寄存器和总寄存器比对输出 PWM，有点类似 8051F 的 PCA 高速输出模式。

但是又出现一个问题，定时器的预装填值到底用不用减 1，PWM 例程里 1000 上限装填的是 999，但是更多的例程是不减 1 的，以后有示波器专门实验一下。

2013 年 2 月 8 日 21:39:33

基本上飞控已经初具雏形，现在 PWM 输出和输入都加上了，现在 PWM 输入调通了，接下来再调 PWM 输出。

在编程中发现，每个通用定时器（2~6）都有四个通道，相当于 PCA，非常强大，可以产生四路 PWM 或者检测输入的四路 PWM。不过每个定时器的端口是由限制的，芯片是定义好的。参见手册（《STM32 中文参考手册》）81 页。其中，我用 TIM2 输入 PWM，所以对应 PA0~3，用 TIM4 输出 PWM，对应 PB6~9。

而原来产生任务时钟基的定时器变为了 TIM3 任务，留出 TIM2 给 PWM 输入，其实用基本定时器 6 或 7 就可以给任务产生时基，但是还不会配置基本定时器，留给以后改变吧。这样可以把 TIM3 通用定时器留给其他功能。毕竟简单任务就应该配置简单定时器

然后我在程序中加入了滑动均值滤波器，基本原理是依次更新数组的数据，然后每次调用都把数组所有数据相加再除以数组大小。代码如下

接下来是头疼了好久的 PWM 输入的问题，程序是参考的 MWC 飞控。MWC 飞控接收遥控信号给了两种方式，一种是用一个端口检测一路 PPM 信号，另一种是用 8 个端口依次扫描每个端口的 PWM 脉宽。这里我采用了后一种，这样可以直接使用遥控接收机的信号，否则要用 PPM 信号需要从接收机里的一个引脚里引出 PPM 信号，或者用解码器，都比较麻烦。
MWC 的流程是先初始化 TIM2，用 TIM2 的四个通道进行读取。当时犯了一个错误就是 MWC 程序 TIM2 已经 GPIOA 端口的 APB 总线时钟在程序其他地方已经打开了，所以程序这里没打开时钟的代码，而我直接用人家的代码没有任何反应，后来才发现这点赶紧把 GPIO，NVIC，TIM2 的初始化代码补全，然后就正常了。
MWC 的端口扫描的基本原理是触发 TIM2 中断后进入中断，然后来一个 for 循环循环判断四次（因为有四路输入），看是哪个通道产生的中断，判断出以后用一个 switch 语句把捕捉到的高电平（或低电平）时间赋给变量 val，然后根据 state->state 标记位判断是高电位时间计数还是低电平时间计数，分别赋给 state->rise 和 state->fall。当然，要获取一个 rise 一个 fall 是两次进入中断函数完成的。然后他判断这两个计数大小，相减或者求相减的补数得到高电平时间。这是一个非常巧妙的方式。

但是一开始的时候整个过程非常占用 CPU 时间，以至于 PWM 输入功能一打开，CPU 只执行 PWM 输入功能，其他的都不执行。然后我就想找关闭 PWM 输入功能的语句。试了好久，关 GPIO 时钟，关中断向量，关 TIM2，都没用。没办法，从初始化函数开始一部分加一部分的关闭尝试，当关闭 TIM2 时钟是发现中断关闭了。这就好说了，我用任务时钟基的标记位 Control_counter == 1 时打开 TIM2 时钟，然后当 TIM2 中断执行 16 次（每个高电平至少需要两次中断读取，4 个高电平就 8 次，保守一些让中断执行 16 次）后关闭 TIM2 时钟，这样就实现了每 2ms 读取一次 PWM 输入的功能。把程序烧进去以后一切就都正常了。

上图是程序运行后通道 1 输入 PWM 后的结果，可以看到检测到了值而且在变化。我用另一个 stm32 产生了 50Hz（舵机标准信号）的 PWM，脉宽从 0%变到 90%（飞控 PWM 输入计数应该从 0 变到 18000，满值是 20000）。可以看到，图上读数从 19000 变到 1200 左右，说明 PWM 输入起作用了。由于在 0%和 100%附近 PWM 信号会失真，所以最大超过 18000 最小未达到 0 也是可以理解的，当然也有可能是没有显示出来的缘故，向上位机输出的频率是 20Hz，而扫描的频率是 500Hz。
然后我一次把 PWM 信号线插入通道 2 3 4，也读出了数据，说明四个通道都没问题。

上图是上位机读到的通道 1 的捕捉值变化，纵坐标是 200 代表 20000，可以看到 PWM 输入很好的跟随了另一个 stm32 的 PWM 输出（从 0 变到 18000 再从 18000 变到 0），但是偶尔会出现噪声。如

但是这不是最致命的，PWM 跟随期间有一段时间会出现下面的情况

噪声很严重，推测是上升沿和下降沿判断出问题了，看来需要对于这个过程多一些判断，避免出现上面的过程。

2013 年 2 月 9 日 12:44:54
读取多路 PWM 没进展中，各种噪声各种紊乱。。。。

读取函数重新改了，改得类似原来 8051F 开发的 PCA 函数了（break 语句卡了好久）。但是读取值向上偏移 2000，去学校用示波器试试，现在不知道是产生 PWM 的 stm32 缘故还是读取的缘故。
只能读出一个通道，是因为 TIM_GetCapture 函数分 TIM_GetCapture1，TIM_GetCapture2，TIM_GetCapture3，TIM_GetCapture4，四个通道。

关闭看门狗中断，提高 TIM2 中断优先级

循环变量不要用全局，污染变量。

实在不行还是用 PPM 吧

2013 年 2 月 11 日 12:19:28

PWM 输入搞不定，暂且用上升沿下降沿切换这样读取吧。。。以后再说

2013 年 2 月 11 日 22:42:50

终于完美解决 PWM 输入问题了~哈哈哈~
原来一直困扰的紊乱问题是由于 PWM 输入波形不稳有噪声，导致错误触发 TIM2 中断造成的，然后发现 stm32 的定时器中断有自带的滤波功能，我把滤波值调到最大，就再也没有紊乱了！

不过还有一个问题就是还是偶尔会有突然变成负的情况，后来才知道那不是变成负的了而是超过一定的范围（大概 30000 多），数就会变成负的，原来用 8051F 和手机串口通信的时候也遇到过这种情况，所以原来一直用和 0 比大小是没有用的。结果我把比 20000 大的都去掉就好多了。

再也没有出现负的的情况。

在分析读回来的数据时候，我发现，这些数据依次是增长的关系，之所以有负的是因为超过 30000 多他自己就会变成负的，由此我彻底明白 stm32 四个中断和计数器关系了，这四个中断相当于计数器的照相机，

计数器计到一定值触发中断通道中断就会记录下来。置位 CCRx 寄存器是不管用的，必须置位计数器 TIMx->CNT！

置位后的结果：

有一部分还是负的是因为量程超过 30000 了。
所以教训就是以后遇到问题不要光假设瞎调，要深入到问题的每一个步骤发现问题。
下面是现在的结果：

OK，明天把 PID 控制部分一写，把 PWM 输出程序再调一下就差不多了~

2013 年 2 月 12 日 14:30:01

今天我把程序改回原来的 4 通道输入了，完全没有问题！

一边姿态解算，一边 PWM 输入。

2013 年 2 月 12 日 21:08:46

OK，历经一个寒假的四旋翼飞控终于有完工的曙光了。现在已经做到除了四路 PWM 和 2.4g 无线通信没调其他都调通了，这两项留给到学校调吧。在家也没有示波器，不方便。

下面介绍一下今天做的 PID 控制。
主要参考《标准的 PID 处理例程》
http://www.amobbs.com/forum.php?mod=viewthread&tid=3231599&highlight=PID
PID 算法比较简单，就是测量值和期望值做差得到误差，然后把误差存起来得到 PrevError 和 LastError，也就是上上次的误差和上次的误差，用来做微分运算（即相邻两次误差的差），然后还有 pp->SumError，存放积分值的。然后把比例积分微分值分别乘以比例积分微分系数就得到控制量。

这个截图里的红色划线代码是把期望值和测量值做差，乘以 2000 再强制转换成 char 型，这样就把比较小的误差值转变成整型了，有利于提高运算速度。精度为 1/2000*180/PI=0.03°，这个精度已经比较高了。这整个过程相当于定点浮点运算。

在 PID 控制函数里，我先通过遥控器的值得到期望值再通过 PID 运算得到四个 PID 输出量，再用这四个输出量控制四个电机。
其实整个 PID 控制是比较简单的，这个飞控只是一个平台，将来在这个平台上将进行遗传和神经网络的验证。

上图是比较暴力的晃动飞控板时测得的数据，红黄蓝是俯仰滚转偏航角的变化。后面紫色的三角形是 PWM 输入跟随，可以看到飞控程序很好的完成了测量和计算工作。并且整个过程是加上 PID 控制的。

2013 年 2 月 13 日 20:26:16

今天主要完善了 PWM 输出功能，原来 PWM 输出之所以调不通是由于通道中断没有初始化，应该分别对四个通道进行初始化。

通过对这几个通道的设置 PWM 波就生成了，本来想用之前测 PWM 占空比的程序测一下生成的占空比，但是由于生成的 PWM 波频率是 72000000/2+1/1000=24kHz，频率太高，单片机无法测出，不过把占空比改变的程序烧进去再把输出查到 LED 灯上，LED 灯的亮度确实在变化，说明 PWM 生成程序起效了。
PWM 生成是通过 STM32 定时器输出比较的 TIM_OCMode_PWM2 模式，这里有两种模式 TIM_OCMode_PWM1、TIM_OCMode_PWM2 其实只是极性不同，就是输出的 PWM 高低电平会相反。很奇怪的是，手册上说 STM32 的定时器有 TIM_ICMode_PWM1 模式用来测 PWM 输入信号的高电平时间和周期，但是写上这个后编译器报错说没有这种模式，到网上查也没有很好的解决方案。
在设置 PWM 输出引脚时有一个问题就是如果设置成 B 端口所有引脚打开，程序就出问题了，如果打开要用的 PB6~7 就没问题，不清楚是为什么，可能是和其他地方的设置冲突了。

在做 PID 运算时出现的问题就是为了节约内存我把变量设置成 char 型，但是参数传递过程中出现问题，我想输出值传入 500，但是上位机显示只有 244，索性把变量设置成 int 型就没问题了。
在用上位机看 PID 效果时我发现俯仰角正负变化相同的角度，和姿态解算的结果不同，导致 PID 运算输出也不同。这样的原因是加速度计没有校准，由于原来对加速度计的精度要求也不高，所以一直没有校准。校准的方式和磁力计类似，不过比较简单，只是单纯的通过上位机的数据找最大值和最小值，这两个值的和除以 2 就是偏差，然后把初始数据减去这个偏差。然后测了一下各个通道的量程，发现基本上都是 8200 左右，也就是说不存在数据组成的是一个椭圆的问题，用 atan 直接计算就可以得到角度。通过校准以后，俯仰角数据就比以前准确了。原来水平放置俯仰角是 1.8°左右，现在已经变成 0.6°了。

上图是水平放置是的读数。
飞控写到这里也就 NRF24L01 模块没加上了，这个去学校加吧，因为家里没有插针，飞控板上的 SPI 模块无法使用。而且用串口也可以。

2013 年 2 月 14 日 23:05:48
试着加 24L01，结果失败了，还是去学校加吧，实在不行还是用串口模块。

2013 年 2 月 20 日 19:46:41
24L01 在开发板上调通了，换到飞控板上就不行，可能是没焊好，开学把排针焊上去试试。

2013 年 2 月 26 日 18:19:14
GPS 模块终于调通了，原来刚才数据读不出来是由于数据超出了 short 型变量，应该改成 long 型。

2013 年 3 月 2 日 14:26:25
今天用 HUB 连上飞控板，发现飞控板非常不稳定，后来直接连到 U 口上发现就正常了，可能是 HUB 在数据集合和分配的时候出问题了吧。
记着用 UART2 时关闭 PWM 输入的 3 4 通道。

遥控器通道值：
油门：1108~2080　　【2】
俯仰：1100~2062（俯->仰）【1】1603
滚转：906~1866（左->右）【0】1386
偏航：2073~1083（左->右）【3】1616

在做期望值计算的时候有个奇怪的现象就是正值时正常的，负值就不正常了，后来发现还是强制转换的事。

PWM 输出示波器一直显示异常，后来居然发现是 GND 和信号线插针搞反了！
偏航角初始化太早了，有误差。偏航目标值的角度弧度转换，搞错了。而且要把目标偏航角限定在 0~360°之间。
偏航角误差过大导致 PID 运算量过大，程序出现问题。
把延时取消 MPU6050 会不出数据。
飞控无法解锁，标记位 uchar 定义，无法超过 500；
偏航一直减少，最后发现偏航通道没有值，偏航通道输入线拔下来了！
使用电调时，不要超过 1KHz，否则不转。
50 6%到 9%
100 12%到 18%
200 28%到 36%
1000 即使 100%也不转
频率越大越平稳
程序里使用 200Hz
1260 1380
采集到的四个桨的转速
组 1:7668 7717 7582 7713 1.01　　1.02
PID：俯仰 滚转 P13 D500，稍有感觉
蓝牙模块指令：

即 AT+BAUDX（X=1~C）
2013 年 3 月 7 日 17:41:50
结构体里定义不能赋值

神经网络加上去以后调节速度特别慢，怀疑是输出太小了，以误差 15°为例。归一化后为 15/180=0.08，然后用 matlab 仿真

一个 0.08 的阶跃信号，控制输出最大为 80 多，显然太小了，PWM 输出量程是 10000，所以把输出放大系数调成 40000，改天再试试。

2013 年 8 月 8 日 16:22:07
TIM1 的中断叫 TIM1_UP_IRQHandler，这样写 TIM1 才能正常使用，和通用的不一样，通用没有中间的 UP。
stm32103CBT6 只有 TIM1,2,3,4,没有 5，具体有哪些要看 stm8 选型手册。
记得换 IIC 口买的飞控是 GPIOB67
2013 年 8 月 11 日 15:56:18
飞控老跑飞，后来发现 PWM 输入中断的抢占优先级和任务时钟信号中断的抢占优先级一样，降低 PWM 输入优先级后解决此问题。
2013 年 8 月 14 日 17:45:29
互补滤波里的四元数要先初始化或者赋予随机数，要不永远是 0。而且四元数要是全局变量。
系统节拍必须小于最小延时。
不用遥控器一定要把遥控器输入计算目标角的语句关掉！
//==

我的今天调好了，yaw 也不会飘了，我之前的问题在于陀螺仪数据没有做单位换算，四元数参加运算的陀螺仪数据需要是弧度/s，之前一直用的度/s！Yaw 会飘的原因是你的 gz 一直在抖吧！

//==//

写这个帖子的目的是分享一下我的经验，没有太多的理论，一切从实际操作出发，写出我对四元数和欧拉角的理解，供比我还新的新手参考，如有错误，请指正，避免误导新人。

我用的算法是 Madgwick 写的 AHRSUpdate 和 IMUUpdate，简单有效，其中 AHRSUpdate 是融合了陀螺仪、加速度计和磁力计，而 IMUUpdate 只融合了陀螺仪和加速度计，但对于做四轴飞行器来说，IMUUpdate 也够用了，不过我没发现有用 AHRSUpdate 的，并且我的一大半的时间都是花在 AHRSUpdate 这个算法上，直到现在也没用这个算法解算出来正确的四元数和欧拉角，什么方法都试过了，导致我现在都开始怀疑这个算法的正确性，希望用过的高手来指点迷径。

做四轴飞行器也是为了好玩，目前我只完成了第一步：姿态融合。接下来才是更重要的，比如上各种什么机架、电调、电机、螺旋桨之类的，然后写各种 PID 控制代码，系统整合以后还要调试各种参数，抗干扰，使系统稳定，最后还可能要加

各种应用器件。

我现在用的算法：先用 IMUUpdate 算出 Pitch 和 Roll，然后再结合互补滤波求出 Yaw，其中求 Yaw 的公式参考自 john800422。

第一部分：硬件
1.传感器：MPU9150（INVENSENSE 公司的，单芯片内集成了加速度计、陀螺仪和磁力计，并且内置 DMP 用于姿态融合，不过只融合了加速度计和陀螺仪，具有自校准功能，价格比 MPU6050 贵很多，但是省 PCB 面积，省事，轴向重合度高。实际上就是把 MPU6050 和磁力计 AK8975 放在同一个芯片里，程序还是使用 MPU6050 的驱动）；
2.MCU：GD32F103CB（Gigadevice 公司的，支持国产，与 ST 同型号的 MCU 直接兼容，性价比更高，外接 8M 晶振，晶振远离传感器，避免干扰磁力计）
3.电源芯片：TLV70233DBVR（TI 的 LDO，输入 2-6V，输出 3.3V，只需要外接 2 个 X7R 无极性陶瓷电容）
4.串口：MAX3232（方便调试）
5.USB 供电

上图：
图 1：PCB 的 3D 效果图，实物图就不传了，太丑，跟别人没法比，测试版，先追求调通得出姿态角：

第二部分：软件
1.使用 keil，uvision4.1.0，工具链：RealView MDK-ARM Version4.12；
2.库文件：ST 官方库 STM32F10x_StdPeriph_Driver version V3.3.0；
3.驱动：官方的 MPU6050 驱动 inv_mpu.c 和 inv_mpu_dmp_motion_driver.c；

先看几个图，然后再说坐标轴的设定和算法部分。

上图：
图 1：系统初始化，顺序从上到下依次是：初始化 MPU、设置需要使用哪些传感器、设置陀螺仪测量范围（我设的是正负 500 度/s）、设置加速度计测量范围（我设的是正负 4g）、配置 fifo、设置采样率、装载 DMP、设置陀螺仪轴向(比较重要)、使能 DMP 的一些玩意儿、设置 DMP 的 FIFO、自校准陀螺仪和加速度计、开启 DMP、开始姿态融合，见下图：

图 2：由四元数求出的最终姿态角，其中 Yaw 为航向角，表示机头偏离正北方多少度，范围-180 到+180；Pitch 为俯仰角，表示机头正方向与水平线的夹角，范围-90 到+90；Roll 为翻滚角，表示机翼与水平线的夹角，范围：-180 到+180。下图为机身水平，且机头正北偏西 37 度左右的数据：

图 3：下图为机翼水平，机头指向正北，且机头向下 25 度的数据：

图 4：下图为机头指向正北，保持水平，且机翼的右翼向下倾斜 23 度的数据：

图 5：对于 Yaw 来说，代码还不够完善，当机头从南偏东 170 度到正南，再到南偏西 170 度时，Yaw 会从-180 先到 0，然后再到+180，如图中红圈部分的数据，而不是我们想象中的-180 直接到+180，表现到实际当中就是，当机头转到正南方向时，会先回转到北，再转到南，这个情况要想办法解决。如下图：

图 6：看下欧拉角的奇异点，在奇异点处一个转动状态对应无穷多组自由度值，当物体转到这些奇异点附近，便没法求解。图中当 Pitch 为+90 度时，机体的姿态便没法控制，Roll 的轴向发生了变化。如下图：

第三部分：如何确定自己的轴向
首先，轴向的定义跟初始化四元数和最后结算的欧拉角有关，跟 IMUUpdate 四元数更新算法无关，换句话说，不管你的轴向如何定义，IMUUpdate 随便用，但是初始化四元数的公式和最后结算欧拉角的公式要做适当的改变，这个后面算法中有说。加速度计也好，陀螺仪也好，磁力计也好，他们的轴向都要满足右手定理，如下图：

再附上一段注释用于解释如何定义合理的轴向，以及如何正确旋转传感器的轴向，解释这么多其实就是说定义好的轴向要

满足右手定理，如下图：

下图，旋转前是[x y z]，旋转后就是[-y x z]：

下图是如何确定旋转角度的正方向，用右手握住坐标轴，拇指指向轴向的正方向，四个指头弯曲的方向就是旋转角度的正方向，在初始化四元数时，计算出的欧拉角的正方向也要满足这个条件：

我的程序使用的轴向如下图所示，未作任何改变：

第四部分：算法
第一步是校准，加速度计和陀螺仪我用的是 MPU9150 内部自校准，磁力计的校准采用如下方法：

第二步是初始化四元数，常见的轴向定义是绕 x 轴旋转是 Roll，绕 y 轴旋转是 Pitch，绕 z 轴旋转是 Yaw，我的程序也是这样定义的，详细说明见程序，这里举个另外一种轴向定义来对初始化四元数进行说明，方便比较。

下面我们来定义绕 x 轴旋转是 Pitch，绕 y 轴旋转是 Roll，绕 z 轴旋转是 Yaw，轴向的正方向如上图一样，不变。
先对加速度计和磁力计的数据进行处理，得到 init_xx 来供我们使用如下图：

然后通过公式计算出初始的 Roll、Pitch、Yaw，注意加负号保证旋转角度的正方向，如下图：

其中 Yaw 的正方向未必对，可以自己去验证下，具体参考公式见附件-ST 电子罗盘计算 Yaw：

然后由上面的初始化欧拉角求出初始化四元数，这时要注意旋转顺序的不同，公式也不同，大部分旋转顺序是 Z-Y-X，我的程序里也用的这个顺序，在这里我们按 Z-X-Y 的顺序来旋转，并得出求四元数的公式以做比较，其旋转矩阵:
q=qyaw*qpitch*qroll=
(cos(0.5*Yaw)+ksin(0.5*Yaw)) * (cos(0.5*Pitch)+isin(0.5* Pitch)) * (cos(0.5*Roll)+jsin(0.5* Roll))
得出初始化四元数计算公式如下图所示：

其中 i，j，k 之间相乘的顺序不能随意变，在前的先计算，在后的后计算相乘的公式如下图：

至此初始化四元数完成。

第三步就是使用 IMUUpdate 算法了，用完以后再根据公式计算出欧拉角，此公式跟旋转顺序和旋转使用的轴向有关，我们的旋转顺序是 Z-X-Y，且绕 Z 是 Yaw，绕 X 是 Pitch，绕 Y 是 Roll，推到过程如下图：
首先得出 3 个方向余旋矩阵：
下图绕 Z 轴 Yaw：

下图绕 X 轴 pitch：

下图绕 Y 轴 Roll：

然后按照我们的 Z-X-Y 顺序求得 C=Croll * Cpitch * Cyaw，如下图：

将上图的方向余旋矩阵 C 与下图的四元数姿态矩阵做对比，即可求出欧拉角，注意上图的方向余旋矩阵 C 是随着我们对坐标轴的定义变化而变化的，而下图的四元数姿态矩阵是固定的：

最后一步就是求出欧拉角，公式如下图：

至此，算法完成，参考如下：
//本程序的读写 mpu9150 部分是在 http://www.amobbs.com/thread-5538389-1-1.html，作者_spetrel 的基础上修改而来；
//本程序 Yaw 的互补滤波出自 http://www.amobbs.com/thread-5542278-1-1.html，作者 john800422；
//本程序采用的 gyro accel 融合算法出自作者 Madgwick；
在此感谢以上作者的无私分享和帮助！

最后，附上我的代码和 AHRS IMU 的原始代码：