摘要

MPU6050是一种非常流行的空间运动传感器芯片，可以获取器件当前的三个加速度分量和三个旋转角速度。由于其体积小巧，功能强大，精度较高，不仅被广泛应用于工业，同时也是航模爱好者的神器，被安装在各类飞行器上驰骋蓝天。

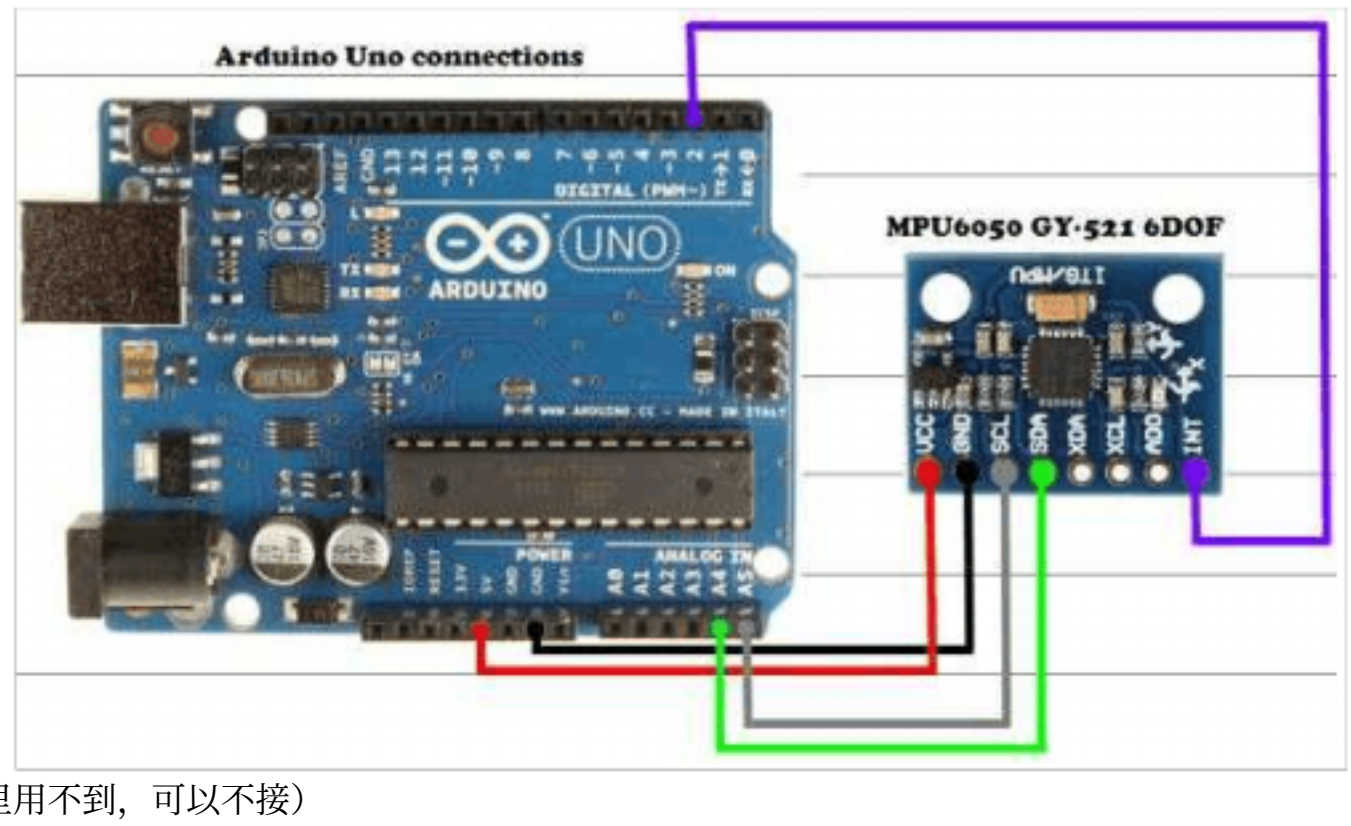
随着Arduino 开发板的普及，许多朋友希望能够自己制作基于 MPU6050的控制系统，但由于缺乏专业知识而难以上手。此外，MPU6050的数据是有较大噪音的，若不进行滤波会对整个控制系统的精准确带来严重影响。

MPU6050芯片内自带了一个数据处理子模块 DMP，已经内置了滤波算法，在许多应用中使用 DMP输出的数据已经能够很好的满足要求。关于如何获取 DMP的输出数据，我将在以后的文章中介绍。本文将直接面对原始测量数据，从连线、芯片通信开始一步一步教你如何利用 Arduino 获取MPU6050的数据并进行卡尔曼滤波，最终获得稳定的系统运动状态。

一、Arduino 与MPU-6050的通信

为避免纠缠于电路细节，我们直接使用集成的 MPU6050模块。MPU6050的数据接口用的是 I2C总线协议，因此我们需要 Wire程序库的帮助来实现 Arduino 与MPU6050之间的通信。请先确认你的 Arduino 编程环境中已安装 Wire库。

Wire库的官方文档（ http://www.arduino.cc/en/Reference/Wire ）中指出：在 UNO板子上， SDA接口对应的是 A4引脚，SCL对应的是 A5引脚。 MPU6050需要5V的电源，可由 UNO板直接供电。按照下图连线。



（紫色线是中断线，这里用不到，可以不接）

MPU6050的数据写入和读出均通过其芯片内部的寄存器实现，这些寄存器的地址都是 1个字节，也就是 8位的寻址空间，其寄存器的详细列表说明书请点击下载： https://www.olimex.com/Products/Modules/Sensors/MOD-MPU6050/resources/RM-MPU-60xxA_rev_4.pdf

1.1 将数据写入 MPU-6050

在每次向器件写入数据前要先打开 Wire的传输模式，并指定器件的总线地址， MPU6050的总线地址是 0x68（AD0引脚为高电平时地址为 0x69）。然后写入一个字节的寄存器起始地址，再写入任意长度的数据。这些数据将被连续地写入到指定的起始地址中，超过当前寄存器长度的将写入到后面地址的寄存器中。写入完成后关闭 Wire的传输模式。下面的示例代码是向MPU6050的0x6B寄存器写入一个字节 0。

Wire.beginTransmission(0x68); // 开启MPU6050的传输 Wire.write(0x6B); // 指定寄存器地址 Wire.write(0); // 写入一个字节的数据 Wire.endTransmission(true); // 结束传输， true 表示释放总线

1.2 从MPU-6050读出数据

读出和写入一样，要先打开 Wire的传输模式，然后写一个字节的寄存器起始地址。接下来将指定地址的数据读到 Wire库的缓存中，并关闭传输模式。最后从缓存中读取数据。下面的示例代码是从 MPU6050的0x3B寄存器开始读取 2个字节的数据：

Wire.beginTransmission(0x68); // 开启MPU6050的传输 Wire.write(0x3B); // 指定寄存器地址 Wire.requestFrom(0x68, 2, true); // 将输据读出到缓存 Wire.endTransmission(true); // 关闭传输模式 int val = Wire.read() << 8 | Wire.read(); // 两个字节组成一个 16位整数

1.3 具体实现

通常应当在 setup 函数中对 Wire库进行初始化：

Wire.begin();

在对MPU6050进行各项操作前，必须启动该器件，向它的 0x6B写入一个字节 0即可启动。通常也是在 setup 函数完成，代码见1.1 节。

二、 MPU6050的数据格式

我们感兴趣的数据位于 0x3B到0x48这14个字节的寄存器中。这些数据会被动态更新，更新频率最高可达 1000HZ。下面列出相关寄存器的地址，数据的名称。注意，每个数据都是 2个字节。

- 0x3B ，加速度计的 X轴分量 ACC_X
- 0x3D ，加速度计的 Y轴分量 ACC_Y
- 0x3F ，加速度计的 Z轴分量 ACC_Z
- 0x41 ，当前温度 TEMP
- 0x43 ，绕 X轴旋转的角速度 GYR_X
- 0x45 ，绕 Y轴旋转的角速度 GYR_Y
- 0x47 ，绕 Z轴旋转的角速度 GYR_Z

MPU6050芯片的座标系是这样定义的：令芯片表面朝向自己，将其表面文字转至正确角度，此时，以芯片内部中心为原点，水平向右的为 X轴，竖直向上的为 Y轴，指向自己的为 Z轴。见下图：

我们只关心加速度计和角速度计数据的含义，下面分别介绍。

2.1 加速度计

加速度计的三轴分量 ACC_X ACC_Y和ACC_Z均为16位有符号整数，分别表示器件在三个轴向上的加速度，取负值时加速度沿座标轴负向，取正值时沿正向。

三个加速度分量均以重力加速度 g的倍数为单位，能够表示的加速度范围，即倍率可以统一设定，有 4个可选倍率： 2g、4g、8g、16g。以ACC_X为例，若倍率设定为 2g（默认），则意味着 ACC_X取最小值 -32768时，当前加速度为沿 X轴正方向 2倍的重力加速度；若设定为 4g，取-32768时表示沿 X轴正方向 4倍的重力加速度，以此类推。显然，倍率越低精度越好，倍率越高表示的范围越大，这要根据具体的应用来设定。

我们用 f 表示倍率， f=0 为2g， f=3 为16g，设定加速度倍率的代码如下：

```
Wire.beginTransmission(0x68); // 开启MPU-6050的传输 Wire.write(0x1C); // 加速度倍率寄存器的地址 Wire.requestFrom(0x68, 1, true); // 先读出原配置 unsigned char acc_conf = Wire.read();acc_conf = ((acc_conf & 0xE7) | (f << 3));Wire.write(acc_conf);Wire.endTransmission(true); // 结束传输， true 表示释放总线
```

再以ACC_X为例，若当前设定的加速度倍率为 4g，那么将 ACC_X读数换算为加速度的公式为：

$$a_x = 4g \times \mathrm{ACC\_X}/32768$$

a_x=4g\times \text{ACC_X}/32768

，g可取当地重力加速度。

2.2 角速度计

绕X、Y和Z三个座标轴旋转的角速度分量 GYR_X GYR_Y和GYR_Z均为16位有符号整数。从原点向旋转轴方向看去，取正值时为顺时针旋转，取负值时为逆时针旋转。

三个角速度分量均以“ 度 /秒 ”为单位，能够表示的角速度范围，即倍率可统一设定，有 4个可选倍率： 250度/秒、500度/秒、1000度/秒、2000度/秒。以GYR_X为例，若倍率设定为 250度/秒，则意味着 GYR_X取正最大值 32768时，当前角速度为顺时针 250度/秒；若设定为 500度/秒，取32768时表示当前角速度为顺时针 500度/秒。显然，倍率越低精度越好，倍率越高表示的范围越大。

我们用 f 表示倍率， f=0 为250度/秒， f=3 为2000度/秒，除角速度倍率寄存器的地址为 0x1B之外，设定加速度倍率的代码与2.1 节代码一致。

以GYR_X为例，若当前设定的角速度倍率为 1000度/秒，那么将 GRY_X读数换算为角速度（顺时针）的公式为：

$$g_x = 1000 \times \mathrm{GYR\_X}/32768$$

g_x=1000\times \text{GYR_X} / 32768

。

三、运动数据

在读取加速度计和角速度计的数据并换算为物理值后，根据不同的应用，数据有不同的解译方式。本章将以飞行器运动模型为例，根据加速度和角速度来算出当前的飞行姿态。

3.1 加速度计模型

我们可以把加速度计想象为一个正立方体盒子里放着一个球，这个球被弹簧固定在立方体的中心。当盒子运动时，根据假想球的位置即可算出当前加速度的值。想象如果在太空中，盒子没有任何受力时，假想球将处于正中心的位置，三个轴的加速度均为 0。见下图：

如果我们给盒子施加一个水平向左的力，那么显然盒子就会有一个向左的加速度，此时盒内的假想球会因为惯性作用贴向盒内的右侧面。如下图所示：

为了保证数据的物理意义，MPU6050 的加速度计是以假想球在三轴上座标值的相反数作为三个轴的加速度值。当假想球的位置偏向一个轴的正向时，该轴的加速度读数为负值，当假想球的位置偏向一个轴的负向时，该轴的加速度读数为正值。

根据以上分析，当我们把 MPU6050 芯片水平放于地方，芯片表面朝向天空，此时由于受到地球重力的作用，假想球的位置偏向 Z 轴的负向，因此 Z 轴的加速度读数应为正，且在理想情况下应为 g。注意，此加速度的物理意义并不是重力加速度，而是自身运动的加速度，可以这样理解：正因为其自身运动的加速度与重力加速度大小相等方向相反，芯片才能保持静止。

3.2 Roll-pitch-yaw 模型与姿态计算

表示飞行器当前飞行姿态的一个通用模型就是建立下图所示坐标系，并用 Roll 表示绕 X 轴的旋转，Pitch 表示绕 Y 轴的旋转，Yaw 表示绕 Z 轴的旋转。

由于 MPU6050 可以获取三个轴向上的加速度，而地球重力则是长期存在且永远竖直向下，因此我们可以根据重力加速度相对于芯片的指向为参考算得当前姿态。

为方便起见，我们让芯片正面朝下固定在上图飞机上，且座标系与飞机的坐标系完全重合，以三个轴向上的加速度为分量，可构成加速度向量。假设当前芯片处于匀速直线运动状态，那么应垂直于地面上向，即指向 Z 轴负方向，模长为

$$|a| = g = \sqrt{x^2 + y^2 + z^2}$$

|a|=g=\sqrt{x^2+y^2+z^2}

（与重力加速度大小相等，方向相反，见 3.1 节）。若芯片（座标系）发生旋转，由于加速度向量仍然竖直向上，所以 Z 轴负方向将不再与重合。见下图。

为了方便表示，上图坐标系的 Z 轴正方向（机腹以及芯片正面）向下，X 轴正方向（飞机前进方向）向右。此时芯片的 Roll 角（黄色）为加速度向量与其在 XZ 平面上投影的夹角，Pitch 角（绿色）与其在 YZ 平面上投影的夹角。求两个向量的夹角可用点乘公式：

$$a \cdot b = |a| \cdot |b| \cdot \cos\theta$$

a \cdot b=|a|\cdot|b|\cdot\cos\theta

，简单推导可得：

$$\phi = \cos^{-1}(\sqrt{x^2 + z^2}/g)$$

\phi=\cos^{-1}(\sqrt{x^2+z^2}/g)

，以及

$$\omega = \cos^{-1}(\sqrt{y^2 + z^2}/g)$$

\omega=\cos^{-1}(\sqrt{y^2+z^2}/g)

注意，因为 arccos 函数只能返回正值角度，因此还需要根据不同情况来取角度的正负值。当 y 值为正时，Roll 角要取负值，当 x 轴为负时，Pitch 角要取负值。

3.4 Yaw 角的问题

因为没有参考量，所以无法求出当前的 Yaw 角的绝对角度，只能得到 Yaw 的变化量，也就是角速度 GYR_Z。当然，我们可以通过对 GYR_Z 积分的方法来推算当前 Yaw 角（以初始值为准），但由于测量精度的问题，推算值会发生漂移，一段时间后就完全失去意义了。然而在大多数应用中，比如无人机，只需要获得 GRY_Z 就可以了。

如果必须要获得绝对的 Yaw 角，那么应当选用 MPU9250 这款九轴运动跟踪芯片，它可以提供额外的三轴罗盘数据，这样我们就可以根据地球磁场方向来计算 Yaw 角了，具体方法此处不再赘述。

四、数据处理与实现

MPU6050 芯片提供的数据夹杂有较严重的噪音，在芯片处理静止状态时数据摆动都可能超过 2%。除了噪音，各项数据还会有偏移的现象，也就是说数据并不是围绕静止工作点摆动，因此要先对数据偏移进行校准，再通过滤波算法消除噪音。

4.1 校准

校准是比较简单的工作，我们只需要找出摆动的数据围绕的中心点即可。我们以 GRY_X 为例，在芯片处理静止状态时，这个读数理论上讲应当为 0，但它往往会存在偏移量，比如我们以 10ms 的间隔读取了 10个值如下：

-158.4, -172.9, -134.2, -155.1, -131.2, -146.8, -173.1, -188.6, -142.7, -179.5

这10个值的均值，也就是这个读数的偏移量为 -158.25 。在获取偏移量后，每次的读数都减去偏移量就可以得到校准后的读数了。当然这个偏移量只是估计值，比较准确的偏移量要对大量的数据进行统计才能获知，数据量越大越准，但统计的时间也就越慢。一般校准可以在每次启动系统时进行，那么你应当在准确度和启动时间之间做一个权衡。

三个角速度读数 GYR_X、GYR_Y 和 GYR_Z 均可通过统计求平均的方法来获得，但三个加速度分量就不能这样简单的完成了，因为芯片静止时的加速度并不为 0。

加速度值的偏移来自两个方面，一是由于芯片的测量精度，导至它测得的加速度向量并不垂直于大地；二是芯片在整个系统（如无人机）上安装的精度是有限的，系统与芯片的座标系很难达到完美重合。前者我们称为读数偏移，后者我们称为角度偏移。因为读数和角度之间是非线性关系，所以要想以高精度进行校准必须先单独校准读数偏移，再把芯片固定在系统中后校准角度偏移。然而，由于校准角度偏移需要专业设备，且对于一般应用来说，两步校准带来的精度提升并不大，因此通常只进行读数校准即可。下面介绍读数校准的方法。我们还以 3.2 节的飞机为例，分以下几个步骤：

1. 首先要确定飞机的坐标系，对于多轴飞行器来说这非常重要。如果坐标系原点的位置或坐标轴的方向存在较大偏差，将会给后面的飞控造成不良影响。
2. 在确定了飞机的坐标系后，为了尽量避免读数偏移带来的影响，首先将 MPU6050 牢牢地固定在飞机上，并者座标系尽可能的重合。当然把 Z轴反过来装也是可以的，就是需要重新推算一套角度换算公式。
3. 将飞机置于水平、坚固的平面上，并充分预热。对于多轴无人机而言，空中悬停时的 XY 平面应当平行 XY 平面。此时，我们认为芯片的加速度方向应当与 Z轴负方向重合，且加速度向量的模长为 g，因此 ACC_Y 的理论值应为 0， ACC_Z 的理论值应为 -16384 （假设我们设定 2g 的倍率， 1g 的加速度的读数应为最大值 -32768 的一半）。
4. 由于 ACC_X 和 ACC_Y 的理论值应为 0，与角速度量的校准类似，这两个读数偏移量可用统计均值的方式校准。ACC_Z 则需要多一步处理，即在统计偏移量的过程中，每次读数都要加上 16384 ，再进行统计均值校准。

4.2 卡尔曼滤波

对于夹杂了大量噪音的数据，卡尔曼滤波器的效果无疑是最好的。如果不想考虑算法细节，可以直接使用 Arduino 的 Klaman Filter 库完成。在我们的模型中，一个卡尔曼滤波器接受一个轴上的角度值、角速度值以及时间增量，估计出一个消除噪音的角度值。跟据当前的角度值和上一轮估计的角度值，以及这两轮估计的间隔时间，我们还可以反推出消除噪音的角速度。

实现代码见 4.3 节。下面介绍卡尔曼滤波算法细节，不感兴趣的可跳过。

（想看的人多了再写）

4.3 实现代码

以下代码在 Arduino 软件1.65版本中编译、烧写以及测试通过。

// 本代码版权归Devymex所有，以GNU GENERAL PUBLIC LICENSE V3.0发布// http://www.gnu.org/licenses/gpl-3.0.en.html// 相关文档参见作者于知乎专栏发表的原创文章：// http://zhuanlan.zhihu.com/devymex/20082486// 连线方法//MPU-UNO//VCC-VCC//GND-GND//SCL-A5//SDA-A4//INT-2 (Optional) #include#include#includefloatfRad2Deg=57.295779513f; // 将弧度转为角度的乘数constintMPU=0x68; //MPU-6050的I2C地址constintnValCnt=7; // 一次读取寄存器的数量constintnCalibTimes=1000; // 校准时读数的次数intcalibData[nValCnt]; // 校准数据unsignedlongnLastTime=0; // 上一次读数的时间floatfLastRoll=0.0f; // 上一次滤波得到的Roll 角floatfLastPitch=0.0f; // 上一次滤波得到的Pitch 角KalmankalmanRoll; //Roll 角滤波器KalmankalmanPitch; //Pitch 角滤波器voidsetup(){Serial.begin(9600); // 初始化串口，指定波特率Wire.begin(); // 初始化Wire库WriteMPUReg(0x6B,0); // 启动MPU6050设备Calibration(); // 执行校准nLastTime=micros(); // 记录当前时间}voidloop(){intreadouts[nValCnt];ReadAccGyr(readouts); // 读出测量值floatrealVals[7];Rectify(readouts,realVals); // 根据校准的偏移量进行纠正// 计算加速度向量的模长，均以g为单位floatfNorm=sqrt(realVals[0]*realVals[0]+realVals[1]*realVals[1]+realVals[2]*realVals[2]);floatfRoll=GetRoll(realVals,fNorm);

计算Roll 角if(realVals[1]>0){fRoll=-fRoll;}floatfPitch=GetPitch(realVals,fNorm); // 计算Pitch 角if(realVals[0]<0){fPitch=-

fPitch;}　// 计算两次测量的时间间隔dt，以秒为单位unsignedlongnCurTime=micros();floatdt=(double)(nCurTime-nLastTime)/1000000.0;　// 对Roll 角和Pitch 角进行卡尔曼滤波floatfNewRoll=kalmanRoll.getAngle(fRoll,realVals[4],dt);floatfNewPitch=kalmanPitch.getAngle(fPitch,realVals[5],dt);　// 跟据滤波值计算角度速floatfRollRate=(fNewRoll-fLastRoll)/dt;floatfPitchRate=(fNewPitch-fLastPitch)/dt;　// 更新Roll 角和Pitch 角fLastRoll=fNewRoll;fLastPitch=fNewPitch;　// 更新本次测的时间nLastTime=nCurTime; // 向串口打印输出Roll 角和Pitch 角，运行时在Arduino 的串口监视器中查看Serial.print("Roll:");Serial.print(fNewRoll);Serial.print('(');Serial.print(fRollRate);Serial.print("),\tPitch:");Serial.print(fNewPitc

向MPU6050写入一个字节的数据// 指定寄存器地址与一个字节的值voidWriteMPUReg(intnReg,unsignedcharnVal){Wire.beginTransmission(MPU);Wire.write(nReg);Wire.write(nVal);Wire.endTransmission(true);}　// 从MPU6050读出一个字节的数据// 指定寄存器地址，返回读出的值unsignedcharReadMPUReg(intnReg){Wire.beginTransmission(MPU);Wire.write(nReg);Wire.requestFrom(MPU,1,true);Wire.endTransmission(true);returnWire.read();}　// 从MPU6050读出加速度计三个分量、温度和三个角速度计// 保存在指定的数组中voidReadAccGyr(int*pVals){Wire.beginTransmission(MPU);Wire.write(0x3B);Wire.requestFrom(MPU,nValCnt*2,true);Wire.endTransmission(true);for(longi=0;i<nValCnt;++i){pVals[i]=Wire.read()<<8|Wire.read();}}　// 对大量读数进行统计，校准平均偏移量voidCalibration(){floatvalSums[7]={0.0f,0.0f,0.0f,0.0f,0.0f,0.0f,0.0};　// 先求和for(inti=0;i<nCalibTimes;++i){intmpuVals[nValCnt];ReadAccGyr(mpuVals);for(intj=0;j<nValCnt;++j){valSums[j]+=mpuVals[j];}}　// 再求平均for(inti=0;i<nValCnt;++i){calibData[i]=int(valSums[i]/nCalibTimes);}calibData[2]+=16384;　// 设芯片Z轴竖直向下，设定静态工作点。}// 算得Roll 角。算法见文档。floatGetRoll(float*pRealVals,floatfNorm){floatfNormXZ=sqrt(pRealVals[0]*pRealVals[0]+pRealVals[2]*pRealVals[2]);floatfCos=fNormXZ/fNorm;returnacos(fCos)*fRad2Deg;}算得Pitch 角。算法见文档。floatGetPitch(float*pRealVals,floatfNorm){floatfNormYZ=sqrt(pRealVals[1]*pRealVals[1]+pRealVals[2]*pRealVals[2]);floatfCos=fNormYZ/fNorm;returnacos(fCos)*fRad2Deg;}对读数进行纠正，消除偏移，并转换为物理量。公式见文档。voidRectify(int*pReadout,float*pRealVals){for(inti=0;i<3;++i){pRealVals[i]=(float)(pReadout[i]-calibData[i])/16384.0f;}pRealVals[3]=pReadout[3]/340.0f+36.53;for(inti=4;i<7;++i){pRealVals[i]=(float)(pReadout[i]-calibData[i])/131.0f;}}