基于 mpu6050 的数字滤波算法

向莹光

2015-10-7

简述

三轴加速度计，三轴陀螺仪 mpu6050 用了很久，接触过很多应用在 6050 数据处理上的算法，这里将它们整理起来。造福于大家，同时抛砖引玉，希望大家都能将自己的好东西分享出来。当然只是知识与经验，模块啥的我是不会分享的，你有钱的话，我们可以商量下。

这些滤波算法是各有所长，有其特色，现整理将其展现给大家，并加上个人看法与见解，不足之处，希望大家指出，修正使之更完善。同时期盼实验室后浪推前浪，顶起一片天。

在介绍滤波器之前，讲明一些东西。Mpu6050 的加速度计，具有长期可靠，短期噪声大，加速度计多采用低通滤波。陀螺仪，短时间可靠长期不稳定，故常用积分求角度，对陀螺仪高通滤波。了解这些特性就可以有选择性的去设计滤波器了----------

一阶低通滤波

先给大家介绍下一阶低通滤波器数学模型的建立吧！（其实是为了装逼嘿嘿！大神请忽视）

右图是一阶 RC 电路，也是硬件一阶低通滤波器

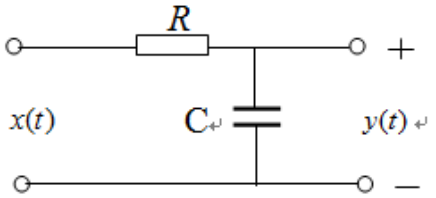

图 1　低通滤波器电路

微分方程：$RC\frac{dy(t)}{dt}+y(t)=x(t)$

差分方程：$RC\frac{y(nT)-y[(n-1)T]}{T}+y(nT)=x(nT)$

整理得：$y(nT)=\frac{RC}{T+RC}y[(n-1)T]+\frac{T}{RC+T}x(nT)$

令 $a=\frac{T}{RC+T}$ 可得：$y(nT)=(1-a)y[(n-1)T]+ax(nT)$

系数 $a=\frac{T}{1/2\pi f_c+T}$　　截止频率 $f_c=\frac{1}{2\pi RC}$

C 程序源码：

```
/*==========================================================================
**函数 : LPF_1st
**功能 : 一阶低通滤波
**备注 : deltaT 采样周期       M_PI= 3.1415926          Fcut 截止频率
**  函数原型：y(n) = （1-a）y(n-1)+a*x(n)
**==========================================================================*/
/*----------------------一阶低通滤波器系数计算-------------------------*/
float LPF_1st_Factor_Cal(float deltaT, float Fcut)
{
    return deltaT / (deltaT + 1 / (2 * M_PI * Fcut));
}

/*----------------------一阶低通滤波器-------------------------*/
float LPF_1st(float oldData, float newData, float lpf_factor)
{
    return oldData * (1 - lpf_factor) + newData * lpf_factor;
}
```

说明：低通滤波器。Mpu6050 内部滤波器的频率典型值为 5Hz，我们一般设置 42Hz，这样硬件初步滤波，然后在软件在滤波一次，截止频率我设置 10Hz，你也可以尝试下 5Hz，20Hz，高的就不用设置了，因为硬件已经滤过一次，软件再滤没什么效果了。

二阶低通滤波

二阶滤波器其实有低通、高通、带通、带阻几种。这里因为时间有限，只整理低通滤波，其他的，大家有兴趣的话，可以帮助完善下。

这里就不再详细了，直接上推导公式，当然你兴趣的话，可以帮我完善 O(∩_∩)O 哈哈~我也可以偷下懒……。

先建立二阶 RC 电路数学模型：

$$(RC)^2 \frac{d^2y(t)}{dt^2} + 2RC\frac{dy(t)}{dt} + y(t) = x(t)$$

令 $\tau = RC$，同时求差分方程，有：

$$\tau^2 \frac{y(nT) - 2y[(n-1)T] + y[(n-2)T]}{T^2} + 2\tau\frac{y(nT) - y[(n-1)T]}{T} + y(nT) = x(nT)$$

整理得：

$$y(nT) = \frac{T^2}{\tau^2 + 2\tau T + T^2}x(nT) + \frac{2\tau^2 + 2\tau T}{\tau^2 + 2\tau T + T^2}y[(n-1)T] - \frac{\tau^2}{\tau^2 + 2\tau T + T^2}y[(n-2)T]$$

令 $a = \frac{1}{2\pi f_c T}$ 可得：

$$y(nT) = \frac{1}{a^2+2a+1}x(nT) + \frac{2a^2+2a}{a^2+2a+1}y[(n-1)T] - \frac{a^2}{a^2+2a+1}y[(n-2)T]$$

C 程序源码：

```
/*====================================================================*
**函数 : LPF_1st
**功能 :二阶低通滤波
**备注 : deltaT 采样周期       M_PI= 3.1415926          Fcut 截止频率
            LPF2ndData_t 二阶低通滤波器系数结构体
**====================================================================*/
/*----------------------二阶低通滤波器系数计算-------------------------*/
void LPF_2nd_Factor_Cal(float deltaT, float Fcut, LPF2ndData_t* lpf_data)
{
    float a = 1 / (2 * M_PI * Fcut * deltaT);
    lpf_data->b0 = 1 / (a*a + 2*a + 1);
    lpf_data->a1 = (2*a*a + 2*a) / (a*a + 2*a + 1);
    lpf_data->a2 = (a*a) / (a*a + 2*a + 1);
}

/*----------------------二阶低通滤波器-----------------------*/
float LPF_2nd(LPF2ndData_t* lpf_2nd, float newData)
{
    float lpf_2nd_data;
    lpf_2nd_data = newData * lpf_2nd->b0 + lpf_2nd->lastout * lpf_2nd->a1
                    - lpf_2nd->preout * lpf_2nd->a2;
    lpf_2nd->preout = lpf_2nd->lastout;
    lpf_2nd->lastout = lpf_2nd_data;
    return lpf_2nd_data;
}
```

说明：二阶低通滤波，相比于一阶低通滤波，对通频带以外的信号抑制能力更强，效果更好。一阶，二阶你都可以试试，那个效果更好用哪个。

互补滤波

滤波器的只是把两部分数据按权重加起来，使其能输出一个有意义的、准确的线性估计。一般互补滤波用到加速度计与陀螺仪的数据融合，因为其各有优点，可有很好的综合利用。互补滤波知识不难，我就不整理了。有人想帮忙的话，感激不尽！

C 程序源码：

```
/*==========================================================================*
**函数 : CF_1st
**功能 :一阶互补滤波
**备注 : deltaT 采样周期 ，tau 时间常数，建议值 1.2f
**==========================================================================*/
/*----------------------互补滤波器系数计算-------------------------*/
float CF_Factor_Cal(float deltaT, float tau)
{
    return tau / (deltaT + tau);
}
/*----------------------一阶互补滤波器-----------------------------*/
float   CF_1st(float   gyroData, float   accData, float cf_factor)
{
    return (gyroData * cf_factor + accData *(1 - cf_factor));
}
```

改进型互补滤波：

数学模型：$angle = a*(oldangle + gryo*dt) + (1-a)*accel$

```
/*----------------------一阶互补滤波器-----------------------------*/
float   CF2_1st(float oldData,float   gyroData, float   accData, float cf_factor,float deltaT)
{
        return ((gyroData*deltaT+oldData )*cf_factor + accData *(1 - cf_factor));
}
```

说明：互补滤波，谁的权重比越大，就是越相信谁。至于上面哪种效果好点，自己用下就知道了，第一种我没用过，改进型互补滤波加入了上一次数据，即前面的结果对现在的结果有影响。在这里搬砖一下比人对该进型滤波的研究。

当$a = 0.98$时，如果滤波器在每秒执行 100 次的循环里运行，滤波器的时间常数将会是：

$$\tau = \frac{adt}{1-a} = \frac{0.98*0.01\text{sec}}{0.02} = 0.49\text{sec}$$

时间常数定义了是该相信陀螺仪还是加速度传感器的界限，当时间周期小于半秒的时候，陀螺仪的积分起主要的作用加速度传感器的噪声将会被滤除，当时间周期大于半秒的时候，加速度传感器要的比重要比陀螺仪大，这时候可能会有漂移。

首先，你会定一个时间常数然后用它去计算滤波器系数。根据时间常数可以调整响应的快慢。

假如你的陀螺仪每秒漂移 2 度（当然，可能是最坏的估计），这时为了保证在每个方向的漂移不会超过几度，你可能需要一个小于 1 秒的时间常数。但是随着时间常数的减小，加速度传感器的噪声就会被更多的引入到系统之中。请记住，要想得到合适的滤波器系数，采样率是很重要的。如果你改变了你的程序，增加了浮点运算，这两个因素的会使采样率就会下降，除非你重新计算你的滤波器条件，否则你的时间常数是不会减小的。

卡尔曼滤波

整理了，一上午，腰酸背痛的，哎！没假期，还没国庆，以后打死不做程序猿、代码君。当然啦，这只是吐槽！搞科技，高收入，媳妇房子以后不用愁，为了将来孩子的奶粉钱，当爹的受点苦算什么，孩子也肯定是亲生的，你们这群坏银不要乱想…….

那么什么是卡尔曼，个人总结一句话，卡尔曼滤波算法就是用上一时刻最优结果估算当前时刻值，再拿这一次的测量值和估算值计算当前最优值（大神勿喷）。卡尔曼算法原理有点高深，我是没懂其精髓。就将别人写的给整理下吧！

卡尔曼就几个基本公式：

X(k|k-1)=A X(k-1|k-1)+B U(k) …….. (1)

P(k|k-1)=A P(k-1|k-1) A’+Q …… (2)

X(k|k)= X(k|k-1)+Kg(k) (Z(k)-H X(k|k-1)) …… (3)

Kg(k)= P(k|k-1) H’ / (H P(k|k-1) H’ + R) …… (4)

P(k|k)=（I-Kg(k) H）P(k|k-1) …… (5)

C 程序源码：

```
/*=============================================================================
**函数 : KalmanFilter
**功能 : 卡尔曼滤波
**输入 : ProcessNiose_Q 系统过程噪声协方差 MeasureNoise_R 测量噪声协方差 ResrcData 修复的值（输入数据）
         p_last 先验估计协方差， x_last 上一次值
**輸出 : 修正后的数据
**备注 :
         Q:过程噪声，Q 增大，动态响应变快，收敛稳定性变坏
         R:测量噪声，R 增大，动态响应变慢，收敛稳定性变好
         参考值 Q 0.02 R 8.00 P 0.9
**===========================================================================*/
double KalmanFilter(const double ResrcData,double ProcessNiose_Q,double MeasureNoise_R,double x_last,double p_last)
{
    double R = MeasureNoise_R;
    double Q = ProcessNiose_Q;
    double x_mid = x_last;
    double x_now;
    double p_mid ;
```

```
    double p_now;
    double kg;

    x_mid=x_last;                 //x_last=x(k-1|k-1),x_mid=x(k|k-1)
    p_mid=p_last+Q;               //p_mid=p(k|k-1),p_last=p(k-1|k-1),Q=噪声
    kg=p_mid/(p_mid+R);           //kg 为 kalman filter，R 为噪声
    x_now=x_mid+kg*(ResrcData-x_mid);//估计出的最优值

    p_now=(1-kg)*p_mid;           //最优值对应的 covariance
    p_last = p_now;               //更新 covariance 值
    x_last = x_now;               //更新系统状态值
    return x_now;
}
```

卡尔曼滤波器修改型：

实验室学长写过适用于 mpu6050 的卡尔曼滤波，互补了陀螺仪与加速度计，很好用。在这里我把它整理出来，方便大家。大家把下面的 5 个公式展开出来，就是代码了。

公式: （宁继超设计）

$$\begin{bmatrix} angle \\ e \end{bmatrix} = \begin{bmatrix} 1 & -T_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} angle \\ e \end{bmatrix} + \begin{bmatrix} T_S \\ 0 \end{bmatrix} gyro \quad \text{……..(1)}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \begin{bmatrix} 1 & -T_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -T_S & 1 \end{bmatrix} + \begin{bmatrix} Q_{\_accel} & 0 \\ 0 & Q_{\_gyro} \end{bmatrix} \quad \text{……(2)}$$

$$\begin{bmatrix} angle \\ e \end{bmatrix} = \begin{bmatrix} angle \\ e \end{bmatrix} + \begin{bmatrix} K_{g1} \\ K_{g2} \end{bmatrix} \left(Accel - \begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} angle \\ e \end{bmatrix} \right) \quad \text{……..(3)}$$

$$\begin{bmatrix} K_{g1} \\ K_{g2} \end{bmatrix} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Big/ \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - R \right) \quad \text{……..(4)}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} = \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} K_{g1} \\ K_{g2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix} \quad \text{……..(5)}$$

C 程序源码：（范家赫编写）

```
/*==========================================================================
参数：Accel   加速度计数据
      Gyro    陀螺仪数据
      Ts      采样时间
/Angle_PK 为上一次滤波结果，Angle_L 为最近一次滤波结果，Kg_1,Kg_2,e_PKre 为上一次漂移量预测值，e_Lat 为最近一次漂移量预测值，y 为残差,s 为协方差
==========================================================================*/
#define Qi    0.01 //  加速度计协方差
#define Qj    0.0001 //陀螺仪协方差
#define Rm    0.5    //系统协方差
float PK[2][2] = {{1,0},{0,1}};
float Angle=0,e=0,y=0,s=0,Kg_1=0,Kg_2=0;
float Kalman_Cal(float Accel,float Gyro,float Ts)
{
   Angle = Angle + Ts * ( Gyro- e );    //状态估计

   //方差估计
   PK[0][0] = PK[0][0] - Ts * ( PK[0][1] + PK[1][0] ) + Ts * Ts * PK[1][1] + Qi;
   PK[0][1] = PK[0][1] - Ts * PK[1][1];
   PK[1][0] = PK[1][0] - Ts * PK[1][1];
   PK[1][1] = PK[1][1] + Qj;

   y = Accel - Angle; //残差

   //最优增益
   Kg_1 = PK[0][0] / ( PK[0][0] + Rm );
   Kg_2 = PK[1][0] / ( PK[0][0] + Rm );

   //数据更新
   Angle = Angle + Kg_1 * y;
         e = e + Kg_2 * y;
   PK[0][0] = ( 1 - Kg_1 ) * PK[0][0];
   PK[1][0] = ( 1 - Kg_1 ) * PK[0][1];
   PK[1][1] = -Kg_2 * PK[0][0] + PK[1][0];
   PK[1][1] = -Kg_2 * PK[0][1] + PK[1][1];

   return Angle;
}
```

说明：卡尔曼滤波是经典的滤波器，但是太高深不好懂所以他的系数也不好调。而且不能很好的兼顾滤波效果与动态响应快慢，并且有滞后性，不适应要求反应迅速的控制系统。相信也有设计很好的卡尔曼滤波算法，但是目前在处理 mpu6050 上我只遇到这些。

IIR 数字滤波

终于快完了，整理也够累的，当然你能看到这儿，对我来说也是值了，至少整理的东西有人看，感谢知遇之恩！

IIR 数字滤波，测试信号里面有，上课时也没怎么听懂，放到实际更不知道怎么用。都怪老师，对的，我们要敢于把责任推给别人，就赖给老师，我们这么聪明，会学不会这玩意儿……..老师啊！那个啥,要不是期末改卷时，你手抖了下，我也不会黑你，不要怪我哈!。

这里我们设计一个 4 阶 IIR 滤波器 ，相关的 IIR 滤波原理不要问我，吼吼，自己找度娘，我只是个搬运工，读书少。数学好的，可以帮帮忙。

4 阶 IIR 数字滤波，差分方程。

函数原型：

$$y(n)=b_0x(n)+b_1x(n-1)+b_2x(n-2)+b_3x(n-3)+b_4x(n-4)$$

$$-a_1y(n-1)-a_2y(n-2)-a_3y(n-3)-a_4y(n-4)$$

到这里，我就不再推导怎么计算系数了，就算你问我，我也只能回答你，哪凉快那待着去。其实我也不会……。那怎么办，不是歇菜了。其实 MATLAB 是个好东西。比如说我们设计一个 4 阶直接 I 型 IIR 低通滤波器，采样频率 500Hz（2ms 采集一次），截止频率为 30hz，就可以调用 MATLAB 命令如下：

```
Fs=500;
[b,a]=butter(4,30/(Fs/2));
[z,p,k]=butter(4,30/(Fs/2));
freqz(b,a,512,Fs)
```

输出系数：

a:[1,-3.0176,3.5072,-1.8476,0.3708]

b:[8.0635e-04,0.0032,0.0048,0.0032,8.0635e-04]

C 程序源码：

```
#define    IIR_ORDER        4          //使用 IIR 滤波器的阶数
double b_IIR[IIR_ORDER+1] ={0.0008f, 0.0032f, 0.0048f, 0.0032f, 0.0008f};   //系数 b
double a_IIR[IIR_ORDER+1] ={1.0000f, -3.0176f, 3.5072f, -1.8476f, 0.3708f};//系数 a
double InPut_IIR[3][IIR_ORDER+1] = {0};
double OutPut_IIR[3][IIR_ORDER+1] = {0};

/*=====================================================================*
**  函数名称: IIR_I_Filter
**  功能描述: IIR 直接 I 型滤波器
**  输       入: InData  为当前数据
**                *x       储存未滤波的数据
**                *y       储存滤波后的数据
```

```
**                *b        储存系数 b
**                *a        储存系数 a
**                nb        数组*b 的长度
**                na        数组*a 的长度
** 输       出: OutData
** 说       明: 无
** 函数原型: y(n) = b0*x(n) + b1*x(n-1) + b2*x(n-2) - a1*y(n-1) - a2*y(n-2)
```

示例：Outdata = IIR_I_Filter(InData, InPut_IIR[0], OutPut_IIR[0], b_IIR, IIR_ORDER+1, a_IIR, IIR_ORDER+1);

```
**==========================================================================*/
double IIR_I_Filter(double InData, double *x, double *y, double *b, short nb, double *a, short na)
{
   double z1,z2;
   short i;
   double OutData;

     for(i=nb-1; i>0; i--)    {x[i]=x[i-1];    }
       x[0] = InData;
     for(z1=0,i=0; i<nb; i++)    { z1 += x[i]*b[i];    }
     for( i=na-1; i>0; i--) {y[i]=y[i-1]; }
     for(z2=0,i=1; i<na; i++) { z2 += y[i]*a[i]; }
     y[0] = z1 - z2;
   OutData = y[0];
   return OutData;
}
```

说明：这段滤波代码，是我前段时间调试四轴时看到的代码，现在整理下。实际使用中发现，CPU 运算越快，滤波效果更好，而且实时响应效果很好，建议大家使用。IIR 数字滤波还有低通，高通，带通，带阻。大家有兴趣的话，可以自己下去 MATLAB 仿真下。告诉大家，Stm32F4 系列的 arm 有浮点运算处理器，别人封装好了数字滤波库，有兴趣的可以研究下，分享给大家。

忙绿了两天，终于将这该死的东西整理完了，代码仅供参考。有什么不对的地方，希望大家积极指出，使之更加完美。