

# 几种常用的滤波算法及其优缺点

# 为何要使用软件滤波

工业自动化系统中的大型系统或是小型设备，均含有各种噪声和干扰。干扰既有来自信号源本体或传感器，也有来自外界干扰。为了进行准确测量和控制，必须消除被测信号中的噪声和干扰。特别是随着自动化程度的提高，许多控制功能通过自动闭环调节来完成，设备控制的效果取决于外部模拟量采集、控制算法、执行输出等等环节，而在现场工业环境中，电磁干扰、源干扰、甚至于传感器本身都会影响外部信号，导致得到的数据失真、波动，如果在数据采集环节即出现问题，那整个系统将无法正常工作。所以要正确选择软件数字滤波的方式处理外部信号的正确采集，从而才能得到真实的数据，实现自动控制。



# 几种常用的滤波算法

- 1、限幅滤波法
- 2、中位值滤波法
- 3、算术平均滤波法
- 4、递推平均滤波法
- 5、中位值平均滤波法
- 6、限幅平均滤波法
- 7、一阶滞后滤波法
- 8、加权递推平均滤波法
- 9、消抖滤波法

# 限幅滤波法（程序判断滤波法）

\*函数名称：AmplitudeLimiterFilter()-限幅滤波法

\*方法：

根据经验判断，确定两次采样允许的最大偏差值（设为A）

每次检测到新值时判断：

如果本次值与上次值之差 $\leq A$ ，则本次值有效

如果本次值与上次值之差 $> A$ ，则本次值无效，放弃本次值，用上次值代替本次值

\*优点：能有效克服因偶然因素引起的脉冲干扰

\*缺点：无法抑制那种周期性的干扰，且平滑度差



\*说明:

### 1、调用函数

GetAD(),该函数用来取得当前值

### 2、变量

Value:最近一次有效采样的值,该变量为全局变量

NewValue:当前采样的值

ReturnValue:返回值

### 3、常量说明

A:两次采样的最大误差值,该值需要使用者根据实际情况设置

\*入口: Value,上一次有效的采样值,在主程序里赋值

\*出口: ReturnValue,返回值,本次滤波结果

```
#define A 10
```

```
unsigned char Value
```

```
unsigned char Filter()
```

```
{
```

```
    unsigned char NewValue;
```

```
    unsigned char ReturnValue;
```

```
    NewValue=GatAD();
```

```
    if((((NewValue-Value)>A)) || ((Value-NewValue)>A)))
```

```
        ReturnValue=Value;
```

```
    else ReturnValue=NewValue;
```

```
    return(ReturnValue);
```

```
}
```



# 中位值滤波法

\*函数名称: MiddlevalueFilter()-中位值滤波法

\*方法:

连续采样N次 (N取奇数)

把N次采样值按大小排列

取中间值为本次有效值

\*优点: 能有效克服因偶然因素引起的波动干扰; 对温度、

液位等变化缓慢的被测参数有良好的滤波效果

\*缺点: 对流量, 速度等快速变化的参数不宜

\*说明:

### 1、调用函数

GetAD(),该函数用来取得当前值

Delay(),基本延时函数

### 2、变量说明

ArrDataBuffer[N]:用来存放一次性采集的N组数据

Temp:完成冒泡法试用的临时寄存器

i,j,k:循环试用的参数值

### 3、常量说明

N: 数组长度



```
#define N 11
```

```
unsigned char MiddlevalueFilter()
```

```
{
```

```
    unsigned char value_buf[N];
```

```
    unsigned char i,j,k,temp;
```

```
    for(i=0;i<N;i++)
```

```
    {
```

```
        value_buf[i] = get_ad();
```

```
        delay();
```

```
    }
```

先进行N次采样

```
for (j=0;j<N-1;j++)
```

```
{
```

```
for (k=0;k<N-j;k++)
```

```
{
```

```
if(value_buf[k]>value_buf[k+1])
```

```
{
```

```
temp = value_buf[k];
```

```
value_buf[k] = value_buf[k+1];
```

```
value_buf[k+1] = temp;
```

```
}
```

```
}
```

```
}
```

```
return value_buf[(N-1)/2];
```

冒泡法排序

排序完成后，直接返回最中间的那个数据

```
}
```



# 算术平均滤波法

\*方法:

连续取 $N$ 个采样值进行算术平均运算

$N$ 值较大时: 信号平滑度较高, 但灵敏度较低

$N$ 值较小时: 信号平滑度较低, 但灵敏度较高

$N$ 值的选取: 一般流量,  $N=12$ ; 压力:  $N=4$

\*优点:

适用于对一般具有随机干扰的信号进行滤波

这样信号的特点是有一个平均值, 信号在某一数值范围附近上下波动

\*缺点:

对于测量速度较慢或要求数据计算速度较快的实时控制不适用

比较浪费RAM

```
#define N 12
```

```
char filter()
```

```
{
```

```
    unsigned int sum = 0;
```

```
    unsigned char i;
```

```
    for (i=0; i<N; i++)
```

```
    {
```

```
        sum += get_ad();
```

```
        delay();
```

```
    }
```

```
    return(char)(sum/N);
```

```
}
```

连续采样N次累加总和

总和除以次数得到算术平均值



# 递推平均滤波法（滑动平均滤波法）

\*方法：

把连续取 $N$ 个采样值看成一个队列

队列的长度固定为 $N$

每次采样到一个新数据放入队尾,并扔掉原来队首的一个数据.(先进先出原则)

把队列中的 $N$ 个数据进行算术平均运算,就可获得新的滤波结果

$N$ 值的选取: 流量,  $N=12$ ; 压力:  $N=4$ ; 液面,  $N=4\sim 12$ ; 温度,  $N=1\sim 4$

## B、优点：

对周期性干扰有良好的抑制作用，平滑度高

适用于高频振荡的系统

## C、缺点：

灵敏度低

对偶然出现的脉冲性干扰的抑制作用较差

不易消除由于脉冲干扰所引起的采样值偏差

不适用于脉冲干扰比较严重的场合

比较浪费RAM



```
#define N 12
```

```
char value_buf[N];  相当于一个12空间的队列
```

```
char i=0;
```

```
char filter()
```

```
{
```

```
char count;
```

```
int sum=0;
```

```
value_buf[i++] = get_ad();
```

```
if (i == N) i = 0;
```

```
for (count=0;count<N,count++)
```

```
    +=  
    sum = value_buf[count];
```

```
return (char)(sum/N);
```

```
}
```

采样一个数据入队，循环使用空间

累加所有的值

累加值除以N得到算法平均值

# 中位值平均滤波法（防脉冲干扰平均滤波法）

## A、方法：

相当于“中位值滤波法”+“算术平均滤波法”

连续采样 $N$ 个数据，去掉一个最大值和一个最小值

然后计算 $N-2$ 个数据的算术平均值

$N$ 值的选取：3~14

## B、优点：

融合了两种滤波法的优点

对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差对周期干扰有良好的抑制作用

平滑度高，适于高频振荡的系统。

## C、缺点：

测量速度较慢，和算术平均滤波法一样，比较浪费RAM



```
#define N 12
```

```
uchar filter()
```

```
{
```

```
    unsigned char i,j,k,l;
```

```
    unsigned char temp,sum=0,value;
```

```
    unsigned char value_buf[N],;
```

```
    for(i=0;i<N;i++)
```

```
    {
```

```
        value_buf[i] = get_ad();
```

```
        delay();
```

```
    }
```

先连续采样N个值

for(j=0;j<N-1;j++) //采样值从小到大排列 (冒泡法)

{

for(i=0;i<N-j;i++)

{

if(value\_buf[i]>value\_buf[i+1])

{

temp = value\_buf[i];

value\_buf[i] = value\_buf[i+1];

value\_buf[i+1] = temp;

}

}

}



for(i=1;i<N-1;i++) 注意此处for循环去掉了0和N，即去掉了一个最小值和一个最大值。

sum += value\_buf[i];

value = sum/(N-2);

return(value);

}

# 限幅平均滤波法

\*方法:

相当于“限幅滤波法”+“递推平均滤波法”

每次采样到的新数据先进行限幅处理，

再送入队列进行递推平均滤波处理

\*优点:

融合了两种滤波法的优点

对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差

\*缺点:

比较浪费RAM



```
#define A 10
```

```
#define N 12
```

```
unsigned char data[];
```

```
unsigned char filter(data[])
```

```
{
```

```
    unsigned char i;
```

```
    unsigned char value,sum;
```

```
    data[N]=GetAD();
```

```
    if(((data[N]-data[N-1])>A || ((data[N-1]-data[N])>A))
```

```
        data[N]=data[N-1];
```

```
    //else data[N]=NewValue;
```

```
for(i=0;i<N;i++)
```

```
{
```

```
    data[i]=data[i+1];
```

```
    sum+=data[i];
```

```
}
```

```
value=sum/N;
```

```
return(value);
```

```
}
```



# 一阶滞后滤波法

## A、方法：

取 $a=0\sim 1$

本次滤波结果 =  $(1-a) * \text{本次采样值} + a * \text{上次滤波结果}$

## B、优点：

对周期性干扰具有良好的抑制作用

适用于波动频率较高的场合

## C、缺点：

相位滞后，灵敏度低

滞后程度取决于 $a$ 值大小

不能消除滤波频率高于采样频率的 $1/2$ 的干扰信号

/\* 为加快程序处理速度假定基数为100, a=0~100 \*/

#define a 50

char value;

char filter()

{

char new\_value;

new\_value = get\_ad();

return ((100-a)\*value + a\*new\_value) ;

}



# 加权递推平均滤波法

## \*方法:

是对递推平均滤波法的改进, 即不同时刻的数据加以不同的权

通常是, 越接近现时刻的数据, 权取得越大。

给予新采样值的权系数越大, 则灵敏度越高, 但信号平滑度越低

## \*优点:

适用于有较大纯滞后时间常数的对象

和采样周期较短的系统

## \*缺点:

对于纯滞后时间常数较小, 采样周期较长, 变化缓慢的信号不能迅速反应系统当前所受干扰的严重程度, 滤波效果差

/\* coe数组为加权系数表，存在程序存储区。\*/

#define N 12

const char code coe[N] = {1,2,3,4,5,6,7,8,9,10,11,12};

const char code sum\_coe =  
1+2+3+4+5+6+7+8+9+10+11+12;

unsigned char filter()

{

unsigned char i;

unsigned char value\_buf[N];

int sum=0;



```
for (i=0;i<N;i++)
```

```
{
```

```
    value_buf[i] = get_ad();
```

```
    delay();
```

```
}
```

```
for (i=0,i<N;i++)
```

```
{
```

```
    value_buf[i]=value_buf[i+1];
```

```
    sum += value_buf[i]*coe[i];
```

```
}
```

```
sum/=sum_coe;
```

```
value=sum/N;
```

```
return(value);
```

```
}
```

# 消抖滤波法

## \*方法:

设置一个滤波计数器, 将每次采样值与当前有效值比较:

如果采样值 = 当前有效值, 则计数器清零

如果采样值  $\neq$  当前有效值, 则计数器+1, 并判断计数器是否  $\geq$  上限N(溢出)

如果计数器溢出, 则将本次值替换当前有效值, 并清计数器

## \*优点:

对于变化缓慢的被测参数有较好的滤波效果,

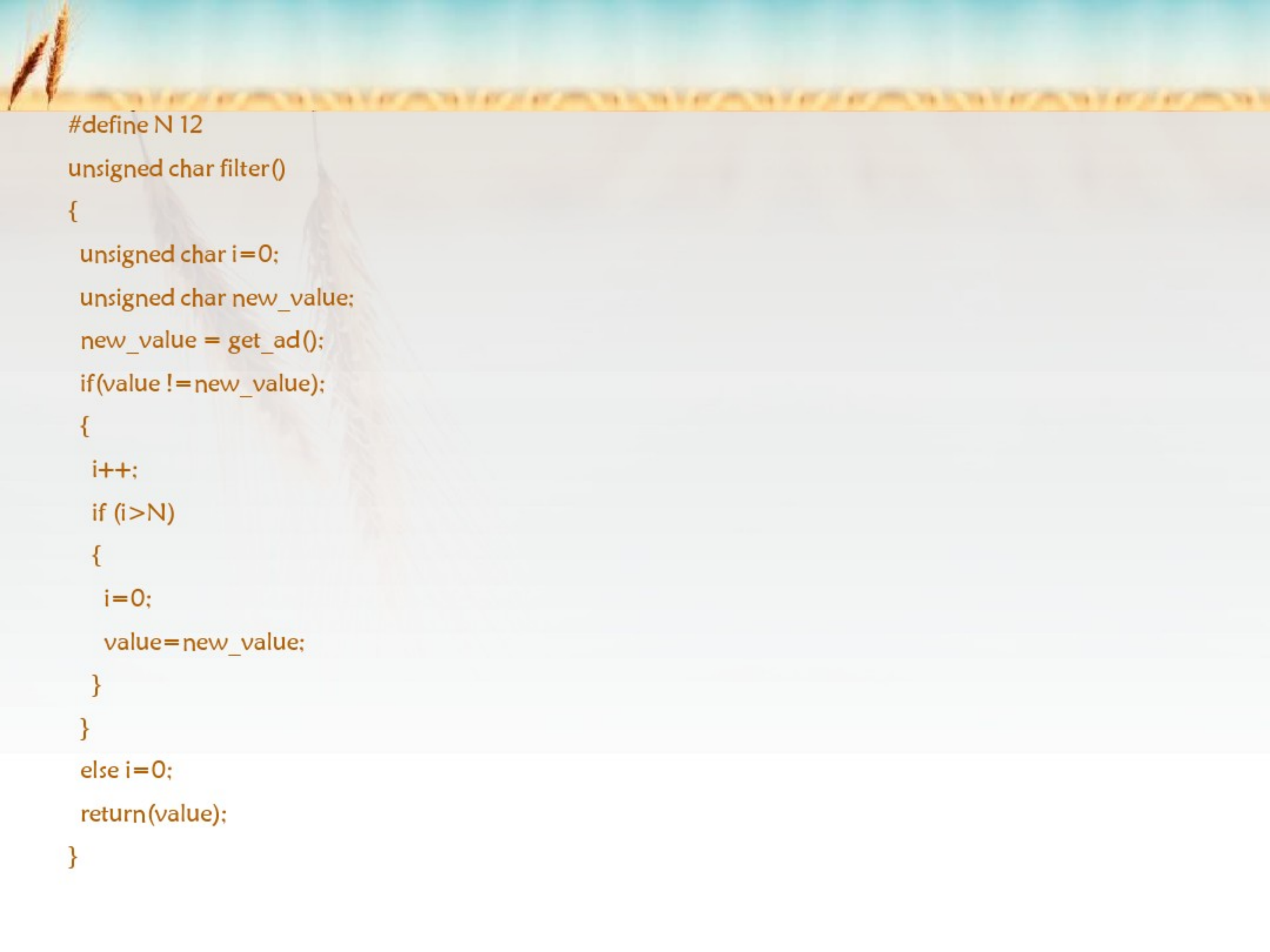
可避免在临界值附近控制器的反复开/关跳动或显示器上数值抖动

## \*缺点:

对于快速变化的参数不宜

如果在计数器溢出的那一次采样到的值恰好是干扰值, 则会将干扰值当作有效值导入系统





```
#define N 12
```

```
unsigned char filter()
```

```
{
```

```
    unsigned char i=0;
```

```
    unsigned char new_value;
```

```
    new_value = get_ad();
```

```
    if(value != new_value);
```

```
    {
```

```
        i++;
```

```
        if (i>N)
```

```
        {
```

```
            i=0;
```

```
            value=new_value;
```

```
        }
```

```
    }
```

```
    else i=0;
```

```
    return(value);
```

```
}
```



thank you