卡尔曼滤波算法
------在 SISO 系统中的应用

“简单来说，卡尔曼滤波器是一个“optimal recursive data processing algorithm（最优化自回归数据处理算法）”。对于解决很大部分的问题，他是最优，效率最高甚至是最有用的。他的广泛应用已经超过 30 年，包括机器人导航，控制，传感器数据融合甚至在军事方面的雷达系统以及导弹追踪等等。近年来更被应用于计算机图像处理，例如头脸识别，图像分割，图像边缘检测等等。”

上文是从网上引用的别人的评价，可以看出，在线性系统里面，用最通俗的讲法来说，该算法是最好的。OK，既然是最好的，那么，就值得尝试一下。

由于卡尔曼算法网上介绍的多是使用在 MISO 中（多输入多输出系统），复杂的矩阵变换对于单片机而言是复杂的，难以实现的，同时，也不是必须的。很多时候单片机处理的只是 SISO 系统（单输入单输出系统），就算是多 MISO 系统，也是可以转化成 SISO 系统来处理的，经过参考网上不少的文章，对该算法有了一个大概的了解后，加入了自己对该算法的理解，将该算法成功的应用到了 SISO 系统中。

卡尔曼滤波算法涉及以下几个变量，这里先给出约定名称：

1、卡尔曼增益：　　KalmanGain　　缩写 KG
2、本次估计值：　　EstimateValueNow　　缩写 EVN
3、上次估计值：　　EstimateValuePre　　缩写 EVP
4、当前估计协方差：EstimateCovarianceNOW　缩写 ECN
5、下次估计协方差：EstimateCovarianceNext 缩写 ECX
6、测　量　值：　　MeasureValue　　缩写 MV
7、当前测量协方差：MeasureCovarianceNow　缩写 MCN
8、下次测量协方差：MeasureCovarianceNext　缩写 MCX
9、估计变化比：　　EstimateChangeRate　　缩写 ECR

首先，给出(公式 1)

$$EVN = ECR \times EVP + KG \times (MV - EVP)$$ ----------(公式 1)

怎么样，是不是和一阶滤波算法比较接近？

一阶滤波算法：

$$NowFilter = a \times \mathrm{Pre}Filter + (1-a) \times NowMeasure$$

我刚看到这个公式的时候就感觉这部分的主要思想和一阶滤波的思想比较接近，只是这里的 KG 是一个动态变化的数据。并且(ECR-KG)和 KG 没有一个和为 1 的限制关系。当然，对于一个系统，你可以将 ECR 认为就是 1，比如一个温度测量系统，温度你就认为基本是一个不变量，本次测量值就应该和上次测量值是一样大，那么此时就可以认为 ECR 就是 1，事实上，很多的时候都可以认为 ECR 就是 1。

既然 KG 是动态变化的，那么应当由一个计算 KG 的(公式 2)

$$KG = \sqrt{\frac{ECN^2}{ECN^2 + MCN^2}}$$ ----------(公式 2)

这里，就要用到协方差了，对于一个具体的系统，你可以认为估计协方差是恒定的，也可以认为测量协方差是恒定的，如果你认为两个都是恒定的，也可以，只是此时就可以看成是一个变相的一阶滤波了，只要参数得当(ECR 为 1 即可)，此时的确就是一个一阶滤波。

不过我们既然说卡尔曼是最好的，当然不能取固定值来使用了，如果这样的话，那岂不是和一阶滤波这样初等的方法没有区别了？之所以卡尔曼是最好的，就是因为他认为测量协方差不是恒定的，因为客观上存在高斯白噪声；同时，主观上的估计协方差也要与时俱进的，外界条件发生了变化，显然我们的估计也要发生变化了。

看来，50+年前的卡尔曼就深入的学习了“与时俱进”的理论了☺

好了，闲话少说，测量协方差和估计协方差是如何“与时俱进”的呢？下面给出(公式 3)和(公式 4)

$$ECX = \sqrt{1 - KG} \times ECN$$ ----------(公式 3)

$$MCX = \sqrt{1 - KG} \times MCN$$ ----------(公式 4)

至于为什么是这样“与时俱进”的呢？这个我就不知道卡尔曼他老人家是如何推测出来的了。反正他给出的计算方法就是这样的。

好了，这里我们计算出了下次运算的测量协方差和估计协方差的值，此时我们不禁要问，第一个测量协方差和估计协方差的值是如何的出来的呢？

问的好！

一个系统都是需要一个初值的，“与时俱进”的好处就是初值可以比较随意的取得，这里的比较随意的意思是，不要将协方差的取值为 0 就可以了。如果测量协方差为 0 就意味着完全相信测量值；如果估计协方差为 0 就意味着完全相信估计值；如果测量协方差和估计协方差都为 0，就是自相矛盾了；显然，这是和我们的“世界观”相冲突的！

所以，世界观告诉我们，只要初始值不为 0，就意味着可以动态调整，既然是可以动态调整，那就是可以与时俱进，既然可以与时俱进，那么就可以做到最好，不愧是“与时俱进”的学习典范啊！政治学习中不忘记科研！这样的好同志哪里找啊？☺

第一次估计值也是可以随意。测量系统都由一个滞后的环节，这个很好理解，既然我们用的是最优的滤波算法，那么一个随意的初始估计值都可以迅速的收敛到实际值的！否则他就不是一个最优的算法。

好了，到此为止，我们再来温故一边整个算法的流程。

首先给出初值：

EVP(0) = 0;(任意初值)

<EVP(0)表示第一次，EVP(1)表示第二次，下同>

ECN(0) = 0.1; (任意不为 0 的初值)

MCN(0) = 0.2; (任意不为 0 的初值)

好了，测量开始：

第一次获得了测量值 MV(0)

第一步，计算 KG(0)，利用(公式 2)

第二步，有了 KG(0)，我们就可以计算当前估计值 EVN(0)了，利用(公式 1)

第三步，计算出了 EVN(0)，那么本次滤波的任务已经完成一半了，我们还需要完成下一次测量所需要递归值就可以了，所以，EVP(1) = EVN(0)。

第四步，利用 (公式 3)(公式 4)计算出 ECX 和 MCX,

所以 ECN(1)=ECX,MCN(1)=MCX。

到此，本次滤波完成。

其中 EVN(X)序列就是我们的滤波输出序列。

注意事项：对于单片机而言，本滤波算法的运算还是比较复杂的，其中涉及比较多的浮点运算和浮点变量。如果资源比较紧张的情况下，不太适合用这种算法。

以下是 C 实现代码：

```
#ifndef __KALMAN_H
#define __KALMAN_H


typedef    struct   str_kalman
{
    fp32    fpKalmanGain;           //卡尔曼增益
    fp32    fpEstimateCovariance;   //估计协方差
    fp32    fpMeasureCovariance;    //测量协方差

    fp32    fpEstimateValue;        //估计值
}STR_KALMAN;
extern    STR_KALMAN    strKalman;

/*********************************************************************************************************
* 函 数 名: KalmanFilterInit
* 功    能: 参数初始化
* 入口参数: 无
* 出口参数: 无
* 原 作 者: leo
* 创建日期: 2008年3月17日
* 修 改 者:
* 修改日期:
*********************************************************************************************************/
extern    void    KalmanFilterInit(void);



/*********************************************************************************************************
* 函 数 名: KalmanFilter
* 功    能: 卡尔曼滤波算法实现
* 入口参数: fpMeasure , 当前测量值
* 出口参数: 卡尔曼滤波后输出值
* 原 作 者: leo
* 创建日期: 2008年3月17日
* 修 改 者:
* 修改日期:
*********************************************************************************************************/
extern    fp32    KalmanFilter(fp32 fpMeasure);




#endif
```

```c
#include "config.h"
#include "math.h"


STR_KALMAN    strKalman;

void    KalmanFilterInit(void);

fp32    KalmanFilter(fp32 fpMeasure);



/*********************************************************************************************
* 函 数 名：KalmanFilterInit
* 功    能：参数初始化
* 入口参数：无
* 出口参数：无
* 原 作 者：leo
* 创建日期：2008年3月17日
* 修 改 者：
* 修改日期：
*********************************************************************************************/
void    KalmanFilterInit(void)
{
    strKalman.fpEstimateValue = 0;
    strKalman.fpEstimateCovariance = 0.1;
    strKalman.fpMeasureCovariance  = 0.02;
}



/*********************************************************************************************
* 函 数 名：KalmanFilter
* 功    能：卡尔曼滤波算法实现
* 入口参数：fpMeasure , 当前测量值
* 出口参数：卡尔曼滤波后输出值
* 原 作 者：leo
* 创建日期：2008年3月17日
* 修 改 者：
* 修改日期：
*********************************************************************************************/
fp32    KalmanFilter(fp32 fpMeasure)
{

    //计算卡尔曼增益
    strKalman.fpKalmanGain = strKalman.fpEstimateCovariance *
                             sqrt(1 / (strKalman.fpEstimateCovariance * strKalman.fpEstimateCovariance +
                                       strKalman.fpMeasureCovariance  * strKalman.fpMeasureCovariance) );
    //计算本次滤波估计值
    strKalman.fpEstimateValue = strKalman.fpEstimateValue +
                                strKalman.fpKalmanGain * (fpMeasure - strKalman.fpEstimateValue);

    //更新 估计协方差
    strKalman.fpEstimateCovariance = sqrt(1-strKalman.fpKalmanGain) * strKalman.fpEstimateCovariance;
    //更新 测量协方差
    strKalman.fpMeasureCovariance  = sqrt(1-strKalman.fpKalmanGain) * strKalman.fpMeasureCovariance;


    //返回估计值
    return    strKalman.fpEstimateValue;
}
```