登录 | 注册

xiahouzuoxin

一个人的喜欢就是把自己对偶然间闪过的念想坚持，直到它变成一种习惯

目录视图 摘要视图 RSS 订阅

个人资料

xiahouzuoxin

访问： 966139次
积分： 10727
等级： BLOG 7
排名： 第912名

原创： 188篇
转载： 44篇
译文： 1篇
评论： 496条

Github

https://github.com/xiahouzuoxin

2014年3月以来所有文章都在Github维护，这里如果存在格式问题，请移步：

个人主页

文章搜索

博客专栏

OpenCV学习一步步
文章：17篇
阅读：114698

DSP算法研究与实现
文章：10篇
阅读：134760

文章分类

ComputerLanuage (24)

Kalman滤波器从原理到实现

2014-09-26 16:37 39194人阅读 评论(48) 收藏 举报

分类： DSP（35）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

转载请注明出处：http://xiahouzuoxin.github.io/notes

Kalman滤波器的历史渊源

> We are like dwarfs on the shoulders of giants, by whose grace we see farther than they. Our study of the works of the ancients enables us to give fresh life to their finer ideas, and rescue them from time's oblivion and man's neglect.
>
> —— Peter of Blois, late twelfth century

太喜欢第一句话了，“我们是巨人肩膀上的矮人，巨人们的优雅让我么看得更比他们更远”，谁说不是呢？

说起Kalman滤波器的历史，最早要追溯到17世纪，Roger Cotes开始研究最小均方问题。但由于缺少实际案例的支撑（那个时候哪来那么多雷达啊啥的这些信号啊），Cotes的研究让人看着显得很模糊，因此在估计理论的发展中影响很小。17世纪中叶，最小均方估计（Least squares Estimation）理论逐步完善，Tobias Mayer在1750年将其用于月球运动的估计，Leonard Euler在1749年、Pierre Laplace在1787分别用于木星和土星的运动估计。Roger Boscovich在1755用最小均方估计地球的大小。1777年，77岁的Daniel Bernoulli（大名鼎鼎的伯努利）发明了最大似然估计算法。递归的最小均方估计理论是由Karl Gauss建立在1809年（好吧，他声称在1795年就完成了），当时还有Adrien Legendre在1805年完成了这项工作，Robert Adrain在1808年完成的，至于到底谁是Boss，矮子们就别管了吧！

在1880年，丹麦的天文学家Thorvald Nicolai Thiele在之前最小均方估计的基础上开发了一个递归算法，与Kalman滤波非常相似。在某些标量的情况下，Thiele的滤波器与Kalman滤波器时等价的，Thiele提出了估计过程噪声和测量噪声中方差的方法（过程噪声和测量噪声是Kalman滤波器中关键的概念）。

上面提到的这么多研究估计理论的先驱，大多是天文学家而非数学家。现在，大部分的理论贡献都源自于实际的工程。“There is nothing so practical as a good theory”，应该就是“实践是检验真理的唯一标准”之类吧。

现在，我们的控制论大Wiener终于出场了，还有那个叫Kolmogorov（柯尔莫戈洛夫）的神人。在19世纪40年代，Wiener设计了Wiener滤波器，然而，Wiener滤波器不是在状态空间进行的（这个学过Wiener滤波的就知道，它是直接从观测空间z(n)=s(n)+w(n)进行的滤波），Wiener是稳态过程，它假设测量是通过过去无限多个值估计得到的。Wiener滤波器比Kalman滤波器具有更高的自然统计特性。这些也限制其只是更接近理想的模型，要直接用于实际工程中需要足够的先验知识（要预知协方差矩阵），美国NASA曾花费多年的时间研究维纳理论，但依然没有在空间导航中看到维纳理论的实际应用。

Algorithm (12)
Linux (25)
Image/Audio/ML... (44)
uCOS-II (4)
Cortex-M3/M4 (6)
FreeThinking (21)
DSP (36)
FPGA (6)
Papers (0)
Network (1)
Git/SVN (6)
电路设计 (25)
数理化基础 (6)
“磨刀不误砍材工” (15)

最新评论

读论文BinarizedNormedGradier
zsszhaoshuaishuai: 你好,那个问题我下午已经解决了,现在出现的是第一阶段选练完成,但是问题出现在:inline floa...

读论文BinarizedNormedGradier
zsszhaoshuaishuai: 你好,我程序在生成的时候代码没有问题,但是运行的时候出现load annotations finis...

搭建Qt界面的OpenCV开发环境
灿哥哥: 学习啦

Linux操作系统原理与应用（陈莉
Rsmygd-SG: 博主，可以发一份给我吗？谢谢，邮箱rsmygd@163.com

Kalman滤波器从原理到实现
cx1806908579: 谢谢博主，写的很好

Markdown中插入数学公式的方法
zishell: 楼主好，方法三：使用MathJax引擎具体怎么用啊，我在.md 文件中：直接写：$$x=\frac{...

stm32之keil开发环境搭建
Hugo0Chen: mark

数字信号处理的学习资源
Full_Speed_Turbo: 果断收藏

白话压缩感知（含Matlab代码）
zhangyaqin032266: 请问楼主minimize (norm(xp, 1));这句是什么意思 谢谢

搭建Qt界面的OpenCV开发环境
: @benjaminsay:报错为imread was not declared in this sc...

CSDN内链

算法研究-July
机器学习-邹晓艺
机器学习-张睿卿
机器学习-邹宇华
图像处理-魏兰
嵌入式-闫明
嵌入式-fudan_abc
嵌入式-周卫
嵌入式-张同浩
嵌入式-唐攀
信号处理-deepdsp
语言结构-费晓行

CSDN外链

Matrix67
酷壳
吴再柱与公民教育
伯乐在线
Stack Overflow
Stack Exchange
Embedded Gurus
dsprelated
FPGA-kingst
gnuarm
Learn C The Hard Way
Analog Dialogue
罗绍峰

在1950末期，大部分工作开始对维纳滤波器中协方差的先验知识通过状态空间模型进行描述。通过状态空间表述后的算法就和今天看到的Kalman滤波已经极其相似了。Johns Hopkins大学首先将这个算法用在了导弹跟踪中，那时在RAND公司工作的Peter Swerling将它用在了卫星轨道估计，Swerling实际上已经推导出了（1959年发表的）无噪声系统动力学的Kalman滤波器，在他的应用中，他还考虑了使用非线性系统动力学和和测量方程。可以这样说，Swerling和发明Kalman滤波器是失之交臂，一线之隔。在kalman滤波器闻名于世之后，他还写信到AIAA Journal声讨要获得Kalman滤波器发明的荣誉（然而这时已经给滤波器命名Kalman了）。总结其失之交臂的原因，主要是Swerling没有直接在论文中提出Kalman滤波器的理论，而只是在实践中应用。

Rudolph Kalman在1960年发现了离散时间系统的Kalman滤波器，这就是我们在今天各种教材上都能看到的，1961年Kalman和Bucy又推导了连续时间的Kalman滤波器。Ruslan Stratonovich也在1960年也从最大似然估计的角度推导出了Kalman滤波器方程。

目前，卡尔曼滤波已经有很多不同的实现。卡尔曼最初提出的形式现在一般称为简单卡尔曼滤波器。除此以外，还有施密特扩展滤波器、信息滤波器以及很多Bierman, Thornton开发的平方根滤波器的变种。也许最常见的卡尔曼滤波器是锁相环，它在收音机、计算机和几乎任何视频或通讯设备中广泛存在。

从牛顿到卡尔曼

从现在开始，就要进行Kalman滤波器探讨之旅了，我们先回到高一，从物理中小车的匀加速直线运动开始。

话说，有一辆质量为m的小车，受恒定的力F，沿着r方向做匀加速直线运动。已知小车在t-ΔT时刻的位移是s(t-1)，此时的速度为v(t-1)。求：t时刻的位移是s(t)，速度为v(t)？

由牛顿第二定律，求得加速度：$a = F/m$

那么就有下面的位移和速度关系：

$$s(t) = s(t-1) + v(t-1)\Delta T + \frac{1}{2}\frac{F}{m}(\Delta T)^2$$

$$v(t) = v(t-1) + \frac{F}{m}\Delta T$$

如果将上面的表达式用矩阵写在一起，就变成下面这样：

$$\begin{pmatrix} s(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} 1 & \Delta T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} s(t-1) \\ v(t-1) \end{pmatrix} + \begin{pmatrix} \frac{(\Delta T)^2}{2} \\ \Delta T \end{pmatrix} \frac{F}{m} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

卡尔曼滤波器是建立在动态过程之上，由于物理量（位移，速度）的不可突变特性，这样就可以通过t-1时刻估计（预测）t时刻的状态，其状态空间模型为：

$$\mathbf{x(t)} = \mathbf{Ax(t-1)} + \mathbf{Bu(t)} + \mathbf{w(t)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

对比一下(1)(2)式，长得及其相似有木有：

$$A = \begin{pmatrix} 1 & \Delta T \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} \frac{(\Delta T)^2}{2} \\ \Delta T \end{pmatrix}$$

匀加速直线运动过程就是卡尔曼滤波中状态空间模型的一个典型应用。下面我们重点关注(2)式，鉴于研究的计算机信号都是离散的，将(2)是表示成离散形式为：

$$\mathbf{x(n)} = \mathbf{Ax(n-1)} + \mathbf{Bu(n)} + \mathbf{w(n)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

其中各个量之间的含义是：

1. x(n)是状态向量，包含了观测的目标（如：位移、速度）
2. u(n)是驱动输入向量，如上面的运动过程是通过受力驱动产生加速度，所以u(n)和受力有关
3. A是状态转移矩阵，其隐含指示了“n-1时刻的状态会影响到n时刻的状态（这似乎和马尔可夫过程有些类似）”
4. B是控制输入矩阵，其隐含指示了“n时刻给的驱动如何影响n时刻的状态”

水木思 6#105

从运动的角度，很容易理解：小车当前n时刻的位移和速度一部分来自于n-1时刻的惯性作用，这通过Ax(n)来度量，另一部分来自于现在n时刻小车新增加的外部受力，通过Bu(n)来度量。

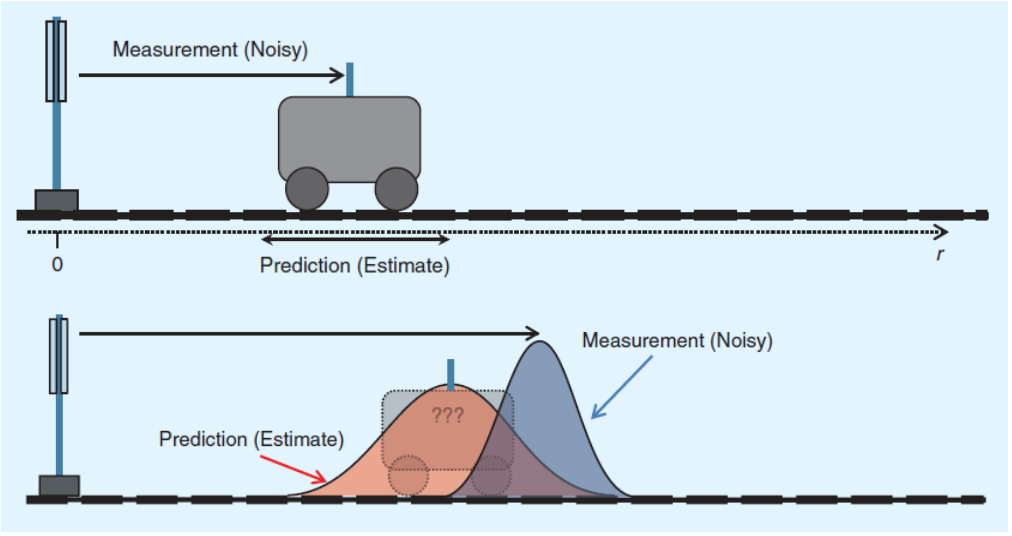
5. w(n)是过程噪声，w(n)~N(0,Q)的高斯分布，过程噪声是使用卡尔曼滤波器时一个重要的量，后面会进行分析。

计算n时刻的位移，还有一种方法：拿一把长的卷尺（嗯，如果小车跑了很长时间，估计这把卷尺就难买到了），从起点一拉，直接就出来了，设测量值为z(n)。计算速度呢？速度传感器往那一用就出来了。

然而，初中物理就告诉我们，“尺子是量不准的，物体的物理真实值无法获得”，测量存在误差，我们暂且将这个误差记为v(n)。这种通过直接测量的方式获得所需物理量的值构成观测空间：

$$\mathbf{z(n)} = \mathbf{H(n)x(n)} + \mathbf{v(n)} \quad \text{.....(4)}$$

z(n)就是测量结果，H(n)是观测矢量，x(n)就是要求的物理量（位移、速度），v(n)~N(0,R)为测量噪声，同状态空间方程中的过程噪声一样，这也是一个后面要讨论的量。大部分情况下，如果物理量能直接通过传感器测量，$\mathbf{H(n)} = \mathbf{I}$。



现在就有了两种方法（如上图）可以得到n时刻的位移和速度：一种就是通过(3)式的状态空间递推计算（Prediction），另一种就是通过(4)式直接拿尺子和传感器测量（Measurement）。致命的是没一个是精确无误的，就像上图看到的一样，分别都存在0均值高斯分布的误差w(n)和v(n)。

那么，我最终的结果是取尺子量出来的好呢，还是根据我们伟大的牛顿第二定律推导出来的好呢？抑或两者都不是！

一场递推的游戏

为充分利用测量值（Measurement）和预测值（Prediction），Kalman滤波并不是简单的取其中一个作为输出，也不是求平均。

设预测过程噪声w(n)~N(0,Q)，测量噪声v(n)~N(0,R)。Kalman计算输出分为预测过程和修正过程如下：

1. 预测

   预测值：

   $$\hat{\mathbf{x}}(\mathbf{n}|\mathbf{n-1}) = \mathbf{A}\hat{\mathbf{x}}(\mathbf{n-1}|\mathbf{n-1}) + \mathbf{Bu(n)} \quad \text{.....(5)}$$

   最小均方误差矩阵：

   $$\mathbf{P(n|n-1)} = \mathbf{AP(n-1|n-1)A^T} + \mathbf{Q} \quad \text{.....(6)}$$

2. 修正

   误差增益：

$$\mathbf{K(n)=P(n|n-1)H^T(n)[R(n)+H(n)P(n|n-1)H^T(n)]^{-1}}.....(7)$$

修正值：

$$\mathbf{\hat{x}(n|n)=A\hat{x}(n|n-1)+K(n)[z(n)-H(n)\hat{x}(n|n-1)]}.....(8)$$ 这里的**A**去掉，参见下面的图

最小均方误差矩阵：

$$\mathbf{P(n|n)=[I-K(n)H(n)]P(n|n-1)}.....(9)$$

从(5)~(9)中：

- x(n)：Nx1的状态矢量
- z(n)：Mx1的观测矢量，Kalman滤波器的输入
- x(n|n-1)：用n时刻以前的数据进行对n时刻的估计结果
- x(n|n)：用n时刻及n时刻以前的数据对n时刻的估计结果，这也是Kalman滤波器的输出
- P(n|n-1)：NxN，最小预测均方误差矩阵，其定义式为

  $$\mathbf{P(n|n-1)=E\{[x(n)-\hat{x}(n|n-1)][x(n)-\hat{x}(n|n-1)]^T\}}$$

  通过计算最终得到(6)式。
- P(n|n)：NxN，修正后最小均方误差矩阵。
- K(n)：NxM，误差增益，从增益的表达式看，相当于“预测最小均方误差”除以“n时刻的测量误差+预测最小均方误差”，直观含义就是用n-1预测n时刻状态的预测最小均方误差在n时刻的总误差中的比重，比重越大，说明真值接近预测值的概率越小（接近测量值的概率越大），这也可以从(8)式中看到。

Kalman滤波算法的步骤是(5)(6)->(7)->(8)(9)。当然，建议找本教材复习下上面公式的推导过程，或参见Wiki上的介绍http://en.wikipedia.org/wiki/Kalman_filter。

公式就是那么的抽象，一旦认真研究懂了却也是茅塞顿开，受益也比只知皮毛的多。尽管如此，我还算更喜欢先感性后理性。仍以上面的运动的例子来直观分析：

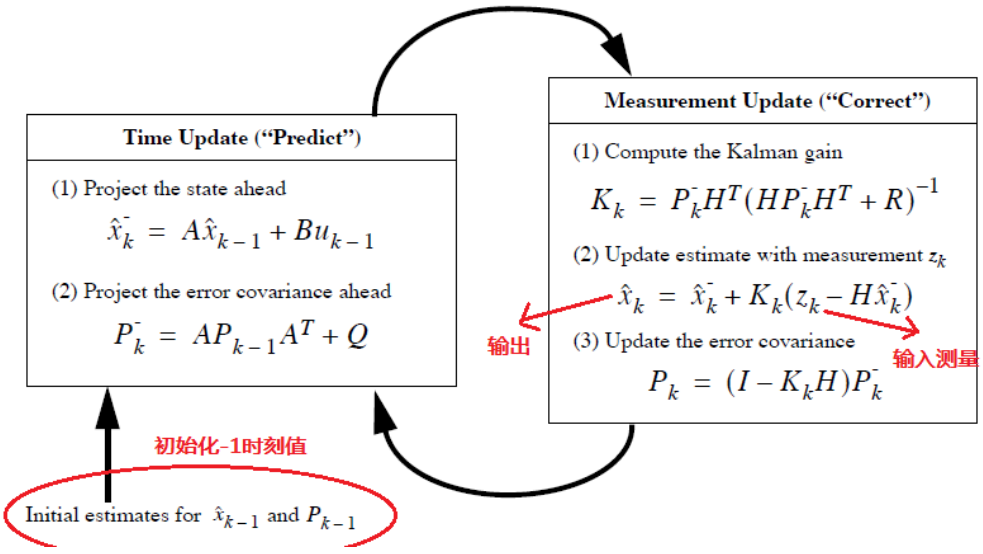
> Example:
>
> 还可以更简单一些：设小车做匀速（而非匀加速）直线运动，方便计算，假设速度绝对的恒定（不波动，所以相关的方差都为0），则u(t)==0恒成立。设预测（过程）位移噪声w(n)~N(0,2^2)，测量位移噪声v(n)~N(0,1^2)，n-1状态的位移 $\hat{x}(n-1|n-1)=50m$，速度为v=10m/s，n时刻与n-1时刻的物理时差为ΔT=1s。同时，也用尺子测了一下，结果位移为z(n)=62m。
>
> 则A = [1 ΔT; 0 1]=[1 1; 0 1]，根据(5)，预测值为
>
> $\hat{x}(n|n-1)=1*\hat{x}(n-1|n-1)+v*\Delta T=[60m;10m/s]$。
>
> 现在已经有了估计值和测量值，哪个更接近真值，这就通过最小均方误差矩阵来决定！
>
> 要求已知上次的修正后的最小均方误差P(n-1|n-1)=[1 0; 0 0]（匀速，所以P(2,2)=0，右斜对角线上为协方差，值为0，P(1,1)为n-1时刻位移量的均方误差，因为要计算P(1,1)还得先递推往前计算P(n-2|n-2)，所以这里暂时假设为1），则根据(6)式，最小预测预测均方误差为P(n|n-1)=[1 0; 0 0][1 1; 0 1][1 0; 0 0]=[1 0; 0 0]。
>
> 由物理量的关系知，H(n)=[1 1]，增益K(n)=[1;0]{1+[1 1][1 0; 0 0][1; 1]}^(-1)=[1/2;0]。
>
> 所以，最后的n时刻估计值既不是用n-1得到的估计值，也不是测量值，而是：$\hat{x}(n|n)=[60;1]+[1/2;0]62-[11][60;1]=[60.5m;1m/s]$，因此，最终的Kalman滤波器的输出位移是60.5m。

从上面的递推关系知道，要估计n时刻就必须知道n-1时刻，那么n=0时刻该如何估计，因此，卡尔曼滤波要初始化的估计值x(-1|-1)和误差矩阵P(-1|-1)，设x(-1,-1)~N(Us, Cs)，则初始化：

$$\mathbf{x(-1|-1)}=\mu_s$$

$$\mathbf{P}(-1|-1)=\mathbf{C_s}$$

综上，借用一张图说明一下Kalman滤波算法的流程：



图中的符号和本文符号稍有差异，主要是P的表示上。从上图也可以看出，Kalman滤波就是给定-1时刻的初始值，然后在预测（状态空间）和修正（观测空间）之间不停的递推，求取n时刻的估计x和均方误差矩阵P。

均方误差中的门道

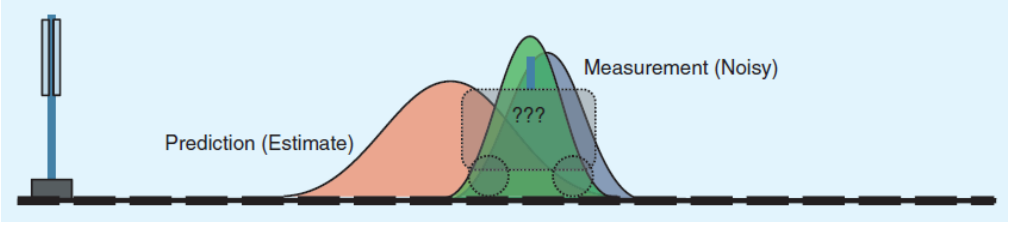
到这里，应该对Kalman滤波有个总体的概念了，有几个观点很重要，是建立Kalman滤波器的基础：

1. 一个是n-1对n时刻估计值，一个是n时刻的测量值，估计值和测量值都存在误差，且误差都假设满足独立的高斯分布
2. Kalman滤波器就是充分结合了估计值和测量值得到n时刻更接近真值的估计结果
3. Kalman滤波器引入状态空间的目的是避免了"像Wiener滤波器一样需要对过去所有[0,n-1]时刻协方差先验知识都已知"，而直接可以通过上一时刻即n-1时刻的状态信息和均方误差信息就可递推得到n时刻的估计。尽管递推使得实际应用中方便了，但n-1对n时刻的估计实际上使用到了所有前[0,n-1]时刻的信息，只不过信息一直通过最小均方误差进行传递到n-1时刻。基于此，Kalman滤波也需要先验知识，即-1时刻的初始值。

在上小节中只看到Kalman的结论，那么Kalman滤波器是如何将估计值和测量值结合起来，如何将信息传递下去的呢？这其中，"独立高斯分布"的假设条件功劳不可谓不大！测量值z(n)~N(uz,σz^2)，估计值x(n)~N(ux,σx^2)。

> Kalman滤波器巧妙的用"独立高斯分布的乘积"将这两个测量值和估计值进行融合！

如下图：估计量的高斯分布和测量量的高斯分布经过融合后为绿色的高斯分布曲线。



$$y_f = \frac{1}{\sqrt{2\pi\sigma_z^2}}\exp^{-\frac{(r-u_z)^2}{2\sigma_z^2}} * \frac{1}{\sqrt{2\pi\sigma_x^2}}\exp^{-\frac{(r-u_x)^2}{2\sigma_x^2}} = \frac{1}{\sqrt{2\pi\sigma^2}}\exp^{-\frac{(r-\mu)^2}{2\sigma^2}}$$

稍微计算一下，通过上式求出u和σ^2，

$$\mu = \frac{u_x\sigma_z^2 + u_z\sigma_x^2}{\sigma_z^2 + \sigma_x^2} = u_x + \frac{\sigma_x^2}{\sigma_z^2 + \sigma_x^2}(u_z - u_x).....(10)$$

$$\sigma^2 = \frac{\sigma_x^2\sigma_z^2}{\sigma_x^2+\sigma_z^2} = \sigma_x^2(1-\frac{\sigma_x^2}{\sigma_x^2+\sigma_z^2}).....(11)$$

现在令

$$K = \frac{\sigma_x^2}{\sigma_x^2+\sigma_z^2}.....(12)$$

则(10)(11)变成：

$$\mu == u_x + K(u_z - u_x).....(13)$$
$$\sigma^2 == \sigma_x^2(1-K).....(14)$$

到这里，请将(13)-(14)与(8)-(9)式对比！标量的情况下，在小车的应用中有：A=1，H=1，正态分布的均值u就是我们要的输出结果，正态分布的方差σz^2就是最小均方误差。推广到矢量的情况，最小均方误差矩阵就是多维正态分布的协方差矩阵。

从(12)式也很容易看到卡尔曼增益K的含义：就是估计量的方差占总方差（包括估计方差和测量方差）的比重。

一切都变得晴朗起来了，然而这一切的一切，却都源自于“估计量和测量量的独立高斯分布”这条假设。进一步总结Kalman滤波器：

> 假设状态空间的n-1时刻估计值和观测空间的n时刻测量值都满足独立高斯分布，Kalman滤波器就是通过高斯分布的乘积运算将估计值和测量值结合，获得最接近真值的n时刻估计。
>
> 高斯分布乘积运算的结果仍为高斯分布，高斯分布的均值对应n时刻的估计值，高斯分布的方差对应n时刻的均方误差。

Matlab程序看过来

下面的一段Matlab代码是从网上找到的，程序简单直接，但作为学习分析用很棒，

```
% KALMANF - updates a system state vector estimate based upon an
%           observation, using a discrete Kalman filter.
%
% Version 1.0, June 30, 2004
%
% This tutorial function was written by Michael C. Kleder
%
% INTRODUCTION
%
% Many people have heard of Kalman filtering, but regard the topic
% as mysterious. While it's true that deriving the Kalman filter and
% proving mathematically that it is "optimal" under a variety of
% circumstances can be rather intense, applying the filter to
% a basic linear system is actually very easy. This Matlab file is
% intended to demonstrate that.
%
% An excellent paper on Kalman filtering at the introductory level,
% without detailing the mathematical underpinnings, is:
% "An Introduction to the Kalman Filter"
% Greg Welch and Gary Bishop, University of North Carolina
% http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html
%
% PURPOSE:
%
% The purpose of each iteration of a Kalman filter is to update
% the estimate of the state vector of a system (and the covariance
% of that vector) based upon the information in a new observation.
% The version of the Kalman filter in this function assumes that
% observations occur at fixed discrete time intervals. Also, this
% function assumes a linear system, meaning that the time evolution
% of the state vector can be calculated by means of a state transition
% matrix.
%
% USAGE:
%
% s = kalmanf(s)
%
% "s" is a "system" struct containing various fields used as input
% and output. The state estimate "x" and its covariance "P" are
```

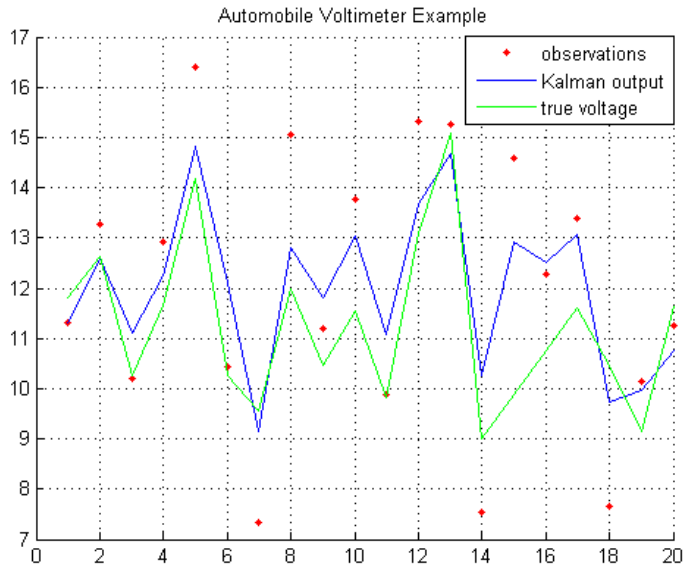
```
% updated by the function. The other fields describe the mechanics
% of the system and are left unchanged. A calling routine may change
% these other fields as needed if state dynamics are time-dependent;
% otherwise, they should be left alone after initial values are set.
% The exceptions are the observation vectro "z" and the input control
% (or forcing function) "u." If there is an input function, then
% "u" should be set to some nonzero value by the calling routine.
%
% SYSTEM DYNAMICS:
%
% The system evolves according to the following difference equations,
% where quantities are further defined below:
%
% x = Ax + Bu + w  meaning the state vector x evolves during one time
%                  step by premultiplying by the "state transition
%                  matrix" A. There is optionally (if nonzero) an input
%                  vector u which affects the state linearly, and this
%                  linear effect on the state is represented by
%                  premultiplying by the "input matrix" B. There is also
%                  gaussian process noise w.
% z = Hx + v       meaning the observation vector z is a linear function
%                  of the state vector, and this linear relationship is
%                  represented by premultiplication by "observation
%                  matrix" H. There is also gaussian measurement
%                  noise v.
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%       v ~ N(0,R) meaning v is gaussian noise with covariance R
%
% VECTOR VARIABLES:
%
% s.x = state vector estimate. In the input struct, this is the
%       "a priori" state estimate (prior to the addition of the
%       information from the new observation). In the output struct,
%       this is the "a posteriori" state estimate (after the new
%       measurement information is included).
% s.z = observation vector
% s.u = input control vector, optional (defaults to zero).
%
% MATRIX VARIABLES:
%
% s.A = state transition matrix (defaults to identity).
% s.P = covariance of the state vector estimate. In the input struct,
%       this is "a priori," and in the output it is "a posteriori."
%       (required unless autoinitializing as described below).
% s.B = input matrix, optional (defaults to zero).
% s.Q = process noise covariance (defaults to zero).
% s.R = measurement noise covariance (required).
% s.H = observation matrix (defaults to identity).
%
% NORMAL OPERATION:
%
% (1) define all state definition fields: A,B,H,Q,R
% (2) define intial state estimate: x,P
% (3) obtain observation and control vectors: z,u
% (4) call the filter to obtain updated state estimate: x,P
% (5) return to step (3) and repeat
%
% INITIALIZATION:
%
% If an initial state estimate is unavailable, it can be obtained
% from the first observation as follows, provided that there are the
% same number of observable variables as state variables. This "auto-
% intitialization" is done automatically if s.x is absent or NaN.
%
% x = inv(H)*z
% P = inv(H)*R*inv(H')
%
% This is mathematically equivalent to setting the initial state estimate
% covariance to infinity.

function s = kalmanf(s)

% set defaults for absent fields:
if ~isfield(s,'x'); s.x=nan*z; end
if ~isfield(s,'P'); s.P=nan; end
if ~isfield(s,'z'); error('Observation vector missing'); end
if ~isfield(s,'u'); s.u=0; end
if ~isfield(s,'A'); s.A=eye(length(x)); end
if ~isfield(s,'B'); s.B=0; end
if ~isfield(s,'Q'); s.Q=zeros(length(x)); end
if ~isfield(s,'R'); error('Observation covariance missing'); end
if ~isfield(s,'H'); s.H=eye(length(x)); end
```

```
if isnan(s.x)
   % initialize state estimate from first observation
   if diff(size(s.H))
      error('Observation matrix must be square and invertible for state
autointialization.');
   end
   s.x = inv(s.H)*s.z;
   s.P = inv(s.H)*s.R*inv(s.H');
else

   % This is the code which implements the discrete Kalman filter:

   % Prediction for state vector and covariance:
   s.x = s.A*s.x + s.B*s.u;
   s.P = s.A * s.P * s.A' + s.Q;

   % Compute Kalman gain factor:
   K = s.P*s.H'*inv(s.H*s.P*s.H'+s.R);

   % Correction based on observation:
   s.x = s.x + K*(s.z-s.H*s.x);
   s.P = s.P - K*s.H*s.P;

   % Note that the desired result, which is an improved estimate
   % of the sytem state vector x and its covariance P, was obtained
   % in only five lines of code, once the system was defined. (That's
   % how simple the discrete Kalman filter is to use.) Later,
   % we'll discuss how to deal with nonlinear systems.

end

return
```

该程序中使用的符号的含义与本文一致，函数前的注释再清晰不过了，就不多说。下面是一段测试代码：

```
% Define the system as a constant of 12 volts:
clear all
s.x = 12;
s.A = 1;
% Define a process noise (stdev) of 2 volts as the car operates:
s.Q = 2^2; % variance, hence stdev^2
% Define the voltimeter to measure the voltage itself:
s.H = 1;
% Define a measurement error (stdev) of 2 volts:
s.R = 2^2; % variance, hence stdev^2
% Do not define any system input (control) functions:
s.B = 0;
s.u = 0;
% Do not specify an initial state:
s.x = nan;
s.P = nan;
% Generate random voltages and watch the filter operate.
tru=[]; % truth voltage
for t=1:20
   tru(end+1) = randn*2+12;
   s(end).z = tru(end) + randn*2; % create a measurement
   s(end+1)=kalmanf(s(end)); % perform a Kalman filter iteration
end
figure
hold on
grid on
% plot measurement data:
hz=plot([s(1:end-1).z],'r.');
% plot a-posteriori state estimates:
hk=plot([s(2:end).x],'b-');
ht=plot(tru,'g-');
legend([hz hk ht],'observations','Kalman output','true voltage',0)
title('Automobile Voltimeter Example')
hold off
```

Kalman的参数中s.Q和s.R的设置非常重要，之前也提过，一般要通过实验统计得到，它们分布代表了状态空间估计的误差和测量的误差。



Kalman滤波器的效果是使输出变得更平滑，但没办法去除信号中原有的椒盐噪声，而且，Kalman滤波器也会跟踪这些椒盐噪声点，因此推荐在使用Kalman滤波器前先使用中值滤波去除椒盐噪声。

Kalman滤波C程序

我就在上面公式的基础上实现了基本的Kalman滤波器，包括1维和2维状态的情况。先在头文件中声明1维和2维Kalman滤波器结构：

```
/*
 * FileName : kalman_filter.h
 * Author   : xiahouzuoxin @163.com
 * Version  : v1.0
 * Date     : 2014/9/24 20:37:01
 * Brief    :
 *
 * Copyright (C) MICL,USTB
 */
#ifndef  _KALMAN_FILTER_H
#define  _KALMAN_FILTER_H

/*
 * NOTES: n Dimension means the state is n dimension,
 * measurement always 1 dimension
 */

/* 1 Dimension */
typedef struct {
    float x;  /* state */
    float A;  /* x(n)=A*x(n-1)+u(n),u(n)~N(0,q) */
    float H;  /* z(n)=H*x(n)+w(n),w(n)~N(0,r)   */
    float q;  /* process(predict) noise convariance */
    float r;  /* measure noise convariance */
    float p;  /* estimated error convariance */
    float gain;
} kalman1_state;

/* 2 Dimension */
typedef struct {
    float x[2];     /* state: [0]-angle [1]-diffrence of angle, 2x1 */
    float A[2][2];  /* X(n)=A*X(n-1)+U(n),U(n)~N(0,q), 2x2 */
    float H[2];     /* Z(n)=H*X(n)+W(n),W(n)~N(0,r), 1x2   */
    float q[2];     /* process(predict) noise convariance,2x1 [q0,0; 0,q1] */
    float r;        /* measure noise convariance */
    float p[2][2];  /* estimated error convariance,2x2 [p0 p1; p2 p3] */
    float gain[2];  /* 2x1 */
} kalman2_state;

extern void kalman1_init(kalman1_state *state, float init_x, float init_p);
extern float kalman1_filter(kalman1_state *state, float z_measure);
extern void kalman2_init(kalman2_state *state, float *init_x, float (*init_p)[2]);
extern float kalman2_filter(kalman2_state *state, float z_measure);
```

```
#endif  /*_KALMAN_FILTER_H*/
```

我都给了有详细的注释，`kalman1_state`是状态空间为1维/测量空间1维的Kalman滤波器，`kalman2_state`是状态空间为2维/测量空间1维的Kalman滤波器。两个结构体都需要通过初始化函数初始化相关参数、状态值和均方差值。

```
/*
 * FileName : kalman_filter.c
 * Author   : xiahouzuoxin @163.com
 * Version  : v1.0
 * Date     : 2014/9/24 20:36:51
 * Brief    :
 *
 * Copyright (C) MICL,USTB
 */

#include "kalman_filter.h"

/*
 * @brief
 *   Init fields of structure @kalman1_state.
 *   I make some defaults in this init function:
 *     A = 1;
 *     H = 1;
 *   and @q,@r are valued after prior tests.
 *
 *   NOTES: Please change A,H,q,r according to your application.
 *
 * @inputs
 *   state - Klaman filter structure
 *   init_x - initial x state value
 *   init_p - initial estimated error convariance
 * @outputs
 * @retval
 */
void kalman1_init(kalman1_state *state, float init_x, float init_p)
{
    state->x = init_x;
    state->p = init_p;
    state->A = 1;
    state->H = 1;
    state->q = 2e2;//10e-6;  /* predict noise convariance */
    state->r = 5e2;//10e-5;  /* measure error convariance */
}

/*
 * @brief
 *   1 Dimension Kalman filter
 * @inputs
 *   state - Klaman filter structure
 *   z_measure - Measure value
 * @outputs
 * @retval
 *   Estimated result
 */
float kalman1_filter(kalman1_state *state, float z_measure)
{
    /* Predict */
    state->x = state->A * state->x;
    state->p = state->A * state->A * state->p + state->q;  /* p(n|n-1)=A^2*p(n-
1|n-1)+q */

    /* Measurement */
    state->gain = state->p * state->H / (state->p * state->H * state->H + state-
>r);
    state->x = state->x + state->gain * (z_measure - state->H * state->x);
    state->p = (1 - state->gain * state->H) * state->p;

    return state->x;
}

/*
 * @brief
 *   Init fields of structure @kalman1_state.
 *   I make some defaults in this init function:
 *     A = {{1, 0.1}, {0, 1}};
 *     H = {1,0};
 *   and @q,@r are valued after prior tests.
 *
 *   NOTES: Please change A,H,q,r according to your application.
 *
 * @inputs
 * @outputs
 * @retval
```

```
 */
void kalman2_init(kalman2_state *state, float *init_x, float (*init_p)[2])
{
    state->x[0]    = init_x[0];
    state->x[1]    = init_x[1];
    state->p[0][0] = init_p[0][0];
    state->p[0][1] = init_p[0][1];
    state->p[1][0] = init_p[1][0];
    state->p[1][1] = init_p[1][1];
    //state->A       = {{1, 0.1}, {0, 1}};
    state->A[0][0] = 1;
    state->A[0][1] = 0.1;
    state->A[1][0] = 0;
    state->A[1][1] = 1;
    //state->H       = {1,0};
    state->H[0]    = 1;
    state->H[1]    = 0;
    //state->q       = {{10e-6,0}, {0,10e-6}};  /* measure noise convariance */
    state->q[0]    = 10e-7;
    state->q[1]    = 10e-7;
    state->r       = 10e-7;  /* estimated error convariance */
}

/*
 * @brief
 *   2 Dimension kalman filter
 * @inputs
 *   state - Klaman filter structure
 *   z_measure - Measure value
 * @outputs
 *   state->x[0] - Updated state value, Such as angle,velocity
 *   state->x[1] - Updated state value, Such as diffrence angle, acceleration
 *   state->p    - Updated estimated error convatiance matrix
 * @retval
 *   Return value is equals to state->x[0], so maybe angle or velocity.
 */
float kalman2_filter(kalman2_state *state, float z_measure)
{
    float temp0 = 0.0f;
    float temp1 = 0.0f;
    float temp = 0.0f;

    /* Step1: Predict */
    state->x[0] = state->A[0][0] * state->x[0] + state->A[0][1] * state->x[1];
    state->x[1] = state->A[1][0] * state->x[0] + state->A[1][1] * state->x[1];
    /* p(n|n-1)=A^2*p(n-1|n-1)+q */
    state->p[0][0] = state->A[0][0] * state->p[0][0] + state->A[0][1] * state-
>p[1][0] + state->q[0];
    state->p[0][1] = state->A[0][0] * state->p[0][1] + state->A[1][1] * state-
>p[1][1];
    state->p[1][0] = state->A[1][0] * state->p[0][0] + state->A[0][1] * state-
>p[1][0];
    state->p[1][1] = state->A[1][0] * state->p[0][1] + state->A[1][1] * state-
>p[1][1] + state->q[1];

    /* Step2: Measurement */
    /* gain = p * H^T * [r + H * p * H^T]^(-1), H^T means transpose. */
    temp0 = state->p[0][0] * state->H[0] + state->p[0][1] * state->H[1];
    temp1 = state->p[1][0] * state->H[0] + state->p[1][1] * state->H[1];
    temp  = state->r + state->H[0] * temp0 + state->H[1] * temp1;
    state->gain[0] = temp0 / temp;
    state->gain[1] = temp1 / temp;
    /* x(n|n) = x(n|n-1) + gain(n) * [z_measure - H(n)*x(n|n-1)]*/
    temp = state->H[0] * state->x[0] + state->H[1] * state->x[1];
    state->x[0] = state->x[0] + state->gain[0] * (z_measure - temp);
    state->x[1] = state->x[1] + state->gain[1] * (z_measure - temp);

    /* Update @p: p(n|n) = [I - gain * H] * p(n|n-1) */
    state->p[0][0] = (1 - state->gain[0] * state->H[0]) * state->p[0][0];
    state->p[0][1] = (1 - state->gain[0] * state->H[1]) * state->p[0][1];
    state->p[1][0] = (1 - state->gain[1] * state->H[0]) * state->p[1][0];
    state->p[1][1] = (1 - state->gain[1] * state->H[1]) * state->p[1][1];

    return state->x[0];
}
```

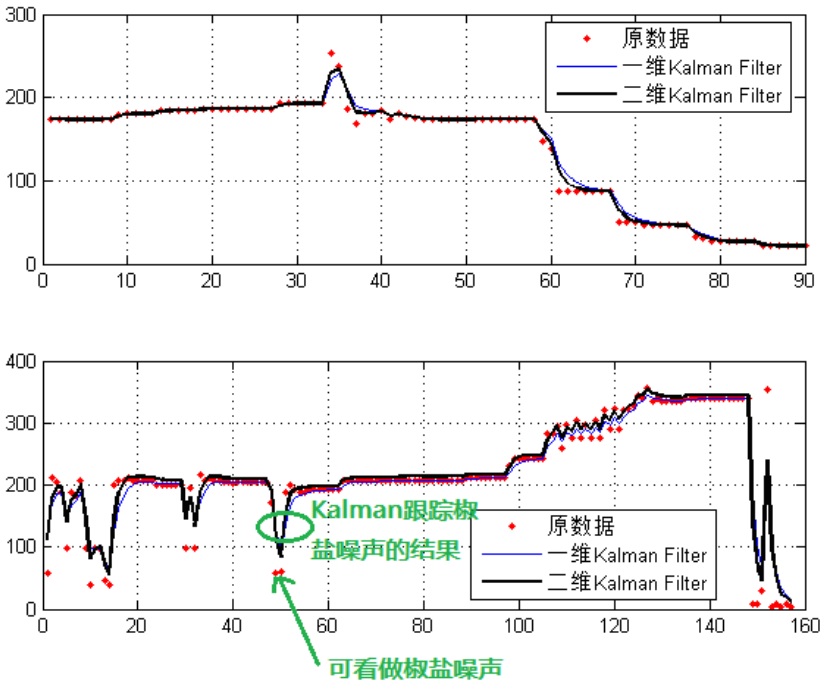
其实，Kalman滤波器由于其递推特性，实现起来很简单。但调参有很多可研究的地方，主要需要设定的参数如下：

1. init_x：待测量的初始值，如有中值一般设成中值（如陀螺仪）
2. init_p：后验状态估计值误差的方差的初始值
3. q：预测（过程）噪声方差

4. r：测量（观测）噪声方差。以陀螺仪为例，测试方法是：保持陀螺仪不动，统计一段时间内的陀螺仪输出数据。数据会近似正态分布，按3σ原则，取正态分布的(3σ)^2作为r的初始化值。

其中q和r参数尤为重要，一般得通过实验测试得到。

找两组声阵列测向的角度数据，对上面的C程序进行测试。一维Kalman（一维也是标量的情况，就我所知，现在网上看到的代码大都是使用标量的情况）和二维Kalman（一个状态是角度值，另一个状态是向量角度差，也就是角速度）的结果都在图中显示。这里再稍微提醒一下：状态量不要取那些能突变的量，如加速度，这点在文章“从牛顿到卡尔曼”一小节就提到过。



Matlab绘出的跟踪结果显示：

> Kalman滤波结果比原信号更平滑。但是有椒盐突变噪声的地方，Kalman滤波器并不能滤除椒盐噪声的影响，也会跟踪椒盐噪声点。因此，推荐在Kalman滤波器之前先使用中值滤波算法去除椒盐突变点的影响。

上面所有C程序的源代码及测试程序都公布在我的Github上，希望大家批评指正其中可能存在的错误。

参考资料

1. Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation. Ramsey Faragher, Lecture Notes.
2. An Introduction to the Kalman Filter. Greg Welch and Gary Bishop.
3. http://robotsforroboticists.com/kalman-filtering/公式彩色着色，含pyhton源码
4. http://alumni.media.mit.edu/~wad/mas864/psrc/kalman.c.txt包含Kalman滤波的C代码
5. http://www.cs.unc.edu/~welch/kalman/比较全的Kalman链接

顶 35 踩 1

上一篇 OpenCV图像处理篇之采样金字塔

下一篇 有关超声探头阻抗的分析

我的同类文章

DSP（35）

- 自适应含噪信号过零率算法 2014-09-11 阅读 2665
- DSP/BIOS使用之初窥门径… 2014-07-25 阅读 2216
- DSP连接不上CCS3.3的问… 2014-07-06 阅读 2501
- 导出CCS3.3数据及使用ma.. 2014-04-22 阅读 4142
- 在DSP671x上使用Timer统… 2014-03-30 阅读 1410
- 白话压缩感知（含Matlab代.. 2014-08-25 阅读 14033
- DSP-BIOS使用入门 2014-07-24 阅读 7414
- 对功率谱的一点理解 2014-07-05 阅读 4117
- 烧写Flash后的DSP程序运… 2014-04-12 阅读 3869
- TMS320C6713烧写Flash… 2014-03-30 阅读 5566

更多文章

猜你在找

iOS8-Swift开发教程
数据结构和算法
C/C++单元测试培训
深入浅出Unity3D——第一篇
从此不求人:自主研发一套PHP前端开发框架
Kalman滤波器从原理到实现
kalman滤波器从原理到实现
Kalman Filter - Kalman滤波器从原理到实现
从卡尔曼滤波到粒子滤波 很详细很明了
CISCO技术17万

查看评论

30楼 cx1806908579 2016-05-30 11:01发表

谢谢博主，写的很好

29楼 qq_28620207 2016-05-06 19:22发表

28楼 IT男的成长记录 2016-01-22 17:39发表

想问一下C程序中的kalman2_filter函数里面的

```
/* p(n|n-1)=A^2*p(n-1|n-1)+q */
state->p[0][0] = state->A[0][0] * state->p[0][0] + state->A[0][1] * state->p[1][0] + state->q[0];
state->p[0][1] = state->A[0][0] * state->p[0][1] + state->A[1][1] * state->p[1][1];
state->p[1][0] = state->A[1][0] * state->p[0][0] + state->A[0][1] * state->p[1][0];
state->p[1][1] = state->A[1][0] * state->p[0][1] + state->A[1][1] * state->p[1][1] + state->q[1];
```

和P(n|n-1)=AP(n-1|n-1)A^T+Q不一致吧，感觉没有乘A^T，但又不知道p[0][1]和p[1][0]是怎么算出来的，请指教。。

27楼 zhihaiwang 2016-01-12 20:50发表

正学习，讲的很好。

26楼 bqrmt 2016-01-06 15:08发表

楼主您好，看了你的帖子，推了一下，以下几个地方感觉好像和公式不一样

1、

> 引用"lktli"的评论：
>
> P(n|n-1)=[1 0; 0 0][1 1; 0 1][1 0; 0 0]=[1 0; 0 0]...

2、h(n)好像不正确，应该为[1 0]，不然最后x(n)计算出来应该是56

3、最后的x(n)的计算结果也不对，应该是[60;10]+[1/2;0]*{62-[1 0][60; 10]}=61，而不是60.5

如果我理解有误，请楼主见谅

Re: IT男的成长记录 2016-01-22 17:03发表

回复bqrmt：我也觉得有点问题，这几天看到的论文里面都是X：[position,velocity]，Z：[position],所以H应该为[1,0]。

统计知识快忘光了，看得我头疼，想问一下你P(n-1|n-1)(公式2里面的那个)是怎么算的啊，我不会算。。。

Re: bqrmt 2016-02-22 17:32发表

回复u012041225：P(n-1|n-1)是上一轮的计算结果啊

25楼 laismith 2015-12-22 21:23发表

最后实验结果卡尔曼对有突变噪声的响应，解释我觉得换成，在特使情况下测量信号的噪声的方差大小不是一个常熟，所以卡尔曼的效果并不理想，本质上还是因为卡尔曼应用的场景就是线性滤波。个人理解欢迎指正

24楼 lktli 2015-12-10 14:09发表

(8)是否正确？等式右边应该去掉系数A吧？

Re: xiahouzuoxin 2015-12-11 10:46发表

回复lktli：嗯，对的，谢谢提醒。下面的 @一实相印 提醒了，所以当时没改过来，现在注明了

23楼 lktli 2015-12-10 13:58发表

P(n|n-1)=[1 0; 0 0][1 1; 0 1][1 0; 0 0]=[1 0; 0 0]；根据（6）式是否应该是P(n|n-1)=[1 1; 0 1][1 0; 0 0][1 0; 1 1] + 2？？

22楼 jr9910 2015-11-10 21:07发表

在Matlab仿真中，这个真值到底应该是12还是系统噪声驱动的状态模型演化的状态值？如果是后者，是不是越滤波出来的约接近博主的true才越好？

21楼 jr9910 2015-11-10 20:38发表

在Matlab仿真当中有个疑问，系统状态的真值到底是那个12，还是像博主程序中设置的被系统噪声驱动的状态？如果是后者，那么理应越跟着系统噪声驱动的状态波动，才越接近真值，越反映系统的实际情况才对。

20楼 anhuidaxue100 2015-08-12 13:51发表

写得好！

19楼 raymond_kwan 2015-07-22 18:44发表

支持一下。

18楼 kangpc 2015-07-08 11:18发表

你好，请问你在求得AR系数之后，结合部分数据和ＡＲ系数进行过外推吗？

17楼 叫我大蘑菇 2015-06-18 21:55发表

相当于,两个协方差一样的分布,用了kalman filter后, 估计出来一个斜方差只有原来一半的分布. 总感觉不对劲儿

16楼 叫我大蘑菇 2015-06-18 21:50发表

你好,按照公式(14), 估计值的协方差永远小于等于预测协方差. 这个结论合理吗? 有点不好理解.

15楼 villee1201 2015-06-17 10:42发表

请问怎么确定q和r参数啊？需要对数据怎么处理？

14楼 EthanGreen 2015-06-10 09:51发表

写的深入浅出的，值得推荐！

13楼 opencv_yys 2015-06-03 09:35发表

在小车Example中，Kalman滤波器的最终输出是56m,楼主在带入小车速度时出现了笔误，小车匀速10m/s，不是1m/s.感谢楼主分享。文章很好。

12楼 opencv_yys 2015-06-03 09:30发表

在小车Example中，Kalman滤波器的最终输出为56m,而不是60.5m。楼主把小车速度带入错误，匀速10m/s,不是1m/s。感谢楼主的分享。

11楼 bian12chengwang 2015-05-26 16:16发表

群主 问下 观测噪声的自协方差矩阵一直不变的吗？在Kalman迭代的时候？

10楼 ZZDL123 2015-05-22 17:14发表

不错

9楼 FreeApe 2015-04-24 11:36发表

大赞楼主！

8楼 u010488386 2015-04-10 09:49发表

为什么状态矢量和观测矢量维数不同呢？谢谢

Re: u010488386 2015-04-10 10:00发表

回复u010488386：是因为，状态矢量包括上一时刻到当前时刻的变化量，而观测量只有当前时刻的状态，例如上面的匀速直线运动，状态矢量有位移速度而观测量只有位移

Re: u010488386 2015-07-20 16:29发表

回复u010488386：谢谢，明白了呢

7楼 一实相印 2015-03-04 10:42发表

写的还行，公式8多乘了个A。

Re: liuxueuestc 2015-05-14 14:29发表

回复zhouyy858：确实，好眼力

6楼 qq_26259465 2015-03-02 11:35发表

xiahouzuoxin您好！
Input argument "s" is undefined.
Error in ==> kalmanf at 150
if ~isfield(s,'x'); s.x=nan*z; end
请问是怎么回事？
望不吝赐教！！

Re: sinat_27680127 2016-03-23 18:19发表

回复qq_26259465：？？

Re: sinat_27680127 2016-03-23 18:18发表

回复qq_26259465：你的问题解决了么

Re: xiahouzuoxin 2015-03-02 19:02发表

回复qq_26259465：输入参数s没有定义的意思，你看看你的函数调用方式是不是有误，s = kalmanf(s)

5楼 amang2014 2015-01-15 11:59发表

讲的挺好的，谢谢了

4楼 JasonLeaster 2014-11-18 12:25发表

赞！
LZ方便留个联系方式交流讨论嘛？

Re: xiahouzuoxin 2014-11-18 12:40发表

回复u011368821：QQ:1126804077，共同学习，加Q时请注明CSDN，谢谢！

3楼 microjunior 2014-11-12 08:44发表

好东西

2楼 zhufeng125 2014-10-11 20:51发表

发现一个小错误~
一场地推的游戏__K(n)：比重越大，说明真值接近预测值的概率越大（应该是越小，比重越大，越接近测量值，比重最大是1，1，1的时候，X（n）就是Z（n））。
用小车运动来入门卡尔曼滤波的例子，简直太棒了！~
其余写得都很好！

Re: zhufeng125 2014-10-12 21:05发表

回复u013172073：1. void kalman2_init(kalman2_state *state, float *init_x, float (*init_p)[2])中最后两个参数是不是有点小问题，是否该改为：
void kalman2_init(kalman2_state *state, float *init_x[], float (*init_p)[2][])。
2. float kalman2_filter(kalman2_state *state, float z_measure);
return state->x[0];
是否可返回x[0]，x[1]组成的数组，这样状态就完整了，相应的修改是：
float * kalman2_filter(kalman2_state *state, float z_measure);
return state->x[2];

Re: xiahouzuoxin 2014-10-12 22:19发表

回复u013172073：嗯，你说的是对的，如果你需要对速度滤波的话。

我这里没加是因为这里只考虑对位移变量进行滤波，所以我在程序注释中说了，
NOTES: n Dimension means the state is n dimension, measurement always 1 dimension
我的状态变量是2维的，输出变量是1维（就是只用了位移）的。呵呵，可能说得有些不清楚！

Re: zhufeng125 2014-10-13 20:15发表

回复xiahouzuoxin：了解~

Re: xiahouzuoxin 2014-10-12 08:05发表

回复u013172073：你好，谢谢指正，错误已修正！
椒盐噪声就是脉冲噪声（比如黑白噪点），图像处理领域习惯性称之为椒盐噪声。因为Kalman滤波具有跟踪测量值的特性，所以在有脉冲噪声的地方会突然出现一个尖峰，其实这可能并不是我们想要的，中值滤波一般能对椒盐噪声起到很好的效果。

Re: zhufeng125 2014-10-12 19:43发表

回复xiahouzuoxin：了解~!

Re: zhufeng125 2014-10-11 21:02发表

回复u013172073：椒盐噪声是什么东东？

1楼 javafile_null 2014-10-01 22:13发表

这头像不是萧大侠吗

Re: xiahouzuoxin 2014-10-04 23:42发表

回复javafile_null：真识货！！

Re: javafile_null 2014-10-14 22:18发表

回复xiahouzuoxin：萧大侠 俺的偶像

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker
OpenStack VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC
WAP jQuery BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML
LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra
CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App
SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP
HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved