**使用卡尔曼滤波进行跟踪**

**一、理论**

卡尔曼滤波器其实是为了解决这样一个问题：为了准确获得系统的当前状态，可以使用模型计算预测的方法，但由于该模型与实际是有偏差的，因此预测的结果是不准确的。也可以使用测量的方法，但测量系统也是有偏差的，测量的结果同样也不准确。卡尔曼滤波就是结合预测值和测量值来计算当前状态的最优估计量，及其对应的偏差。卡尔曼滤波的5个基本公式中，1、2式分别计算了模型预测结果和模型预测偏差，3式将预测结果和测量结果结合计算当前状态的最优估计，4式为卡尔曼增益，它在5式中被用到，5式计算最优估计的偏差，这在下一步的迭代中会用到。  
    首先，我们先要引入一个离散控制过程的系统。该系统可用一个线性随机微分方程来描述：

X(k)=A X(k-1)+B U(k)+W(k)

再加上系统的测量值：

Z(k)=H X(k)+V(k)

    上两式子中，X(k)是k时刻的系统状态，U(k)是k时刻对系统的控制量。A和B是系统参数，对于多模型系统，他们为矩阵。 Z(k)是k时刻的测量值，H是测量系统的参数，对于多测量系统，H为矩阵。W(k)和V(k)分别表示过程和测量的噪声。他们被假设成高斯白噪声 (White Gaussian Noise)，他们的covariance 分别是Q，R（这里我们假设他们不随系统状态变化而变化）。  
    对于满足上面的条件(线性随机微分系统，过程和测量都是高斯白噪声)，卡尔曼滤波器是最优的信息处理器。下面我们来用他们结合他们的covariances 来估算系统的最优化输出。  
    首先我们要利用系统的过程模型，来预测下一状态的系统。假设现在的系统状态是k，根据系统的模型，可以基于系统的上一状态而预测出现在状态：

X(k|k-1)=A X(k-1|k-1)+B U(k) ……….. (1)

式(1)中，X(k|k-1)是利用上一状态预测的结果，X(k-1|k-1)是上一状态最优的结果，U(k)为现在状态的控制量，如果没有控制量，它可以为0。  
    到现在为止，我们的系统结果已经更新了，可是，对应于X(k|k-1)的covariance还没更新。我们用P表示covariance：

P(k|k-1)=A P(k-1|k-1) A’+Q ……… (2)

式(2)中，P(k|k-1)是X(k|k-1)对应的covariance，P(k-1|k-1)是X(k-1|k-1)对应的 covariance，A’表示A的转置矩阵，Q是系统过程的covariance。式子1，2就是卡尔曼滤波器5个公式当中的前两个，也就是对系统的预测。

现在我们有了现在状态的预测结果，然后我们再收集现在状态的测量值。结合预测值和测量值，我们可以得到现在状态(k)的最优化估算值X(k|k)：

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

其中Kg为卡尔曼增益(Kalman Gain)：

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

到现在为止，我们已经得到了k状态下最优的估算值X(k|k)。但是为了要另卡尔曼滤波器不断的运行下去直到系统过程结束，我们还要更新k状态下X(k|k)的covariance：

P(k|k)=（I-Kg(k) H）P(k|k-1) ……… (5)

其中I 为1的矩阵，对于单模型单测量，I=1。当系统进入k+1状态时，P(k|k)就是式子(2)的P(k-1|k-1)。这样，算法就可以自回归的运算下去。

卡尔曼滤波器的原理基本描述了，式子1，2，3，4和5就是他的5 个基本公式。根据这5个公式，可以很容易的实现计算机的程序。

下面的例子说明了卡尔曼滤波器。

假设我们要研究的对象是一个房间的温度。根据你的经验判断，这个房间的温度是恒定的，也就是下一分钟的温度等于现在这一分钟的温度（假设我们用一分钟来做时间单位）。假设你对你的经验不是100%的相信，可能会有上下偏差几度。我们把这些偏差看成是高斯白噪声（White Gaussian Noise），也就是这些偏差跟前后时间是没有关系的而且符合高斯分配（Gaussian Distribution）。另外，我们在房间里放一个温度计，但是这个温度计也不准确的，测量值会比实际值偏差。我们也把这些偏差看成是高斯白噪声。

好了，现在对于某一分钟我们有两个有关于该房间的温度值：你根据经验的预测值（系统的预测值）和温度计的值（测量值）。下面我们要用这两个值结合他们各自的噪声来估算出房间的实际温度值。

假如我们要估算k时刻的是实际温度值。首先你要根据k-1时刻的温度值，来预测k时刻的温度。因为你相信温度是恒定的，所以你会得到k时刻的温度预测值是跟k-1时刻一样的，假设是23度，同时该值的高斯噪声的偏差是5度（5是这样得到的：如果k-1时刻估算出的最优温度值的偏差是3，你对自己预测的不确定度是4度，他们平方相加再开方，就是5）。然后，你从温度计那里得到了k时刻的温度值，假设是25度，同时该值的偏差是4度。

由于我们用于估算k时刻的实际温度有两个温度值，分别是23度和25度。究竟实际温度是多少呢？相信自己还是相信温度计呢？究竟相信谁多一点，我们可以用他们的covariance来判断。因为Kg^2=5^2/(5^2+4^2)，所以Kg=0.78，我们可以估算出k时刻的实际温度值是：23+0.78\*(25-23)=24.56度。可以看出，因为温度计的covariance比较小（比较相信温度计），所以估算出的最优温度值偏向温度计的值。

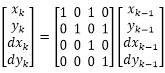
现在我们已经得到k时刻的最优温度值了，下一步就是要进入k+1时刻，进行新的最优估算。到现在为止，好像还没看到什么自回归的东西出现。对了，在进入k+1时刻之前，我们还要算出k时刻那个最优值（24.56度）的偏差。算法如下：((1-Kg)\*5^2)^0.5=2.35。这里的5就是上面的k时刻你预测的那个23度温度值的偏差，得出的2.35就是进入k+1时刻以后k时刻估算出的最优温度值的偏差（对应于上面的3）。

就是这样，卡尔曼滤波器就不断的把covariance递归，从而估算出最优的温度值。他运行的很快，而且它只保留了上一时刻的covariance。上面的Kg，就是卡尔曼增益（Kalman Gain）。

**二、实例**

         下面的是一个追踪鼠标的例子。鼠标在屏幕上的位置是一个点，同时，它还有速度，因此可以使用( x, y, dx, dy )来描述一个鼠标的状态，一共有4个元素。鼠标的位置是可以测量的，为(x , y)，一共有两个元素。

在预测阶段，根据前一状态计算后一状态的方程（系统模型）为：



其中的4×4矩阵称为转移矩阵。当然该矩阵个元素不一定是1和0，而是根据实际模型确定的。

鼠标位置作为测量值，测量方程（测量模型）为：

C:\Users\寅\Desktop\2.JPG

其中的单位矩阵是测量矩阵，当然，该矩阵不一定是单位矩阵，而是根据实际情况确定的，由于这个例子中，鼠标位置是可以直接测量得到的，因此测量方程如上所示。

在OpenCV中使用卡尔曼滤波器使用类KalmanFilter，在构造函数中传入状态元素数量、测量元素数量，控制元素的数量。除此外还要设置转移矩阵、测量矩阵、过程噪声协方差矩阵，测量噪声协方差矩阵，后验错误协方差矩阵。

使用KalmanFilter的predict函数对鼠标状态进行预测，得到的预测结果为位置和速度，它是一个4行1列的矩阵。得到预测值后还要结合测量值进行矫正，调用KalmanFilter的correct方法，重复以上过程进行迭代。

KalmanFilter类成员变量意义如下：

Mat statePre;           //!< predicted state (x'(k)): x(k)=A\*x(k-1)+B\*u(k)

Mat statePost;          //!< corrected state (x(k)): x(k)=x'(k)+K(k)\*(z(k)-H\*x'(k))

Mat transitionMatrix;   //!< state transition matrix (A)

Mat controlMatrix;      //!< control matrix (B) (not used if there is no control)

Mat measurementMatrix;  //!< measurement matrix (H)

Mat processNoiseCov;    //!< process noise covariance matrix (Q)

Mat measurementNoiseCov;//!< measurement noise covariance matrix (R)

Mat errorCovPre; /!< priori error estimate covariance matrix (P'(k)): P'(k)=A\*P(k-1)\*At + Q)\*/

Mat gain;               //!< Kalman gain matrix (K(k)): K(k)=P'(k)\*Ht\*inv(H\*P'(k)\*Ht+R)

Mat errorCovPost;  //!< posteriori error estimate covariance matrix (P(k)): P(k)=(I-K(k)\*H)\*P'(k)

代码：

#include "stdafx.h"

#include <opencv2\opencv.hpp>

#include <vector>

#include <iostream>

#include <sstream>

using namespace std;

using namespace cv;

const int winHeight = 512;

const int winWidth = 512;

Point mousePos = Point(winWidth >> 1, winHeight >> 1);

void mouseEvent(int event, int x, int y, int flags, void\*)

{

    if (event == CV\_EVENT\_MOUSEMOVE)

    {

        mousePos = Point(x, y);

    }

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

    CvFont font;

    cvInitFont(&font, CV\_FONT\_HERSHEY\_SCRIPT\_COMPLEX, 1, 1);

    const int stateNum = 4;

    const int measureNum = 2;

    KalmanFilter kf(stateNum, measureNum, 0);

    Mat meatureMent(measureNum, 1, CV\_32FC1);

    RNG rng = RNG(-1);

    float A[stateNum][stateNum] = {//transition matrix

        1, 0, 1, 0,

        0, 1, 0, 1,

        0, 0, 1, 0,

        0, 0, 0, 1

    };

    kf.transitionMatrix = Mat(stateNum, stateNum, CV\_32FC1, A);

    setIdentity(kf.measurementMatrix, 1.0);

    setIdentity(kf.processNoiseCov, Scalar::all(1e-5));

    setIdentity(kf.measurementNoiseCov, Scalar::all(1e-1));

    setIdentity(kf.errorCovPost, Scalar::all(1));

    Mat img = Mat(Size(winWidth, winHeight), CV\_8UC3, Scalar::all(255));

    imshow("RESUTL", img);

    setMouseCallback("RESUTL",mouseEvent);

    while (1)

    {

        //kalman预测值

        Mat predic = kf.predict();

        Point predic\_pt = Point(predic.at<float>(0,0), predic.at<float>(1,0));

        //测量值

        meatureMent.at<float>(0,0) = mousePos.x;

        meatureMent.at<float>(1,0) = mousePos.y;

        //更新

        kf.correct(meatureMent);

        //更新后的值

        Point post\_pt = Point(kf.statePost.at<float>(0, 0), kf.statePost.at<float>(1, 0));

        //绘制结果

        img = Mat(Size(winWidth, winHeight), CV\_8UC3, Scalar::all(255));

        circle(img, predic\_pt, 5, Scalar(0, 255, 0), 3);

        circle(img, mousePos, 5, Scalar(255, 0, 0), 3);

        circle(img, post\_pt, 5, Scalar(0 ,0 , 255), 3);

        stringstream ss;

        string text;

        ss << "Current\_Pos(" << mousePos.x << "," << mousePos.y << ")" << endl;

        ss >> text;

        putText(img, text, Point(10, 60), 1, 1, Scalar(255, 0, 0));

        ss.clear();

        text.clear();

        ss << "Predict\_Pos(" << predic\_pt.x << "," << predic\_pt.y << ")" << endl;

        ss >> text;

        putText(img, text, Point(10, 80), 1, 1, Scalar(0, 255, 0));

        ss.clear();

        text.clear();

        ss << "Post\_Pos(" << post\_pt.x << "," << post\_pt.y << ")" << endl;

        ss >> text;

        putText(img, text, Point(10, 100), 1, 1, Scalar(0, 0, 255));

        imshow("RESUTL", img);

        int key = waitKey(3);

        if (key == 27)

            break;

    }

    return 0;

}

