基于误差状态卡尔曼滤波的姿态融合检测及matlab代码

[1. 误差状态卡尔曼滤波器 2](#_Toc28103)

[2. 系统方程 3](#_Toc14582)

[(1) 状态变量 3](#_Toc11381)

[(2) imu的测量值 3](#_Toc67)

[(3) 预测方程及雅克比矩阵 3](#_Toc12715)

[3. kalman filter loop计算 6](#_Toc12269)

[4. Matlab代码 6](#_Toc29412)

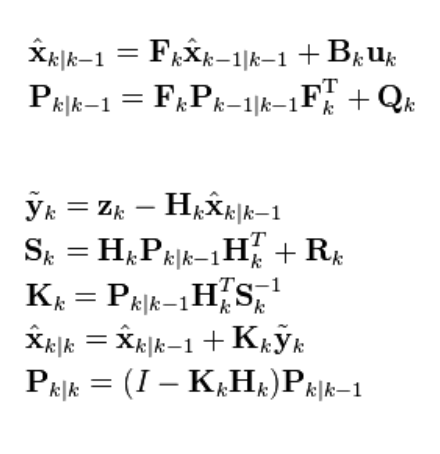
[5. 代码下载链接 10](#_Toc20792)

[6. 参考文献​ 10](#_Toc24389)

# 误差状态卡尔曼滤波器

误差状态卡尔曼滤波器(ESKF,即Error-State Kalman Filter），和一般的KF、EKF没有什么本质的区别，只是状态量的选取不同罢了。

以位姿估计为例，普通的kalman filter的状态量（名义状态量）一般都是位置p、速度v、四元数q，而ESKF的状态量（误差状态量）是上述状态量的误差，如位置的误差δp、速度误差δv、姿态误差δq，而kalman得5个核心公式是没有区别的。kalman滤波的主要公式表达如下：



​ 传统的滤波方法，只能是在有用信号与噪声具有不同频带的条件下才能实现．20世纪40年代，N．维纳和A．H．柯尔莫哥罗夫把信号和噪声的统计性质引进了滤波理论，在假设信号和噪声都是平稳过程的条件下，利用最优化方法对信号真值进行估计，达到滤波目的，从而在概念上与传统的滤波方法联系起来，被称为维纳滤波。这种方法要求信号和噪声都必须是以平稳过程为条件。60年代初，卡尔曼(R．E．Kalman)和布塞(R． S．Bucy)发表了一篇重要的论文《线性滤波和预测理论的新成果》，提出了一种新的线性滤波和预测理论，被称之为卡尔曼滤波。特点是在线性状态空间表示的基础上对有噪声的输入和观测信号进行处理，求取系统状态或真实信号。

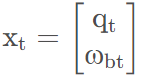
这种理论是在时间域上来表述的，基本的概念是：在线性系统的状态空间表示基础上，从输出和输入观测数据求系统状态的最优估计。这里所说的系统状态，是总结系统所有过去的输入和扰动对系统的作用的最小参数的集合，知道了系统的状态就能够与未来的输入与系统的扰动一起确定系统的整个行为。

卡尔曼滤波不要求信号和噪声都是平稳过程的假设条件。对于每个时刻的系统扰动和观测误差（即噪声），只要对它们的统计性质作某些适当的假定，通过对含有噪声的观测信号进行处理，就能在平均的意义上，求得误差为最小的真实信号的估计值。因此，自从卡尔曼滤波理论问世以来，在通信系统、电力系统、航空航天、环境污染控制、工业控制、雷达信号处理等许多部门都得到了应用，取得了许多成功应用的成果。例如在图像处理方面，应用卡尔曼滤波对由于某些噪声影响而造成模糊的图像进行复原。在对噪声作了某些统计性质的假定后，就可以用卡尔曼的算法以递推的方式从模糊图像中得到均方差最小的真实图像，使模糊的图像得到复原。

# **系统方程**

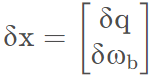
## 状态变量

1）真实状态量：



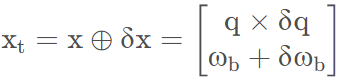
其中，*qt*为真实四元数，*ωbt*为陀螺仪的真实bias

2）状态量（名义状态量）：

，​

δq，δ*ωb*​分别为真实状态量和名义状态量的误差。

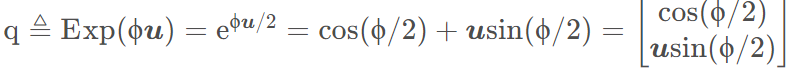
那么他们之间有这样的关系：



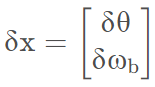
其中，δq和δθ关系为：



由于：



也就是常用的轴角公式，由于δ θ \delta\thetaδθ只需要3个变量表示，所以误差状态量又可以表示为更简洁的形式：



## imu的测量值

1）加速度计

at = am + an

其中at为真值，am为测量值，an为测量噪声。​

2）陀螺仪

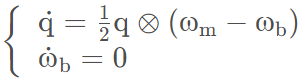


其中, ωt为真值，ωm为测量值，ωn为测量噪声。

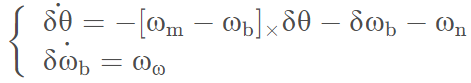
## 预测方程及雅克比矩阵

这里我们是用陀螺仪的数据对姿态进行预测。

对于名义状态量，有：



对于误差状态量，有：



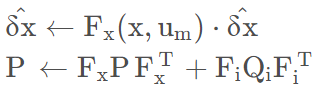
上式中，ωn为陀螺仪数据噪声，ωω为陀螺仪零偏的噪声（象征陀螺仪零偏的稳定性）；[∙]×为反对称矩阵。

这里，我们把误差状态的系统方程的一般形式表示为：

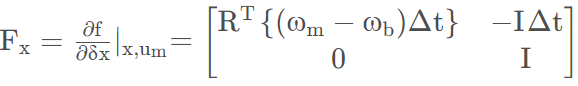


上式中，um为系统的输入，i为噪声。​

那么其预测方程可以写为：



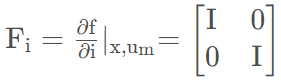
其中:

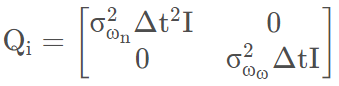


上式中，









需要注意的是,δx不需要做预测，因为每次初始化δx都为0，所以预测值肯定也是0。

有人会问，它会一直是0吗，经过几次运算迭代一直是0？

如果有这样的疑问，那就说明还没有从常规kalman filter的模式中走出来。ESKF每次迭代后，都会将最优估计得到的δx叠加到x中，所以对于下一时刻的开始，认为x是没有误差的，也就是认为δx为0了。

总结，在kalman预测这一步，每一轮新的kalman迭代，需要做的事情是：

1）对名义状态量进行预测；

3）计算雅克比矩阵

2）对误差状态量进行预测；（δ x \delta xδx的值是0，可以不计算，但是不能认为它没有意义）

2.4 测量方程及雅克比矩阵

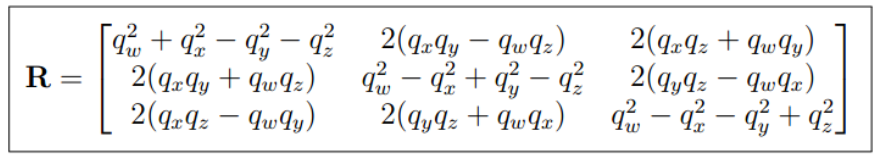
观测方程可以表示为下面的形式



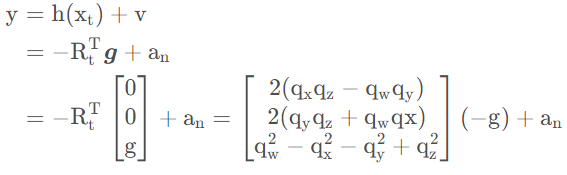
这里使用imu的加速度进行姿态的校正，假设的前提是没有机动加速度，只有重力加速度，所以有:



其中是通过qt得到的旋转矩阵的转置。

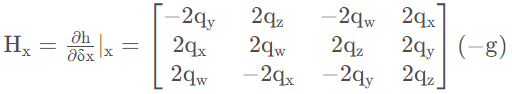


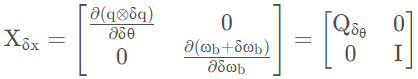
带入RT，可以把上式化简为：

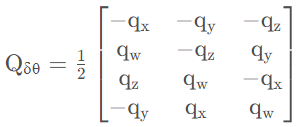


然后将h(xt)对δx求偏导，即



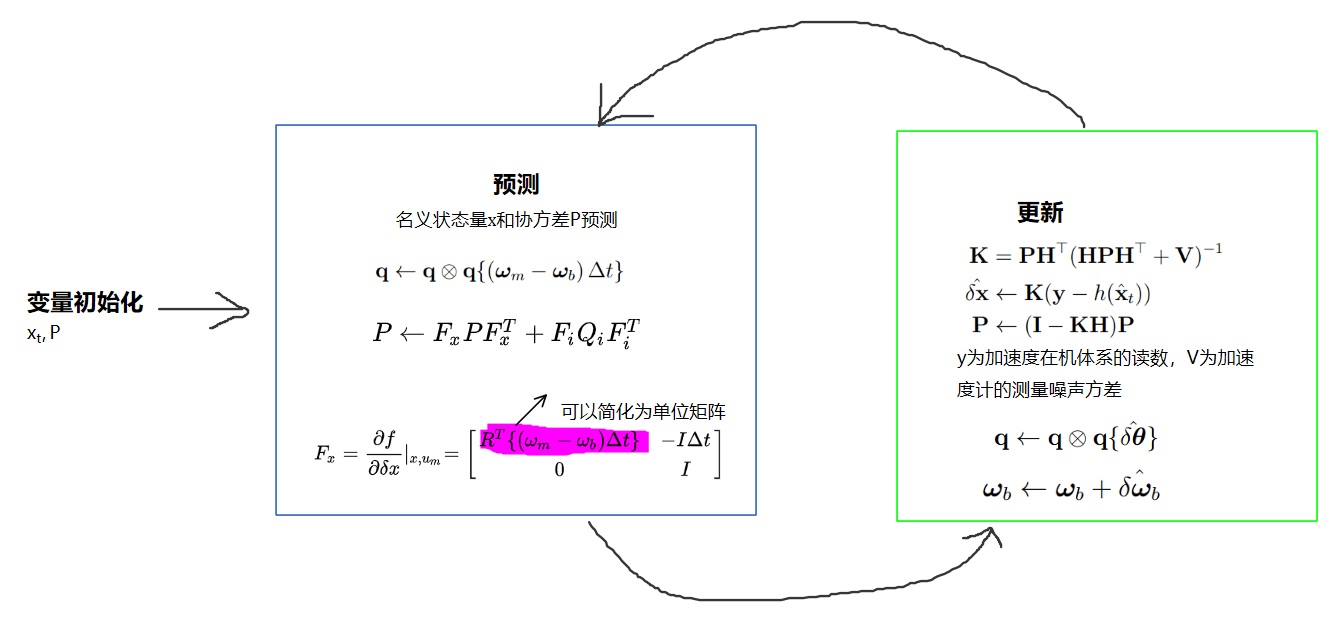






# **kalman filter loop计算**

进入算法实践环节，接下来就可以按照Kalman Filter的“套路”计算了，大致的流程如下。

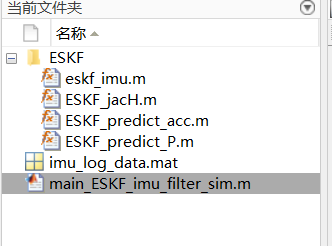


上图中的每一个公式或者变量都可以在前文中找到定义和说明，你要做的就是按照上述的流程进行计算就行。

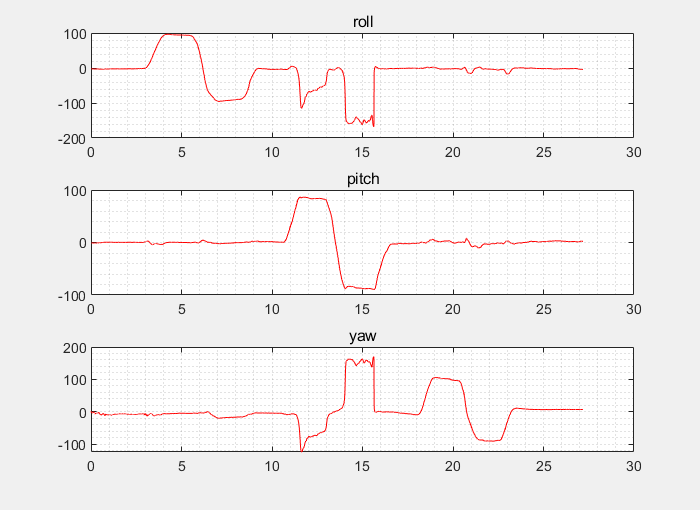
# **Matlab代码**

有人说，没有代码代码，还是太抽象，好吧，这里奉上MATLAB脚本和仿真数据。

代码文件的结构如下。



仿真结果如下。



%% 使用IMU的数据，基于ESKF的方法估计姿态

clc;clear;close;

%% 添加路径，加载数据

addpath('ESKF')

load('imu\_log\_data.mat');

mean\_dt = mean(diff(imu.t));

%% 调用算法，记录数据

eskf.t = imu.t;

[eskf.quat, eskf.gyb, eskf.cov] = eskf\_imu(imu.t, imu.gyr, imu.acc);

[eskf.yaw, eskf.pitch, eskf.roll] = quat2angle(eskf.quat,'ZYX');

%% 画图

figure('Name', 'euler in degree');

h1 = subplot(3,1,1);

plot(eskf.t, rad2deg(eskf.roll), 'r');

grid minor;title('roll')

h2 = subplot(3,1,2);

plot(eskf.t, rad2deg(eskf.pitch), 'r');

grid minor;title('pitch')

h3 = subplot(3,1,3);

plot(eskf.t, rad2deg(eskf.yaw), 'r');

grid minor;title('yaw')

linkaxes([h1 h2 h3],'x');​

function [state\_quat, state\_gyb, state\_cov] = eskf\_imu(t, gyr, acc)

%% 参数初始化

quat = quaternion(1,0,0,0);

init\_quat\_var = power(4, 2);

init\_gyb\_var = power(1e-3, 2);

% 初始化P矩阵

P(1:3,1:3) = init\_quat\_var \* eye(3);

P(4:6,4:6) = init\_gyb\_var \* eye(3);

acc\_meas\_noise\_var = power(2, 2); % 加速度计噪声方差

R = eye(3) \* acc\_meas\_noise\_var;

gyr\_noise = power(1e-2, 2);

gyr\_drift\_noise = power(1e-3, 2);

gyb = zeros(1,3);

%% 申请内存，可以加快仿真速度

len = length(gyr);

state\_quat = zeros(len,4);

state\_gyb = zeros(len,3);

state\_cov = zeros(len,6);

%% 循环计算

for index = 1:len

if (index == 1)

dt = saturate\_user(t(2) - t(1),0.001,0.05);

else

dt = saturate\_user(t(index) - t(index-1),0.001,0.05);

end

% ---- 预测 ----- %

% 预测姿态角

d\_angle = (gyr(index,:) - gyb) \* dt;

dq = quaternion(d\_angle, 'rotvec'); % 将旋转向量转化为四元数

new\_quat = quat \* dq;

quat = new\_quat.normalize;

% 预测协方差

P = ESKF\_predict\_P(dt, P, gyr\_noise, gyr\_drift\_noise);

% --- 更新 ----- %

y = ESKF\_predict\_acc(quat); %

H = ESKF\_jacH(quat);

S = H \* P \* H' + R;

K = P \* H' / S;

P = P - K \* H \* P;

innov = K \* (acc(index,:) - y)';

dq = quaternion(innov(1:3)', 'rotvec');

new\_quat = quat \* dq;

quat = new\_quat.normalize;

gyb = gyb + innov(4:6)';

state\_quat(index,:) = quat.compact;

state\_gyb(index,:) = gyb;

state\_cov(index,:) = diag(P)';

end

end

function out = saturate\_user(in, min, max)

if (in < min)

out = min;

elseif (in > max)

out = max;

else

out = in;

end

end​

function new\_P = ESKF\_predict\_P(dt, P, gyro\_noise, gyr\_drift\_noise)

Fx = eye(6);

Fx(1:3,4:6) = -eye(3) \* dt;

Fi = eye(6);

Qi(1:3, 1:3) = eye(3) \* gyro\_noise \* dt \* dt;

Qi(4:6, 4:6) = eye(3) \* gyr\_drift\_noise \* dt;

new\_P = Fx \* P \* transpose(Fx) + Fi \* Qi \* transpose(Fi);

end

function pred\_meas = ESKF\_predict\_acc(quat)

[qw, qx, qy, qz] = parts(quat);

gravity = 9.80665;

pred\_meas(1) = -gravity \* 2 \* (qx \* qz - qw \* qy);

pred\_meas(2) = -gravity \* 2 \* (qw \* qx + qy \* qz);

pred\_meas(3) = -gravity \* (qw \* qw - qx \* qx - qy \* qy + qz \* qz);

end

function H = ESKF\_jacH(quat)

[qw, qx, qy, qz] = parts(quat);

Hx = zeros(3,7);

Xdx = zeros(7,6);

Hx(1:3,1:4) = -9.80665 \* 2 \* [ -qy, qz, -qw, qx;...

qx, qw, qz, qy;...

qw, -qx, -qy, qz];

Xdx(1:4,1:3) = 0.5 \* [-qx, -qy, -qz;...

qw, -qz, qy;...

qz, qw, -qx;...

-qy, qx, qw];

Xdx(5:7,4:6) = eye(3);

H = Hx \* Xdx;

end

如果还有什么不清楚，欢迎评论区交流。

# 代码下载链接

可以在这里下载。链接: 点击下载。

# 参考文献​

<https://blog.csdn.net/zhoupian/article/details/117236240>