EL41: Filtering and Navigation

TD3 : Hardware Implementation of Kalman Filter in Attitude Estimation

**Introduction:**

This TD will start from introducing an attitude measurement system which consist of accelerometers, gyros, magnetic sensors, FPGA (Field Programmable Gate Array) etc. Then we select the unit quaternion as the state vector and give one state space model of the system. Based on the model, a Kalman Filter is constructed using gyros to predict while accelerometers and magnetic sensors to update the measurement. This TD will show you how a Kalman filter is realized by hardware method.

**Objectives:**

1) Understand the structure of the attitude measurement system.

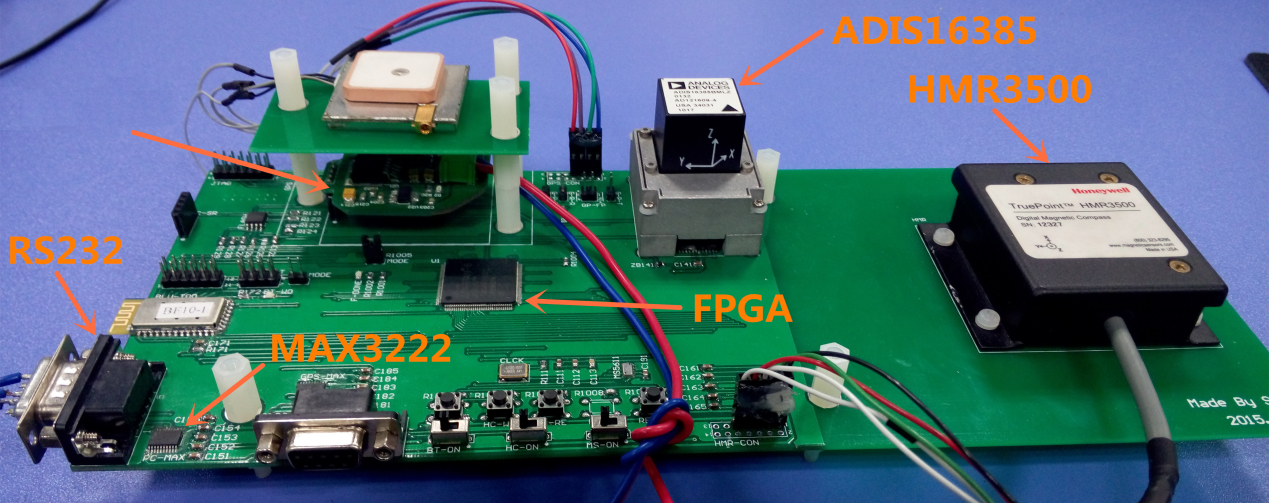
2) Design a Kalman filter with the given system to fuse the outputs of the gyros, the accelerometers and the magnetic sensor.

3) Compare the performance of the Kalman filter realized by software and by hardware

**Tasks:**

1) The attitude measurement system.

Fig 1 shows the construction of the attitude measurement system. The system takes the FPGA as controller and designs a set of small navigation platform. This platform includes sensors module, communication module and power module as in Fig 2. Auxiliary navigation sensors, such as magnetic sensors, are integrated in the hardware system for the sake of the accumulated error derived from INS.



**Module**

**Power**

Fig. 1 The attitude measurement system.

Sensors Module

External

Power

Source

Control and

Calculation Module

Power Module

Host

Computer

Communication

Module

Fig. 2 Module diagram of the attitude measurement system.

2) A state space model of the attitude measurement system.

We select the unit quaternion as the state vector *x*(*k*) and the observation vector *y*(*k*). Formula 1 is the dynamic equation. It describes the prediction of the attitude using gyros. Formula 2 is the observation equation. It gives the attitude observed by accelerometers and magnetic sensors.

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

A is the state transition matrix. H is the process noise coupling matrix.

Ck is the state output matrix. Γk is the output noise coupling matrix.

v1 and v2 are Gaussian white noise.

CONSIDER the form of A and Ck.

3) Design a Kalman filter with the given system in task 2.

AHRS

GYRO

ACC & MAG

Fig. 3 Principle of the Kalman filter.

CONSIDER the form of the five KF equations.

**Appendix I. Code in MATLAB.**

function [ att ] = kalmanQuater( iacc, iara, ihmr, iahr, sacc, sara, shmr, sahr, tim )

%UNTITLED Summary of this function goes here

% Detailed explanation goes here

% iacc = acceleration angle of ADIS;

% iara = angular rate of ADIS;

% ihmr = attitude of HMR;

% sacc = period of acc;

% sara = period of ang rate;

% shmr = period of HMR;

% aveacc = average of total acceleration;

% accerrjud = acceleration error judgement;

% magerrjud = magnetic field error judgement;

% tim = time for initial value;

% tec = periode for correction to acc ang;

format long;

% Time and frequency

Tacc=sacc;

Tara=sara;

Thmr=shmr;

Tahr=sahr;

facc=1/Tacc;

fara=1/Tara;

fhmr=1/Thmr;

fahr=1/Tahr;

iara=[iara(:,1:2) -iara(:,3)];

% Number

countInterval=round(Thmr/Tara);

% mar

marara=size(iara);

maracc=size(iacc);

marhmr=size(ihmr);

marahr=size(iahr);

% First attitude

att0=zeros(1,3);

linnum=fix(fahr\*tim);

for lin=1:1:linnum % First 10s is static, it could get attitude from acceleration angle

att0(1,1:3)=att0(1,1:3)+iahr(lin,1:3);

end

att0(1,1:3)=att0(1,1:3)/linnum;

% Position

P=[ 39.116332\*pi/180; 117.359522\*pi/180; 5];

% wien, wenn, winn, wibb, wnbb

wie=vpa(7.2921151467e-5); % earth rotation angular rate

wien=[ 0; wie\*cos(P(1)); wie\*sin(P(1)) ];

wenn=zeros(3,1);

winn=zeros(3,1);

winn=wien+wenn;

wibb=iara(:,1:3)';

wnbb=zeros(3,1);

% Cbn Attitude translation matrix: b->n

Cbn=zeros(3,3);

Cnb=zeros(3,3);

Cbn=attToCbn(att0);

Cnb=Cbn'

% Quaternion

quat=zeros(4,1);

quat=cbnToQua(Cbn); % Quaternion initialization

quat=quatNormal(quat); % Quaternion normalize

quatEst=quat; % Quaternion estimation

quatObs=zeros(4,1); % Observation quaternion

% Kalman filter

A=zeros(4,4); % State transition matrix

P=eye(4,4); % Estimation error autocorrelation matrix

Q=0.0001\*eye(4,4); % Gauss white noise autocorrelation matrix

R=10\*eye(4,4); % Gauss white noise autocorrelation matrix

P=eye(4,4); % Noise autocorrelation matrix

K=zeros(4,4); % Kalman gain

H=eye(4,4); % State output matrix

acceleLin=0; % Acceleration data line

hmrLin=0; % HMR data line

% Attitude

att=zeros(1,3); % Attitude

out=zeros(marara(1,1), 3);

attObs=zeros(1,3);

for lin=1:1:marara(1,1)

wnbb=wibb(:,lin)-Cnb\*winn;

angnbb=wnbb\*Tara;

detThe=angnbbToDetThe(angnbb);

A=eye(4,4)+0.5\*detThe;

hmrLin=fix(lin\*Tara/Thmr);

acceleLin=fix(lin\*Tara/Tacc);

if hmrLin==0

hmrLin=hmrLin+1;

elseif hmrLin > marhmr(1,1)

hmrLin=marhmr(1,1);

end

if acceleLin==0;

acceleLin=acceleLin+1;

elseif acceleLin>maracc(1,1)

acceleLin=maracc(1,1);

end

attObs(1,3)=ihmr(hmrLin,3);

attObs(1,1:2)=iacc(acceleLin,1:2);

Cbn=attToCbn(attObs);

Cnb=Cbn';

quatObs=cbnToQua(Cbn);

quatObs=quatNormal(quatObs);

quatEst=A\*quat; % First step: state estimation

P=A\*P\*A'+Q; % Second step: state estimation error

K=P\*H'\*(inv(H\*P\*H'+R)); % Third step: Kalman gain

quat=quatEst+K\*(quatObs-H\*quatEst); % Fourth step: state estimation

P=(eye(4,4)-K\*H)\*P; % Fifth step: state estimation error autocorrelation matrix

quat=quatNormal(quat);

Cbn=QuaToCbn(quat);

Cnb=Cbn';

att(lin,:)=CbnToAtt(Cbn);

lin

end

end

**Appendix II. Code in C.**

/\* Kalman attitude calculation \*/

**void** **attCalculKalmanFun**(**void**){

/\*-----------------------------------------------------------------\*/

/\* Definition \*/

Xfloat32 angDelnbb[3]={0.0F,0.0F,0.0F}; /\* Angle increment nb-b temporary \*/

/\* Matrix \*/

Xfloat32 delAngMat[4][4]=EYE4; /\* angDelnbb anti symmetric matrix \*/

Xfloat32 uniMat[4][4]=EYE4; /\* Unit matrix \*/

Xfloat32 A[4][4]=EYE4; /\* State transformation matrix \*/

Xfloat32 P[4][4]=EYE4; /\* Estimation error autocorrelation matrix \*/

Xfloat32 matTem1[4][4]=EYE4; /\* Temporary matrix \*/

Xfloat32 matTem2[4][4]=EYE4; /\* Temporary matrix \*/

Xfloat32 Q[4][4]={{0.0001F,0.0F,0.0F,0.0F}, /\* H\*Q\*H' for second step\*/

{0.0F,0.0001F,0.0F,0.0F},

{0.0F,0.0F,0.0001F,0.0F},

{0.0F,0.0F,0.0F,0.0001F}};

Xfloat32 R[4][4]={{10.0F,0.0F,0.0F,0.0F}, /\* T\*Q\*T' for third step \*/

{0.0F,10.0F,0.0F,0.0F},

{0.0F,0.0F,10.0F,0.0F},

{0.0F,0.0F,0.0F,10.0F}};

// Xfloat32 C[4][4]=EYE4; /\* Observation matrix \*/

Xfloat32 K[4][4]=EYE4; /\* Kalamn gain \*/

/\* Attitude \*/

Xfloat32 attObs[3]={0.0F,0.0F,0.0F}; /\* Attitude observed for Kalman filter \*/

/\* Quaternion \*/

Xfloat32 quaObs[4]={0.0F,0.0F,0.0F,0.0F}; /\* Quaternion observed \*/

Xfloat32 quaEst[4]={0.0F,0.0F,0.0F,0.0F}; /\* Quaternion estimated \*/

Xfloat32 quaTem1[4]={0.0F,0.0F,0.0F,0.0F}; /\* Quaternion temporary \*/

Xfloat32 quaTem2[4]={0.0F,0.0F,0.0F,0.0F}; /\* Quaternion temporary \*/

/\*-----------------------------------------------------------------\*/

/\* Data prepared \*/

/\* angDelnbb, delAngMat, A \*/

matAddFun(wnbb, wnbb, angDelnbb, PER\_W\_A, 0.0F, 1, 3); /\* angDelnbb \*/

vecToMat4(angDelnbb, delAngMat[0]); /\* delAngMat \*/

matAddFun(uniMat[0], delAngMat[0], A[0], 1.0F, 0.5F, 4, 4); /\* A \*/

/\* attObs, quaObs \*/

attCalByAccFun(aibbAve, attObs); /\* Attitude observation pitch and roll from acceleration \*/

\*(attObs+2)=\*(datArrHmr+2);

attToCbnFun(attObs, cbn[0]); /\* cbn observed \*/

cbnToQuaFun(cbn[0], quaObs); /\* quaObs \*/

quaNorFun(quaObs); /\* quaObs normalization \*/

/\*-----------------------------------------------------------------\*/

/\* Kalman step \*/

/\* First step: state estimation ---> quaEst \*/

matMulFun(A[0], qua, quaEst, 4, 4, 1); /\* quaEst=A\*qua \*/

/\* Second step: state estimation error ---> P \*/

matTraFun(A[0], matTem1[0], 4, 4); /\* matTem1=A' \*/

matMulFun(P[0], matTem1[0], matTem2[0], 4, 4, 4); /\* matTem2=P\*A' \*/

matMulFun(A[0], matTem2[0], matTem1[0], 4, 4, 4); /\* matTem1=A\*P\*A' \*/

matAddFun(matTem1[0], Q[0], P[0], 1.0F, 1.0F, 4, 4); /\* P \*/

/\* Third step: Kalman gain ---> K \*/

/\* C=I=C' \*/

matAddFun(P[0], R[0], matTem1[0], 1.0F, 1.0F, 4, 4); /\* matTem1=C\*P\*C'+R \*/

matInvFun(matTem1[0], matTem2[0], 4); /\* matTem2=inv(C\*P\*C'+R) \*/

matMulFun(P[0], matTem2[0], K[0], 4, 4, 4); /\* K Kalman gain \*/

/\* Fourth step: state estimation ---> qua \*/

matAddFun(quaObs, quaEst, quaTem1, 1.0F, -1.0F, 4, 1); /\* quaTem=quaObs-quaEst \*/

matMulFun(K[0], quaTem1, quaTem2, 4, 4, 1); /\* quaTem2=K\*(quaObs-quaEst) \*/

matAddFun(quaEst, quaTem2, qua, 1.0F, 1.0F, 4, 1); /\* qua=quaEst+K\*(quaObs-quaEst) \*/

/\* Fifth step: state estimation error autocorrelation matrix ---> P \*/

matAddFun(uniMat[0], K[0], matTem1[0], 1.0F, -1.0F, 4, 4); /\* matTem1=I-K\*C=I-K\*I= \*/

matMulFun(matTem1[0], P[0], matTem2[0], 4, 4, 4); /\* matTem2=(I-K\*C)\*P \*/

matAddFun(matTem2[0], matTem2[0], P[0], 1.0F, 0.0F, 4, 4); /\* P=(I-K\*C)\*P \*/

/\*-----------------------------------------------------------------\*/

/\* Calculating attitude \*/

quaNorFun(qua); /\* qua normalization \*/

quaToCbnFun(qua, cbn[0]); /\* cbn \*/

cbnToAttFun(cbn[0], attOut); /\* attOut \*/

\*attOut=(\*attOut)\*RAD\_TO\_DEG;

\*(attOut+1)=(\*(attOut+1))\*RAD\_TO\_DEG;

\*(attOut+2)=(\*(attOut+2))\*RAD\_TO\_DEG;

}