

```

%% Attitude Determination using EKF
%% Mohamed Temam Nasri
%% Matlab Implementation
%% University of Manitoba
%% Advisor: Prof. Witold Kinsner
%% October 23, 2012
%% Version 2.1
clc
clear all
close all

%% Load Reference Vector and Quaternions
load sun_inertial_00001.csv
load igrf_inertial_00001.csv
load sun_005_body_00001.csv
load igrf_005_body_00001.csv
load quaternion_bi_005_00001.csv
load quaternion_005_00001.csv

%%
tlefile = 'dtusat1.TLE';
tle = readTLE(tlefile);
[epoch_yr, epoch_m, epoch_d, epoch_h, epoch_min, epoch_s] = days2ymdhms(tle.epoch_year,
tle.epoch_day);
Ttdb = cal_Ttdb(epoch_yr, epoch_m, epoch_d, epoch_h, epoch_min, epoch_s);

jd1= juliandate(epoch_yr, epoch_m, epoch_d, epoch_h, epoch_min, epoch_s);

%% ***** Magnetic fField *****
%IGRF model initializations 21001.5
[G,H] = IGRF2005;      % IGRF coefficients for 2005
nmax = 13;              % max degree of geopotential
mmax = 13;              % max order of geopotential
Kschmidt = schmidt(nmax,mmax);

%% Defining Noise Terms
sigma_v1 = 0.001; % (rad) Angle sensor noise (sigma n Crassidis).
sigma_n = sigma_v1; % crassidis
r = sigma_v1^2;
R_k=[r*eye(3,3) zeros(3);zeros(3) r*eye(3,3) ];
sigma_v2 = 5*1.e-6; % (rad/sec^(1/2)) sampled rate sensor noise( gyro noise)
sigma_v = sigma_v2; % sigma v crassidis
sigma_u1 = 2*1.e-8; % (rad/sec^(3/2) bias drift
sigma_u = sigma_u1; % sigma u crassidis

```

```

%sigma1 = 0.0005; % Sun Sensor Noise(Unit Vector)
sigma1 = 0.001; % Sun Sensor Noise(Unit Vector)
sigma2 = 0.001; % Magnetometer Noise (Unit Vector)
sigma_tt=(1/sigma1^2)+(1/sigma2^2)
sigma_t=sqrt(1/sigma_tt)
%% Initializations
integration=0;
w_k_1=0;
p_quat=[];
p_c=[];
p_bias=[];
integration=0;

%% setting up Matrixes for Graphs
R=[];B=[];A=[];S=[];P=[];
co_var=[];co_var1=[];co_var2=[];
t=[0:.01:20];
bias1=[];bias2=[];bias3=[];
theta_a=[];

wka=[];
wk=[];

k1=[];
k2=[];
bias=[0.1;0.1;0.1]*10^-4;
%bias=[0;0;0]
bias_e1=[];
bias_e2=[];
bias_e3=[];
si_a=[];
phi_a=[];
t_sec=[];
theta=[];
q1=[];q2=[];q3=[];q4=[];
qe1=[];qe2=[];qe3=[];qe4=[];
qa1=[];qa2=[];qa3=[];qa4=[];

%% Initial Kalman using any initial value to q_k_1
[qs, co_var]=kalman_triad (0,jd1, 0);

%q_k_1=[0; 0; 0; 1];

q_k_1=[qs(1) ;qs(2) ;qs(3); qs(4)];

```

```
%% Initial Bias
```

```
b_k_1=[0 ;0; 0];  
w_k=[0.01 ; 0.01 ; 0.2];
```

```
%% Initial co variance matrix  
P_k_1 = [[.2e-3*eye(3) zeros(3)];  
         [zeros(3) 1.e-3*eye(3)]];
```

```
%% Sampling Time  
Ts=0.864;
```

```
for i=0:1:2000  
    sec=0000;  
    delta_sim=i*(0.0144);
```

```
%% *****Inertial Reference Vectors*****
```

```
rsun=[sun_inertial_00001(i+1,2) sun_inertial_00001(i+1,3) sun_inertial_00001(i+1,4)];
```

```
rsun=rsun/(sqrt(sun_inertial_00001(i+1,2)^2+sun_inertial_00001(i+1,3)^2+sun_inertial_00001(i  
+1,4)^2));
```

```
bvect=[igrf_inertial_00001(i+1,1) igrf_inertial_00001(i+1,2) igrf_inertial_00001(i+1,3)];
```

```
%% ***** Body Frame Reference Vectors*****
```

```
mag_meas_b=[igrf_005_body_00001(i+1,6) ;igrf_005_body_00001(i+1,7);  
igrf_005_body_00001(i+1,8)];
```

```
[Hterm_b resd_b dot_prod_m]=bvector_kalman_EKF(q_k_1,mag_meas_b,bvect);  
sun_meas_b=[sun_005_body_00001(i+1,2); sun_005_body_00001(i+1,3);  
sun_005_body_00001(i+1,4)];  
sun_meas_b=sun_meas_b/sun_005_body_00001(i+1,5);
```

```
[Hterm_s resd_s dot_prod_s] = sun_vector_EKF( q_k_1,sun_meas_b,rsun);
```

```
%% update the value of output matrix H  
resd = [resd_b; resd_s] ;
```

```

H_om = [[Hterm_b zeros(3)];
        [Hterm_s zeros(3)]];
H_om_T = H_om';
%EKF param
[PHI_k, PHI_k_T, Q_k] = ekfparams(w_k);
%% Predicted covariance matrix
M_k = PHI_k*P_k_1*PHI_k_T+Q_k;
P_k_1 = M_k;
%% Next cycle estimates

p_quat=[p_quat;P_k_1(1,1)]

p_bias=[p_bias;P_k_1(4,4)];
%% gain for step K
HMHTR = H_om*M_k*H_om_T+R_k;
K_k = (M_k*H_om_T)*(inv(HMHTR));

K_k_q = [K_k(1,:); K_k(2,:); K_k(3,:)];

K_k_b = [K_k(4,:); K_k(5,:); K_k(6,:)];

k1=[k1;K_k_q(1,1)];
k2=[k2;K_k_b(1,1)];
%% State update and error covariance update

P_update = (eye(6) - K_k*H_om)*M_k;

deltaq_up_red = K_k_q*resd; % Kq*resd
%
deltaq_up = [0.5*deltaq_up_red/sqrt(1-0.25*(deltaq_up_red'*deltaq_up_red))];

mag_deltaq_up = sqrt(deltaq_up'*deltaq_up);
deltaq_up = [deltaq_up_red; sqrt(1-(deltaq_up_red'*deltaq_up_red))];
q_update = quatprod(deltaq_up,q_k_1);

mag_q_up = sqrt(q_update'*q_update);
if(mag_q_up ~= 1)
    %normalize
    q_update = q_update/mag_q_up;
end

deltab_up = K_k_b*resd;
b_k = b_k_1+deltab_up;

```

```

bias_e1=[bias_e1;bias(1)-b_k(1)];
bias_e2=[bias_e2;bias(2)-b_k(2)];
bias_e3=[bias_e3;bias(3)-b_k(3)];

%% Read body rates
w_ka=[quaternion_005_00001(i+1,6);
quaternion_005_00001(i+1,8)];
w_ka=w_ka*pi/180;

%% Adding the noise to the Actual Gyro Readings
noise=(sigma_u+sigma_v)*randn([3,1]);

% noise=0;

w_k_1=w_ka+noise+bias;

wka=[wka; w_ka'];
wk=[wk; w_k_1'];

%Remove Estimated Bias from the Gyro readings
w_k_1=w_k_1-b_k;
% First order quaternion propagation equation
w_a = (w_k_1+w_k)/2;
OMEGA_w_a = omegaop(w_a);
OMEGA_w_kplus1 = omegaop(w_k_1);
OMEGA_w_k = omegaop(w_k);
%q_pred = (expm(0.5*OMEGA_w_a*Ts)+((48^
1)*((OMEGA_w_kplus1*OMEGA_w_k)-(OMEGA_w_k*OMEGA_w_kplus1))*Ts^2))*q;
wmag = sqrt(w_a'*w_a);
q_k_1 = ((cos(wmag*Ts/2)*eye(4)+((1/wmag)*sin(wmag*Ts/2)*OMEGA_w_a))+((48^
1)*((OMEGA_w_kplus1*OMEGA_w_k)-(OMEGA_w_k*OMEGA_w_kplus1))*Ts^2))*q_update;
%store a copy of current estimates before going to next time step
w_k = w_k_1;

%% Next Cycle Bias Propagation
b_k_1=b_k;
%% Error Calculation

q_actual=[quaternion_005_00001(i+1,2);quaternion_005_00001(i+1,3);quaternion_005_00001(i
+1,4);quaternion_005_00001(i+1,5)];

qe=[ q_actual(4) q_actual(3) -q_actual(2) q_actual(1);

```

```

        -q_actual(3) q_actual(4) q_actual(1) q_actual(2);
        q_actual(2) -q_actual(1) q_actual(4) q_actual(3);
        -q_actual(1) -q_actual(2) -q_actual(3) q_actual(4)]*[-q_update(1);
                                                -q_update(2);
                                                -q_update(3);
                                                q_update(4)];

        qe_mag=sqrt(qe(1)^2+qe(2)^2+qe(3)^2+qe(4)^2);
        qe=qe/qe_mag
%         if(qe(4)<0)
%             qe=-qe;
%         end

        [e1,e2,e3,a]=Quaternion_to_Axis(qe(1),qe(2),qe(3),qe(4));

        theta=[theta;a];

q1=[q1;q_update(1)];
q2=[q2;q_update(2)];
q3=[q3;q_update(3)];
q4=[q4;q_update(4)];

qa1=[qa1;q_actual(1)];
qa2=[qa2;q_actual(2)];
qa3=[qa3;q_actual(3)];
qa4=[qa4;q_actual(4)];

qe1=[qe1;qe(1)];
qe2=[qe2;qe(2)];
qe3=[qe3;qe(3)];
qe4=[qe4;qe(4)-1];

bias1=[bias1;b_k_1(1)];
bias2=[bias2;b_k_1(2)];
bias3=[bias3;b_k_1(3)];

        t_sec=[t_sec; i*0.0144];
end
error=0;
error1=[];

```

```

for j=1:1:length(t_sec)
    error=error+theta(j);
end
error=error/length(t_sec);
for j=1:1:length(t_sec)
    error1=[error1;error];
end

%% output for black/white printing
figure(1)
title(' Estimated and Actual Values');

subplot(2,1,1)
plot(t_sec,q3,'.')
hold on
plot(t_sec,qa3,'-')
axis([0 length(t_sec)*0.0144 -0.5 0.25]);
xlabel('Minutes')
ylabel('quaternion 3')

subplot(2,1,2)
plot(t_sec,q4,'.')
hold on
plot(t_sec,qa4,'-')
axis([0 length(t_sec)*0.0144 0 0.6]);
xlabel('Minutes')
ylabel('quaternion 4')

legend('Estimated','Actual')

figure(2)

title(' Estimated and Actual Values');
subplot(2,1,1)
plot(t_sec,q1,'.')
hold on
plot(t_sec,qa1,'-')
axis([0 length(t_sec)*0.0144 -0.7 -0.1]);
xlabel('Minutes')
ylabel('quaternion 1')

subplot(2,1,2)
plot(t_sec,q2,'.')

```

```
hold on
plot(t_sec,qa2,'-')
axis([0 length(t_sec)*0.0144 -0.9 -0.5]);
xlabel('Minutes')
ylabel('quaternion 2')
```

```
legend('Estimated','Actual')
```