Lista 02 - Parte 05

Autor: Francisco Castro

Objetivo

Alinhamento inicial de plataforma solidária e calibração de sensores inerciais com veículo estacionário em posição conhecida mediante implementação de filtro de Kalman linearizado (LKF) e estendido (EKF) com aumento do vetor de estado para estimar erro nos sensores e erro de velocidade terrestre – vetor de medidas concatena a velocidade terrestre computada pelo INS com magnetômetro.

Introdução

Selecionou-se uma latitude λ, altitude h e atitude verdadeira D_t_b para simular as medidas dos acelerômetros e girômetros

```
lat0 = 0;
lon0 = 0;
h = 600;
D_t_b = eye(3);
seed = 1;
```

Escolheu-se a atitude verdadeira do veículo como o NED de forma arbitrária, mas sem perda de generalidade, a fim de facilitar as contas e análises. Nada impede de escolher-se outra atitude verdadeira aqui e o restante do código acompanhará a mudança de acordo, bastando redefinir g_NED_t e Omega_NED_t de acordo.

Determinou-se as medidas dos acelerômetros e girômetros cujos parâmetros são como a seguir

```
% Girômetro
sigma_gir = 1e-1;           % dev pad de vetor seq. branca aditiva [grau/h]
deriva = 1e-1 * [1 1 1];    % vetor deriva [grau/h]

% Acelerômetro
sigma_acel = 1e1;           % dev pad de vetor seq. branca aditiva [mg]
bias = 100e-3 * [1 1 1];    % vetor bias [mg]
```

Tomou-se cuidado ao escolher os valores de bias e deriva para que os erros não cresçam rapidamente e terminem rendendo o modelo linearizado da dinâmica dos erros de INS inválido em um curto prazo de tempo.

Para estimar valores razoáveis para o bias e deriva dos sensores, baseou-se em uma pesquisa que levou em consideração, no final das contas, os valores sugeridos por Patrick Paranhos em sua tese acadêmica (Tabela 1, seção 2.2, disponível em <https://www.maxwell.vrac.puc-rio.br/15124/15124_3.PDF>.

Método TRIAD

Use TRIAD com as medidas simuladas durante uma janela de tempo curta para inicializar a atitude do INS antes de sua operação no modo de navegação integrado ao LKF.

```
% Definições de amostragem
t_TRIAD = 15;                       % Janela de tempo para inicialização [s]
t_EXP = 60;                       % Janela de tempo para simulação [s]
frequenciaAmostragem = 25;          % Hz (frequência de amostragem)
dt = 1/frequenciaAmostragem;        % s  (intervalo de tempo entre medidas)

% Estruturação do método
syms a_x a_y a_z w_x w_y w_z
Asp_t = [a_x ; a_y ; a_z];
Omega_t = [w_x; w_y; w_z];

Asp_b = D_t_b*Asp_t;
Omega_b = D_t_b*Omega_t;

syms g Omega lambda
Asp_NED = [0;0;-g];
Omega_NED = [Omega*cos(lambda); 0 ; -Omega*sin(lambda)];

A = [...
    Asp_NED/norm(Asp_NED), ...
    cross(Asp_NED,Omega_NED)/norm(cross(Asp_NED,Omega_NED)), ...
    cross(Asp_NED,cross(Asp_NED,Omega_NED))/norm(cross(Asp_NED,cross(Asp_NED,Omega_NED)))...
    ];
B = [...
    Asp_b/norm(Asp_b), ...
    cross(Asp_b,Omega_b)/norm(cross(Asp_b,Omega_b)), ...
    cross(Asp_b,cross(Asp_b,Omega_b))/norm(cross(Asp_b,cross(Asp_b,Omega_b)))...
    ];

D_NED_B = B*inv(A);                 % Expressão simbólica para a DCM NED->B

% Definição e inicialização dos parâmetros
k = t_EXP/dt;                       % Quantidade total de medidas
n = 3;                              % Dimensão dos vetores de medidas
Q = sigma_acel^2 * eye(n);
R = sigma_gir^2 * eye(n);
RQ = chol(Q);
RR = chol(R);

% Simulação das medidas
g_NED_t = [0,0,9.81]*1e3;           % Gravidade descrita em St = NED [mg]
g_b = D_t_b*g_NED_t';
```

```
Omega_NED_t = ...
    7.29e-5*180/pi*3600 * ...
    [cos(lat0), 0 , -sin(lat0)]; % Vel. ang. da Terra em St = NED [grau/h]
Omega_b = D_t_b*Omega_NED_t';
rng(seed);                       % Semente do gerador de números aleatórios
Asp_b = ...
    repmat(g_b' + bias,k,1) + ...
    randn(k, n)*RQ;              % Medidas de força esp. do corpo [mg]
w_b = ...
    repmat(Omega_b' + deriva,k,1) + ...
    randn(k, n)*RR;              % Medidas de vel. ang. do corpo [grau/h]

% Como o veículo está parado, seu acelerômetro medirá a reação à gravidade
% e o seu girômetro medirá a velocidade angular da Terra
g = norm(mean(Asp_b(1:k,:)));    % Mag. da média das medidas de acel. [mg]
Omega = norm(mean(w_b(1:k,:)));  % Mag. da média das medidas de gir. [grau/h]

% Tomada da média das medidas no tempo de inicialização para o TRIAD
w_x = mean(w_b(1:k,1));
w_y = mean(w_b(1:k,2));
w_z = mean(w_b(1:k,3));
a_x = mean(Asp_b(1:k,1));
a_y = mean(Asp_b(1:k,2));
a_z = mean(Asp_b(1:k,3));

% Parâmetros do elipsóide de referência (WGS-84)
e = 1/298.25; % achatamento
R_0 = 6.378138e6; % m

% Raios do modelo da Terra
R_E = R_0*(1 + e*(sin(lat0))^2);             % raio leste-oeste
R_N = R_0*(1 - e*(2 - 3*(sin(lat0))^2));     % raio norte-sul
R_e = R_0*(1 - e*(sin(lat0))^2);             % raio terrestre

% Parâmetro g_0 que calibra a gravidade atuante no local
g_0 = g/((1+.0053*(sin(lat0))^2)*(1-2*h/R_e));% [mg]

% Resultado do TRIAD
lambda = lat0;
D_NED_B = double(subs(D_NED_B));
q0 = DCMtoQuaternion(D_NED_B);
euler0 = quatToEuler(q0);
```

Parte 1 - LKF

Filtro de Kalman linearizado em torno da solução de atitude do INS D_b_p_INS_k usando as medidas de erro de velocidade terrestre

Inicialização Salichev

```
% Estabilização vertical
h_m = h;
estabVert.B = 0.0001;
estabVert.C = 0.1;
estabVert.T_h = 60;

% Condições iniciais
q_NED_b = q0;
V_NED = [0,0,0]';
lat = lat0;
lon = lon0;
alt = h;
h_aux = h;

y = [...
    q_NED_b; ...
    V_NED; ...
    lat; ...
    lon; ...
    alt; ...
    h_aux ...
];
```

Inicialização Kalman

```
% Realização
n = 12;
x = zeros(k,n);
u = (zeros(k,n)+1);

a11 = -crossToMatrix(double(subs(2*Omega_NED)));
a12 = -crossToMatrix(double(subs(g_NED)));
a22 = -crossToMatrix(double(subs(Omega_NED)));

F = [...
    a11, a12, quatToDCM(q_NED_b), zeros(3); ...
    zeros(3), a22, zeros(3), -quatToDCM(q_NED_b); ...
    zeros(6,3), zeros(6,3), zeros(6,3), zeros(6,3) ...
    ];
G = zeros(n);

% Observação
z = zeros(k,3);
H = [eye(3), zeros(3,9)]; % Observa-se apenas o erro na velocidade

% Definição dos ruídos
q = 0.01 * ones(n,1);
r = 1 * ones(3,1);              % Covariância do ruído de observação
Q = diag(q.^0.5);               % Covariância do ruído de
```

```
R = diag(r.^0.5);
RQ = chol(Q);
RR = chol(R);
w = randn(k, n)*RQ;
v = randn(k, 3)*RR;

% Definição dos estimadores de mínima covariância
p00 = eye(n);
P_posteriori = p00;      % P(0|0)

% Condições iniciais
x(1,:) = zeros(1,n);
z(1,:) = (H*x(1,:)' + v(1,:)')';
i = 2;
modTerra.R_0 = R_0;
modTerra.e = e;
modTerra.g_0 = g_0;

% Inicialização dos vetores de estimação
x_priori = x;
x_posteriori = x;
```

Integração numérica incremental

```
for k = t_TRIAD/(4*dt):dt:t_EXP/(4*dt)

    %% Salichev
    % Conjunto de 4 medidas
    omega_B_k = (w_b(4*(i-1)-3 : 4*(i-1)-3+3, :))';
    Asp_B_k = (Asp_b(4*(i-1)-3 : 4*(i-1)-3+3, :))';

    % Atualização de estado
    y = fs(k, y, omega_B_k.*dt, Asp_B_k.*dt, modTerra, estabVert, h_m, dt);

    %% Filtro de Kalman
    % Atualização da matriz da dinâmica do filtro
    q_NED_b = y(1:4);       % Utiliza o quatérnion computado pelo Salichev
    F = [...
        a11, a12, quatToDCM(q_NED_b), zeros(3); ...
        zeros(3), a22, zeros(3), -quatToDCM(q_NED_b); ...
        zeros(6,3), zeros(6,3), zeros(6,3), zeros(6,3) ...
        ];

    % Realização: o estado evolui para x_n
    x(i,:) = (F*x(i-1,:)' + G*u(i-1,:)' + w(i-1,:)')';

    % Estimação: estimamos x-_n com base em x+_n-1
    x_priori(i,:) = (F*x_posteriori(i-1,:)'+G*u(i-1,:)')';

    % Obtemos P-_n a partir de P+_n-1
```

```
    P_priori = F*P_posteriori*F'+Q;

    % Obtemos K_n a partir de P-_n
    K = P_priori*H'/(H*P_priori*H'+R);

    % Obtemos P+_n a partir de P-_n
    P_posteriori = (eye(n)-K*H)*P_priori;
    variancia(i,:) = [P_posteriori(1,1),P_posteriori(1,1)];

    % Observação: medimos z_n
    z(i,:) = (H*x(i,:)' + v(i,:)')';

    % Previsão (Filtro de Kalman): estimamos x+_n com base em x-_n, z_n e K_n
    x_posteriori(i,:) = x_priori(i,:)+(K*(z(i,:)'-H*x_priori(i,:)'))';

    i = i+1;
end
```

Função de atualização de estado do algoritmo de Salichev

```
function [y_new] = fs(t, y, omega_B_i, Asp_B_i, modTerra, estabVert, h_m, T)

    %% Preparação e inicializações auxiliares
    N = 1;
    E = 2;
    D = 3;

    R_0 = modTerra.R_0;
    e = modTerra.e;
    g_0 = modTerra.g_0;

    B = estabVert.B;
    C = estabVert.C;
    T_h = estabVert.T_h;

    Omega = 7.2921e-5;


    %% Mapeamento de variáveis
    q_NEDold_bold = y(1:4);
    V_NED = y(5:7);
    lat = y(8);
    lon = y(9);
    alt = y(10);
    h_aux = y(11);


    %% Passo 0: quaternion de rotação do NEDold para NEDnew a cada 4 amostras de freqüên-
    % cia rápida dos sensores inerciais, computado com frequência lenta.
```

```
%% Modelo da Terra
R_E = R_0*(1 + e*(sin(lat))^2);            % raio leste-oeste
R_N = R_0*(1 - e*(2 - 3*(sin(lat))^2));    % raio norte-sul

rho_NED = [...
    V_NED(E)/(R_E+alt), ...
    -V_NED(N)/(R_N+alt), ...
    -V_NED(E)/(R_E+alt)*tan(lat) ...
    ]';
Omega_NED = [Omega*cos(lat),0,-Omega*sin(lat)]';

omega_NEDi_NED = rho_NED + Omega_NED;  % taxa de transporte usa estimativas de velo-
                                       % cidade terrestre e posição mais recentes
                                       % disponíveis

omega_versor = omega_NEDi_NED/norm(omega_NEDi_NED); % eixo de rotação instantânea
                                                    % unitário

q_NEDold_NEDnew = [...
     cos(norm(omega_NEDi_NED*4*T)/2);...
     omega_versor(:)*sin(norm(omega_NEDi_NED*4*T)/2)
     ];


%% Passo 1: incremento da velocidade de empuxo \delta U_{f,b,k} computada com fre-
% quência rápida. Índice k representa o instante inicial de um conjunto de 4 amos-
% tras; k+1 o instante inicial do próximo conjunto de 4 amostras, sem superposição
% com o anterior.
alpha(:,1) = omega_B_i(:,1);
alpha(:,2) = omega_B_i(:,2);
alpha(:,3) = omega_B_i(:,3);
alpha(:,4) = omega_B_i(:,4);

delta_beta(:,1) = Asp_B_i(:,1);
delta_beta(:,2) = Asp_B_i(:,2);
delta_beta(:,3) = Asp_B_i(:,3);
delta_beta(:,4) = Asp_B_i(:,4);

W_k_0 = zeros(3,1);   % W_k(0) = (0,0,0)^T
W_k_old = W_k_0;

for m=1:4 % sculling correction
    W_k_new = delta_beta(:,m) - cross(alpha(:,m), W_k_old) + W_k_old;
    W_k_new = delta_beta(:,m) - cross(alpha(:,m), W_k_new) + W_k_old;
    W_k_old = W_k_new;
end

delta_U_f_b_k = W_k_new;


%% Passo 2: Incremento angular computado com quatro amostras incrementais e quaternion
% de rotação q^{bold}_{bnew} do corpo na atitude anterior (bold) para o corpo na nova
% atitude (bnew). (new é a estampa de tempo k+1 e old é a estampa de tempo k).
```

```
% coning correction
P1 = crossToMatrix(alpha(:,1));
P2 = crossToMatrix(alpha(:,2));
P3 = crossToMatrix(alpha(:,3));
P4 = crossToMatrix(alpha(:,4));

delta_phi = alpha(:,1)+alpha(:,2)+alpha(:,3)+alpha(:,4)+...
    2/3*(P1*alpha(:,2)+P3*alpha(:,4))+...
    1/2*(P1+P2)*(alpha(:,3)+alpha(:,4))+...
    1/30*(P1-P2)*(alpha(:,3)-alpha(:,4));

delta_phi_versor = delta_phi/norm(delta_phi);

q_bold_bnew = [...
     cos(norm(delta_phi)/2);...
     delta_phi_versor(:)*sin(norm(delta_phi)/2)
     ];

% quatérnion associado a \delta U_{f,b,k}
delta_U_f_b_k_q = quat(delta_U_f_b_k);


%% Passo 3: Computa em frequência lenta o quatérnion de rotação q^{NEDnew}_{bnew}
% mediante atualização de q^{bold}_{NEDold} devido à rotação de S_{NED} para posterior
% transformação da força específica de S_b para S_{NED}. (new é a estampa de tempo k+1
% e old é a estampa de tempo k).
q_NEDnew_bnew = quatInv(...
    quatProd(...
        quatProd(...
            quatInv(q_bold_bnew), ...
            quatInv(q_NEDold_bold)), ...
        q_NEDold_NEDnew)...
    );
delta_U_f_NED_k_q = quatProd(...
    quatProd(...
        q_NEDnew_bnew, ...
        delta_U_f_b_k_q), ...
    quatInv(q_NEDnew_bnew)...
    );


%% Passo 4: Atualiza em baixa freqüência (a cada período 4T) velocidade terrestre e
% posição. Usa posição (latitude, longitude e altitude) e velocidade terrestre V_{NED}
% disponíveis para computar incremento de velocidade terrestre devido à gravidade.
R_e = R_0*(1 - e*(sin(lat))^2);
g = g_0*(1+.0053*(sin(lat))^2)*(1-2*alt/R_e);
g_NED = [0,0,g]';

delta_Vg_NED = (...
    -cross((rho_NED + 2*Omega_NED), V_NED) + ...
    g_NED + ...
    [0,0,B*(alt-h_aux)]'...
    )*4*T;
```

```
    delta_V_NED = delta_Vg_NED + quatToVector(delta_U_f_NED_k_q);
    V_NED = V_NED + delta_V_NED;  % atualiza V_{NED}

    % Não precisamos atualizar posição, uma vez que o veículo está
    % estacionário

    y_new = [...
        q_NEDnew_bnew; ...
        V_NED; ...
        lat; ...
        lon; ...
        alt; ...
        h_aux ...
    ];
end
```