```
function [aircraft_state_est, wind_inertial_est] =
EstimatorAttitudeGPSSmoothing(time, gps_sensor, inertial_sensors,
sensor_params)
%
% gps_sensor = [pn; pe; ph; Vg; chi]
%
% inertial_sensors = [y_accel; y_gyro; y_pressure; y_dyn_pressure];
%

persistent phat
persistent qhat
persistent rhat

persistent press_stat
persistent press_dyn

persistent phi_hat
persistent theta_hat
persistent P_est

persistent xhat_gps
persistent P_gps

h_ground = sensor_params.h_ground;
density = stdatmo(h_ground);



Ts_imu = sensor_params.Ts_imu;
Ts_gps = sensor_params.Ts_gps;
g = sensor_params.g;

%%%%%%%%%%%%%%%%%%%%%%
%%% angular velocity
%%%%%%%%%%%%%%%%%%%%%%
a_omega = 1000;
alpha_omega = exp(-a_omega*Ts_imu);

if(isempty(phat))
    phat = inertial_sensors(4);
else
    phat = LowPassFilter(phat, inertial_sensors(4), alpha_omega);
end

if(isempty(qhat))
    qhat = inertial_sensors(5);
else
    qhat = LowPassFilter(qhat, inertial_sensors(5), alpha_omega);
end

if(isempty(rhat))
    rhat = inertial_sensors(6);
```

```
else
    rhat = LowPassFilter(rhat, inertial_sensors(6), alpha_omega);
end

%%%%%%%%%%%%%%%
%%% height
%%%%%%%%%%%%%%%

a_h = 1000;% <======================STUDENT COMPLETE, SAME AS SIMPLE
ESTIMATOR
alpha_h = exp(-a_h*Ts_imu);

if(isempty(press_stat))
    press_stat = inertial_sensors(7);
else
    press_stat = LowPassFilter(press_stat, inertial_sensors(7), alpha_h);
end
hhat = press_stat / (density * g) + h_ground;%
<======================STUDENT COMPLETE, SAME AS SIMPLE ESTIMATOR

%%%%%%%%%%%%%%%
%%% airspeed
%%%%%%%%%%%%%%%

a_Va = 1000;% <======================STUDENT COMPLETE, SAME AS SIMPLE
ESTIMATOR
alpha_Va = exp(-a_Va*Ts_imu);

if(isempty(press_dyn))
    press_dyn = inertial_sensors(8);
else
    press_dyn = LowPassFilter(press_dyn, inertial_sensors(8), alpha_Va);
end
Va = sqrt(2/density * press_dyn);% <======================STUDENT COMPLETE,
SAME AS SIMPLE ESTIMATOR

%%%%%%%%%%%%%%%%%%%
%%% orientation
%%%%%%%%%%%%%%%%%%%
Q = 0.01*((pi/180)^2)*eye(2);
R = sensor_params.sig_accel*sensor_params.sig_accel*eye(3);

if(isempty(phi_hat))
    phi_hat = 0;
    theta_hat = 0;
    P_est = ((30*pi/180)^2)*eye(2);
else
```

Propagate

```
    [xdot, A] = AttitudeFilterUpdate(phi_hat, theta_hat, phat, qhat, rhat);
    phi_hat = phi_hat + xdot(1)*Ts_imu;
```

```
    theta_hat = theta_hat + xdot(2)*Ts_imu;
    P_est = P_est + Ts_imu*(A*P_est + P_est*A' + Q);
```

Measurement update

```
    [zhat, H] = AttitudeFilterMeasurement(phi_hat, theta_hat, phat, qhat,
rhat, Va, g);
    L = P_est*H'*inv(R+H*P_est*H');
    P_est = (eye(2)-L*H)*P_est;
    xhat = [phi_hat; theta_hat]+L*(inertial_sensors(1:3,1)-zhat);
    phi_hat = xhat(1);
    theta_hat = xhat(2);
end



%%%%%%%%%%%%%%%%%%%%
%%% GPS smoothing
%%%%%%%%%%%%%%%%%%%%
Qgps = [10^2 0 0 0 0 0 0;...    %pn
     0 10^2 0 0 0 0 0;...    % pe
     0 0 2^2 0 0 0 0;...     %Vg
     0 0 0 (5*pi/180)^2 0 0 0;...       %chi
     0 0 0 0 25 0 0;...     %wn
     0 0 0 0 0 25 0;...     %we
     0 0 0 0 0 0 (5*pi/180)^2];     %psi


Rgps = [sensor_params.sig_gps(1)^2 0 0 0 0 0;...
    0 sensor_params.sig_gps(2)^2 0 0 0 0;...
    0 0 sensor_params.sig_gps_v^2 0 0 0;...
    0 0 0 (sensor_params.sig_gps_v/20)^2 0 0;...
    0 0 0 0 sensor_params.sig_gps_v^2 0;...
    0 0 0 0 0 sensor_params.sig_gps_v^2];


if(isempty(xhat_gps))
    xhat_gps = [gps_sensor(1); gps_sensor(2); gps_sensor(4); gps_sensor(5);
0; 0; gps_sensor(5)];
    P_gps = 10*Qgps;
else
```

Propagate

```
    [xdot_gps, A_gps] = GPSSmoothingUpdate(xhat_gps, Va, qhat, rhat,
phi_hat, theta_hat,g);

    xhat_gps = xhat_gps + xdot_gps*Ts_imu;% <=============== Assumes filter
runs at IMU rate
    P_gps = P_gps + Ts_imu*(A_gps*P_gps + P_gps*A_gps' + Qgps);%
<=============== Assumes filter runs at IMU rate
```

```
    if(mod(time, Ts_gps)==0) % <=============== Only update at GPS rate
```

Measurement update

```
        [zhat_gps, H_gps] = GPSSmoothingMeasurement(xhat_gps, Va);
        ygps = [gps_sensor(1); gps_sensor(2); gps_sensor(4); gps_sensor(5);
0; 0];
        L_gps = P_gps*H_gps'*inv(Rgps+H_gps*P_gps*H_gps');
        P_gps = (eye(7)-L_gps*H_gps)*P_gps;

        yerr_gps = ygps-zhat_gps;

        while(yerr_gps(4)>pi)
            yerr_gps(4) = yerr_gps(4) - 2*pi;
        end
        while(yerr_gps(4)< -pi)
            yerr_gps(4) = yerr_gps(4) + 2*pi;
        end

        xhat_gps = xhat_gps + L_gps*(yerr_gps);
    end
end


%%%%%%%%%%%%%%%
%%% output
%%%%%%%%%%%%%%%
wind_body_est = TransformFromInertialToBody([xhat_gps(5);xhat_gps(6);0] ,
[phi_hat; theta_hat; xhat_gps(7)]);
air_rel_est = [Va*cos(theta_hat); 0; Va*sin(theta_hat)];
vel_body_est = air_rel_est + wind_body_est;

aircraft_state_est =  [xhat_gps(1); xhat_gps(2); -hhat;
                       phi_hat; theta_hat; xhat_gps(7);
                       vel_body_est;
                       phat; qhat; rhat];

wind_inertial_est = [xhat_gps(5);xhat_gps(6);0];

end %function


%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
% Low Pass Filter
%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

function ynew = LowPassFilter(yold, unew, alpha)
    ynew = alpha*yold + (1-alpha)*unew;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Attitude Filter Equations
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

function [xdot, A] = AttitudeFilterUpdate(phi, theta, p, q, r)

xdot =   [p + q*sin(phi)*tan(theta) + r*cos(phi)*tan(theta);
        q*cos(phi) - r*sin(phi)];

A = [q*cos(phi)*tan(theta) - r*sin(phi)*tan(theta), (q*sin(phi) +
r*cos(phi))/(cos(theta)^2);
    -q*sin(phi) - r*cos(phi), 0];

end

function [y, H] = AttitudeFilterMeasurement(phi, theta, p, q, r, Va, g)

y = [q*Va*sin(theta) + g*sin(theta);
    r*Va*cos(theta) - p*Va*sin(theta) - g*cos(theta)*sin(phi);
    -q*Va*cos(theta) - g*cos(theta)*cos(phi)];

H = [0, q*Va*cos(theta) + g*cos(theta);
    -g*cos(phi)*cos(theta), -r*Va*sin(theta) - p*Va*cos(theta) +
g*sin(phi)*sin(theta);
    g*sin(phi)*cos(theta), (q*Va + g*cos(phi))*sin(theta)];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% GPS Smoothing Filter Equations
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

% xh = [pn, pe, Vg, chi, wn, we, psi]
function [xdot, A] = GPSSmoothingUpdate(xh, Va, q, r, roll, pitch, g)
% GPS estimate values
Vg = xh(3);
chi = xh(4);
psi = xh(7);
wn = xh(5);
we = xh(6);

% Angles
phi = roll;
theta = pitch;

% ROC
psidot = q*sin(roll)/cos(pitch) + r*cos(roll)/cos(pitch);
Vg_dot = ((Va*sin(xh(7))+xh(6))*(Va*psidot*cos(xh(7)))-(Va*cos(xh(7))
```

```
+xh(5))*(Va*psidot*sin(xh(7))))/xh(3);

dVgdpsi = -psidot*Va*(wn*cos(psi) + we*sin(psi)) / Vg;
dchiDotdVg = -g/Vg^2 * tan(phi)*cos(chi - psi);
dchiDotdchi = -g/Vg * tan(phi) * sin(chi - psi);
dchiDotdpsi = g/Vg * tan(phi) *sin(chi - psi);

% State dot
xdot = [Vg*cos(chi);
    Vg*sin(chi);
    ((Va*cos(psi) + wn)*(-Va*psidot*sin(psi)) + (Va*sin(psi) +
we)*(Va*psidot*cos(psi))) / Vg;
    g/Vg * tan(phi)*cos(chi - psi);
    0; 0;
    q*sin(phi)/cos(theta) + r*cos(phi)/cos(theta)];

% Jacobian
A = [0, 0, cos(chi), -Vg*sin(chi), 0, 0, 0;
    0, 0, sin(chi), Vg*cos(chi), 0, 0, 0;
    0, 0, -Vg_dot/Vg, 0, -psidot*Va*sin(psi)/Vg, psidot*Va*cos(psi)/Vg,
dVgdpsi;
    0, 0, dchiDotdVg, dchiDotdchi, 0, 0, dchiDotdpsi;
    zeros(3, 7)];

end

% xh = [pn, pe, Vg, chi, wn, we, psi]
function [y, H] = GPSSmoothingMeasurement(xh, Va)
pn = xh(1);
pe = xh(2);
Vg = xh(3);
chi = xh(4);
wn = xh(5);
we = xh(6);
psi = xh(7);

y = [pn; pe; Vg; chi; Va*cos(psi) + wn - Vg*cos(chi); Va*sin(psi) + we -
Vg*sin(chi)];

H = [1, 0, 0, 0, 0, 0, 0;
    0, 1, 0, 0, 0, 0, 0;
    0, 0, 1, 0, 0, 0, 0;
    0, 0, 0, 1, 0, 0, 0;
    0, 0, -cos(chi), Vg*sin(chi), 1, 0, -Va*sin(psi);
    0, 0, -sin(chi), -Vg*cos(chi), 0, 1, Va*cos(psi)];
end
```

Published with MATLAB® R2023b