```
% quadsim_sensors.m
%
% Developed for JHU EP 525.461, UAV Systems & Control
% Adapted from design project in "Small Unmanned Aircraft: Theory and
% Practice", RWBeard & TWMcClain, Princeton Univ. Press, 2012
%
function out = quadsim_sensors(uu, P)

    % Extract variables from input vector uu
    %   uu = [f_and_m(1:6); x(1:12); wind_ned(1:3); time(1)];
    k=(1:6);           f_and_m=uu(k);   % Forces and Moments, body
    k=k(end)+(1:12);   x=uu(k);         % states
    k=k(end)+(1:3);    wind_ned=uu(k);  % wind vector, ned, m/s
    k=k(end)+(1);      time=uu(k);      % Simulation time, s

    % Extract forces and moments from f_and_m
    fb_x = f_and_m(1); % Total force along body x, N
    fb_y = f_and_m(2); % Total force along body y, N
    fb_z = f_and_m(3); % Total force along body z, N
    mb_x = f_and_m(4); % Total moment about body x, N-m
    mb_y = f_and_m(5); % Total moment about body y, N-m
    mb_z = f_and_m(6); % Total moment about body z, N-m

    % Extract state variables from x
    pn    = x(1);   % North position, m
    pe    = x(2);   % East position, m
    pd    = x(3);   % Down position, m
    u     = x(4);   % body-x groundspeed component, m/s
    v     = x(5);   % body-y groundspeed component, m/s
    w     = x(6);   % body-z groundspeed component, m/s
    phi   = x(7);   % EulerAngle: roll, rad
    theta = x(8);   % EulerAngle: pitch, rad
    psi   = x(9);   % EulerAngle: yaw, rad
    p     = x(10);  % body rate about x, rad/s
    q     = x(11);  % body rate about y, rad/s
    r     = x(12);  % body rate about z, rad/s

    % Gyro
    p_gyro = p + P.sigma_noise_gyro*randn; % rad/s
    q_gyro = q + P.sigma_noise_gyro*randn; % rad/s
    r_gyro = r + P.sigma_noise_gyro*randn; % rad/s

    % Accelerometer
    ax = ((1/P.mass)*fb_x) + P.gravity*sin(theta);
    ay = ((1/P.mass)*fb_y) - P.gravity*cos(theta)*sin(phi);
    az = ((1/P.mass)*fb_z) - P.gravity*cos(theta)*cos(phi);

    ax_accel= ax + P.sigma_noise_accel*randn;
    ay_accel= ay + P.sigma_noise_accel*randn;
    az_accel= az + P.sigma_noise_accel*randn;

    % Barometric Pressure Altimeter
```

```
    P0 = 101325;
    R = 8.31432;
    M = 0.0289644;%
    T = 5/9*(P.air_temp_F-32)+273.15;

    persistent bias_static_press
    if(time==0)
        bias_static_press = P.sigma_bias_static_press*randn;
    end
    true_static_press = P0*exp(((-M*P.gravity)*(P.h0_ASL-pd))/(R*T));
    static_press = true_static_press + bias_static_press +
P.sigma_noise_static_press*randn;

    persistent bias_diff_press
    if(time==0)
        bias_diff_press = P.sigma_bias_diff_press*randn;
    end
    R_ned2b = eulerToRotationMatrix(phi,theta,psi);
    v_wb = R_ned2b*wind_ned;
    v_ab = [u;v;w]-v_wb;
    Va = sqrt(v_ab(1)^2+v_ab(2)^2+v_ab(3)^2);
    true_diff_press = 0.5*P.rho*Va^2;
    diff_press = true_diff_press + bias_diff_press +
P.sigma_noise_diff_press*randn;

    persistent bias_mag
    if(time==0)
        bias_mag = P.sigma_bias_mag*randn;
    end
    psi_mag = psi + bias_mag + P.sigma_noise_mag*randn; % Magnetometer
measurement, rad

    % GPS Position and Velocity Measurements
    persistent time_gps_prev gps_north_error gps_east_error gps_alt_error ...
               pn_gps pe_gps alt_gps Vn_gps Ve_gps Vd_gps
    if(time==0)
        gps_north_error = P.sigma_bias_gps_north*randn;
        gps_east_error = P.sigma_bias_gps_east*randn;
        gps_alt_error = P.sigma_bias_gps_alt*randn;
        time_gps_prev = -inf;
    end

    if(time>time_gps_prev+P.Ts_gps)

        % Gauss-Markov growth of GPS position errors
        gps_north_error = exp(-P.Ts_gps/P.tau_gps)*gps_north_error +
P.sigma_eta_gps_north*randn*sqrt(P.Ts_gps);
        gps_east_error  = exp(-P.Ts_gps/P.tau_gps)*gps_east_error +
P.sigma_eta_gps_east*randn*sqrt(P.Ts_gps);
        gps_alt_error   = exp(-P.Ts_gps/P.tau_gps)*gps_alt_error +
P.sigma_eta_gps_alt*randn*sqrt(P.Ts_gps);

        % GPS Position Measurements
        pn_gps = pn + gps_north_error;
```

```
        pe_gps = pe + gps_east_error;
        alt_gps= -pd + gps_alt_error;

        % GPS Velocity Measurements
            % Compute Rotation Matrices
        R_ned2b = eulerToRotationMatrix(phi,theta,psi);
        R_b2ned = R_ned2b';
            % Manipulate states
        vg_ned = R_b2ned*[u; v; w];
        Vn_gps = vg_ned(1) + P.sigma_noise_gps_speed*randn;
        Ve_gps = vg_ned(2) + P.sigma_noise_gps_speed*randn;
        Vd_gps = vg_ned(3) + P.sigma_noise_gps_speed*randn;

        time_gps_prev = time;
    end

    % Compile output vector
    out = [ ...
            pn_gps; ...
            pe_gps; ...
            alt_gps;  ...
            Vn_gps; ...
            Ve_gps; ...
            Vd_gps; ...
            p_gyro; ...
            q_gyro; ...
            r_gyro; ...
            ax_accel;...
            ay_accel;...
            az_accel;...
            static_press; ...
            diff_press; ...
            psi_mag;...
            0; % future use
            0; % future use
            0; % future use
          ]; % Length: 18

end
```

```
Not enough input arguments.

Error in quadsim_sensors (line 11)
    k=(1:6);           f_and_m=uu(k);   % Forces and Moments, body
```

Published with MATLAB® R2023a