```
% quadsim_control.m
%
% Flight control logic for quadsim
%
% 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_control(uu,P)
    persistent call_count;

    % Check if call_count is empty (first call)
    if isempty(call_count)
        call_count = 1;
    end

    % Extract variables from input vector uu
    %   uu = [traj_cmds(1:4); estimates(1:23); time(1)];
    k=(1:4);         traj_cmds=uu(k); % Trajectory Commands
    k=k(end)+(1:23); estimates=uu(k); % Feedback state estimates
    k=k(end)+(1);    time=uu(k);      % Simulation time, s

    % Extract variables from traj_cmds
    h_c      = traj_cmds(1);  % commanded altitude (m)
    Vhorz_c  = traj_cmds(2);  % commanded horizontal speed (m/s) (change from uavsim)
    chi_c    = traj_cmds(3);  % commanded course (rad)
    psi_c    = traj_cmds(4);  % yaw course (rad) (change from uavsim)

    % Extract variables from estimates
    pn_hat       = estimates(1);  % inertial North position, m
    pe_hat       = estimates(2);  % inertial East position, m
    h_hat        = estimates(3);  % altitude, m
    Va_hat       = estimates(4);  % airspeed, m/s
    phi_hat      = estimates(5);  % roll angle, rad
    theta_hat    = estimates(6);  % pitch angle, rad
    psi_hat      = estimates(7);  % yaw angle, rad
    p_hat        = estimates(8);  % body frame roll rate, rad/s
    q_hat        = estimates(9);  % body frame pitch rate, rad/s
    r_hat        = estimates(10); % body frame yaw rate, rad/s
    Vn_hat       = estimates(11); % north speed, m/s
    Ve_hat       = estimates(12); % east speed, m/s
    Vd_hat       = estimates(13); % downward speed, m/s
    wn_hat       = estimates(14); % wind North, m/s
    we_hat       = estimates(15); % wind East, m/s
    future_use   = estimates(16:23);

    % Initialize controls to trim (to be with PID logic)
    delta_e=P.delta_e0;
    delta_a=P.delta_a0;
    delta_r=P.delta_r0;
    delta_t=P.delta_t0;

    % Initialize autopilot commands (may be overwritten with PID logic)
    phi_c = 0;
    theta_c = 0;
```

```
Vhx_c_list = zeros(1, 2001);
Vhy_c_list = zeros(1, 2001);

Vhx_hat_list = zeros(1, 2001);
Vhy_hat_list = zeros(1, 2001);

R_ned2b = eulerToRotationMatrix(phi_hat,theta_hat,psi_hat);
vgb_hat = R_ned2b*[Vn_hat; Ve_hat; Vd_hat];
hdot_hat = -vgb_hat(3);

% Set "first-time" flag, which is used to initialize PID integrators
firstTime=(time==0);

% Flight control logic
%   <code goes here>
% e.g.
%    delta_a = PID_roll_hold(phi_c, phi_hat, p_hat, firstTime, P);
%
% Note: For logging purposes, use variables:
%         Vhorz_c,  chi_c, h_c, phi_c, theta_c, psi_c

% Getting trajectory commands
[WP_n, WP_e, h_c, psi_c] = get_quadsim_trajectory_commands(time);
chi_c = atan2(WP_e-pe_hat, WP_n - pn_hat);
k_pos = 0.25;
Vhorz_c = k_pos*sqrt((WP_e-pe_hat)^2 + (WP_n - pn_hat)^2);
if (Vhorz_c > 8)
     Vhorz_c = 8;
end
%
Vn_cmd = Vhorz_c*cos(chi_c);
Ve_cmd = Vhorz_c*sin(chi_c);

Vh_c = [cos(psi_hat) sin(psi_hat); -sin(psi_hat) cos(psi_hat)]*[Vn_cmd; Ve_cmd];
Vhx_c = Vh_c(1); Vhy_c = Vh_c(2);

Vh_hat = [cos(psi_hat) sin(psi_hat); -sin(psi_hat) cos(psi_hat)]*[Vn_hat; Ve_hat];
Vhx_hat = Vh_hat(1); Vhy_hat = Vh_hat(2);

Vhy_c_list(call_count) = Vhy_c;
Vhx_c_list(call_count) = Vhx_c;
Vhy_hat_list(call_count) = Vhy_hat;
Vhy_hat_list(call_count) = Vhy_hat;

if(firstTime)
     % Initialize integrators
     PIR_vhorz_hold_x(0,0,0,firstTime,P);
     PIR_vhorz_hold_y(0,0,0,firstTime,P);
     PIR_roll_hold(0,0,0,firstTime,P);
     PIR_pitch_hold(0,0,0,firstTime,P);
     PIR_alt_hold(0,0,0,firstTime,P);
     PIR_yaw_hold(0,0,0,firstTime,P);
end

% Controls
theta_c = PIR_vhorz_hold_x(Vhx_c, Vhx_hat, 0, firstTime, P);
phi_c = PIR_vhorz_hold_y(Vhy_c, Vhy_hat, 0, firstTime, P);
delta_t = PIR_alt_hold(h_c, h_hat, hdot_hat, firstTime, P);
```

```
    delta_a = PIR_roll_hold(phi_c, phi_hat, p_hat, firstTime, P);
    delta_e = PIR_pitch_hold(theta_c, theta_hat, q_hat, firstTime, P);
    delta_r = PIR_yaw_hold(psi_c, psi_hat, r_hat, firstTime, P);

    % Compile vector of control surface deflections
    delta = [ ...
            delta_e; ...
            delta_a; ...
            delta_r; ...
            delta_t; ...
        ];

    % Override control delta with manual flight delta
    if P.manual_flight_flag
        error('Manual flight not supported in quadsim')
    end

    % Compile autopilot commands for logging/vis
    ap_command = [ ...
            Vhorz_c; ...
            h_c; ...
            chi_c; ...
            phi_c; ...
            theta_c;
            psi_c; ... % change from uavsim
            0; ... % future use
            0; ... % future use
            0; ... % future use
        ];

    call_count = call_count + 1;

    % Compile output vector
    out=[delta;ap_command]; % 4+9=13

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% roll_hold
%   - regulate roll using aileron
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_roll_hold(phi_c, phi_hat, p_hat, init_flag, P)

    % Set up PI with rate feedback
    y_c = phi_c;
    y = phi_hat;
    y_dot = p_hat;

    kp = 0.11;
    ki = 0.0075;
    kd = 0.025;

    u_lower_limit = -0.1;
    u_upper_limit = +0.1;

    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
```

```
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    error = y_c - y;  % Error between command and response
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot;

    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
            error_int = error_int - P.Ts*error;
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% alt_hold
%   - regulate altitude using throttle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_alt_hold(h_c, h_hat, hdot_hat, init_flag, P)

    % Set up PI with rate feedback
    y_c = h_c;
    y = h_hat;
    y_dot = hdot_hat;

    kp = 0.05;
    ki = 0.0001;
    kd = 0.07;

    u_lower_limit = 0.1;
    u_upper_limit = 0.9;

    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    error = y_c - y;  % Error between command and response
    if error > 10
        error = 10;
    elseif error < -10
        error = -10;
    end
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot + 0.5;
```

```
    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
            error_int = error_int - P.Ts*error;
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% pitch_hold
%   - regulate pitch using elevator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_pitch_hold(theta_c, theta_hat, q_hat, init_flag, P)

    % Set up PI with rate feedback
    y_c = theta_c;
    y = theta_hat;
    y_dot = q_hat;

    kp = 0.11;
    ki = 0.0075;
    kd = 0.025;

    u_lower_limit = -0.1;
    u_upper_limit = 0.1;

    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    error = y_c - y;  % Error between command and response
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot;

    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
```

```
            error_int = error_int - P.Ts*error;
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% yaw_hold
%   - regulate yaw using rudder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_yaw_hold(psi_c, psi_hat, r_hat, init_flag, P)

    % Set up PI with rate feedback
    y_c = psi_c;
    y = psi_hat;
    y_dot = r_hat;

    kp = 0.11;
    ki = 0.002;
    kd = 0.07;

    u_lower_limit = -0.1;
    u_upper_limit = 0.1;

    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    % error = y_c - y;  % Error between command and response
    error = mod(y_c - y + pi, 2*pi)-pi;
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot;

    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
            error_int = error_int - P.Ts*error;
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% horizontal velocity hold - x axis
%   - regulate horizontal velocity hold through pitch - x axis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_vhorz_hold_x(Vhx_c, Vhx_hat, not_used, init_flag, P)
```

```
    % Set up PI with rate feedback
    y_c = Vhx_c;
    y = Vhx_hat;
    y_dot = 0;

    kp = -0.035;
    ki = -0.025;
    kd = -0.00001;

    u_lower_limit = -P.theta_max;
    u_upper_limit = +P.theta_max;

    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    error = y_c - y;  % Error between command and response
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot;

    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
            error_int = error_int - P.Ts*error;
        end
    end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% horizontal velocity hold - y axis
%   - regulate horizontal velocity hold through roll - y axis
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function u = PIR_vhorz_hold_y(Vhy_c, Vhy_hat, not_used, init_flag, P)

    % Set up PI with rate feedback
    y_c = Vhy_c;
    y = Vhy_hat;
    y_dot = 0;

    kp = 0.0349;
    ki = 0.02;
    kd = 0.00001;

    u_lower_limit = -P.phi_max;
    u_upper_limit = +P.phi_max;
```

```
    % Initialize integrator (e.g. when t==0)
    persistent error_int;
    if( init_flag )
        error_int = 0;
    end

    % Perform "PI with rate feedback"
    error = y_c - y;  % Error between command and response
    error_int = error_int + P.Ts*error; % Update integrator
    u = kp*error + ki*error_int - kd*y_dot;

    % Output saturation & integrator clamping
    %   - Limit u to u_upper_limit & u_lower_limit
    %   - Clamp if error is driving u past limit
    if u > u_upper_limit
        u = u_upper_limit;
        if ki*error>0
            error_int = error_int - P.Ts*error;
        end
    elseif u < u_lower_limit
        u = u_lower_limit;
        if ki*error<0
            error_int = error_int - P.Ts*error;
        end
    end

end
```

```
Not enough input arguments.

Error in quadsim_control (line 19)
    k=(1:4);         traj_cmds=uu(k); % Trajectory Commands
```

Published with MATLAB® R2023a