Contents

PID Control of Quadcopter Altitude Near Hover

Acknowledgement: Prof. Peter Seiler

```
%------------------------------------------------
function [wn, zeta] = Low_Fidelity_Model(DISP_NAME, m, kT, Kp, Kd)
```

Vehicle Parameters

close all; clear all; m = 65e-3; % Mass, kg

```
    T_START = 19; % s
    T_STEADY = 30; % s, Chosen as start of steady-state response from observation
    T_END = T_START+25; % s
    g = 9.81; % m/s^2
    % kT = ; % Thrust coefficient, N
    % kT = input('Enter the estimated thrust coefficient, N, kT: ');
    umax = 500; % Maximum motor input command, unitless
    umin = 0; % Minimum motor input command, unitless
```

Step change in altitude reference, m

```
    Tf    = 25;           % Final simulation time, sec
    hdes0 = 0.7;          % Initial altitude, m
    hdesf = 1.25;         % Final altitude, m
```

Initial Conditions

```
    h0 = hdes0;        % Initial altitude, m
    hdot0 = 0;         % Initial altitude velocity, m/s
    hddmax = (4*kT*umax-m*g)/m;    % Maximum upward acceleration, m/s^2
```

```
Not enough input arguments.

Error in Low_Fidelity_Model (line 26)
    hddmax = (4*kT*umax-m*g)/m;    % Maximum upward acceleration, m/s^2
```

PD Control With Gravity Feedforward

Consider the following PD control law with gravity feedforward u = Kp*(hdes-h) - Kd*hdot + (mhat*g)/(4*kThat) Here u is the ***individual motor command**** input (unitless). Also, (mhat,kThat) indicate that these are estimates used by the controller. The parrot drone Simulink diagram uses:5 uhat = Kphat*(hdes-h) - Kdhat*hdot + mhat*g where uhat is the ***total thrust**** command (N). These are related by uhat = (4*kT)*u Thus gains computed using the simplified second-order model must be scaled by (4*kT) before implementing in the parrot Simulink diagram.

NOTE - An integral gain is included below for testing but this can be set to zero to simply to the PD control given above.

```
% Select Gains -- Gains are specified using simplified second order model

GainCase = 2;
switch GainCase
    case 1
        % Select closed-loop natural frequency and damping ratio
        wn = input('Desired natural freq, rad/sec, wn: ');      % Desired natural freq, rad/sec
        zeta = input('Desired damping ratio, unitless, zeta: ' );  % Desired damping ratio, unitless

        % Closed-loop ODE with PD control + perfect gravity cancellation:
        %   m hdd = (4*kT)*( Kp*(hdes-h) - Kd*hdot )
        % --> hdd + (4*kT*Kd/m) hdot + (4*kT*Kp/m) h = (4*kT*Kp/m)*hdes
        Kp = wn^2*m/(4*kT);
        Kd = 2*zeta*wn*m/(4*kT);
        Ki = 0;

    case 2
        % Kp = input('Enter the proportional gain, Kp: ');
        % Kd = input('Enter the derivative gain, Kd: ');
        Ki = 0;

        wn = sqrt(Kp*4*kT/m);
        zeta = 4*kT*Kd/(2*wn*m);
end
```

Closed-Loop Transfer Function (hdes to h)

This constructs the closed-loop model from altitude reference to altitude. It assumes perfect cancellation of the gravitational force by the feedforward term. It also neglects the saturation that limits the motor command.

```
% Plant dynamcs (neglecting gravity term)
%   m d^2(h)/dt^2 = (4*kT)*u
% The state-space model below has input motor command and outputs [h;hdot]
Ag = [0 1; 0 0];
Bg = [0; 4*kT/m];
Cg = eye(2);
Dg = 0;
G = ss(Ag,Bg,Cg,Dg);
time = [];
h_arr = [];

tstep = 1;   % Step time, sec
% Controller: PI and D terms
if Ki==0
    Cpi = ss(Kp);
else
    Cpi = tf([Kp Ki],[1 0]);
end
Cd = ss(Kd);

% Form closed-loop from r to h, T
systemnames = 'G Cpi Cd';
inputvar = '[r]';
outputvar = '[G(1)]';
input_to_G = '[Cpi+Cd]';
input_to_Cpi = '[r-G(1)]';
input_to_Cd = '[-G(2)]';
%T = sysic;

if false
    % Code to debug:  Represent controller
    % (*)  u = Kp e + Kd edot + Ki integral(e)
    % The actual implementation only uses rate-feedback, i.e. Kd*edot is
    % replaced by -Kd*hdot.  However, the form in (*) above is much easier
    % to construct with standard functions and will yield the same poles
    % (only the closed-loop zeros will be different)
```

```
    Gb = tf(4*kT/m,[1 0 0]);
    if Ki==0
        Cpid = tf([Kd Kp],1);
    else
        Cpid = tf([Kd Kp Ki],[1 0]);
    end
    Tb = feedback(Gb*Cpid,1);
end
```

Convert Gains

There are two sets of gains: 1) (Kd,Kp,Ki) for simple second order model with motor command input 2) (Kdhat,Kphat,Kihat) for parrot drone simulink model with total thrust command input. These are the gains that should be used in the parrot drone Simulink model.

```
Kdhat = (4*kT)*Kd;
Kphat = (4*kT)*Kp;
Kihat = (4*kT)*Ki;
```

Simulate Closed-Loop

```
% Parameter estimates
mhat = m;
kThat = kT;

% Disturbance Force, N
Fd = 0;

% Simulate system

% Allow sim to be called as a function
% and have proper variable scope
options = simset('SrcWorkspace','current');
sim('QuadPID',[0 Tf],options);
```

Plot results

```
font_size = 12;
line_size = 15;
line_width = 1;
color = 'b';

figure();
hold on
text(T_STEADY,1,"w_{n} = " + wn + newline + "zeta = " + zeta);
yline(hdesf,'Linewidth',line_width,'Color','black','DisplayName','z_{ref}')
plot(tsim+T_START,h,'Linewidth',line_width,'Color',color,'DisplayName','Low Sim');
title(sprintf('%s: $\\hat{K_{p}} = %s, \\hat{K_{d}} = %s$',DISP_NAME,num2str(Kphat),num2str(Kdhat)),'Interpreter','latex');
xlabel('Time (s)','fontsize',font_size);
ylabel('Altitude (m)','fontsize',font_size);
legend('show','Location','best');
set(gca,'XMinorGrid','off','GridLineStyle','-','FontSize',line_size)
xlim([T_START T_END+1]);
ylim([0.4 1.5]);
grid on;




time  = [time tsim];
h_arr = [h_arr h];

% Steady-state error
idxs = find(time >= T_STEADY-T_START); % Indices of steady state region
z_arr = h_arr(idxs); % Z values being investigated
```

```
    zs = double(mean(z_arr)); % m, Experimental settling value
    hss = zs - hdesf; % m, Steady state error

    line_name = "h_{ss} = " + hss;
    text(T_STEADY,1.05*zs,line_name,'Color',color)
    yline(zs,"--",'Linewidth',line_width,'Color',color,'HandleVisibility','off')

    % Settling Time
    if zeta < 0.5 % Underdamped case
        ts = 3/(zeta*wn);
        line_name = "ts = " + (ts-1) + " (theory)";
        text(1.01*(ts+T_START),.6,line_name,'Color','r')
        xline(ts+T_START,"--",'Linewidth',line_width,'Color','r','HandleVisibility','off')
    end

    % Find last time z dipped below 95% of z_settle
    ts_idxs = find(h_arr <= 0.95*zs);
    if isempty(ts_idxs)
        ts1 = 0;
    else
        ts1 = time(ts_idxs(end));
    end

    % Find last time z rose above 105% of z_settle
    ts_idxs = find(h_arr >= 1.05*zs);
    if isempty(ts_idxs)
        ts2 = 0;
    else
        ts2 = time(ts_idxs(end));
    end

    ts = max(ts1,ts2); % s, Settling time (use the later time)

    line_name = "ts = " + (ts-1);
    text(1.01*(ts+T_START),.6,line_name,'Color',color)
    xline(ts+T_START,"--",'Linewidth',line_width,'Color',color,'HandleVisibility','off')
```

```
end
```

Published with MATLAB® R2020b