A Design Study Approach to Classical Control

Randal W. Beard   Timothy W. McLain
Brigham Young University

Updated: May 25, 2016

Homework B.13

The objective of this problem is to design an observer that estimates the state of the system and to use the estimated state in the controller designed in Homework B.12.

(a) Modify the simulink diagram from HW B.12 as described in the description of HW A.13 to add a plot of the true state x, and estimated state $\hat{x}$, and the observation error $x - \hat{x}$.

(b) For the sake of understanding the function of the observer, for this problem we will use exact parameters, without an input disturbance. Modify `pendulum_dynamics.m` so that the parameters known to the controller are the actual plant parameters (uncertainty parameter $\alpha = 0$).

(d) In the control block, add an observer to estimate the state $\hat{x}$, and use the estimate of the state in your feedback controller. Tune the poles of the controller and observer to obtain good performance.

(e) As motivation for the next chapter, add an input disturbance to the system of 0.05 and observe that there is steady state error in the response even though there is an integrator. This is caused by a steady state error in the observation error. In the next chapter we will show how to remove the steady state error in the observation error.

Solution

Matlab code used to design the observer based controller is shown below:

```
% inverted Pendulum parameter file
clear all

% system parameters known to controller
P.m1 = 0.25;   % kg
P.m2 = 1;      % kg
P.ell = 0.5; % m
P.b = 0.05; % N m
P.g = 9.8; % m/s^2

% initial conditions
P.z0 = 0;
P.zdot0 = 0;
P.theta0 = 0;
P.thetadot0 = 0;

% input constraint
P.F_max = 5;

% sample rate for controller
P.Ts = 0.01;

% gain for dirty derivative
P.sigma = 0.05;


% state space design
P.A = [...
    0, 0, 1, 0;...
    0, 0, 0, 1;...
    0, -P.m1*P.g/P.m2, -P.b/P.m2, 0;...
    0, (P.m1+P.m2)*P.g/P.m2/P.ell, P.b/P.m2/P.ell, 0;...
];
P.B = [0; 0; 1/P.m2; -1/P.m2/P.ell ];
P.C = [...
    1, 0, 0, 0;...
    0, 1, 0, 0;...
    ];

% form augmented system
```

```
Cout = [1,0,0,0];
A1 = [P.A, zeros(4,1); -Cout, 0];
B1 = [P.B; 0];

% tunning parameters
tr_z = 1.5;  % rise time for position
tr_theta = .5; % rise time for angle
zeta_z   = 0.707; % damping ratio position
zeta_th  = 0.707; % damping ratio angle
integrator_pole = -10;

% compute gains
wn_th    = 2.2/tr_theta; % natural frequency for angle
wn_z     = 2.2/tr_z; % natural frequency for position
des_char_poly = conv(conv([1,2*zeta_z*wn_z,wn_z^2],...
                     [1,2*zeta_th*wn_th,wn_th^2]),...
                     poly(integrator_pole));
des_poles = roots(des_char_poly);

% is the system controllable?
if rank(ctrb(A1,B1))≠5,
    disp('System Not Controllable');
else % if so, compute gains
    K1   = place(A1,B1,des_poles);
    P.K  = K1(1:4);
    P.ki = K1(5);
end

% observer design
wn_th_obs   = 10*wn_th;
wn_z_obs    = 10*wn_z;
des_obsv_char_poly = conv([1,2*zeta_z*wn_z_obs,wn_z_obs^2],...
                      [1,2*zeta_th*wn_th_obs,wn_th_obs^2]);
des_obsv_poles = roots(des_obsv_char_poly);

% is the system observable?
if rank(obsv(P.A,P.C))≠4,
    disp('System Not Observable');
else % if so, compute gains
    P.L = place(P.A', P.C', des_obsv_poles)';
end
```

Matlab code for the observer based control is shown below:

```
function out=pendulum_ctrl(in,P)
    z_r     = in(1);
    z_m      = in(2);
    theta_m = in(3);
    t       = in(4);

    % implement observer
    persistent xhat       % estimated state (for observer)
    persistent F
    if t<P.Ts,
        xhat = [0;0;0;0];
        F    = 0;
    end
    N = 10;
    for i=1:N,
        xhat = xhat + ...
            P.Ts/N*(P.A*xhat+P.B*F...
                    +P.L*([z_m;theta_m]-P.C*xhat));
    end
    zhat = xhat(1)

    % integrator
    error = z_r - zhat;
    persistent integrator
    persistent error_d1
    % reset persistent variables at start of simulation
    if t<P.Ts==1,
        integrator  = 0;
        error_d1    = 0;
    end
    integrator = integrator + (P.Ts/2)*(error+error_d1);
    error_d1 = error;

    % compute the state feedback controller
    F_unsat = -P.K*xhat - P.ki*integrator;
    F = sat( F_unsat, P.F_max);

    % integrator anti-windup
    if P.ki≠0,
       integrator = integrator + P.Ts/P.ki*(F-F_unsat);
    end

    out = [F; xhat];

end
```

```
%--------------------------------------------------------------------------
% saturation function
function out = sat(in,limit)
    if     in > limit,      out = limit;
    elseif in < -limit,     out = -limit;
    else                    out = in;
    end
end
```

See the wiki for the complete solution.