A Design Study Approach to Classical Control

Randal W. Beard   Timothy W. McLain
Brigham Young University

Updated: May 26, 2016

Homework B.14

(a) Modify your solution from HW B.13 so that the uncertainty parameter in `pendulum_dynamics.m` is $\alpha = 0.2$, representing 20% inaccuracy in the knowledge of the system parameters, and so that the input disturbance is 0.5. Also, add noise to the output channels z_m and θ_m with standard deviation of 0.001. Before adding the disturbance observer, run the simulation and note that the controller is not robust to the large input disturbance.

(b) Add a disturbance observer to the controller, and verify that the steady state error in the estimator has been removed. Tune the system to get good response.

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
% form augmented system for disturbance observer
A2 = [P.A, P.B; zeros(1,4), zeros(1,1)];
C2 = [P.C, zeros(2,1)];
% pick observer poles
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);
dist_obsv_pole = -1;

% is the system observable?
if rank(obsv(A2,C2))≠5,
    disp('System Not Observable');
else % if so, compute gains
    L2 = place(A2', C2', [des_obsv_poles;dist_obsv_pole])';
    P.L = L2(1:4,:);
    P.Ld = L2(5,:);
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 dhat       % estimate disturbance
    persistent F          % delayed input (for observer)
    if t<P.Ts,
        xhat = [0;0;0;0];
        dhat = 0;
        F    = 0;
    end
    N = 10;
    for i=1:N,
        xhat = xhat + ...
            P.Ts/N*(P.A*xhat+P.B*(F+dhat)...
                    +P.L*([z_m;theta_m]-P.C*xhat));
        dhat = dhat + P.Ts/N*P.Ld*([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 - dhat;
    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.