# A Design Study Approach to Classical Control

Randal W. Beard Timothy W. McLain
Brigham Young University

Updated: May 26, 2016

## Homework C.14

**(a)** Modify the parameter file from HW C.18 to use inexact parameters, as in HW C.15 and HW C.17. Explicitly add an additional input disturbance of 1 Newton-meters. Before adding the disturbance observer, run the simulation and observe the bias in the state. The integrator anti-windup that limits integral action when $\dot{\phi}$ is small will likely not work because of the state bias. You may need to comment out the anti-windup scheme for the integrator to work, in which case you will get large steady state error.

**(b)** Add a disturbance observer to the controller, and verify that the steady state error in the estimator has been removed. Experiment with the system to understand the response with and without the integrator, and with and without anti-windup.

**(a)** Modify your solution from HW C.13 so that the uncertainty parameter in `satellite_dynamics.m` is $\alpha = 0.2$, representing 20% inaccuracy in the knowledge of the system parameters, and so that the input disturbance is 1.0. Also, add noise to the output channels $z_m$ and $\theta_m$ with standard deviation of 0.001.

**(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:

```
clear all

% initial conditions
P.theta0    = 0;
P.phi0      = 0;
P.thetadot0 = 0;
P.phidot0   = 0;

% system parameters known to controller
P.Js = 5; % kg m^2
P.Jp = 1;  % kg m^2
P.k = 0.15; % N m
P.b = 0.05; % N m s

% maximum torque
P.taumax = 5; %Nm

% 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;...
    -P.k/P.Js, P.k/P.Js, -P.b/P.Js, P.b/P.Js;...
    P.k/P.Jp, -P.k/P.Jp, P.b/P.Jp, -P.b/P.Jp;...

];
P.B = [0; 0; 1/P.Js; 0];
P.C = [...
    1, 0, 0, 0;...
    0, 1, 0, 0;...
    ];

% form augmented system
Cout = [0,1,0,0];
A1 = [P.A, zeros(4,1); -Cout, 0];
```

```
B1 = [P.B; 0];

% tuning parameters
wn_th   = 0.6;
zeta_th = 0.707;
wn_phi    = 1.1;
zeta_phi  = 0.707;
integrator_pole = -1;

% gain selection
ol_char_poly = charpoly(P.A);
des_char_poly = conv(conv([1,2*zeta_th*wn_th,wn_th^2],...
                [1,2*zeta_phi*wn_phi,wn_phi^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_phi_obs    = 10*wn_phi;
dist_obsv_pole = -1;
des_obsv_char_poly = conv(conv(...
                [1,2*zeta_phi*wn_phi_obs,wn_phi_obs^2],...
                [1,2*zeta_th*wn_th_obs,wn_th_obs^2]),...
                poly(dist_obsv_pole));
des_obsv_poles = roots(des_obsv_char_poly);

% 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)';
    P.L = L2(1:4,:);
```

```
    P.Ld = L2(5,:);
end
```

Matlab code for the observer based control is shown below:

```
function out=satellite_ctrl(in,P)
    phi_r     = in(1);
    theta_m   = in(2);
    phi_m     = in(3);
    t         = in(4);

    % implement observer
    persistent xhat       % estimated state (for observer)
    persistent dhat       % estimate disturbance
    persistent tau        % delayed input (for observer)
    if t<P.Ts,
        xhat = [0;0;0;0];
        dhat = 0;
        tau    = 0;
    end
    N = 10;
    for i=1:N,
        xhat = xhat + ...
            P.Ts/N*(P.A*xhat+P.B*(tau+dhat)...
                    +P.L*([theta_m;phi_m]-P.C*xhat));
        dhat = dhat + P.Ts/N*P.Ld*([theta_m;phi_m]-P.C*xhat);
    end
    phihat = xhat(2);

    % integrator
    error = phi_r - phihat;
    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
    tau_unsat = -P.K*xhat - P.ki*integrator - dhat;
    tau = sat( tau_unsat, P.taumax);
```

```

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

    out = [tau; 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.