A Design Study Approach to Classical Control

Randal W. Beard   Timothy W. McLain
Brigham Young University

Updated: August 29, 2016

Homework B.10

The objective of this problem is to implement the PID controller using Matlab code using only measured outputs of the system.

(a) Modify the system dynamics file so that the parameters m_1, m_2, ℓ and b vary by up to 20% of their nominal value each time they are run (uncertainty parameter $= 0.2$).

(b) Rearrange the block diagram so that the controller is implemented as an m-function implemented at the sample rate of $T_s = 0.01$. The controller should only assume knowledge of the position z and the angle θ, as well as the reference position z_r.

(e) Implement the nested PID loops designed in Problems B.8 using an m-function called pendulum_ctrl.m. Use the dirty derivative gain of $\tau = 0.05$. Tune the integrator to remove the steady state error caused by the uncertain parameters.

Solution

See http://controlbook.byu.edu for the complete solution.

The Matlab code for the controller is shown below.

```
function F=pendulum_ctrl(in,P)
    z_r   = in(1);
```

```
    z     = in(2);
    theta = in(3);
    t     = in(4);

    % set persistent flag to initialize integrators and
    % differentiators at the start of the simulation
    persistent flag
    if t<P.Ts,
        flag = 1;
    else
        flag = 0;
    end

    % compute the desired angled angle using the outer loop control
    theta_r = PID_z(z_r,z,flag,P.kp_z,P.ki_z,P.kd_z,...
                    30*pi/180,P.Ts,P.sigma);
    % compute the force using the inner loop
    F       = PD_th(theta_r,theta,flag,P.kp_th,P.kd_th,...
                     P.F_max,P.Ts,P.sigma);

end

%------------------------------------------------------------------------
% PID control for position
function u = PID_z(z_c,z,flag,kp,ki,kd,limit,Ts,sigma)
    % declare persistent variables
    persistent integrator
    persistent zdot
    persistent error_d1
    persistent z_d1
    % reset persistent variables at start of simulation
    if flag==1,
        integrator  = 0;
        zdot    = 0;
        error_d1    = 0;
        z_d1    = 0;
    end

    % compute the error
    error = z_c-z;
    % update derivative of z
    zdot = (2*sigma-Ts)/(2*sigma+Ts)*zdot...
           + 2/(2*sigma+Ts)*(z-z_d1);
    % update integral of error
    if abs(zdot)<.1,
```

```
            integrator = integrator + (Ts/2)*(error+error_d1);
        end
        % update delayed variables for next time through the loop
        error_d1 = error;
        z_d1     = z;

        % compute the pid control signal
        u_unsat = kp*error + ki*integrator - kd*zdot;
        u = sat(u_unsat,limit);

        % integrator anti-windup
        if ki≠0,
            integrator = integrator + Ts/ki*(u-u_unsat);
        end
    end


    %-----------------------------------------------------------------
    % PID control for angle theta
    function u = PD_th(theta_c,theta,flag,kp,kd,limit,Ts,sigma)
        % declare persistent variables
        persistent thetadot
        persistent theta_d1
        % reset persistent variables at start of simulation
        if flag==1,
            thetadot     = 0;
            theta_d1     = 0;
        end

        % compute the error
        error = theta_c-theta;
        % update derivative of y
        thetadot = (2*sigma-Ts)/(2*sigma+Ts)*thetadot...
                   + 2/(2*sigma+Ts)*(theta-theta_d1);
        % update delayed variables for next time through the loop
        error_d1 = error;
        theta_d1 = theta;

        % compute the pid control signal
        u_unsat = kp*error - kd*thetadot;
        u = sat(u_unsat,limit);

    end

    %-----------------------------------------------------------------
```

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