**param.m**

clear all

% system parameters

P.g = 9.81;

P.l = 0.25;

P.m = 0.1;

P.k1 = 0.02;

P.k2 = 0.01;

P.b = 0.1;

% max torque available

P.tau\_max = 3;

P.tau\_e = P.m\*P.g\*P.l;

P.theta0 = 0;

P.thetadot0 = 0;

% sample time for controller

P.Ts = 0.01;

% dirty derivative time constant

P.tau = 0.005;

% PART 3

% proportional gain

P.kp = (P.tau\_max-P.tau\_e)/(20\*pi/180);

% natural frequency

P.wn = sqrt((P.kp+P.k1)/(P.m\*P.l^2));

%zeta

P.zeta = 0.707;

% derivative gain

P.kd = 2\*P.zeta\*P.wn\*P.m\*P.l^2 - P.b;

% integrator gain

P.ki = 10;

% PART 4

w\_c = 30;

ml\_sqr = P.m\*P.l^2;

num = 1/ml\_sqr;

den = [1 P.b/ml\_sqr P.k1/ml\_sqr];

s = tf('s');

H = tf(num,den);

% sisotool(H)

phase\_desired = 70;

[mag,phase] = bode(H, w\_c);

phi\_max = phase\_desired-(phase+180);

phi\_max\_rad = phi\_max\*pi/180;

alpha\_lead = (1-sin(phi\_max\_rad))/(1+sin(phi\_max\_rad));

z\_lead = w\_c\*sqrt(alpha\_lead);

p\_lead = w\_c/sqrt(alpha\_lead);

gain = 14;

D\_lead = tf(gain\*[1 z\_lead], [1 p\_lead]);

ess\_desired = 0.3;

alpha\_lag = 2.2/ess\_desired;

z\_lag = w\_c/10;

p\_lag = z\_lag/alpha\_lag;

D\_lag = tf([1 z\_lag], [1 p\_lag]);

% bode plots

% hold off;

% hold on;

% bode(H)

% bode(H\*D\_lead)

% bode(H\*D\_lead\*D\_lag)

% legend('Open loop', 'Lead', 'Lead-Lag')

% grid on

% PART 5

P.F = [0 1; -P.k1/ml\_sqr -P.b/ml\_sqr];

P.G = [0; 1/ml\_sqr];

P.H = [1 0];

P.J = 0;

pp = [-25+15j -25-15j];

P.K = place(P.F,P.G,pp);

NN = inv([P.F P.G; P.H P.J])\*[0;0;1];

P.Nx = NN(1:2);

P.Nu = NN(3);

P.Nbar = P.Nu + P.K\*P.Nx;

pe = 5\*pp;

P.LL = place(P.F',P.H',pe)';

% PART 6

rltool(H)

**rodmass\_dynamics.m**

%=============================================================================

% mdlDerivatives

% Return the derivatives for the continuous states.

%=============================================================================

%

function sys=mdlDerivatives(t,x,u,P)

theta = x(1);

thetadot = x(2);

tau = u(1);

% state equations go here...

ml\_sqr = P.m\*P.l^2;

thetaddot = -P.b\*thetadot/ml\_sqr - P.k1\*theta/ml\_sqr - P.k2\*theta^3/ml\_sqr - P.g\*cos(theta)/P.l + tau/ml\_sqr;

sys = [thetadot; thetaddot];

% end mdlDerivatives

**ctrl\_pid.m**

function tau = ctrl\_pid(in,P)

theta\_c = in(1);

theta = in(2);

t = in(3);

% dirty derivative equation

% xdot = (2\*tau-Ts)/(2\*tau+Ts)\*xdot + 2/(2\*tau+Ts)\*(x-x\_d1);

% integrator equation

% integrator = integrator + (Ts/2)\*(error+error\_d1);

persistent flag

if t< P.Ts,

flag = 1;

else

flag = 0;

end

tau\_tilde = PID\_th(theta\_c,theta,flag,P.kp,P.ki,P.kd,P.tau\_max,P.Ts,P.tau);

tau = P.tau\_e + tau\_tilde;

end

%------------------------------------------------------------

% PID control for angle theta

function u = PID\_th(theta\_c,theta,flag,kp,ki,kd,limit,Ts,tau)

% declare persistent variables

persistent integrator

persistent thetadot

persistent error\_d1

persistent theta\_d1

% reset persistent variables at start of simulation

if flag==1,

integrator = 0;

thetadot = 0;

error\_d1 = 0;

theta\_d1 = 0;

end

% compute the error

error = theta\_c-theta;

% update integral of error

integrator = integrator + (Ts/2)\*(error+error\_d1);

% update derivative of y

thetadot = (2\*tau-Ts)/(2\*tau+Ts)\*thetadot + 2/(2\*tau+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 + ki\*integrator - kd\*thetadot;

u = sat(u\_unsat,limit);

% integrator anti-windup

if ki~=0,

integrator = integrator + Ts/ki\*(u-u\_unsat);

end

end

function out = sat(in,limit)

if in > limit, out = limit;

elseif in < -limit, out = -limit;

else out = in;

end

end

**ctrl\_est.m**

function out = ctrl\_est(in,P)

theta\_c = in(1);

theta = in(2);

t = in(3);

x = in(4:5);

persistent xhat\_

persistent tau

% equilibrium state, input, output

x\_e = [theta\_c; 0];

tau\_e = P.k1\*theta\_c + P.k2\*theta\_c^3+P.m\*P.g\*P.l\*cos(theta\_c);

N = 10;

if t<P.Ts,

xhat\_ = [0; 0];

tau = 0;

else

for i=1:N,

xhat\_ = xhat\_ + P.Ts/N\*(P.F\*xhat\_+P.G\*tau+ P.LL\*(theta-P.H\*xhat\_));

end

end

%xhat\_-x

xhat = xhat\_; % use estimated state

% xhat = x; % use true state

% state feedback with nonlinear feedforward

tau = -P.K\*(xhat-x\_e) + tau\_e;

out = [tau; xhat];

end

function out = sat(in,limit)

if in > limit, out = limit;

elseif in < -limit, out = -limit;

else out = in;

end

end