GUI Representation of Discrete-Time Compensator Design

Introduction:

In an undergraduate Control Systems class a major goal is to have the students understand how to design a discrete-time compensator. However, a significant part of understanding this design is how changing the different inputs to the design process (percent overshoot, settling time and sampling time) affect the design of the controller. Once the student understands how to do the compensator design, repeating the process multiple times simply consumes a large amount of time for little educational gain.

O’Brien and Watkins created a unified method of controller design that can be used to address this problem(Is this true? They wrote the code?). However, taking their code a step further, we have written a Graphical User Interface (GUI) that takes user input and outputs the controller transfer function, uncompensated step response, uncompensated root locus or bode response, compensated root locus or bode response, and compensated step response. This allows the student to easily change the input parameters of the compensator design and immediately see the affect on the system response.

In this paper, we consider Bode and Root Locus Proportional, Proportional-Integral, and Proportional-Derivative controller design.

Developing the GUI:

The design of the GUI was driven by the commonalities to the three types of controllers being designed: the type of analysis, the type of controller, the settling time, the percent overshoot, the sampling time, and the numerator and denominator of the continuous-time transfer function. For ease of creating and because this GUI was solely for personal use, I used editable text boxes to obtain all of this information and simply labeled each text box with a static text box. The Generate button uses a function as its callback that takes these inputs, does the controller design, and outputs the transfer function of the controller to another static text box displayed on the right side of the GUI. The compensated and uncompensated responses are displayed on pop-up windows.

Coding the Compensator Design:

One of the advantages to having a student create this GUI is that they have to write the code for the GUI to run. This helps the student understand compensator design and gives the student experience coding in MATLAB and interfacing that code to a GUI. For an undergraduate with only a few engineering courses completed, all of these objectives are valuable.

The GUI function begins by creating a handle for each of the text boxes with user input. These handles are used to extract the information from the box and convert it to the required type, whether String, double, or vector. This information is separated based on the type of compensator desired and the type of analysis desired. The code, as written by the student, follows those outlined by O’Brien and Watkins.

Example:

(Need to do example here)

Conclusion:

As was the objective for O’Brien and Watkins, creating a GUI in this way ensures that the students understand the process of creating a control system and have the freedom to adjust input parameters as needed to adjust the closed-loop performance without having to perform the labor of the calculations. This GUI can easily be developed by students at the end of their first course by following the procedures outlined in the reference paper.

h\_analysis = findobj('Tag','AnalysisBox');

h\_type = findobj('Tag','TypeBox');

h\_settle = findobj('Tag','SettleTime');

h\_overshoot = findobj('Tag','PO');

h\_sample = findobj('Tag','SampleTime');

h\_num = findobj('Tag','NumBox');

h\_den = findobj('Tag','DenBox');

h\_control = findobj('Tag','Controller');

analysis = get(h\_analysis , 'String');

type = get(h\_type,'String');

Ts = str2double( get(h\_settle,'String'));

po = str2double( get(h\_overshoot,'String'));

T = str2double( get(h\_sample,'String'));

num = str2num( get(h\_num,'String'));

den = str2num( get(h\_den,'String'));

Gs = tf(num,den);

Gz = c2d(Gs,T);

po = po/100;

phi=atan(-pi/log(po));

zeta=cos(phi);

wn=4/(zeta\*Ts);

figure(1); step(Gz); title('Uncompensated Step Response');

if(strcmp(analysis,'Bode'))

% calculate phase margin

PM=atan(2\*zeta/sqrt(-2\*zeta^2+sqrt(1+4\*zeta^4)));

% calculate gain crossover frequency

wgc=2\*zeta\*wn/tan(PM);

% find continuous time design point

sd=j\*wgc;

% find discrete time design point

zd=exp(sd\*T);

figure(2); bode(Gz); title('Uncompensated Bode Response');

X=evalfr(Gz,zd);

thc=PM-pi-angle(X);

else % Root Locus Design

sigmad=zeta\*wn;

wd=wn\*sqrt(1-zeta^2);

% Continuous and discrete time design points

sd=-sigmad+j\*wd;

zd=exp(sd\*T);

figure(2);

rlocus(Gz); title('Uncompensated Root Locus');

hold on;

line(real(zd),imag(zd),'Marker','^');

hold off;

X=evalfr(Gz,zd);

thc=pi-angle(X);

end

if(strcmp(type,'PI'))

xd=real(zd);

yd=imag(zd);

thp=angle(zd-1);

thz=thc+thp;

al=xd-yd/tan(thz);

Dz=zpk(al,1,1,T);

Y=evalfr(Dz,zd);

K=1/(abs(X)\*abs(Y));

Dz=K\*Dz;

Gop=Dz\*Gz;

Gcl=feedback(Gop,1);

figure(3); bode(Gop); title('Compensated Bode Response'); grid on;

figure(4); step(Gcl); title('Step Response');grid on;

[z,p,k,Ts] = zpkdata(Dz);

str=[k(1) '(z-' z(1) ')/(z-1)'];

set(h\_control, 'String', str);

elseif(strcmp(type,'PD'))

xd=real(zd);

yd=imag(zd);

thp=angle(zd);

thz=thc+thp;

al=xd-yd/tan(thz);

Dz=zpk(al,0,1,T);

Y=evalfr(Dz,zd);

K=1/(abs(X)\*abs(Y));

Dz=K\*Dz;

Gop=Dz\*Gz;

Gcl=feedback(Gop,1);

figure(3); step(Gcl); title('Compensated Step Response');grid on;

[z,p,k,Ts] = zpkdata(Dz);

str=[k(1) '(z-' z(1) ')/z'];

set(h\_control, 'String', str);

else % Proportional

K=1/abs(X);

Gop=K\*Gz;

Gcl=minreal(K\*Gz/(1+K\*Gz));

figure(3); step(Gcl); title('Compensated Step Response');

set(h\_control, 'String', ['K= ' num2str(K)]);

end

if(strcmp(analysis,'Bode'))

figure(4); bode(Gop); title('Compensated Bode Response');

else

figure(4);rlocus(Gop);title('Compensated root locus');

hold on

line(real(zd),imag(zd),'Marker','^');

hold off

end