5 Module 5: Analog to Digital Transforms

5.1 Overview

This module will examine the differences between digital and analog representations of systems. We will look at ways of transforming an analog model of a system into an digital model. There is no exact digital equivalent to an analog system. We will study some of the different ways of doing this.

The background material for this module is Chapter 3, Chapter 4 (except Section 4.3) and Chapter 6 of the text. I suggest reading this material before, or in conjunction with, working on this module.

5.2 Notes

This module introduces some of the differences between analog and digital design. We will find significant differences between analog and digital systems, even when the digital system is supposed to be representative of the analog system.

For example, if we have a second order, stable system, it is stable for any proportional negative feedback gain. To see this think of where the root locus lies: entirely in the left half plane. We will see here that such a system can be destabilized with a proportional digital controller. The underlying reason is that the zero-order hold/sampler configuration introduces phase. Between the samples, the input to the plant is fixed, and it is essentially running open-loop.

The generic framework is shown in Figure 1, below. The reference input, $r(k)$, will be considered to be in the discrete time domain, and we are interested in the output, $y(k)$, only in the discrete time domain.

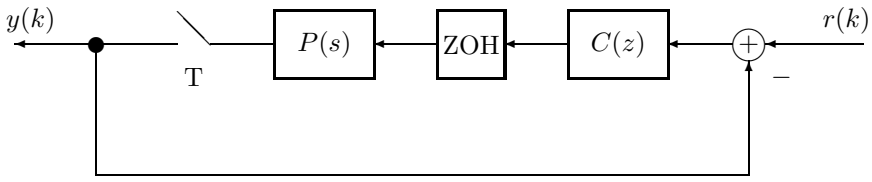


Figure 1: Closed loop control system — continuous plant and discrete controller

The basic idea is to transform the combination of a zero order hold, continuous system, and sampler into an equivalent discrete time system. This is illustrated schematically in Figure 2. $P(z)$ can then be analyzed in a closed loop configuration with $C(z)$.

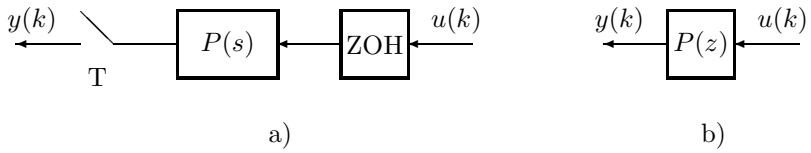


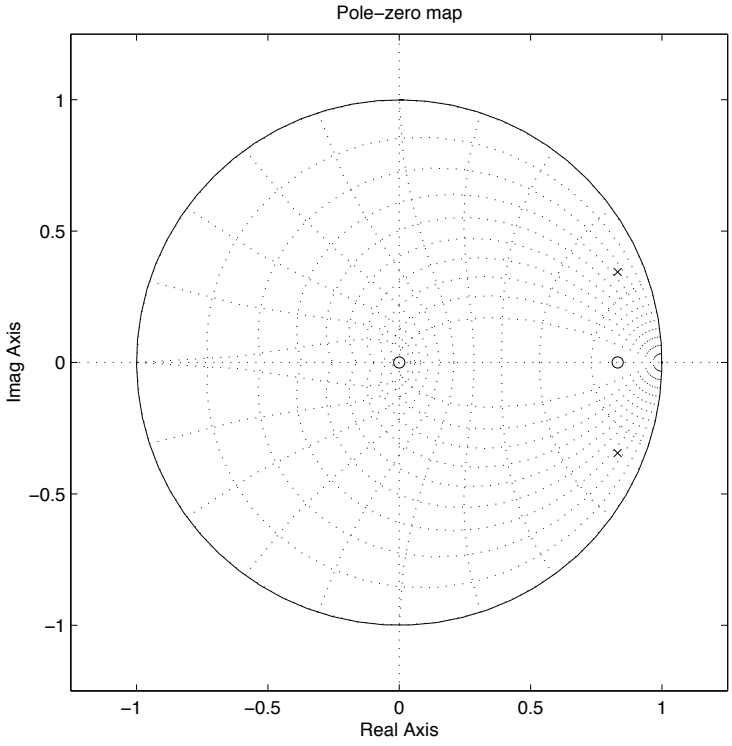
Figure 2: Zero order hold equivalence. a) Actual system: sampler, $P(s)$ and zero order hold. b) Discrete time equivalent, $P(z)$

The analysis and design used here is primarily based on root locus. The root locus graphical rules apply to polynomials in z as well as s, and the rules are the same. However, we are now interested in whether the roots stay in the unit disk, rather than whether they stay in the left half plane.

5.3 Module 5: Listing and Graphics

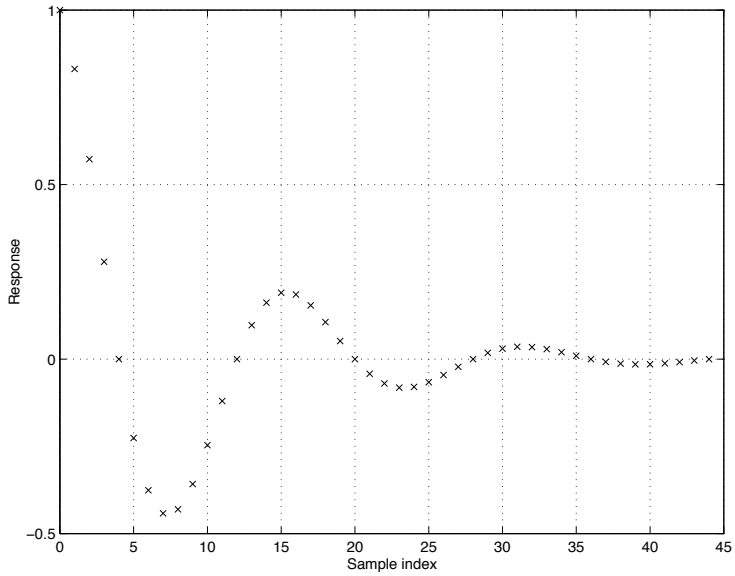
```
clf;axis([1,2,3,4]);axis;
echo on
%-----------------------------------------------------------
%
%          Dept. of Electrical & Computer Eng.
%       University of California, Santa Barbara.
%
%               ECE 147 B:  Digital Control Systems
%
%               module5:    Analog to Digital Transforms
%
%       ------------------------------------------------

% modified: RSS; 4/Feb/97.  Incorrect graph label fixed.
%       modified: RSS; 12/Jan/99. Upgraded to Matlab 5, mutools
%                 dependent code removed.
%       modified: RSS; 22/Jan/2009: stepfun calls removed (obsolete).
%
% This module looks at the differences between the
% analog and digital representation of systems in
% Matlab.  Various transformations, and their
% properties are studied.  The relevant sections in the
% text are Chapters 2 and 4.
%
% Please adjust the Graph and Command windows for maximum size
% viewing before proceeding.

pause % press any key to continue.

% Consider a digital system with 2 poles and 2 zeros.  We
% will form this system in terms of polar co-ordinates.
% Recall that this formulation was used in the lecture
% notes.

r = 0.9; % pole radius
theta = pi/8; % pole angle

% The transfer function we will look at is:
%
%           z(z - r cos(theta))
%  P =  ---------------------------
%        z^2 - 2r cos(theta)z + r^2
%

T = -1;
```

```
p = tf([1,-r*cos(theta),0], [1,-2*r*cos(theta),r^2], T);

% Note that we have used a sample time of T = -1, which
% signifies that it is unspecified.

pause % press any key to continue.

% Let's check the poles and zeros of p.

clf
zgrid
axis('square')
axis([-1.25,1.25,-1.25,1.25])
pzmap(p)
```



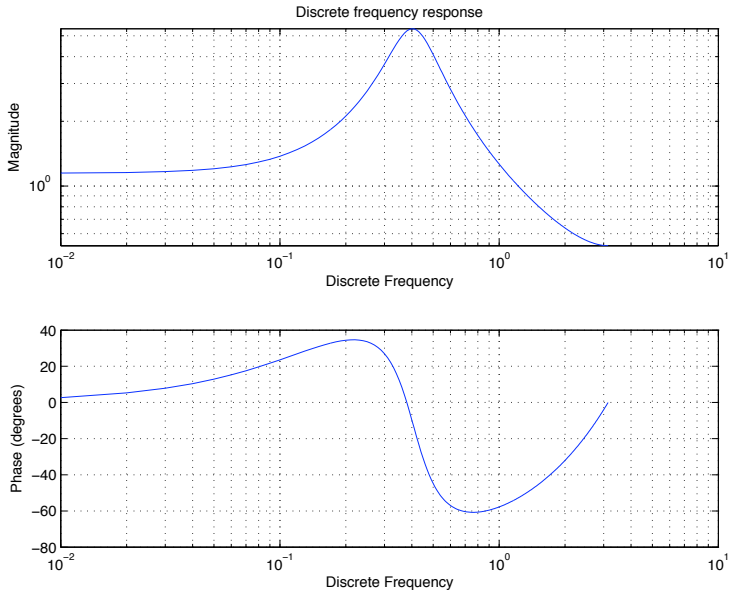

```
% Note that both poles are inside the unit circle.  We
% can check the radius of each pole via:

abs(pole(p))

%  We can also check the angle.

angle(pole(p))
```

```
% It matches up as we expect.

pause   % press any key to continue

% This is exactly the pole-zero pattern referred to in
% the lecture.  We already know the time response to this
% pattern.

pause   % press any key to continue.

% The pulse response is calculated via the impulse
% command (it will also do the impulse response of a
% continuous time system).

kfinal = 45;
y = impulse(p,kfinal);
clf
plot([0:kfinal-1],y,'x')
grid
xlabel('Sample index')
ylabel('Response')
```



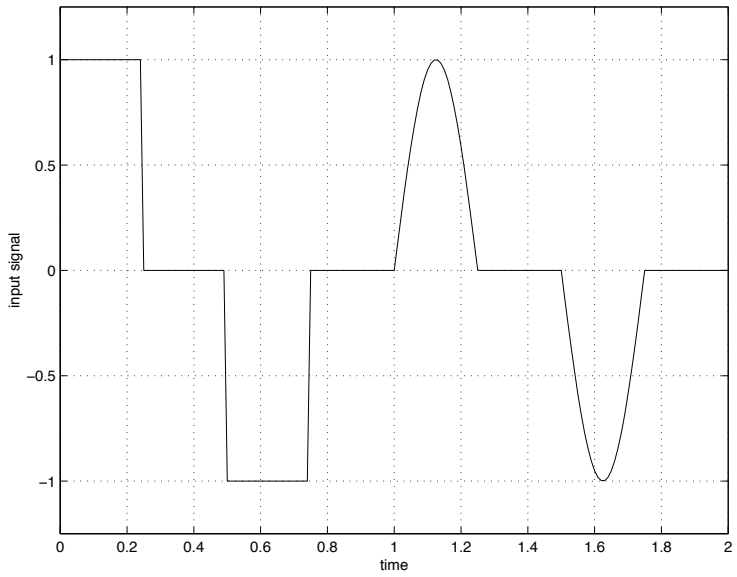

```
% Even though the impulse command generates a plot it
% is better not to use it for discrete time systems.
% Instead, plot it via a plot command, taking into account
% the fact that the response begins at zero.  It helps
% to use a point-style plot so that Matlab doesn't join
```

```
% the dots.  After all the output is only defined at the
% specified indices.  Joining the dots is meaningless and
% can actually be misleading.

pause % press any key to continue.

% Now we will look at the frequency response.  There are
% several functions available.  Useful ones include
% bode (which gives gain and phase) and freqresp (which
% gives a complex valued response).  The most useful is
% freqresp because you can decide how to format the plot.

Omega = [0:0.01:pi]';   % specify discrete freq. vector.
p_f = freqresp(p,Omega);

p_f = squeeze(p_f);     % this reduces p_f to a vector

subplot(2,1,1)
loglog(Omega,abs(p_f))
xlabel('Discrete Frequency')
ylabel('Magnitude')
title('Discrete frequency response')
grid

subplot(2,1,2)
semilogx(Omega,angle(p_f)*360/(2*pi))
xlabel('Discrete Frequency')
ylabel('Phase (degrees)')
grid
```

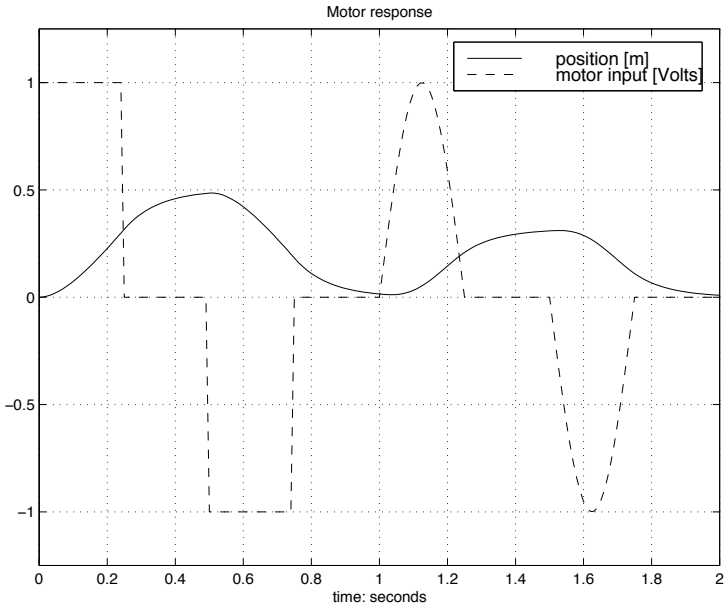



```
% In discrete systems the frequency response is evaluated
% around the unit circle.  The axis in this case is really
% the angle from zero (or a small number on a log scale) to
% pi.  In the case where Ts = -1,  the frequency response uses
% T = 1 in the mapping:  omega = exp(j*Omega*T);

pause   % press any key to continue.

% Now we will study the various discrete approximations
% to continuous transfer functions.  The Matlab function
% of interest here is c2d which uses a string argument
% to implement:
%
%      'zoh'            Zero order hold equivalent
%      'foh'            First order hold equivalent
%      'tustin'         Bilinear (Tustin) approximation.
%      'prewarp'        Tustin approximation with frequency prewarping.
%      'matched'        Matched pole-zero method (for SISO systems only).
%

pause   % press any key to continue

% OK, lets make up a continuous system to work with.

% Consider a DC motor, with a voltage input and a
% position output.  Recall that this has a transfer
% function of the form,
%
```

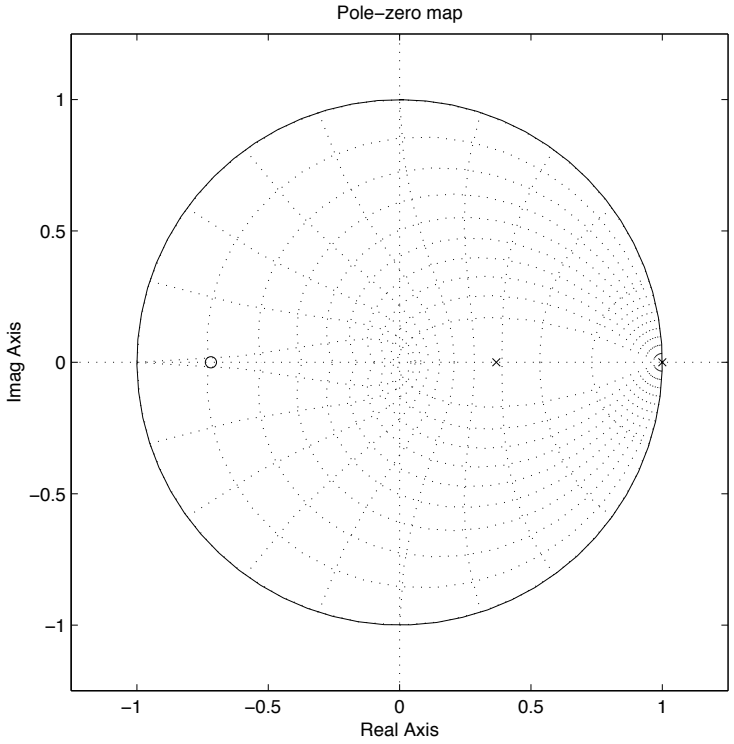
```
%       x             A
%     ----    =   ---------  =: Motor(s),
%       v          s(s + B)
%
% where v is the input voltage and x is the position.
% The continuous system is denoted by M,  We can also label
% the inputs and outputs.

A = 20;
B = 10;
Motor = tf(A,[1,B,0]);
set(Motor,'InputName','motor input [Volts]');
set(Motor,'OutputName','position [m]');

% Check the poles:

pole(Motor)

pause   % press any key to continue.

% Now we will simulate this to give a basis for comparison
% later.  We have to be careful here because there is an
% integrator in the system.  Put in a +ve and then -ve pulse,
% followed by two opposite sign half cycles of a sine wave.
% This should bring the system back to the same position.

% We build this up in pieces....
% Note the use of logical tests (<=, &, etc) as masks and .*
% as element-by-element multiplication to build the signal.

tvec = [0:0.01:2]';
u = (tvec <= 0.25);      % u = 1 for t <= 0.25
u = u + -1*((tvec >= 0.5)&(tvec <= 0.75));
u = u + sin(4*pi*(tvec-1)).*((tvec >= 1)&(tvec <= 1.25));
u = u - sin(4*pi*(tvec-1.5)).*((tvec >= 1.5)&(tvec <= 1.75));

clf
plot(tvec,u,'-')
axis([0,max(tvec),-1.25,1.25])
xlabel('time')
ylabel('input signal')
grid

pause   % press any key to continue.
```




```
% Now do a simulation for the specified voltage input.

[y,t] = lsim(Motor,u,tbase);

%  Notice the use of the named inputs and outputs here.

clf
plot(t,y,'-',tbase,u,'--')
axis([0,max(t),-1.25,1.25])
xlabel('time: seconds')
title('Motor response')
legend(char(get(Motor,'OutputName')),char(get(Motor,'InputName')))
grid

pause   % press any key to continue.
```

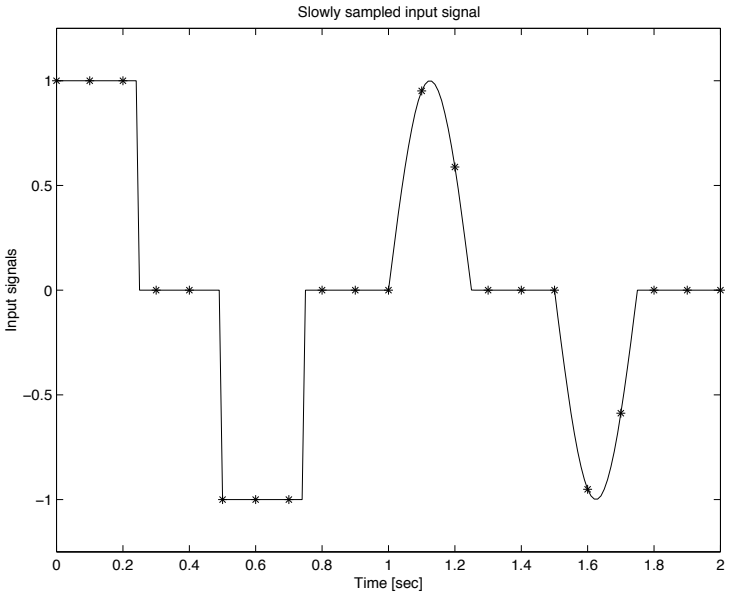



```
% Now we will look at some approximations to this.  The
% first one to try is a ZOH equivalence.  This gives
% the discrete system that would arise if we put a zero-order
% hold before the motor and sampled the output.

% We will choose a large sampling time to see the effects
% of sampling

Ts = 0.1;               % sample period.

% Now form the zero order hold equivalent.  We can append
% a label so that we know how Motorz was derived.

Motorz = c2d(Motor,Ts,'zoh');
set(Motorz,'Notes','ZOH equivalent of Motor');

pause   % press any key to continue

% Where are the poles now?

clf
zgrid
axis('square')
axis([-1.25,1.25,-1.25,1.25])
pzmap(Motorz)
```

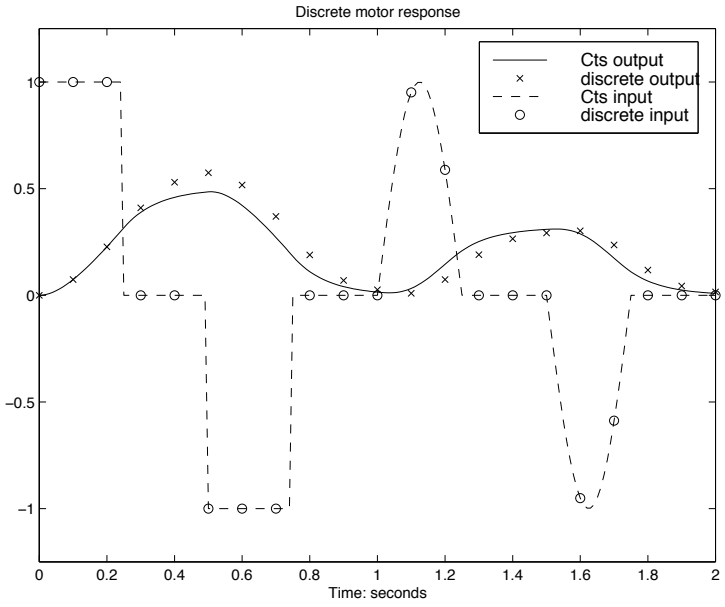



```
% The integrator is still there (pole at 1) and it
% shows up as marginally stable.  The other pole is
% stable as we expect.

pause   % press any key to continue.

% Lets try the simulation - this time for the ZOH equivalent.
% We will use the same input.  Now we have to sample this
% input at 0.1 seconds.  Recall that the signal u has a
% spacing of 0.01 seconds (i.e 10x as fast).

Tsbase = zeros(ceil(length(tbase)/10),1);
uT = Tsbase;
j = 1;
for i = 1:10:length(tbase),
  Tsbase(j) = tbase(i); uT(j) = u(i); j = j+1; end

% Check this by plotting the result

clf;
plot(tbase,u,'-',Tsbase,uT,'*')
axis([0,max(t),-1.25,1.25])
xlabel('Time [sec]')
```

```
ylabel('Input signals')
title('Slowly sampled input signal')

pause   % press any key to continue.
```

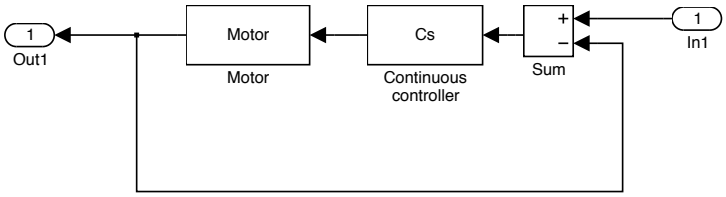



```
% OK. Now we can simulate the equivalent discrete system.
% Again, the lsim command is used.

yT = lsim(Motorz,uT,Tsbase);

clf
plot(t,y,'-',Tsbase,yT,'x',tbase,u,'--',Tsbase,uT,'o')
axis([0,max(t),-1.25,1.25])
xlabel('Time: seconds')
title('Discrete motor response')
legend('Cts output','discrete output','Cts input','discrete input')
```



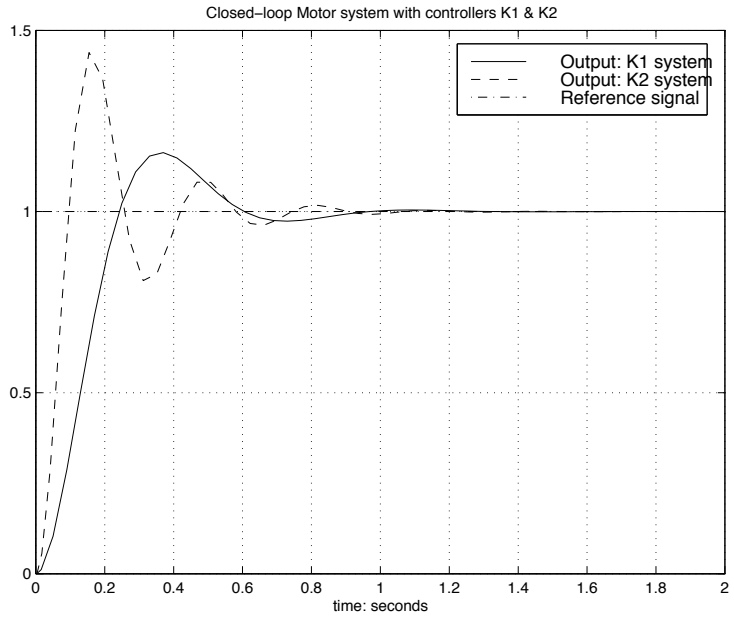

```
% The continuous time response is also shown.  Note that the
% output differs in the discrete case.  To see why, look at
% the input samples, imagine a zero-order hold with each of
% them, and note that this is not the same as the original
% input.  The output is qualitatively similar.

pause   % press any key to continue

% Now we will design a controller for our motor.  The first
% thing that we will do is design a continuous controller
% for the continuous plant, Motor.  Then we will discretize
% that controller and see what the resulting system looks
% like.

% The easiest thing to look at initially is a proportional
% controller.  Picture in your mind a root locus diagram
% for the original two pole system.   (It's easy.  For a
% warm up exercise, do this when you have finished running
% this module).

% We will look at two choices for K.

K1 = 5;
K2 = 20;

% Now form the closed loop, unity gain negative feedback
% systems, for each controller.  This is done using
```

```
% a simulink model: m5cts.  This simply specifies a controller
% K, which we will initially set to K1.

Cs = tf(K1,1);
m5cts
```



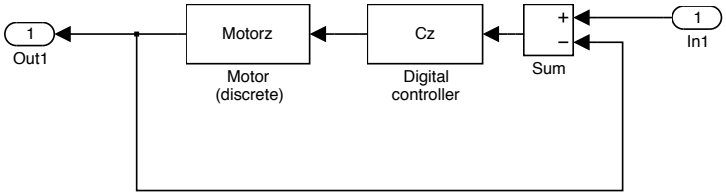

```
pause   % press any key to continue.

% We can analyze these closed-loop systems via the following
% procedure.  The results are labeled clpK1 and clpK2 with
% K1 and K2 respectively.  The ss() function creates an LTI
% state-space system which is equivalent to a transfer function.
% We will deal with state-space realizations of transfer
% functions later in the course.

[a,b,c,d] = linmod('m5cts');
clpK1 = ss(a,b,c,d);

% Now set up for the other controller

Cs = tf(K2,1);
[a,b,c,d] = linmod('m5cts');
clpK2 = ss(a,b,c,d);
clear a b c d

% Check the poles for these two systems:

pole(clpK1)

pole(clpK2)

% Naturally, both are stable.  One is a bit oscillatory.

pause   % press any key to continue

% Now simulate these with a step input as the reference
% to be tracked.  In this case we will use the command line
% form of simulink for the simulation.  Note that we could
% also use lsim(clpK1,...) etc to achieve the same result.
```

```
ut = [0:0.01: 2]';
ustep = ones(length(ut),1);

Cs = tf(K1,1);
tfinal = 2;
[yt1,x1,yclpK1] = sim('m5cts',tfinal,[],[ut,ustep]);
Cs = tf(K2,1);
[yt2,x2,yclpK2] = sim('m5cts',tfinal,[],[ut,ustep]);

clf
plot(yt1,yclpK1,'-',yt2,yclpK2,'--',ut,ustep,'-.')
xlabel('time: seconds')
title('Closed-loop Motor system with controllers K1 & K2')
legend('Output: K1 system','Output: K2 system','Reference signal')
grid
```

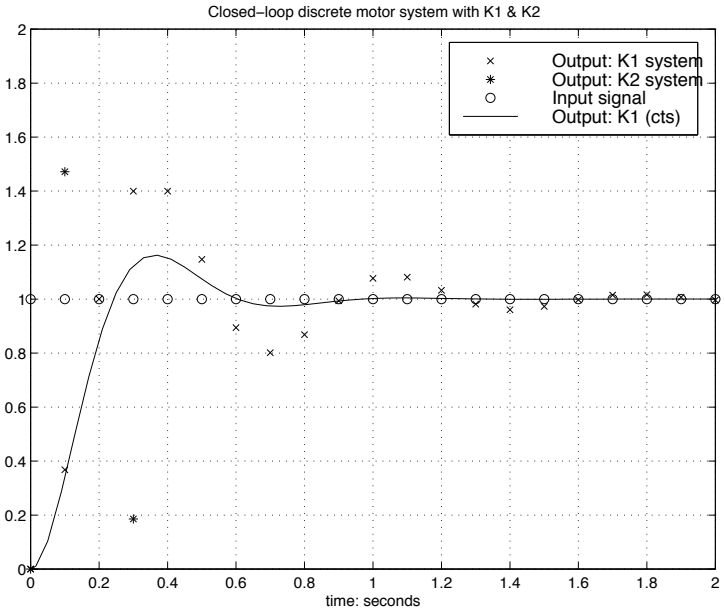



```
% K1 looks good.  If we want to improve the rise-time, as
% K2 does, we can't do it with a proportional controller.

pause   % press any key to continue.

% Now we will apply these controllers to our digital
% system.  Converting K1 and K2 to a digital representation
% is easy - there is nothing to do!  A constant gain is
% the same from a continous and a discrete point of view.
% Simulink models of the interconnection have again been
% provided.  These are very similar to the continuous time
```

```
% version;  the only difference is that the discretized
% model is used in place of the continuous one in the LTI
% system block.

Cz = tf(K1,1,Ts);
m5disc
```

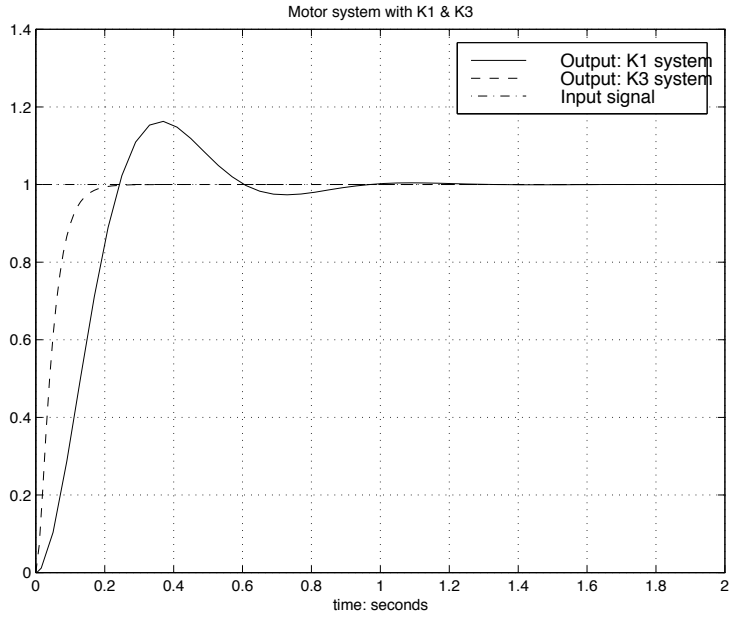



```
% Note that we use dlinmod to obtain the closed-loop system

[a,b,c,d] = dlinmod('m5disc',Ts);
clpMzK1 = ss(a,b,c,d,Ts);

Cz = tf(K2,1,T);
[a,b,c,d] = dlinmod('m5disc',Ts);
clpMzK2 = ss(a,b,c,d,Ts);
clear a b c d

% And now calculate the two time responses.  For
% variety we will use the closed-loop discrete
% systems, clpMzK1 and clpMzK2, with lsim.

uT = [0:Ts:2]';
ustepT = ones(length(uT),1);

yclpMzK1 = lsim(clpMzK1,ustepT,uT);
yclpMzK2 = lsim(clpMzK2,ustepT,uT);

plot(uT,yclpMzK1,'x',uT,yclpMzK2,'*',uT,ustepT,'o',yt1,yclpK1,'-')
axis([0,2,0,2])
xlabel('time: seconds')
title('Closed-loop discrete motor system with K1 & K2')
legend('Output: K1 system','Output: K2 system',...
          'Input signal','Output: K1 (cts)')
grid

pause   % press any key to continue
```



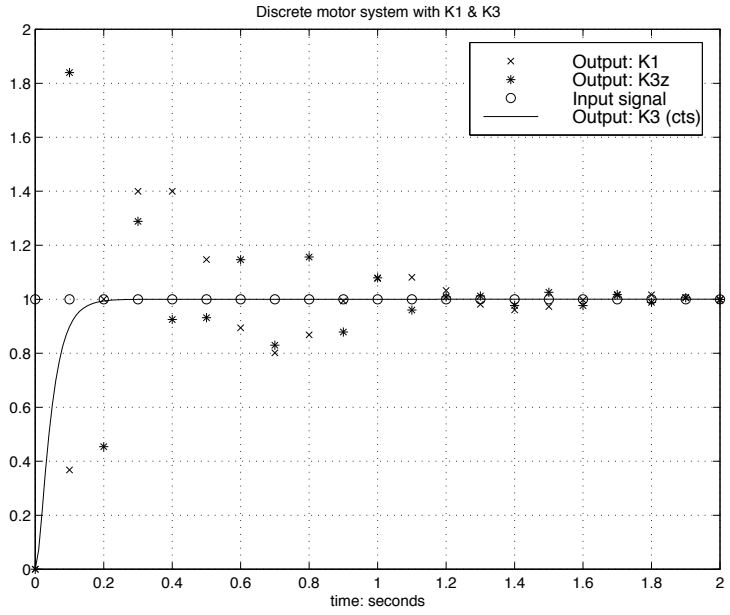

```
% What happened to yclpMzK2?  It went offscale very quickly.
% The closed loop discrete system with K2 is unstable.
% We can easily check this by looking at the magnitudes
% of the poles.

abs(pole(clpMzK1))

% These look good:  they are inside the unit circle.

abs(pole(clpMzK2))

% And these are not!

% We will have to watch out for this in discrete systems.
% We know that in the continuous case, this two pole system
% would be stable for any positive value of K.  Now in
% the discrete case, large values of K make the closed
% loop system unstable.  Why is this?

pause   % press any key to continue

% Even the controller K1 on the discrete system is not as good
% as K1 on the continuous system.  A large part of this is due
% to the fact that we only get to act on the system every Ts
% seconds.  In the continuous case, we are acting on the
% system continuously.  We might expect that we can do more,
% more easily in the continuous case.
```

```
% Now we will try to get more out of our system by designing
% an even better continuous controller.  Maybe then, when we
% discretize it, we will end up with a better discrete controller.

% Now for a better continuous design.

pause   % press any key to continue.

% We already know that a lead-lag controller will work fairly
% well on this system.  I'll put the zero of the controller
% on top of the pole of the system.  Remember that this
% will only work in the left half plane.

K3 = 10;
Cs = tf(K3*[1/10,1],[1/100,1]);

% Now form the closed loop, unity gain negative feedback
% systems, for each controller.  Simulink is again used.

[a,b,c,d] = linmod('m5cts');
clpK3 = ss(a,b,c,d);
clear a b c d

pause   % press any key to continue.

% Check the poles of the closed loop continuous system

pole(clpK3)

% Stable. Now simulate this with a step input.

[yclpK3,yt3] = lsim(clpK3,ustep,ut);

plot(yt1,yclpK1,'-',yt3,yclpK3,'--',ut,ustep,'-.')
xlabel('time: seconds')
title('Motor system with K1 & K3')
legend('Output: K1 system','Output: K3 system','Input signal')
grid

pause   % press any key to continue.
```




```
% K3 is excellent.  Now to get a discretized version.
% You will try out a number of different schemes to do
% this.  I will only look at one: bilinear transformation.
% We already know that this guarantees that the digital
% version of K3 (called K3z here) will be stable.  Does
% it guarantee that the closed loop will be stable?

Cz = c2d(Cs,Ts,'tustin');       % form digital version

% Now form the discrete-time closed loop with
% simulink

[a,b,c,d] = dlinmod('m5disc',Ts);

clpMzK3 = ss(a,b,c,d,Ts);
clear a b c d

% OK.  To be on the safe side, let's check the location
% of the closed loop poles.

pole(clpMzK3)

% And are they in the unit circle?

abs(pole(clpMzK3))

% Yes.  Good, we can at least continue.
```

```
pause   % press any key to continue.

% And now calculate the time responses.

yclpMzK3 = lsim(clpMzK3,ustepT,uT);

plot(uT,yclpMzK1,'x',uT,yclpMzK3,'*',uT,ustepT,'o',yt3,yclpK3,'-')
axis([0,2,0,2])
xlabel('time: seconds')
title('Discrete motor system with K1 & K3')
legend('Output: K1','Output: K3z','Input signal','Output: K3 (cts)')
grid
```




```
% K3z is even worse than K1!  Yet in the continuous version
% it was much better.  We really do have to be careful with
% discrete design.  We will look at some of the tradeoffs
% involved here in the problem set.

% End of Module 5

%------------------------------------------------------------
echo off
```

5.4 Problems

1. Use a quick trial and error approach to find a good proportional controller for the ZOH motor model (with $T = 0.1$) in the module. Produce a simulation and calculate the pole positions. Compare these to $K1$ analyzed in the module.

2. Plot the root locus for the discrete system and use it to explain why not all discrete proportional controllers are stabilizing, when, in the continuous case they are all stabilizing. Explain, in physical terms, why this is so. Use your root locus to predict a value of K that is marginally stable. Do a closed loop simulation with this value of K.

3. Now we will look at the effect of selecting other discretization methods. Consider the $K3$ design given in the module. Demonstrate a method which gives an unstable discrete controller.

   Examine the tustin method and find a prewarping frequency which gives a better result than the bilinear method studied in the module.

   Generate simulations and root-locus plots for the following:

   - $K3$.
   - Your best prewarped tustin version of $K3$.
   - A discrete version of $K3$ by any of the other methods that give a stable closed loop system.

   In each case keep T fixed at $T = 0.1$. Plot all of the time-domain responses on the same plot.