ENEL441

Unit 6

Digital Control

1. Introduction

In the previous units we considered the design of a compensator that would control a plant to achieve a desired closed loop response. This was done from the perspective of continuous time plants, compensators and sensor transfer functions. All of the signals flowing around the loop were of continuous time. In this section, we consider how to implement the compensator with a digital microcontroller. We will focus here on the initial step of mapping continuous time functionality into a discrete time sampled domain for a SISO control. This forms the basis of MIMO control found in complex mechatronic systems. MIMO will be developed in in ENEL541. As a start, we can have multiple sensor feedback and control multiple actuators simultaneously. This is a MIMO feedback control system.

Consider as an example the drone control that was introduced in the first lecture repeated here in Figure 1. Here we have to control four motors separately and there is a set of disparate sensor inputs from all digital GPS to analog inputs such as MEMS accelerometers. It would be impossible to implement such a control loop without basing it on a digital microcontroller. This is primarily due to the complexity of controlling four motors simultaneously. There are four motors which need to be controlled in a coordinated fashion based on a set of disparate set of sensors and additional nonlinear constraints added for ensuring stability.

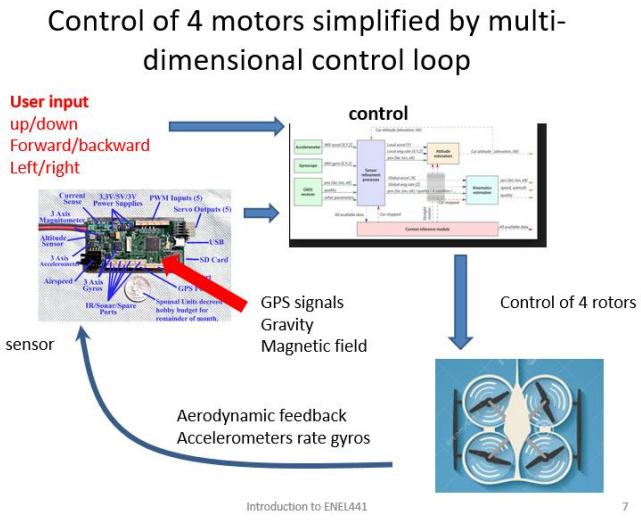


Figure 1 *Drone controller*

With digital control it is possible to implement a high level of processing complexity required with the incremental cost of a modest microcontroller. MIMO systems can be implemented with a complex set of control criteria and constraints that would be overly cumbersome to implement in analog systems. Hence in the drone example, it is important to minimize power consumption of the electric motors. Therefore, we consider the power usage for a given task into account as a penalty in an optimization of a control objective. That is the transient response time, overshoot, power consumption are all criteria that go into the design of the compensator algorithm. Generalized compensator optimization is a large part of ENEL541.

Let us start with the single input single output (SISO) feedback system that we have developed in this course. However, now instead of the compensator being an analog continuous time transfer function we will consider a digital implementation as shown in Figure 2. Here r(k) is the reference input but is given as a discrete time sampled signal as opposed to a continuous time signal. The compensator is implemented by the microcontroller with a discrete time sampled output c(k). This digital signal is converted into a continuous time analog signal and is fed to the actuator resulting in u(t), the input to the plant. The plant responds to u(t) resulting in the actual output y(t). This plant output is sensed and sampled by the ADC resulting in a discrete time sample of the feedback signal given as $y_s(k)$.

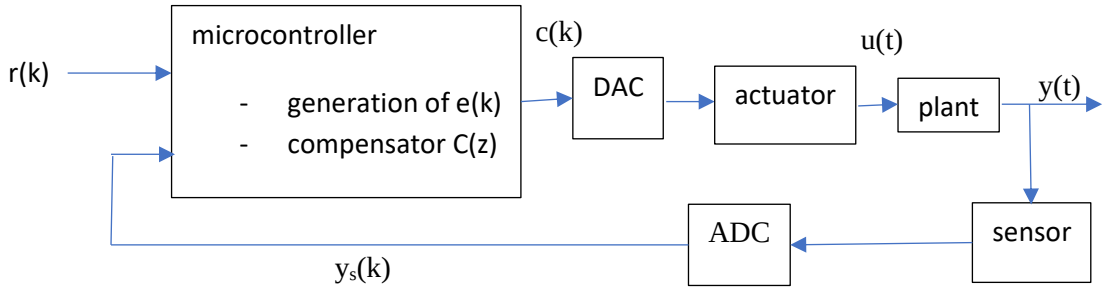


Figure 2 *Basic SISO feedback control loop with a digital implementation of the compensator*

In Lab 4, Simulink will be used to model the digital control with ADC and DAC components that are non-ideal in that they have quantization error and saturate with larger magnitude inputs. In this lecture unit more of the background will be developed that complements the lab content. The focus will be the implementation of a conventional control loop based on continuous time analysis and design that is mapped into a digital processor implementation. In other words, the digital compensator implementation will be based on the analog continuous time prototype of say a PID controller. This is then mapped into a discrete time controller that is implemented on a microcontroller. Then additional 'digital' control can be added such as changing control configurations for different operation conditions. This is generally a sufficient approach especially when the plant can be modelled as a continuous time system and that the sensor feedback is analog.

2. Sampling of a continuous signal

Consider a simulation of a continuous time sine-wave generator that is sampled and quantized with an ADC. The samples are then converted back to analog form based on a DAC. This is modelled approximately by the Simulink model of Figure 3.

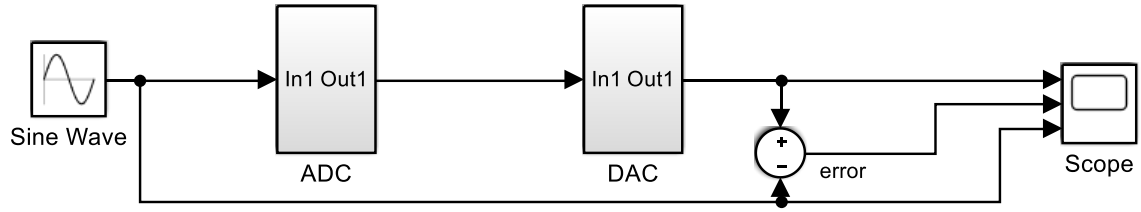


Figure 3 *Simulation of a continuous time signal that is sampled and quantized by an ADC and then converted back to digital form with a DAC*

The model of the ADC is shown in Figure 4. Introduce T_s as the sampling interval which in this case is set to be T_s=0.1. The zero-order block represents the sample and hold function of the ADC in which the signal is sampled at an instance in time (intervals of T_s) and then the sample is held while the quantizer can convert it into digital form. A true model of an ADC would map the output of the quantizer into a digital number representation. However, for simplicity of simulation we will bypass this operation here.

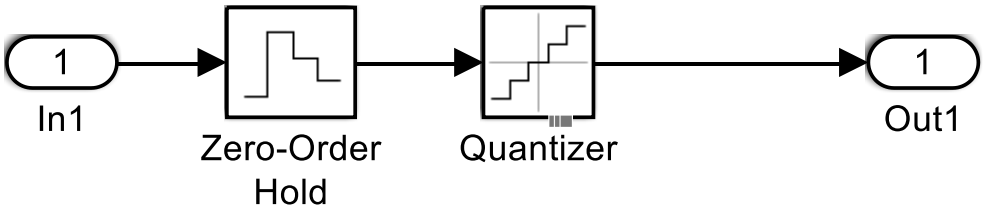


Figure 4 *Model of ADC based on a zero order hold sampled with a sampling interval of T_s and quantizer*

The ZOH is found in the control block-set and the quantizer in the DSP block-set in the Simulink library. The model of the DAC is shown in Figure 5 is simply a zero-order hold. This represents the function of setting a voltage based on the digitized sample read into the DAC at intervals of T_s. As we are bypassing the mapping of voltage sample to number representation in the ADC and similarly bypassing the mapping of a number representation to voltage sample in the DAC we simply use an additional zero order hold.

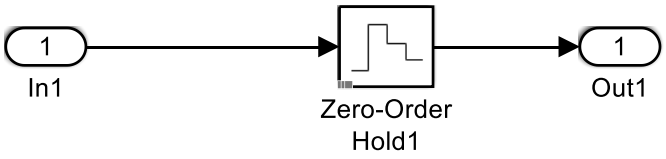


Figure 5 *Model of DAC based on a zero order hold sampled with a sampling interval of Ts*

Running the simulation of the ADC and DAC circuit of Figure 3 with a T_s of 0.1 and a quantizer interval set to 0.05 we obtain the signal waveforms as shown in Figure 6. The error of digitizing the signal is the difference between the continuous time analog signal and the staircase sampled signal. The contributions to this error are the finite sampling interval and the finite quantization interval. Hence, the signal digitization error can be made negligibly small if these intervals are sufficiently small. From a circuit implementation perspective, the limitation is the precision and sampling rate of the ADC as well as the DSP (digital signal processing) following the ADC.

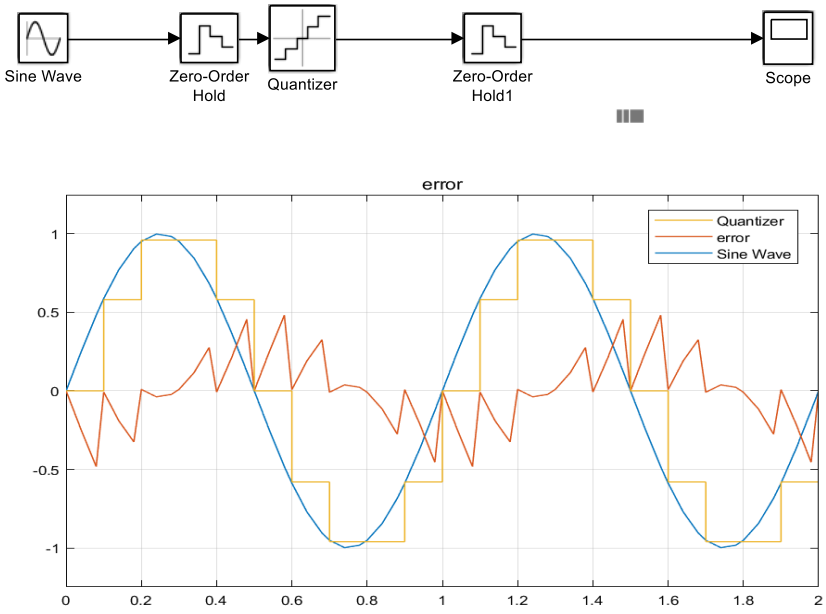


Figure 6 *Simulation of ADC/DAC that samples the continuous time signal in blue resulting in the digital signal given by the orange staircase signal. Each level of the quantized signal is stored as a binary number. For this example a sinusoidal input of 1 Hz and peak amplitude of 1 volt with a sampling interval of 0.1 seconds was used. The quantizer dialog box and set the quantization interval to 0.05. Note that for the Simulink simulation it is not necessary to use two ZOH blocks in series. The second one can be removed.*

The distortion of the discrete sampling and quantization is easy to simulate but can only be accounted for approximately by calculation. For the discrete time sampling we can determine the statistics of the aliasing error. This happens when we sample at lower than the Nyquist rate. Recall that in order to have distortion free sampling and reconstruction that the sampling rate has to be twice that of the highest frequency component. This is fine for dealing with communications like signals that are bandlimited and where we are generally considering statistical averages of the aliasing distortion noise. However, in a control system we are more interested in the transient response of the error signal for a given input change such as a step input or impulse input. Note that the frequency content of a transient response say of a step input that goes through a transfer function of H(s) has frequency content that goes to zero slowly with frequency and therefore it is not practical to consider a 'Nyquist frequency'. Hence we will always have some aliasing noise to deal with. As an example, consider the step response of an RC low pass filter with

$$H(s)=\frac{1}{1+sRC}$$

Let RC=1. Then the frequency response of the step transient is given as

$$H_{step}(s)=\frac{1}{s}\frac{1}{1+s}$$

If we plot the frequency response of $H_{step}(s)$ as shown in Figure 7 it is evident that there is no 'highest frequency' of the signal and hence no sampling rate above which the aliasing noise goes to zero.

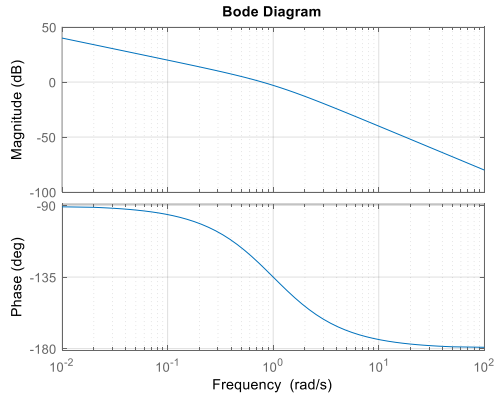


Figure 7 *Frequency response of transient of the RC filter. Note that the magnitude gets small with frequency but is never zero and therefore there is not a Nyquist frequency. Regardless of the sampling rate there will always be aliasing noise*

Typically, we would sample at about 10x the 3 dB bandwidth of the spectrum. Then the aliasing noise is manageable and can be treated as a noise disturbance input to the control loop. However, in this case what is the 3 dB bandwidth s we have a pole at zero. So a rule of thumb can be to sample at 10x the bandwidth of the plant which in this case is $10/2\pi$. Does this offer sufficient accuracy? Well this

depends on the control loop and this is where simulation becomes important. For mechatronics control loop applications the loop bandwidth is typically smaller and the response rates of motors and sensors are on the order of milliseconds to tens of seconds. In other words, much slower than a typical microcontroller and ADC/DAC operating with clocks in the >1 MHz range. Hence the effects of the discrete time sampling become negligible. However, sensors such as rotation encoders, limit switches, ultrasonic sensors, MEMS inertial sensors are typically relatively slow having equivalent sampling rates in the range of 10's to 100's of samples per second. These become the issues in digital feedback control systems. Generally, the slower the sampling rate the more delay gets imputed into the loop and there are then issues with instability.

The other distortion is the quantization. Accounting for quantization noise is also very difficult and we usually just resort to an approximation of saying that it is a white Gaussian noise injected into the loop. However, this is really only accurate if the aliasing and quantization noise is relatively small compared to the signals propagating through the loop.

In practice we take the following approach. We start with a fast and high precision ADC and DAC in our control loop. Then we determine the loop performance. If the cost of the ADC, DAC and DSP is negligible (as it often is) then we leave the processing as it is. However, in some cases, the digital processing has to operate at very low power consumption or we cannot afford a precision ADC in the loop. In this case we will be forced to reduce the precision and sampling rate. Usually we resort then to simulation to determine the effects the sampling and finite precision on the control dynamics of our loop.

As an example of a transient response consider the simulation comparison of Figure 8 and Figure 9. This shows a Simulink simulation of a digitized ramp signal passed through a plant that is subsequently digitized again. As observed, the main distortion that would concern us is the lag of the digitized version.

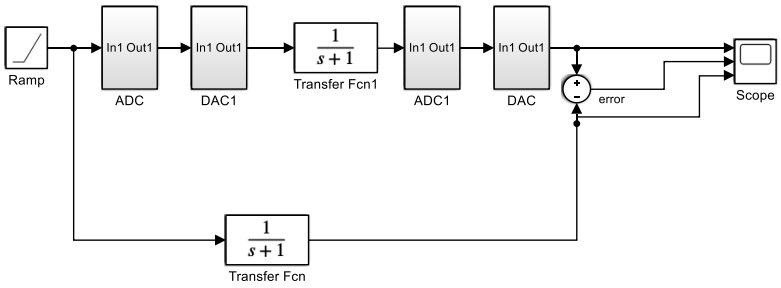


Figure 8 *Comparison of a DSP processing of a plant and the continuous time*

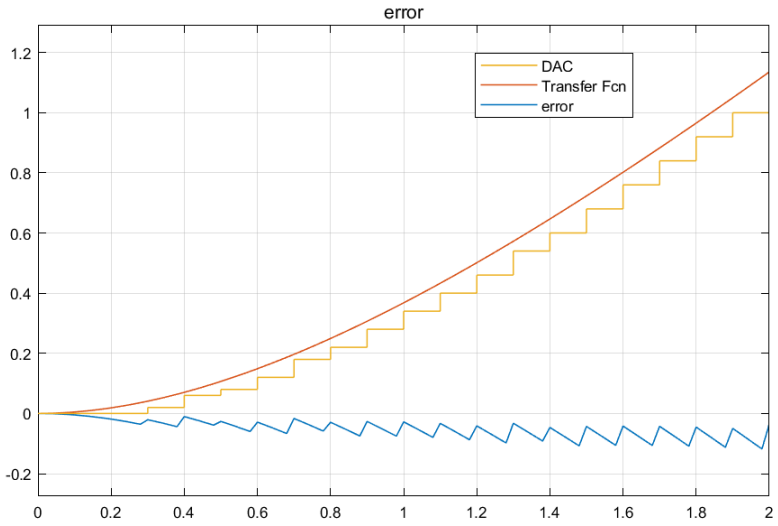


Figure 9 *Simulated results with a sampling interval of Ts=.1 and quantization levels at 0.02.*

From this example there is a lag generated that will certainly affect the dynamics of the closed loop response. As discussed in the previous units, the extra delay in the loop contributes to the phase shift in the loop which degrades the stability of the overall control loop. This is a main consideration in the mapping of a digital control loop from the continuous time prototype.

3. Analysis of a mixed signal loop

Consider Figure 2 of a generic digital control loop that contains both a digital version of a controller and a continuous time plant. In between we have conversion from the digital controller output to the continuous time actuator signal produced by a DAC. Likewise we have a sensor output that produces a continuous time signal that is digitized by an ADC and fed into the compensator.

Suppose we want to have the compensator be an integral feedback with some gain and that the plant is represented by a simple transfer function of

$$H(s) = \frac{1}{1+s}$$

The integrator compensator with a gain of 2 is given as

$$C(s) = \frac{2}{s}$$

The continuous time version of the control loop is given in Figure 10 with the step response in Figure 11.

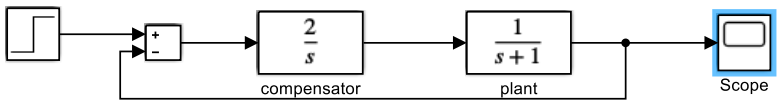


Figure 10 *Continuous time control loop*

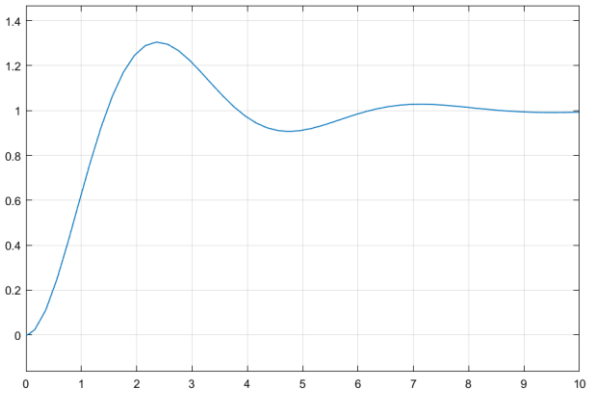


Figure 11 *Step response of control loop*

Next we consider a digital version of the compensator. This is determined by the program **c2d()** which converts a continuous time transfer function to a discrete time. Assume that we have a sampling interval of

Ts=0.1

for this example. This is what Matlab gives us:

```
Ts =

    0.1000

>> Hcd = c2d(tf(2,[1 0]),Ts)

Hcd =

   0.2
  -----
  z - 1

Sample time: 0.1 seconds
Discrete-time transfer function
```

Based on the discrete time transfer function given as the Z transform of Hcd(z), we can map this into a difference equation which is of the form of a recursive accumulator as

$$u(k+1) = u(k) + 0.2e(k)$$

where u(k) is the discrete output sequence of the compensator and e(k) is the input error sequence.

$$H_{cd}(z) = \frac{0.2}{z-1} = \frac{0.2z^{-1}}{1-z^{-1}}$$

Note that normally we would specify

Hcd = c2d(tf(2,[1 0]),Ts,'zoh')

with the option of 'zoh' but this is apparently the default as used by c2d. Simulating the mixed signal control loop in Simulink is straight forward with the mixed implementation as shown in Figure 12.

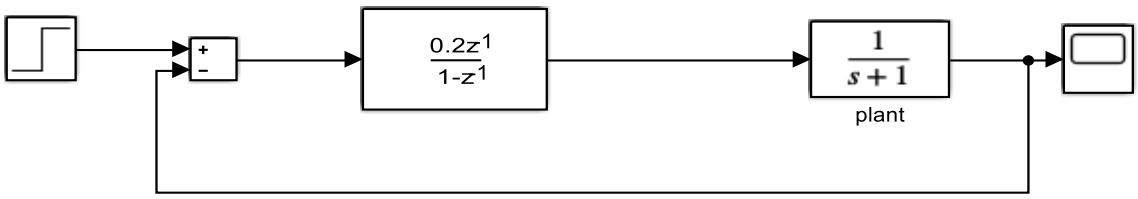
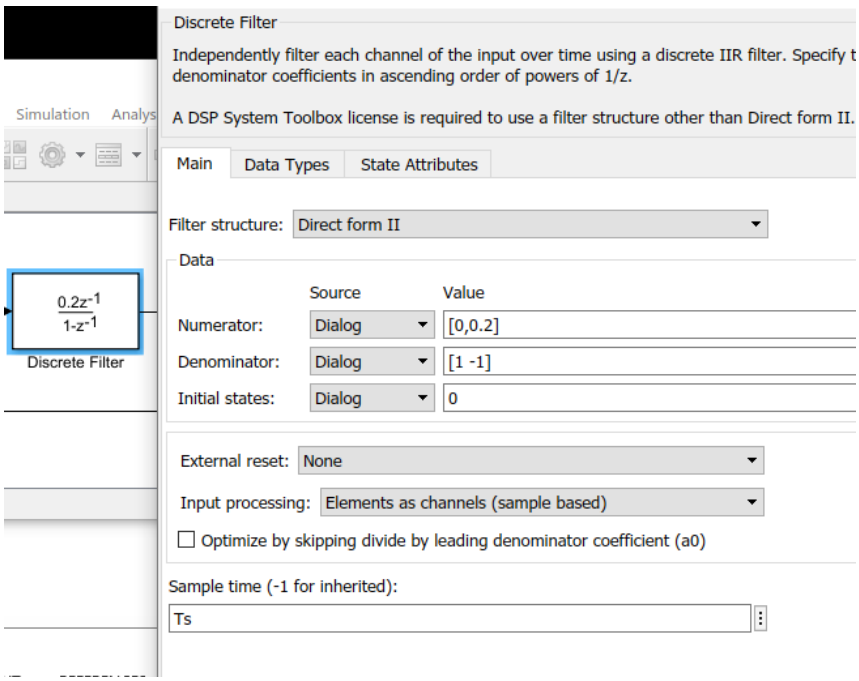


Figure 12 *Simulation of mixed signal control loop with a digital compensator*

Note that in the simulation of Figure 12 we have ignored including the ADC and DAC models. This implies that the DAC and ADC sample at the rate of 1/Ts and that the quantization distortion is ignored.

Also, you have to tell Simulink the sampling interval. To do this open the dialog box for the discrete filter and enter Ts for the sampling time.



Be aware that Matlab keeps changing the appearance of the dialog boxes in Simulink for different versions and hence what you will see in Matlab will be different than what is shown here.

Now let us consider a comparison of the two responses with the digital and continuous time implementations of the compensator.

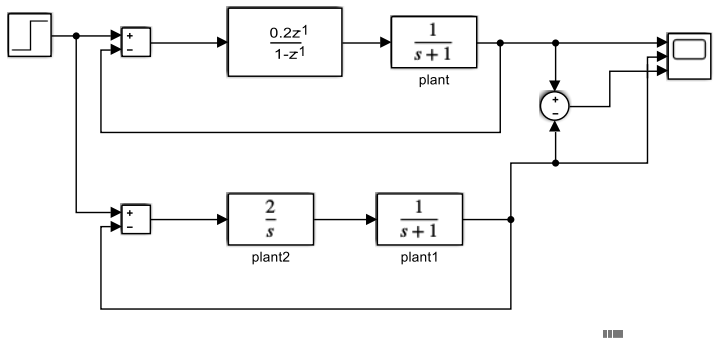


Figure 13 *Simulink comparison of digital discrete time and continuous time implementations of the control loop*

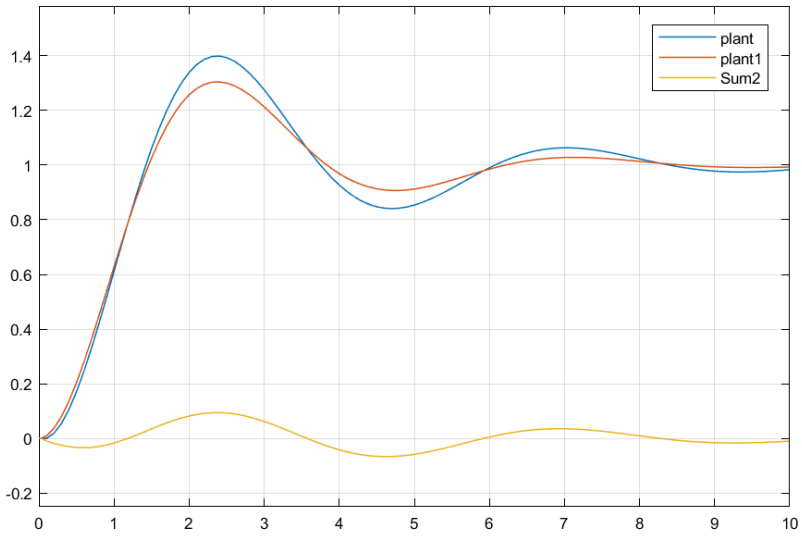
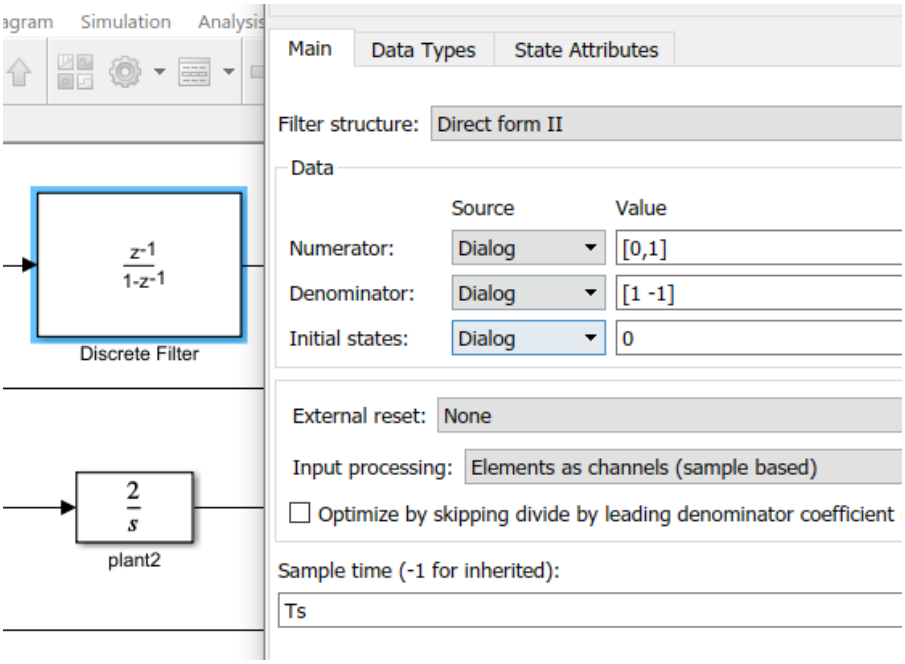


Figure 14 *Step response comparison of discrete time and continuous time, blue is continuous time simulation, red is with the discrete time filter and yellow is the difference between the two simulations*

In the comparison of Figure 14 note that the digital version has slightly higher overshoot than the continuous time version. The delay is about the same. Note though that if we increase the sampling interval then the loop eventually becomes unstable. First we need to determine the equivalent digital filter for the compensator as

```
>> Ts = .5

Ts =

    0.5000

>> Hcd = c2d(tf(2,[1 0]),Ts,'zoh')

Hcd =

    1
  -----
  z - 1

Sample time: 0.5 seconds
Discrete-time transfer function.
```

Then we input the digital compensator into Simulink by changing the dialog box for the filter as



Insure that Ts is set to 0.5 before running the simulation. This results in the output as shown in Figure 15. Note that the error grows with time and is therefore unstable.

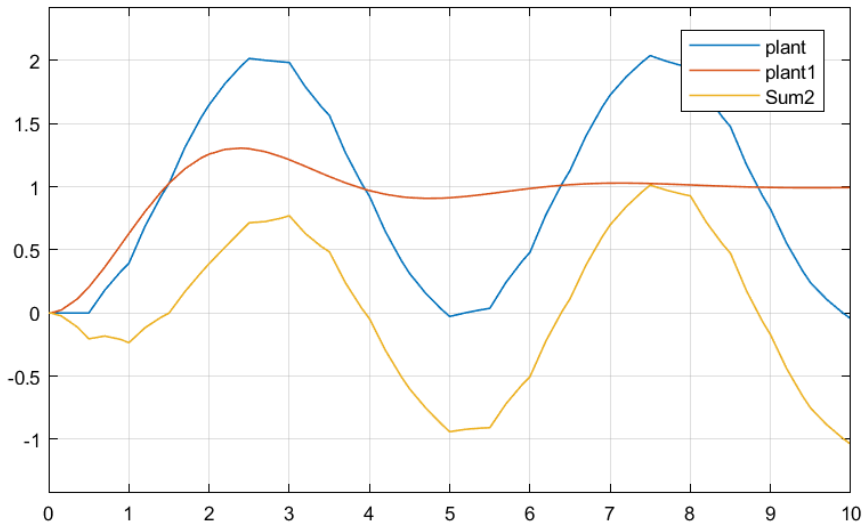


Figure 15 *Step response with Ts=0.5*

This brings up an issue with implementing a discrete time sampled compensator to replace the continuous time compensator. The finite T_s introduces an additional delay into the loop and this results in a phase shift that increases with frequency. This can result in instability if the phase margin of the continuous time loop is too small. We can analyze this instability based on using root locus plots in the z-plane. Instead of having the right hand plane of the s-plane as the region of instability, in the discrete time implementation the region outside the unit circle in the Z-plane is the region of instability.

Alternately, we can approximate the delay of T_s as an additional transfer function and then include this in the continuous time root locus analysis. We will consider this shortly.

What assumptions is Simulink making in order to facilitate the mixed signal simulation? When the mapping from continuous to discrete time of the compensator transfer function is done we specify 'zoh' which implies an assumption that the equivalent ADC on the input samples the signal at an interval of Ts and holds the value of the signal for Ts. This is the equivalent pulse input into the continuous time transfer function that is sampled forming a sequence that is subsequently converted back into the Z domain as a Z domain transfer function. There is a lot that goes on in c2d() and we cannot do it justice with the brief introduction given here.

Next consider the output of the discrete time filter that is passed onto the continuous time plant. Simulink assumes that there is a ZOH block in between these functions that is sampled at the same rate as the compensator.

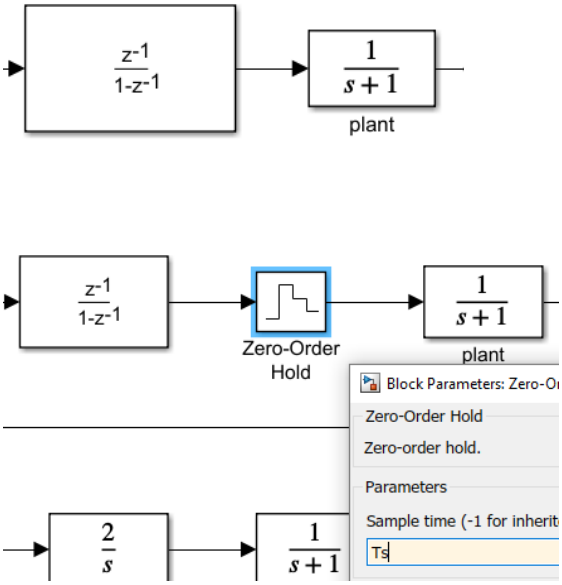
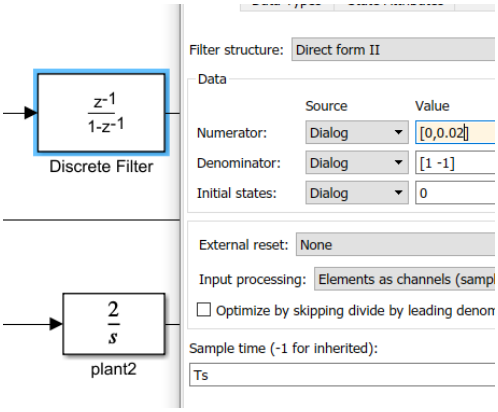


Figure 16 *Simulink implies a ZOH block (top) which can be explicitly placed as in the bottom scheme*

Suppose we choose Ts=0.01 which is a much higher sampling rate. Then the discrete time and continuous time implementations should result in the same step response. To test this we first change the Z transform of the compensator transfer function as follows:

```
Ts =

    0.0100

>> Hcd = c2d(tf(2,[1 0]),Ts,'zoh')

Hcd =

  0.02
  -----
  z - 1

Sample time: 0.01 seconds
Discrete-time transfer function.
```

Implement the change by opening the compensator dialog box and entering the new numerator and denominator.



Now the responses are virtually identical.

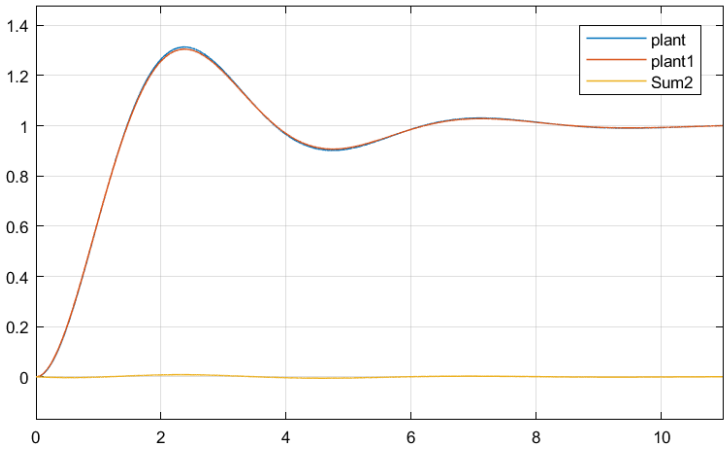


Figure 17 *Simulink output for Ts=0.01*

Note that it is often convenient to have a general LTI system, a block that is found in the control system toolbox of the Simulink library. We insert it as shown in Figure 18.

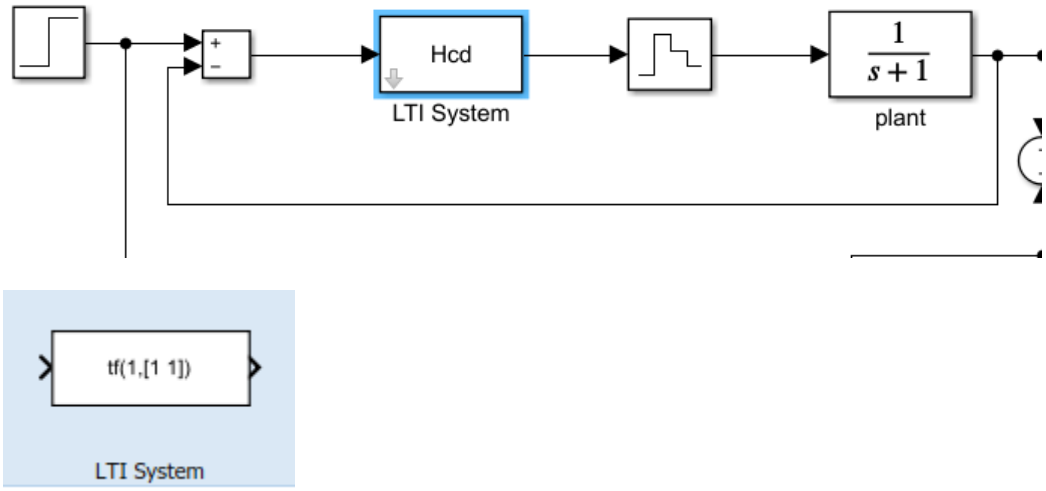
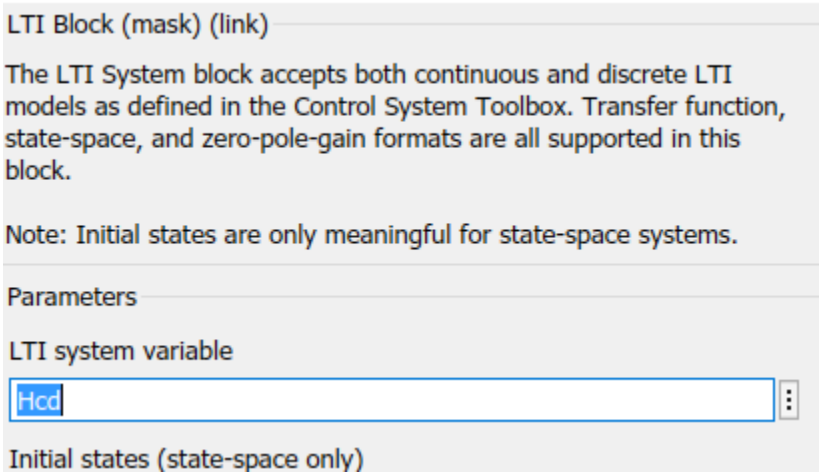


Figure 18 *Simulink model with LTI system used for the discrete time model of the compensator*

This block allows you to specify the transfer function in the workspace with a name that is copied into Simulink. Hence we only have to input the variable name of the transfer function system in the dialog box.



4. Stability analysis

It would be nice to understand the motion of the poles and see why with larger T_s the system starts to become unstable. How to analyze this? The best approach is to convert the plant to a discrete time form and then combine this with the compensator. Then use rlocus() which creates the root locus plot

for the open loop transfer function. However, this is the transfer function of a Z transform and so the instability is based on crossing the unit circle instead of the jw axis. Otherwise it is the same as before. Figure 19 shows the root locus plots. The top plot is the continuous time for the compensator and the plant and as expected, the gain can be increased arbitrarily. The only consequence is that the damping of the closed loop pole is decreased with increasing loop gain. In the discrete version, the Z transform of both the compensator and the plant is calculated with c2d() and then combined. But now the root locus is with respect to the unit circle grid which shows that the closed loop pole crosses the circle for a gain of about 3.6.

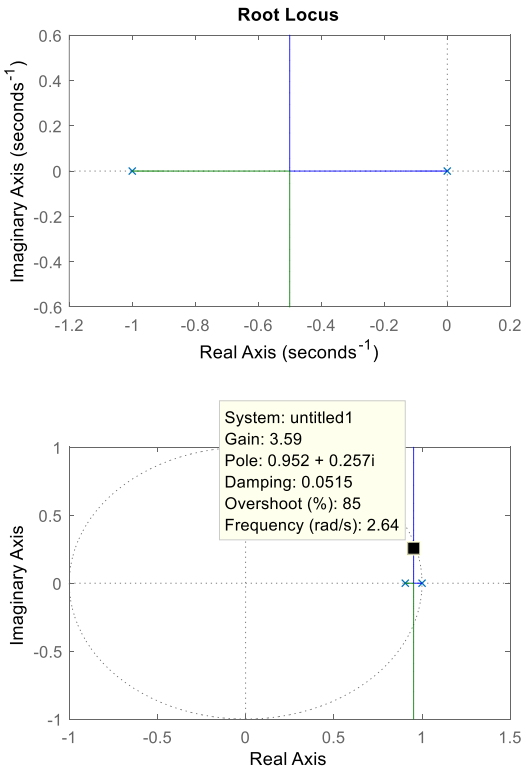


Figure 19 *Root locus analysis of the compensator and plant*

```
Ts = 0.1;
Hc = tf(2,[1 0]);
Hcd = c2d(Hc,Ts,'zoh');
Hp = tf(1,[1,1]);
Hpd = c2d(Hp,Ts,'zoh');

figure(1); subplot(211);rlocus(Hc*Hp)
subplot(212);rlocus(Hcd*Hpd)
```

Consider now what happens if Ts is increased from 0.1 to 0.5. The root locus is plotted in Figure 20 which shows the same behavior as before except that the gain at the point of crossing the unit circle and

hence becoming unstable is much lower. In this case the gain can only be 0.9 implying that relative to the previous case, the closed loop bandwidth will be less.

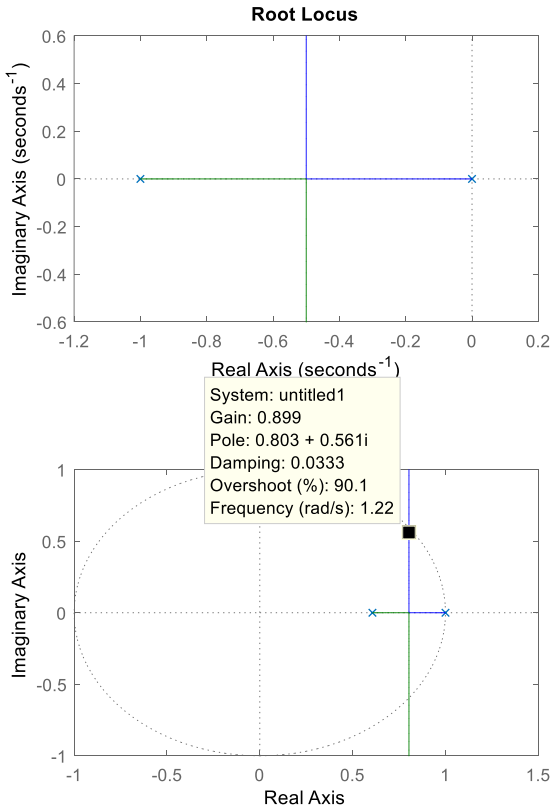


Figure 20 *Root locus analysis of the compensator and plant for T_s=0.5*

5. Programming steps for compensator

Consider that we have determined the appropriate compensator to use in the continuous time domain such that we have $H_c(s)$. Then we use the routine c2d() with the option of 'zoh' to convert this into a discrete time sampled version of the compensator which results in $H_{cd}(z)$. Then we take $H_{cd}(z)$ and convert this into a difference equation. This difference equation gives us a recursive equation which we can use in the real time program for implementing the compensator. In the current case we have

$$H_{cd}(z) = \frac{0.2}{z-1}$$

which maps into the difference equation of

$$u(k+1) = u(k) + 0.2e(k)$$

which is easily turned into a real time processor update that is executed at each sampling time interval step.

Design Example

Consider an inverted pendulum on a fixed pivot point. Assume that the torque is the output of the actuator driver such that u(t) is a torque. The plant has a transfer function of

$$H_p(s) = \frac{1}{s^2 - 1}$$

The angle of the pendulum is y(t) and can be observed directly. The goal is to design a digital compensator that will stabilize the loop and then map this into a digital processor. Determine the update equation for this.

There is no requirement for an input, just that the loop is stabilized. But we do need the loop to be driven towards zero error for an applied impulse function input. Hence $e(\infty) = 0$ for the impulse input. Assuming we can place all of the poles in the LHP then the FVT states that we do not require an integrator. We just can't have a closed loop pole at s=0.

Start with sisotool

```
>> Hp = tf(1,[1,0,-1])

Hp =

     1
  -------
  s^2 - 1
```

We add a lead compensator circuit which results in the following.

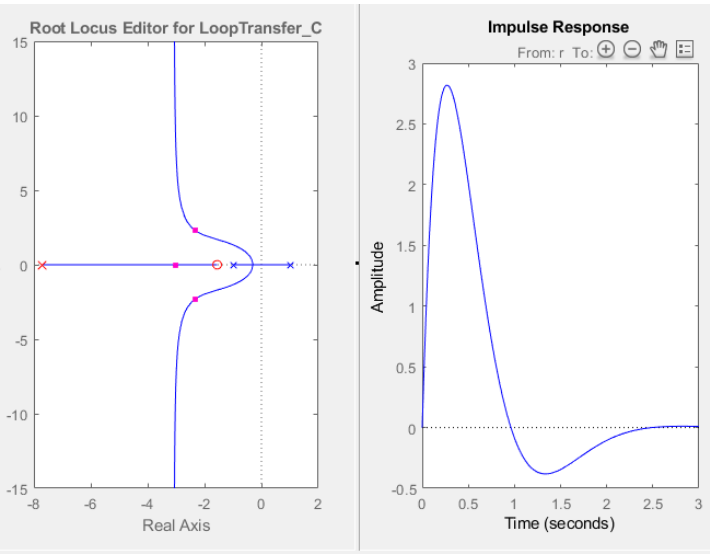


Figure 21 *Root locus for the inverted pendulum plant*

The lead compensator is given as

```
C =

  26.005 (s+1.558)
  ----------------
     (s+7.708)
```

Next plug this into the Simulink model as shown in Figure 22. Note that the pulse generator is uses to give a repetitive impulse disturbance.

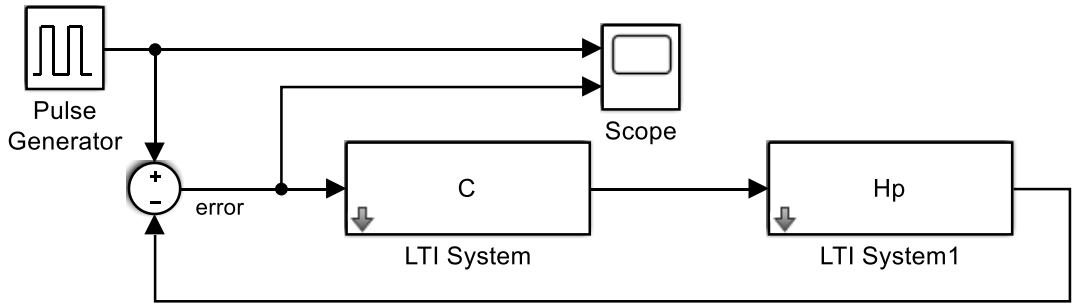


Figure 22 *Simulink model of the inverted pendulum with the compensation C*

The Simulink output is shown in Figure 23.

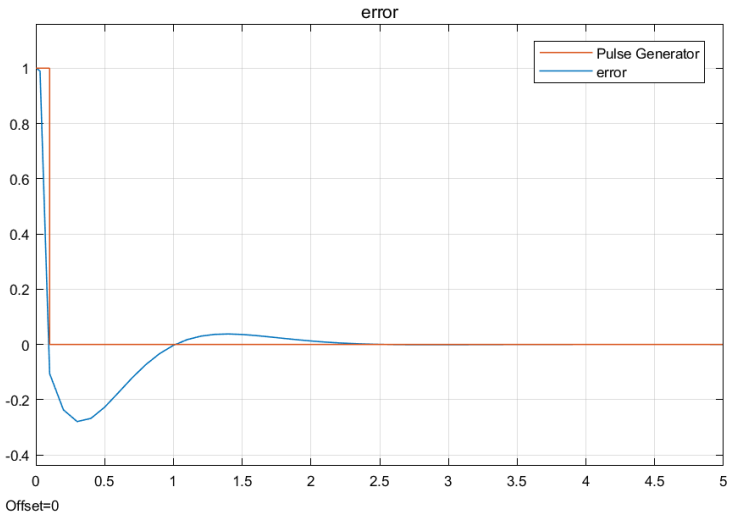


Figure 23 *Simulation of the inverted pendulum with stable compensator and error due to impulsive input is driven towards zero.*

Next consider the modification for the digital compensator. We will assume a sampling time of Ts=0.02 and use c2d to create the discrete time transfer function.

Now modify and get digital discrete time version.

```
>> Ts = .02;
>> Cd = c2d(C,Ts,'zoh')

Cd =

  26.005 (z-0.9711)
  -----------------
      (z-0.8571)

Sample time: 0.02 seconds
Discrete-time zero/pole/gain model.
```

Run this in the comparative Simulink model as shown in Figure 24 which also has the output scope plot for this simulation. As observed the continuous time and discrete time are virtually the same.

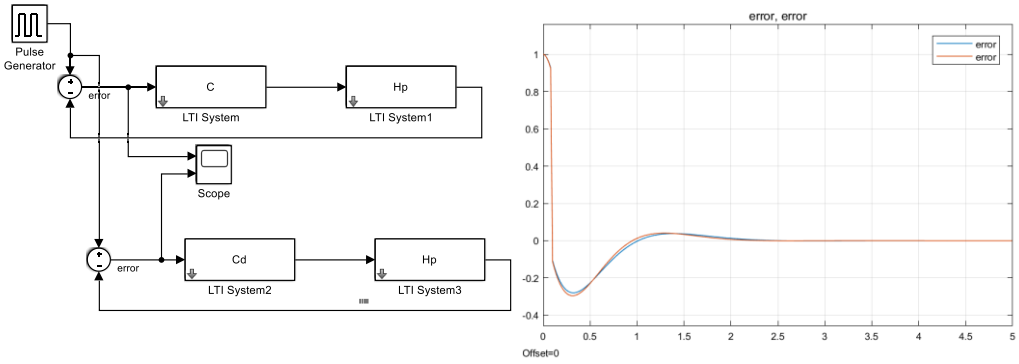


Figure 24 *Comparative Simulink and scope plot*

But now in the discrete time model we are assuming the input value for the same instance that we are calculating the update for the compensator output. This results in a implicit update relation that is not convenient to implement. So for a more realistic update we will add an additional delay of one time step. Now the overall digital recursive equation is

u(k) = 0.8571 u(k-1)+ 26.005(e(k-1)-.9711 e(k-2))

Let us put this into the Simulink as shown in Figure 25.

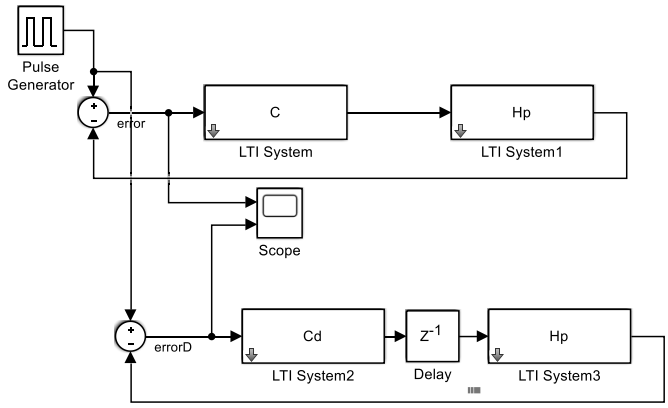


Figure 25 *Comparative Simulink with delay for realizability in the compensator*

The comparative scope plot is shown in Figure 26.

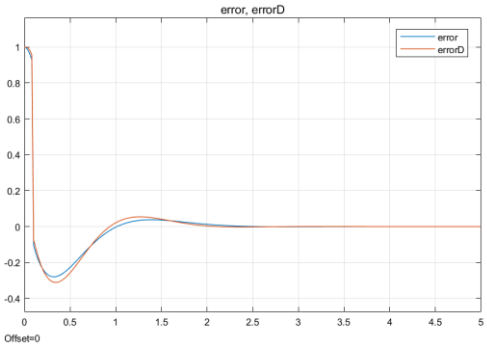


Figure 26 *Comparative Simulink with delay in the discrete time compensator*

How to derive the discrete time equation from the continuous time

The mapping from the continuous time transfer function to the discrete time transfer function is not exact as would be expected. Continuous time is equivalent to discrete time sampling only if the sampling rate is infinite which it is not. The clock in the microcontroller can generally operate at a rate that is much higher than the time constants associated with the continuous time plant. However, sensors generally cannot sample fast enough such that they behave as being continuous in time.

As the mapping between continuous and discrete time is not exact we have to decide how best to implement the approximations. One idea is to make the impulse response of the continuous and discrete time the same. This is the impulse invariance method that the Matlab routine of c2d() uses. Another method is to use a numerical approximation for the time domain derivatives in the differential equation describing the continuous time transfer function. This will be used here as it is direct.

As an example suppose that we have a continuous time transfer function given as

$$H(s) = \frac{1}{s+1}$$

Then the differential equation relating e(t) as the input and v(t) as the output is

$$\frac{dv}{dt} + v = e$$

Now we have to come up with an approximation to the derivative which can be based on the forward Euler as

$$\frac{dv}{dt} \approx \frac{v_{k+1} - v_k}{\Delta}$$

where Δ is the time sampling interval and the current time is $t = (k+1)\Delta$. Then we can set up the difference equation as

$$\frac{v_{k+1} - v_k}{\Delta} = -v_k + e_k$$

From here we can get the recursion equation as

$$v_{k+1} = v_k(1 - \Delta) + \Delta e_k$$

This gives us a z plane pole of

$$1 - \Delta$$

which is stable as it is inside the unit circle. Furthermore we can state the equivalent Z transform transfer function as

$$H(z) = \frac{\Delta}{z - (1 - \Delta)}$$

Consider another example for continuous time transfer function given as

$$H(s) = \frac{s + 2}{s(s + 1)}$$

Then the differential equation relating e(t) as the input and v(t) as the output is

$$\frac{d^2v}{dt^2} + \frac{dv}{dt} = \frac{de}{dt} + 2e$$

Now the second order derivative can be approximated as a central difference given as

$$\frac{d^2v}{dt^2} \approx \frac{v_{k+2} - 2v_{k+1} + v_k}{\Delta^2}$$

The first order derivatives are

$$\frac{de}{dt} \approx \frac{e_{k+1} - e_k}{\Delta}$$

$$\frac{dv}{dt} \approx \frac{v_{k+1} - v_k}{\Delta}$$

where Δ is the time sampling interval and the current time is $t = (k + 1)\Delta$. Then we can set up the difference equation as

$$\frac{v_{k+2} - 2v_{k+1} + v_k}{\Delta^2} + \frac{v_{k+1} - v_k}{\Delta} = \frac{e_{k+1} - e_k}{\Delta} + 2e_k$$

From here we can extract a recursive difference equation and then write the transfer function in the Z domain as in the previous example. H(z) can be used in a Simulink simulation with the LTI block. Also it

can be used in the sisotool. Note that combining a discrete time and a continuous time in sisotool forces the c2d() of the continuous time components such that the analysis is done in the discrete time domain.

As a final example consider the problem of a continuous time transfer function given as

$$H(s) = \frac{e^{-2\Delta s}}{s+1}$$

where there is a delay of 2Δ associated with the transfer function. Easiest to split into two parts. The first is the DEQ as before with $\frac{dv}{dt} + v = e$. This results in the recursion equation of $v_{k+1} = v_k(1-\Delta) + \Delta e_k$ as before. Now add in the delay of 2Δ connected with the input such that we modify the difference equation as

$$v_{k+1} = v_k(1-\Delta) + \Delta e_{k-2}$$

where we have delayed the input by two time intervals.