ACTIVE OPTICAL SYSTEMS

AN022

Error Rejection with an Integrator Control System

Author: **Justin Mansell**
Revision: **11/13/10**

This application note describes how to model the error rejection ratio from an integrator control system with latency. This application note applies to any control system using this form of control including adaptive optics systems and optical trackers.

Control Methodology

In this application note, we are analyzing the performance of a simple integrator control system with a discrete-time modeling approach. The Matlab code for this control model is in the appendix. In our model, we are including the effect of processing latency between signal sampling and command execution. In this model, we are not including the actuator dynamics, but it could be easily added. The model begins by establishing a logarithmic spaced set of frequencies for analysis. At each frequency, a discrete-time performance of the system is evaluated over three periods at the sample resolution that is equal to 10 times the signal sample (measurement) rate. This will enable us to see the effect of the control system at a higher resolution than the sample rate.

This model works on an event-based methodology. The first sample is taken at t=0 s. After the sample is taken, the new command is calculated and added to a first-in-first-out (FIFO) command queue. This new command is scheduled to be applied at the current time plus the latency time, which is given in the header as a multiple of the sample (aka measurement) period. Commands are calculated as the prior command times a leak factor, which is typically slightly below 1.0, minus the error times a gain factor, which is typically a negative value between 0 and -1.0. When the new command is applied, it is removed from the command queue to setup the next command. In our model, the root-mean-square (RMS) difference between the signal and the command is calculated to determine the efficacy of the control system. The reduction ratio is the ratio of the rms amplitude of the error (signal minus command) to the rms of the signal.

Figure 1 shows the error rejection ratio function determined by executing the code in the appendix. The control model was for a 2500 Hz measurement rate, a gain of -60%, a 0.99 leak factor, and a 1.5x latency, which corresponds to 600 µs. The low-frequency response fairly closely follows the expected 20 dB per decade slope. The unity gain cross-over is at about 179 Hz. The high gain causes a significant 7.5 dB overshoot, which peaks around 375 Hz. This peak is at a frequency

approximately 6.7 times less than the sample frequency. The overshoot crosses the unity gain point again at about 800 Hz, which is about 3 times less than the sample frequency.

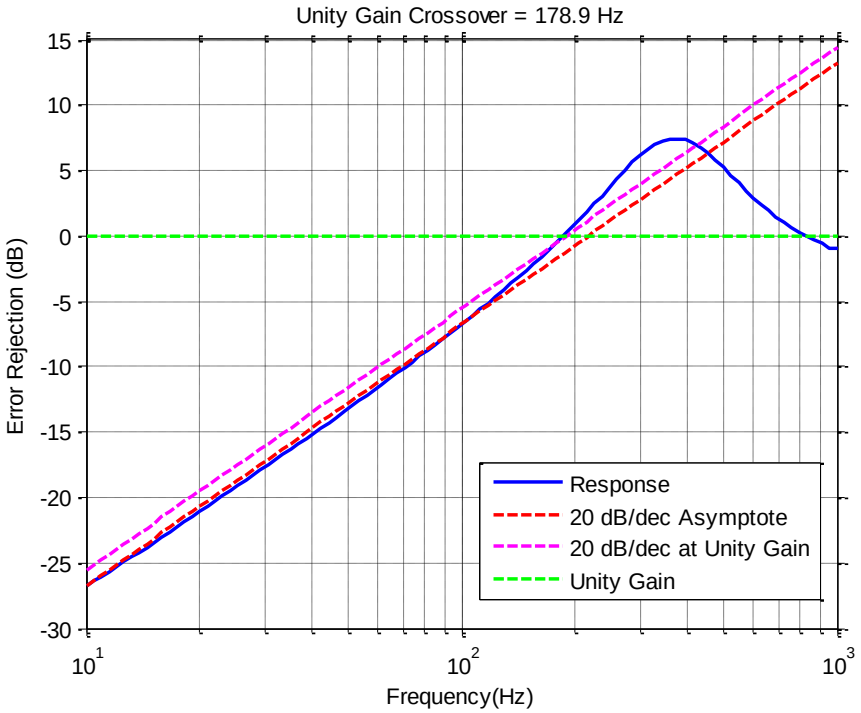


Figure 1 - Error Rejection Ratio determined by the Matlab time-domain control model.

There are two different ways of calculating the new command in the script below. The current method is :

```
newcmd(end+1) = leak*newcmd(end) - gain*err;
```

An alternative method is:

```
newcmd(end+1) = leak*cmd - gain*err;
```

In doing some subsequent experimentation, we have generally found that second method causes significantly less overshoot, but a reduced unity-gain cross-over as well. The result of this change to the system is shown below in Figure 2.

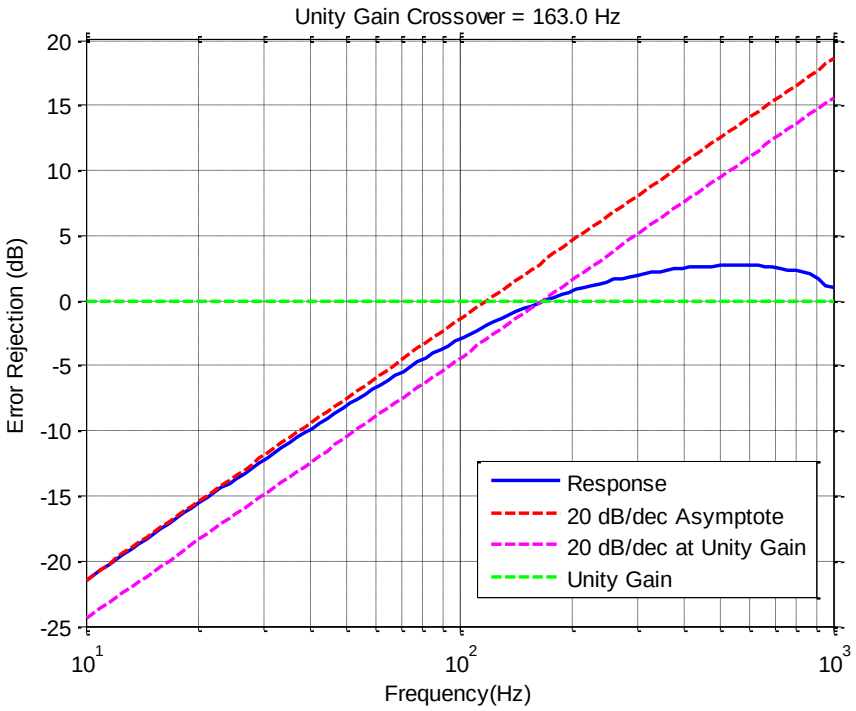


Figure 2 - Error Rejection with command based on current command.

Effect of Latency on the Error Rejection Ratio

Using this script as a base, we studied the effect of latency on the error rejection ratio. We used the first command control methodology discussed above which was based on the previously sent command instead of the currently executing command. Figure 3 shows the error rejection for different amount of latency.

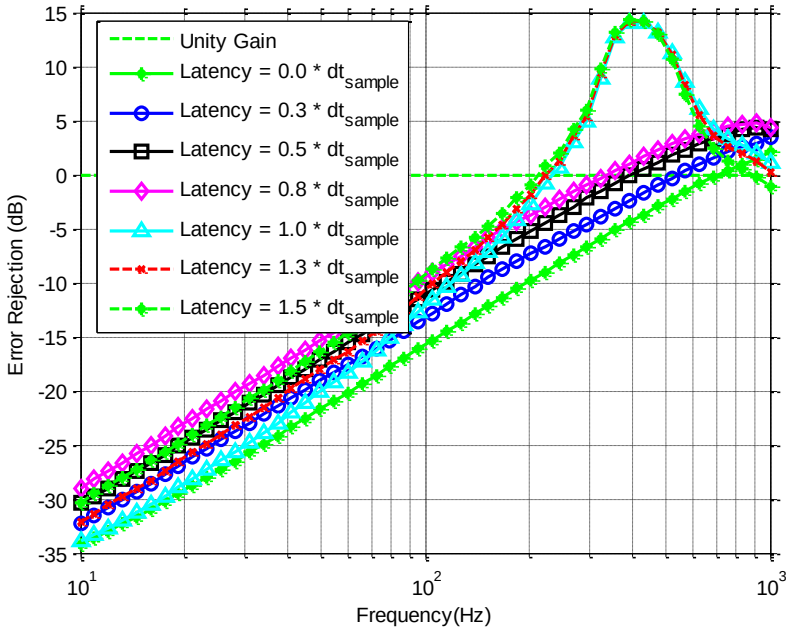


Figure 3 - Effect of Latency on Error Rejection

Figure 4 shows the effect of latency on the unity gain frequency.

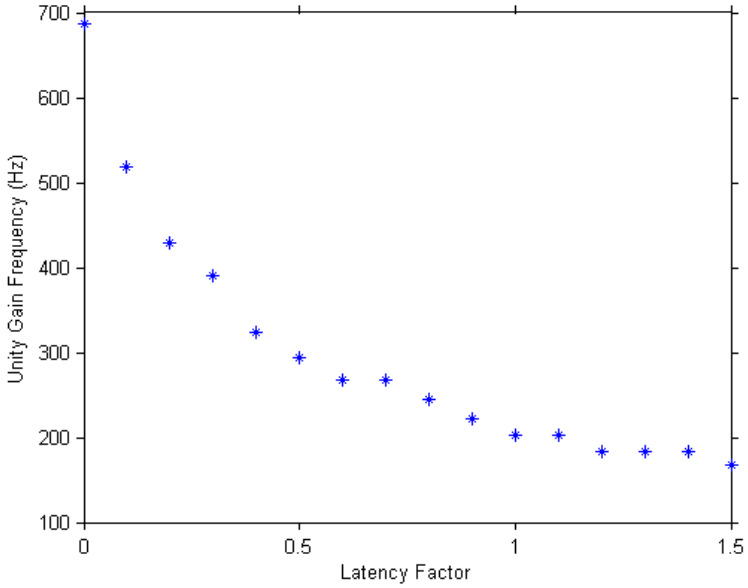


Figure 4 – Effect of latency on the unity gain frequency

Effect of Gain on Error Rejection Ratio

Figure 5 shows the results of using this model to study the effect of gain on the error rejection ratio.

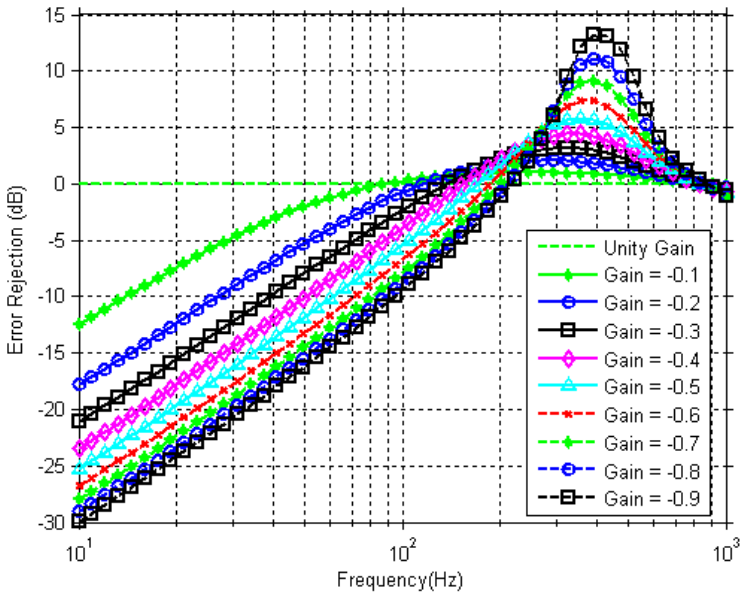
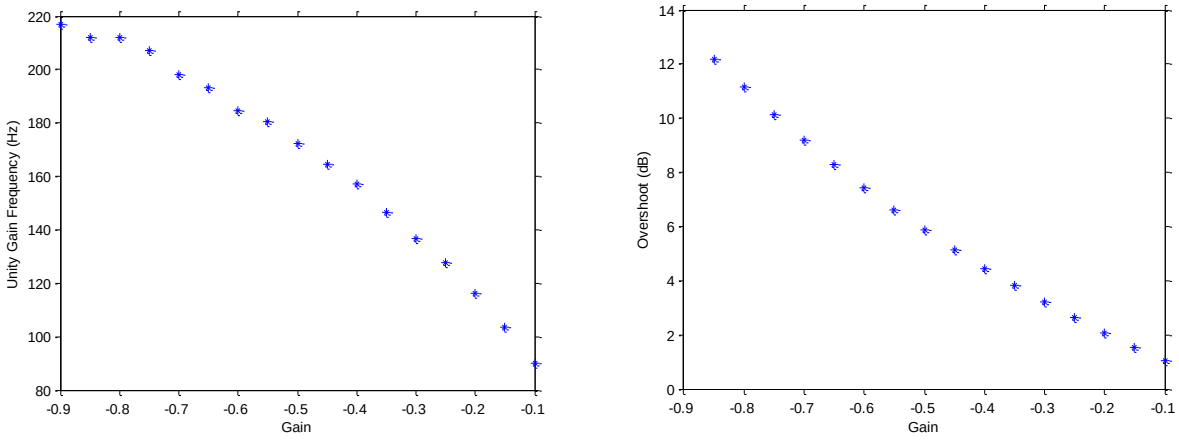


Figure 5 - Error Rejection Ratio for Varying Gain

Figure 6 (a) shows the effect of control gain on the unity gain frequency. Figure 6 (b) shows the overshoot of the error rejection ratio with respect to the control gain.



(a) (b)

Figure 6 - The unity gain frequency (a) and the overshoot (b) with respect to the gain.

Conclusions

This application note presents a simple model for time-domain control and explores the effect of latency and control gain on the error rejection ratio. In the future, this script can be extended to include additional effects like actuator dynamics and more advanced control methodologies.

Appendix: Full Matlab Code

```
%Time Domain Control Simulation
close all; clear all; clc;
dbg=0;
fsample = 2500; %11x11 sub-apertures,  88x88 pixels
gain = -0.6;
leak = 0.99;
dtSample = 1/fsample
dtLatency = 1.5 * dtSample
dt = dtSample * 0.1;
fv = logspace(1,3,100);
frequencyMax=max(fv)

%scan over frequency
if (dbg) figure; end;
b=0;
for f = fv;
    b=b+1;
    tmax =3/f; %three periods

    %clear saving variables
    clear errSave;
    clear cmdSave;
    clear signalSave;
    clear newcmd;
    clear applyCmd;

    cmd=0;
    err=0;
    sampleTime=0;
    applyCmd(1) = sampleTime + dtLatency;
    c=0;
    newcmd(1)=0;
    for t = 0:dt:tmax
        c=c+1;
        signal = sin(2*pi*f*t);
        if (t>sampleTime)
            %newcmd is a FIFO queue
            newcmd(end+1) = leak*newcmd(end) - gain*err;
            %newcmd(end+1) = leak*cmd - gain*err; % alternative approach
                        % which is likely better
            applyCmd(end+1) = sampleTime + dtLatency;
            sampleTime = sampleTime + dtSample;
        end;
        if (length(applyCmd)>1)
            if (t>applyCmd(2))
                cmd = newcmd(2);
                %newcmd is a FIFO queue
                newcmd = newcmd(2:end);
                applyCmd = applyCmd(2:end);
            end;
        end;
        err = signal - cmd;
```

```
        signalSave(c)=signal;
        cmdSave(c)=cmd;
        errSave(c)=err;
    end;

    if (dbg)
        clf;
        plot(signalSave,'r*-');
        hold on;
        plot(cmdSave,'bo-');
        plot(errSave,'g--');
        legend('signal','command','error');
        drawnow;
    end;
    rmsError(b)=RMS(errSave);
    rmsSignal(b)=RMS(signalSave);
end;
%% summarize results
figure;
rmsErrdB=20*log10(rmsError./rmsSignal);
Overshoot = max(rmsErrdB)
semilogx(fv,rmsErrdB,'b-','linewidth',2);
xlabel('Frequency(Hz)');
ylabel('Error Rejection (dB)');
grid on;
hold on;
fc=200;
hold on;
ideal = 20 * log10(fv);
ideal = ideal - ideal(1) + rmsErrdB(1);
for ii=1:length(fv)-1;
    if (rmsErrdB(ii)<0 &&rmsErrdB(ii+1)>=0) break; end;
end;
mi=ii;
title(sprintf('Unity Gain Crossover = %.1f Hz',fv(mi)));
semilogx(fv,ideal,'r--','linewidth',2);
unityGainOffset = (ideal(mi)-rmsErrdB(mi));
semilogx(fv,ideal - unityGainOffset,'m--','linewidth',2);
semilogx(fv,fv.*0+0,'g--','linewidth',2);
legend('Response','20 dB/dec Asymptote','20 dB/dec at Unity Gain','Unity
Gain','Location','Best');
```