Magnetic Bearing Lab # 5: Digital Controller Design for a Magnetic Bearing System*

Nancy Morse, Roy Smith, Brad Paden and Vivek Moudgal
e-mail: nancy@me-server.ucsb.edu
Version 1

September 13, 1996

In *Magnetic Bearing Labs # 3* and *# 4*, we designed a notch filter and an analog lead/filter controller to control a single axis of the Magnetic Moments MBC 500 magnetic bearing system. This single-input single-output system is represented as $G(s)$ in the closed-loop configuration shown in Figure 1 below and is detailed in block diagram form in Figure 2. In Figure 1, $N(s)$ represents our analog notch filter and $C(s)$ represents our analog lead/filter controller henceforth called simply the lead controller. In this lab, we will determine a discrete-time approximation to this notch filter/lead controller combination. As shown in Figure 3, a digital implementation of such a discrete-time controller will include an analog to digital converter (ADC) on the input of the controller, and a digital to analog converter (DAC) on the output of the controller. The controller itself will be implemented in software using a computer. The algorithm for implementing the controller will be expressed as a Z-transform representation of the notch and lead controller, $N(z)$ and $C(z)$, respectively. In the lab that follows, we will discuss the choice of the sampling frequency for the digital system and convert our analog notch filter and lead controller into an equivalent digital representation. We

*Lab Requirements: Completion of *Magnetic Bearing Lab # 2: Magnetic Bearing System Identification*, *Magnetic Bearing Lab # 3: Notch Filtering of Resonant Modes*, and *Magnetic Bearing Lab # 4: Lead Controller Design for a Magnetic Bearing System*. The equipment needed to perform this lab includes an MBC 500 Magnetic Bearing, an IBM -PC/AT type computer, a dSPACE DS1003 Processor Board, a DS2001 I/O Board, a DS2101 I/O Board, various coaxial and other types of cables and adaptors. The hardware required to perform the optional section 5.4 includes a Burr-Brown UAF42 14-pin DIP chip, a $\pm$ 12V or $\pm$ 15V power supply, an integrated circuit breadboard, three 100k Ω potentiometers, a 1M Ω resistor and circuit wire. The software required to perform this lab includes MATLAB, μ-TOOLS, SIMULINK, The Math Works' Real-Time Workshop and dSPACE Real-Time Interface.

will then implement this digitally using the dSPACE DS1003 Processor Board; the dSPACE ADC and DAC boards, the DS2001 and DS 2101 I/O Boards, respectively; and supporting software. Finally, we include an optional discussion on the design and implementation of an anti-aliasing filter.

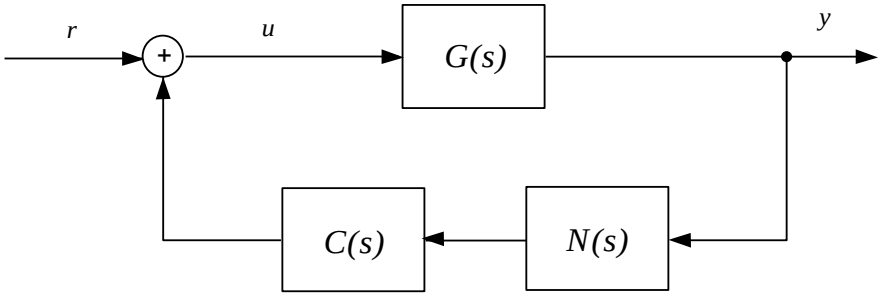

Figure 1: Magnetic Bearing Continuous-Time Closed-loop Configuration

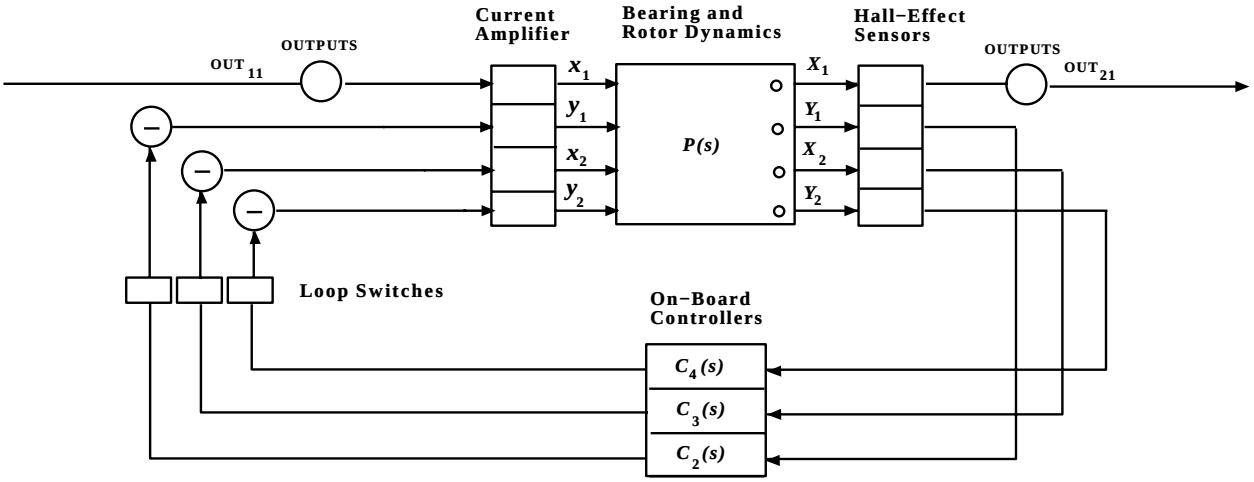

Figure 2: Magnetic Bearing Block Diagram

5.1 Choosing the Sampling Frequency

The analog to digital converter in the digital controller design shown in Figure 3 converts its analog input into a digital signal at discrete time instants as determined by its internal sampler. The frequency of these sample times is referred to as the sampling frequency. The choice of this one design parameter is critical. (See [1] pp. 483-515.) The higher the sampling frequency, the closer the digital controller will approximate the function of an analog controller; however, the higher the sampling frequency, the more expensive the con-

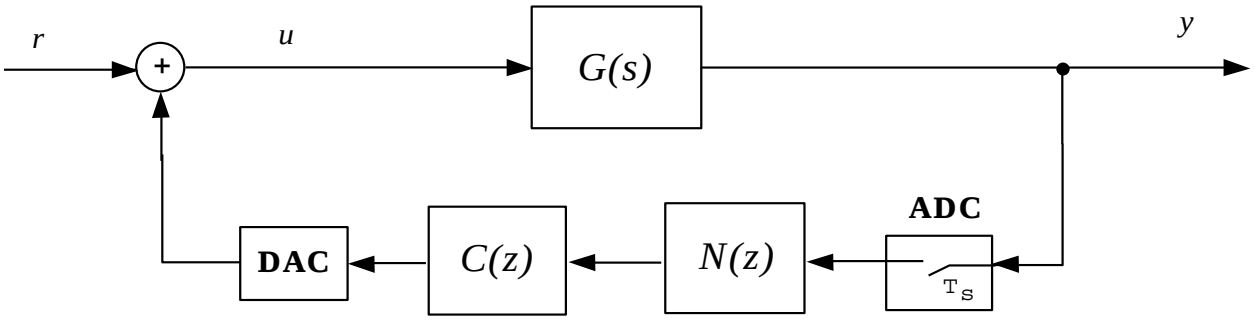

Figure 3: Magnetic Bearing Discrete-Time Closed-loop Configuration

troller hardware will be. Thus, the slowest sampling frequency which provides for adequate controller performance should be used.

The absolute lowest frequency which should be considered for the sampling rate of a digital system is twice the maximum frequency to which our controller will be applying significant control effort. Call this frequency ω_c. A sample rate of $2\omega_c$ is, according to the sampling theorem , the minimum frequency which will avoid aliasing for frequencies less than or equal to ω_c. (See [1] pp. 107-111.) Often, ω_c is simply the closed-loop system bandwidth; however, in our case and certain others, the controller acts over a wider range of frequencies. In our system, the controller acts to provide stabilizing damping to the rotor resonances near 800 Hz and 2000 Hz. Thus, in our case, ω_c is near 2000 Hz and we must choose our sampling frequency greater than twice the "2000 Hz" resonance frequency to avoid aliasing of frequencies within this range.

Another factor to consider in choosing the sampling rate is the smoothness of the controller output. For slow sampling rates the response of the controller to changing inputs is delayed causing a very rough and discontinuous output. In order to keep the controller delay within 10% of the analog system output rise time, the sampling frequency should be chosen at least 20 times the closed-loop system bandwidth.

The next three exercises outline the process for determination of the closed-loop system bandwidth and the precise frequency of the "2000 Hz" mode for the purpose of choosing the sampling frequency. Finally, in Exercise 4 we will choose the sample rate.

Exercise 1: In *Magnetic Bearing Lab # 3*, we designed a notch filter having transfer func-

tion

$$N(s) = \frac{A_N(s^2 + \omega_n^2)}{s^2 + s\frac{\omega_n}{Q} + \omega_n^2}.$$

In Exercise 11 of *Lab # 3*, we determined values for each of the notch parameters A_N, ω_n and Q. Similarly, in *Magnetic Bearing Lab # 4* we designed a lead controller with filtering of high frequencies. The transfer function for this controller took the form

$$C(s) = k\frac{Ts + 1}{(\alpha Ts + 1)(T_0 s + 1)}$$

where the controller parameters k, T, α and T_0 were determined in Exercise 8. Using the lead controller and notch transfer functions we will calculate the frequency response corresponding to each. In the MATLAB algorithm below, two frequency response vectors are created. The vector `C` is created to contain the lead controller frequency response, and the vector `N` is created to contain the notch frequency response each at the frequency points in radians per second contained in the vector `rps`. The algorithm expects the user to have already defined the lead controller and notch parameters as the variables `k`, `T`, `alpha`, `T0`, `An`, `omegan`, and `Q`.

```
leadnum=[k*T k];
leadden=conv([alpha*T 1],[T0 1]);
rps=2*pi*logspace(1,4,801);
C=freqs(leadnum,leadden,rps)';
notchnum=[An 0 An*omegan^2];
notchden=[1 omegan/Q omegan^2];
N=freqs(notchnum,notchden,rps)';
```

Using the algorithm above and the controller parameter values determined previously, find the frequency response vectors `C` and `N` corresponding to the lead controller and the notch filter, respectively.

Exercise 2: The closed-loop transfer function corresponding to the continuous-time system

shown in Figure 1 is given as follows:

$$\frac{y}{r} = \frac{G}{1 + GNC}.$$

Using the bearing frequency response vector `g` obtained in Exercise 9 of *Lab # 2* and the vectors `N` and `C` determined above, calculate the closed-loop system frequency response `systemfr` by using the MATLAB command given below.

```
systemfr=g./(ones(size(g))+g.*N.*C);
```

Now, using the MATLAB algorithm below, plot the Bode diagram of the closed-loop system and determine the closed-loop system bandwidth. Recall that the bandwidth is defined to be the point after which the magnitude is consistently -3dB below its DC gain. For ease of determining the closed-loop bandwidth, the algorithm that follows plots a line at the -3dB point on the magnitude plot.

```
vsys=vpck(systemfr,rps);
vplot('bode_g',vsys,10^(-3/20))
```

Exercise 3: Now using the Bode plot of the closed-loop system given in Exercise 2 above, determine the frequency of the second flexible rotor mode. This should be located near 2000 Hz. For a more exact determination of this frequency, it may be helpful to create and search through a vector of the magnitude of the system frequency response stacked on top of the corresponding frequencies in Hz. The following MATLAB command creates such a vector and displays it to the screen.

```
format short e
[abs(systemfr) rps/2/pi]'
```

Exercise 4: Using the information obtained in Exercises 1-3, determine the smallest sampling frequency that can be used to control this magnetic bearing system with our lead

controller and notch filter in the feedback loop without having aliasing problems in the region of our rotor resonances. Moreover, select a sample rate which will keep the controller output delay within 10% of our analog signal risetime.

Once a digital hardware implementation has been chosen, there is usually no reason not to sample as fast as the hardware will allow. The controller performance will most likely improve with faster sampling rate. For this lab, we will outline the use of the dSPACE DS1003 Processor Board and DS2001 and DS2101 I/O Boards. Using this hardware and the supporting software, the maximum sampling rate which can be used to implement our lead controller with notch filter is most likely in the range of 25kHz to 35kHz. This rate should be compared to the minimum rate determined in Exercise 4. For our application, it will almost certainly suffice.

5.2 Designing a Digital Approximation to the Analog Controller

In *Magnetic Bearing Lab # 3* and *Lab # 4* we designed and tuned a lead controller and notch filter which gave good analog performance. We would now like to find a digital lead controller and notch filter which closely approximate the function of their analog counterparts. These "equivalent" digital components can be found by a bilinear transformation of each analog transfer function. (See [1] pp.133-141.) A bilinear transformation is one way to convert a Laplace transform representation of a transfer function to a Z transform representation of the same function. Such a transformation takes the form:

$$z = \frac{1 + \frac{T_s s}{2}}{1 - \frac{T_s s}{2}}.$$

The transformation maps the stable region in the Laplace domain, the left half of the complex plane, into the stable region of the Z domain, the inside of the unit circle. In this way, stable analog filters are always mapped to stable digital filters. The mapping of frequencies using the bilinear transformation is a non-linear operation satisfying the relationship

$$\tan \frac{\omega_d T_s}{2} = \frac{\omega_c T_s}{2}$$

where ω_d is the discrete-time frequency, ω_c is the corresponding continuous-time frequency, and T_s is the sample period for the discrete-time implementation. Thus, analog frequencies

are warped in this non-linear way in the conversion to a digital representation. In order to correct for this warping effect, a critical frequency for a given filter may be prewarped before performing the bilinear transformation so that after the transformation the critical frequency is warped into its designed value. The bilinear transform with prewarping at the frequency ω_p corresponds to the following mapping:

$$z = \frac{\omega_p + \tan\frac{\omega_p T_s}{2} s}{\omega_p - \tan\frac{\omega_p T_s}{2} s}.$$

Our controller consists of a lead stage which operates mostly in low frequencies, a notch operating near 800 Hz and high frequency filtering at even higher frequencies. Because our sampling frequency is much higher than the frequency of our lead stage, it will not experience much distortion in the transformation. The notch frequency, however, is critical to the design. We would like it to remain at exactly the frequency of the 800 Hz mode after the transformation so that it will effectively notch this mode in the digital implementation. The cutoff frequency of our high frequency filtering is not expected to be as critical to the controller performance as the notch frequency. Thus, in performing the bilinear transformation of our controller, the notch frequency ω_n is the frequency we will want to prewarp. Typically, transfer functions are transformed in small portions to avoid numerical errors in the calculation of the digital transfer function. For this reason, we will transform the notch filter and lead controller separately.

MATLAB has functions built-in to perform the bilinear transform with prewarping. Recall the notch transfer function:

$$N(s) = \frac{A_N(s^2 + \omega_n^2)}{s^2 + s\frac{\omega_n}{Q} + \omega_n^2}.$$

In Exercise 1, you should have created the following vectors representing the numerator and denominator for the notch transfer function.

```
notchnum=[An 0 An*omegan^2];
notchden=[1 omegan/Q omegan^2];
```

The following MATLAB algorithm performs the transformation, creating a discrete-time equivalent transfer function represented by the vectors `dnotchnum` and `dnotchden`. It as-

sumes that the sampling period T_s has been placed in a variable `Ts`.

```
[a,b,c,d]=tf2ss(notchnum,notchden);
[ad,bd,cd,dd]=c2dm(a,b,c,d,Ts,'prewarp',omegan);
[dnotchnum,dnotchden]=ss2tf(ad,bd,cd,dd);
```

Exercise 5: Use the MATLAB algorithm above to find an equivalent discrete-time representation for the continuous-time notch filter we designed in *Magnetic Bearing Lab # 3*.

Exercise 6: Now use the MATLAB algorithm below and the previously defined controller transfer function vectors `leadnum` and `leadden` to perform a bilinear transformation of our lead controller with prewarping at the notch frequency ω_n. Again, we assume that the sample period is contained in the variable `Ts`.

```
[a,b,c,d]=tf2ss(leadnum,leadden);
[ad,bd,cd,dd]=c2dm(a,b,c,d,Ts,'prewarp',omegan);
[dleadnum,dleadden]=ss2tf(ad,bd,cd,dd);
```

Exercise 7: Using the algorithm below, plot the discrete-time controller frequency response versus the continuous-time response. Notice the coincidence of the notch frequency in the two cases.

```
num=conv(leadnum,notchnum);
den=conv(leadden,notchden);
cresp=freqs(num,den,rps);
dnum=conv(dleadnum,dnotchnum);
dden=conv(dleadden,dnotchden);
dresp=freqz(dnum,dden,rps/2/pi,1/Ts);
vresp=vpck([cresp' dresp'],rps/2/pi);
vplot('bode_g',vresp)
```

The MATLAB command `axis` may be needed to alter the range of the axes for the plot. The command

`subplot(2,1,1), axis([xmin xmax ymin ymax])`

will change the axes of the magnitude plot where appropriate numbers are placed in the variables `xmin, xmax, ymin` and `ymax`. Similarly, the command

`subplot(2,1,2), axis([xmin xmax ymin ymax])`

will alter the axes of the phase plot.

5.3 Building the Digital Controller

In order to build our digital controller, we will use the dSPACE Processor Board DS1003 and dSPACE ADC and DAC boards, the DS2001 and DS2101, respectively. The dSPACE hardware and corresponding software will automatically convert a digital controller modeled in SIMULINK into a working digital controller. It then allows us to conveniently tune the controller parameters until our controller is sufficiently robust [1].

First, we must create a MATLAB file, which when executed, will define in the MATLAB workspace, all of the variables used to define our controller in our SIMULINK model. For our controller discrete-time implementation, the variables `dnotchnum`, `dnotchden`, `dleadnum` and `dleadden` will be used by the SIMULINK model to define the notch and lead controller transfer function numerators and denominators. Thus, we would like to combine the MATLAB code used in Exercises 5 and 6 to form one file which will define all of these when executed. This file will also be used to perform the tuning of the controller. Thus, we would like all of the tuned parameters listed at the top of the program for easy access. Then the controller numerators and denominators should be calculated using these parameters. For example, one set of tuned parameters could be `An`, `omegan` and `Q` for the notch filter and `k`, `alpha`, `T` and `T0` for the lead controller. The tuned parameters should be parameters which completely define the controller and which make tuning easiest for the designer. For example, instead of `alpha`, `T` and `T0`, the designer may find it simpler to think of the controller pole and zero locations in Hz to perform the tuning.

[1] For this portion of the lab, it is assumed that the dSPACE hardware and software have been correctly installed on an IBM PC/AT or similar machine and that the user has a basic working knowledge of SIMULINK.

Exercise 8: Create a MATLAB file which defines the notch and lead controller discrete-time transfer function numerators and denominators. Define these using tunable parameters of choice listed near the top of the program for easy access.

We now must define a SIMULINK model of the notch and lead controller combination similar to the one shown in Figure 4. Typing `simulink` at the MATLAB prompt will pull up the SIMULINK Block Library. Begin a new file for building the SIMULINK controller model by choosing "File" and then "New..." from the menu at the top of the SIMULINK window. The SIMULINK window contains blocks which can be used to define the SIMULINK model for the notch filter and lead controller discrete-time transfer functions. These can be found by double clicking on the block entitled "Discrete". Simply drag the "Discrete Transfer Fcn" block from the "Discrete" window into the new file window. Two such blocks will be needed; one for the notch and one for the lead controller. The name of each block can be modified and its parameters defined. The list of definable parameters can be accessed by double clicking with the left mouse button on the block in question. In each case, type in the variable names corresponding to the numerator and denominator defined in Exercise 8. You will also need to specify the sample time in seconds.

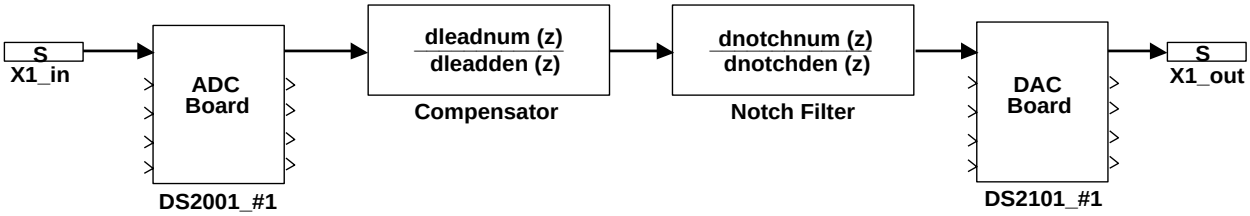

Figure 4: Compensator Simulink Block Diagram

Now the dSPACE hardware, ADC and DAC blocks as well as input and output plugs, should be added to the block diagram. These can be accessed from a separate dSPACE block library by typing `dslib` from the MATLAB prompt. These blocks can be added to the controller block diagram in the same way that SIMULINK library blocks were used previously. Define the parameters for each of these blocks and connect them up as shown in Figure 4.

Now that our controller model had been defined in SIMULINK, we need to choose the sample rate at which the controller will operate. The sample rate may be specified from the

"Code" menu option at the top of the SIMULINK model. By selecting "Real-time Options..." from this menu, the sample period in seconds may be specified by the parameter "Step Size". An initial sample rate will need to be chosen based on an estimate of the time required to complete one controller cycle. For an initial design, a high estimate of the step size is probably best to ensure that the controller will run without any problems; however, this time can later be adjusted until the fastest sample rate has been determined.

Exercise 9: Create a SIMULINK block diagram model of our notch filter and lead controller which includes the dSPACE ADC and DAC boards and input and output plugs. Specify the sample rate at which the controller will operate.

Before executing our controller on the dSPACE hardware, we must connect the magnetic bearing system physically to the I/O boards as described by Figure 5. The magnetic bearing input IN_1 must be connected to the first DAC output port. In addition, the magnetic bearing output OUT_{21} must be connected to the first ADC input port. Now the first loop switch on the magnetic bearing system front panel must be opened to disconnect the first on-board controller.

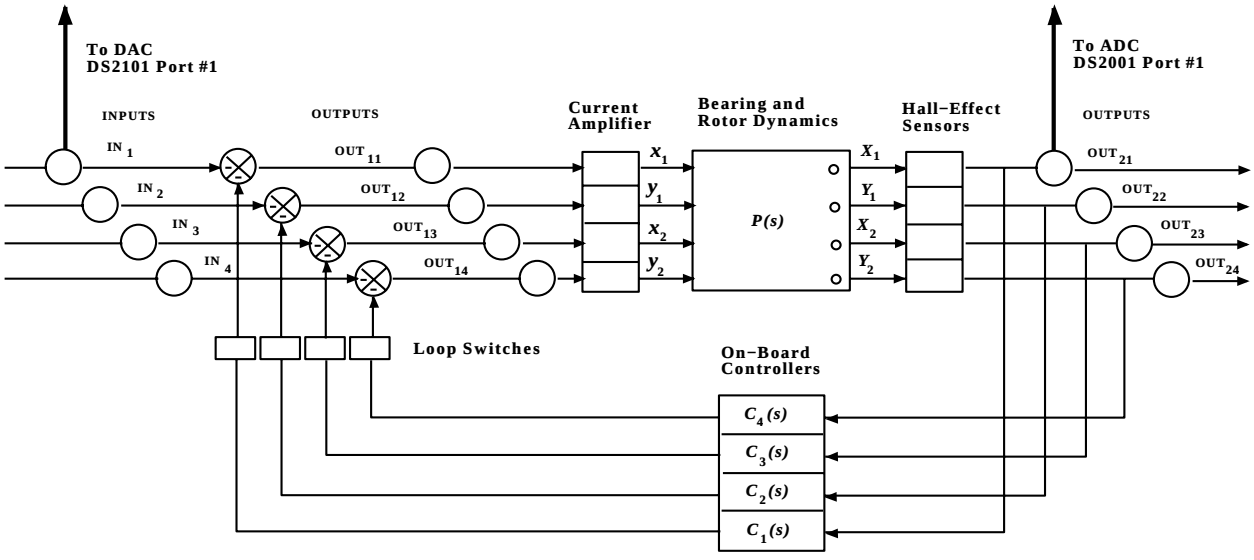

Figure 5: Magnetic Bearing System and Controller Connections

Next, the controller needs to be downloaded to the dSPACE hardware. To activate the device drivers for the dSPACE hardware, simply type the appropriate initialization

command at the DOS or Unix prompt. Then, from the "Code" option along the top of the SIMULINK controller model, choose the option "Generate and Build Real-time". This turns the SIMULINK model into executable code and downloads it to the dSPACE hardware. Controller execution begins immediately. Hence, the bearing system may now be turned on and the performance of the digital controller evaluated.

Exercise 10: Using the procedure described above, connect the bearing system to the dSPACE hardware and download the controller code. Turn on the bearing system and check the controller performance.

Since virtually no controller operates within specifications from its initial design, some tuning of the parameters will probably need to be done. This process begins with modification of the MATLAB file created in Exercise 8 which defines the controller variables used in the SIMULINK model. Once the new parameter values have been entered into the MATLAB file and that file has been executed from the MATLAB prompt, new code can be generated and executed by again selecting "Generate and Build Real-time" from the SIMULINK model. This process can be repeated until the controller meets all specifications.

Exercise 11: Begin tuning the controller to achieve stability and robustness to disturbances. Consult *Magnetic Bearing Lab # 4* for some tuning guidelines. Remember that it is often best to adjust only one parameter at a time, recording the parameter values along with the corresponding response. This will help isolate the effect of each individual parameter.

Several other techniques may be used to achieve stability and robustness in addition to those already discussed. The following is a list of some possibilities.

- If one or more of the bending modes is ringing, it may be desireable to add additional poles (or move existing poles) near the resonance frequencies to add additional phase lag and further reduce the controller gain at those critical frequencies. To determine which mode is ringing, you may notice the tone of the ring and compare it to the tone produced when tapping the bar at the midpoint of the bar for the first bending mode and one fourth the bar length for the second bending mode. Also, you may try

to damp the ringing by physically holding the bar at its midpoint and one fourth its length and notice which is most effective at damping the ringing.

- Digital design enables the use of higher order notch filters and filters with multiple notches with almost no additional cost in design time or time of implementation. MATLAB offers the filter design function `ellip` which will automatically create elliptical filters of a given discription. This may be used to generate wider and deeper notches to notch out the 800 Hz and/or the 2000 Hz bending modes. Details on the syntax for the function `ellip` can be found with the help command within MATLAB. As an example, the following MATLAB command creates a discrete-time notch filter numerator and denominator `dnotchnum` and `dnotchden`, respectively. This notch is second order, has 0.5 dB of ripple in the passband, has a stopband that is 40 dB down and notches from 2000 Hz to 2100 Hz. In this case, `Fs` is the sampling frequency in Hz.

```
[dnotchnum,dnotchden]=ellip(1,0.5,40,2*[2000 2100]/Fs,'stop');
```

- Because of non-linearities and aliasing within the system, higher order frequencies may be exciting lower order bending modes and causing instability. If this appears to be a problem, an anti-aliasing filter may be added at the output of the magnetic bearing system to attenuate higher order frequencies. Details on the design of such a filter can be found in the following optional section.

5.4 Designing an Anti-Aliasing Filter (Optional)

Anti-aliasing filters are used to reduce the gain of the system response to frequencies higher than one half the sampling frequency, referred to as the Nyquist frequency. Because of aliasing, these frequency components will appear to the controller as lower frequency signals and the controller will attempt to compensate for them. This will result in control signals which may degrade instead of help the system performance. Thus, the anti-aliasing filter is

chosen to be a low-pass filter which attenuates signals in this frequency range so that their effect on the control action is reduced.

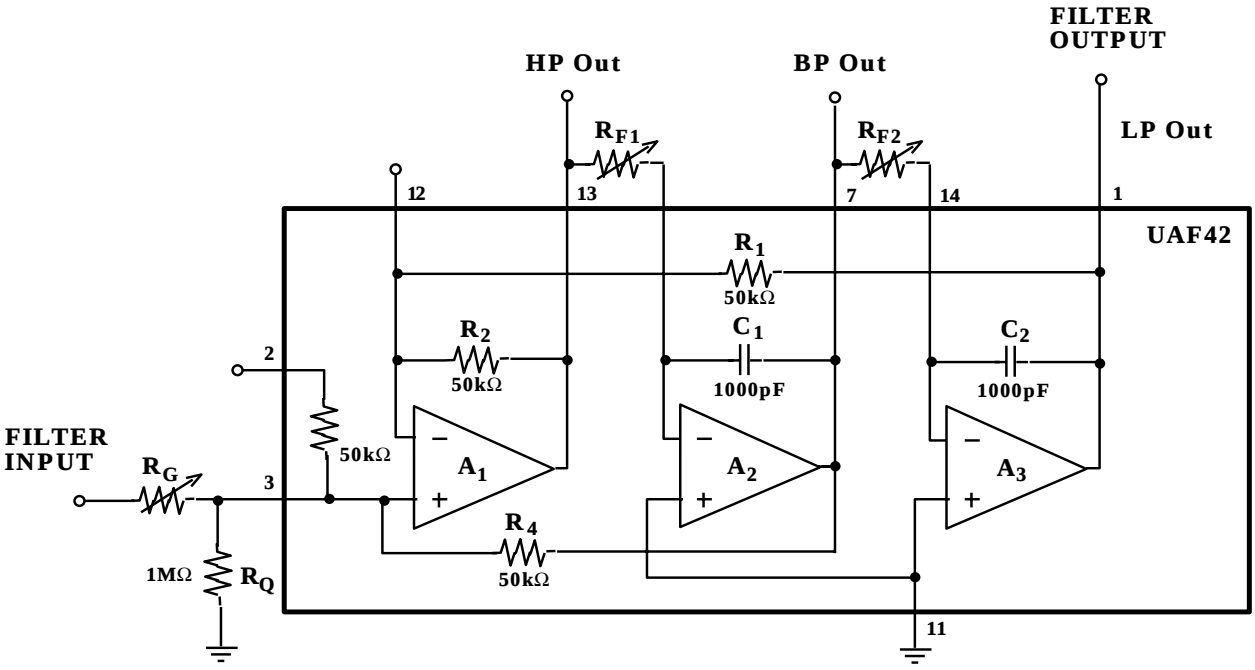

Figure 6: Anti-Aliasing Filter Schematic Diagram

In this lab, we describe the design of an anti-aliasing filter which is a second order low-pass filter with complex conjugate poles. We will use the Burr-Brown filter chip UAF42 to implement the filter with schematic diagram contained in Figure 6. The low-pass filter transfer function we will be implementing is as follows:

$$\frac{A_{LP}\omega_n^2}{s^2 + s\frac{\omega_n}{Q} + \omega_n^2}.$$

An ideal low-pass filter has unity gain and zero phase in its passband and zero gain in its stopband. Of course, real filters cannot completely remove all signals past their cut-off frequency and will also attenuate and add phase lag to some signals lower than their cut-off frequency. Thus, the designer's task is to choose the cut-off frequency and damping of the filter to reach a good compromise between the two filter objectives. Clearly, we would like the filter DC gain to be equal to one. Thus, we must have $A_{LP} = 1$. Choice of ω_n and Q, however, are up to the designer's preference. They should be chosen so that the filter has negligible effect at frequencies where our controller is applying control effort and filters out frequencies greater than the Nyquist frequency.

Exercise 12: The following MATLAB algorithm plots the Bode plot of the low-pass filter transfer function above for given values of ω_n and Q. For comparison purposes, the low-pass transfer function with $\omega_n = 2\pi \times 5000$ and $Q = 1$ is displayed on the same plot. These values for ω_n and Q were chosen as typical values to demonstrate the procedure and should be changed by the designer to correspond to the designer's current "best" filter design to which other designs will be compared.

```
%Define vectors of frequencies
hz=logspace(1,5,801);
rps=2*pi*hz;

%Determine filter transfer function and frequency response
fnum=[omegan^2];
fden=[1 omegan/Q omegan^2];
fsys = nd2sys(fnum,fden);
vresp1 = frsp(fsys,rps);

%Define second filter parameters
omegan2=2*pi*5e+3;
Q2=1;

%Determine second filter transfer function and frequency response
f2num=[omegan2^2];
f2den=[1 omegan2/Q2 omegan2^2];
f2sys = nd2sys(f2num,f2den);
vresp2= frsp(f2sys,rps);

%Form varying matrix and plot via vplot
vresp1hz=scliv(vresp1,1/2/pi);
```

```
vresp2hz=scliv(vresp2,1/2/pi);
vplot('bode_g',vresp1hz, vresp2hz)
subplot(2,1,1), xlabel('Frequency [Hz]')
subplot(2,1,2), xlabel('Frequency [Hz]')
```

Use the above algorithm to search for the filter transfer function which reaches the "best" compromise between no effect on low frequencies and attenuating high frequencies.

For the circuit of Figure 6, the resistances R_G, R_Q, R_{F1} and R_{F2} are the design variables which may be chosen to select the filter gain A_{LP}, the cut-off frequency ω_n, the filter damping factor Q and the input impedance of the filter. The following equations relate each of the circuit variables to each of our controller parameters.

$$\begin{aligned} A_{LP} &= \frac{2}{1+\frac{R_G}{R_Q}+\frac{R_G}{50\times 10^3}} \\ \omega_n^2 &= \frac{10^{18}}{R_{F1}R_{F2}} \\ Q &= \frac{1}{2}+(2.5\times 10^4)\frac{R_G+R_Q}{R_G R_Q}\left(\frac{R_{F1}}{R_{F2}}\right)^{1/2} \\ input\ impedance &= R_G + R_Q \| Z_f \end{aligned}$$

The symbol $\|$ is used to denote the parallel combination of impedances. Here Z_f is the input impedance of the UAF42 filter chip. Z_f should be near 50k ohms. Using the fact that $A_{LP} = 1$, our first equation can be written:

$$R_G = R_Q \| (50 \times 10^3).$$

Combining this equation with that for the input impedance

$$input\ impedance = R_G + R_Q \| (50 \times 10^3)$$

we can see that by selecting

$$\begin{aligned} R_Q &= 1\text{M ohm} \\ R_G &= R_Q \| (50 \times 10^3) = 47.6\text{k ohms} \end{aligned}$$

we can make the input impedance nearly 100k ohms. Then we have the resistances R_{F1} and R_{F2} with which to select both ω_n and Q for the filter.

Exercise 13: Using the equations for ω_n and Q given above, the values for R_Q and R_G calculated above, and the values for ω_n and Q determined in Exercise 12, calculate the values of R_{F1} and R_{F2} so that the circuit of Figure 6 implements the desired low-pass transfer function.

Exercise 14: Build the low-pass filter shown in Figure 6 using potentiometers of appropriate size tuned to the values determined above for the resistances R_G, R_{F1}, and R_{F2}. A 1M Ω fixed value resistor may be used for R_Q.

Exercise 15: Now use the dynamic signal analyzer to check the transfer function of the anti-aliasing filter. Connect the analyzer "Source" signal to the FILTER INPUT and the analyzer "Channel 1" input. Then connect the FILTER OUTPUT to the analyzer "Channel 2" input. Following the procedure outlined in *Magnetic Bearing Lab # 2* Exercises 2-4, obtain a Bode plot of the filter frequency response. Check to see that the DC gain of the filter is unity and that the filter has the correct cutoff frequency. If either of these are incorrect, check your parameter calculations to determine where the error occurred. Modify your circuit and then check its frequency response once more.

Exercise 16: Connect the filter between the bearing system output and the dSPACE ADC input as shown in Figure 7. The system can now be turned on and its performance evaluated.

References

[1] Franklin, Gene F., J. David Powell and Michael L. Workman, **Digital Control of Dynamic Systems**, Second Edition, Addison-Wesley Publishing Co., 1990.

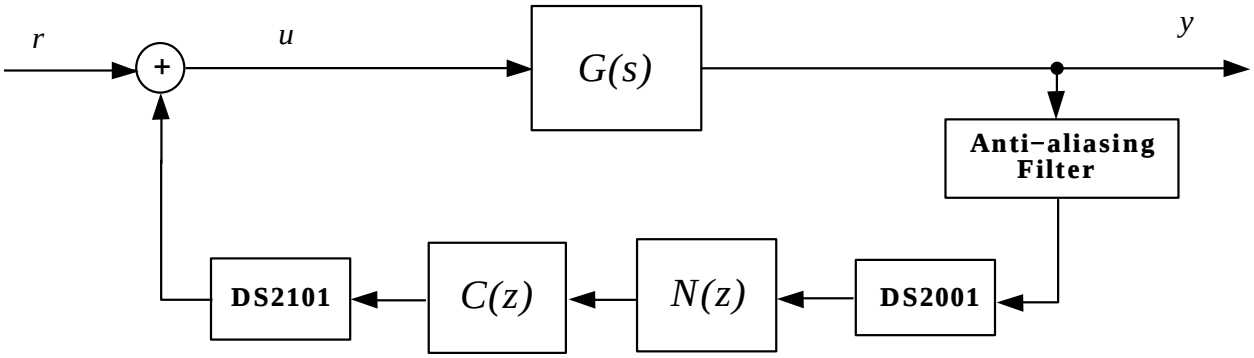

Figure 7: Anti-aliasing Filter Connection