EE 661 *Modern Control*

Project #2: ***Transmission Zeros, Characteristic Loci***

Due Date: Monday, November 17, 2014

PART A

(1) The "compensated open-loop" model in a "regulator configuration" consists of the discrete-time *cantilevered-beam model* followed by the *controller model*. That is, they are in series, with the controller placed in the feedback path, but the loop is not closed (yet).

Download the cantilevered-beam (the system) model from *MyCourses*

`cbeam_3x2_sysmodel_Ts20kHz_194pole__struc.zip`.

This is a highly accurate discrete-time model (20 kHz sample rate) of a doubly-cantilevered-beam. In the zip file, the system model is a 194-pole state-space model, saved in a Matlab structure-variable that has fields **a**, **b**, **c**, **d**, and **ts**.

Download the controller model from *MyCourses*

`cbeam_controller_2x3_Ts20kHz_40pole__struc.zip`.

This is a discrete-time controller, designed for the two-input three-output discrete-time (20 kHz sample rate) cantilevered-beam model. In the zip file, the controller is a 40-pole state-space model, saved in a Matlab structure-variable that has fields **a**, **b**, **c**, **d**, and **ts**.

- ***Mathematically derive*** the (regulator-architecture) compensated open-loop (*i.e.*, the two models in series), which is this**:**

$$\begin{bmatrix} w_{k+1} \\ x_{k+1} \end{bmatrix} = \begin{bmatrix} \bar{A} & \bar{B}C \\ 0 & A \end{bmatrix} \begin{bmatrix} w_k \\ x_k \end{bmatrix} + \begin{bmatrix} \bar{B}D \\ B \end{bmatrix} r_k$$

$$y_k = [\bar{C} \quad \bar{D}C] \begin{bmatrix} w_k \\ x_k \end{bmatrix} + [\bar{D}D] r_k$$

- Based on this equation, construct the 234-pole compensated open-loop model. Do not use Matlab's high-level functions for this. Using the **`ss`** function, construct the **ss**-object model directly from the matrices $\{\bar{A}, \bar{B}, \bar{C}, \bar{D}\}$ that make up the controller, and the matrices $\{A, B, C, D\}$ that make up the 194-pole cantilevered-beam model, as indicated in the equations above. Don't forget to include a 5th argument, the **sample period ts**, so Matlab knows it's a discrete-time model (not a continuous-time model).

(2) Solve the generalized eigenvalue problem (GEVP) that yields the transmission zeros of the compensated open-loop, using Matlab's **`eig`** function. Verify the correctness of that answer by comparing it to the result computed by Matlab's **`tzero`** function. Make a plot that contains both sets of zeros, but use different 'marker' types for each set, such as squares and triangles.

(3) Assume that a scalar gain factor, ρ, is applied equally to both inputs to the system. Using the **`tzero`** function, by the method shown in lecture, compute the closed-loop poles of the system for each one of at least 20,000 different values of ρ, logarithmically spaced from 10^{-3} to 10^{6}. **Be sure to pre-allocate the memory** for the matrix that will store the eigenvalues *(i.e.*, the closed-loop pole locations) at all those different gains, one column for each value of the gain -- otherwise the **for**-loop will be ridiculously slow (it will take awhile, anyway).

Don't use a Matlab ss-object inside a for-loop (it's dog-slow) — make copies of the *A*, *B*, *C*, and *D* matrices and use *those* inside the loop.

Create a root locus of the system by plotting *all* of the closed-loop poles in a single *z*-plane plot. Plot them all with markers only, for example, by the command

```
plot( closed_loop_poles(:),'.','markersize',6,'linewidth',3 )
```

In the *same* axes, also plot the system's transmission zeros as small red "**o**"s, and the open-loop system poles as small green "**x**"s. If done correctly, each root locus will start at an open-loop pole and end near one of the transmission zeros (if you took the gain all the way to infinity, the loci would end exactly where the transmission zeros are).

Also, in the root locus, use cyan squares to highlight the closed-loop poles where the gain ρ is exactly unity, on each branch of the root locus. These locations are the "design values" that the closed-loop poles are intended to have, using the given controller.

You must use my function "**zgrid_hires**" to overlay high-resolution unit circle, damping and natural-frequency lines.

Don't use Matlab's "**zgrid**" function, because it creates a "low-resolution" plot of the unit circle and the lines of constant damping factor and constant natural-frequency. Matlab's function is noticeably defective at low frequencies (*i.e.*, near $z = +1$), where it is obviously "reticulated" (*i.e.*, it doesn't look smooth) because not enough different points are plotted in that region.

Instead, use the "high-resolution" zgrid m-function, **zgrid_hires**, which can be downloaded in a zip-file from myCourses. This function creates much better damping (zeta) and natural-frequency (ω_n) grid lines than Matlab's function does, particularly noticeable at low frequencies (*i.e.*, near z = 1).

Be sure to title and label your plot.

Continued on next page …

PART B

Compute the compensated open-loop FRF, based on the **measured system-FRF**, followed by (*i.e.*, multiplied by) the controller-FRF computed from the controller model (that's the same 40-pole controller state-space model as in Part A).

The reason to use the ***measured system-FRF*** is that there is no better model of the cantilevered-beam, itself. You will have to download this measured FRF data from *myCourses*, in a zipped mat-file by the name

`cbeam_3x2_frfdata_Ts20kHz_hlfadv_20kPts__struc.zip`

This mat-file contains a structure-variable called

`cbeam_3x2_frfdata_Ts20kHz_hlfadv_20kPts`

which has fields "**`ResponseData`**" (3x2x19999 matrix of complex numbers) and "**`Frequency`**" (19999x1 vector of frequencies, in units of radians/second).

When you compute the FRF of the controller, make sure that you use the same set of frequencies (*i.e.*, the 19999 frequencies) that are in the downloaded FRF data. After multiplying the controller FRF times the measured cantilevered-beam FRF (*slice*-by-*slice*), the resulting matrix should be **2x2x19999.**

Compute the "**characteristic loci**" of the compensated open-loop FRF, and plot these as loci in a Nichols-type plot. Plot the data first, before calling Matlab's "**`ngrid`**" function to create the Nichols grid, because the **`ngrid`** function will use the existing plot to determine the *x*-axis range that needs to have the grid (notice that the grid is periodic along the *x*-axis).

There will be two loci (curves) in the plot, because each *slice* (*i.e.*, each index along the third dimension) of the compensated FRF is 2-by-2, and therefore contributes two eigenvalues to the characteristic loci, at each frequency.

What to Submit

Submit a separate m-file for each part, ready-to-run, as email attachments. Before you submit your m-files, test that they actually run under the following conditions**:** In Matlab, clear the entire workspace by typing **`clear`** and then type the name of your m-file. That's how I will test them. Do not put the **`clear`** command in the m-file, itself, that's very poor programming practice.

If your functions call other m-files that you have written, be sure to also include those other functions as attachments. Otherwise, your functions will not run in my Matlab environment.
