MIMO Robust Design Example

Please hand in a short (2-3 page) summary of the ideas below. Consolidate plots so that the message is uncluttered and clear. Aim for Friday night, May 23 at 8:00 PM, which is the end our scheduled final exam period. There will be no final exam.

- Generalized open-loop plant diagram

- State-space for $P$: $\alpha = 10,$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} 0 & \alpha & 1 & 0 \\ -\alpha & 0 & 0 & 1 \\ 1 & \alpha & 0 & 0 \\ -\alpha & 1 & 0 & 0 \end{bmatrix}$$

- Sensor Noise weights

$$w_n(s) = \frac{12(s + 25)}{5(s + 6000)}, \quad W_n := w_n(s)I_2$$

- Output disturbance weights (tracking objective)

$$w_p(s) = \frac{s + 4}{2(s + 0.02)}, \quad W_p := w_p(s)I_2$$

- Uncertainty weights

$$W_1(s) = \frac{10(s + 4)}{s + 200}, \quad W_2(s) = \frac{10(s + 24)}{3(s + 200)}$$

- Uncertainty model: $z_i = \hat{\delta}_i w_i,$ and

$$\hat{\delta}_1 \in S, \quad \|\hat{\delta}_1\|_{\infty} \leq 1, \quad \hat{\delta}_2 \in S, \quad \|\hat{\delta}_2\|_{\infty} \leq 1$$
Tasks:

1. Interpret the weight $w_p$ as a tracking objective in terms of bandwidth (rise-time), steady-state tracking error, and sensitivity peaking in terms of disturbance rejection.

2. Create a SYSIC program to build the open-loop system $G$, shown above, with inputs/outputs ordered as

$$
\begin{array}{cccc}
   & w & \leftarrow & z \\
   e & \leftarrow & d \\
y & \leftarrow & u \\
\end{array}
$$

3. Using `dkitgui`, design a robust controller to “minimize” the appropriate $\mu$ from the $[z;d]$ to $[w;e]$ channels. What is the uncertainty block structure? You should get $\max_\omega \mu \left[ F_L(G, K) \right] \approx 0.87$. In the Parameter window, be sure to increase the the Max Auto-Order in the D-scale Prefit frame. Make the maximum order 10 or so. That should be more than enough.

4. Check that the controller is stable (no guarantee it will be, the guarantee is that it will be stabilizing, but not necessarily stable itself). Try doing truncated balanced realizations to reduce the order of the controller.

At the command line, this would look like (say you are using $K7$).

```matlab
>> spoles(K7)
>> rfid(spoles(K7))
>> [K7b,hsv] = sysbal(K7);
>> [hsv (1:length(hsv))']
>> semilogy(1:length(hsv),hsv,1:length(hsv),hsv,'+');
>> K7red = strunc(K7b,12); % try a 12th order truncation
>> clp = starp(G,K7red);
>> rfid(spoles(clp)) % check that the closed-loop is stable
>> clpg = frsp(clp,OMEGA); % take frequency response
>> mubnds = mu(clpg,[1 1;1 1;4 2]); % block structure includes
>> % the performance block as well. In DKITGUI, this
>> % was automatically handled based on the
>> % error/disturbance dimensions.
```

In the tool, you can enter `strunc(K7b,12)` in the Setup window in the box labeled `<Controller>`. Then press the `<Controller>` button. This will load that particular value for the controller, and will enable the Form Closed Loop button back
in the **Iteration** window, as `<Form Closed Loop>`. If you press that, the closed-loop will be formed with this just-loaded controller, and the program will verify that the closed-loop is stable (otherwise message will appear in Message bar at window bottom). Then you can run **Frequency Response** and **Compute Mu** to accomplish the same as the last 4 commands in the sequence above. Then, the 12 can be changed to a different truncation order, followed by a click on `<Controller>` button in **Setup**, and quickly rerun the 3-step analysis.

Using this brute-force approach (reduce, recompute), find a reduced order controller that stabilizes the nominal system and achieves

\[
\max_{\omega} \mu [F_L(G, K_{\text{red}})] \approx 0.93
\]

5. Use the controller in a traditional “tracking” architecture. Simulate the response of the closed-loop system to unit-step reference inputs in each of the two channels. Do this for a variety (> 20) of “randomly” generated \(\hat{\delta}_1\) and \(\hat{\delta}_2\) that satisfy the norm bound. Comment on the robustness of the MIMO tracking capability, especially with regard to the objectives implied by the performance weight \(w_p\).

6. Make the value of \(\alpha\) uncertain, say

\[
\alpha = 10 (1 + \beta \delta_{\alpha})
\]

where \(\beta\) is a fixed real number, representing the percentage uncertainty in \(\alpha\) (away from nominal value of 10). Build a system \(H\) with additional uncertainty inputs/outputs, \(z_{\alpha}/w_{\alpha}\), so that a feedback of \(z_{\alpha} = (\delta_{\alpha}I_2) w_{\alpha}\) (i.e., \(F_U(H, \delta_{\alpha}I_2)\)) represents the uncertain \(P\).

7. Using the reduced-order controller from above, determine (to within the accuracy of the \(\mu\) bounds) the largest value of \(\beta\) such that the uncertain system (with one real parameter and two unmodeled dynamics) is robustly stable. Here you should take the bounds on the unmodeled dynamics perturbations to be 1, as described earlier.

8. Modify the sensor noise weight to see how that affects the controller gain, and the open-loop gain \(PC\). Specifically, try

\[
>> \text{wnmod} = \text{sow}(0.01,70,30);
>> \text{OMEGA} = \text{logspace}(-2,3,100);
>> \text{wng} = \text{frsp(wn,OMEGA)};
>> \text{wnmodg} = \text{frsp(wnmod,OMEGA)};
>> \text{vplot('liv,lm',wng,'r',wnmodg,'g')};
\]
Note that this implies much more sensor noise than in the previous design, and will (hopefully) force the controller gain (and the open-loop gain) to be reduced at high frequency.

9. Now, redo the design with this sensor noise weight. In the SETUP window, type `mod` in the Iteration Suffix panel. This way, the variables put into the workspace will have that as a suffix (like `K5mod`). I get $\mu$ down to about 0.96 in 5 iterations. Then try

```matlab
>> K5g = frsp(K5,OMEGA); % from the original design
>> K5modg = frsp(K5mod,OMEGA);
>> vplot('liv,lm',K5g,'r',K5modg,'g');
>> L5 = mmult(P,K5);
>> Lmod5 = mmult(P,K5mod);
>> L5g = frsp(L5,OMEGA);
>> L5modg = frsp(L5mod,OMEGA);
>> vplot('liv,lm',L5g,'r',L5modg,'g');
>> vplot('liv,lm',vsvd(L5g),r',vsvd(L5modg),g');
```

Indeed, beyond 40 rad/sec, the gains are significantly reduced, and roll-off much faster than before.

10. Check that `K5mod` is indeed stable, and using a balanced realization, verify that some of its states can probably be truncated. There is no need to go through the reduction that we did earlier, but you can if you like. Redo the simulations in part 5, and note any differences in performance.