Structured H_∞ Synthesis in MATLAB

P. Gahinet * **P. Apkarian** **

* *MathWorks, 3 Apple Hill, Natick, MA 01760, USA (Tel: +1 508-647-7368; e-mail: pascal@mathworks.com).*
** *ONERA-CERT, Centre d'études et de recherches de Toulouse, Control System Department, 2 av. Edouard Belin, 31055 Toulouse, France - and - Institut de Mathématiques, Université Paul Sabatier, Toulouse, France (Tel: +33 5.62.25.27.84 ; e-mail: apkarian@cert.fr)*

Abstract: This paper presents new MATLAB-based tools for tuning fixed-structure linear control systems using the H_∞ methodology. Unlike traditional H_∞ synthesis, these tools can directly tune arbitrary control architectures consisting of one or more feedback loops and one or more fixed-order, fixed-structure control elements. This makes them ideally suited for deploying H_∞ techniques in real-world applications.

Keywords: Structured controllers, decentralized control, H_∞ synthesis, robustness, MATLAB.

1. INTRODUCTION

H_∞ theory and its refinements Stein and Doyle [1991], McFarlane and Glover [1992], Zhou et al. [1996] provide a powerful frequency-domain framework for capturing standard control design requirements such as speed of response, bandwidth, disturbance rejection, and robust stability. This framework extends classical control techniques such as Bode loop shaping or gain/phase margin analysis to multi-loop and MIMO control architectures. Design requirements are expressed in terms of H_∞ gain constraints, and efficient algorithms are available to synthesize MIMO controllers that meet these requirements Doyle et al. [1989], *Robust Control Toolbox* [2011]. Yet traditional H_∞ synthesis has practical limitations that have slowed its adoption in industry. In particular, H_∞ controllers are monolithic whereas most embedded control architectures are decentralized collections of simple control elements such as gains and PID controllers. In addition, the order (complexity) of H_∞ controllers tends to be high whereas embedded controllers tend to have low complexity. As a result, engineers must either abandon familiar decentralized architectures, or painstakingly re-interpret the results of H_∞ synthesis in terms of their specific architecture. Neither option being easy, hand-tuning or optimization-based tuning tend to remain the norm.

This paper presents new tools for *Structured H_∞ Synthesis* that overcome the limitations listed above. These tools leverage state-of-the-art nonsmooth optimizers Apkarian and Noll [2006a], Apkarian et al. [2007], Burke et al. [2006] to directly and efficiently tune arbitrary control architectures. By "arbitrary," we mean any single- or multiple-loop block diagram arrangement containing any number and type of linear control elements, from simple gains and PIDs to more complex notch filters and observer-based controllers. Note that H_∞ synthesis of fixed-order, fixed-structure MIMO controllers is just a special case. While the underlying optimization programs are typically non-convex and NP-hard, the nonsmooth optimizers fare well on a wide range of applications and often produce results on par with full-order H_∞ synthesis but for much simpler controller structures. Finally, the optimization-based nature of these tools makes them well suited to tackle challenging extensions such as multi-model, multi-objective H_∞ synthesis with time-domain specifications Bompart et al. [2008], Simoes et al. [2009].

The paper is organized as follows. Section 2 discusses the standard formulation of structured H_∞ synthesis and the representation of tunable control elements. Section 3 reviews the basics of expressing design requirements as H_∞ constraints. Section 4 gives a brief overview of the nonsmooth solvers at the heart of this approach. Section 5 highlights the key features of related software tools available in *Robust Control Toolbox* [2011]. Finally, Sections 6 illustrates the potential of this technology on a realistic example.

2. FRAMEWORK FOR TUNING FIXED CONTROL STRUCTURES

As mentioned earlier, our starting point is any linear control architecture with one or more fixed-structure blocks to tune. An example of such architecture is shown in Figure 6 where the shaded blocks are tunable. Since there are infinitely many possible architectures, we use a standardized representation called *Standard Form* that is both general and simple enough to work with. The *Standard Form* is depicted in Figure 1 and consists of two main components:

- An LTI model $P(s)$ which combines all fixed (non tunable) blocks in the control system
- A structured controller $C(s) = \mathrm{Diag}(C_1(s), \ldots, C_N(s))$ which combines all tunable control elements. Each control element $C_j(s)$ is assumed to be linear time invariant and to have some prescribed structure.

External inputs to the system such as reference signals and disturbances are gathered in w and performance-related outputs such as error signals are gathered in z (see Section 3 for more details).

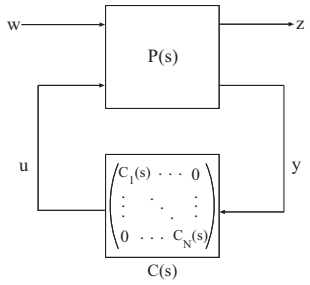

Fig. 1. Standard Form for Structured H_∞ Synthesis

It is well known from Robust Control theory that any block diagram can be rearranged into this Standard Form by isolating the tunable blocks and collapsing the rest of the diagram into $P(s)$. The resulting model has the same structure as the uncertain models used in μ analysis with the uncertainty blocks $\Delta_j(s)$ replaced by the control elements $C_j(s)$ Redheffer [1960], Doyle et al. [1991], Varga and Looye [1999], Varga and Magni [2005]. Figure 1 is also reminiscent of standard H_∞ synthesis Doyle et al. [1989] but differs in one key aspect, namely, the special structure of the controller $C(s)$. Since systematic procedures and automated tools are available to transform any architecture to the Standard Form of Figure 1, we assume that the control architecture is specified in this form.

The next challenge is to describe the tunable control elements. Again we are faced with a wide range of possible structures, from simple gains and PIDs to more complex lead-lag compensators and observer-based controllers. Since our approach is based on optimization, it is natural to use parametric models of such components. For example, a PID can be parameterized by four scalars K_p, K_i, K_d, T_f as

$$C_j(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1}. \tag{1}$$

Similarly, a state-space model with fixed order n can be parameterized by four matrices A, B, C, D of suitable sizes. For convenience we provide a library of basic tunable blocks such as static gains, PIDs, and fixed-order transfer functions. Yet this library leaves out many useful control elements, e.g., it cannot model the lowpass and notch filters

$$\frac{a}{s+a}, \quad \frac{s^2 + 2\zeta_1\omega_0 s + \omega_0^2}{s^2 + 2\zeta_2\omega_0 s + \omega_0^2} \tag{2}$$

because of the couplings between numerator and denominator parameters.

Rather than expanding the library of pre-defined tunable blocks, a more scalable approach is to let users dynamically create tunable structures that suit their needs. To do this, we introduce a basic building block called "real parameter" (`realp`). If a is a real parameter, then any well-posed rational function $R(a)$ can be written as the LFT:

$$R(a) = F_l(M, a \otimes I) \tag{3}$$

where M is a fixed matrix and $a \otimes I := \mathrm{Diag}(a, \ldots, a)$ Bett and Lemmon [1997]. The LFT expression (3) corresponds to the following static block diagram:

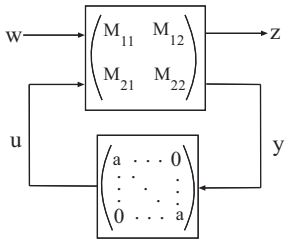

Fig. 2. LFT Model of Rational Expressions in a

More generally, we can model any rational function of the scalar- or matrix-valued parameters $a_1, \ldots, a_M$ by recasting arithmetic operations between these parameters as interconnection operations between LFT models [Zhou and Doyle, 1998, p. 165]. This results in an LFT model similar to Figure 2 where $a \otimes I$ is replaced by

$$\mathrm{Diag}(a_1 \otimes I_{k_1}, \ldots, a_M \otimes I_{k_M}). \tag{4}$$

Since any interconnection of LFT models is an LFT model, and the Standard Form of Figure 1 is itself an LFT model, it is easily seen that if some tunable element $C_j(s)$ is a rational function of the parameters $a_1, \ldots, a_M$, the Standard Form can be rearranged so that $C_j(s)$ is replaced by $a_1 \otimes I_{k_1}, \ldots, a_M \otimes I_{k_M}$ along the diagonal of $C(s)$. In other words, we can absorb the specific structure of $C_j(s)$ into $P(s)$ and keep only the low-level tunable parameters $a_1, \ldots, a_M$ in $C(s)$. The resulting block-diagonal controller $C(s)$ is then a mix of predefined blocks (e.g., PID) and possibly repeated tunable parameters a_j . Note that unlike μ-analysis, "repeated" blocks do not affect the optimization outcome and only incur some negligible overhead.

To illustrate how this comes together in the software tools, first consider the lowpass filter $F(s) = \frac{a}{s+a}$ where a is tunable. This tunable element is specified by the MATLAB commands:

```
a = realp('a',1); % a is initialized to 1
F = tf(a,[1 a]);   % creates a/(s+a)
```

This automatically builds the following LFT parameterization of $F(s)$:

$$F(s) = F_l\left(\begin{pmatrix} 0 & 0 & 1 \\ 1/s & -1/s & 0 \\ 1/s & -1/s & 0 \end{pmatrix}, \begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}\right). \tag{5}$$

Next consider the observer-based controller

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + Bu + L(y - C\hat{x} - Du) \\ &= (A - BK - LC + LDK)\hat{x} + Ly \\ u &= -K\hat{x} \end{aligned} \tag{6}$$

where the gain matrices K, L are tunable. This observer structure is specified as:

```
% For a plant with nx states, nu controls,
% and ny measurements:
K = realp('K',zeros(nu,nx));
L = realp('L',ones(nx,ny));
OBC = ss(A-B*K-L*(C-D*K),L,-K,0);
```

The resulting LFT parameterization `OBC` of the observer structure has three copies of the parameter `K` and two copies of the parameter `L`. Note that in both examples,

it is possible to construct more efficient parameterizations using a single copy of `a`, `K`, `L`.

3. H_∞ FORMULATION OF DESIGN REQUIREMENTS

Now that we have a framework for describing arbitrary control architectures and linear control elements, we turn to the question of recasting the design requirements as H_∞ constraints. At the heart of H_∞ synthesis is the H_∞ norm, which measured the peak input/output gain of a given transfer function:

$$\|H(s)\|_\infty := \max_\omega \overline{\sigma}(H(j\omega)). \tag{7}$$

In the SISO case, this norm is just the peak gain over frequency. In the MIMO case, it measures the peak 2-norm of the frequency response $H(j\omega)$ over frequency.

From classical control, we know that most control design requirements are equivalent to gain constraints on the open- or closed-loop response. For example, good tracking in a particular frequency band is equivalent to small gain from reference to error signals in this band. Similarly, bandwidth and roll-off requirements can be enforced by shaping the open-loop response gain. For example, consider the elementary feedback loop of Figure 3 and suppose we want integral action with bandwidth ω_c. Then $\lambda(s) = \omega_c/s$ is a suitable shape for the open-loop response $L(s) = G(s)C(s)$ as depicted in Figure 4. The zero dB crossover frequency ω_c dictates how fast the system responds and is limited by factors such as stability, delays, actuator bandwidth, and modeling errors.

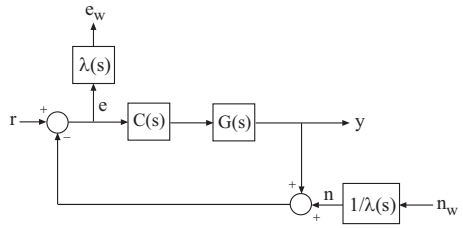

Fig. 3. Loop Shaping Configuration

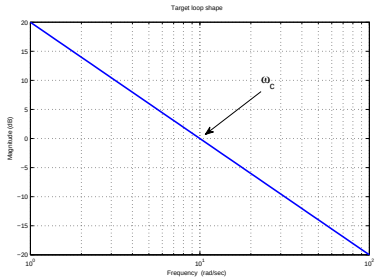

Fig. 4. Target Loop Shape $|\lambda(j\omega)| = \omega_c/\omega$

Once we express requirements in terms of open- or closed-loop gains, there are well-established procedures for deriving the corresponding H_∞ constraints Skogestad and Postlethwaite [1996]. For example, if the closed-loop transfer function $H(s)$ from (r, n_w) to (y, e_w) in Figure 3 satisfies $\|H(s)\|_\infty < 1 + \delta$, then for small δ:

- The open-loop gain $|L(j\omega)|$ approximately matches the target loop shape $|\lambda(j\omega)|$
- The sensitivity functions satisfy $\|S\|_\infty < 1 + \delta$ and $\|T\|_\infty < 1 + \delta$ at all frequencies, which guarantees adequate gain/phase margins and small overshoot.

In other words, the H_∞ constraint $\|H(s)\|_\infty < 1 + \delta$ enforces the desired loop shape along with good stability margins.

In general, this procedure leads to one or more normalized H_∞ constraints $\|H_j(s)\|_\infty < 1$ on frequency-weighted closed-loop transfer functions $H_1(s), \ldots, H_M(s)$. Introducing

$$H(s) := \mathrm{Diag}(H_1(s), \ldots, H_M(s)), \tag{8}$$

we can consolidate all requirements into the single constraint $\|H(s)\|_\infty < 1$. Consequently, the tuning of a particular control architecture can be recast as the following program:

Structured H_∞ Program: Tune the free parameters of $C(s) := \mathrm{Diag}(C_1(s), \ldots, C_N(s))$ to enforce closed-loop (internal) stability and $\|H(s)\|_\infty < 1$.

Note that structured H_∞ synthesis can handle constraints on several *independent* transfers $H_1(s), \ldots, H_M(s)$. To see this, observe that for $H_j(s) := F_l(P_j(s), C(s))$, the Standard Form for $H(s)$ looks like

$$H(s) = F_l\left(P(s), \mathrm{Diag}(C(s), \ldots, C(s))\right) \tag{9}$$

where $P(s)$ is some rearrangement of the input and output channels of $\mathrm{Diag}(P_1(s), \ldots, P_M(s))$. So constraining two or more closed-loop transfer functions $H_j(s)$ leads to repeating the controller $C(s)$ multiple times in the Standard Form. The resulting block-diagonal controller structure is beyond the scope of standard H_∞ algorithms but poses no problem in our framework since this merely amounts to repeating the tunable blocks along the diagonal (see Section 2 for a discussion of repeated blocks). This is an important difference with traditional H_∞ synthesis where all requirements must be expressed in terms of a *single* closed-loop transfer function, and this additional capability greatly simplifies the H_∞ formulation as illustrated in Section 6.

4. NONSMOOTH H_∞ SOLVERS

The Structured H_∞ Program of Section 3 can be solved using specialized nonsmooth optimization techniques. This section gives a high-level overview of the structured H_∞ solver at the heart of our software tools. The full details can be found in Apkarian and Noll [2006a, 2007] and a variety problem studies as well as applications can be found in Apkarian [2010]. Note that related nonsmooth techniques and software for control applications are discussed in Burke et al. [2006], Michiels and Niculescu [2007] and references therein.

Structured H_∞ synthesis requires solving semi-infinite, nonconvex, and nonsmooth programs of the form

$$\begin{aligned} &\underset{C(s)}{\text{minimize}} \, \|F_l(P(s), C(s))\|_\infty \iff \\ &\underset{x\in\mathbb{R}^k}{\text{minimize}} \, \max_{\omega\in[0,\,\infty]} \, \overline{\sigma}\left(F_l(\widehat{P}(j\omega), x)\right), \end{aligned} \tag{10}$$

where $C(s)$ is the structured controller of Section 2 and the vector x gathers all low-level tunable parameters in $C(s)$. While this is a challenging mathematical programming problem, it is important to note that the right-hand-side function in (10) is the composition of the convex but nonsmooth function $\max_\omega \circ \overline{\sigma}(.)$ with the nonconvex but differentiable mapping $x \to F_l(\widehat{P}(j\omega), x)$. Such composite functions are Clarke regular Clarke [1983], which means that a complete description of the Clarke subdifferential is accessible. To simplify the discussion, rewrite program (10) as

$$\underset{x\in\mathbb{R}^k}{\text{minimize}}\, f_\infty(x) := \max_{\omega\in[0,\,\infty]} f(\omega, x)\,. \tag{11}$$

Clarke regularity ensures that critical points x^* (typically local minima) are characterized by the condition $0 \in \partial f(x)$.

To solve (11), we construct a tangent model around the current iterate x that constitutes a "quadratic first-order" local approximation of the original problem. An adequate descent direction h is then computed by solving a convex quadratic program of the form:

$$\begin{aligned}&\underset{h\in\mathbb{R}^k}{\text{minimize}}\, \hat{f}_\infty(x+h) := \\ &\quad \max_{\omega\in\Omega_f} f(\omega, x) - f_\infty(x) + \Phi_\omega^T h + \frac{1}{2}h^T Q h\end{aligned} \tag{12}$$

where Ω_f is some finite set of frequencies and $\Phi_\omega \in \partial f(\omega, x)$ is a subgradient of $f(\omega, x)$. There are many possible implementations of this scheme Apkarian and Noll [2006a], but a minimal requirement is that Ω_f contain the active frequencies ω_a achieving the peak value in (11):

$$f_\infty(x) = f(\omega_a, x)\,.$$

This simple requirement is enough for the algorithm to converge. Yet by adding a few extra well-chosen frequencies, we can often improve the quality of the tangent model (12) and take longer steps at each iteration. In particular, including frequencies bracketing the active peaks can dramatically accelerate convergence Apkarian and Noll [2006b,a]. See Figure 5 for an illustration of this strategy. Note that multiple active frequencies typically arise as the optimization progresses due to the "waterbed effect". The corresponding loss of differentiability can often spell trouble for smooth nonlinear programming algorithms Bertsekas [1995] which may fail to find a joint descent direction for all active peaks.

The resulting algorithm is guaranteed to converge to a critical point (a local minimum in practice) and has proved very effective on a wide range of test problems including large scale systems Simoes et al. [2009]. Arguably, the function $f_\infty(.)$ is non convex and there is no guarantee of reaching the global optimum. But "convex" formulations (e.g., in terms of LMIs Boyd et al. [1994]) often rely on conservative and expensive relaxations or resort to biconvex schemes like BMIs Safonov et al. [1994] and D-K iterations [Zhou and Doyle, 1998, p. 381]. As a result, our approach is very competitive with such formulations and has the advantage of tuning the original control structure and controller parameters.

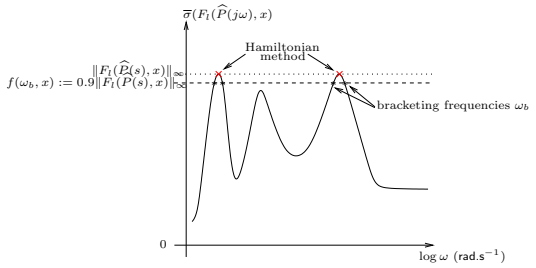

Fig. 5. Frequency selection to build tangent program

5. SOFTWARE TOOLS

This section gives a brief overview of new software tools to formulate and solve structured H_∞ problems for arbitrary linear control architectures. These tools are available in *Robust Control Toolbox* [2011]. Note that some functionality discussed here is only available in the R2011a version.

As discussed in Section 2, the structure of individual tunable elements is described in terms of parameterization. The software provides pre-defined parameterizations for gains, PIDs, fixed-order transfer functions, and state-space models. For example

```
C1=ltiblock.tf('C1',2,3) % 2 zeros, 3 poles
```

parameterizes $C_1(s)$ as a strictly proper third-order transfer function. In addition, users can create their own parameterization using the `realp` building block (real parameter) and standard arithmetic operations, see Section 2 for examples.

The H_∞ formulation of design requirements remains a manual process. This is the same process as for standard H_∞ synthesis, somewhat simplified by the fact that we can independently constrain several closed-loop transfer functions (see Section 3). Tools are available to derive the Standard Form of Figure 1 in both MATLAB and Simulink. Note that the objects used to parameterize $C_1(s), \ldots, C_N(s)$ can be combined with regular LTI objects to build "parametric" models of the relevant closed-loop transfer functions. For example, consider the simple scenario where the requirements for the feedback loop of Figure 3 can be expressed as

$$\|w_S S\|_\infty < 1, \quad \|w_T T\|_\infty < 1 \tag{13}$$

where $S = 1/(1+L)$, $T = L/(1+L)$, and w_S, w_T are suitable frequency-weighting functions. Assuming the tunable block $C(s)$ is a PID controller, the aggregate transfer function $H(s) = \mathrm{Diag}(w_S S, w_T T)$ is obtained by:

```
G = tf([1 2],[1 5 10]);       % plant model
C = ltiblock.pid('C','pid'); % define PID
S = feedback(1,G*C);
T = feedback(G*C,1);
H0 = blkdiag(wS * S, wT * T);
```

The variable `H0` contains a MATLAB representation of the (untuned) Standard Form for $H(s)$ and depends on the tunable PID block `C`.

Once the Standard Form is available, the `hinfstruct` command invokes the solver outlined in Section 4 to

optimize the free parameters of $C_1, \ldots, C_N$. For example, the PID gains in the simple example above are tuned by

```
H = hinfstruct(H0);
```

The output `H` contains the tuned Standard Form of $H(s)$ and you can access the tuned PID controller `C` with

```
H.Blocks.C
```

Note that `hinfstruct` can be configured to automatically run multiple optimizations from randomly generated starting points. This helps mitigate the local nature of the optimizer and increases the likelihood of finding parameter values that meet the design requirements.

6. AIRCRAFT AUTOPILOT EXAMPLE

This section illustrates the use of `hinfstruct` to tune the longitudinal autopilot for a supersonic transport aircraft flying at Mach 0.7 and altitude 5000 ft. Further details on the model used in this application can be found in Boiffier [1998]. The aircraft model is fairly conventional and given in state-space form as

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du\,, \end{aligned} \tag{14}$$

where the state vector is $x := [\, V \;\; \gamma \;\; \alpha \;\; q \;\; H \,]$ and the state variables are the aerodynamic speed V (m/s), climb angle γ (rad), angle of attack α (rad), pitch rate q (rad/s), and altitude H (m). The elevator deflection δ_m (rad) is used to control the vertical load factor N_z (m/s^2). Two measurements are used for feedback: N_z and the pitch rate q. Typical of such models, the open-loop dynamics include the α oscillation with frequency and damping ratio $\omega_n = 1.7$ (rad/s) and $\xi = 0.33$, the phugoid mode $\omega_n = 0.64$ (rad/s) and $\xi = 0.06$, and the slow altitude mode, $\lambda = -0.0026$. See the corresponding demo in *Robust Control Toolbox* [2011] for the model data.

The control structure considered here is the conventional longitudinal control law depicted in Figure 6. The autopilot is comprised of four tunable elements:

- PI controller with proportional and integral gains K_p and K_i
- Static feedback gain K_q on the pitch rate q. This gain is used to improve damping of the responses
- Feedforward gain K_f, which helps achieve better performance and reduce the burden on the feedback controller
- Second-order roll-off filter

$$F_{ro}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}. \tag{15}$$

Note that pure integral action in the PID controller is not possible because this would cancel the zero at $s = 0$ in the N_z channel of the aircraft model $G(s)$. Instead we use a pseudo-integrator $1/(s + 0.001)$. Here steady-state errors are of no concern here since only the transient response to the acceleration command N_{zc} matters.

There are three main design requirements. **Setpoint tracking** is expressed as a model following objective and involves minimizing the tracking error e between N_z and the output of the reference model

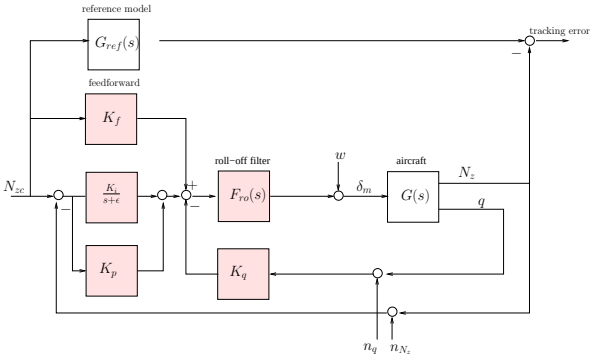

Fig. 6. Aircraft Autopilot

$$G_{ref}(s) := \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

with $\zeta = 0.7$ and $\omega_n = 1.7$. This reference model captures the desired time response characteristics by keeping the natural frequency of the α-oscillation mode and increasing its damping to a suitable value. We formulate this requirement as $\|W_1(s)T_1(s)\|_\infty < 1$ where $T_1(s)$ is the closed-loop transfer function from N_{zc} to e and

$$W_1(s) := 15 \left(\frac{s}{s + 0.05} \, \frac{5}{s + 5} \right)^2 \tag{16}$$

is a bandpass filter emphasizing the frequency band where good tracking is desired

High-frequency roll-off is required to attenuate noise, increase robustness to unmodeled dynamics, and prevent high-frequency control activity that would saturate the actuator physical limits.This is expressed as $\|W_2(s)T_2(s)\|_\infty < 1$ where $T_2(s)$ is the closed-loop transfer from the noise inputs n_{N_z}, n_q to the control signal δ_m, and

$$W_2(s) := \frac{s}{s + 8} \, \frac{(1/8^2)s^2 + (\sqrt{2}/8)s + 1}{(1/800^2)s^2 + (\sqrt{2}/800)s + 1} \tag{17}$$

is a high-pass filter chosen to enforce second-order roll-off past 8 rad/s.

Stability margins can be viewed as imposing some minimum distance between the open-loop response and the critical point, or equivalently an upper bound on the gain of the sensitivity function $T_3(s)$ from w to δ_m (see Section 3). Here we impose a minimum distance of 0.8, or equivalently $\|W_3(s)T_3(s)\|_\infty < 1$ with $W_3(s) = 0.8$. In terms of classical gain and phase margins, this guarantees gain margins of at least -5.10dB and 13.97dB and phase margins of at least $\pm 47 = \pm \arccos(\frac{1^2+1^2-0.8^2}{2\times 1\times 1})$ degrees.

Accordingly, we need to tune the autopilot parameters $K_p, K_i, K_q, K_f, \zeta, \omega_n$ to enforce closed-loop stability and $\|H(s)\|_\infty < 1$ where

$$H := \mathrm{Diag}(W_1T_1, W_2T_2, W_3T_3). \tag{18}$$

Each closed-loop transfer function $T_j(s)$ can be written in Standard Form as $T_j = F_l(P_j, C)$ where $C(s) := \mathrm{Diag}(K_p, K_i, K_q, K_f, F_{ro}(s))$. Starting from a Simulink model of the block diagram in Figure 6, we use the command `linlft` to compute the plant models P_1, P_2, P_3. Next we specify the tunable components using the `realp` command and form the structured controller $C(s)$:

```
Ki = realp('Ki',0); Kp = realp('Kp',0);
Kq = realp('Kq',0); Kf = realp('Kf',0);
wn = realp('wn',3); zeta = realp('zeta',0.8);
Fro = tf(wn^2,[1 2*zeta*wn wn^2]);
C = blkdiag(Kp,Ki,Kq,Kf,Fro);
```

Note that most gains are initialized to zero (open loop). Finally, we build a (parametric) model `H0` of the transfer function $H(s)$ in (18) using:

```
H0=blkdiag(W1*lft(P1,C),W2*lft(P2,C),W3*lft(P3,C))
```

We are now ready to tune the parameters with `hinfstruct`:

```
[H,gam] = hinfstruct(H0);
Final: Peak gain = 1.06, Iterations = 57
```

This returns the tuned $H(s)$ along with the best achieved H_∞ norm `gam = 1.06`. Starting from an open-loop configuration with zero gains, this optimization runs in 4 seconds on a Desktop PC with a 2.4GHz Intel Core2 Quad CPU and 6.00Gb of RAM. The tuned values of the controller parameters are as follows:

$$K_p = -0.0048, \quad K_i = -0.022, \quad K_q = -0.29,$$
$$K_f = -0.033, \quad \zeta = 0.79, \quad \omega_n = 6.46.$$

To validate the design, Figure 7 compares the actual response N_z to a step command N_{zc} with the response of the reference model G_{ref} (dashed). The plot on the right shows the overall deflection δ_m (solid) and the respective contributions of the feedforward and feedback paths. Finally, Figure 8 shows the open-loop response measured at w and the corresponding stability margins (18 dB gain margin and 90 degrees phase margin). Note that the roll-off constraint is inactive and could be tightened.

REFERENCES

P. Apkarian. Internet pages. `http://pierre.apkarian.free.fr`, 2010.

P. Apkarian and D. Noll. Nonsmooth H_∞ synthesis. *IEEE Trans. Aut. Control*, 51(1):71–86, 2006a.

P. Apkarian and D. Noll. Nonsmooth optimization for multidisk H_∞ synthesis. *European J. of Control*, 12(3):229–244, 2006b.

P. Apkarian and D. Noll. Nonsmooth optimization for multiband frequency domain control design. *Automatica*, 43(4):724–731, April 2007.

P. Apkarian, V. Bompart, and D. Noll. Nonsmooth structured control design with application to PID loop-shaping of a process. *Int. J. Robust and Nonlinear Control*, 17(14):1320–1342, 2007.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, USA, Belmont, Mass., 1995.

Christopher J. Bett and Michael Lemmon. On linear fractional representations of multidimensional rational matrix functions. Technical report, Eindhoven, University of Technology, 1997.

J. Boiffier. *The Dynamics of Flight, the Equations*. John Wiley & Sons, New York, 1998.

V. Bompart, P. Apkarian, and D. Noll. Control design in the time- and frequency-domain using nonsmooth techniques. *Syst. Control Letters*, 57(3):271–282, 2008.

S. Boyd, L. ElGhaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in Systems and Control Theory*, volume 15 of *SIAM Studies in Applied Mathematics*. SIAM, Philadelphia, 1994.

J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton. Stabilization via nonsmooth, nonconvex optimization. *IEEE Trans. Aut. Control*, 51(11):1760–1769, November 2006.

F. H. Clarke. *Optimization and Nonsmooth Analysis*. Canadian Math. Soc. Series. John Wiley & Sons, New York, 1983.

J. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis. State-space solutions to standard H_2 and H_∞ control problems. *IEEE Trans. Aut. Control*, AC-34(8):831–847, August 1989.

J. Doyle, A. Packard, and K. Zhou. Review of LFT's, LMI's and μ. In *Proc. IEEE Conf. on Decision and Control*, volume 2, pages 1227–1232, Brighton, December 1991.

D. McFarlane and K. Glover. A loop shaping design procedure using H_∞ synthesis. *IEEE Trans. Aut. Control*, 37(6):759–769, 1992.

W. Michiels and S. I. Niculescu. *Stability and stabilization of time-delay systems. An eigenvalue based approach*, volume 12 of *Advances in Design and Control*. SIAM Publications, 2007.

R. M. Redheffer. On a certain linear fractional transformation. *J. Math. and Phys.*, 39:269–286, 1960.

Robust Control Toolbox. The MathWorks Inc., Natick, MA, 2011.

M. G. Safonov, K. C. Goh, and J. H. Ly. Control System Synthesis via Bilinear Matrix Inequalities. In *Proc. American Control Conf.*, pages 45–49, 1994.

A. M. Simoes, Diego C. Savelli, P. C. Pellanda, N. Martins, and P. Apkarian. robust design of a tcsc oscillation damping controller in a weak 500-kv interconnection considering multiple power flow scenarios and external disturbances. *IEEE Trans. on Power Systems*, 24(1):226–236, 2009.

S. Skogestad and I. Postlethwaite. *Multivariable feedback design - analysis and design*. Wiley, 1996.

G. Stein and J. C. Doyle. Beyond singular values and loop shapes. *J. Guidance and Control*, 14:5–16, 1991.

A. Varga and G. Looye. Symbolic and numerical software tools for LFT-based low order uncertainty modeling. In *Proc. CACSD'99 Symposium, Cohala*, pages 1–6, 1999.

A. Varga and J.F. Magni. Enhanced LFR-toolbox for Matlab. *Aerospace Science and Technology*, 9(2):173–180, 2005.

K. Zhou and J. Doyle. *Essentials of robust control*. Prentice-Hall International, Inc., 1998.

K. Zhou, J. C. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice Hall, 1996.

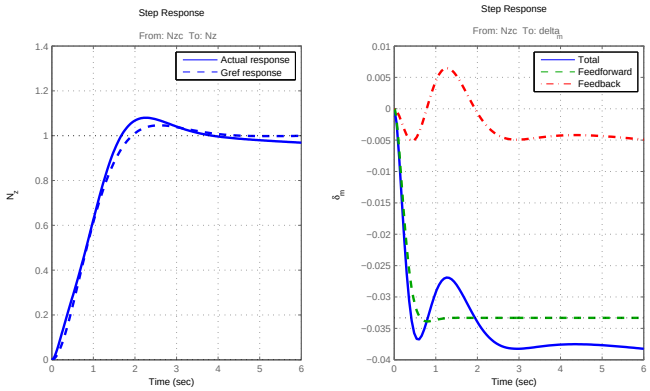

Fig. 7. Response to a Step Command N_{zc}

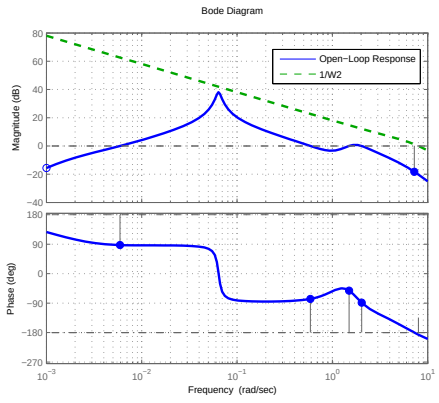

Fig. 8. Open-Loop Response