Tuning Controllers Against Multiple Design Requirements

Pierre Apkarian

***Abstract*— In this paper, we introduce a new technique for tuning arbitrarily structured controllers against multiple control requirements. Control requirements encompass soft or hard design constraints in the usual H_2 and H_∞ design metrics as well as supplemental requirements such as**

- **μ constraints on MIMO margins at specified opening sites,**
- **constraints on the closed-loop dynamics,**
- **loop-shaping constraints at specified loop opening sites,**
- **controller stability enforcement.**

Our algorithmic approach is a non-smooth technique relying on a one-parameter driving function which leads to locally optimal solutions of the design problem. This has been implemented in the MATLAB-based tool `SYSTUNE` which can be regarded as an extension of `HINFSTRUCT` from the Robust Control Toolbox [19]. The new `SYSTUNE` retains the same flexibility and simplicity as `HINFSTRUCT` to specify control structures but is tailored to handle multiple models and requirements.

Two challenging applications are discussed to illustrate the capabilities of the new technique.

I. Introduction

Control engineers have to face with a vast array of design requirements. Requirements might involve gain attenuation in prescribed frequency intervals, settling time constraints, robustness to variations in model physical parameters to cite a few. Both the number and differences in nature of these requirements pose a major impediment to control tuning. Moreover, real-world problems dictate using simple controllers such as PIDs, structured or limited-complexity controllers to ease implementation, validation and possibly on-site re-tuning. Ignoring implementation constraints, a sound approach to handle multiple requirements is certainly the LMI methods developed in [1]–[3]. These techniques however often yield unduly complex controllers and post-processing is mandatory to obtain viable solutions. Note the post-processing stage may include order reduction, truncation of fast dynamics, data re-scaling and remains challenging on its own.

This paper discusses a new approach to controller tuning against multiple possibly conflicting requirements. The general cast is shown in Fig. 1. Given a plant $P(s)$ with Input/Output (I/O) design requirements (w_i, z_i), control inputs u and measured outputs y, a fixed-structured controller $C(s,p)$ is sought to meet a set of constraints:

$$\|T_{w_1\to z_1}(C(s,p))\| \le 1, \ldots, \|T_{w_N\to z_N}(C(s,p))\| \le 1\,. \tag{1}$$

The notation $T_{w\to z}$ is used to represent the closed-loop map from signal w to signal z. The symbol $\|\ \|$ refers to either the H_∞ or to the H_2 norms possibly restricted to prescribed frequency intervals. It is worth noticing constraints stand individually and are not aggregated into a single objective as is the case in classical H_∞ or H_2 syntheses. The controller $C(s,p)$ is assumed decentralized and built from lower granularity structured blocks $C_i(s,p)$, a setting which covers most practical situations. The parameter $p \in \mathbb{R}^n$ regroups all tunable parameters in the controller, i.e., PID gains, filter coefficients, etc. Further details on built-in and customized controller structures are given in [4].

Problem (1) is merely a feasibility problem. A more challenging cast is when both *soft* objectives and *hard* constraints are present. This is then formalized as

$$\begin{array}{ll}\underset{p}{\text{minimize}} & \max\limits_{i=1,\ldots,n_o}\{\|T_{w_i\to z_i}(C(s,p))\|\}\\ \text{subject to} & \|T_{w_j\to z_j}(C(s,p))\| \le 1,\ j=1,\ldots,n_c\,.\end{array} \tag{2}$$

Program (2) emphasizes the fact that constraints $\|T_{w_j\to z_j}(C(s,p))\| \le 1,\ j=1,\ldots,n_c$ have higher priority and parameters p outside the constraint set should be discarded. The proposed resolution technique should then try its best to minimize the $\max$-function objective subject to constraint feasibility.

We also add to the cast in (2), design requirements that do not express in terms of simple norm constraints. This includes

- internal stability in closed-loop,
- constraints on the closed-loop dynamics,
- SISO or MIMO margins based on μ at specified loop opening sites,
- loop-shaping constraints at specified loop opening sites,
- stability constraints on the controller dynamics.

Note while margin constraints, open-loop shaping or enforcing controller stability remain intractable in the LMI setting, they do not necessitate a special treatment with the non-smooth approach. Also, the framework discussed so far is easily extended to problems where the generalized plant $P(s)$ originates from multiple model instances. This is the case when a single controller is sought for a family of operating conditions, or when deviations of some physical coefficients from their nominal values must be accounted for. See Fig. 2. Again the cast in (2) possibly enriched with supplemental constraints is the appropriate formalization of such problems.

Solving such problems cannot be addressed with conventional techniques and computing global solutions is beyond reach. We therefore suggest a specialized non-smooth technique which computes local solutions and is provably convergent to do so. A longstanding experience with an earlier version for unconstrained programs seems to indicate

Pierre Apkarian is with ONERA, 2 Av. Ed. Belin, 31055, Toulouse, France `Pierre.Apkarian@onera.fr`

this technique perform well in practice, both in terms of speed of execution and quality of the solutions [5].

In this paper, we briefly introduce an extension of [4], [6] to problems involving multiple soft and hard design requirements. Our algorithmic approach relies on a $\max$-type one-parameter driving function whose local minima are KKT points of problem (2) for suitable values of the parameter. This approach was originally proposed in [7] for a single hard constraint in the special case of distance to instability.

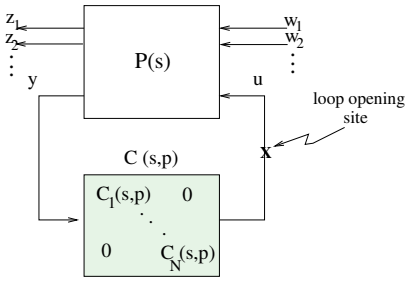

Fig. 1. Synthesis against multiple requirements

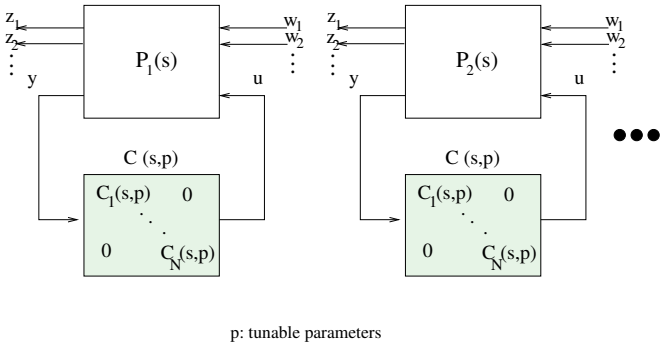

Fig. 2. Synthesis against multiple requirements and models

The paper is organized as follows. Section II introduces central concepts and underlying principles of the non-smooth technique for solving problems with multiple soft and hard requirements. Two non-trivial control applications, reliable flight control, section III, and formation flight guidance, section IV, serve to illustrate the potential of this new technology.

II. Non-smooth algorithm for multiple requirements

Our approach to tuning against multiple requirements uses non-smooth casts of the form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) \le c, \quad c \in \mathbb{R}, \end{array} \tag{3}$$

where both f and g are $\max$-functions

$$f(x) := \max_{i=1,\ldots,n_f} f_i(x),\; g(x) := \max_{i=1,\ldots,n_g} g_j(x), \tag{4}$$

and $x := p \in \mathbb{R}^n$ gathers all tunable controller parameters.

Individual branches f_i and g_j can express a vast array of closed- or open-loop requirements including

- H_∞-norm requirements possibly limited to specific frequency bands,
- H_2-norm requirements or variance constraints,
- Loop-shaping specifications at prescribed loop-opening sites,
- Constraints on closed-loop dynamics,
- Stability constraints on controller dynamics,
- MIMO or SISO stability margins at specified loop-opening sites.

Using shifting if necessary, we will assume throughout all functions f_i and g_j take on positive values. It has been shown in [7] that solving (3) can be based on a one-parameter family of programs

$$P_\mu : \underset{x}{\text{minimize}} \quad h_\mu(x) := \max\{f(x), \mu g(x)\}, \tag{5}$$

where μ is chosen so that the solution x_μ to P_μ satisfies the Karush-Kuhn-Tucker conditions for criticality of program (3).

From a practical viewpoint, problem (3) is solved through a sequence of problems P_μ where μ is driven by a bisection scheme. If constraints $g(x) \le c$ are not competing with f, minimizing f is enough and we are done. Leaving aside this trivial case, solving problem (3) relies on decreasing or increasing μ according to constraint feasibility or not since ultimately we should have binding constraints $g(x_\mu) = c$.

1) Initialize lower bound $\underline{\mu} = 0$
2) Find a strictly feasible point $g(x_f) < c$. Stop if infeasible, otherwise go to 3.
3) Initialize upper-bound $\overline{\mu} = f(x_f)/g(x_f)$, and set $\mu = (\overline{\mu} + \underline{\mu})/2$
4) Stop if $|\overline{\mu} - \underline{\mu}| \le \epsilon$, otherwise solve problem P_μ for x_μ
5) If $g(x_\mu) > c$, set $\underline{\mu} = \mu$ otherwise set $\overline{\mu} = \mu$
6) Update $\mu = (\overline{\mu} + \underline{\mu})/2$, and loop over 4

Note a feasible initial point x_f is easily computed based on the program $\underset{x}{\text{minimize}}\, g(x)$ with early termination as soon as $g(x) \le c$. The justification of the upper-bound initialization is as follows. If initialized with x_f strictly feasible, we have for program $P_{\overline{\mu}}$

$$\begin{array}{rcl} \min\limits_x \max\{f(x),\, \frac{f(x_f)}{g(x_f)} g(x)\} & < & f(x_f) \\ & = & \overline{\mu} g(x_f) \\ & < & \overline{\mu} c \end{array} \tag{6}$$

We infer $g(x_{\overline{\mu}}) < c$ for program $P_{\overline{\mu}}$. Hence $\overline{\mu}$ is an upper-bound for μ.

Substantial speed-up is obtained by initializing each new problem P_μ by the solution of the previous subproblem. Given any μ, the driving function h_μ is also a $\max$-function which facilitates computation of jet information such as Clarke gradients. By the convex hull rule, we have

$$\partial h_\mu(x) = \begin{cases} \partial f(x) \text{ if } f(x) > \mu g(x), \\ \mu \partial g(x) \text{ if } \mu g(x) > f(x), \\ \{\alpha \partial f(x) + (1-\alpha)\mu \partial g(x) : \alpha \in [0,1]\} \\ \text{ if } f(x) = \mu g(x)\,. \end{cases}$$

It follows that every problem P_μ can be solved by adapting the baseline technique presented in [6] which is guaranteed to converge to local solutions even for any remote starting points.

Note the outlined approach must not be confused with the progress function approach discussed at length in [8] (see also [9] for applications to control problems) nor to barrier or exact penalty algorithms presented in [10]. Both progress function and barrier algorithms are feasible methods meaning that algorithm iterates should move in the feasible set. Our proposal is an *infeasible* method where iterates are allowed to move across constraint boundary with potentially larger descent steps and therefore better progress at each iteration.

What we have described is a local method. As such it leads to critical points which are local minima in practice. Using multiple restarts is therefore advisable to mitigate the inconvenience of local optimality.

Alternative control-related local optimization techniques and heuristics include the gradient sampling technique described in [11], derivative-free optimization techniques discussed in [12]–[15], particle swarm optimization methods, see [16] and references therein, and also evolutionary computation techniques [17].

III. Reliable flight control

A typical instance where handling multiple models and requirements makes sense is when the system experiences large deviations from normal operation. This is the case in flight control when failures of some control surfaces take place due to unpredicted events.

The application discussed in this section deals with reliable flight control of an aircraft undergoing outages in the elevator and aileron actuators. The flight control system is required to maintain stability and adequate performance in both nominal operation and degraded conditions where some actuators are no longer effective due to control surface impairment. In addition, wind gusts must be alleviated in all conditions including extreme outage situations to maintain aircraft safety. The aircraft control loop is given in Fig. 3.

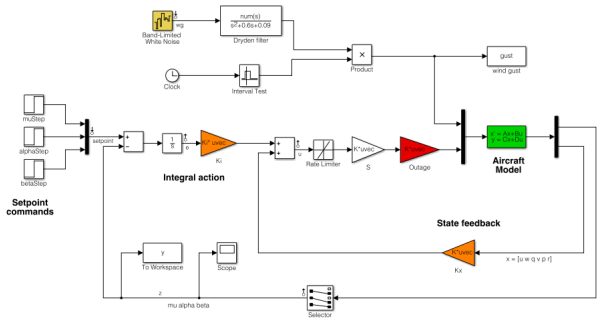

Fig. 3. Synthesis Interconnection

The aircraft is modeled as a classical 6th-order state-space system with state variables given in Table I (units are m/s for velocities and deg/s for angular rates).

The state vector is available for control as well as the flight-path bank angle rate μ (deg/s), the angle of attack α (deg), and the sideslip angle β (deg). Control inputs are the deflections of the right elevator, left elevator, right aileron, left aileron, and rudder. All deflections are in degrees. Elevators are grouped symmetrically to generate the angle of attack. Ailerons are grouped anti-symmetrically to generate roll motion. This leads to 3 control actions as shown in Fig. 3.

TABLE I
Aircraft State Description

u	x-body axis velocity
w	z-body axis velocity
q	pitch rate
v	y-body axis velocity
p	roll rate
r	yaw rate

The controller consists of state-feedback control K_x in the inner loop and MIMO integral action K_i in the outer loop. The matrix-valued integral gain K_i is 3×3 while the state-feedback gain K_x is 3×6. Overall this represents 27 tuning parameters.

In addition to nominal operation, we consider 8 outage occurrences of the control surfaces. This is implemented in the Simulink diagram through the 5×5 diagonal matrix at the aircraft input. Correspondences between failure cases and gain values are clarified in Table II.

TABLE II
Outage Cases Where 0 Stands for Failure

Outage cases	Diagonal of Outage Gain				
nominal mode	1	1	1	1	1
right elevator outage	0	1	1	1	1
left elevator outage	1	0	1	1	1
right aileron outage	1	1	0	1	1
left aileron outage	1	1	1	0	1
left elevator and right aileron outage	1	0	0	1	1
right elevator and right aileron outage	0	1	0	1	1
right elevator and left aileron outage	0	1	1	0	1
left elevator and left aileron outage	1	0	1	0	1

Control requirements are as follows:

- Good tracking performance in μ, α, and β in nominal operating mode with adequate decoupling of the three axes.
- Maintain performance in the presence of wind gust of $5\,m/s$.
- Limit stability and performance degradation in the face of actuator outage.

To express the 1st requirement, we use an LQG-like cost function that penalizes the integrated tracking error e and the control effort u:

$$J = \lim_{T\to\infty} E\left(\frac{1}{T}\int_0^T \|W_e e\|^2 + \|W_u u\|^2 dt\right). \qquad (7)$$

The diagonal weights W_e and W_u are the main tuning knobs for trading responsiveness and control effort and emphasizing some channels over others.

We use the `TuningGoal.WeightedVariance` requirement of `SYSTUNE` to express this cost function, and

use a less stringent performance weight W_e for the outage scenarios. We have $W_e = \text{diag}([20\ 30\ 20]), W_u = I_3$ in the nomical case and $W_e = \text{diag}([8\ 12\ 8]); W_u = I_3$ for outage conditions.

For wind gust alleviation, we limit the variance of the error signal e due to the white noise w_g driving the Dryden wind gust model. Again we formulate a less stringent requirement for the outage scenarios. The variance of e is limited to 0.01 in the nominal case and to 0.03 for the outage scenarios. Also, wind alleviation for all conditions is considered a *hard* constraint meaning it should be met against performance requirement which is a *soft* requirement. This leads to a constrained non-smooth program as discussed in section II. We have in the language of section II, $f(x) := \max_{i=1,\ldots,9} f_i(x)$ and $g(x) := \max_{i=1,\ldots,9} g_i(x)$, where i refers to nominal and outage scenarios. The f_i's are square roots of J in (7) with appropriate weightings W_e and W_u. Similarly, the g_i's are RMS values associated with the transfer functions from white noise w_g to error signal e suitably weighted to reflect variance bounds of 0.01 and 0.03. Decision variables x involve entries in K_i and K_x.

Given the synthesis interconnection in Fig. 3 with appropriately named input and output signals, tracking requirements for each model are specified through the following MATLAB syntax:

```
% Nominal tracking requirement
We = diag([20 30 20]);   Wu = eye(3);
SoftNom=TuningGoal.WeightedVariance('setpoint',
{'e','u'}, blkdiag(We,Wu), []);
SoftNom.Models = 1;     % nominal model

% Tracking requirement for outage conditions
We = diag([8 12 8]);   Wu = eye(3);
SoftOut=TuningGoal.WeightedVariance('setpoint',
{'e','u'}, blkdiag(We,Wu), []);
SoftOut.Models = 2:9;  % outage scenarios
```

Note the `Models` field indicates which model is involved for each requirement. Similarly, prescribing wind gust alleviation is done as follows:

```
% Nominal gust alleviation
HardNom = TuningGoal.Variance('wg','e',0.01);
HardNom.Models = 1;

% Gust alleviation  for outage conditions
HardOut = TuningGoal.Variance('wg','e',0.03);
HardOut.Models = 2:9;
```

Next we use the `slTunable` interface to acquire a closed-loop model for each of the nine flight conditions:

```
for k = 9:-1:1
   outage = OutageCases(k,:) ;
   ST = slTunable('reliableAircraft',{'Ki','Kx'}) ;
   T0(:,:,k) = ST.getIOTransfer({'setpoint';'wg'},
   {'e';'u'}) ;
end
```

The resulting generalized state-space array T_0 contains nine tunable models parameterized by the gains K_i and K_x. We set the wind gust velocity to $5\,m/s$ and initialize tunable gains

```
GustSpeed = 5;
Ki = eye(3);
Kx = zeros(3,6);
```

Controllers are then computed using the non-smooth technique implemented in `SYSTUNE`.

```
[T,fSoft,gHard] = systune(T0,[SoftNom;SoftOut], ...
[HardNom;HardOut]);
```

The object T is the tuned version of T_0 from which tuned valued of K_i and K_x can be retrieved using

```
Ki = getBlockValue(T, 'Ki');  Ki = Ki.d;
Kx = getBlockValue(T, 'Kx');  Kx = Kx.d;
```

For comparison purpose, we have computed a controller for the nominal case alone thus disregarding outage scenarios.

```
[T,fSoft,gHard] = systune(T0(:,:,1),SoftNom, ...
HardNom);
```

Using the simulator in Fig. 3, responses to setpoint commands in μ, α and β with a gust speed of $5\,m/s$ are shown in Fig. 4 for the nominal controller and in Fig. 5 for the reliable controller. As expected, nominal responses are good but they strongly deteriorate when faced with outage cases. The reliable controller maintains better performance in outage operating modes. The optimal performance (square root of LQG cost J in (7)) for the reliable design is only slightly worse than for the nominal tuning (26 vs. 23). For nine 6×4 models of order 11, this required 30.4 seconds cpu time on Mac OS X with 2.66 GHz Intel Core i7 and 8 Go Memory.

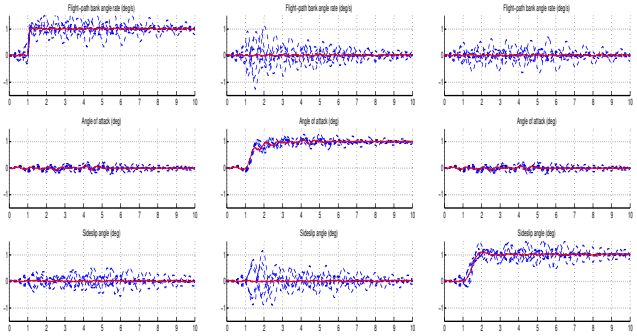

Fig. 4. Time responses with nominal controller with μ, α and β setpoints with wind gust (left to right).

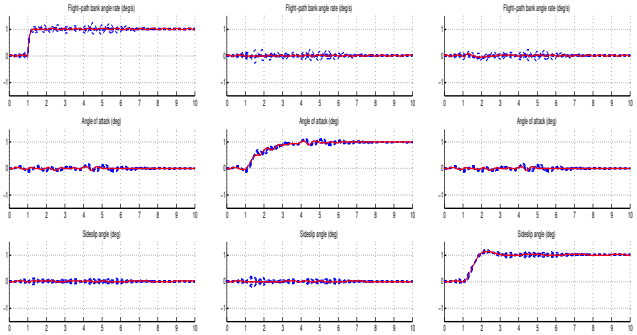

Fig. 5. Time responses with reliable controller with μ, α and β setpoints with wind gust (left to right).

IV. GUIDANCE OF A UAV FORMATION

This example deals with the guidance of a leader-follower formation of 3 UAVs. The leader is UAV 1 and the followers are UAVs 2 and 3. The objective is to maintain prescribed velocities for all UAVs and prescribed distances along the x and y axes between UAVs 1 and 2, and UAVs 2 and 3. Each UAV has only access to partial information which leads to a decentralized control problem. We refer the reader to [18] for more details on this application.

We consider the motion of the formation in the horizontal (x, y) plane. The state of each UAV is described by:

- Leader (UAV 1): Two-entry vector δv_1 of speed errors of leader in x and y directions
- Followers (UAV 2 and UAV 3): Two-entry vectors $\delta d_2, \delta d_3$ of errors in x,y distances with predecessor and two-entry vectors $\delta v_2, \delta v_3$ of speed errors.

Each UAV is controlled by a two-entry thrust vector $\delta u_i, i = 1, 2, 3$ (thrusts in x and y directions). The linearized dynamics for the entire formation are:

$$\dot{X} = AX + BU,$$

where X is the overall state vector obtained by stacking $\delta v_1, \delta d_2, \delta v_2, \delta d_3, \delta v_3$ and U is the overall thrust vector obtained by stacking $\delta u_1, \delta u_2, \delta u_3$. Units are ft, ft/s, and ft/s^2.

The leader sets the target velocity for the entire formation and only worries about achieving this target velocity. Meanwhile, the followers must control both their velocity and their distance to their predecessor. Each follower only knows its own speed and its position relative to its predecessor, which leads to the decentralized control law:

$$\delta u_1 = K_1 \delta v_1, \quad \delta u_2 = K_2 \begin{pmatrix} \delta d_2 \\ \delta v_2 \end{pmatrix}, \quad \delta u_3 = K_2 \begin{pmatrix} \delta d_3 \\ \delta v_3 \end{pmatrix},$$

where K_1 is a 2×2 gain matrix and K_2 is a 2×4 gain matrix. Note that we use the same gain K_2 for both followers since they are interchangeable UAVs. Overall, this amounts to state-feedback control with the block-diagonal structure:

$$U = KX = \begin{pmatrix} K_1 & 0 & 0 \\ 0 & K_2 & 0 \\ 0 & 0 & K_2 \end{pmatrix} X.$$

The closed-loop system is shown in Fig. 6.

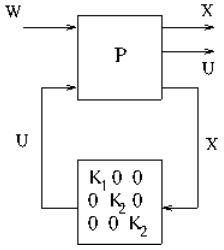

Fig. 6. Closed-loop system

The input vector W is a white noise process which models speed and velocity disturbances. The feedback gains K_1, K_2 must be tuned to ensure:

- Stability of the formation flight
- Appropriate decay of the distances and velocities to their desired values. The reference is zero here since we are working with error dynamics.
- Reasonable thrust levels
- Adequate MIMO gain and phase margins at the plant inputs.

Again we use a quadratic LQG cost of the form

$$J = \lim_{T \to \infty} E \left(\frac{1}{T} \int_0^T (X^T Q X + U^T R U) dt \right)$$

to capture the first three requirements. The value J^2 is just the variance of the weighted output signal

$$Z = \begin{pmatrix} Q^{1/2} X \\ R^{1/2} U \end{pmatrix},$$

for a unit-variance white noise input W (see Fig. 6). We set $Q = I_{10}$ and $R = I_6$ for the initial tuning.

For the last requirement, we specify a minimum of 10 dB and 45 degrees of simultaneous gain and phase margins at the plant inputs. This is formulated as a hard requirement meaning that controllers that do not meet these constraints are discarded by the algorithm. Note such margins are truly MIMO and correspond to simultaneous gain and phase distortions at the plant inputs. Internally, the requirement of 10 dB and 45 degrees is normalized by `SYSTUNE` and formulated as a disk constraint [20]:

$$\rho \| D(s)(I - (I + L)^{-1} L) D(s)^{-1} \|_\infty \leq 1, \qquad (8)$$

with the definition $L := KP_{22}$ and $\rho = \max(\rho_g, \rho_p)$ with

$$\begin{aligned} \rho_g &= \max \left(\left| \tfrac{10^{10/20}-1}{10^{10/20}+1} \right|, \left| \tfrac{10^{-10/20}-1}{10^{-10/20}+1} \right| \right), \\ \rho_p &= \tan(45 * 180/2\pi), \end{aligned}$$

and $D(s)$ is a 6×6 diagonal static D-scale consistent with the diagonal uncertainty structure. Optionally, one could use dynamic D-scales but static ones turned out sufficient in this case.

Summing up, our formulation includes an H_2-norm objective in combination with a μ upper-bound hard constraint. Invoking `SYSTUNE` as described in section III yields the following feedback components:

$$\begin{aligned} K_1 &= \begin{bmatrix} -2.124 & 7.45e{-}08 \\ -4.39e{-}07 & -2.124 \end{bmatrix}, \\ K_2 &= \begin{bmatrix} 1.126 & 3.89e{-}06 & -2.001 & -3.78e{-}06 \\ 5.18e{-}08 & 1.126 & -1.12e{-}06 & -2.001 \end{bmatrix} \end{aligned}$$

Simulations of the UAV formation to initial errors of 100 ft/s in x-speed, 150 ft/s in y-speed, 400 ft in x-distance, and 300 ft in y-distance are displayed in Figs. 7 and 8.

Finally, we check in Fig. 9 phase and gain margins at the plant inputs formulated as the disk constraint in (8). Note the 0 dB boundary materializes 10 dB and 45 degrees gain and phase margins.

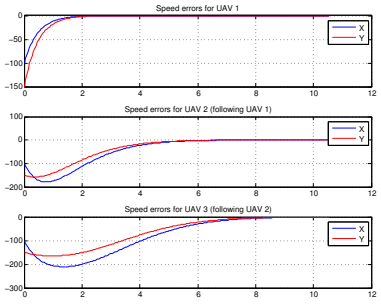

Fig. 7. Velocity errors of leader and followers in (x, y) plane

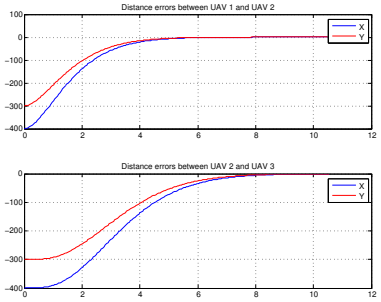

Fig. 8. Distance errors of followers in (x, y) plane

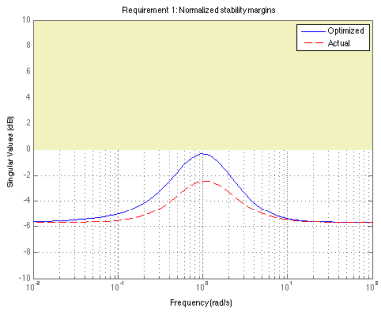

Fig. 9. Margin assessment

CONCLUSION

We have introduced a new non-smooth programming technique for solving realistic complex problems. Its implementation SYSTUNE ranges much far beyond HINFSTRUCT and can handle multiple models as well as a variety of control design requirements. A core ingredient is the formulation of problems involving soft and hard constraints through a driving function whose critical points in the limit are KKT points of the original problem. This can be implemented very efficiently by exploiting basic sub-differential properties of max functions. Our assessment of SYSTUNE indicates that the proposed method is a powerful practical tool which broadens the capabilities of control engineers in solving challenging design problems.

ACKNOWLEDGEMENTS

SYSTUNE and HINFSTRUCT were implemented in collaboration with Pascal Gahinet (MathWorks). Non-smooth concepts and algorithms for control design were developed in the past decade with Dominikus Noll (Maths Institute, Toulouse, France).

REFERENCES

[1] D. D. Peaucelle and D. Arzelier, "Robust multi-objective control toolbox," in *Computer Aided Control System Design Conference*, oct. 2006.

[2] S. Boyd, C. Barratt, and S. Norman, "Linear controller design: Limits of performance via convex optimization," *Proc. IEEE*, vol. 78, no. 3, pp. 529–574, Mar. 1990.

[3] C. Scherer, "Multi-objective control without Youla parameterization," in *Perspectives in robust control*, ser. Lecture Notes in Control and Information Sciences, S. O. Moheimani, Ed. Springer Berlin / Heidelberg, 2001, vol. 268, pp. 311–325.

[4] P. Gahinet and P. Apkarian, "Decentralized and fixed-structure H_∞ control in MATLAB," in *Proc. IEEE Conf. on Decision and Control*, dec. 2011, pp. 8205 –8210.

[5] P. Apkarian, "Internet pages," http://pierre.apkarian.free.fr, 2010.

[6] P. Apkarian and D. Noll, "Nonsmooth H_∞ synthesis," *IEEE Trans. Aut. Control*, vol. 51, no. 1, pp. 71–86, 2006.

[7] ——, "Nonsmooth optimization for multiband frequency domain control design," *Automatica*, vol. 43, no. 4, pp. 724–731, April 2007.

[8] E. Polak, *Optimization : Algorithms and Consistent Approximations*. Applied Mathematical Sciences, 1997.

[9] A. Simoes, P. Apkarian, and D. Noll, "A nonsmooth progress function for frequency shaping control design," *IET Control Theory & Applications*, vol. 2, no. 4, pp. 323–336, April 2008.

[10] D. P. Bertsekas, *Nonlinear Programming*. Belmont, Mass.: Athena Scientific, USA, 1995.

[11] J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton, "Stabilization via nonsmooth, nonconvex optimization," *IEEE Trans. Aut. Control*, vol. 51, no. 11, pp. 1760–1769, Nov. 2006.

[12] E. Simon, "Optimal static output feedback design through direct search," in *Proc. IEEE Conf. on Decision and Control*, 2011, pp. 296–301.

[13] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*, ser. MPS-SIAM Series on Optimization. Philadelphia: SIAM, 2008.

[14] C. Audet and J. E. Dennis Jr., "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[15] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: new perspectives on some classical and modern methods," *SIAM Review*, vol. 45, no. 3, pp. 385–482, 2003.

[16] A. Oi, C. Nakazawa, T. Matsui, H. Fujiwara, K. Matsumoto, H. Nishida, J. Ando, and M. Kawaura, "Development of PSO-based PID tuning method," in *International Conference on Control, Automation and Systems*, oct. 2008, pp. 1917 –1920.

[17] J. Lieslehto, "PID controller tuning using evolutionary programming," in *American Control Conference*, vol. 4, 2001, pp. 2828 –2833 vol.4.

[18] J. Lavaei, A. Momeni, and A. Aghdam, "A model predictive decentralized control scheme with reduced communication requirement for spacecraft formation," *IEEE Trans. on Control System Technology*, vol. 16, no. 2, pp. 268 –278, march 2008.

[19] Robust Control Toolbox 4.2, "The MathWorks Inc. Natick, MA, USA," 2012.

[20] D. Gangsaas, K. Bruce, J. Blight, and U.-L. Ly, "Application of modern synthesis to aircraft control: Three case studies," *IEEE Trans. Aut. Control*, vol. AC-31, no. 11, pp. 995–1014, Nov. 1986.