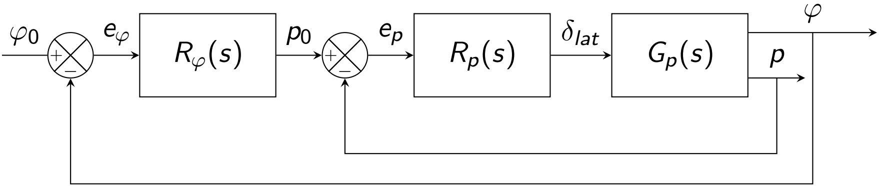
Aerospace Control Systems project

Máté-Erik Moni **Antonio Pucciarelli**
Atefeh Esmaeilzadeh Rostam

Politecnico di Milano

July 7, 2022

Lateral Attitude Control Plant



Blocks:

- **Roll angle**: $R_\varphi(s) = K_\varphi$; **P** controller
- **Roll rate**: $R_p(s) = K_{p,P} + \frac{K_{p,I}}{s} + \frac{K_{p,D}\ s}{1+s\ T_p}$; **PID** controller
- **Lateral dynamics** system: $G_p(s)$;

Lateral Dynamics System

The lateral dynamics system is described by:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{C} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{D} \end{bmatrix} u$$

where:

$$\boldsymbol{y} = \begin{bmatrix} p \\ \varphi \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} v \\ p \\ \varphi \end{bmatrix}, u = \delta_{lat}$$

and:

$$\boldsymbol{A} = \begin{bmatrix} Y_v & Y_p & g \\ L_v & L_p & 0 \\ 0 & 1 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} Y_\delta \\ L_\delta \\ 0 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

POLITECNICO MILANO 1863

Parameters & Uncertainties

All the uncertainties are given in terms of **standard deviation**, σ, referred to a **Gaussian** distribution.

Stability derivatives

- $Y_v = -0.264 \frac{1}{s} (4.837\%)$
- $Y_p = 0 \frac{m}{s \;\; rad} (0\%)$
- $L_v = -7.349 \frac{rad \;\; s}{m} (4.927\%)$
- $L_p = 0 \frac{1}{s} (0\%)$
- $g = 9.806 \frac{m}{s^2} (0\%)$

Control derivatives

- $Y_\delta = 9.568 \frac{m}{s^2} (4.647\%)$
- $L_\delta = 1079.339 \frac{rad}{s^2} (2.672\%)$

POLITECNICO MILANO 1863

Solution Target

- **Nominal** and **uncertain** system analysis
- **Feedback** system assembly
- **Nominal** system tuning: K_φ, $K_{p,P}$, $K_{p,I}$ & $K_{p,D}$
- **Robust** analysis and design
- **Verification** of results
  - **Uncertain model** for the feedback system
  - **Monte-Carlo** simulation
  - **Uncertain model** using real Δ and μ-analysis verification

Design requirements

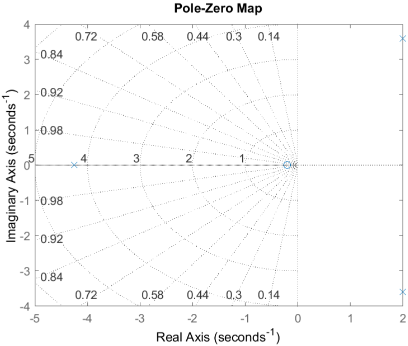
- φ response to φ_0 is a 2nd order response ($\omega_n \geq 10\frac{rad}{s}$ and $\xi \geq 0.9$)
- $|\delta_{lat}| \leq 5$ for a doublet change in φ_0, with a 10° amplitude
- Robust stability for a $\pm 3\sigma$ uncertainty treated as deterministic bounds

POLITECNICO MILANO 1863

lateralDynamics.m

```
% loading quadricopter data
load('ANTRdata.mat');
% stability derivatives -- uncertainty expressed as 3 * sigma with Gauss distribution
Y_v = ureal('Y_v', Y_v, 'percentage', 4.837*3); % 1/s      % uncertainty 4.837%
L_v = ureal('L_v', L_v, 'percentage', 4.927*3); % rad s/m  % uncertainty 4.927%
% control derivatives -- uncertainty expressed as 3 * sigma with Gauss distribution
Y_d = ureal('Y_d', Y_d, 'percentage', 4.647*3); % m/s2    % uncertainty 4.647%
L_d = ureal('L_d', L_d, 'percentage', 2.762*3); % rad/s2  % uncertainty 2.762%
%% lateral dynamics system assembly
A = [Y_v, Y_p, g; L_v, L_p, 0; 0, 1, 0];
B = [Y_d; L_d; 0];
C = [0, 1, 0; 0, 0, 1];
D = [0; 0];
%% conversion from time domain to state space domain
% lateral dynamics state space model assembly
G = ss(A, B, C, D);
% input and output declaration
% input name
G.u = '\delta_{lat}';
% output name
G.y = {'p','\phi'};
% setting up nominal G -> nominal plant model
G_nom = G.nominal;
```

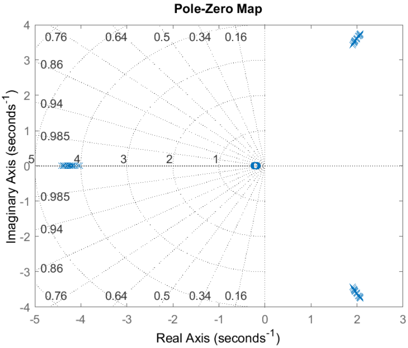
Poles & Zeros – Nominal System



- The lateral dynamics system is **unstable**, there are **2 poles** with $\Re > 0$. In order to **stabilize** the system there is the **need** of controllers.

```
% lateral dynamics matrix assembly
run lateralDynamics;
% ...
% poles/zeros study for
%   the nominal configuration
pzplot(G.Nominal);
```

Poles & Zeros – Uncertain System

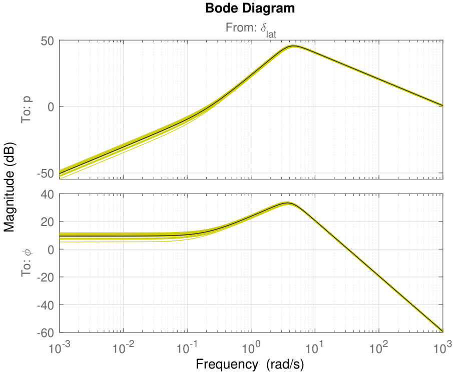


- The $\Re > 0$ poles **still** remain in the **right-hand-side** plane.
- The $\Re < 0$ pole and zero *stay* in the **left-hand-side** plane.

```
% lateral dynamics matrix assembly
run lateralDynamics;
% ...
% poles/zeros study for
%   the uncertain configuration
pzplot(G);
```

POLITECNICO MILANO 1863

Frequency Response Function

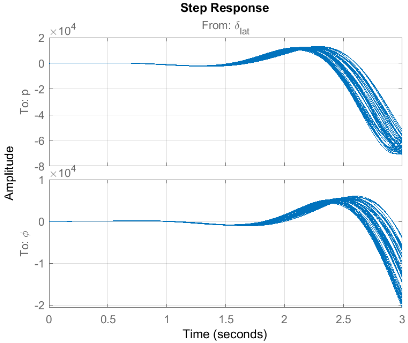


Taking into account the design requirements (6), the frequency response for the uncertain system varies. This variation affects:

- **gain**
- **poles**
- **zeros**

```
% lateral dynamics matrix assembly
run lateralDynamics;
% ...
% uncertain G & nominal G bode plot
bodemag(G, 'y', G.Nominal, 'k', {1e-3,1e+3});
```

Step Response



The step response brings to the **divergence** of the lateral dynamics motion. The uncertainties on data modify the position of **poles** and **zeros**. These changes **affect** the behaviour of the step response.

```
% lateral dynamics matrix assembly
run lateralDynamics;
% ...
% number of samples
n = 50;
% study time interval [s]
interval = 3;
% step response
step(usample(G, n), interval);
```

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` → tunes the system. Program steps:

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` $\rightarrow$ tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` → tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`
- **P** & **PID** block generation; `pid()`

POLITECNICO MILANO 1863

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**. [1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` → tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`
- **P** & **PID** block generation; `pid()`
- **Sensitivity** transfer function **generation**:
  - $e_\varphi = S\ \varphi_0$
  - $\delta_{lat} = Q\ \varphi_0$

POLITECNICO MILANO 1863

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` $\rightarrow$ tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`
- **P** & **PID** block generation; `pid()`
- **Sensitivity** transfer function **generation**:
  - $e_\varphi = S\ \varphi_0$
  - $\delta_{lat} = Q\ \varphi_0$
- **Constraint** generation:
  - 1st **constraint** generation, computation of an alternative **sensitivity weight** function; `sensitivityWeight.m`
  - 2nd **constraint** generation, **trial** and **error** based **control performance** weight function

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` $\rightarrow$ tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`
- **P** & **PID** block generation; `pid()`
- **Sensitivity** transfer function **generation**:
  - $e_\varphi = S\ \varphi_0$
  - $\delta_{lat} = Q\ \varphi_0$
- **Constraint** generation:
  - 1st **constraint** generation, computation of an alternative **sensitivity weight** function; `sensitivityWeight.m`
  - 2nd **constraint** generation, **trial** and **error** based **control performance** weight function
- **Requirements** setup

Overall Steps

Aim: tune the **P** and **PID** blocks applying response **constraints**.
[1, Ch. 2] has been used as reference for the nominal design.

`feedbackDesign.m` → tunes the system. Program steps:

- **Lateral dynamics** block generation; `lateralDynamics.m`
- **P** & **PID** block generation; `pid()`
- **Sensitivity** transfer function **generation**:
  - $e_\varphi = S\ \varphi_0$
  - $\delta_{lat} = Q\ \varphi_0$
- **Constraint** generation:
  - 1st **constraint** generation, computation of an alternative **sensitivity weight** function; `sensitivityWeight.m`
  - 2nd **constraint** generation, **trial** and **error** based **control performance** weight function
- **Requirements** setup
- Control **tuning** → K_φ, $K_{p,P}$, $K_{p,I}$ & $K_{p,D}$

P & PID Assembly

```
%% PID based loop (inner loop) assembly --> input ep = eInner
% PID controller definition -- roll rate
Rp = tunablePID('Rp', 'PID');
Rp.Kp.Value = 1e+2;
Rp.Ki.Value = 1e+2;
Rp.Kd.Value = 1e+2;
Rp.Tf.Value = 1e-3;
% input and output name declaration
Rp.u = 'e_{p}';
Rp.y = '\delta_{lat}';
% error at the inner node --> summation node
eInner = sumblk('e_{p} = p0 - p');
%% P based loop (outer loop) assembly --> input ephi = eOuter
% P controller definition -- roll angle
Rphi = tunablePID('Rphi', 'P');
Rphi.Kp.Value = 1e+2;
% input and output name declaration
Rphi.u = 'e_{\phi}';
Rphi.y = 'p0';
% error at the outer node --> summation node
eOuter = sumblk('e_{\phi} = \phi0 - \phi');
```

1st Constraint Setup

$$\textbf{Target}\text{ response: } \varphi = \frac{\omega_n^2}{s^2 + 2\,\xi\,\omega_n\,s + \omega_n^2} \cdot \varphi_0 \tag{1}$$

There were some *problems* with `systune()` function if plugged in the **performance weight** related to (1)[1].

$$\textbf{Sensitivity weight}\text{ based on [1, Eq. (2.73)]: } W_P = \frac{\frac{s}{M} + \omega}{s + \omega\,A} \tag{2}$$

POLITECNICO MILANO 1863

[1] `WSinv` in `feedbackDesign.m`.

1st Constraint Setup

$$\textbf{Target}\text{ response: } \varphi = \frac{\omega_n^2}{s^2 + 2\,\xi\,\omega_n\,s + \omega_n^2} \cdot \varphi_0 \tag{1}$$

There were some *problems* with `systune()` function if plugged in the **performance weight** related to (1)[1].

$$\textbf{Sensitivity weight}\text{ based on [1, Eq. (2.73)]: } W_P = \frac{\frac{s}{M} + \omega}{s + \omega\ A} \tag{2}$$

(2) has to **track** the sensitivity function **properties** related to (1).
`sensitivityWeight.m` **computes** M, ω & A values using:

[1] `WSinv` in `feedbackDesign.m`.

1st Constraint Setup

$$\textbf{Target response: } \varphi = \frac{\omega_n^2}{s^2 + 2\,\xi\,\omega_n\,s + \omega_n^2} \cdot \varphi_0 \qquad (1)$$

There were some *problems* with `systune()` function if plugged in the **performance weight** related to (1)[1].

$$\textbf{Sensitivity weight} \text{ based on [1, Eq. (2.73)]: } W_P = \frac{\frac{s}{M} + \omega}{s + \omega\ A} \qquad (2)$$

(2) has to **track** the sensitivity function **properties** related to (1). `sensitivityWeight.m` **computes** M, ω & A values using:

- $-80db$ at **steady state**

[1] `WSinv` in `feedbackDesign.m`.

1st Constraint Setup

$$\textbf{Target response: } \varphi = \frac{\omega_n^2}{s^2 + 2\,\xi\,\omega_n\,s + \omega_n^2} \cdot \varphi_0 \tag{1}$$

There were some *problems* with `systune()` function if plugged in the **performance weight** related to (1)[1].

$$\textbf{Sensitivity weight based on [1, Eq. (2.73)]: } W_P = \frac{\frac{s}{M} + \omega}{s + \omega\ A} \tag{2}$$

(2) has to **track** the sensitivity function **properties** related to (1). `sensitivityWeight.m` **computes** M, ω & A values using:

- $-80db$ at **steady state**
- ω_n as reference for the **crossover** frequency

POLITECNICO MILANO 1863

[1] `WSinv` in `feedbackDesign.m`.

1st Constraint Setup

$$\textbf{Target response: } \varphi = \frac{\omega_n^2}{s^2 + 2\,\xi\,\omega_n\,s + \omega_n^2} \cdot \varphi_0 \quad (1)$$

There were some *problems* with `systune()` function if plugged in the **performance weight** related to (1)[1].

$$\textbf{Sensitivity weight} \text{ based on [1, Eq. (2.73)]: } W_P = \frac{\frac{s}{M} + \omega}{s + \omega\ A} \quad (2)$$

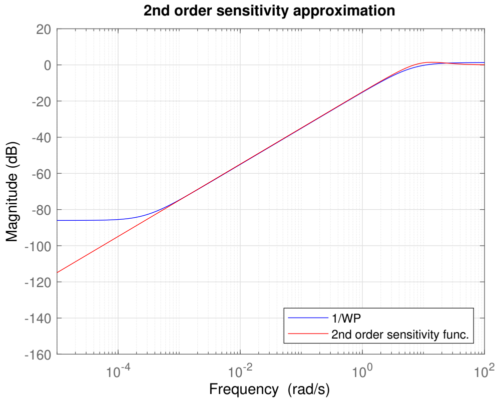
(2) has to **track** the sensitivity function **properties** related to (1). `sensitivityWeight.m` **computes** M, ω & A values using:

- $-80db$ at **steady state**
- ω_n as reference for the **crossover** frequency
- $\xi \geq 0.9 \rightarrow$ [1] suggests $M \lessapprox 1.185$

[1] `WSinv` in `feedbackDesign.m`.

POLITECNICO MILANO 1863

1^{st} Constraint Approximation Result

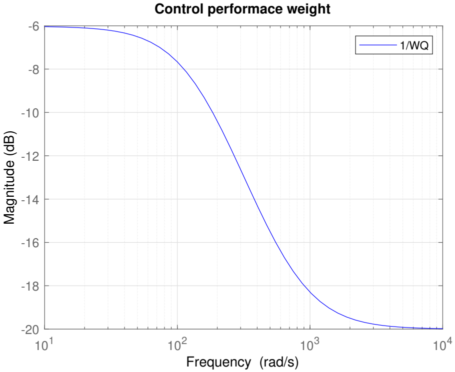


`sensitivityWeight.m` returns:

- $M = 1.16$
- $A = 0.00005$
- $\omega = 5.67$

2^{nd} Constraint Setup

Target response: $|\varphi_0| \to |\delta_{lat}|$.
In this case a **pure trial** and **error** approach has been used for W_Q.
As before the **control performance weight** follows $W_Q = \frac{\frac{s}{M}+\omega}{s+\omega\ A}$.



Trial and **error** results:

- $M = 0.1$
- $A = 0.5$
- $\omega = 1.43e + 3$

Tuning Code

```
% connect() function
% input:
% --- phi0
% outputs:
% --- phi error (eOuter)
% --- delta
T0 = connect(G_nom, Rp, eInner, Rphi, eOuter, {'\phi0'}, {'e_{\phi}', '\delta_{lat}'});
% ...
% PERFORMANCE WEIGHT ASSEMBLY
% ...
% requirement vector assembly
req = [ TuningGoal.WeightedGain('\phi0', 'e_{\phi}', 1/WPinv, 1);
        TuningGoal.WeightedGain('\phi0', '\delta_{lat}', 1/WQinv, 1) ];
%% controllers tuning -- P & PID
opt = systuneOptions('RandomStart', nTest, 'SoftTol', 1e-7, 'Display', 'final');
% tuning control system
[T, J, ~] = systune(T0, req, opt);
% getting values from the tuning results
Rp   = T.blocks.Rp;
Rphi = T.blocks.Rphi;
```

Tuning Results

Stacking approach has been used for the **tuning**. Where:

$$||W_P\ S||_\infty < 1 \tag{3}$$

$$||W_Q\ Q||_\infty < 1 \tag{4}$$

$$\sigma = \sqrt{|W_p\ S|^2 + |W_Q\ Q|^2} < 1 \tag{5}$$

`systune()` has been used for the controllers **tuning**.
`systune()` works on `T0` transfer function that links:

- $\varphi_0 \to e_\varphi$
- $\varphi_0 \to \delta_{lat}$

Tuning Results

Stacking approach has been used for the **tuning**. Where:

$$||W_P\ S||_\infty < 1 \tag{3}$$

$$||W_Q\ Q||_\infty < 1 \tag{4}$$

$$\sigma = \sqrt{|W_p\ S|^2 + |W_Q\ Q|^2} < 1 \tag{5}$$

`systune()` has been used for the controllers **tuning**.
`systune()` works on `T0` transfer function that links:

- $\varphi_0 \rightarrow e_\varphi$
- $\varphi_0 \rightarrow \delta_{lat}$

The **performance requirements** are stored in `req` object.

Tuning Results

Stacking approach has been used for the **tuning**. Where:

$$||W_P\ S||_\infty < 1 \tag{3}$$

$$||W_Q\ Q||_\infty < 1 \tag{4}$$

$$\sigma = \sqrt{|W_p\ S|^2 + |W_Q\ Q|^2} < 1 \tag{5}$$

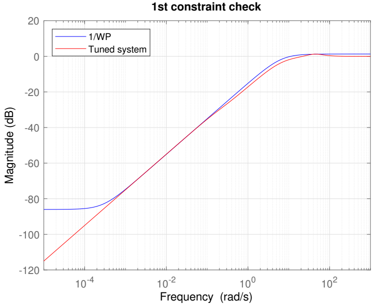
`systune()` has been used for the controllers **tuning**.
`systune()` works on `T0` transfer function that links:

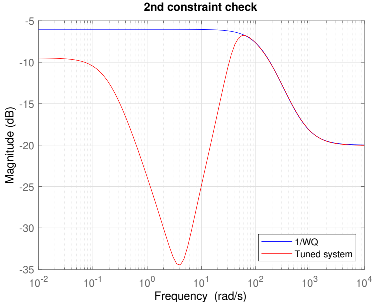
- $\varphi_0 \rightarrow e_\varphi$
- $\varphi_0 \rightarrow \delta_{lat}$

The **performance requirements** are stored in `req` object.
`systune()` computes:

- $K_\varphi = 7.16$
- $K_{p,P} = 0.045$
- $K_{p,I} = 1.22$
- $K_{p,D} = -0.000157$

W_P^{-1} & W_Q^{-1} vs Tuning Results



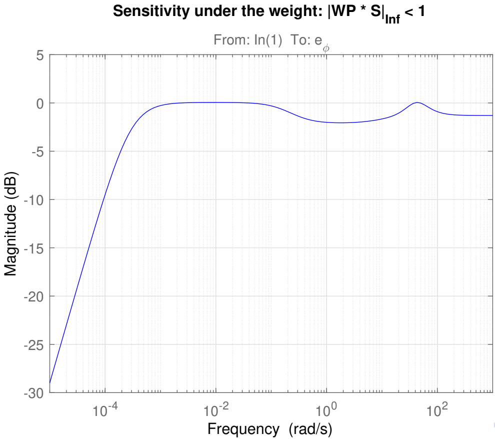


POLITECNICO MILANO 1863

S Upperbound Check

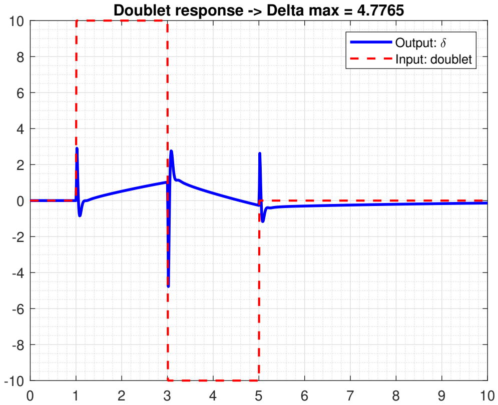
The **performance weight** is expressed as a **boundary** transfer function:

$$||W_P\ S||_\infty < 1 \tag{6}$$



Doublet Response

From W_Q the closed loop system has been tuned with respect to $|\varphi_0| \rightarrow |\delta_{lat}|$.



1. Problem Description
2. $G_p(s)$ Analysis
3. Feedback Design
   - Nominal Dynamic System
   - Uncertain Dynamic System
4. Robust Analysis
5. Verification
6. References

POLITECNICO MILANO 1863

Uncertain Model

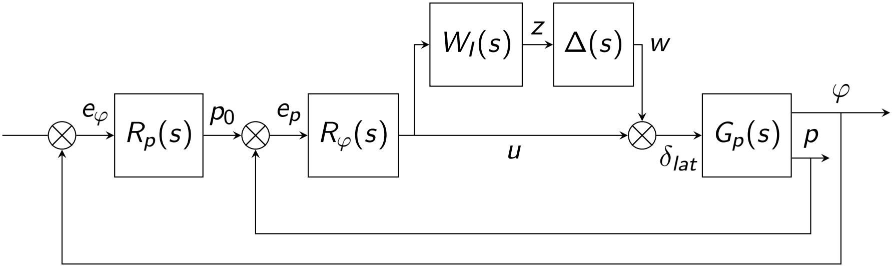
- The **deterministic uncertainties** can be modeled using the `ureal` function of MATLAB in the range of 3σ.

```
% stability derivatives
Y_v = ureal('Y_v',Y_v,'percentage',4.837*3); %1/s
L_v = ureal('L_v',L_v,'percentage',4.927*3); %rad s/m
% control derivatives
Y_d = ureal('Y_d',Y_d,'percentage',4.647*3); %m/s2
L_d = ureal('L_d',L_d,'percentage',2.762*3); %rad/s2
```

- Unlike in the previous case, here we are not going to pull out the nominal plant so the analysis will be carried out with the uncertain one.

POLITECNICO MILANO 1863

$M - \Delta$ Construction



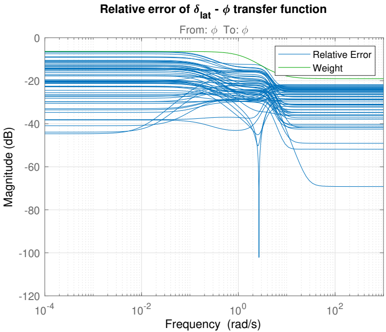
For our robustness analysis we use a system representation in which the uncertain perturbations are *pulled out* into a **complex** uncertain matrix $\Delta(s)$.

Type of representation: Input Multiplicative uncertainty.

- $W_I(s)$: Multiplicative weight
- $\Delta(s)$: Complex perturbation which is $||\Delta(j\omega)|| \leq 1, \forall\omega$

Uncertain Weight

- The **relative error** is $\frac{G_{nom} - G_{unc}}{G_{nom}} = W_I(j\omega) \cdot \Delta(j\omega)$
- Knowing that $\Delta(j\omega) \leq 1$ gives us the upperbound: $\frac{G_{nom} - G_{unc}}{G_{nom}} \leq W_I(j\omega)$
- Consequently $W_I = 0.1114\ s + \frac{0.664}{s+1.387}$




```matlab
% from \delta_lat to \phi -> take the outer loop
G_phi = G(2,1);
% nominal plant
G_phi_nom = getNominal(G_phi);
% sampled uncertain plant
G_phi_smpl = usample(G_phi, 60);
% the weight order is one
[G_phi, Info] = ucover(G_phi_smpl, G_phi_nom, 1);
% nominal and uncertain plant relative error
Rel_e = (G_phi_nom - G_phi_smpl)/G_phi_nom;
% uncertain weight
W_I = minreal(tf(Info.W1));
```

Robust Stability by $M - \Delta$

In order to study the robust stability:

Robust Stability by $M - \Delta$

In order to study the robust stability:

1. The feedback system is represented in the $M - \Delta$ form

Robust Stability by $M - \Delta$

In order to study the robust stability:

1. The feedback system is represented in the $M - \Delta$ form
2. The magnitude of M has to be found

POLITECNICO MILANO 1863

Robust Stability by $M - \Delta$

In order to study the robust stability:

1. The feedback system is represented in the $M - \Delta$ form
2. The magnitude of M has to be found
3. The robust stability needs to be checked

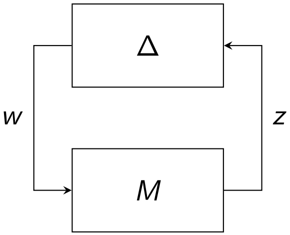
POLITECNICO MILANO 1863

Robust Stability by $M - \Delta$

In order to study the robust stability:

1. The feedback system is represented in the $M - \Delta$ form
2. The magnitude of M has to be found
3. The robust stability needs to be checked

By applying the **Nyquist stability criterion** we assume that $M = W_I F(s)$ and Δ are stable. The Nyquist stability condition then determines RS if and only if the **loop transfer function** M does not encircle $-1, \forall\ \Delta$, [1].



$$\text{RS} \Leftrightarrow |1 + M\Delta| > 0, \forall\omega, \forall|\Delta| < 1$$
$$||M(j\omega)|| \leq 1, \forall\omega, \forall|\Delta| < 1, \forall\omega$$

MATLAB Implementation

```
W_I.u = '\delta_{lat}'; % weight input
W_I.y = 'z'; % weight output
G_n = getNominal(G); % nominal plant
G_n.u = 'u_delta_{lat}'; % input of nominal plant
Sum_u = sumblk('u_delta_{lat} = \delta_{lat} + w'); % multipicative sum
eOuter_new = sumblk('e_{\phi} = - \phi'); % not considering setpoint in sum block
M = connect(Rphi, eInner, Rp, W_I, Sum_u, G_n, eOuter_new,{'w'}, {'z'}); % M TF
M_s = minreal(M); % simplified M TF
get(M_s)
```

Where M is constituted by:

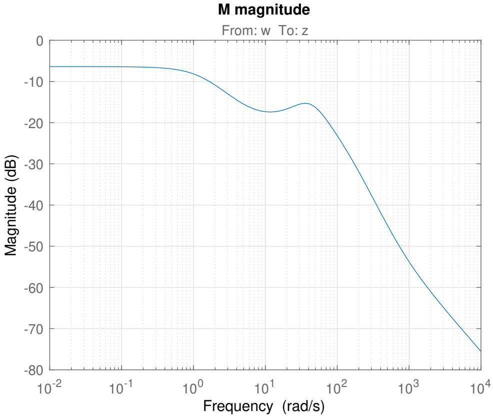
$$\boldsymbol{A} = \begin{bmatrix} -1.387 & 0 & -0.014 & -0.099 & 1.227 & -0.032 \\ 0 & -0.264 & -0.133 & 8.859 & 11.745 & -0.305 \\ 0 & -7.349 & -14.975 & -107.296 & 1324.926 & -34.468 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -7.164 & 0 & 0 \\ 0 & 0 & 203.224 & 1456.077 & 0 & -203.224 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 \\ 9.568 \\ 1079.339 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} 0.509 & 0 & -0.0015 & -0.011 & 0.136 \end{bmatrix}, \boldsymbol{D} = 0$$

POLITECNICO MILANO 1863

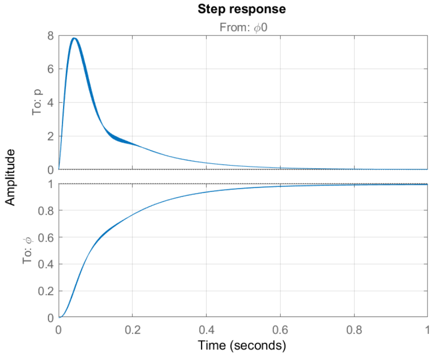
Magnitude of M

By taking a look at the magnitude of M, it can be stated that **robust stability** was achieved.

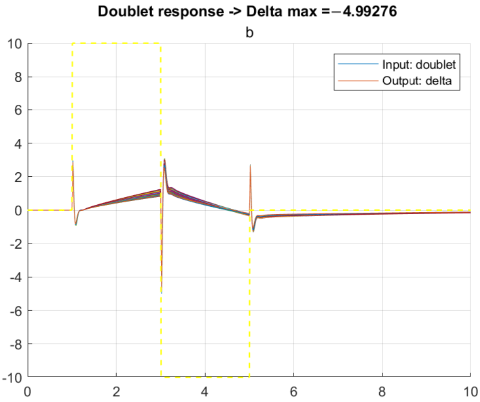


Time Domain Result

- The output reaches its desired value after a short period which confirms that robust stability is satisfied
- **Roll rate** reaches **zero** at steady state while also considering uncertainties



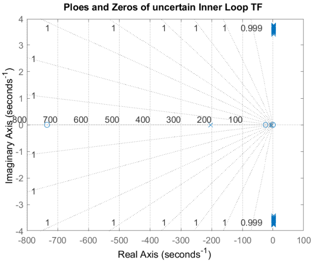
Uncertain Doublet Response

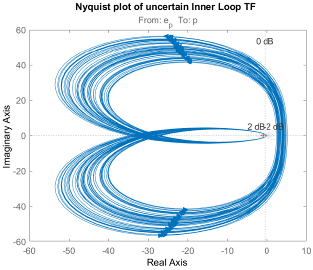


Inner Loop Stability

Nyquist Stability Criterion has been used to ensure the Inner loop stability.

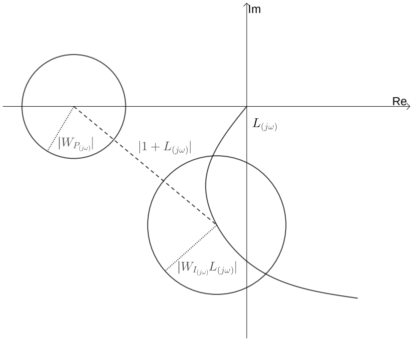
- No. of RHP poles: **2 RHP complex conjugate poles** near zero
- The -1 points needs to be encircled **twice**





Robust Performance – I

Geometric derivation

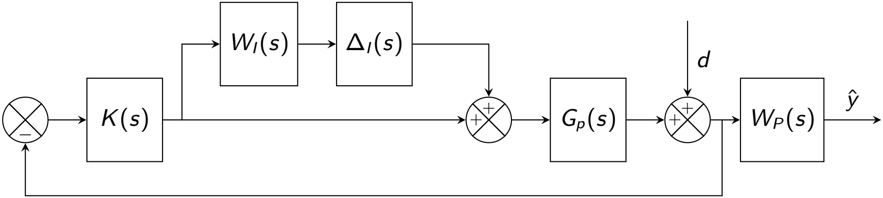


$$\text{RP} \Leftrightarrow |W_P| + |W_I\ L| \leq |1 + L|,\ \forall\ \omega$$

$$\text{RP} \Leftrightarrow \left|\frac{W_P}{1+L}\right| + \left|\frac{W_I\ L}{1+L}\right| \leq 1,\ \forall\ \omega$$

$$\boxed{RP \Leftrightarrow max_\omega(|W_P S| + |W_I F|) \leq 1}$$

Robust Performance – II



For RP we require the performance condition to be satisfied for all possible plants including the worst uncertain case:
Algebraic derivation of RP condition:

$$\text{RP} \Leftrightarrow max_{S_P}|W_P S_P| \leq 1, \forall\omega \tag{7}$$

In which $S_P = \frac{1}{I+L_P} = \frac{1}{1+L+W_I L \Delta_I}$

$$\text{RP} \Leftrightarrow max_{S_P}|W_P S_P| = \frac{|W_P|}{|1+L|} - |W_I L| \tag{8}$$

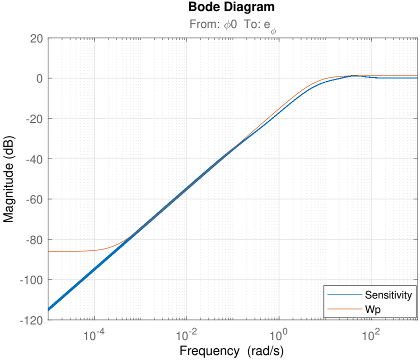
Finally, the following result is obtained:

$$\boxed{\text{RP} \Leftrightarrow max_\omega(|W_P S| + |W_I F|) \leq 1} \tag{9}$$

Robust Performance – III

Considering worst case weighted sensitivity:

$$RP \Leftrightarrow max_{S_p} ||W_p\ S_p|| < 1, \forall\ \omega \tag{10}$$

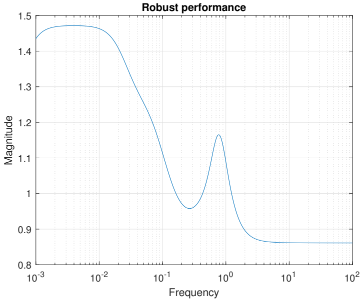


Robust Performance – IV

The RP condition in **SISO** case can be closely approximated by mixed sensitivity $H_{\inf}$ condition:

$$\text{RP} \Leftrightarrow max_{\omega} \sqrt{|W_p \; S_p|^2 + |W_I \; F|^2} < 1 \tag{11}$$

and it can be done in **MIMO** case by structured singular value as we will see later.



We can simply define an index to analyse RP by summing up the NP and RS inequalities

1. Problem Description
2. $G_p(s)$ Analysis
3. Feedback Design
4. Robust Analysis
5. Verification
   - Uncertain Model with Real Δ and μ Analysis
   - Monte-Carlo Simulation
6. References

$M - \Delta$ Construction

- Separate nominal and uncertain parts of the uncertain parameters as we did before, while here δ_is are **real** numbers such that $||\delta_i|| \leq 1$:
  - $Y_v = Y_{v0}\ (1 + r_{Y_v}\delta_1)$;
  - $L_v = L_{v0}\ (1 + r_{L_v}\delta_2)$;
  - $Y_\delta = Y_{\delta 0}\ (1 + r_{Y_\delta}\delta_3)$;
  - $L_\delta = L_{\delta 0}\ (1 + r_{L_\delta}\delta_4)$;

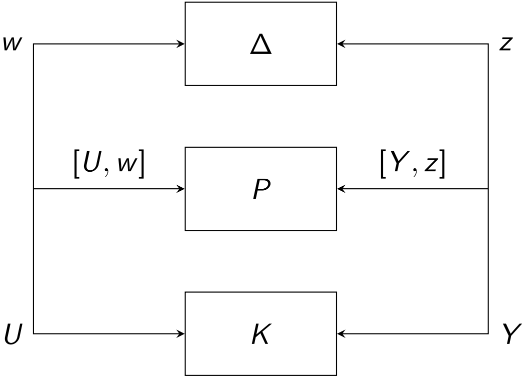
  where $Y_{i0} = \frac{Y_m+Y_M}{2}$ and $r_i = \frac{Y_m-Y_M}{Y_m+Y_M}$
- Update the state space coefficient Matrices:

$$\boldsymbol{A} = \boldsymbol{A}_{nom} + \begin{bmatrix} r_{Y_v} \\ 0 \\ 0 \end{bmatrix} \delta_1 \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ r_{L_v} \\ 0 \end{bmatrix} \delta_2 \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{B} = \boldsymbol{B}_{nom} + \begin{bmatrix} r_{Y_\delta} \\ 0 \\ 0 \end{bmatrix} \delta_3 + \begin{bmatrix} 0 \\ r_{L_\delta} \\ 0 \end{bmatrix} \delta_4$$

POLITECNICO MILANO 1863

μ Analysis – Structured Singular Value



- $\Delta(s)$: $\text{diag}\Delta_i$
- P: Nominal Plant Matrix (5x6)
- K: Controller Matrix

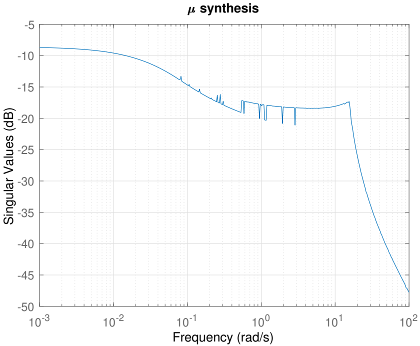
P and K together construct the M which finally gives us the well known $M - \Delta$ model.

μ Analysis – Code

```
% separates the uncertain part from certain part
[P, Delta, BlkStruct] = lftdata(G);
P.u = {'w1','w2','w3','w4','\delta_{lat}'}; % inputs of P
P.y = {'z1','z2','z3','z4','p','\phi'}; % outputs of P
eOuter_new = sumblk('e_{\phi} = - \phi');

% computing M for mu synthesis
M_mu = connect(Rphi, eInner, Rp, P, eOuter_new, ...
      {'w1','w2','w3','w4'}, ...
      {'z1','z2','z3','z4'});
```

Stability Analysis

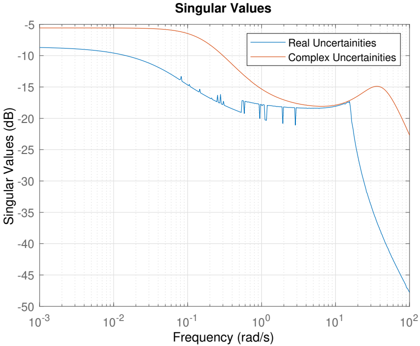


- Assuming having each individual perturbation to be stable
- **Condition for RS in μ synthesis:**
- RS if $\bar{\sigma}(M(j\omega)) < 1, \forall\omega$

MATLAB code:

```
omega  = logspace(-3, 2, 500);
bounds = mussv(frd(M_mu, omega), [-1 0; -1 0; -1 0; -1 0]);
```

Uncertainty Impact on μ Stability



Effect of Real uncertainty:

- Stability margin gets **tighter** when using real uncertainties
- We are less conservative

MATLAB code:

```
bounds     = mussv(frd(M_mu,omega), [-1 0; -1 0; -1 0; -1 0]); % real uncertainties
bounds_com = mussv(frd(M_mu,omega), [ 1 0;  1 0;  1 0;  1 0]); % complex uncertainties
```

POLITECNICO MILANO 1863

Monte-Carlo Simulation

Monte-Carlo simulation is a technique used to study how a model responds to randomly generated inputs.

[2] In `monteCarlo.m`: 500 times.

Monte-Carlo Simulation

Monte-Carlo simulation is a technique used to study how a model responds to randomly generated inputs.

1. Randomly generate `N` inputs.

[2] In `monteCarlo.m`: 500 times.

Monte-Carlo Simulation

Monte-Carlo simulation is a technique used to study how a model responds to randomly generated inputs.

1. Randomly generate `N` inputs.
2. Run a simulation for each of the `N` inputs[2].

[2] In `monteCarlo.m`: 500 times.

Monte-Carlo Simulation

Monte-Carlo simulation is a technique used to study how a model responds to randomly generated inputs.

1. Randomly generate `N` inputs.
2. Run a simulation for each of the `N` inputs[2].
3. Aggregate and assess the outputs from the simulations. Common measures include the mean value of an output, the distribution of output values, and the minimum or maximum output value.

[2] In `monteCarlo.m`: 500 times.

Monte-Carlo Simulation

Monte-Carlo simulation is a technique used to study how a model responds to randomly generated inputs.

1. Randomly generate `N` inputs.
2. Run a simulation for each of the `N` inputs[2].
3. Aggregate and assess the outputs from the simulations. Common measures include the mean value of an output, the distribution of output values, and the minimum or maximum output value.

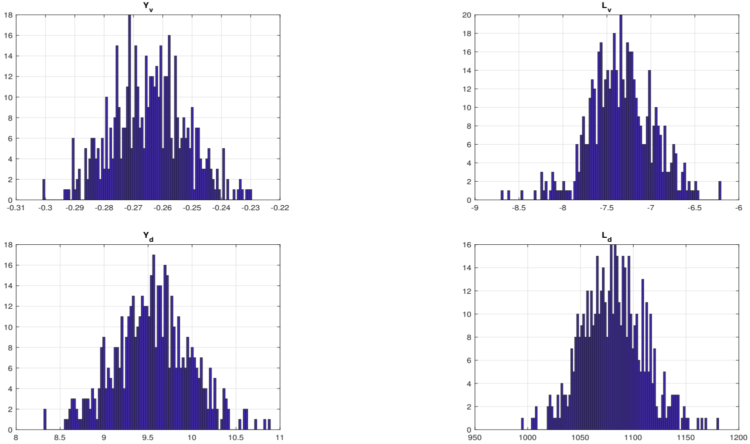
MATLAB code:

```
% covariance
sigma_Yv = ((4.837/100)*abs(Y_v))^2;
sigma_Lv = ((4.927/100)*abs(L_v))^2;
sigma_Yd = ((4.647/100)*abs(Y_d))^2;
sigma_Ld = ((2.762/100)*abs(L_d))^2;
```

```
% gaussian distribution
Y_v_gd = gmdistribution(Y_v, sigma_Yv);
L_v_gd = gmdistribution(L_v, sigma_Lv);
Y_d_gd = gmdistribution(Y_d, sigma_Yd);
L_d_gd = gmdistribution(L_d, sigma_Ld);
```

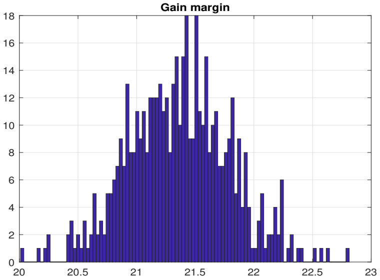
[2] In `monteCarlo.m`: 500 times.

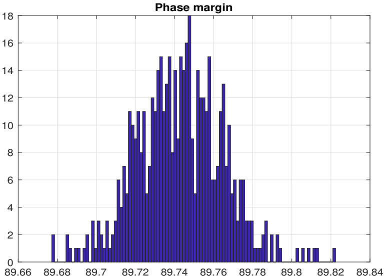
Gaussian Distribution of Uncertain Parameters



Simulation Results for System Margins

Nominal Gain Margin (G_m)	Nominal Phase Margin (P_m)
21.3451	89.7431





1. Problem Description
2. $G_p(s)$ Analysis
3. Feedback Design
4. Robust Analysis
5. Verification
6. References

[1] Sigurd Skogestad and Ian Postlethwaite.
Multivariable feedback control: analysis and design.
Citeseer, 2007.

Thank you!

Máté-Erik Moni 968436
Antonio Pucciarelli 974675
Atefeh Esmaeilzadeh Rostam 969954

POLITECNICO MILANO 1863