Nano-Hexapod - Test Bench

Dehaeze Thomas

June 30, 2021

Contents

This document is dedicated to the experimental study of the nano-hexapod shown in Figure 0.1.

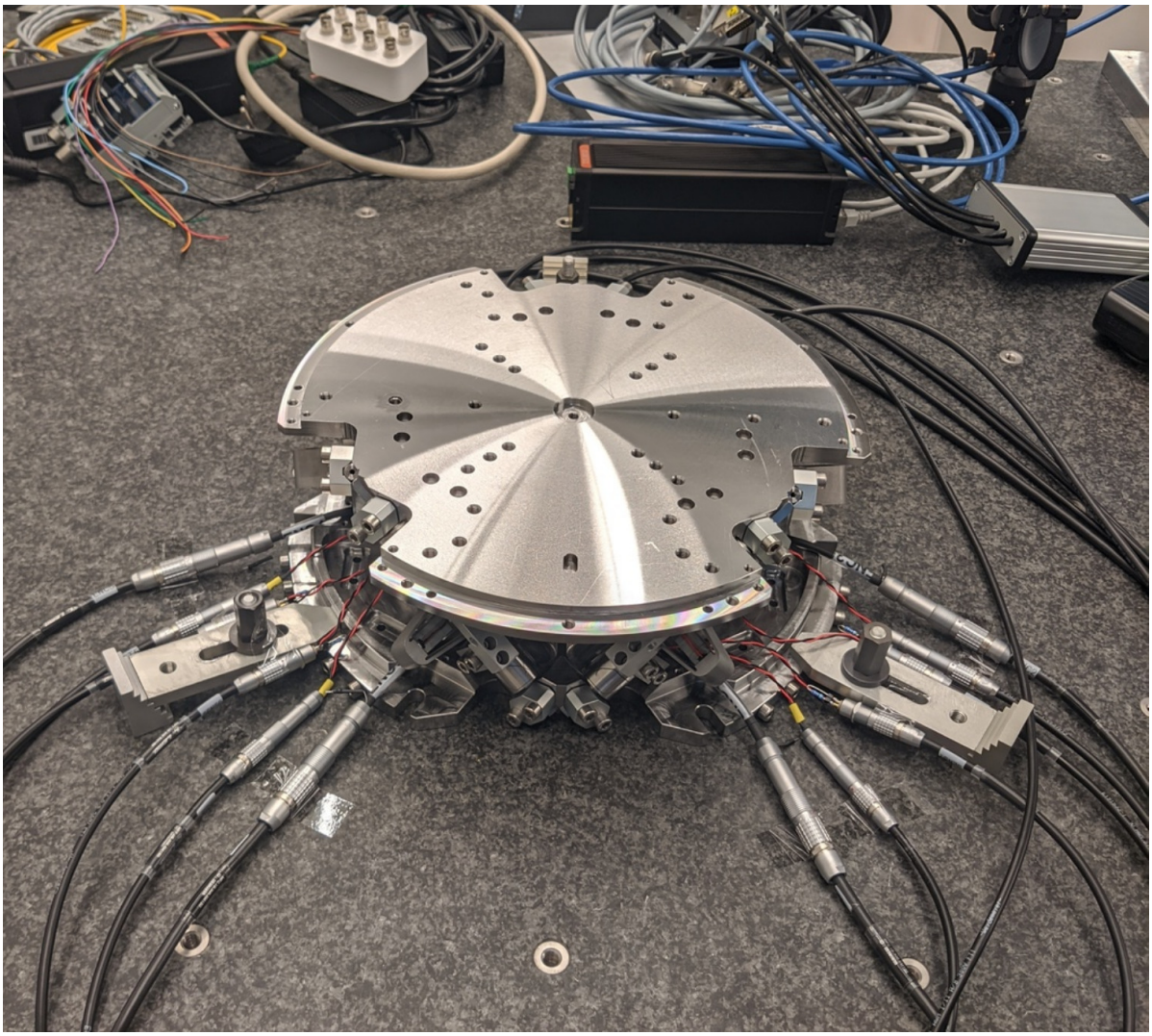

Figure 0.1: Nano-Hexapod

Note

Here are the documentation of the equipment used for this test bench (lots of them are shwon in Figure 0.2):

- Voltage Amplifier: PiezoDrive PD200
- Amplified Piezoelectric Actuator: Cedrat APA300ML
- DAC/ADC: Speedgoat IO313
- Encoder: Renishaw Vionic and used Ruler
- Interferometers: Attocube

In Figure 0.3 is shown a block diagram of the experimental setup. When possible, the notations are consistent with this diagram and summarized in Table 0.1.

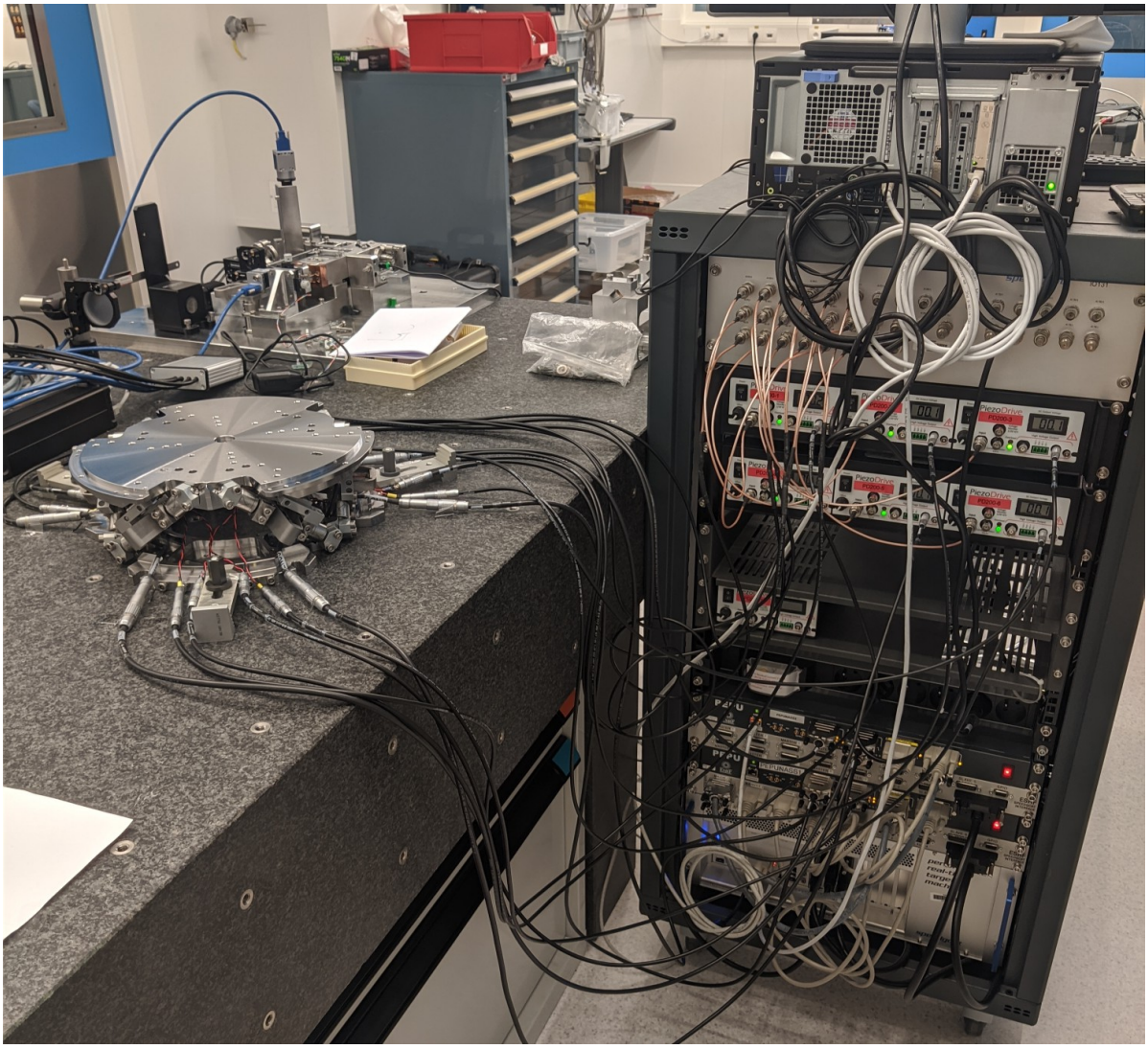

Figure 0.2: Nano-Hexapod and the control electronics

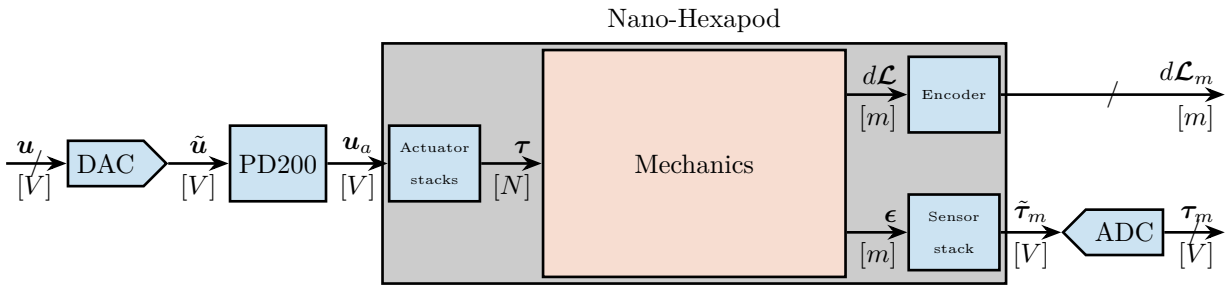

Figure 0.3: Block diagram of the system with named signals

Table 0.1: List of signals

	Unit	Matlab	Vector	Elements
Control Input (wanted DAC voltage)	`[V]`	`u`	$\boldsymbol{u}$	u_i
DAC Output Voltage	`[V]`	`u`	$\tilde{\boldsymbol{u}}$	$\tilde{u}_i$
PD200 Output Voltage	`[V]`	`ua`	$\boldsymbol{u}_a$	$u_{a,i}$
Actuator applied force	`[N]`	`tau`	$\boldsymbol{\tau}$	τ_i
Strut motion	`[m]`	`dL`	$d\boldsymbol{\mathcal{L}}$	$d\mathcal{L}_i$
Encoder measured displacement	`[m]`	`dLm`	$d\boldsymbol{\mathcal{L}}_m$	$d\mathcal{L}_{m,i}$
Force Sensor strain	`[m]`	`epsilon`	$\boldsymbol{\epsilon}$	ϵ_i
Force Sensor Generated Voltage	`[V]`	`taum`	$\tilde{\boldsymbol{\tau}}_m$	$\tilde{\tau}_{m,i}$
Measured Generated Voltage	`[V]`	`taum`	$\boldsymbol{\tau}_m$	$\tau_{m,i}$
Motion of the top platform	`[m,rad]`	`dX`	$d\boldsymbol{\mathcal{X}}$	$d\mathcal{X}_i$
Metrology measured displacement	`[m,rad]`	`dXm`	$d\boldsymbol{\mathcal{X}}_m$	$d\mathcal{X}_{m,i}$

This document is divided in the following sections:

- Section 1: the encoders are fixed to the struts
- Section 2: the encoders are fixed to the plates

1 Encoders fixed to the Struts

1.1 Introduction

In this section, the encoders are fixed to the struts.

It is divided in the following sections:

- Section 1.2: the transfer function matrix from the actuators to the force sensors and to the encoders is experimentally identified.
- Section 1.3: the obtained FRF matrix is compared with the dynamics of the simscape model
- Section 1.4: decentralized Integral Force Feedback (IFF) is applied and its performances are evaluated.
- Section 1.5: a modal analysis of the nano-hexapod is performed

1.2 Identification of the dynamics

1.2.1 Load Measurement Data

Matlab
```matlab
%% Load Identification Data
meas_data_lf = {};

for i = 1:6
    meas_data_lf(i) = {load(sprintf('mat/frf_data_exc_strut_%i_noise_lf.mat', i), 't', 'Va', 'Vs', 'de')};
    meas_data_hf(i) = {load(sprintf('mat/frf_data_exc_strut_%i_noise_hf.mat', i), 't', 'Va', 'Vs', 'de')};
end
```

1.2.2 Spectral Analysis - Setup

Matlab
```matlab
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_data_lf{1}.t(end) - (meas_data_lf{1}.t(1)))/(length(meas_data_lf{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));
```

```
% And we get the frequency vector
[~, f] = tfestimate(meas_data_lf{1}.Va, meas_data_lf{1}.de, win, [], [], 1/Ts);

i_lf = f < 250; % Points for low frequency excitation
i_hf = f > 250; % Points for high frequency excitation
```

1.2.3 DVF Plant

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 1.1).

Matlab
```
%% Coherence
coh_dvf = zeros(length(f), 6, 6);

for i = 1:6
    coh_dvf_lf = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
    coh_dvf_hf = mscohere(meas_data_hf{i}.Va, meas_data_hf{i}.de, win, [], [], 1/Ts);
    coh_dvf(:,:,i) = [coh_dvf_lf(i_lf, :); coh_dvf_hf(i_hf, :)];
end
```

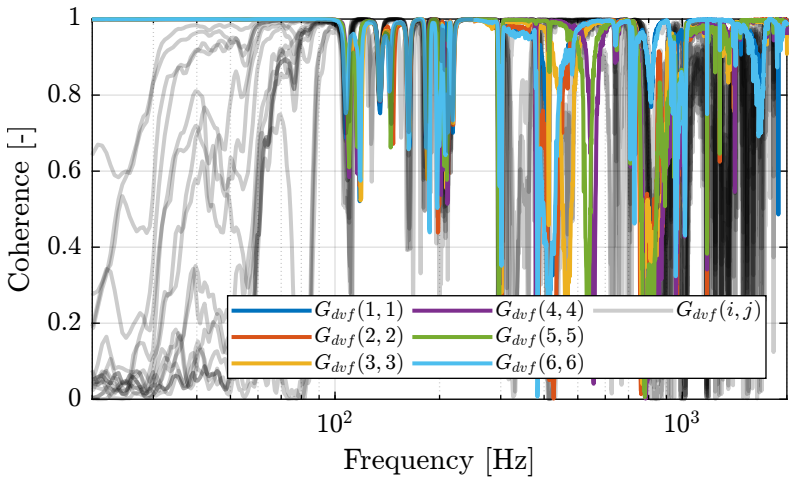

Figure 1.1: Obtained coherence for the DVF plant

Then the 6x6 transfer function matrix is estimated (Figure 1.2).

Matlab
```
%% DVF Plant (transfer function from u to dLm)
G_dvf = zeros(length(f), 6, 6);

for i = 1:6
    G_dvf_lf = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
    G_dvf_hf = tfestimate(meas_data_hf{i}.Va, meas_data_hf{i}.de, win, [], [], 1/Ts);
    G_dvf(:,:,i) = [G_dvf_lf(i_lf, :); G_dvf_hf(i_hf, :)];
end
```

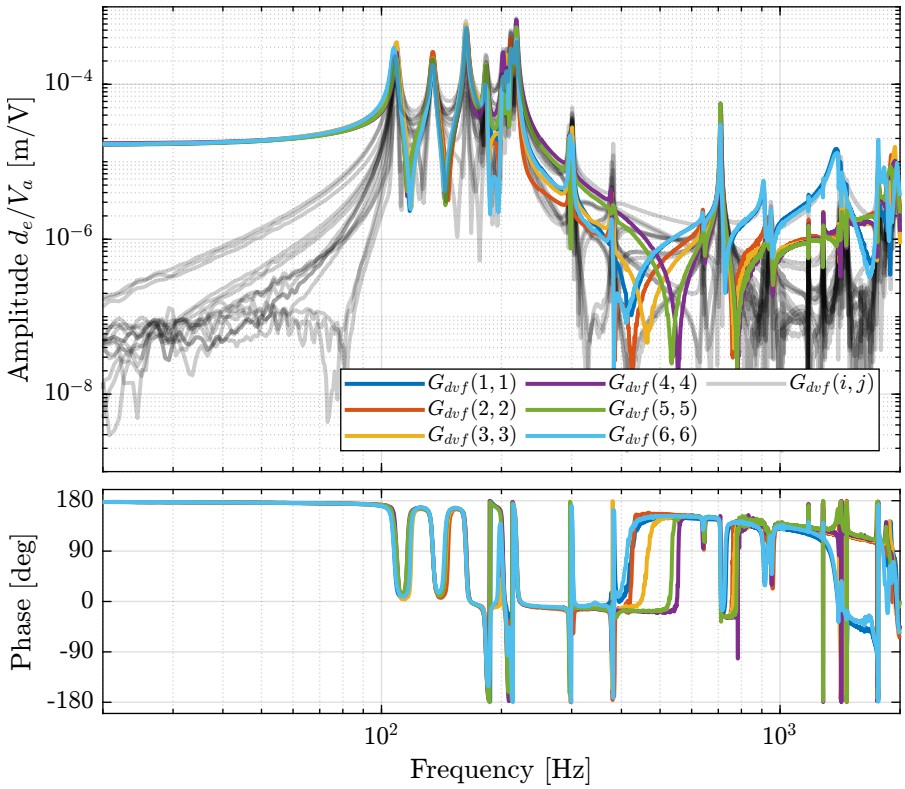

Figure 1.2: Measured FRF for the DVF plant

1.2.4 IFF Plant

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 1.3).

Matlab

```
%% Coherence for the IFF plant
coh_iff = zeros(length(f), 6, 6);

for i = 1:6
    coh_iff_lf = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
    coh_iff_hf = mscohere(meas_data_hf{i}.Va, meas_data_hf{i}.Vs, win, [], [], 1/Ts);
    coh_iff(:,:,i) = [coh_iff_lf(i_lf, :); coh_iff_hf(i_hf, :)];
end
```

Then the 6x6 transfer function matrix is estimated (Figure 1.4).

Matlab

```
%% IFF Plant
G_iff = zeros(length(f), 6, 6);

for i = 1:6
    G_iff_lf = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
    G_iff_hf = tfestimate(meas_data_hf{i}.Va, meas_data_hf{i}.Vs, win, [], [], 1/Ts);
    G_iff(:,:,i) = [G_iff_lf(i_lf, :); G_iff_hf(i_hf, :)];
end
```

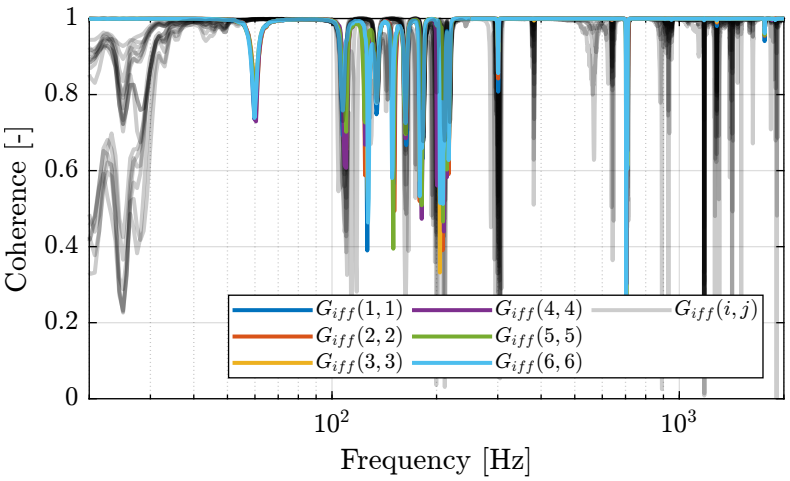

Figure 1.3: Obtained coherence for the IFF plant

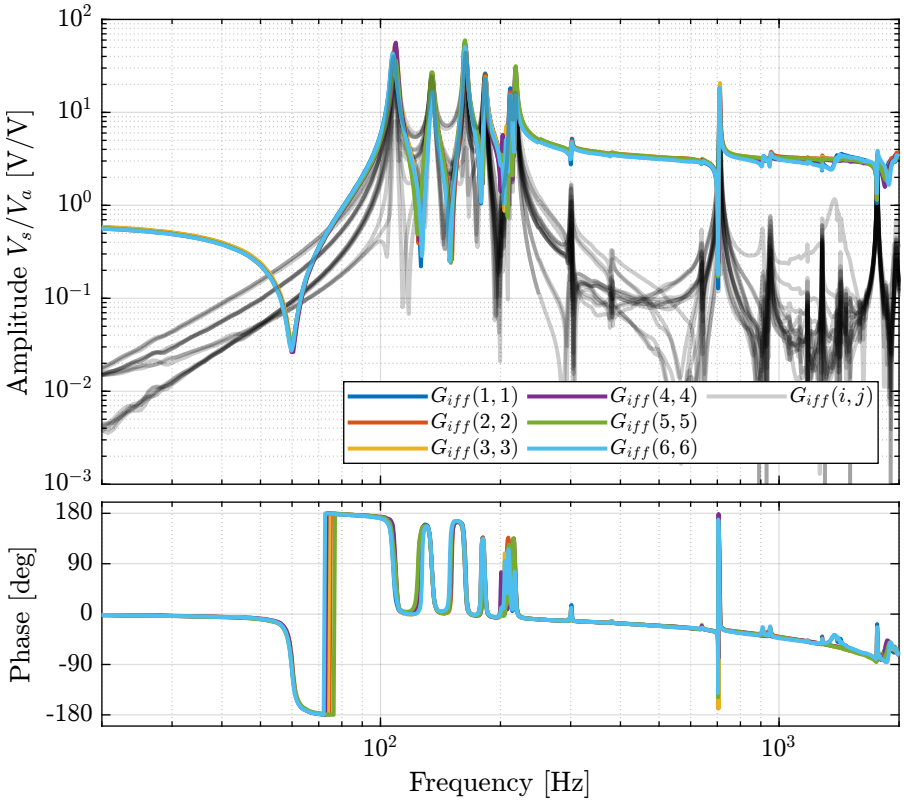

Figure 1.4: Measured FRF for the IFF plant

1.2.5 Save Identified Plants

```Matlab
save('matlab/mat/identified_plants_enc_struts.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

1.3 Comparison with the Simscape Model

In this section, the measured dynamics is compared with the dynamics estimated from the Simscape model.

1.3.1 Load measured FRF

```Matlab
%% Load data
load('identified_plants_enc_struts.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

1.3.2 Dynamics from Actuator to Force Sensors

```Matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof');
```

```Matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

1.3.3 Dynamics from Actuator to Encoder

```Matlab
%% Initialization of the Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');
```

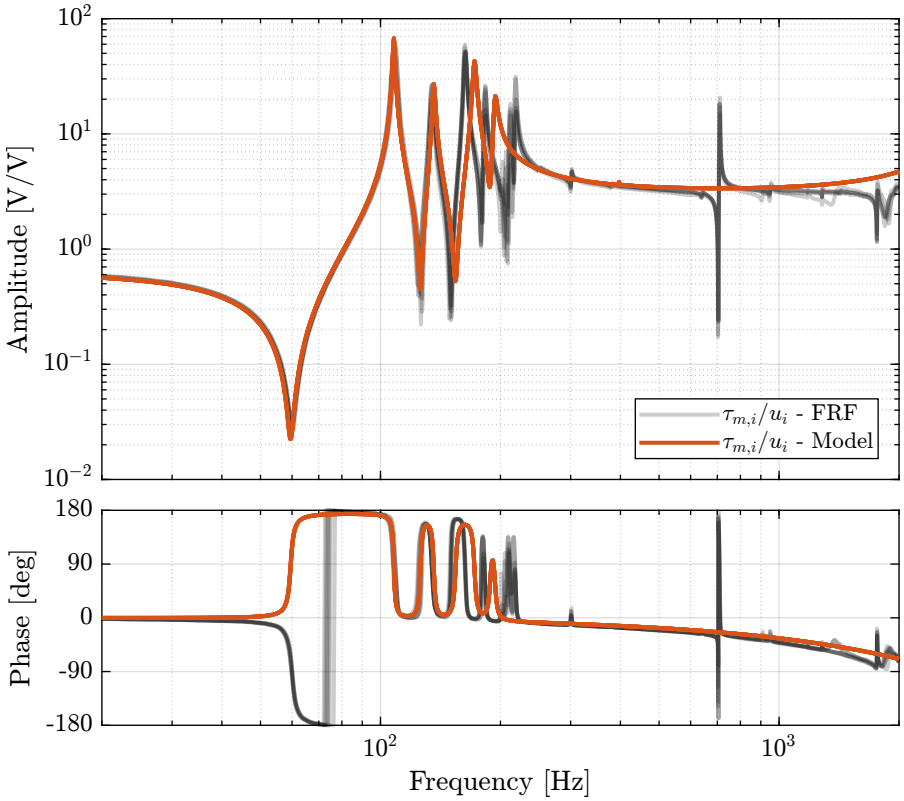

Figure 1.5: Diagonal elements of the IFF Plant

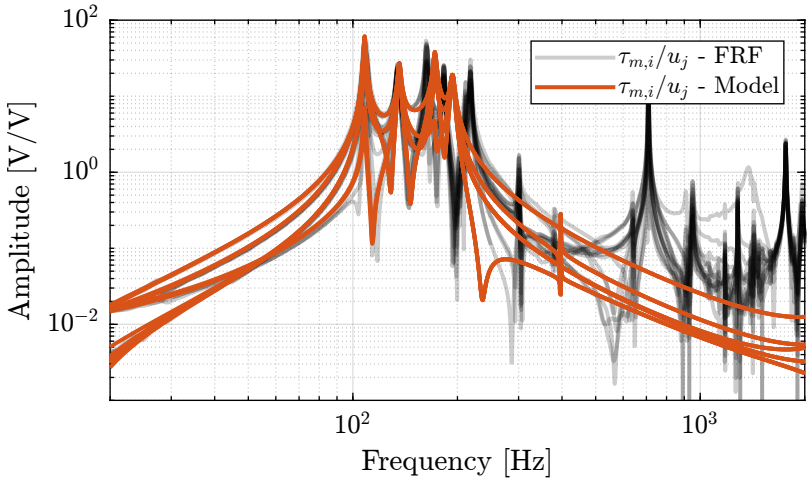

Figure 1.6: Off diagonal elements of the IFF Plant

```Matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

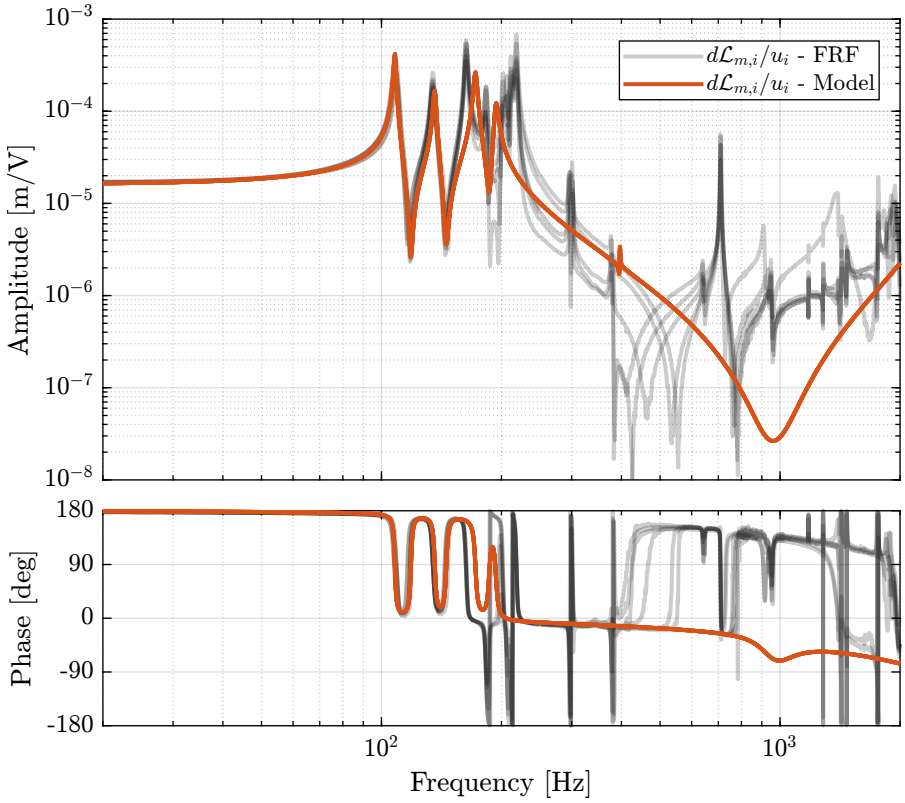

Figure 1.7: Diagonal elements of the DVF Plant

1.3.4 Effect of a change in bending damping of the joints

```Matlab
%% Tested bending dampings [Nm/(rad/s)]
cRs = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1];
```

```Matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

Then the identification is performed for all the values of the bending damping.

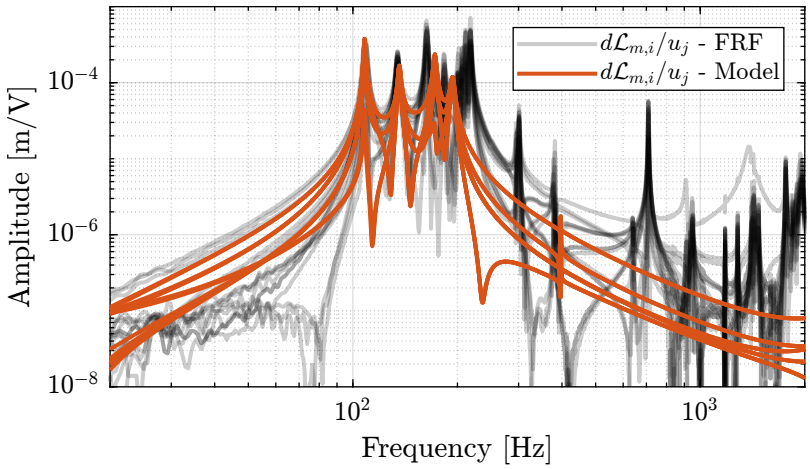

Figure 1.8: Off diagonal elements of the DVF Plant

Matlab
```
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(cRs), 1)};

for i = 1:length(cRs)
    n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                           'flex_top_type', '4dof', ...
                                           'motion_sensor_type', 'struts', ...
                                           'actuator_type', 'flexible', ...
                                           'flex_bot_cRx', cRs(i), ...
                                           'flex_bot_cRy', cRs(i), ...
                                           'flex_top_cRx', cRs(i), ...
                                           'flex_top_cRy', cRs(i));

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

- Could be nice
- Actual damping is very small

1.3.5 Effect of a change in damping factor of the APA

Matlab
```
%% Tested bending dampings [Nm/(rad/s)]
xis = [1e-3, 5e-3, 1e-2, 5e-2, 1e-1];
```

Matlab
```
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

```matlab
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(xis), 1)};

for i = 1:length(xis)
    n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                           'flex_top_type', '4dof', ...
                                           'motion_sensor_type', 'struts', ...
                                           'actuator_type', 'flexible', ...
                                           'actuator_xi', xis(i));

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

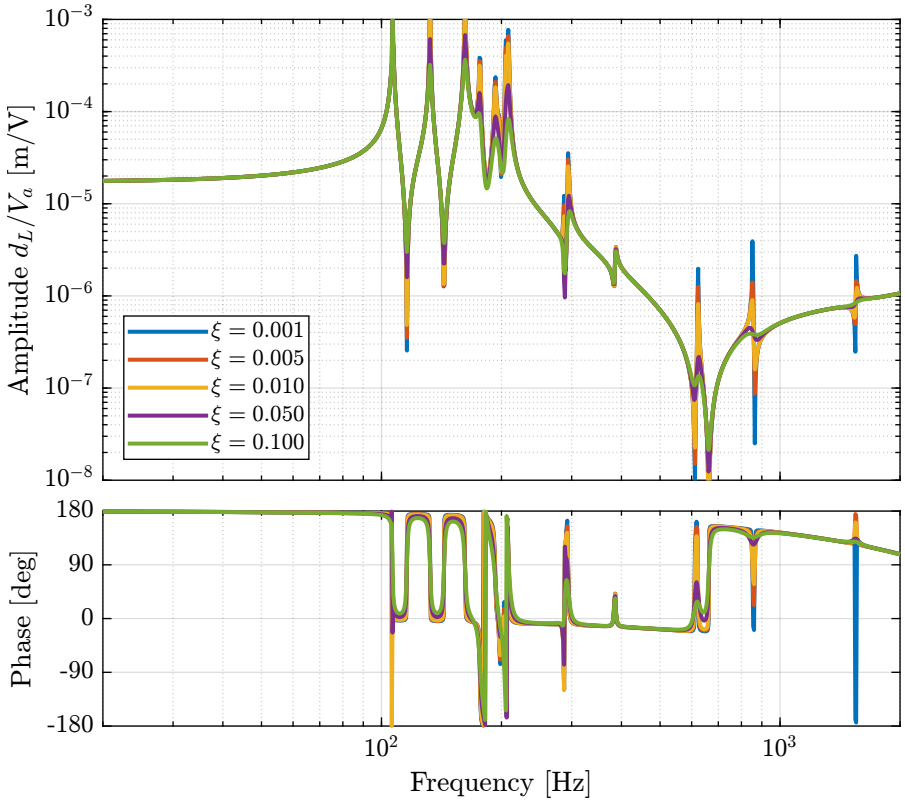

Figure 1.9: Effect of the APA damping factor ξ on the dynamics from u to $d\mathcal{L}$

Important

Damping factor ξ has a large impact on the damping of the "spurious resonances" at 200Hz and 300Hz.

Question

Why is the damping factor does not change the damping of the first peak?

1.3.6 Effect of a change in stiffness damping coef of the APA

Matlab
```
m_coef = 1e1;
```

Matlab
```
%% Tested bending dampings [Nm/(rad/s)]
k_coefs = [1e-6, 5e-6, 1e-5, 5e-5, 1e-4];
```

Matlab
```
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

Matlab
```
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(k_coefs), 1)};
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');

for i = 1:length(k_coefs)
    k_coef = k_coefs(i);

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

1.3.7 Effect of a change in mass damping coef of the APA

Matlab
```
k_coef = 1e-6;
```

Matlab
```
%% Tested bending dampings [Nm/(rad/s)]
m_coefs = [1e1, 5e1, 1e2, 5e2, 1e3];
```

Matlab
```
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders
```

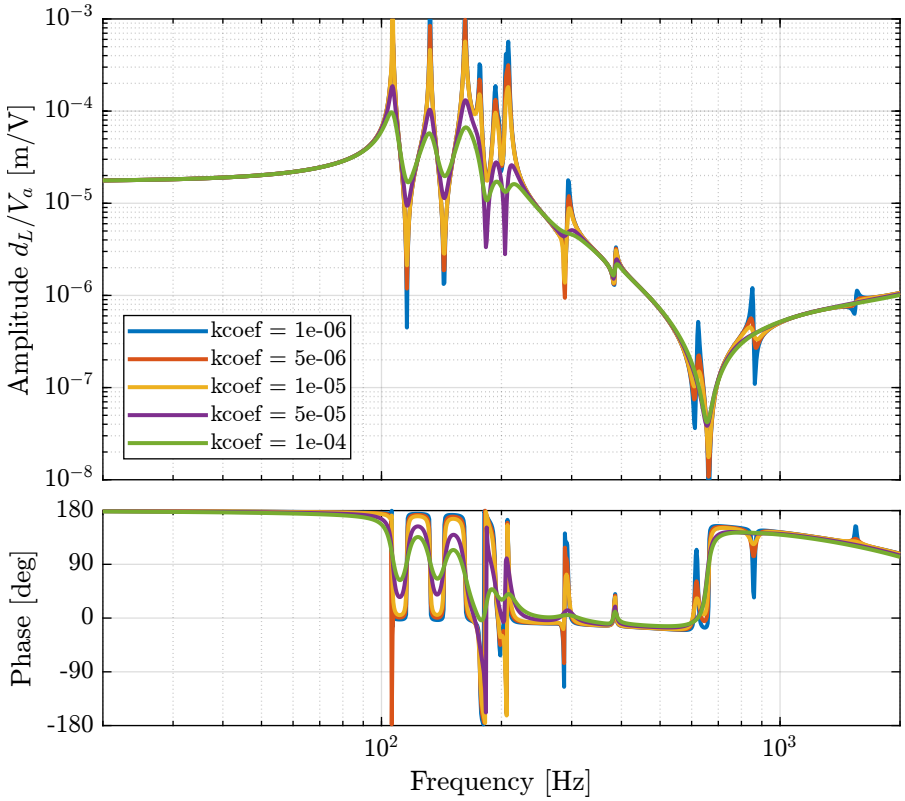

Figure 1.10: Effect of a change of the damping "stiffness coeficient" on the transfer function from u to $d\mathcal{L}$

```Matlab
%% Idenfity the transfer function from actuator to encoder for all bending dampins
Gs = {zeros(length(m_coefs), 1)};
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible');

for i = 1:length(m_coefs)
    m_coef = m_coefs(i);

    G = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
    G.InputName  = {'Va1', 'Va2', 'Va3', 'Va4', 'Va5', 'Va6'};
    G.OutputName = {'dL1', 'dL2', 'dL3', 'dL4', 'dL5', 'dL6'};

    Gs(i) = {G};
end
```

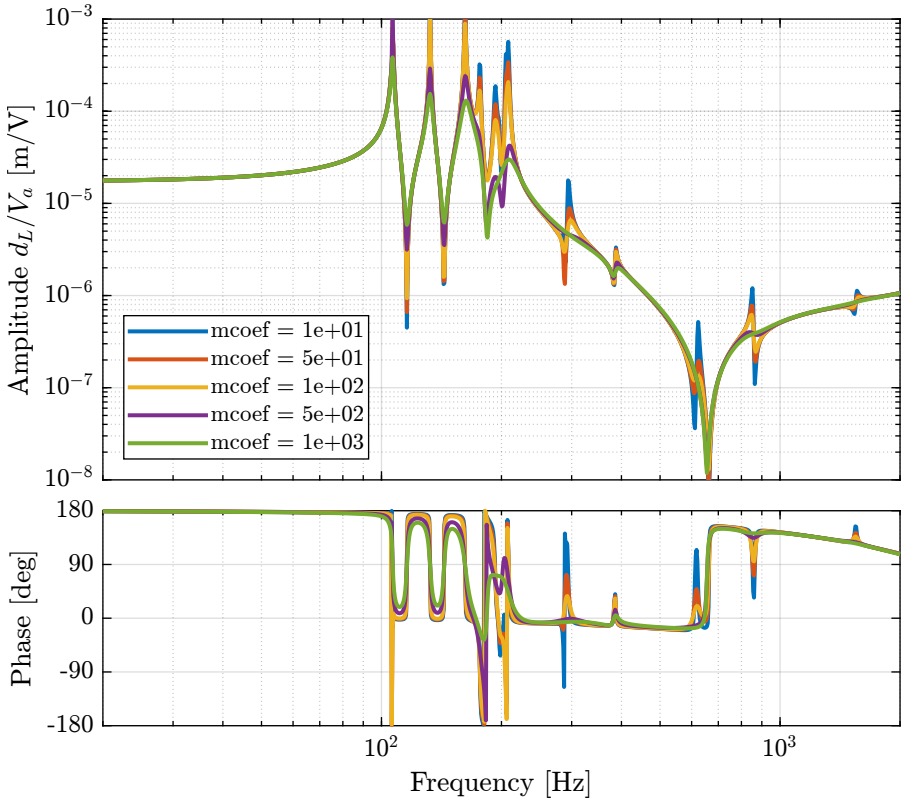

Figure 1.11: Effect of a change of the damping "mass coeficient" on the transfer function from u to $d\mathcal{L}$

1.3.8 Using Flexible model

```Matlab
d_aligns = [[-0.05,  -0.3,  0];
            [ 0,      0.5,  0];
            [-0.1,   -0.3,  0];
            [ 0,      0.3,  0];
            [-0.05,   0.05, 0];
            [0,       0,    0]]*1e-3;
```

Matlab
```
d_aligns = zeros(6,3);
% d_aligns(1,:) = [-0.05,  -0.3,  0]*1e-3;
d_aligns(2,:) = [ 0,      0.3,  0]*1e-3;
```

Matlab
```
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

Question

Why do we have smaller resonances when using flexible APA? On the test bench we have the same resonance as the 2DoF model. Could it be due to the compliance in other dof of the flexible model?

Matlab
```
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

Matlab
```
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

1.3.9 Flexible model + encoders fixed to the plates

Matlab
```
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors
```

Matlab
```
d_aligns = [[-0.05,  -0.3,  0];
            [ 0,      0.5,  0];
            [-0.1,   -0.3,  0];
            [ 0,      0.3,  0];
            [-0.05,   0.05, 0];
            [0,       0,    0]]*1e-3;
```

```Matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

```Matlab
Gdvf_struts = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

```Matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'actuator_d_align', d_aligns);
```

```Matlab
Gdvf_plates = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

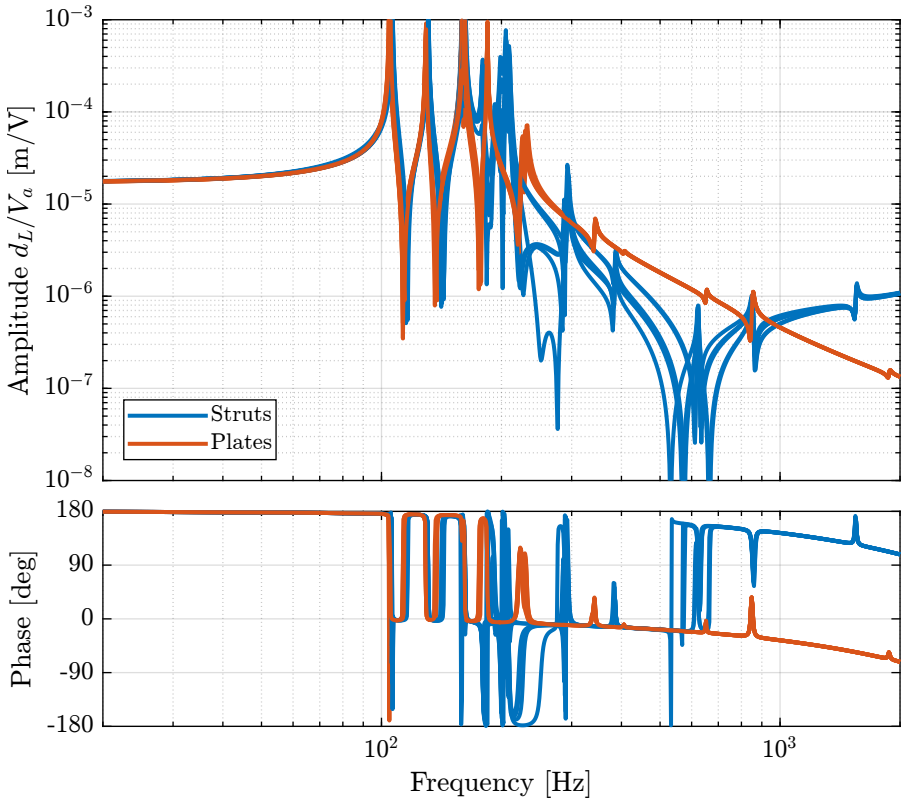

Figure 1.12: Comparison of the dynamics from V_a to d_L when the encoders are fixed to the struts (blue) and to the plates (red). APA are modeled as a flexible element.

1.4 Integral Force Feedback

1.4.1 Identification of the IFF Plant

Matlab
```
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof');
```

Matlab
```
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

1.4.2 Root Locus and Decentralized Loop gain

Matlab
```
%% IFF Controller
Kiff_g1 = -(1/(s + 2*pi*40))*... % Low pass filter (provides integral action above 40Hz)
           (s/(s + 2*pi*30))*... % High pass filter to limit low frequency gain
           (1/(1 + s/2/pi/500))*... % Low pass filter to be more robust to high frequency resonances
           eye(6); % Diagonal 6x6 controller
```

Then the "optimal" IFF controller is:

Matlab
```
%% IFF controller with Optimal gain
Kiff = g*Kiff_g1;
```

Matlab
```
save('matlab/mat/Kiff.mat', 'Kiff')
```

1.4.3 Multiple Gains - Simulation

Matlab
```
%% Tested IFF gains
iff_gains = [4, 10, 20, 40, 100, 200, 400];
```

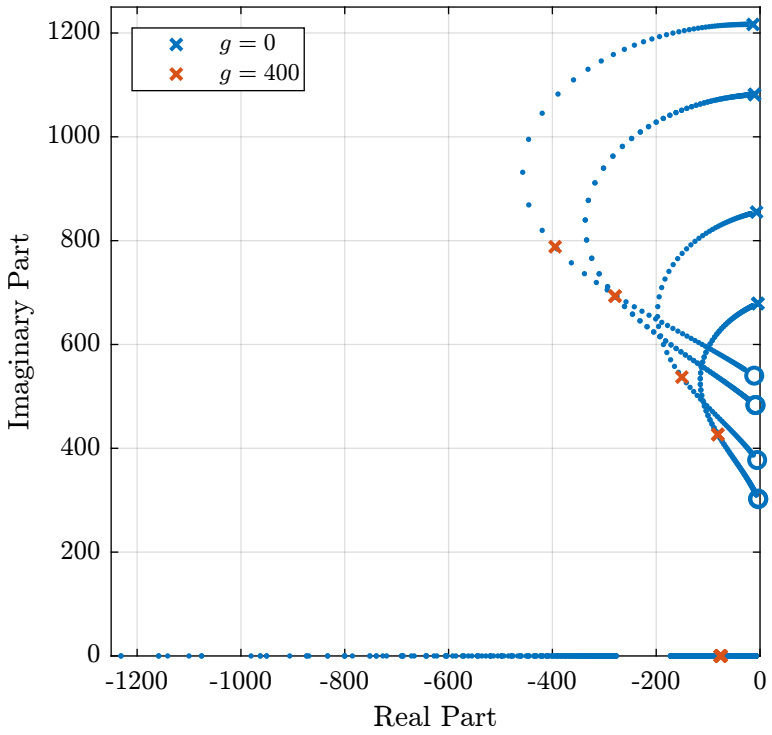

Figure 1.13: Root Locus for the IFF control strategy

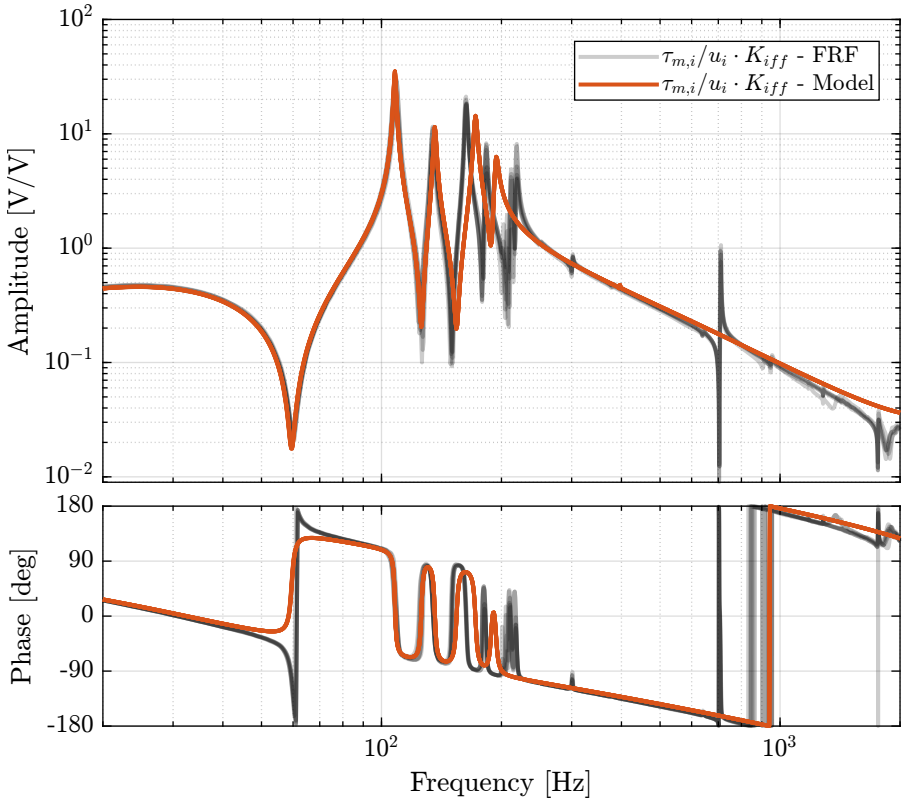

Figure 1.14: Bode plot of the "decentralized loop gain" $G_{\text{iff}}(i,i) \times K_{\text{iff}}(i,i)$

```Matlab
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', '2dof', ...
                                       'controller_type', 'iff');
```

```Matlab
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
Gd_iff = {zeros(1, length(iff_gains))};

clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Strut Displacement (encoder)

for i = 1:length(iff_gains)
    Kiff = iff_gains(i)*Kiff_g1*eye(6); % IFF Controller
    Gd_iff(i) = {exp(-s*Ts)*linearize(mdl, io, 0.0, options)};

    isstable(Gd_iff{i})
end
```

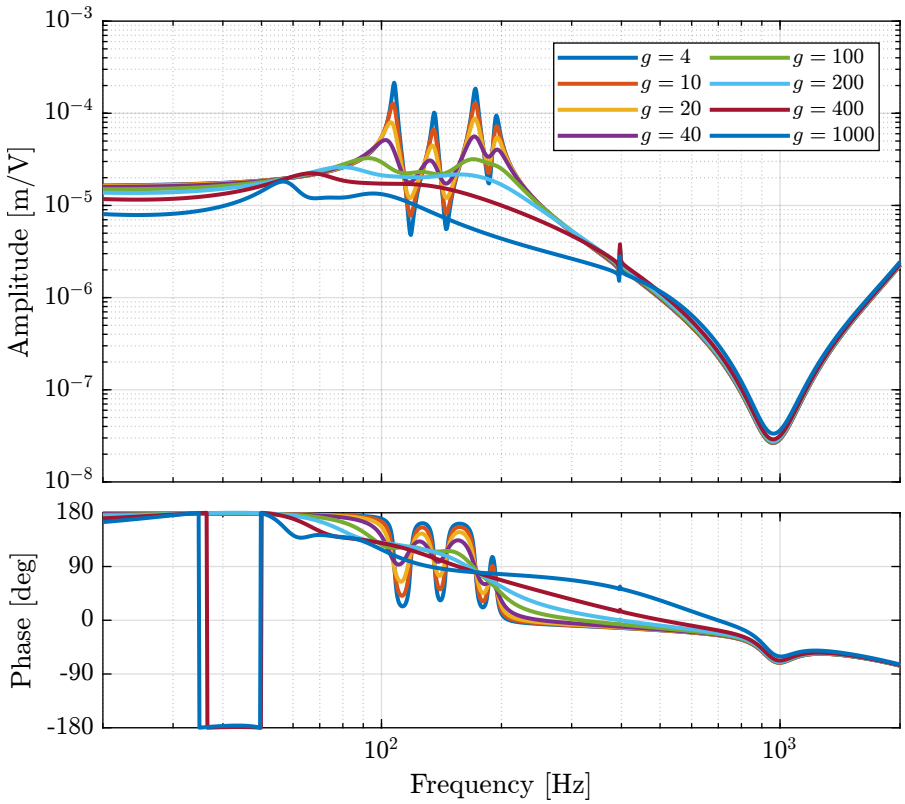

Figure 1.15: Effect of the IFF gain g on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

1.4.4 Experimental Results - Gains

Let's look at the damping introduced by IFF as a function of the IFF gain and compare that with the results obtained using the Simscape model.

Load Data

```Matlab
%% Load Identification Data
meas_iff_gains = {};

for i = 1:length(iff_gains)
    meas_iff_gains(i) = {load(sprintf('mat/iff_strut_1_noise_g_%i.mat', iff_gains(i)), 't', 'Vexc', 'Vs', 'de', 'u')};
end
```

Spectral Analysis - Setup

```Matlab
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_iff_gains{1}.t(end) - (meas_iff_gains{1}.t(1)))/(length(meas_iff_gains{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_gains{1}.Vexc, meas_iff_gains{1}.de, win, [], [], 1/Ts);
```

DVF Plant

```Matlab
%% DVF Plant (transfer function from u to dLm)
G_iff_gains = {};

for i = 1:length(iff_gains)
    G_iff_gains{i} = tfestimate(meas_iff_gains{i}.Vexc, meas_iff_gains{i}.de(:,1), win, [], [], 1/Ts);
end
```

Important

The IFF control strategy is very effective for the damping of the suspension modes. It however does not damp the modes at 200Hz, 300Hz and 400Hz (flexible modes of the APA). This is very logical.
Also, the experimental results and the models obtained from the Simscape model are in agreement.

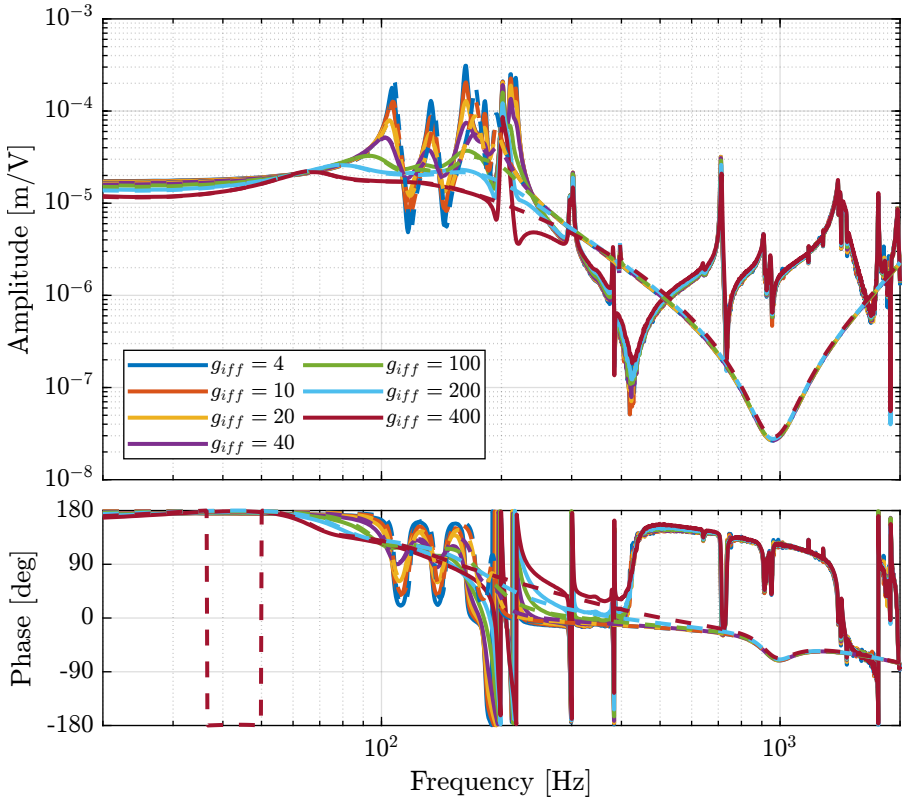

Figure 1.16: Transfer function from u to $d\mathcal{L}_m$ for multiple values of the IFF gain

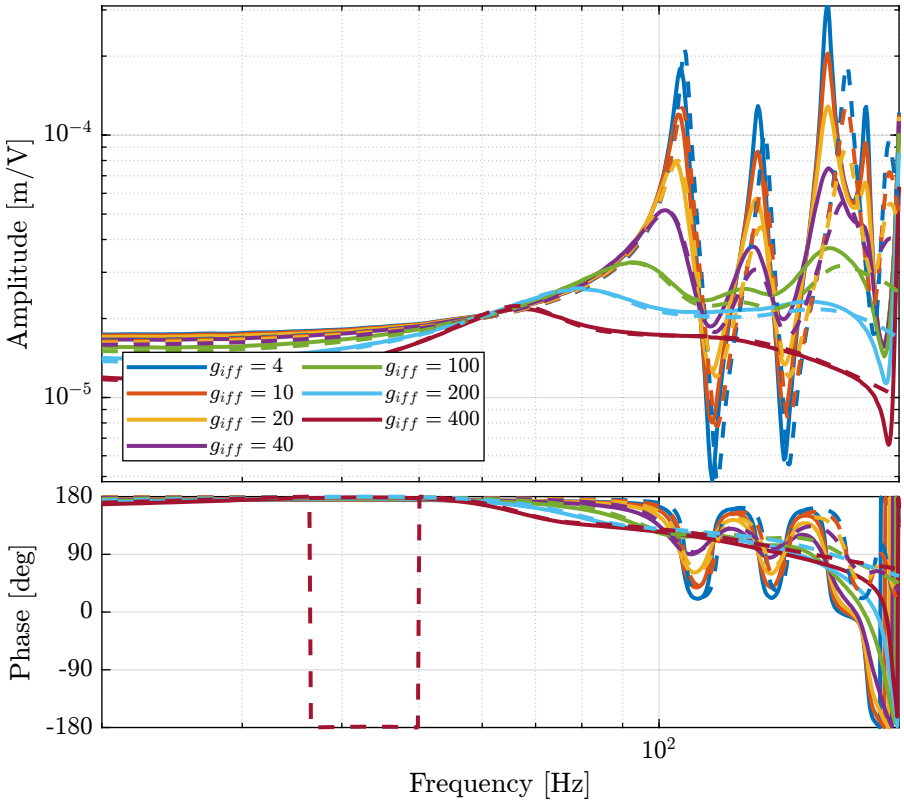

Figure 1.17: Transfer function from u to $d\mathcal{L}_m$ for multiple values of the IFF gain (Zoom)

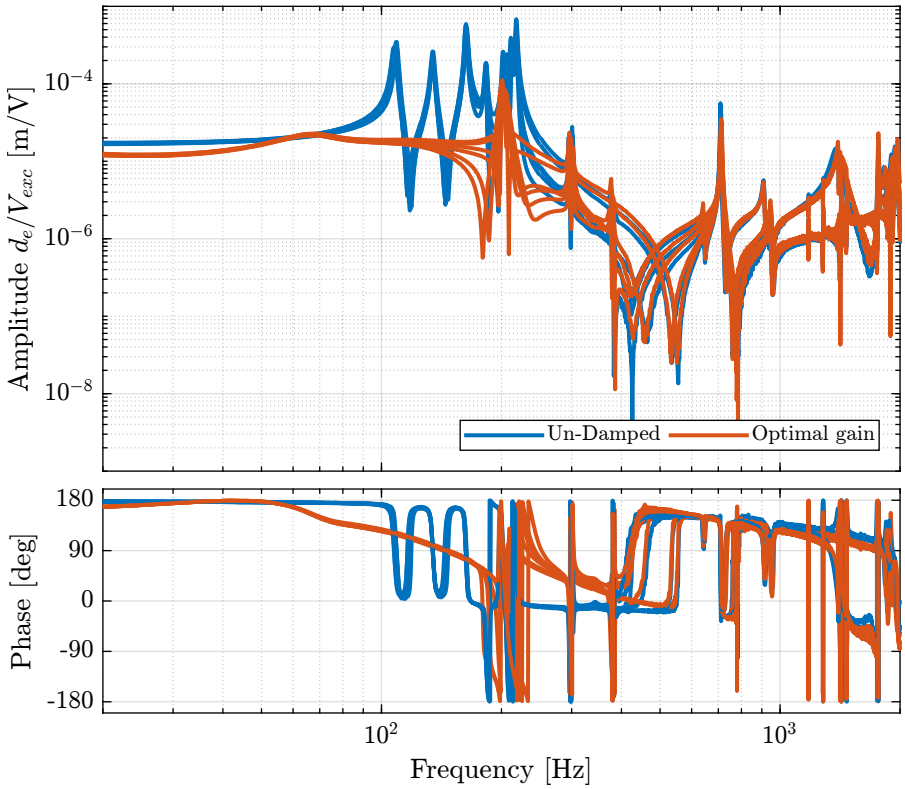

Figure 1.18: Comparison of the diagonal elements of the tranfer function from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ without active damping and with optimal IFF gain

Experimental Results - Comparison of the un-damped and fully damped system

Question

A series of modes at around 205Hz are also damped.
Are these damped modes at 205Hz additional "suspension" modes or flexible modes of the struts?

1.4.5 Experimental Results - Damped Plant with Optimal gain

Let's now look at the 6×6 damped plant with the optimal gain $g = 400$.

Load Data

Matlab
```matlab
%% Load Identification Data
meas_iff_struts = {};

for i = 1:6
    meas_iff_struts(i) = {load(sprintf('mat/iff_strut_%i_noise_g_400.mat', i), 't', 'Vexc', 'Vs', 'de', 'u')};
end
```

Spectral Analysis - Setup

Matlab
```matlab
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_iff_struts{1}.t(end) - (meas_iff_struts{1}.t(1)))/(length(meas_iff_struts{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_struts{1}.Vexc, meas_iff_struts{1}.de, win, [], [], 1/Ts);
```

DVF Plant

Matlab
```matlab
%% DVF Plant (transfer function from u to dLm)
G_iff_opt = {};

for i = 1:6
    G_iff_opt{i} = tfestimate(meas_iff_struts{i}.Vexc, meas_iff_struts{i}.de, win, [], [], 1/Ts);
end
```

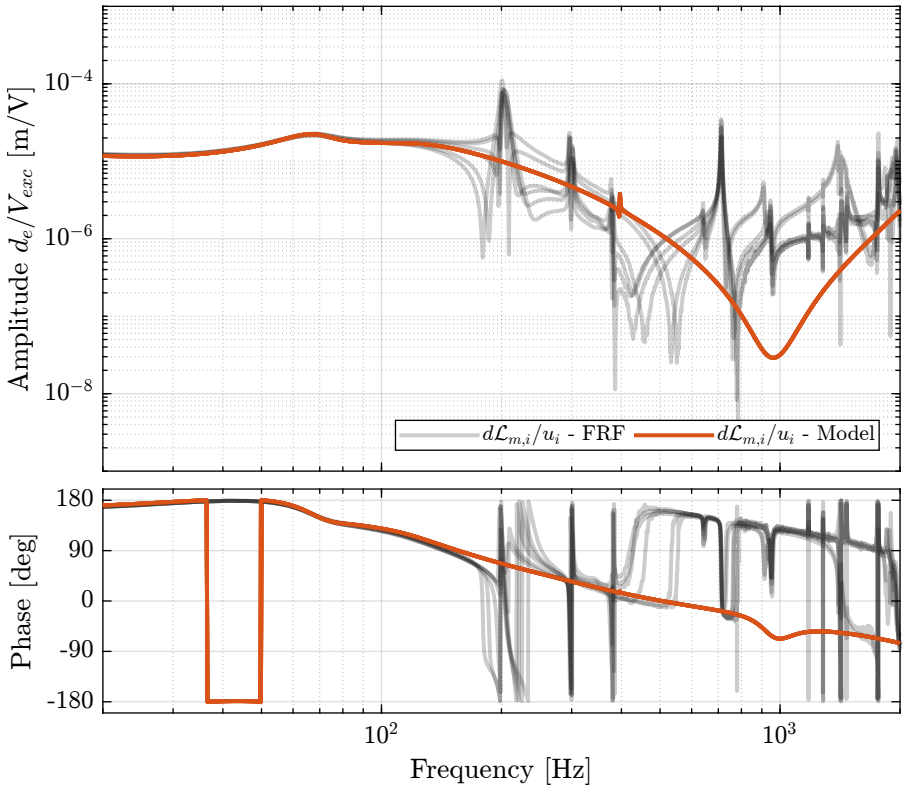

Figure 1.19: Comparison of the diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

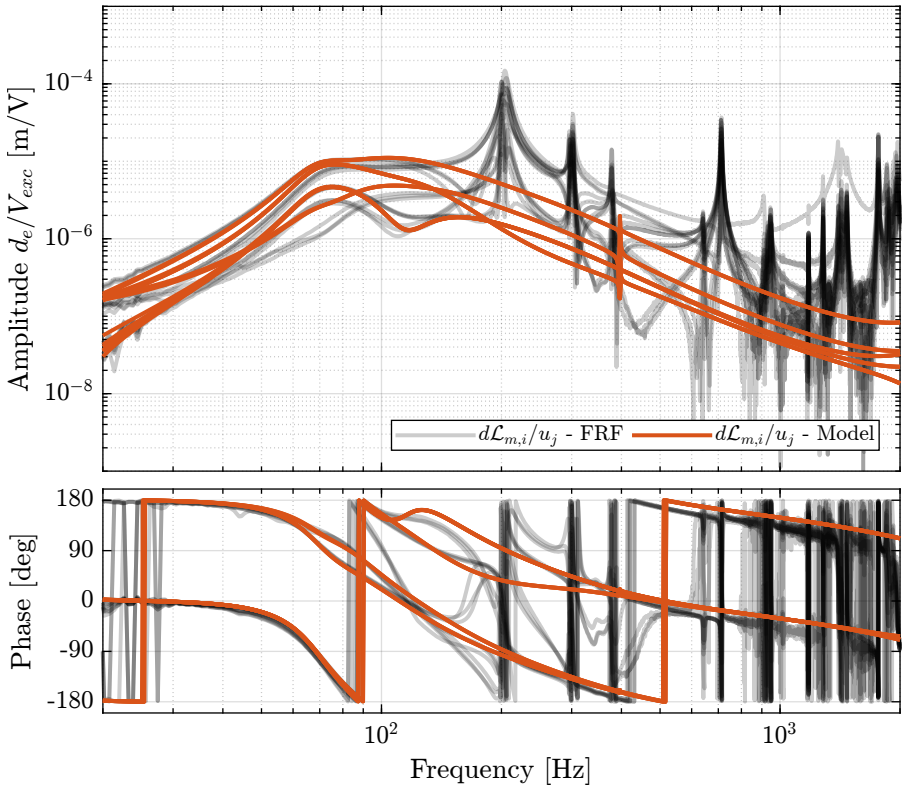

Figure 1.20: Comparison of the off-diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

Important

With the IFF control strategy applied and the optimal gain used, the suspension modes are very well damped. Remains the undamped flexible modes of the APA (200Hz, 300Hz, 400Hz), and the modes of the plates (700Hz).
The Simscape model and the experimental results are in very good agreement.

1.5 Modal Analysis

Several 3-axis accelerometers are fixed on the top platform of the nano-hexapod as shown in Figure 1.23.

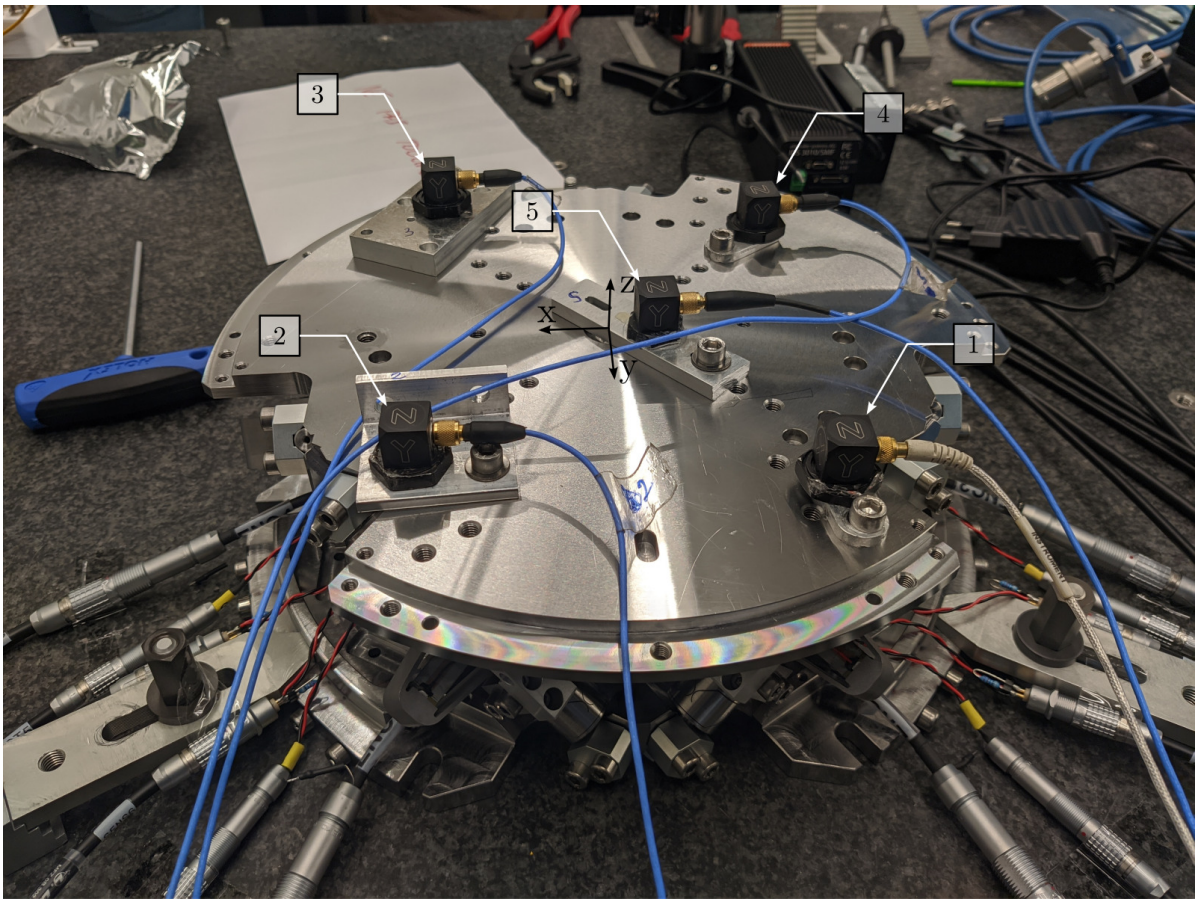

Figure 1.21: Location of the accelerometers on top of the nano-hexapod

The top platform is then excited using an instrumented hammer as shown in Figure 1.22.

1.5.1 Effectiveness of the IFF Strategy - Compliance

In this section, we wish to estimated the effectiveness of the IFF strategy concerning the compliance.

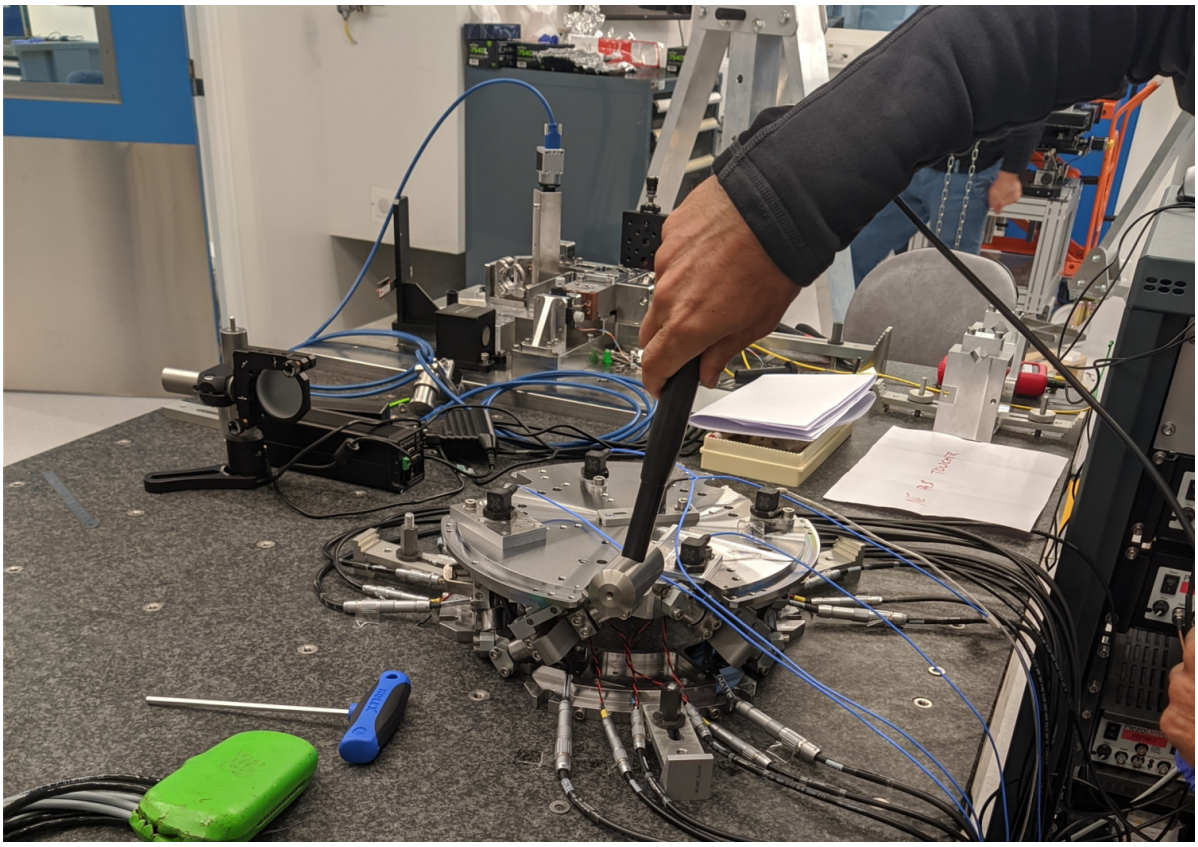

Figure 1.22: Example of an excitation using an instrumented hammer

The top plate is excited vertically using the instrumented hammer two times:

1. no control loop is used
2. decentralized IFF is used

The data is loaded.

```matlab
frf_ol  = load('Measurement_Z_axis.mat'); % Open-Loop
frf_iff = load('Measurement_Z_axis_damped.mat'); % IFF
```

The mean vertical motion of the top platform is computed by averaging all 5 accelerometers.

```matlab
%% Multiply by 10 (gain in m/s^2/V) and divide by 5 (number of accelerometers)
d_frf_ol = 10/5*(frf_ol.FFT1_H1_4_1_RMS_Y_Mod + frf_ol.FFT1_H1_7_1_RMS_Y_Mod + frf_ol.FFT1_H1_10_1_RMS_Y_Mod +
↪  frf_ol.FFT1_H1_13_1_RMS_Y_Mod + frf_ol.FFT1_H1_16_1_RMS_Y_Mod)./(2*pi*frf_ol.FFT1_H1_16_1_RMS_X_Val).^2;
d_frf_iff = 10/5*(frf_iff.FFT1_H1_4_1_RMS_Y_Mod + frf_iff.FFT1_H1_7_1_RMS_Y_Mod + frf_iff.FFT1_H1_10_1_RMS_Y_Mod +
↪  frf_iff.FFT1_H1_13_1_RMS_Y_Mod + frf_iff.FFT1_H1_16_1_RMS_Y_Mod)./(2*pi*frf_iff.FFT1_H1_16_1_RMS_X_Val).^2;
```

The vertical compliance (magnitude of the transfer function from a vertical force applied on the top plate to the vertical motion of the top plate) is shown in Figure 1.23.

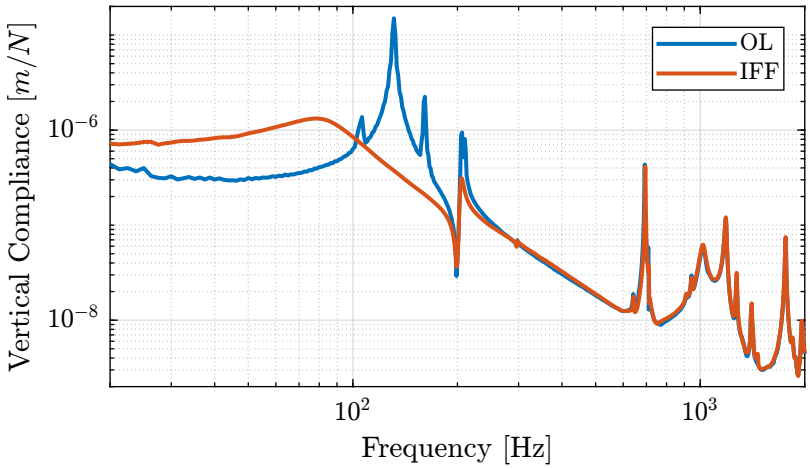

Figure 1.23: Measured vertical compliance with and without IFF

Important

From Figure 1.23, it is clear that the IFF control strategy is very effective in damping the suspensions modes of the nano-hexapode. It also has the effect of degrading (slightly) the vertical compliance at low frequency.
It also seems some damping can be added to the modes at around 205Hz which are flexible modes of the struts.

1.5.2 Comparison with the Simscape Model

Let's now compare the measured vertical compliance with the vertical compliance as estimated from the Simscape model.

The transfer function from a vertical external force to the absolute motion of the top platform is identified (with and without IFF) using the Simscape model. The comparison is done in Figure 1.24. Again, the model is quite accurate!

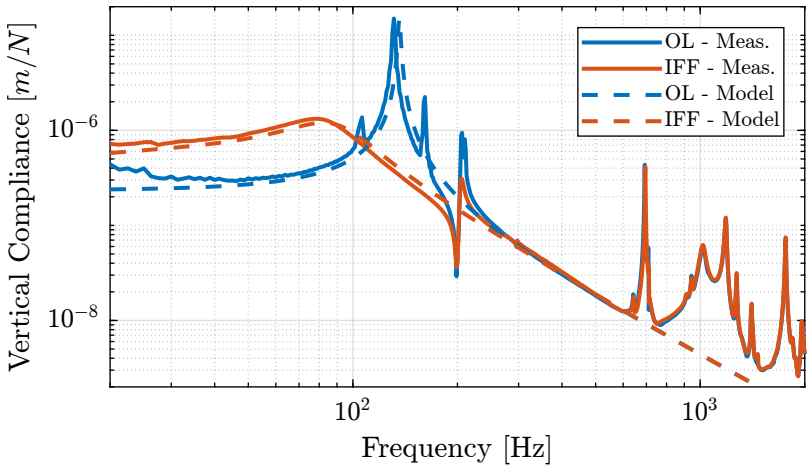

Figure 1.24: Measured vertical compliance with and without IFF

1.5.3 Obtained Mode Shapes

Then, several excitation are performed using the instrumented Hammer and the mode shapes are extracted.

We can observe the mode shapes of the first 6 modes that are the suspension modes (the plate is behaving as a solid body) in Figure 1.25.

Then, there is a mode at 692Hz which corresponds to a flexible mode of the top plate (Figure 1.26).

The obtained modes are summarized in Table 1.1.

Table 1.1: Description of the identified modes

Mode	Freq. [Hz]	Description
1	105	Suspension Mode: Y-translation
2	107	Suspension Mode: X-translation
3	131	Suspension Mode: Z-translation
4	161	Suspension Mode: Y-tilt
5	162	Suspension Mode: X-tilt
6	180	Suspension Mode: Z-rotation
7	692	(flexible) Membrane mode of the top platform

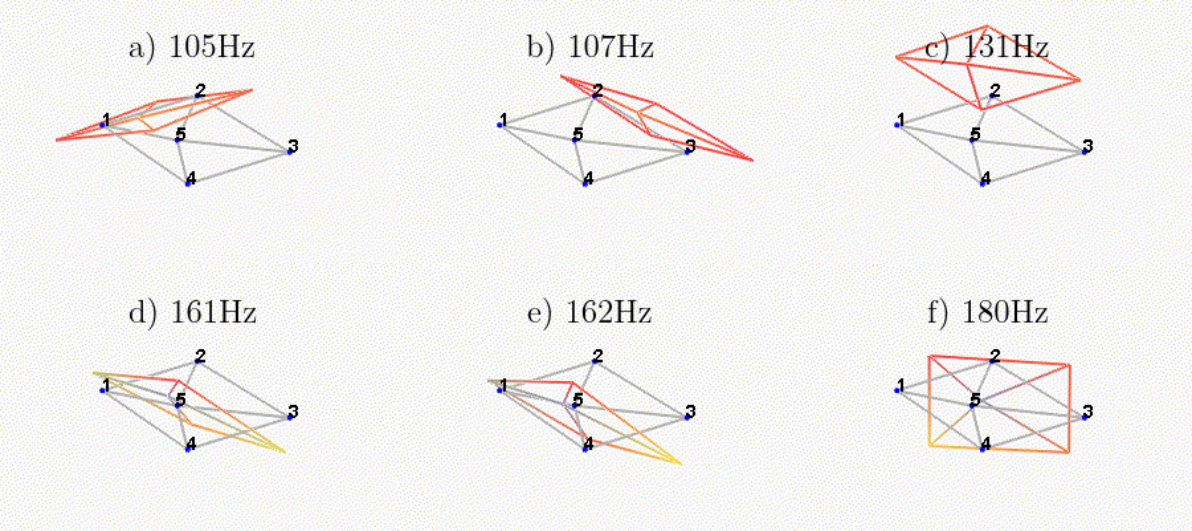

Figure 1.25: Measured mode shapes for the first six modes

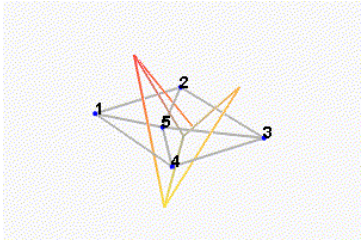

Figure 1.26: First flexible mode at 692Hz

1.6 Accelerometers fixed on the top platform

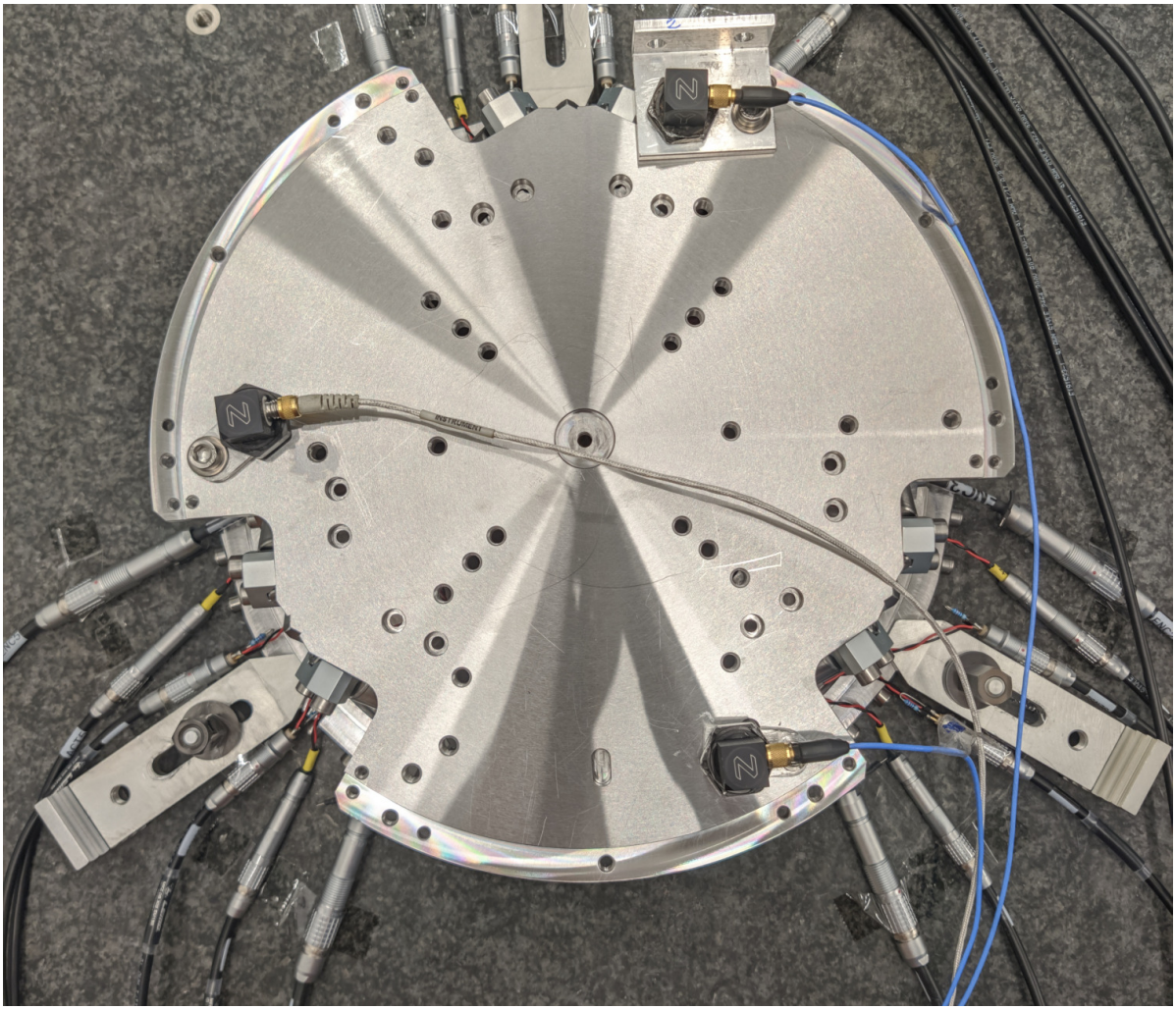

Figure 1.27: Accelerometers fixed on the top platform

1.6.1 Experimental Identification

Matlab

```
%% Load Identification Data
meas_acc = {};

for i = 1:6
    meas_acc(i) = {load(sprintf('mat/meas_acc_top_plat_strut_%i.mat', i), 't', 'Va', 'de', 'Am')};
end
```

Matlab

```
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_acc{1}.t(end) - (meas_acc{1}.t(1)))/(length(meas_acc{1}.t)-1);
```

```
% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_acc{1}.Va, meas_acc{1}.de, win, [], [], 1/Ts);
```

The sensibility of the accelerometers are $0.1V/g \approx 0.01V/(m/s^2)$.

Matlab
```
%% Compute the 6x6 transfer function matrix
G_acc = zeros(length(f), 6, 6);

for i = 1:6
    G_acc(:,:,i) = tfestimate(meas_acc{i}.Va, 1/0.01*meas_acc{i}.Am, win, [], [], 1/Ts);
end
```

1.6.2 Location and orientation of accelerometers

Matlab
```
Opm = [ 0.047, -0.112, 10e-3;
        0.047, -0.112, 10e-3;
       -0.113,  0.011, 10e-3;
       -0.113,  0.011, 10e-3;
        0.040,  0.113, 10e-3;
        0.040,  0.113, 10e-3]';

Osm = [-1,  0, 0;
        0,  0, 1;
        0, -1, 0;
        0,  0, 1;
       -1,  0, 0;
        0,  0, 1]';
```

1.6.3 COM

Matlab
```
Hbm = -15e-3;

M = getTransformationMatrixAcc(Opm-[0;0;Hbm], Osm);
J = getJacobianNanoHexapod(Hbm);
```

Matlab
```
G_acc_CoM = zeros(size(G_acc));

for i = 1:length(f)
    G_acc_CoM(i, :, :) = inv(M)*squeeze(G_acc(i, :, :))*inv(J');
end
```

1.6.4 COK

```Matlab
Hbm = -42.3e-3;

M = getTransformationMatrixAcc(Opm-[0;0;Hbm], Osm);
J = getJacobianNanoHexapod(Hbm);
```

```Matlab
G_acc_CoK = zeros(size(G_acc));

for i = 1:length(f)
    G_acc_CoK(i, :, :) = inv(M)*squeeze(G_acc(i, :, :))*inv(J');
end
```

1.6.5 Comp with the Simscape Model

```Matlab
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'struts', ...
                                       'actuator_type', 'flexible', ...
                                       'MO_B', -42.3e-3);
```

```Matlab
%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/D'],  1, 'openoutput'); io_i = io_i + 1; % Relative Motion Outputs

G = linearize(mdl, io, 0.0, options);
G.InputName  = {'F1', 'F2', 'F3', 'F4', 'F5', 'F6'};
G.OutputName = {'D1', 'D2', 'D3', 'D4', 'D5', 'D6'};
```

Then use the Jacobian matrices to obtain the "cartesian" centralized plant.

```Matlab
Gc = inv(n_hexapod.geometry.J)*...
     G*...
     inv(n_hexapod.geometry.J');
```

2 Encoders fixed to the plates

In this section, the encoders are fixed to the plates rather than to the struts as shown in Figure 2.1.

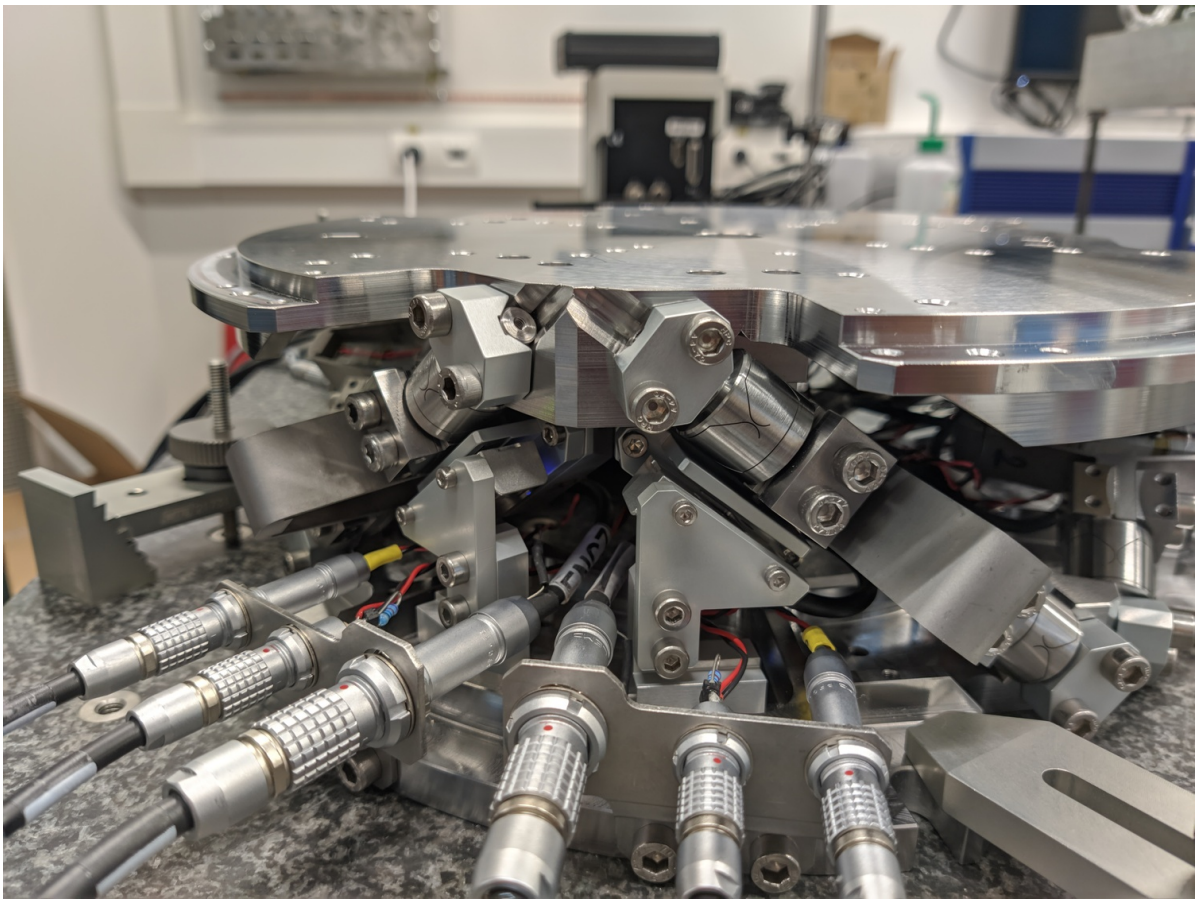

Figure 2.1: Nano-Hexapod with encoders fixed to the struts

It is structured as follow:

- Section 2.1: The dynamics of the nano-hexapod is identified
- Section 2.2: The identified dynamics is compared with the Simscape model
- Section 2.3: The Integral Force Feedback (IFF) control strategy is applied and the dynamics of the damped nano-hexapod is identified and compare with the Simscape model
- Section 2.4: The High Authority Control (HAC) in the frame of the struts is developed

- Section 2.5: Some reference tracking tests are performed in order to experimentally validate the HAC-LAC control strategy.

2.1 Identification of the dynamics

2.1.1 Load Measurement Data

Matlab
```
%% Load Identification Data
meas_data_lf = {};

for i = 1:6
    meas_data_lf(i) = {load(sprintf('mat/frf_exc_strut_%i_enc_plates_noise.mat', i), 't', 'Va', 'Vs', 'de')};
end
```

2.1.2 Spectral Analysis - Setup

Matlab
```
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_data_lf{1}.t(end) - (meas_data_lf{1}.t(1)))/(length(meas_data_lf{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_data_lf{1}.Va, meas_data_lf{1}.de, win, [], [], 1/Ts);
```

2.1.3 DVF Plant

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 2.2).

Matlab
```
%% Coherence
coh_dvf = zeros(length(f), 6, 6);

for i = 1:6
    coh_dvf(:, :, i) = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
end
```

Then the 6x6 transfer function matrix is estimated (Figure 2.3).

Matlab
```
%% DVF Plant (transfer function from u to dLm)
G_dvf = zeros(length(f), 6, 6);

for i = 1:6
```

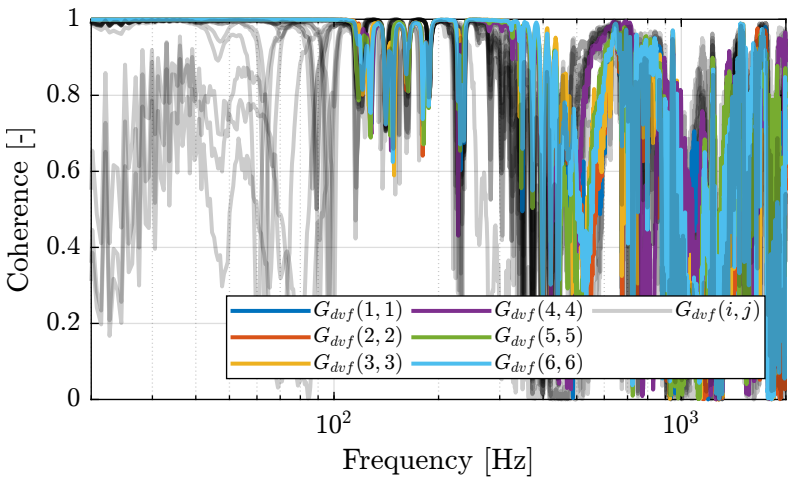

Figure 2.2: Obtained coherence for the DVF plant

```
    G_dvf(:,:,i) = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.de, win, [], [], 1/Ts);
end
```

2.1.4 IFF Plant

First, let's compute the coherence from the excitation voltage and the displacement as measured by the encoders (Figure 2.4).

Matlab
```
%% Coherence for the IFF plant
coh_iff = zeros(length(f), 6, 6);

for i = 1:6
    coh_iff(:,:,i) = mscohere(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
end
```

Then the 6x6 transfer function matrix is estimated (Figure 2.5).

Matlab
```
%% IFF Plant
G_iff = zeros(length(f), 6, 6);

for i = 1:6
    G_iff(:,:,i) = tfestimate(meas_data_lf{i}.Va, meas_data_lf{i}.Vs, win, [], [], 1/Ts);
end
```

2.1.5 Save Identified Plants

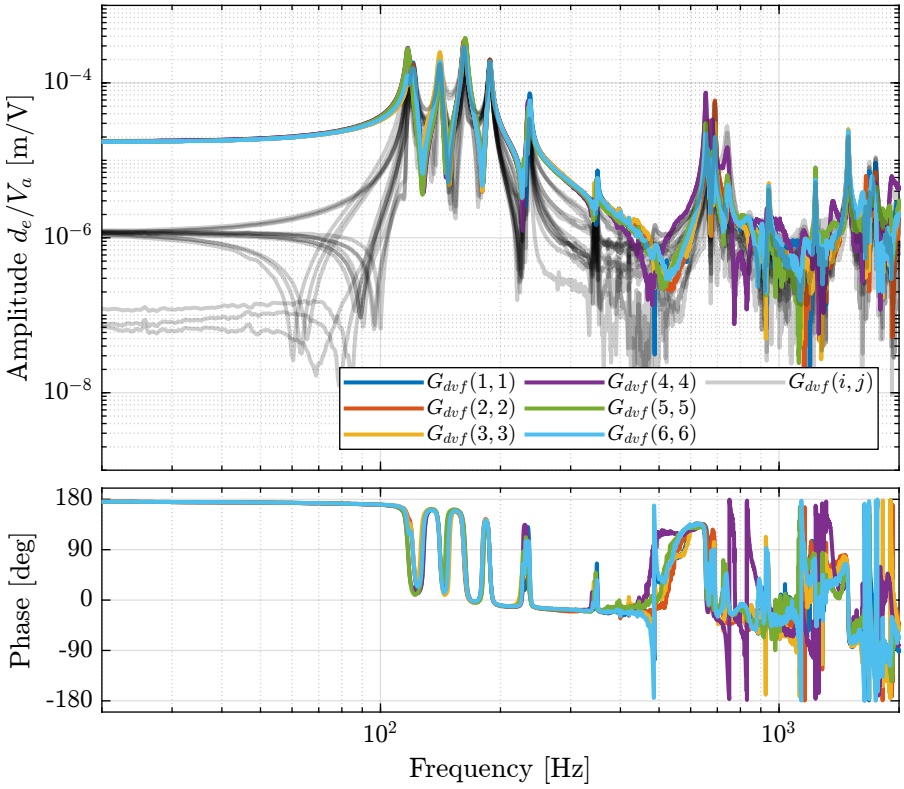

Figure 2.3: Measured FRF for the DVF plant

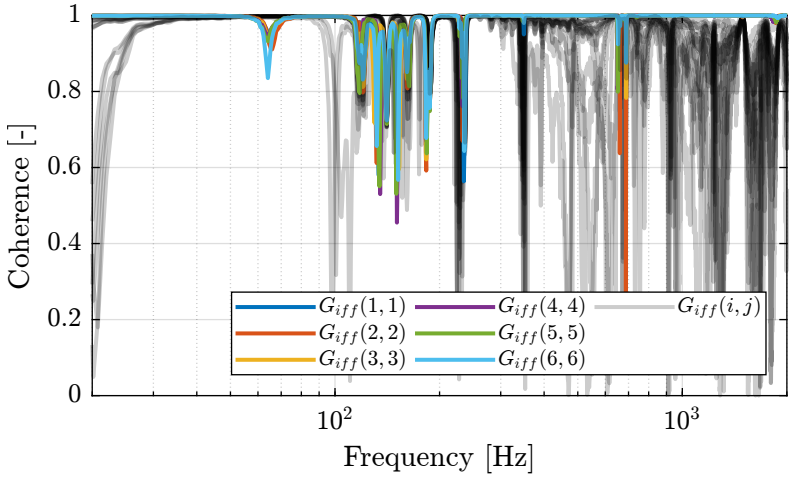

Figure 2.4: Obtained coherence for the IFF plant

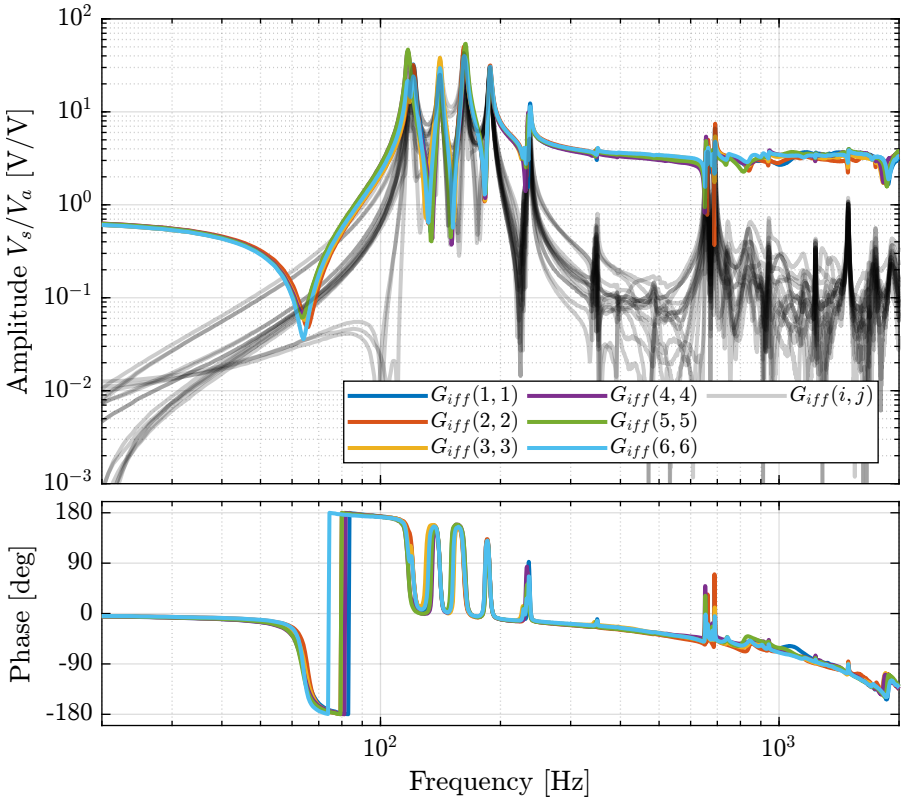

Figure 2.5: Measured FRF for the IFF plant

```matlab
save('matlab/mat/identified_plants_enc_plates.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

2.2 Comparison with the Simscape Model

In this section, the measured dynamics is compared with the dynamics estimated from the Simscape model.

2.2.1 Load measured FRF

```matlab
%% Load data
load('identified_plants_enc_plates.mat', 'f', 'Ts', 'G_iff', 'G_dvf')
```

2.2.2 Dynamics from Actuator to Force Sensors

```matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible');
```

```matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dum'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

2.2.3 Dynamics from Actuator to Encoder

```matlab
%% Initialization of the Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible');
```

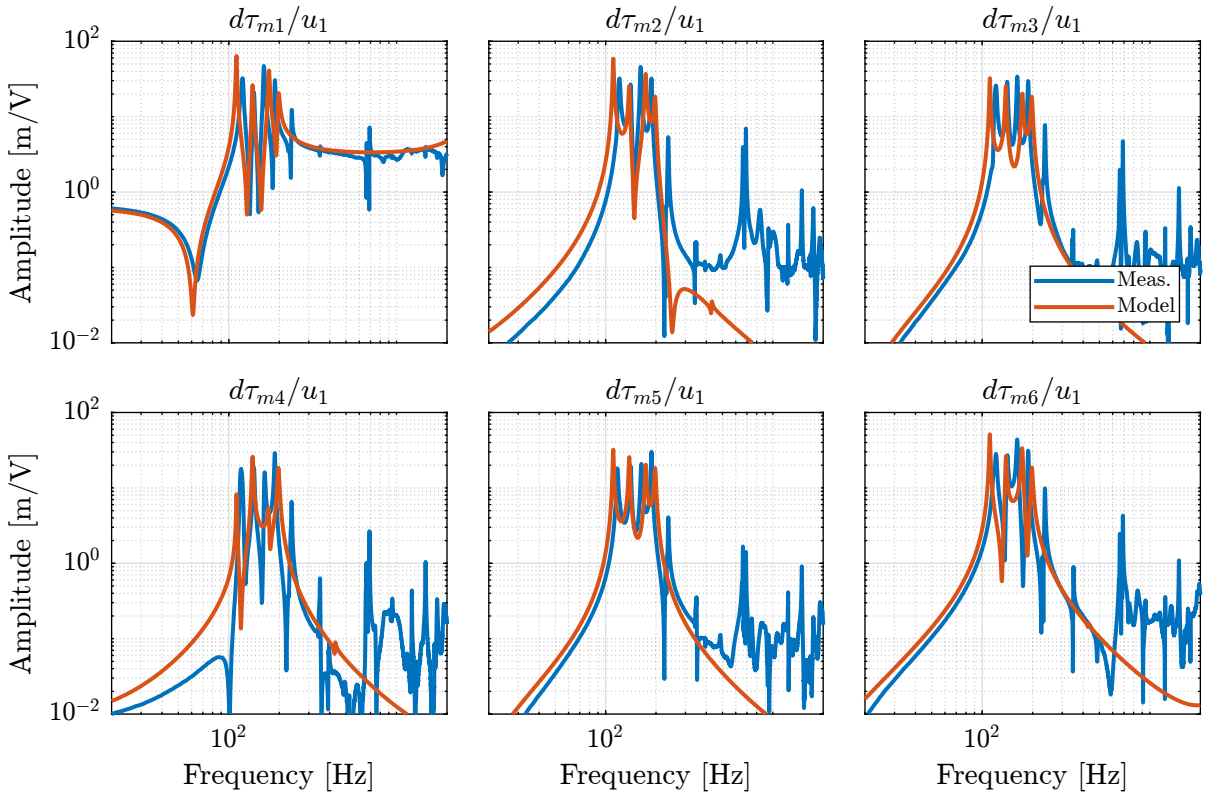

Figure 2.6: IFF Plant for the first actuator input and all the force senosrs

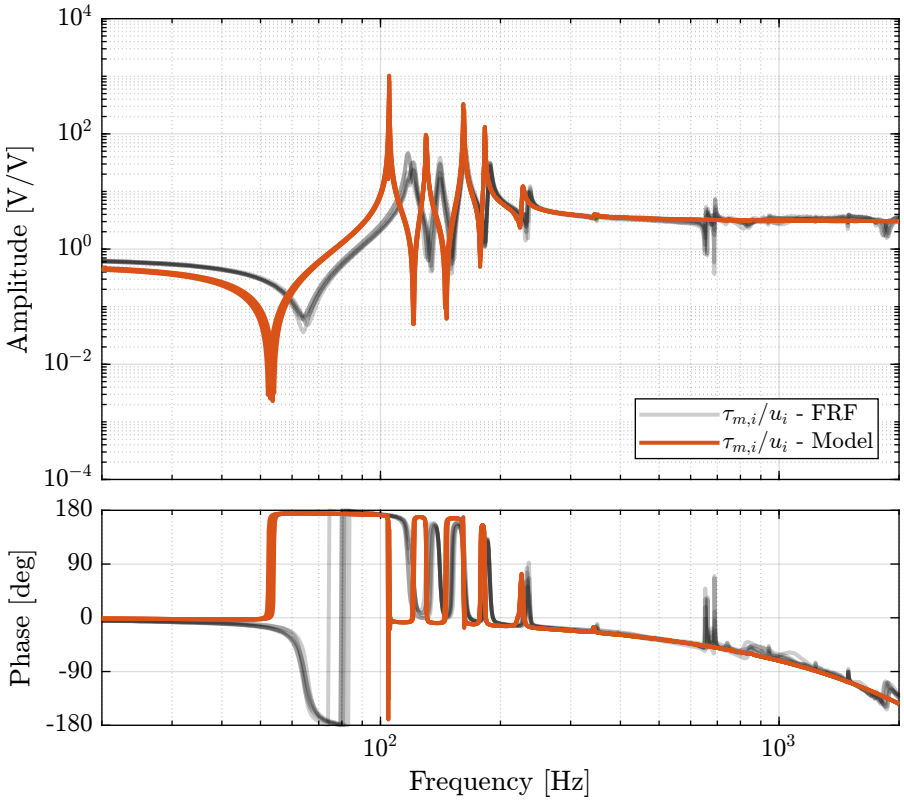

Figure 2.7: Diagonal elements of the IFF Plant

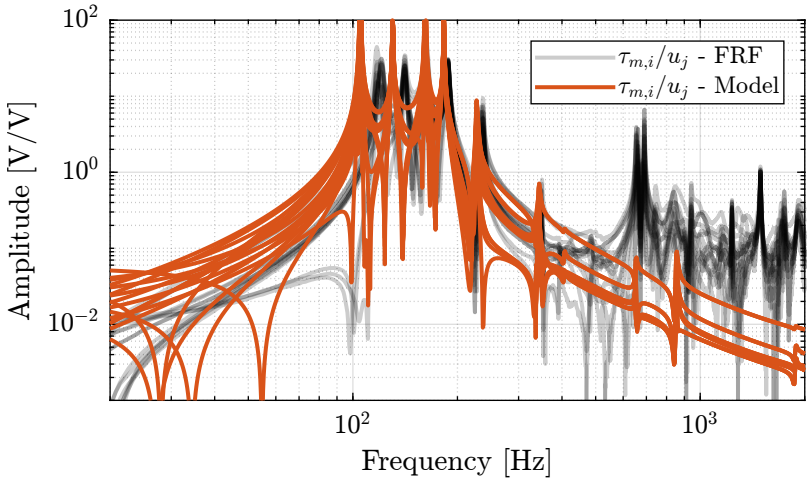

Figure 2.8: Off diagonal elements of the IFF Plant

```Matlab
%% Identify the DVF Plant (transfer function from u to dLm)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Encoders

Gdvf = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

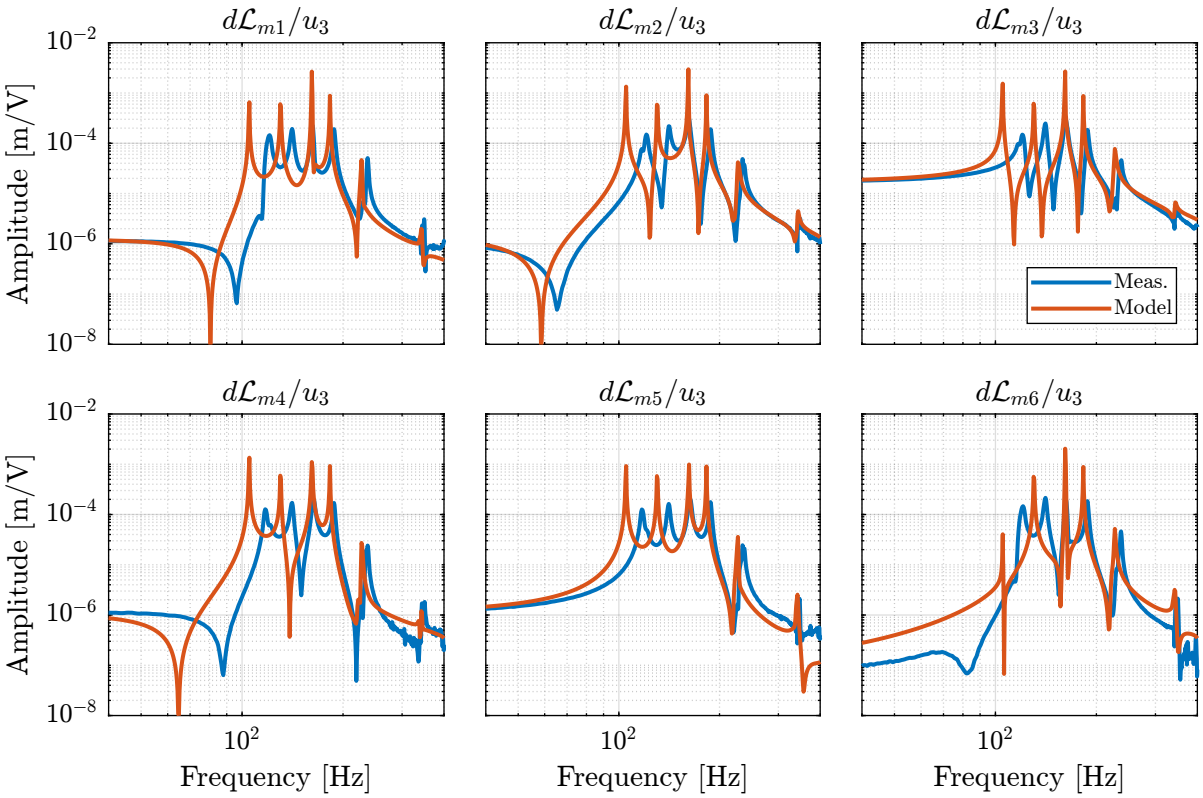

Figure 2.9: DVF Plant for the first actuator input and all the encoders

2.3 Integral Force Feedback

2.3.1 Identification of the IFF Plant

```Matlab
%% Initialize Nano-Hexapod
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', '2dof');
```

```Matlab
%% Identify the IFF Plant (transfer function from u to taum)
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');   io_i = io_i + 1; % Actuator Inputs
```

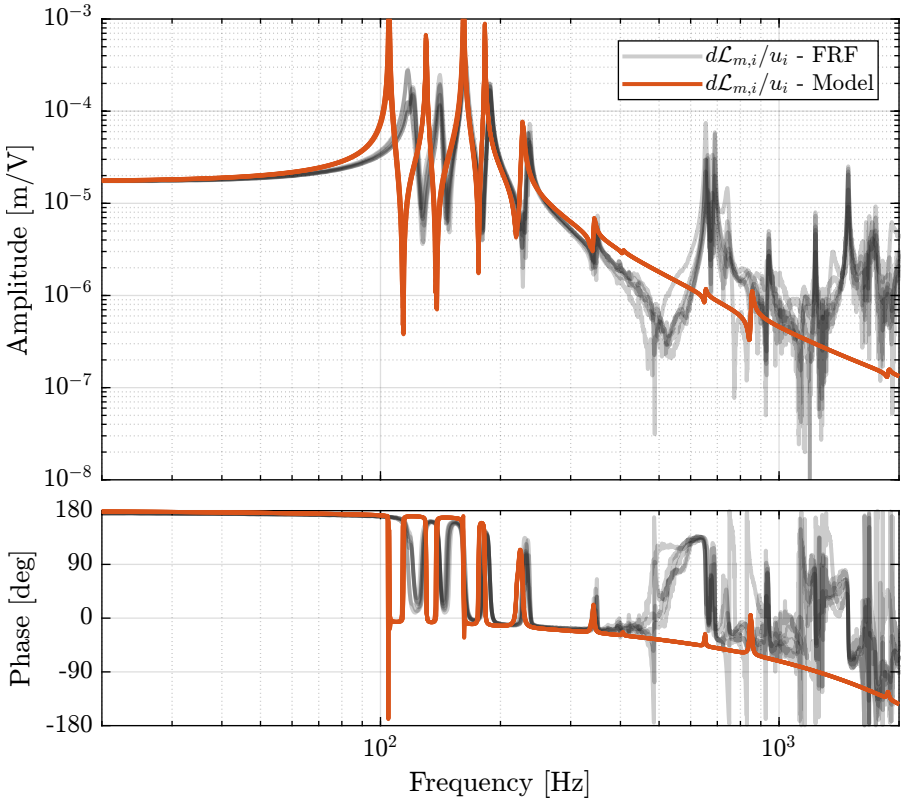

Figure 2.10: Diagonal elements of the DVF Plant

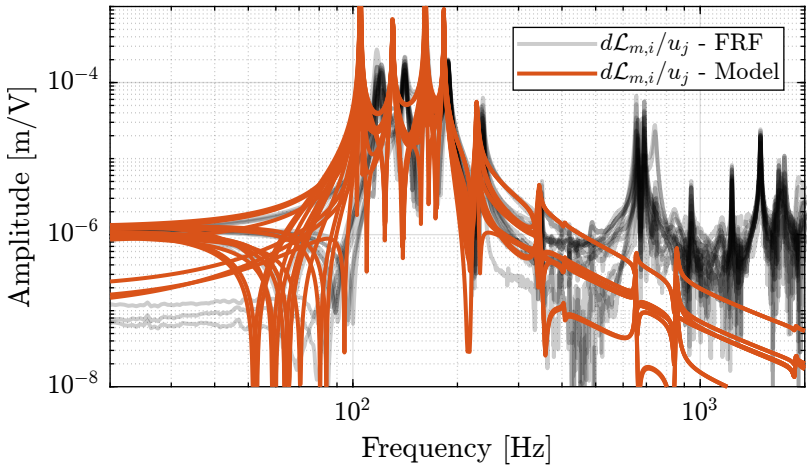

Figure 2.11: Off diagonal elements of the DVF Plant

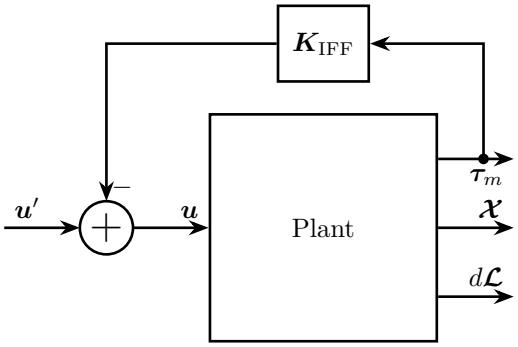

Figure 2.12: Integral Force Feedback Strategy

```
io(io_i) = linio([mdl, '/Fm'],  1, 'openoutput'); io_i = io_i + 1; % Force Sensors

Giff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

2.3.2 Effect of IFF on the plant - Simscape Model

Matlab
```
load('Kiff.mat', 'Kiff')
```

Matlab
```
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Plate Displacement (encoder)
```

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible');
```

Matlab
```
Gd_ol = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'iff');
```

Matlab

```
Gd_iff = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

Results

```
1
```

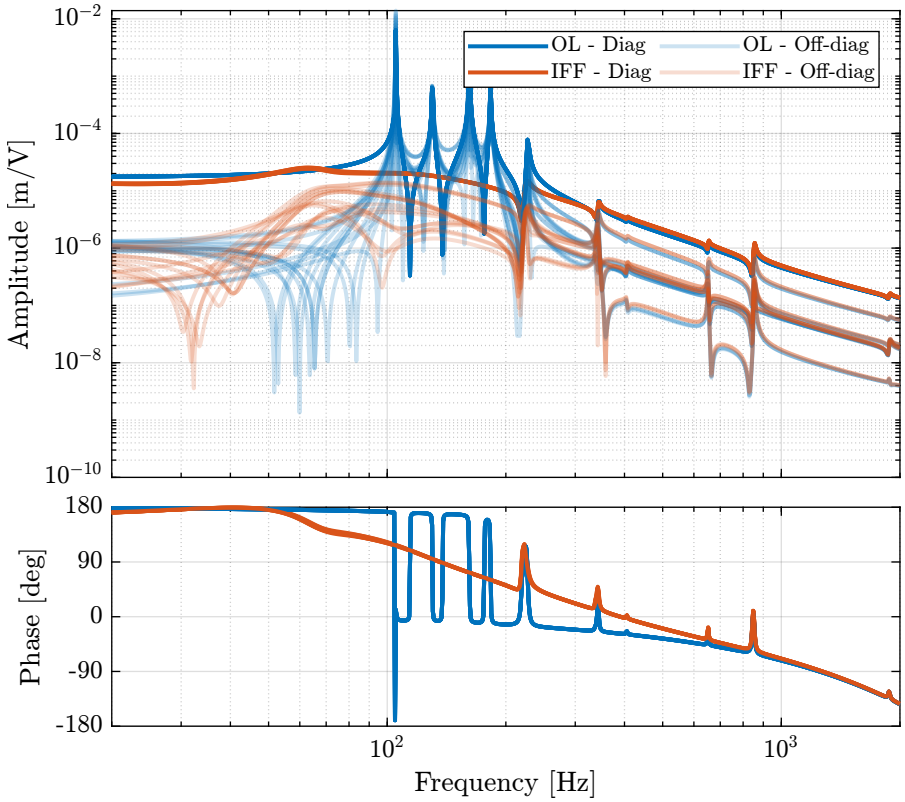

Figure 2.13: Effect of the IFF control strategy on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

2.3.3 Experimental Results - Damped Plant with Optimal gain

Let's now look at the 6×6 damped plant with the optimal gain $g = 400$.

Load Data

Matlab

```
%% Load Identification Data
meas_iff_plates = {};

for i = 1:6
    meas_iff_plates(i) = {load(sprintf('mat/frf_exc_iff_strut_%i_enc_plates_noise.mat', i), 't', 'Va', 'Vs', 'de', 'u')};
end
```

Spectral Analysis - Setup

Matlab
```
%% Setup useful variables
% Sampling Time [s]
Ts = (meas_iff_plates{1}.t(end) - (meas_iff_plates{1}.t(1)))/(length(meas_iff_plates{1}.t)-1);

% Sampling Frequency [Hz]
Fs = 1/Ts;

% Hannning Windows
win = hanning(ceil(1*Fs));

% And we get the frequency vector
[~, f] = tfestimate(meas_iff_plates{1}.Va, meas_iff_plates{1}.de, win, [], [], 1/Ts);
```

Simscape Model

Matlab
```
load('Kiff.mat', 'Kiff')
```

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'iff');
```

Matlab
```
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'], 1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Plate Displacement (encoder)
```

Matlab
```
Gd_iff_opt = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

DVF Plant

Matlab
```
%% IFF Plant
G_enc_iff_opt = zeros(length(f), 6, 6);

for i = 1:6
    G_enc_iff_opt(:,:,i) = tfestimate(meas_iff_plates{i}.Va, meas_iff_plates{i}.de, win, [], [], 1/Ts);
end
```

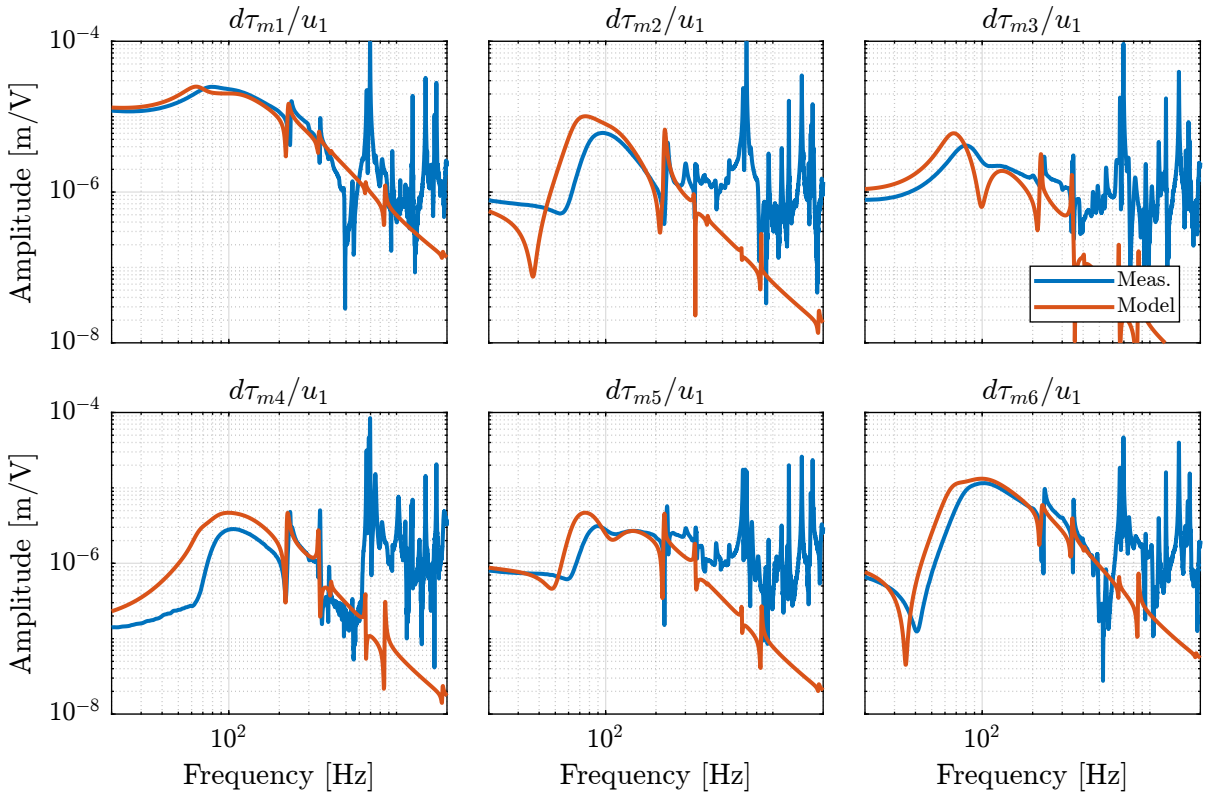

Figure 2.14: FRF from one actuator to all the encoders when the plant is damped using IFF

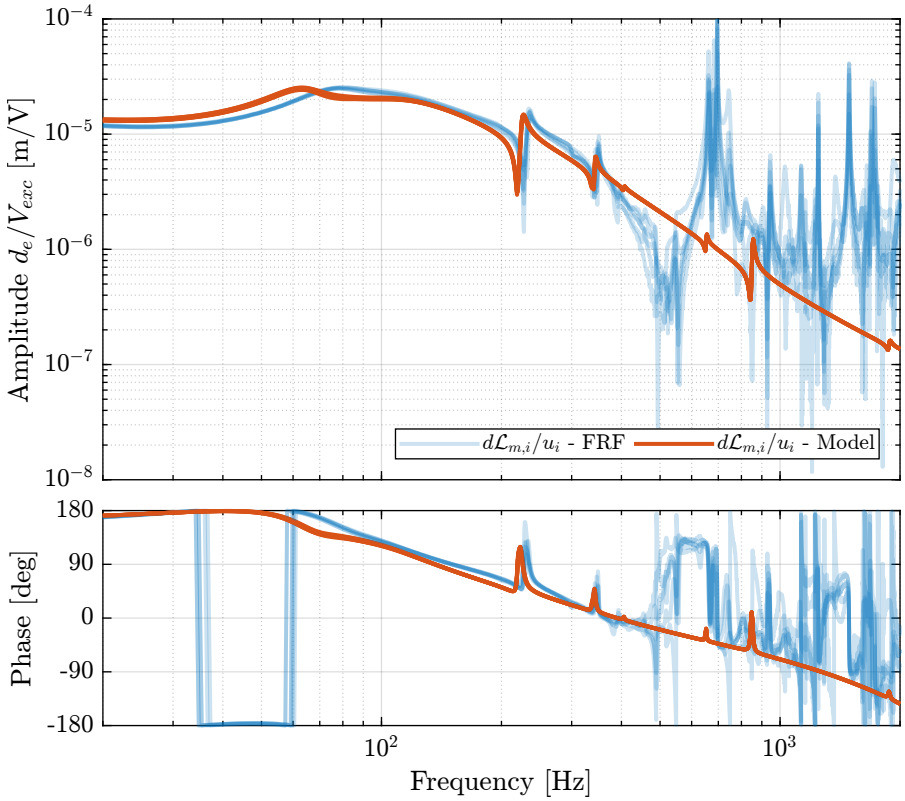

Figure 2.15: Comparison of the diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

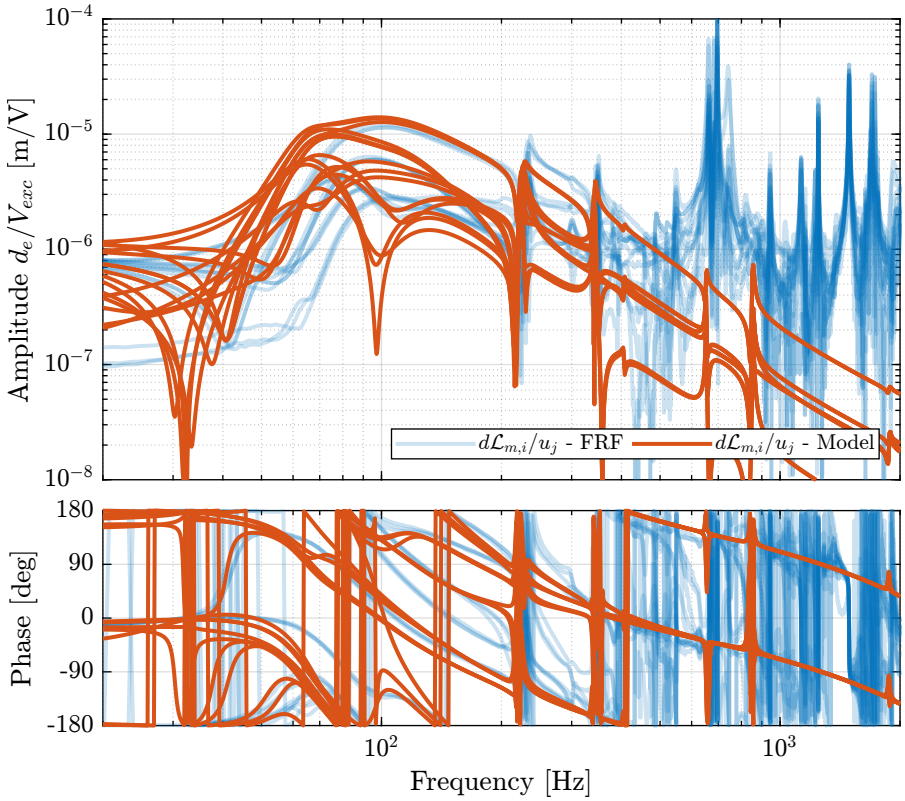

Figure 2.16: Comparison of the off-diagonal elements of the transfer functions from $\boldsymbol{u}$ to $d\boldsymbol{\mathcal{L}}_m$ with active damping (IFF) applied with an optimal gain $g = 400$

2.3.4 Effect of IFF on the plant - FRF

```Matlab
load('identified_plants_enc_plates.mat', 'f', 'G_dvf');
```

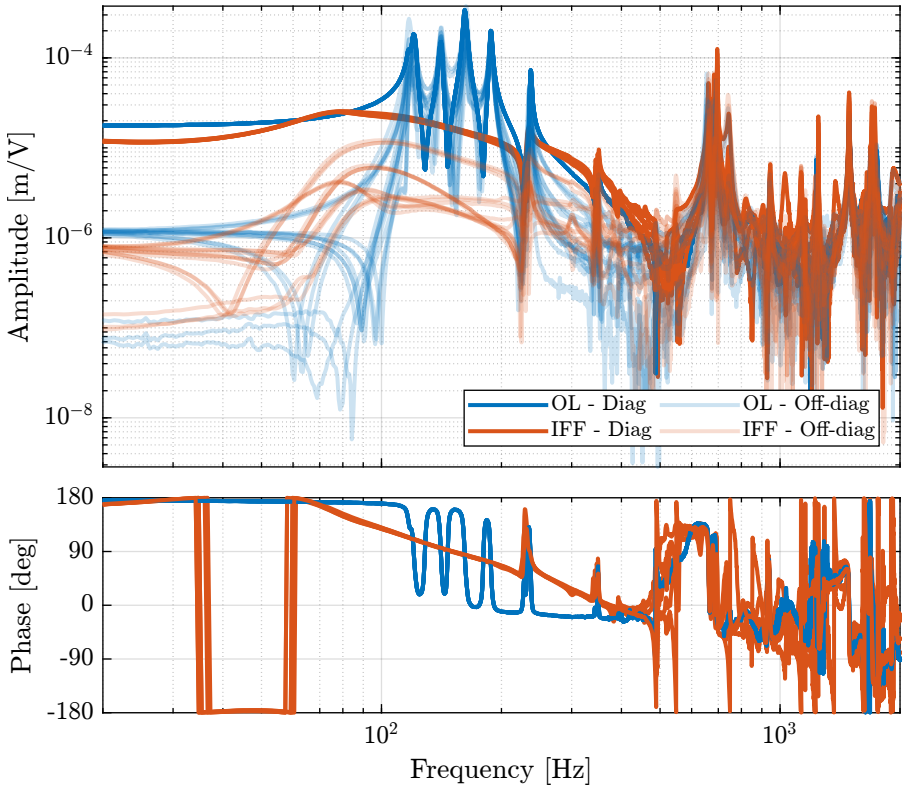

Figure 2.17: Effect of the IFF control strategy on the transfer function from $\boldsymbol{\tau}$ to $d\boldsymbol{\mathcal{L}}_m$

2.3.5 Save Damped Plant

```Matlab
save('matlab/mat/damped_plant_enc_plates.mat', 'f', 'Ts', 'G_enc_iff_opt')
```

2.4 HAC Control - Frame of the struts

In a first approximation, the Jacobian matrix can be used instead of using the inverse kinematic equations.

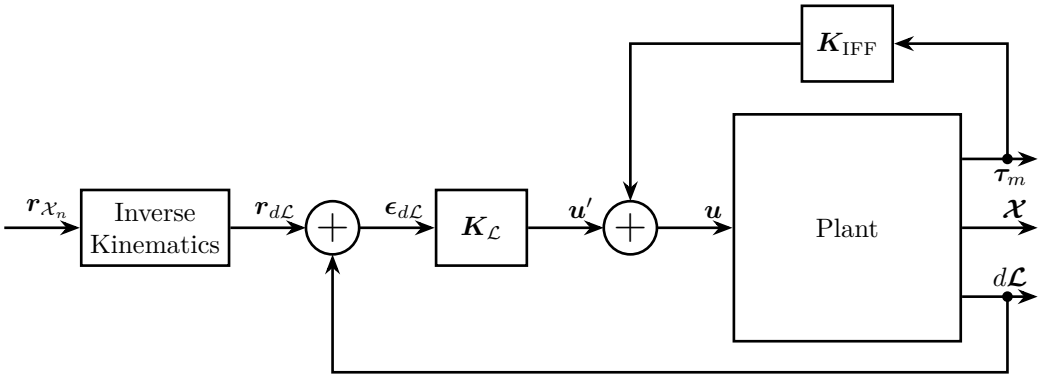

Figure 2.18: HAC-LAC: IFF + Control in the frame of the legs

2.4.1 Simscape Model

Let's start with the Simscape model and the damped plant.

Apply HAC control and verify the system is stable.

Then, try the control strategy on the real plant.

Matlab
```
load('Kiff.mat', 'Kiff')
```

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', 'flexible', ...
                                       'controller_type', 'iff');
```

Matlab
```
%% Identify the (damped) transfer function from u to dLm for different values of the IFF gain
clear io; io_i = 1;
io(io_i) = linio([mdl, '/du'],  1, 'openinput');  io_i = io_i + 1; % Actuator Inputs
io(io_i) = linio([mdl, '/dL'], 1, 'openoutput'); io_i = io_i + 1; % Plate Displacement (encoder)
```

Matlab
```
Gd_iff_opt = exp(-s*Ts)*linearize(mdl, io, 0.0, options);
```

Matlab
```
isstable(Gd_iff_opt)
```

Results
```
1
```

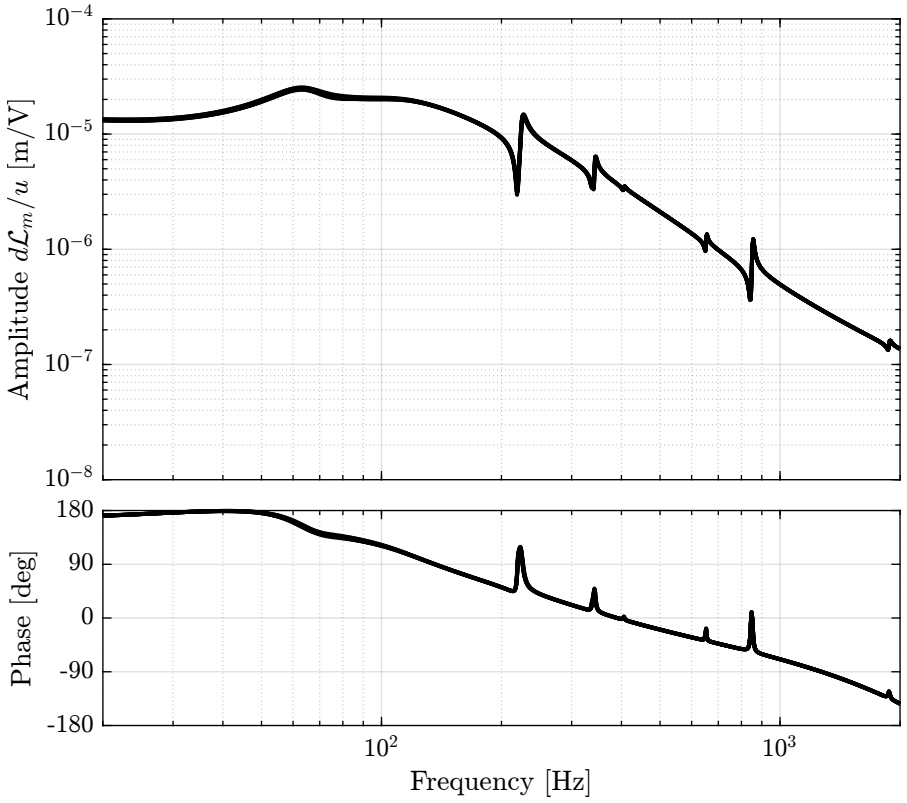

Figure 2.19: Transfer function from u to $d\mathcal{L}_m$ with IFF (diagonal elements)

2.4.2 HAC Controller

Let's try to have 100Hz bandwidth:

Matlab
```
%% Lead
a  = 2;  % Amount of phase lead / width of the phase lead / high frequency gain
wc = 2*pi*100; % Frequency with the maximum phase lead [rad/s]

H_lead = (1 + s/(wc/sqrt(a)))/(1 + s/(wc*sqrt(a)));

%% Low Pass filter
H_lpf = 1/(1 + s/2/pi/200);

%% Notch
gm = 0.02;
xi = 0.3;
wn = 2*pi*700;

H_notch = (s^2 + 2*gm*xi*wn*s + wn^2)/(s^2 + 2*xi*wn*s + wn^2);
```

Matlab
```
Khac_iff_struts = -(1/(2.87e-5)) * ... % Gain
                  H_lead * ...       % Lead
                  H_notch * ...      % Notch
                  (2*pi*100/s) * ... % Integrator
                  eye(6);            % 6x6 Diagonal
```

Matlab
```
save('matlab/mat/Khac_iff_struts.mat', 'Khac_iff_struts')
```

Matlab
```
Lhac_iff_struts = Khac_iff_struts*Gd_iff_opt;
```

2.4.3 Experimental Loop Gain

- ☐ Find a more clever way to do the multiplication

Matlab
```
L_frf = zeros(size(G_enc_iff_opt));

for i = 1:size(G_enc_iff_opt, 1)
    L_frf(i, :, :) = squeeze(G_enc_iff_opt(i,:,:))*freqresp(Khac_iff_struts, f(i), 'Hz');
end
```

2.4.4 Verification of the Stability using the Simscape model

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '4dof', ...
                                       'flex_top_type', '4dof', ...
```

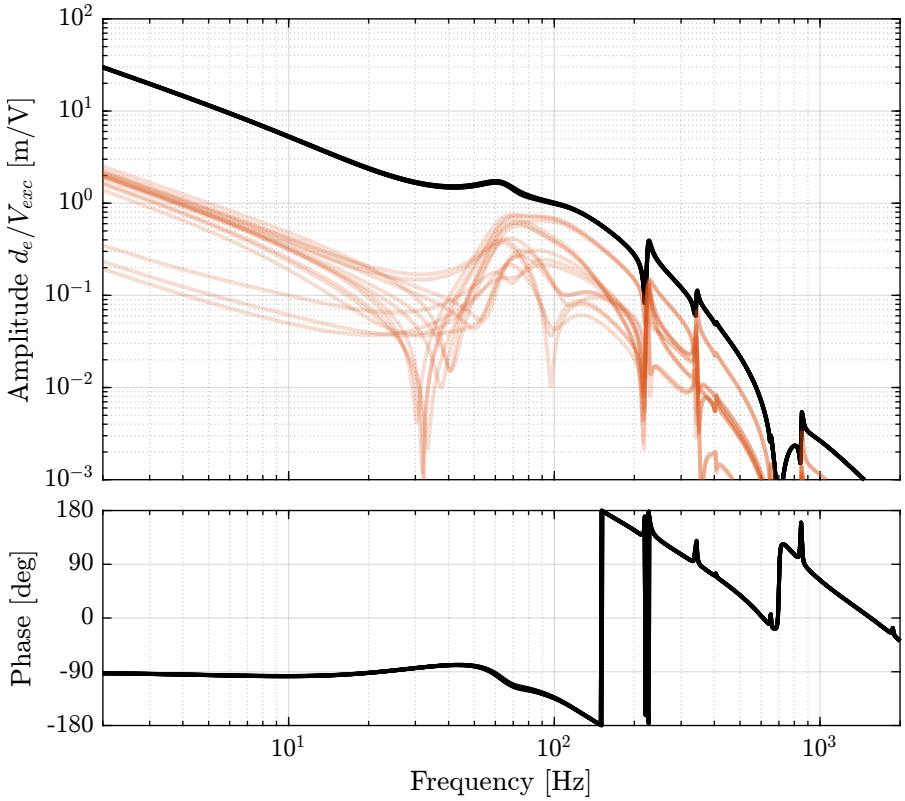

Figure 2.20: Diagonal and off-diagonal elements of the Loop gain for "HAC-IFF-Struts"

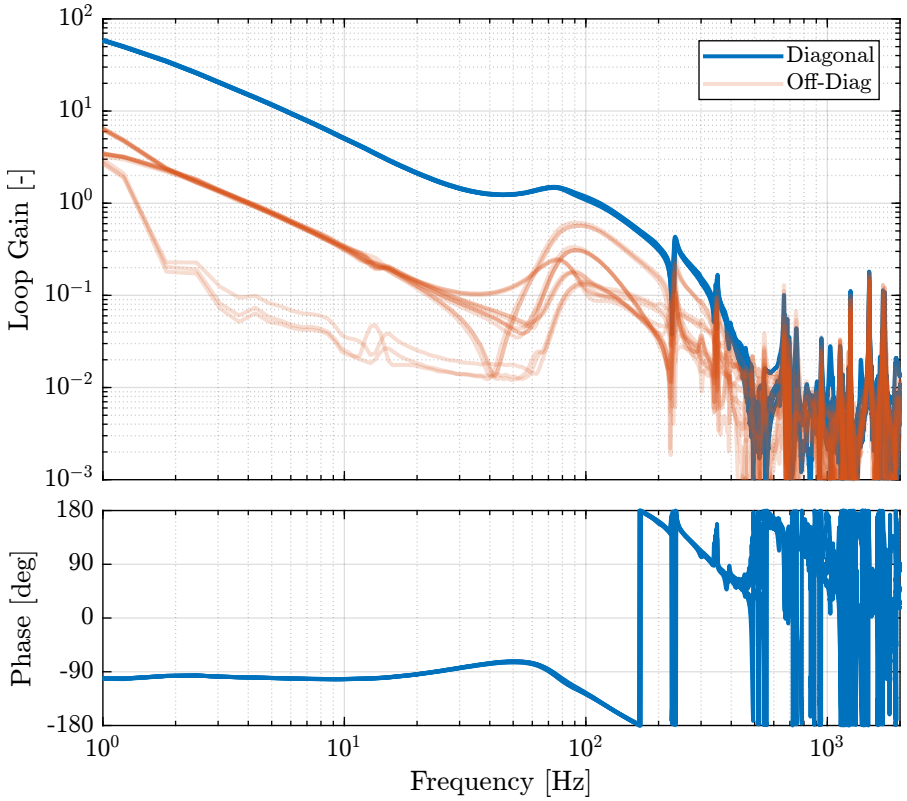

Figure 2.21: Diagonal and Off-diagonal elements of the Loop gain (experimental data)

```
                                        'motion_sensor_type', 'plates', ...
                                        'actuator_type', 'flexible', ...
                                        'controller_type', 'hac-iff-struts');
```

Matlab
```
Gd_iff_hac_opt = linearize(mdl, io, 0.0, options);
```

Matlab
```
isstable(Gd_iff_hac_opt)
```

Results
```
1
```

2.5 Reference Tracking

2.5.1 Load

Matlab
```
load('Khac_iff_struts.mat', 'Khac_iff_struts')
```

2.5.2 Y-Z Scans

Generate the Scan

Matlab
```
Rx_yz = generateYZScanTrajectory(...
    'y_tot', 4e-6, ...
    'z_tot', 8e-6, ...
    'n', 5, ...
    'Ts', 1e-3, ...
    'ti', 2, ...
    'tw', 0.5, ...
    'ty', 2, ...
    'tz', 1);
```

Matlab
```
figure;
hold on;
plot(Rx_yz(:,1), Rx_yz(:,3), ...
     'DisplayName', 'Y motion')
plot(Rx_yz(:,1), Rx_yz(:,4), ...
     'DisplayName', 'Z motion')
hold off;
xlabel('Time [s]');
ylabel('Displacement [m]');
legend('location', 'northeast');
```

Matlab
```
figure;
plot(Rx_yz(:,3), Rx_yz(:,4));
xlabel('y [m]'); ylabel('z [m]');
```

Reference Signal for the Strut lengths

Let's use the Jacobian to estimate the wanted strut length as a function of time.

Matlab
```
dL_ref = [n_hexapod.geometry.J*Rx_yz(:, 2:7)']';
```

Matlab
```
figure;
hold on;
for i=1:6
    plot(Rx_yz(:,1), dL_ref(:, i))
end
xlabel('Time [s]'); ylabel('Displacement [m]');
```

Time domain simulation with 2DoF model

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '2dof', ...
                                       'flex_top_type', '3dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', '2dof', ...
                                       'controller_type', 'hac-iff-struts');
```

Matlab
```
set_param(mdl,'StopTime', num2str(Rx_yz(end,1)))
set_param(mdl,'SimulationCommand','start')
```

Matlab
```
out.X.Data = out.X.Data - out.X.Data(1,:);
```

2.5.3 "NASS" reference path

Generate Path

Matlab
```
ref_path = [ ...
    0,  0, 0;
    0,  0, 1; % N
    0,  4, 1;
```

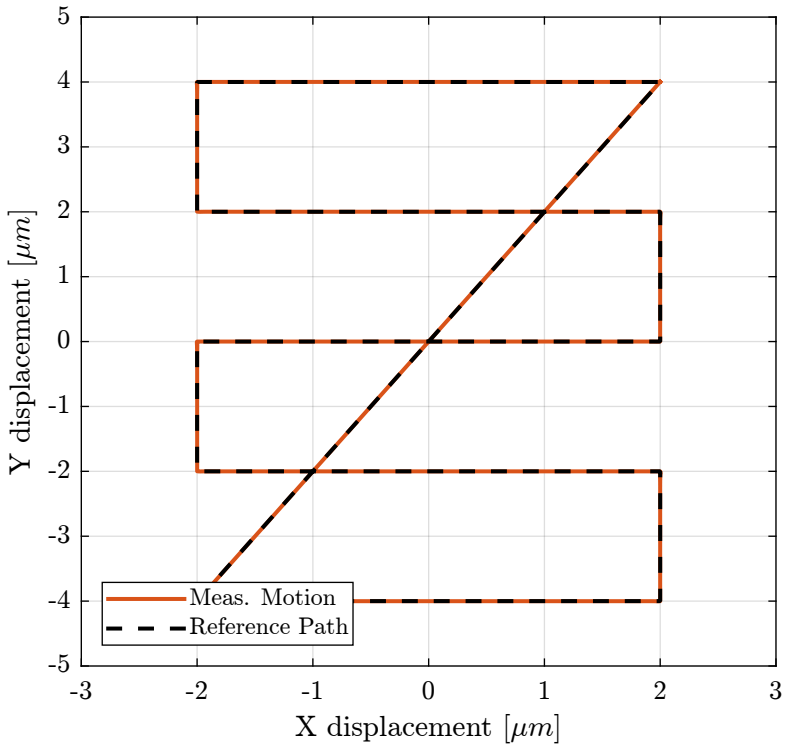

Figure 2.22: Simulated Y-Z motion

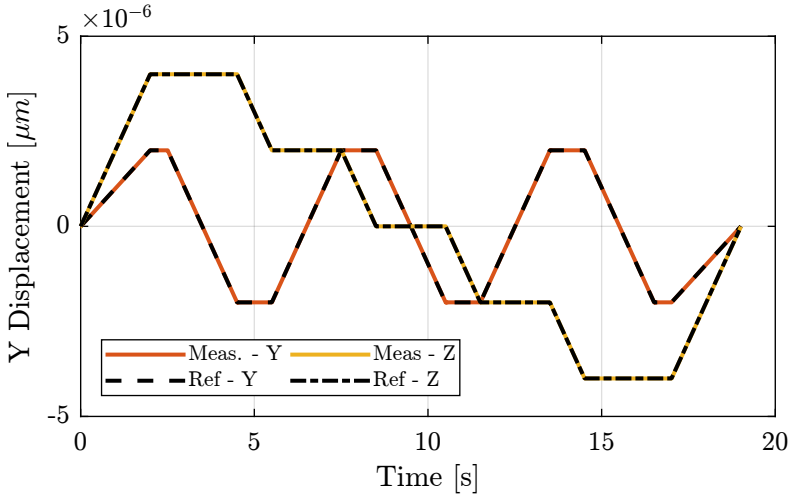

Figure 2.23: Y and Z motion as a function of time as well as the reference signals

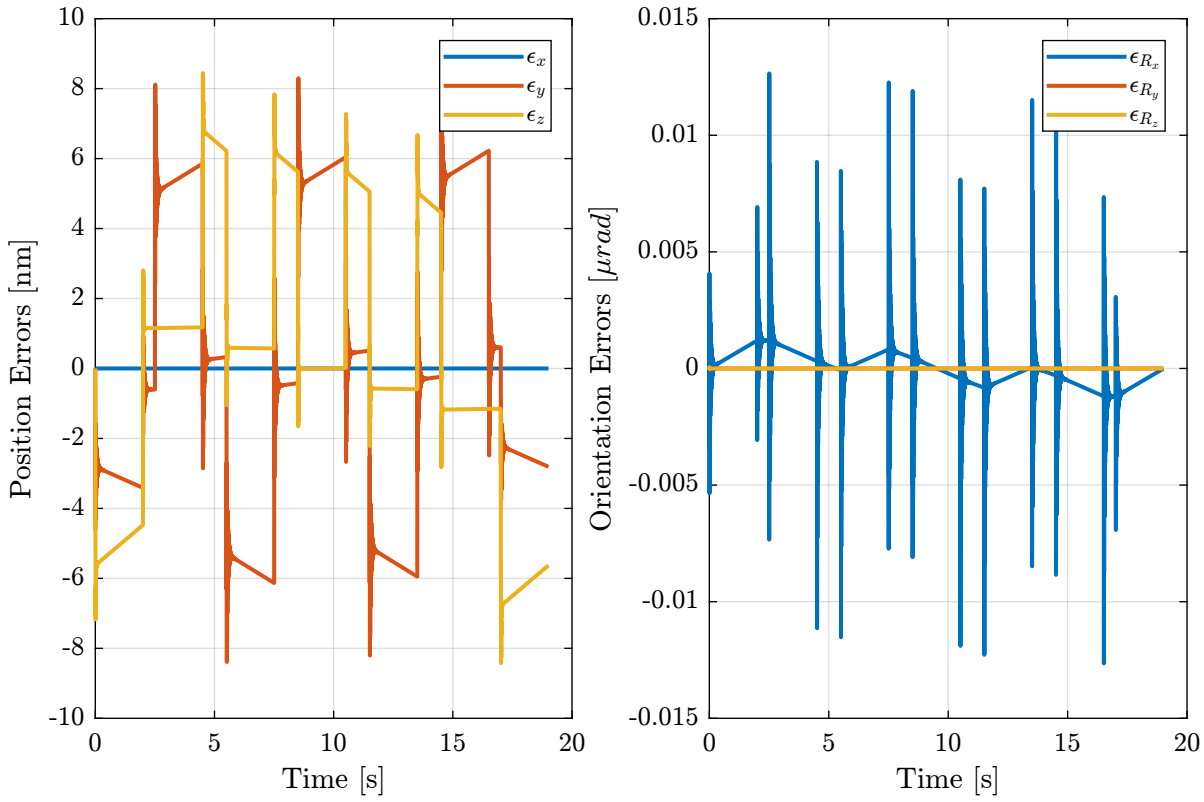

Figure 2.24: Positioning errors as a function of time

```
    3,  0, 1;
    3,  4, 1;
    3,  4, 0;
    4,  0, 0;
    4,  0, 1; % A
    4,  3, 1;
    5,  4, 1;
    6,  4, 1;
    7,  3, 1;
    7,  2, 1;
    4,  2, 1;
    4,  3, 1;
    5,  4, 1;
    6,  4, 1;
    7,  3, 1;
    7,  0, 1;
    7,  0, 0;
    8,  0, 0;
    8,  0, 1; % S
    11, 0, 1;
    11, 2, 1;
    8,  2, 1;
    8,  4, 1;
    11, 4, 1;
    11, 4, 0;
    12, 0, 0;
    12, 0, 1; % S
    15, 0, 1;
    15, 2, 1;
    12, 2, 1;
    12, 4, 1;
    15, 4, 1;
    15, 4, 0;
          ];

% Center the trajectory arround zero
ref_path = ref_path - (max(ref_path) - min(ref_path))/2;

% Define the X-Y-Z cuboid dimensions containing the trajectory
X_max = 10e-6;
Y_max =  4e-6;
Z_max =  2e-6;

ref_path = ([X_max, Y_max, Z_max]./max(ref_path)).*ref_path; % [m]
```

Matlab

```
Rx_nass = generateXYZTrajectory('points', ref_path);
```

Matlab

```
figure;
plot(1e6*Rx_nass(Rx_nass(:,4)>0, 2), 1e6*Rx_nass(Rx_nass(:,4)>0, 3), 'k.')
xlabel('X [$\mu m$]');
ylabel('Y [$\mu m$]');
axis equal;
xlim(1e6*[min(Rx_nass(:,2)), max(Rx_nass(:,2))]);
ylim(1e6*[min(Rx_nass(:,3)), max(Rx_nass(:,3))]);
```

Matlab

```
figure;
plot3(Rx_nass(:,2), Rx_nass(:,3), Rx_nass(:,4), 'k-');
xlabel('x');
ylabel('y');
zlabel('z');
```

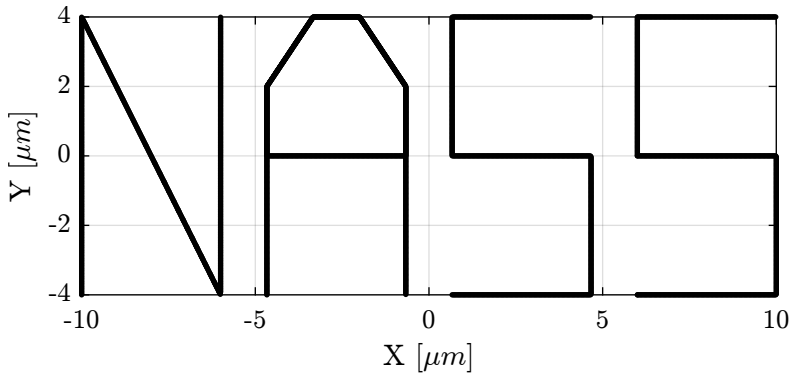

Figure 2.25: Reference path corresponding to the "NASS" acronym

Matlab
```
figure;
hold on;
plot(Rx_nass(:,1), Rx_nass(:,2));
plot(Rx_nass(:,1), Rx_nass(:,3));
plot(Rx_nass(:,1), Rx_nass(:,4));
hold off;
```

Matlab
```
figure;
scatter(Rx_nass(:,2), Rx_nass(:,3), 1, Rx_nass(:,4), 'filled')
colormap winter
```

Simscape Simulations

Matlab
```
%% Initialize the Simscape model in closed loop
n_hexapod = initializeNanoHexapodFinal('flex_bot_type', '2dof', ...
                                       'flex_top_type', '3dof', ...
                                       'motion_sensor_type', 'plates', ...
                                       'actuator_type', '2dof', ...
                                       'controller_type', 'hac-iff-struts');
```

Matlab
```
set_param(mdl,'StopTime', num2str(Rx_nass(end,1)))
set_param(mdl,'SimulationCommand','start')
```

Matlab
```
out.X.Data = out.X.Data - out.X.Data(1,:);
```

2.5.4 Save Reference paths

```Matlab
save('matlab/mat/reference_path.mat', 'Rx_yz', 'Rx_nass')
```

2.5.5 Experimental Results

2.6 Feedforward (Open-Loop) Control

2.6.1 Introduction

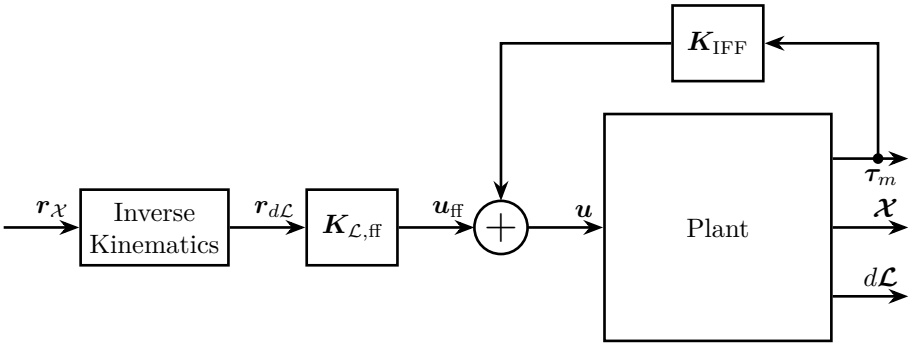

Figure 2.26: Feedforward control in the frame of the legs

2.6.2 Simple Feedforward Controller

Let's estimate the mean DC gain for the damped plant (diagonal elements:)

```Results
1.773e-05
```

The feedforward controller is then taken as the inverse of this gain (the minus sign is there manually added as it is "removed" by the `abs` function):

```Matlab
Kff_iff_L = -1/mean(diag(abs(squeeze(mean(G_enc_iff_opt(f>2 & f<4,:,:))))));
```

The open-loop gain (feedforward controller times the damped plant) is shown in Figure 2.27.

And save the feedforward controller for further use:

```Matlab
Kff_iff_L = zpk(Kff_iff_L)*eye(6);
```

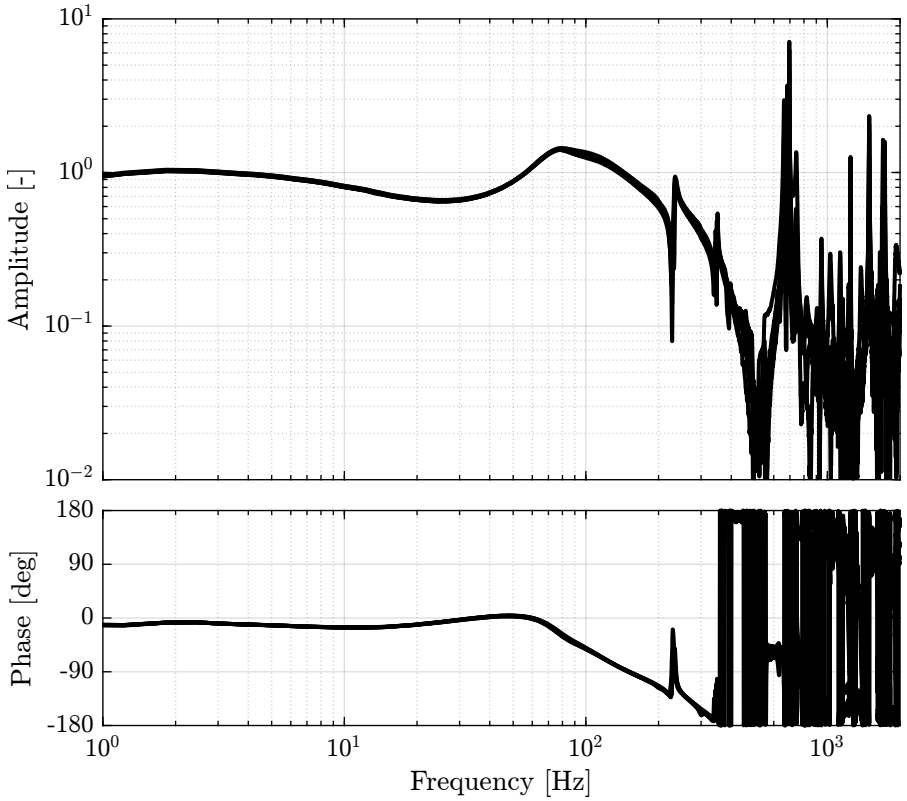

Figure 2.27: Diagonal elements of the "open loop gain"

```Matlab
save('matlab/mat/feedforward_iff.mat', 'Kff_iff_L')
```

2.6.3 Test with Simscape Model

```Matlab
load('reference_path.mat', 'Rx_yz');
```

2.7 Feedback/Feedforward control in the frame of the struts

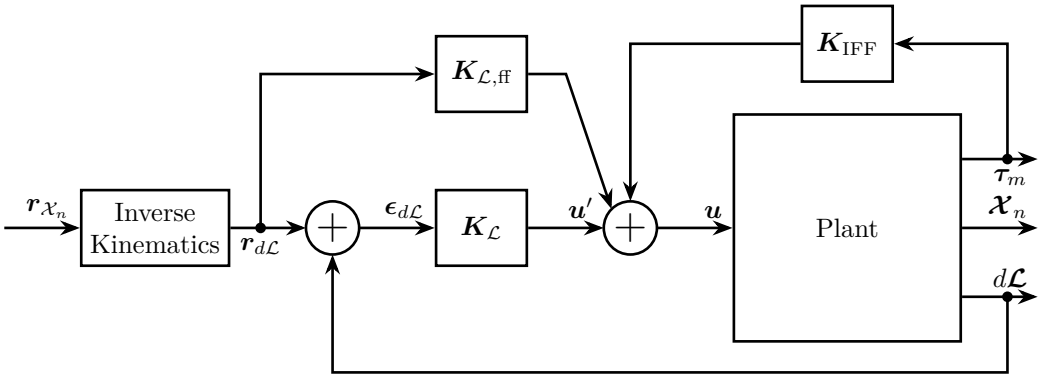

Figure 2.28: Feedback/Feedforward control in the frame of the legs

3 Functions

3.1 `generateXYZTrajectory`

Function description

Matlab
```
function [ref] = generateXYZTrajectory(args)
% generateXYZTrajectory -
%
% Syntax: [ref] = generateXYZTrajectory(args)
%
% Inputs:
%    - args
%
% Outputs:
%    - ref - Reference Signal
```

Optional Parameters

Matlab
```
arguments
    args.points double {mustBeNumeric} = zeros(2, 3) % [m]

    args.ti    (1,1) double {mustBeNumeric, mustBePositive} = 1 % Time to go to first point and after last point [s]
    args.tw    (1,1) double {mustBeNumeric, mustBePositive} = 0.5 % Time wait between each point [s]
    args.tm    (1,1) double {mustBeNumeric, mustBePositive} = 1 % Motion time between points [s]

    args.Ts    (1,1) double {mustBeNumeric, mustBePositive} = 1e-3 % Sampling Time [s]
end
```

Initialize Time Vectors

Matlab
```
time_i = 0:args.Ts:args.ti;
time_w = 0:args.Ts:args.tw;
time_m = 0:args.Ts:args.tm;
```

XYZ Trajectory

```Matlab
% Go to initial position
xyz = (args.points(1,:))'*(time_i/args.ti);

% Wait
xyz = [xyz, xyz(:,end).*ones(size(time_w))];

% Scans
for i = 2:size(args.points, 1)
    % Go to next point
    xyz = [xyz, xyz(:,end) + (args.points(i,:)' - xyz(:,end))*(time_m/args.tm)];
    % Wait a litle bit
    xyz = [xyz, xyz(:,end).*ones(size(time_w))];
end

% End motion
xyz = [xyz, xyz(:,end) - xyz(:,end)*(time_i/args.ti)];
```

Reference Signal

```Matlab
t = 0:args.Ts:args.Ts*(length(xyz) - 1);
```

```Matlab
ref = zeros(length(xyz), 7);

ref(:, 1) = t;
ref(:, 2:4) = xyz';
```

3.2 `generateYZScanTrajectory`

Function description

```Matlab
function [ref] = generateYZScanTrajectory(args)
% generateYZScanTrajectory -
%
% Syntax: [ref] = generateYZScanTrajectory(args)
%
% Inputs:
%    - args
%
% Outputs:
%    - ref - Reference Signal
```

Optional Parameters

```Matlab
arguments
    args.y_tot (1,1) double {mustBeNumeric} = 10e-6 % [m]
```

```
    args.z_tot (1,1) double {mustBeNumeric} = 10e-6 % [m]

    args.n     (1,1) double {mustBeInteger, mustBePositive} = 10 % [-]

    args.Ts    (1,1) double {mustBeNumeric, mustBePositive} = 1e-4 % [s]

    args.ti    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.tw    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.ty    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
    args.tz    (1,1) double {mustBeNumeric, mustBePositive} = 1 % [s]
end
```

Initialize Time Vectors

Matlab
```
time_i = 0:args.Ts:args.ti;
time_w = 0:args.Ts:args.tw;
time_y = 0:args.Ts:args.ty;
time_z = 0:args.Ts:args.tz;
```

Y and Z vectors

Matlab
```
% Go to initial position
y = (time_i/args.ti)*(args.y_tot/2);

% Wait
y = [y, y(end)*ones(size(time_w))];

% Scans
for i = 1:args.n
    if mod(i,2) == 0
        y = [y, -(args.y_tot/2) + (time_y/args.ty)*args.y_tot];
    else
        y = [y,  (args.y_tot/2) - (time_y/args.ty)*args.y_tot];
    end

    if i < args.n
        y = [y, y(end)*ones(size(time_z))];
    end
end

% Wait a litle bit
y = [y, y(end)*ones(size(time_w))];

% End motion
y = [y, y(end) - y(end)*time_i/args.ti];
```

Matlab
```
% Go to initial position
z = (time_i/args.ti)*(args.z_tot/2);

% Wait
z = [z, z(end)*ones(size(time_w))];

% Scans
for i = 1:args.n
    z = [z, z(end)*ones(size(time_y))];

    if i < args.n
        z = [z, z(end) - (time_z/args.tz)*args.z_tot/(args.n-1)];
```

```
    end
end

% Wait a litle bit
z = [z, z(end)*ones(size(time_w))];

% End motion
z = [z, z(end) - z(end)*time_i/args.ti];
```

Reference Signal

Matlab

```
t = 0:args.Ts:args.Ts*(length(y) - 1);
```

Matlab

```
ref = zeros(length(y), 7);

ref(:, 1) = t;
ref(:, 3) = y;
ref(:, 4) = z;
```

3.3 `getTransformationMatrixAcc`

Function description

Matlab

```
function [M] = getTransformationMatrixAcc(Opm, Osm)
% getTransformationMatrixAcc -
%
% Syntax: [M] = getTransformationMatrixAcc(Opm, Osm)
%
% Inputs:
%    - Opm - Nx3 (N = number of accelerometer measurements) X,Y,Z position of accelerometers
%    - Opm - Nx3 (N = number of accelerometer measurements) Unit vectors representing the accelerometer orientation
%
% Outputs:
%    - M - Transformation Matrix
```

Transformation matrix from motion of the solid body to accelerometer measurements

Let's try to estimate the x-y-z acceleration of any point of the solid body from the acceleration/angular acceleration of the solid body expressed in $\{O\}$. For any point p_i of the solid body (corresponding to an accelerometer), we can write:

$$\begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + p_i \times \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{3.1}$$

We can write the cross product as a matrix product using the skew-symmetric transformation:

$$\begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & p_{i,z} & -p_{i,y} \\ -p_{i,z} & 0 & p_{i,x} \\ p_{i,y} & -p_{i,x} & 0 \end{bmatrix}}_{P_{i,[\times]}} \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{3.2}$$

If we now want to know the (scalar) acceleration a_i of the point p_i in the direction of the accelerometer direction $\hat{s}_i$, we can just project the 3d acceleration on $\hat{s}_i$:

$$a_i = \hat{s}_i^T \cdot \begin{bmatrix} a_{i,x} \\ a_{i,y} \\ a_{i,z} \end{bmatrix} = \hat{s}_i^T \cdot \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} + \left(\hat{s}_i^T \cdot P_{i,[\times]}\right) \cdot \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \tag{3.3}$$

Which is equivalent as a simple vector multiplication:

$$a_i = \begin{bmatrix} \hat{s}_i^T & \hat{s}_i^T \cdot P_{i,[\times]} \end{bmatrix} \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} \hat{s}_i^T & \hat{s}_i^T \cdot P_{i,[\times]} \end{bmatrix} {}^O\vec{x} \tag{3.4}$$

And finally we can combine the 6 (line) vectors for the 6 accelerometers to write that in a matrix form. We obtain Eq. (3.5).

Important

The transformation from solid body acceleration ${}^O\vec{x}$ from sensor measured acceleration $\vec{a}$ is:

$$\vec{a} = \underbrace{\begin{bmatrix} \hat{s}_1^T & \hat{s}_1^T \cdot P_{1,[\times]} \\ \vdots & \vdots \\ \hat{s}_6^T & \hat{s}_6^T \cdot P_{6,[\times]} \end{bmatrix}}_{M} {}^O\vec{x} \tag{3.5}$$

with $\hat{s}_i$ the unit vector representing the measured direction of the i'th accelerometer expressed in frame $\{O\}$ and $P_{i,[\times]}$ the skew-symmetric matrix representing the cross product of the position of the i'th accelerometer expressed in frame $\{O\}$.

Let's define such matrix using matlab:

Matlab

```
M = zeros(length(Opm), 6);

for i = 1:length(Opm)
    Ri = [0,          Opm(3,i), -Opm(2,i);
          -Opm(3,i),  0,         Opm(1,i);
           Opm(2,i), -Opm(1,i),  0];
    M(i, 1:3) = Osm(:,i)';
    M(i, 4:6) = Osm(:,i)'*Ri;
end
```

```Matlab
end
```

3.4 `getJacobianNanoHexapod`

Function description

```Matlab
function [J] = getJacobianNanoHexapod(Hbm)
% getJacobianNanoHexapod -
%
% Syntax: [J] = getJacobianNanoHexapod(Hbm)
%
% Inputs:
%    - Hbm - Height of {B} w.r.t. {M} [m]
%
% Outputs:
%    - J - Jacobian Matrix
```

Transformation matrix from motion of the solid body to accelerometer measurements

```Matlab
Fa = [[-86.05,  -74.78, 22.49],
      [ 86.05,  -74.78, 22.49],
      [ 107.79, -37.13, 22.49],
      [ 21.74,   111.91, 22.49],
      [-21.74,   111.91, 22.49],
      [-107.79, -37.13, 22.49]]'*1e-3; % Ai w.r.t. {F} [m]

Mb = [[-28.47, -106.25, -22.50],
      [ 28.47, -106.25, -22.50],
      [ 106.25, 28.47,  -22.50],
      [ 77.78,  77.78,  -22.50],
      [-77.78,  77.78,  -22.50],
      [-106.25, 28.47,  -22.50]]'*1e-3; % Bi w.r.t. {M} [m]

H = 95e-3; % Stewart platform height [m]
Fb = Mb + [0; 0; H]; % Bi w.r.t. {F} [m]

si = Fb - Fa;
si = si./vecnorm(si); % Normalize

Bb = Mb - [0; 0; Hbm];

J = [si', cross(Bb, si)'];
```