Nano-Hexapod - Test Bench

Dehaeze Thomas

June 9, 2021

Contents

Note

Here are the documentation of the equipment used for this test bench:

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

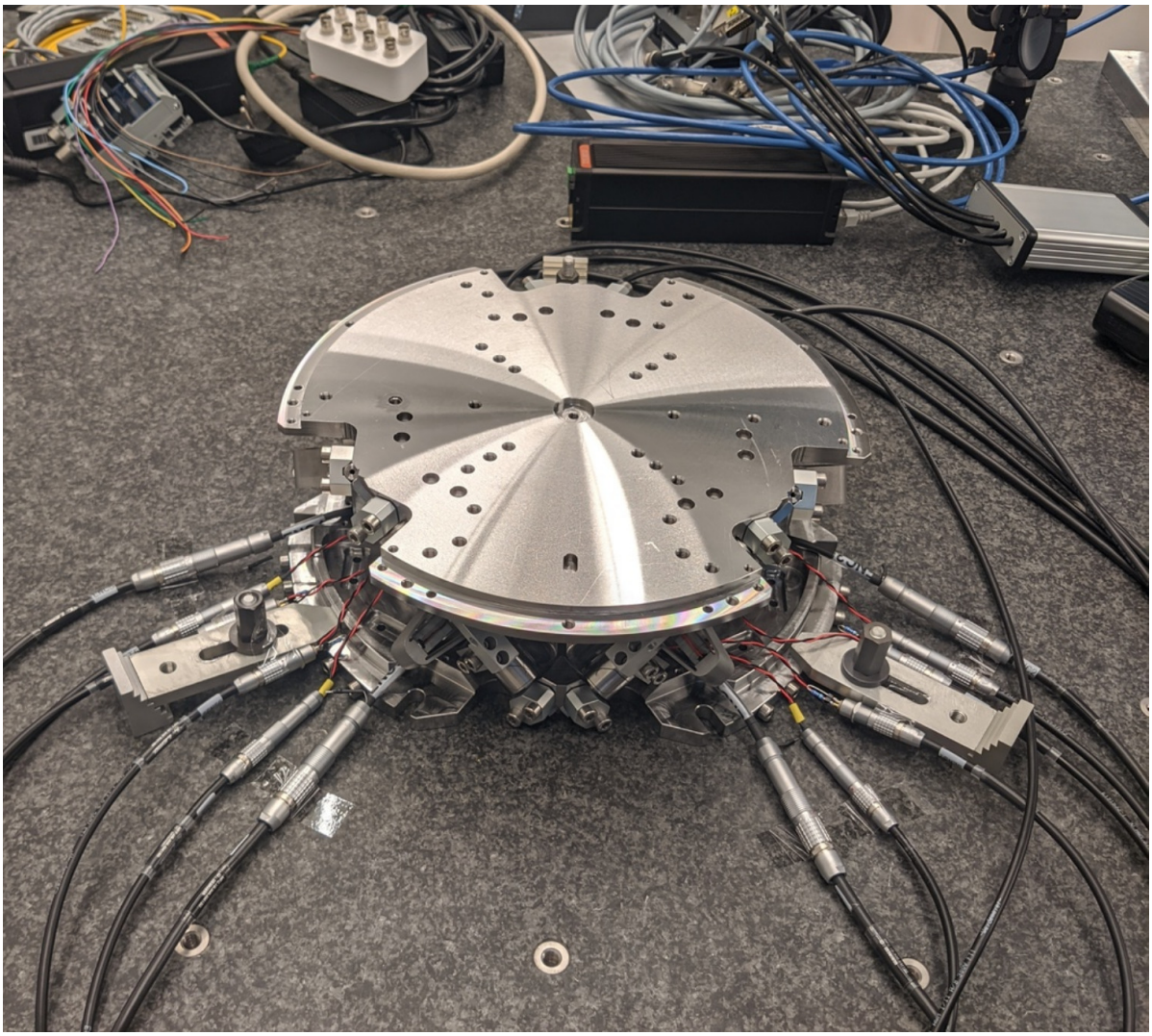

Figure 0.1: Nano-Hexapod

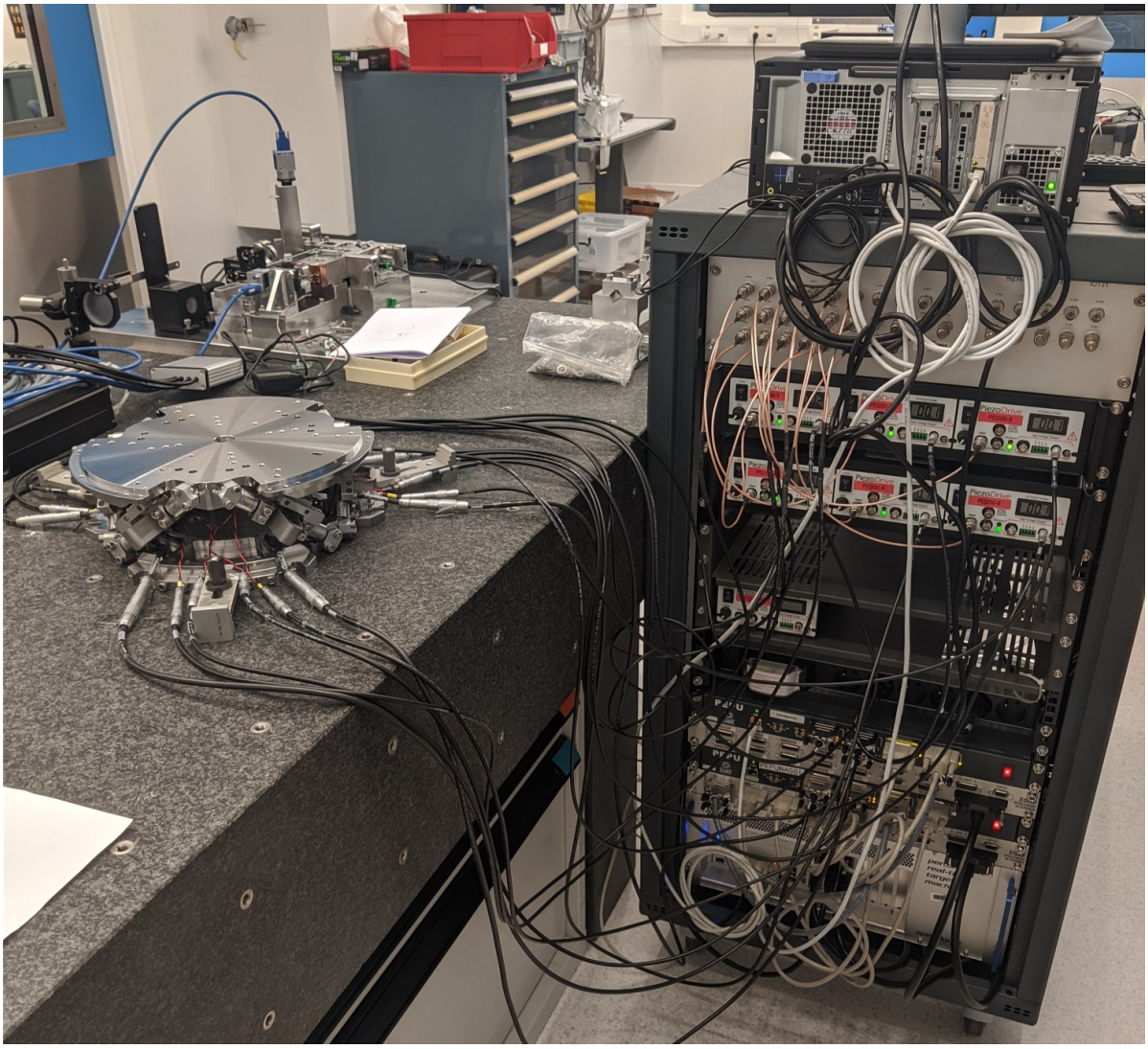

Figure 0.2: Nano-Hexapod and the control electronics

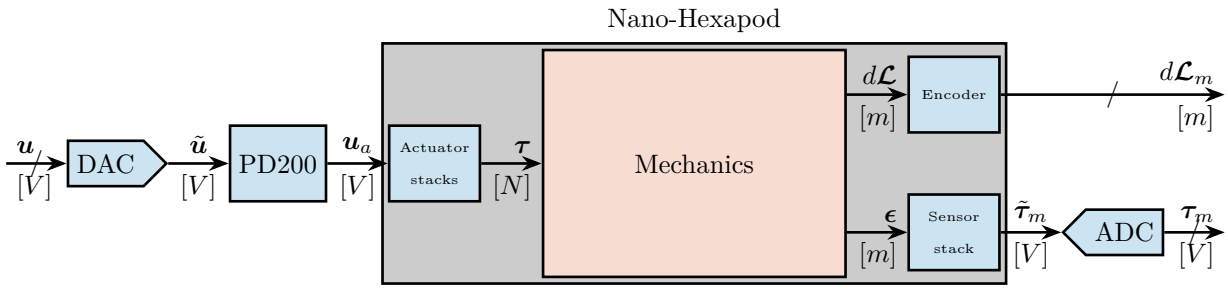

Figure 0.3: Block diagram of the system with named signals

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`	$d\boldsymbol{\mathcal{X}}_m$	$d\mathcal{X}_{m,i}$

1 Encoders fixed to the Struts

1.1 Introduction

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

1.2 Load Data

```Matlab
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
```

1.3 Spectral Analysis - Setup

```Matlab
% 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.

```Matlab
[~, f] = tfestimate(meas_data_lf{1}.Va, meas_data_lf{1}.de, win, [], [], 1/Ts);
```

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

1.4 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_lf = zeros(length(f), 6, 6);
coh_dvf_hf = zeros(length(f), 6, 6);

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

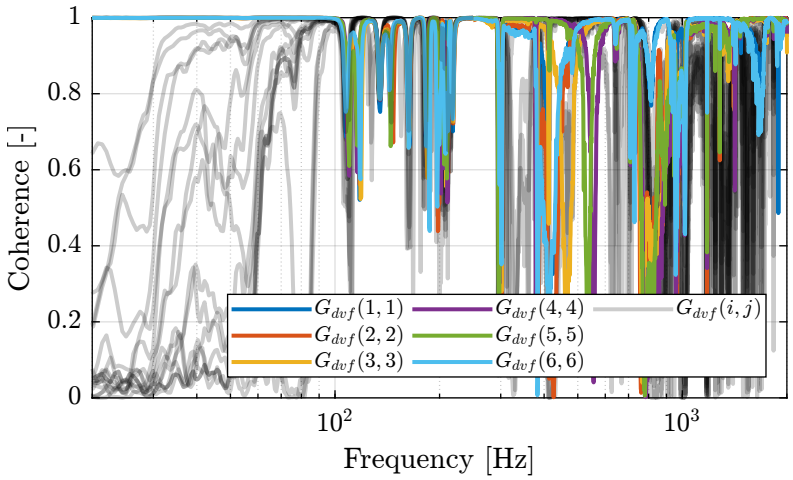

Figure 1.1: Obtained coherence for the DVF plant

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

Matlab
```
%% DVF Plant
G_dvf_lf = zeros(length(f), 6, 6);
G_dvf_hf = zeros(length(f), 6, 6);

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

1.5 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
coh_iff_lf = zeros(length(f), 6, 6);
coh_iff_hf = zeros(length(f), 6, 6);
```

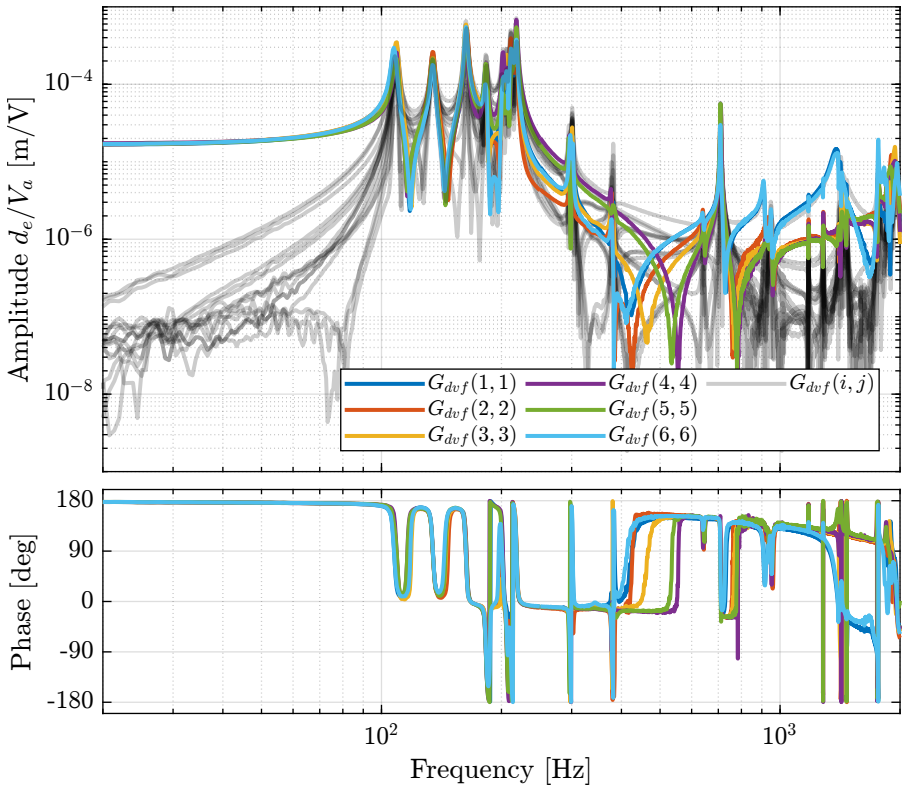

Figure 1.2: Measured FRF for the DVF plant

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

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

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

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

1.6 Jacobian

The Jacobian is used to transform the excitation force in the cartesian frame as well as the displacements.

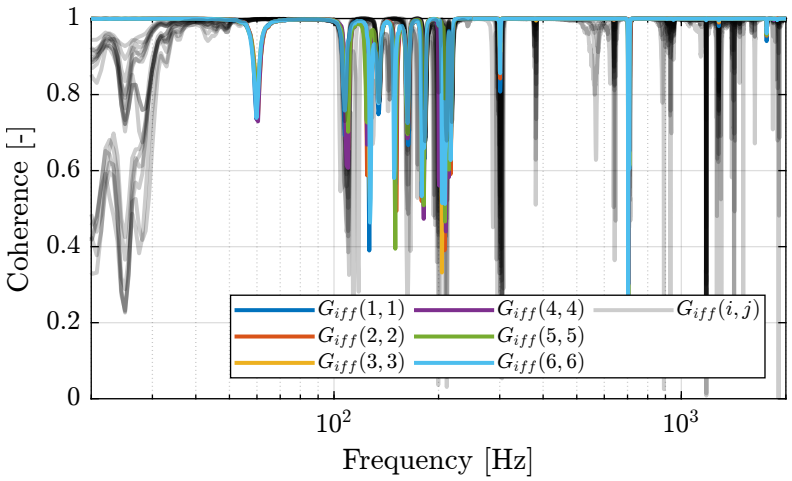

Figure 1.3: Obtained coherence for the IFF plant

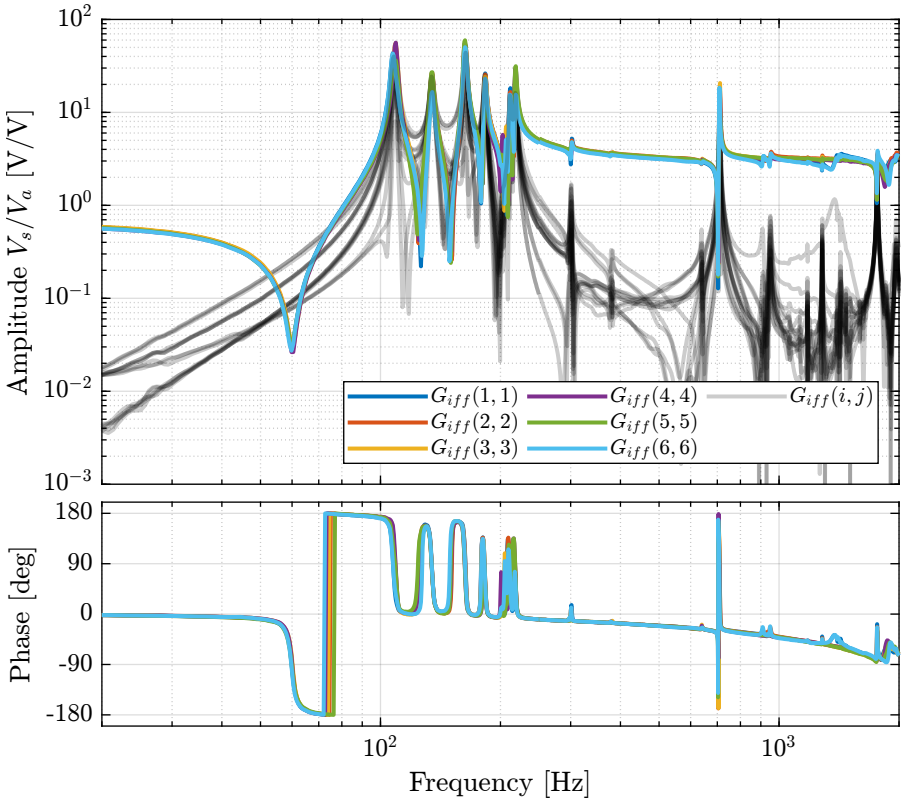

Figure 1.4: Measured FRF for the IFF plant

Consider the plant shown in Figure 1.5 with:

- τ the 6 input voltages (going to the PD200 amplifier and then to the APA)
- $d\mathcal{L}$ the relative motion sensor outputs (encoders)
- $\boldsymbol{\tau}_m$ the generated voltage of the force sensor stacks
- J_a and J_s the Jacobians for the actuators and sensors

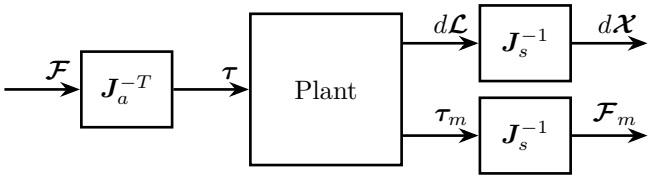

Figure 1.5: Plant in the cartesian Frame

First, we load the Jacobian matrix (same for the actuators and sensors).

```Matlab
load('jacobian.mat', 'J');
```

1.6.1 DVF Plant

The transfer function from $\mathcal{F}$ to $d\mathcal{X}$ is computed and shown in Figure 1.6.

```Matlab
G_dvf_J_lf = permute(pagemtimes(inv(J), pagemtimes(permute(G_dvf_lf, [2 3 1]), inv(J'))), [3 1 2]);
G_dvf_J_hf = permute(pagemtimes(inv(J), pagemtimes(permute(G_dvf_hf, [2 3 1]), inv(J'))), [3 1 2]);
```

1.6.2 IFF Plant

The transfer function from $\mathcal{F}$ to $\mathcal{F}_m$ is computed and shown in Figure 1.7.

```Matlab
G_iff_J_lf = permute(pagemtimes(inv(J), pagemtimes(permute(G_iff_lf, [2 3 1]), inv(J'))), [3 1 2]);
G_iff_J_hf = permute(pagemtimes(inv(J), pagemtimes(permute(G_iff_hf, [2 3 1]), inv(J'))), [3 1 2]);
```

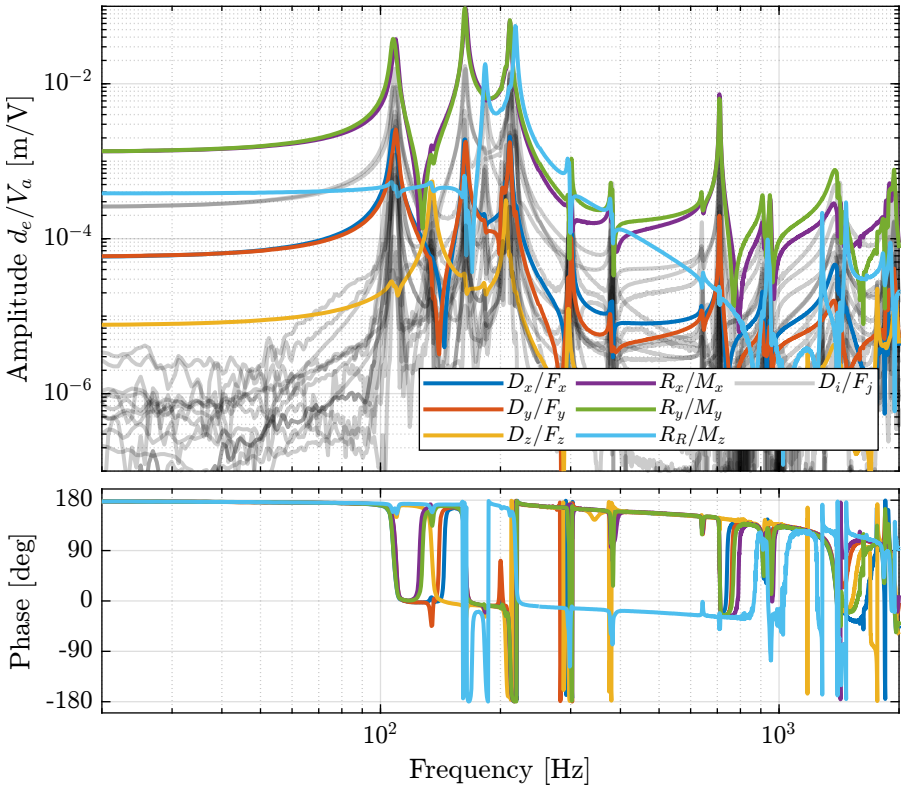

Figure 1.6: Measured FRF for the DVF plant in the cartesian frame

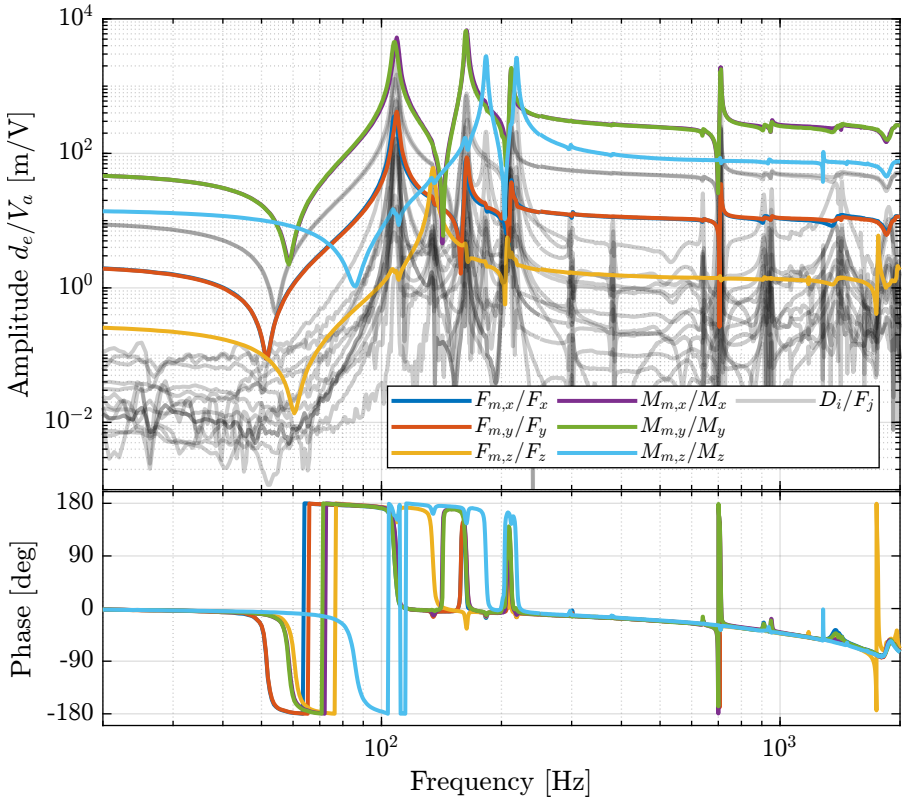

Figure 1.7: Measured FRF for the IFF plant in the cartesian frame