Diagonal control using the SVD and the Jacobian Matrix

Dehaeze Thomas

February 5, 2021

Contents

In this document, the use of the Jacobian matrix and the Singular Value Decomposition to render a physical plant diagonal dominant is studied. Then, a diagonal controller is used.

These two methods are tested on two plants:

- In Section 1 on a 3-DoF gravimeter
- In Section 5 on a 6-DoF Stewart platform

1 Gravimeter - Simscape Model

1.1 Introduction

In this part, diagonal control using both the SVD and the Jacobian matrices are applied on a gravimeter model:

- Section 1.2: the model is described and its parameters are defined.
- Section 1.3: the plant dynamics from the actuators to the sensors is computed from a Simscape model.
- Section 1.4: the plant is decoupled using the Jacobian matrices.
- Section 1.5: the Singular Value Decomposition is performed on a real approximation of the plant transfer matrix and further use to decouple the system.
- Section 1.6: the effectiveness of the decoupling is computed using the Gershorin radii
- Section 1.7: the effectiveness of the decoupling is computed using the Relative Gain Array
- Section 1.8: the obtained decoupled plants are compared
- Section 1.9: the diagonal controller is developed
- Section 1.10: the obtained closed-loop performances for the two methods are compared
- Section 1.11: the robustness to a change of actuator position is evaluated
- Section 1.12: the choice of the reference frame for the evaluation of the Jacobian is discussed
- Section 1.13: the decoupling performances of SVD is evaluated for a low damped and an highly damped system

1.2 Gravimeter Model - Parameters

The model of the gravimeter is schematically shown in Figure 1.1.

The parameters used for the simulation are the following:

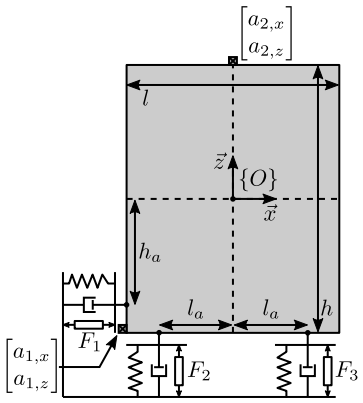

Figure 1.1: Model of the gravimeter

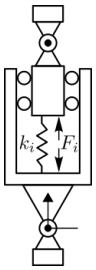

Figure 1.2: Model of the struts

Matlab
```
l  = 1.0; % Length of the mass [m]
h  = 1.7; % Height of the mass [m]

la = l/2; % Position of Act. [m]
ha = h/2; % Position of Act. [m]

m = 400; % Mass [kg]
I = 115; % Inertia [kg m^2]

k = 15e3; % Actuator Stiffness [N/m]
c = 2e1; % Actuator Damping [N/(m/s)]

deq = 0.2; % Length of the actuators [m]

g = 0; % Gravity [m/s2]
```

1.3 System Identification

Matlab
```
%% Name of the Simulink File
mdl = 'gravimeter';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/F1'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F2'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F3'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 1, 'openoutput'); io_i = io_i + 1;
```

```
io(io_i) = linio([mdl, '/Acc_top'], 2, 'openoutput'); io_i = io_i + 1;

G = linearize(mdl, io);
G.InputName  = {'F1', 'F2', 'F3'};
G.OutputName = {'Ax1', 'Ay1', 'Ax2', 'Ay2'};
```

The inputs and outputs of the plant are shown in Figure 1.3.

More precisely there are three inputs (the three actuator forces):

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_2 \end{bmatrix} \tag{1.1}$$

And 4 outputs (the two 2-DoF accelerometers):

$$\boldsymbol{a} = \begin{bmatrix} a_{1x} \\ a_{1y} \\ a_{2x} \\ a_{2y} \end{bmatrix} \tag{1.2}$$

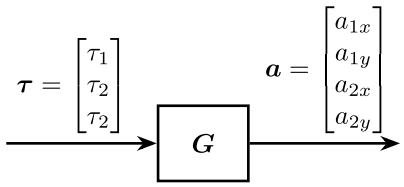

Figure 1.3: Schematic of the gravimeter plant

We can check the poles of the plant:

-0.12243+13.551i
-0.12243-13.551i
-0.05+8.6601i
-0.05-8.6601i
-0.0088785+3.6493i
-0.0088785-3.6493i

As expected, the plant as 6 states (2 translations + 1 rotation)

Matlab
```
size(G)
```

Results
```
State-space model with 4 outputs, 3 inputs, and 6 states.
```

The bode plot of all elements of the plant are shown in Figure 1.4.

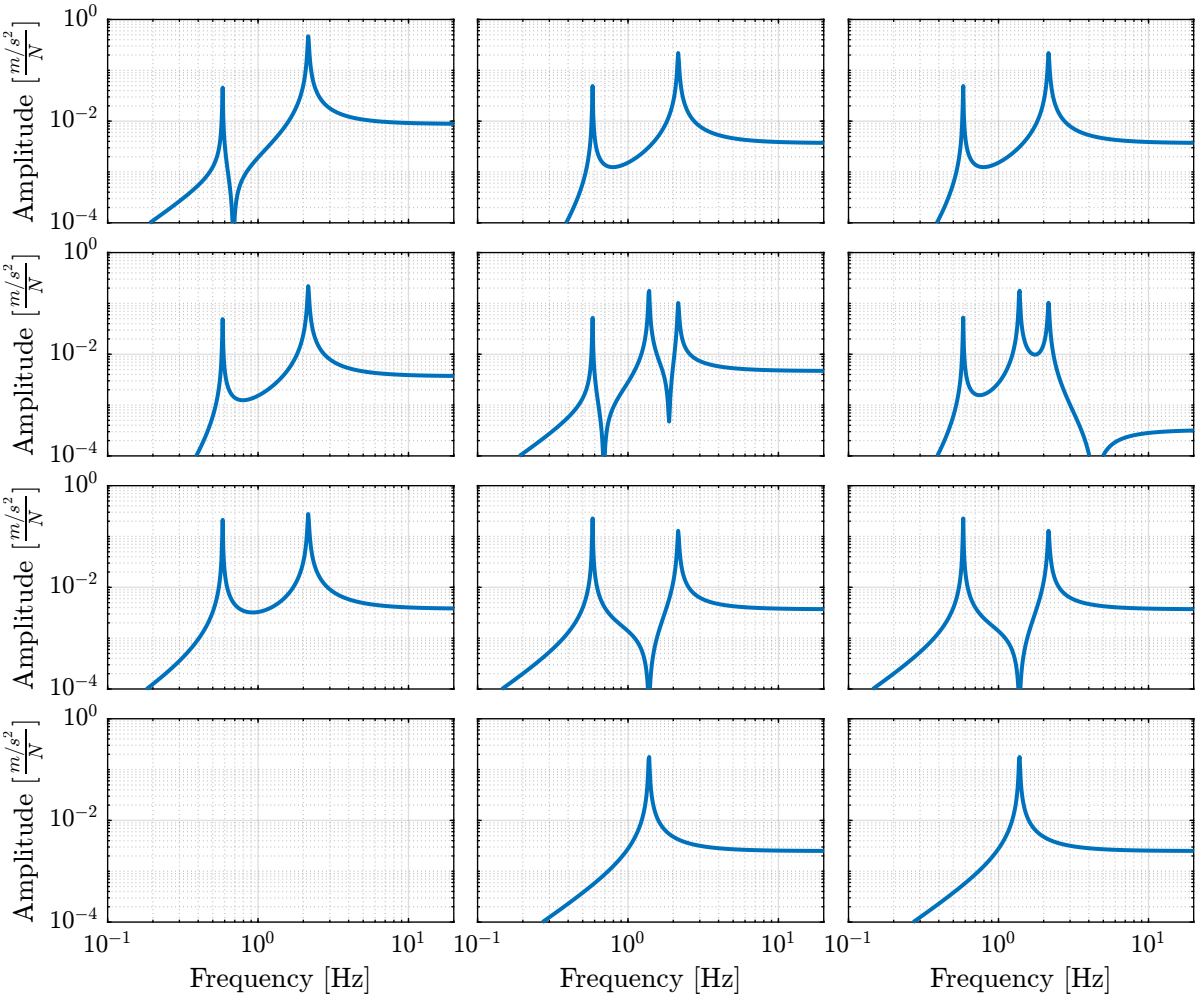

Figure 1.4: Open Loop Transfer Function from 3 Actuators to 4 Accelerometers

1.4 Decoupling using the Jacobian

Consider the control architecture shown in Figure 1.5.

The Jacobian matrix J_τ is used to transform forces applied by the three actuators into forces/torques applied on the gravimeter at its center of mass:

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = J_\tau^{-T} \begin{bmatrix} F_x \\ F_y \\ M_z \end{bmatrix} \tag{1.3}$$

The Jacobian matrix J_a is used to compute the vertical acceleration, horizontal acceleration and rotational acceleration of the mass with respect to its center of mass:

$$\begin{bmatrix} a_x \\ a_y \\ a_{R_z} \end{bmatrix} = J_a^{-1} \begin{bmatrix} a_{x1} \\ a_{y1} \\ a_{x2} \\ a_{y2} \end{bmatrix} \tag{1.4}$$

We thus define a new plant as defined in Figure 1.5.

$$\boldsymbol{G}_x(s) = J_a^{-1} \boldsymbol{G}(s) J_\tau^{-T}$$

$\boldsymbol{G}_x(s)$ correspond to the 3×3 transfer function matrix from forces and torques applied to the gravimeter at its center of mass to the absolute acceleration of the gravimeter's center of mass (Figure 1.5).

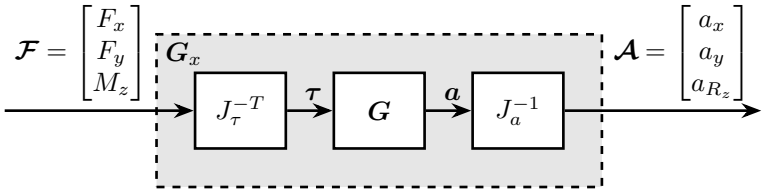

Figure 1.5: Decoupled plant $\boldsymbol{G}_x$ using the Jacobian matrix J

The Jacobian corresponding to the sensors and actuators are defined below:

Matlab
```
Ja = [1 0 -h/2
      0 1  l/2
      1 0  h/2
      0 1  0];

Jt = [1 0 -ha
      0 1  la
      0 1 -la];
```

And the plant $\boldsymbol{G}_x$ is computed:

Matlab
```
Gx = pinv(Ja)*G*pinv(Jt');
Gx.InputName  = {'Fx', 'Fy', 'Mz'};
Gx.OutputName  = {'Dx', 'Dy', 'Rz'};
```

Results

```
size(Gx)
State-space model with 3 outputs, 3 inputs, and 6 states.
```

The diagonal and off-diagonal elements of G_x are shown in Figure 1.6.

It is shown at the system is:

- decoupled at high frequency thanks to a diagonal mass matrix (the Jacobian being evaluated at the center of mass of the payload)
- coupled at low frequency due to the non-diagonal terms in the stiffness matrix, especially the term corresponding to a coupling between a force in the x direction to a rotation around z (due to the torque applied by the stiffness 1).

The choice of the frame in this the Jacobian is evaluated is discussed in Section 1.12.

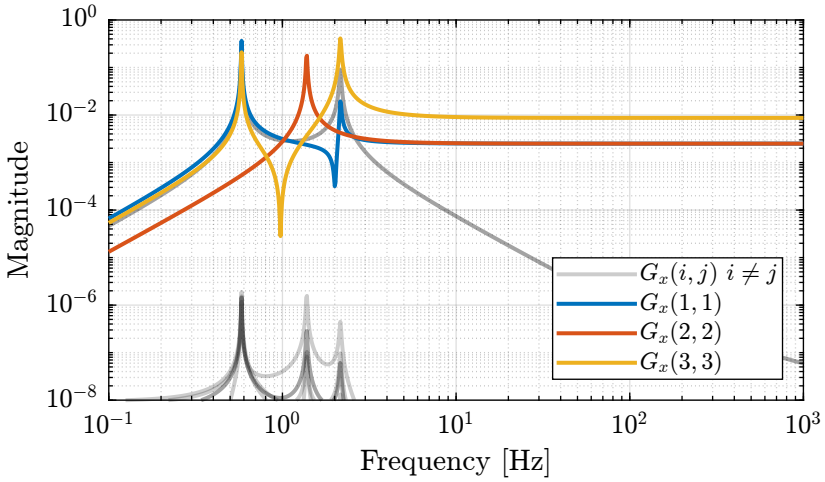

Figure 1.6: Diagonal and off-diagonal elements of G_x

1.5 Decoupling using the SVD

In order to decouple the plant using the SVD, first a real approximation of the plant transfer function matrix as the crossover frequency is required.

Let's compute a real approximation of the complex matrix H_1 which corresponds to the the transfer function $G(j\omega_c)$ from forces applied by the actuators to the measured acceleration of the top platform evaluated at the frequency ω_c.

Matlab

```
wc = 2*pi*10; % Decoupling frequency [rad/s]

H1 = evalfr(G, j*wc);
```

The real approximation is computed as follows:

```
Matlab
D = pinv(real(H1'*H1));
H1 = pinv(D*real(H1'*diag(exp(j*angle(diag(H1*D*H1.'))/2))));
```

Table 1.1: Real approximate of G at the decoupling frequency ω_c

0.0092	-0.0039	0.0039
-0.0039	0.0048	0.00028
-0.004	0.0038	-0.0038
8.4e-09	0.0025	0.0025

Now, the Singular Value Decomposition of H_1 is performed:

$$H_1 = U\Sigma V^H$$

```
Matlab
[U,S,V] = svd(H1);
```

Table 1.2: U matrix

-0.78	0.26	-0.53	-0.2
0.4	0.61	-0.04	-0.68
0.48	-0.14	-0.85	0.2
0.03	0.73	0.06	0.68

Table 1.3: V matrix

-0.79	0.11	-0.6
0.51	0.67	-0.54
-0.35	0.73	0.59

The obtained matrices U and V are used to decouple the system as shown in Figure 1.7.

The decoupled plant is then:

$$\boldsymbol{G}_{SVD}(s) = U^{-1}\boldsymbol{G}(s)V^{-H}$$

```
Matlab
Gsvd = inv(U)*G*inv(V');
```

```
Results
size(Gsvd)
State-space model with 4 outputs, 3 inputs, and 6 states.
```

The 4th output (corresponding to the null singular value) is discarded, and we only keep the 3×3 plant:

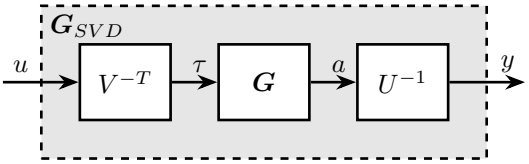

Figure 1.7: Decoupled plant G_{SVD} using the Singular Value Decomposition

```Matlab
Gsvd = Gsvd(1:3, 1:3);
```

The diagonal and off-diagonal elements of the "SVD" plant are shown in Figure 1.8.

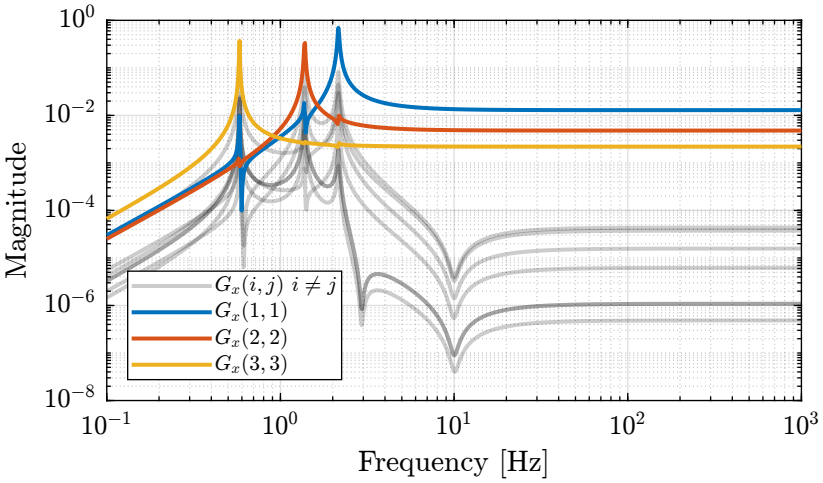

Figure 1.8: Diagonal and off-diagonal elements of G_{svd}

1.6 Verification of the decoupling using the "Gershgorin Radii"

The "Gershgorin Radii" is computed for the coupled plant $G(s)$, for the "Jacobian plant" $G_x(s)$ and the "SVD Decoupled Plant" $G_{SVD}(s)$:

The "Gershgorin Radii" of a matrix S is defined by:

$$\zeta_i(j\omega) = \frac{\sum\limits_{j \neq i} |S_{ij}(j\omega)|}{|S_{ii}(j\omega)|}$$

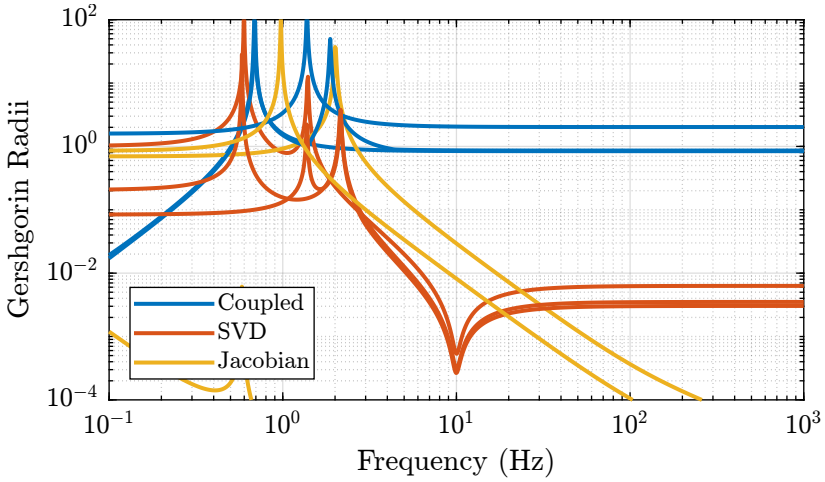

Figure 1.9: Gershgorin Radii of the Coupled and Decoupled plants

1.7 Verification of the decoupling using the "Relative Gain Array"

The relative gain array (RGA) is defined as:

$$\Lambda\big(G(s)\big) = G(s) \times \big(G(s)^{-1}\big)^T \tag{1.5}$$

where $\times$ denotes an element by element multiplication and $G(s)$ is an $n \times n$ square transfer matrix.

The obtained RGA elements are shown in Figure 1.10.

The RGA-number is also a measure of diagonal dominance:

$$\text{RGA-number} = \|\Lambda(G) - I\|_{\text{sum}} \tag{1.6}$$

1.8 Obtained Decoupled Plants

The bode plot of the diagonal and off-diagonal elements of G_{SVD} are shown in Figure 1.12.

Similarly, the bode plots of the diagonal elements and off-diagonal elements of the decoupled plant $G_x(s)$ using the Jacobian are shown in Figure 1.13.

1.9 Diagonal Controller

The control diagram for the centralized control is shown in Figure 1.14.

The controller K_c is "working" in an cartesian frame. The Jacobian is used to convert forces in the cartesian frame to forces applied by the actuators.

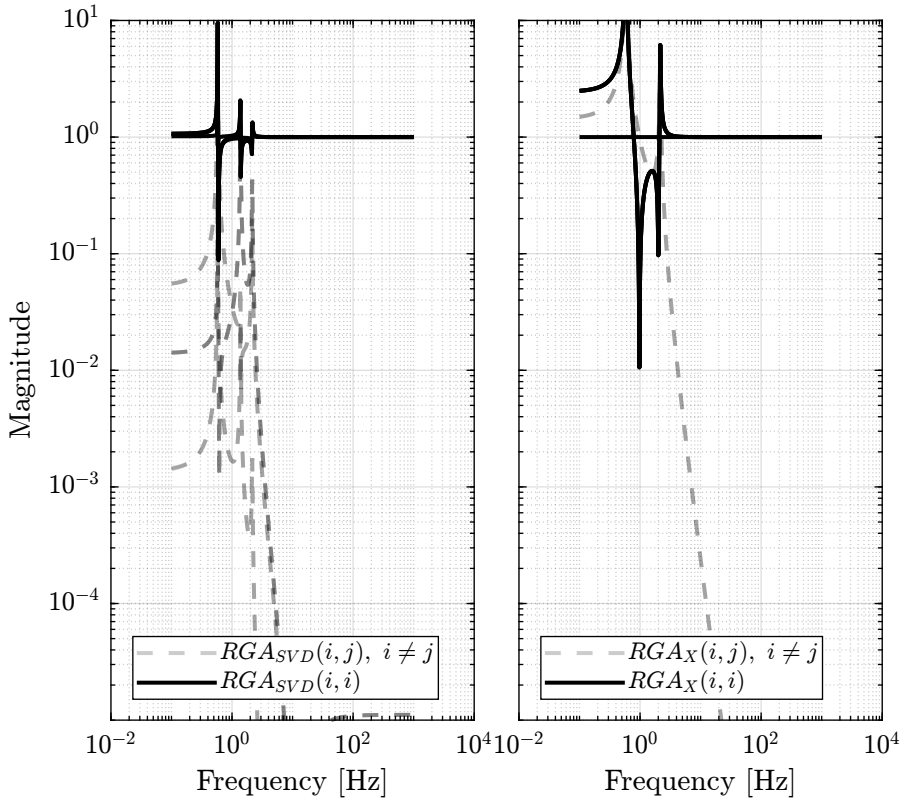

Figure 1.10: Obtained norm of RGA elements for the SVD decoupled plant and the Jacobian decoupled plant

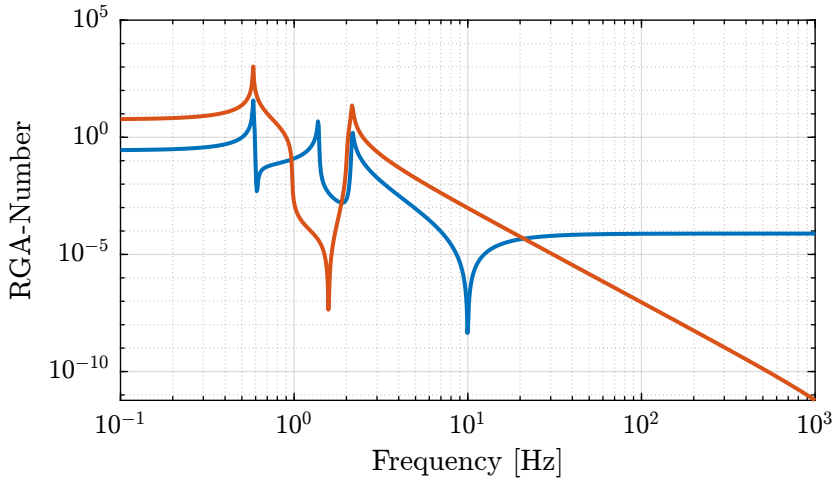

Figure 1.11: RGA-Number for the Gravimeter

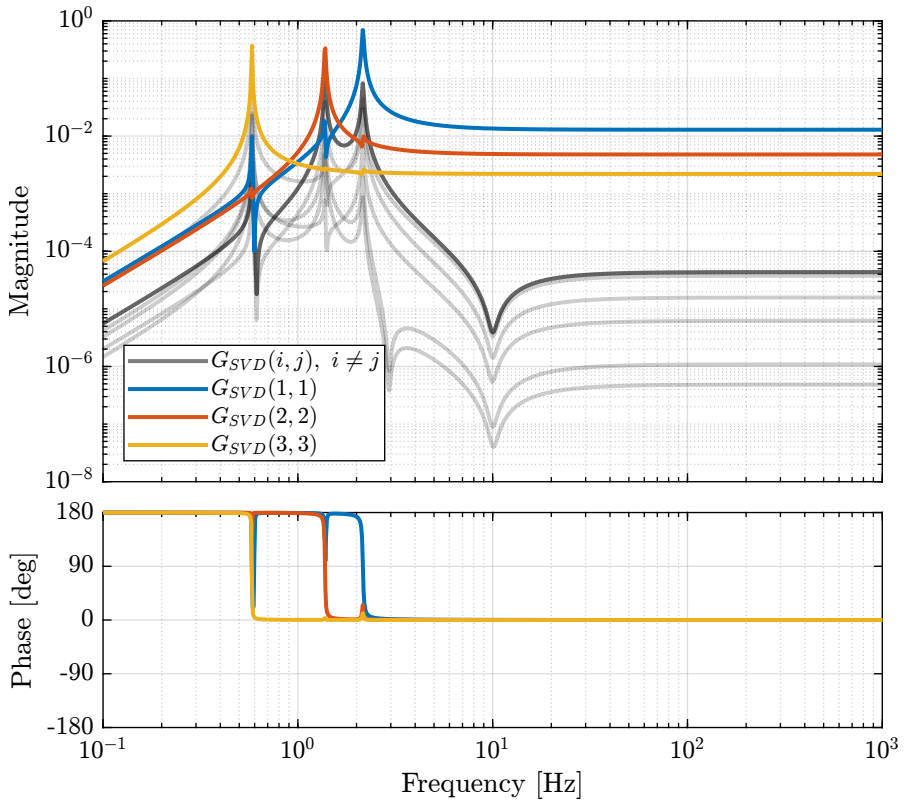

Figure 1.12: Decoupled Plant using SVD

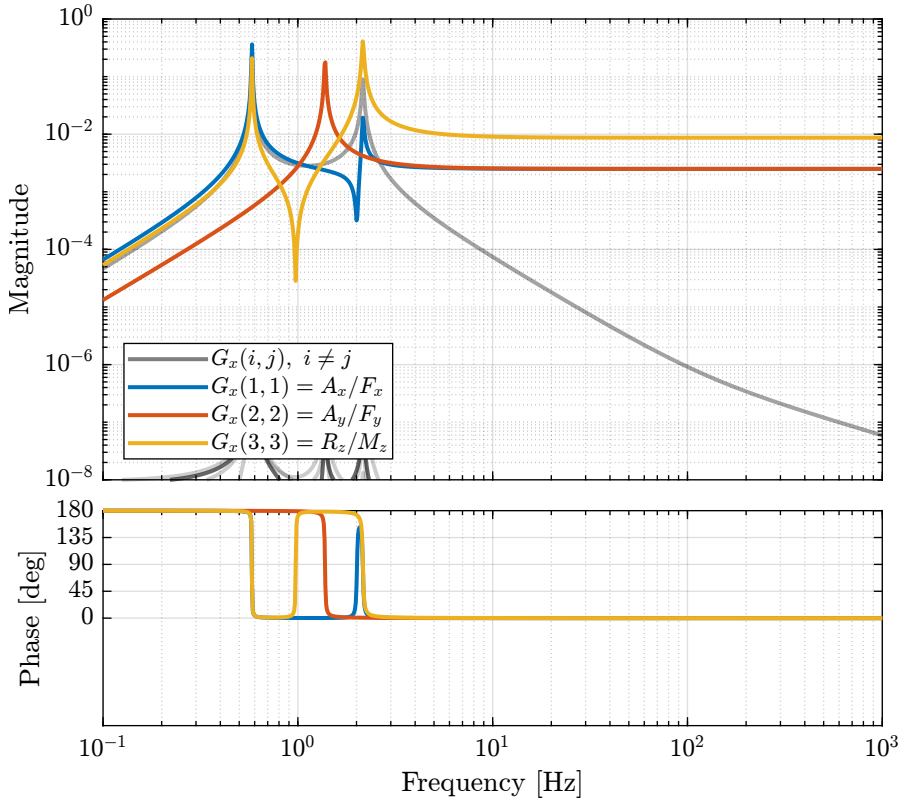

Figure 1.13: Gravimeter Platform Plant from forces (resp. torques) applied by the legs to the acceleration (resp. angular acceleration) of the platform as well as all the coupling terms between the two (non-diagonal terms of the transfer function matrix)

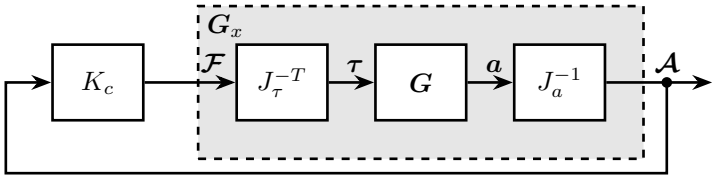

Figure 1.14: Control Diagram for the Centralized control

The SVD control architecture is shown in Figure 1.15. The matrices U and V are used to decoupled the plant G.

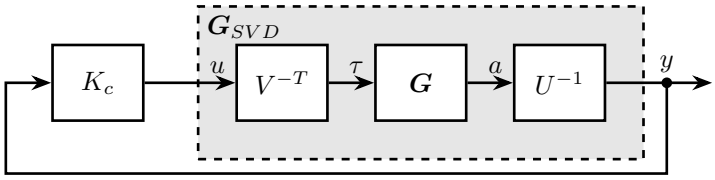

Figure 1.15: Control Diagram for the SVD control

We choose the controller to be a low pass filter:

$$K_c(s) = \frac{G_0}{1 + \frac{s}{\omega_0}}$$

G_0 is tuned such that the crossover frequency corresponding to the diagonal terms of the loop gain is equal to ω_c

Matlab
```
wc = 2*pi*10;  % Crossover Frequency [rad/s]
w0 = 2*pi*0.1; % Controller Pole [rad/s]
```

Matlab
```
K_cen = diag(1./diag(abs(evalfr(Gx, j*wc))))*(1/abs(evalfr(1/(1 + s/w0), j*wc)))/(1 + s/w0);
L_cen = K_cen*Gx;
```

Matlab
```
K_svd = diag(1./diag(abs(evalfr(Gsvd, j*wc))))*(1/abs(evalfr(1/(1 + s/w0), j*wc)))/(1 + s/w0);
L_svd = K_svd*Gsvd;
U_inv = inv(U);
```

The obtained diagonal elements of the loop gains are shown in Figure 1.16.

1.10 Closed-Loop system Performances

Now the system is identified again with additional inputs and outputs:

- x, y and R_z ground motion
- x, y and R_z acceleration of the payload.

Matlab
```
%% Name of the Simulink File
mdl = 'gravimeter';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/Dx'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Dy'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Rz'], 1, 'openinput');  io_i = io_i + 1;
```

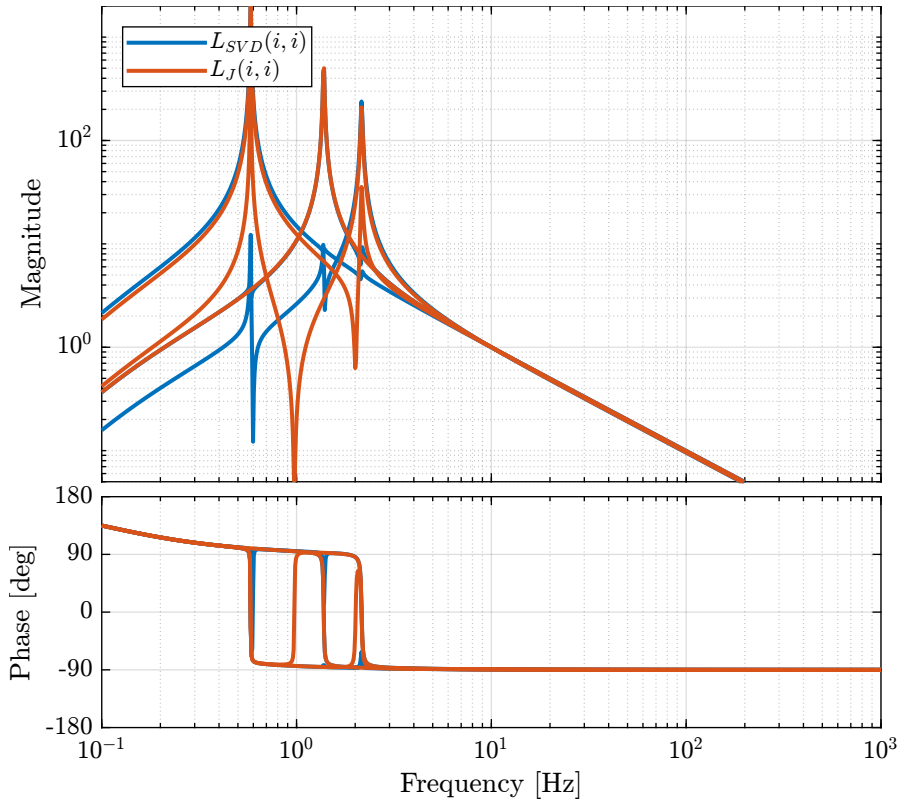

Figure 1.16: Comparison of the diagonal elements of the loop gains for the SVD control architecture and the Jacobian one

```
io(io_i) = linio([mdl, '/F1'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F2'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F3'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 3, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 2, 'openoutput'); io_i = io_i + 1;

G = linearize(mdl, io);
G.InputName  = {'Dx', 'Dy', 'Rz', 'F1', 'F2', 'F3'};
G.OutputName = {'Ax', 'Ay', 'Arz', 'Ax1', 'Ay1', 'Ax2', 'Ay2'};
```

The loop is closed using the developed controllers.

Matlab
```
G_cen = lft(G, -pinv(Jt')*K_cen*pinv(Ja));
G_svd = lft(G, -inv(V')*K_svd*U_inv(1:3, :));
```

Let's first verify the stability of the closed-loop systems:

Matlab
```
isstable(G_cen)
```

Results
```
ans =
  logical
   1
```

Matlab
```
isstable(G_svd)
```

Results
```
ans =
  logical
   1
```

The obtained transmissibility in Open-loop, for the centralized control as well as for the SVD control are shown in Figure 1.17.

1.11 Robustness to a change of actuator position

Let say we change the position of the actuators:

Matlab
```
la = l/2*0.7; % Position of Act. [m]
ha = h/2*0.7; % Position of Act. [m]
```

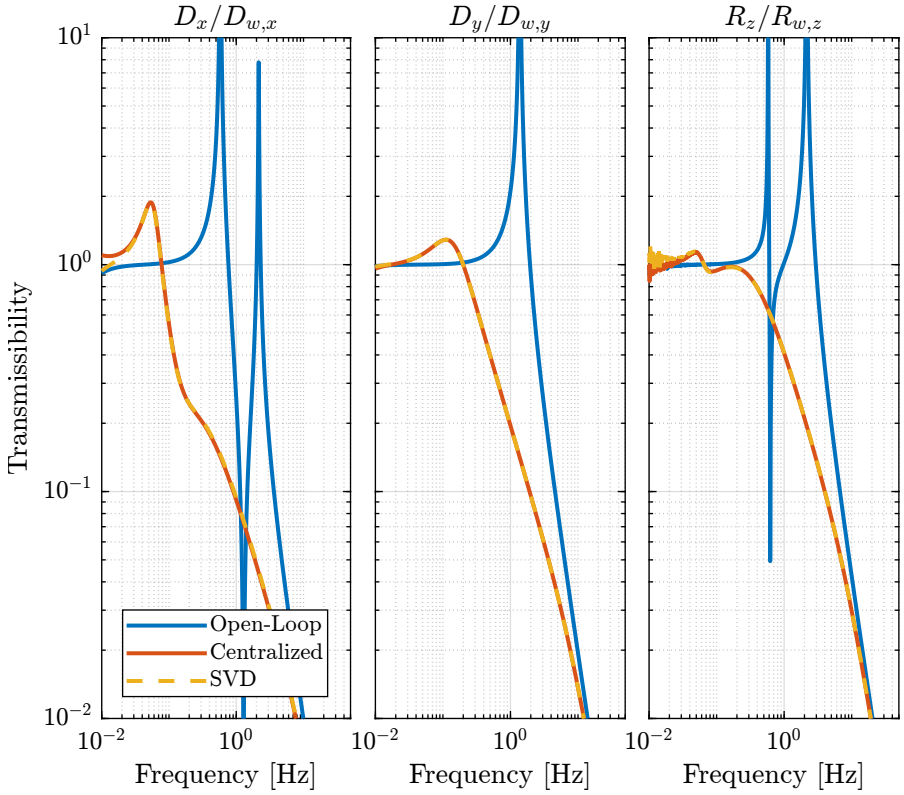

Figure 1.17: Obtained Transmissibility

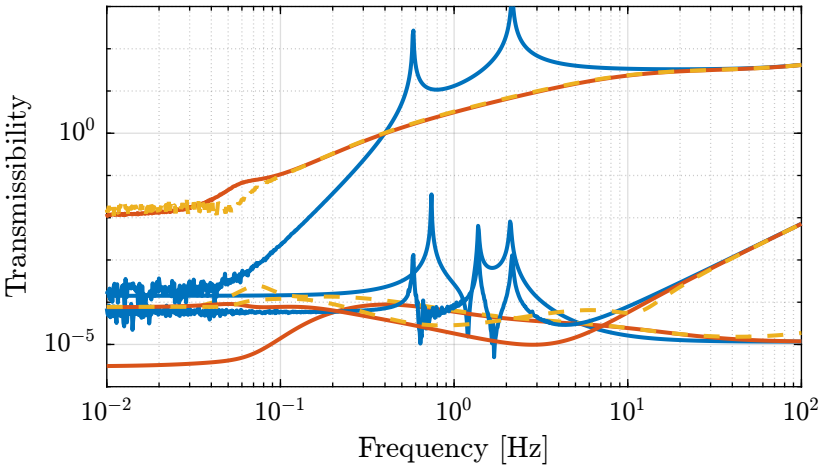

Figure 1.18: Obtain coupling terms of the transmissibility matrix

```Matlab
%% Name of the Simulink File
mdl = 'gravimeter';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/Dx'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Dy'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Rz'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F1'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F2'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F3'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Abs_Motion'], 3, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 2, 'openoutput'); io_i = io_i + 1;

G = linearize(mdl, io);
G.InputName  = {'Dx', 'Dy', 'Rz', 'F1', 'F2', 'F3'};
G.OutputName = {'Ax', 'Ay', 'Arz', 'Ax1', 'Ay1', 'Ax2', 'Ay2'};
```

The loop is closed using the developed controllers.

```Matlab
G_cen_b = lft(G, -pinv(Jt')*K_cen*pinv(Ja));
G_svd_b = lft(G, -inv(V')*K_svd*U_inv(1:3, :));
```

The new plant is computed, and the centralized and SVD control architectures are applied using the previously computed Jacobian matrices and U and V matrices.

The closed-loop system are still stable in both cases, and the obtained transmissibility are equivalent as shown in Figure 1.19.

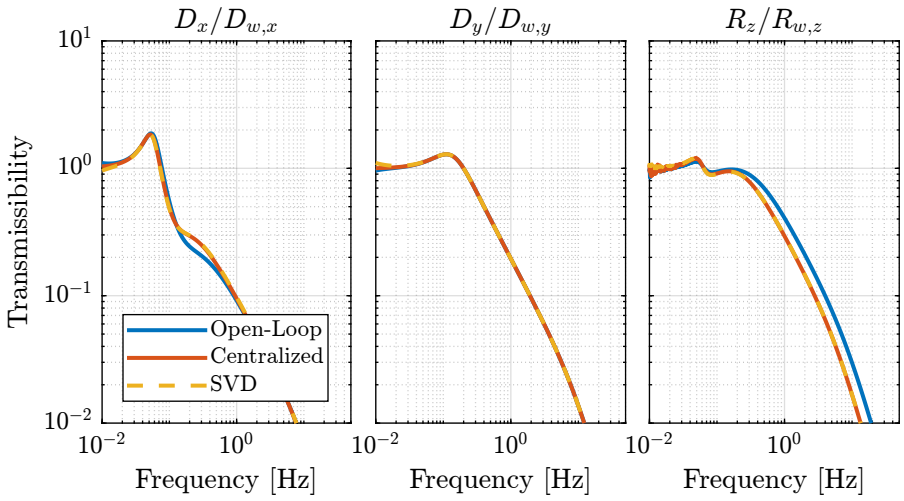

Figure 1.19: Transmissibility for the initial CL system and when the position of actuators are changed

1.12 Choice of the reference frame for Jacobian decoupling

If we want to decouple the system at low frequency (determined by the stiffness matrix), we have to compute the Jacobian at a point where the stiffness matrix is diagonal. A displacement (resp. rotation) of the mass at this particular point should induce a **pure** force (resp. torque) on the same point due to stiffnesses in the system. This can be verified by geometrical computations.

If we want to decouple the system at high frequency (determined by the mass matrix), we have tot compute the Jacobians at the Center of Mass of the suspended solid. Similarly to the stiffness analysis, when considering only the inertia effects (neglecting the stiffnesses), a force (resp. torque) applied at this point (the center of mass) should induce a **pure** acceleration (resp. angular acceleration).

Ideally, we would like to have a decoupled mass matrix and stiffness matrix at the same time. To do so, the actuators (springs) should be positioned such that the stiffness matrix is diagonal when evaluated at the CoM of the solid.

1.12.1 Decoupling of the mass matrix

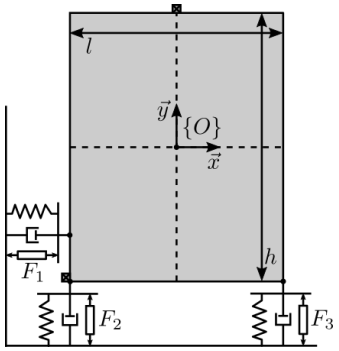

Figure 1.20: Choice of {O} such that the Mass Matrix is Diagonal

Matlab
```
la = l/2; % Position of Act. [m]
ha = h/2; % Position of Act. [m]
```

Matlab
```
%% Name of the Simulink File
mdl = 'gravimeter';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/F1'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F2'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F3'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 2, 'openoutput'); io_i = io_i + 1;

G = linearize(mdl, io);
G.InputName  = {'F1', 'F2', 'F3'};
G.OutputName = {'Ax1', 'Ay1', 'Ax2', 'Ay2'};
```

Decoupling at the CoM (Mass decoupled)

```Matlab
JMa = [1 0 -h/2
       0 1  l/2
       1 0  h/2
       0 1  0];

JMt = [1 0 -ha
       0 1  la
       0 1 -la];
```

```Matlab
GM = pinv(JMa)*G*pinv(JMt');
GM.InputName  = {'Fx', 'Fy', 'Mz'};
GM.OutputName  = {'Dx', 'Dy', 'Rz'};
```

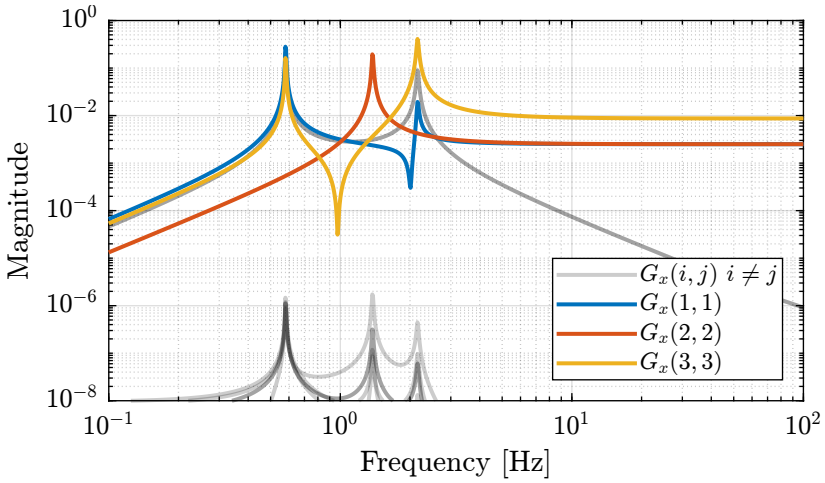

Figure 1.21: Diagonal and off-diagonal elements of the decoupled plant

1.12.2 Decoupling of the stiffness matrix

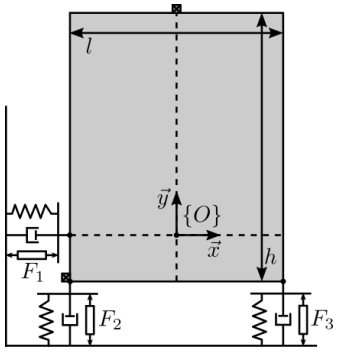

Figure 1.22: Choice of {O} such that the Stiffness Matrix is Diagonal

Decoupling at the point where K is diagonal (x = 0, y = -h/2 from the schematic {O} frame):

```matlab
JKa = [1 0  0
       0 1 -l/2
       1 0 -h
       0 1  0];

JKt = [1 0  0
       0 1 -la
       0 1  la];
```

And the plant $\boldsymbol{G}_x$ is computed:

```matlab
GK = pinv(JKa)*G*pinv(JKt');
GK.InputName  = {'Fx', 'Fy', 'Mz'};
GK.OutputName  = {'Dx', 'Dy', 'Rz'};
```

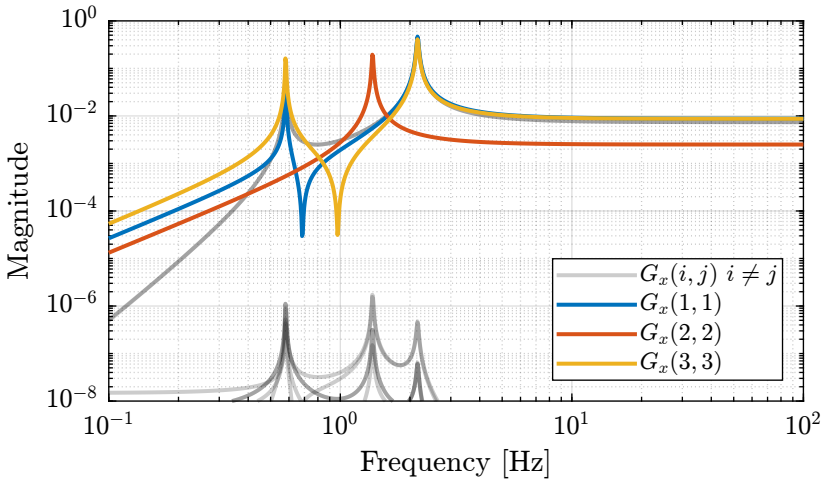

Figure 1.23: Diagonal and off-diagonal elements of the decoupled plant

1.12.3 Combined decoupling of the mass and stiffness matrices

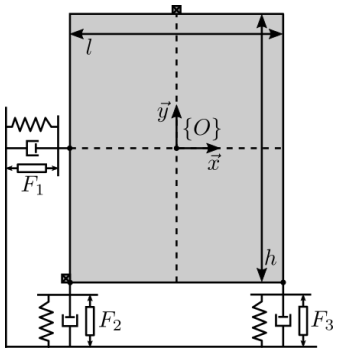

Figure 1.24: Ideal location of the actuators such that both the mass and stiffness matrices are diagonal

To do so, the actuator position should be modified

```Matlab
la = l/2; % Position of Act. [m]
ha = 0; % Position of Act. [m]
```

```Matlab
%% Name of the Simulink File
mdl = 'gravimeter';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/F1'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F2'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/F3'], 1, 'openinput');  io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_side'], 2, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 1, 'openoutput'); io_i = io_i + 1;
io(io_i) = linio([mdl, '/Acc_top'], 2, 'openoutput'); io_i = io_i + 1;

G = linearize(mdl, io);
G.InputName  = {'F1', 'F2', 'F3'};
G.OutputName = {'Ax1', 'Ay1', 'Ax2', 'Ay2'};
```

```Matlab
JMa = [1 0 -h/2
       0 1  l/2
       1 0  h/2
       0 1  0];

JMt = [1 0 -ha
       0 1  la
       0 1 -la];
```

```Matlab
GKM = pinv(JMa)*G*pinv(JMt');
GKM.InputName  = {'Fx', 'Fy', 'Mz'};
GKM.OutputName  = {'Dx', 'Dy', 'Rz'};
```

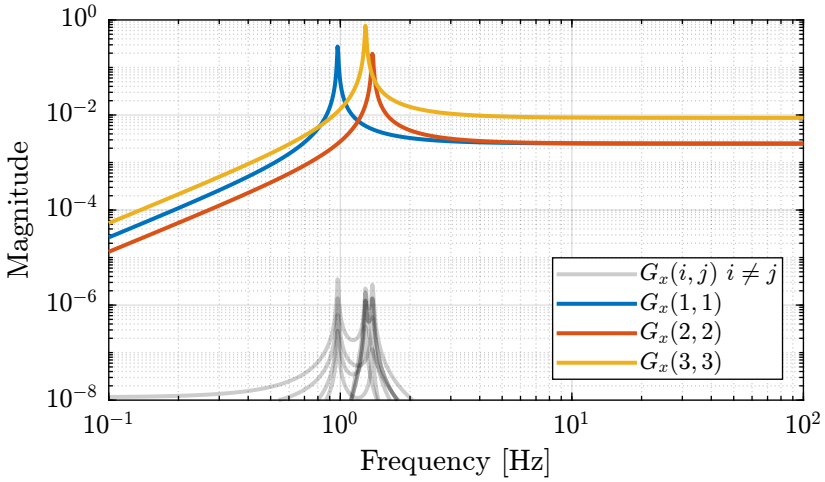

Figure 1.25: Diagonal and off-diagonal elements of the decoupled plant

1.12.4 Conclusion

Ideally, the mechanical system should be designed in order to have a decoupled stiffness matrix at the CoM of the solid.

If not the case, the system can either be decoupled as low frequency if the Jacobian are evaluated at a point where the stiffness matrix is decoupled. Or it can be decoupled at high frequency if the Jacobians are evaluated at the CoM.

1.13 SVD decoupling performances

As the SVD is applied on a **real approximation** of the plant dynamics at a frequency ω_0, it is foreseen that the effectiveness of the decoupling depends on the validity of the real approximation.

Let's do the SVD decoupling on a plant that is mostly real (low damping) and one with a large imaginary part (larger damping).

Start with small damping, the obtained diagonal and off-diagonal terms are shown in Figure 1.26.

```Matlab
c = 2e1; % Actuator Damping [N/(m/s)]
```

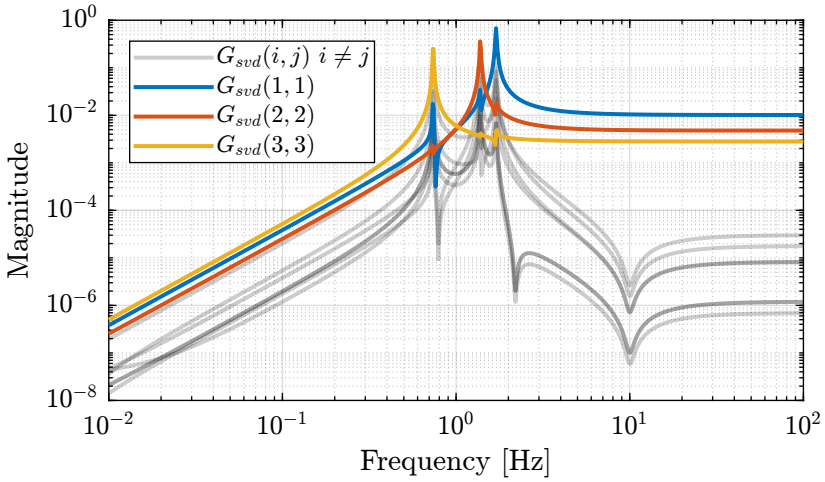

Figure 1.26: Diagonal and off-diagonal term when decoupling with SVD on the gravimeter with small damping

Now take a larger damping, the obtained diagonal and off-diagonal terms are shown in Figure 1.27.

```Matlab
c = 5e2; % Actuator Damping [N/(m/s)]
```

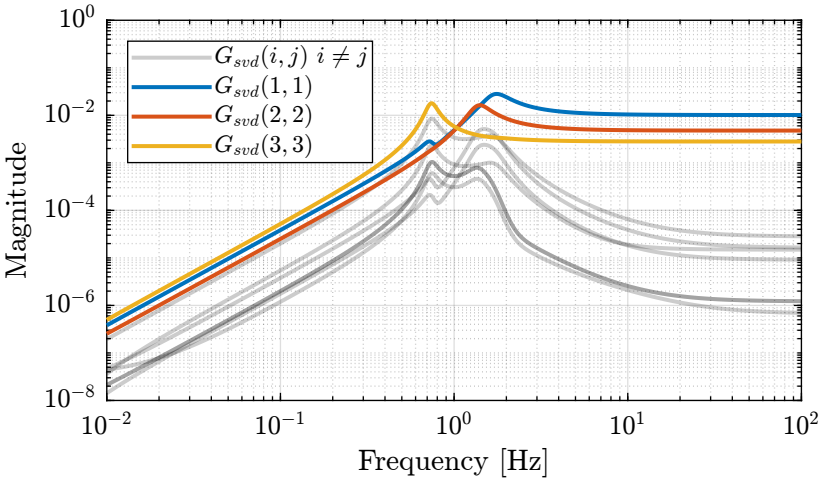

Figure 1.27: Diagonal and off-diagonal term when decoupling with SVD on the gravimeter with high damping

2 Analytical Model

2.1 Model

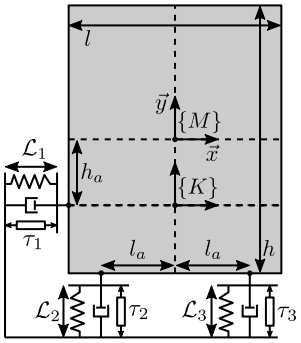

Figure 2.1: Model of the gravimeter

- collocated actuators and sensors

2.2 Stiffness and Mass matrices

Stiffness matrix:

$$\mathcal{F}_{\{O\}} = -K_{\{O\}}\mathcal{X}_{\{O\}} \tag{2.1}$$

with:

- $\mathcal{X}_{\{O\}}$ are displacements/rotations of the mass x, y, R_z expressed in the frame $\{O\}$
- $\mathcal{F}_{\{O\}}$ are forces/torques $\mathcal{F}_x$, $\mathcal{F}_y$, $\mathcal{M}_z$ applied at the origin of $\{O\}$

Mass matrix:

$$\mathcal{F}_{\{O\}} = M_{\{O\}}\ddot{\mathcal{X}}_{\{O\}} \tag{2.2}$$

Consider the two following frames:

- $\{M\}$: Center of mass => diagonal mass matrix

$$M_{\{M\}} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}$$

$$K_{\{M\}} = \begin{bmatrix} k_1 & 0 & k_1 h_a \\ 0 & k_2 + k_3 & 0 \\ k_1 h_a & 0 & k_1 h_a + (k_2 + k_3) l_a \end{bmatrix}$$

- $\{K\}$: Diagonal stiffness matrix

$$K_{\{K\}} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 + k_3 & 0 \\ 0 & 0 & (k_2 + k_3) l_a \end{bmatrix}$$

  ☐ Compute the mass matrix $M_{\{K\}}$ Needs two Jacobians => complicated matrix

2.3 Equations

☐ Ideally write the equation from τ to $\mathcal{L}$

$$\mathcal{L} = \begin{bmatrix} \mathcal{L}_1 \\ \mathcal{L}_2 \\ \mathcal{L}_3 \end{bmatrix} \tag{2.3}$$

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{2.4}$$

2.4 Jacobians

Usefulness of Jacobians:

- $J_{\{M\}}$ converts $\dot{\mathcal{L}}$ to $\dot{\mathcal{X}}_{\{M\}}$:

$$\dot{\mathcal{X}}_{\{M\}} = J_{\{M\}} \dot{\mathcal{L}}$$

- $J^T_{\{M\}}$ converts τ to $\mathcal{F}_{\{M\}}$:

$$\mathcal{F}_{\{M\}} = J^T_{\{M\}} \tau$$

- $J_{\{K\}}$ converts \$K\}\$to $\dot{\mathcal{X}}_{\{K\}}$:

$$\dot{\mathcal{X}}_{\{K\}} = J_{\{K\}} \dot{\mathcal{K}}$$

- $J^T_{\{K\}}$ converts τ to $\mathcal{F}_{\{K\}}$:

$$\mathcal{F}_{\{K\}} = J^T_{\{K\}} \tau$$

Let's compute the Jacobians:

$$J_{\{M\}} = \begin{bmatrix} 1 & 0 & h_a \\ 0 & 1 & -l_a \\ 0 & 1 & l_a \end{bmatrix} \tag{2.5}$$

$$J_{\{K\}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_a \\ 0 & 1 & l_a \end{bmatrix} \tag{2.6}$$

2.5 Parameters

Matlab
```
l  = 1.0; % Length of the mass [m]
h  = 2*1.7; % Height of the mass [m]

la = l/2; % Position of Act. [m]
ha = h/2; % Position of Act. [m]

m = 400; % Mass [kg]
I = 115; % Inertia [kg m^2]

c1 = 2e1; % Actuator Damping [N/(m/s)]
c2 = 2e1; % Actuator Damping [N/(m/s)]
c3 = 2e1; % Actuator Damping [N/(m/s)]

k1 = 15e3; % Actuator Stiffness [N/m]
k2 = 15e3; % Actuator Stiffness [N/m]
k3 = 15e3; % Actuator Stiffness [N/m]
```

2.6 Transfer function from τ to $\delta\mathcal{L}$

Mass, Damping and Stiffness matrices expressed in $\{M\}$:

Matlab
```
Mm = [m 0 0 ;
      0 m 0 ;
      0 0 I];

Cm = [c1    0     c1*ha ;
      0     c2+c3 0 ;
      c1*ha 0     c1*ha + (c2+c3)*la];

Km = [k1    0     k1*ha ;
      0     k2+k3 0 ;
      k1*ha 0     k1*ha + (k2+k3)*la];
```

Jacobian $J_{\{M\}}$:

Matlab
```
Jm = [1 0  ha ;
      0 1 -la ;
      0 1  la];
```

Matlab
```
Mt = inv(Jm')*Mm*inv(Jm);
Ct = inv(Jm')*Cm*inv(Jm);
Kt = inv(Jm')*Km*inv(Jm);
```

Table 2.1: M_t

400.0	340.0	-340.0
340.0	504.0	-304.0
-340.0	-304.0	504.0

Table 2.2: K_t

15000.0	0.0	0.0
0.0	24412.5	-9412.5
0.0	-9412.5	24412.5

Matlab
```
Gt = s^2*inv(Mt*s^2 + Ct*s + Kt);
% Gt = JM*s^2*inv(MM*s^2 + CM*s + KM)*JM';
```

2.7 Transfer function from $\mathcal{F}_{\{M\}}$ to $\mathcal{X}_{\{M\}}$

Matlab
```
Gm = inv(Jm)*Gt*inv(Jm');
```

Table 2.3: $M_{\{M\}}$

400.0	0.0	0.0
0.0	400.0	0.0
0.0	0.0	115.0

2.8 Transfer function from $\mathcal{F}_{\{K\}}$ to $\mathcal{X}_{\{K\}}$

Jacobian:

Matlab
```
Jk = [1 0  0
      0 1 -la
      0 1  la];
```

Mass, Damping and Stiffness matrices expressed in $\{K\}$:

Matlab
```
Mk = Jk'*Mt*Jk;
Ck = Jk'*Ct*Jk;
Kk = Jk'*Kt*Jk;
```

Table 2.4: $K_{\{M\}}$

15000.0	0.0	12750.0
0.0	30000.0	0.0
12750.0	0.0	27750.0

Table 2.5: $M_{\{K\}}$

400.0	0.0	-340.0
0.0	400.0	0.0
-340.0	0.0	404.0

Matlab
```
% Gk = s^2*inv(Mk*s^2 + Ck*s + Kk);
Gk = inv(Jk)*Gt*inv(Jk');
```

2.9 Analytical

2.9.1 Parameters

Matlab
```
syms la ha m I c k positive
```

Matlab
```
Mm = [m 0 0 ;
      0 m 0 ;
      0 0 I];

Cm = [c    0   c*ha ;
      0    2*c 0 ;
      c*ha 0   c*(ha+2*la)];

Km = [k    0   k*ha ;
      0    2*k 0 ;
      k*ha 0   k*(ha+2*la)];
```

Matlab
```
Jm = [1 0  ha ;
      0 1 -la ;
      0 1  la];
```

Table 2.6: $K_{\{K\}}$

15000.0	0.0	0.0
0.0	30000.0	0.0
0.0	0.0	16912.5

Matlab
```
Mt = inv(Jm')*Mm*inv(Jm);
Ct = inv(Jm')*Cm*inv(Jm);
Kt = inv(Jm')*Km*inv(Jm);
```

Matlab
```
Jk = [1 0  0
      0 1 -la
      0 1  la];
```

Mass, Damping and Stiffness matrices expressed in $\{K\}$:

Matlab
```
Mk = Jk'*Mt*Jk;
Ck = Jk'*Ct*Jk;
Kk = Jk'*Kt*Jk;
```

Matlab
```
['\begin{equation} M_{\{K\}} = ', latex(simplify(Kk)), '\end{equation}']
```

$$M_{\{K\}} = \begin{pmatrix} k & 0 & 0 \\ 0 & 2\,k & 0 \\ 0 & 0 & k\left(-\mathrm{ha}^2 + \mathrm{ha} + 2\,\mathrm{la}\right) \end{pmatrix} \tag{2.7}$$

3 Diagonal Stiffness Matrix for a planar manipulator

3.1 Model and Assumptions

Consider a parallel manipulator with:

- b_i: location of the joints on the top platform are called b_i
- $\hat{s}_i$: unit vector corresponding to the struts
- k_i: stiffness of the struts
- τ_i: actuator forces
- O_M: center of mass of the solid body

Consider two frames:

- $\{M\}$ with origin O_M
- $\{K\}$ with origin O_K

As an example, take the system shown in Figure 3.1.

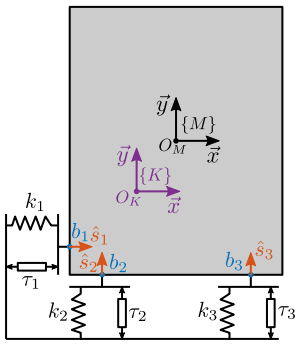

Figure 3.1: Example of 3DoF parallel platform

3.2 Objective

The objective is to find conditions for the existence of a frame $\{K\}$ in which the Stiffness matrix of the manipulator is diagonal. If the conditions are fulfilled, a second objective is to fine the location of the frame $\{K\}$ analytically.

3.3 Conditions for Diagonal Stiffness

The stiffness matrix in the frame $\{K\}$ can be expressed as:

$$K_{\{K\}} = J_{\{K\}}^T \mathcal{K} J_{\{K\}} \tag{3.1}$$

where:

- $J_{\{K\}}$ is the Jacobian transformation from the struts to the frame $\{K\}$
- $\mathcal{K}$ is a diagonal matrix with the strut stiffnesses on the diagonal

The Jacobian for a planar manipulator, evaluated in a frame $\{K\}$, can be expressed as follows:

$$J_{\{K\}} = \begin{bmatrix} {}^K\hat{s}_1^T & {}^K b_{1,x}\,{}^K\hat{s}_{1,y} - {}^K b_{1,x}\,{}^K\hat{s}_{1,y} \\ {}^K\hat{s}_2^T & {}^K b_{2,x}\,{}^K\hat{s}_{2,y} - {}^K b_{2,x}\,{}^K\hat{s}_{2,y} \\ \vdots & \vdots \\ {}^K\hat{s}_n^T & {}^K b_{n,x}\,{}^K\hat{s}_{n,y} - {}^K b_{n,x}\,{}^K\hat{s}_{n,y} \end{bmatrix} \tag{3.2}$$

Let's omit the mention of frame, it is assumed that vectors are expressed in frame $\{K\}$. It is specified otherwise.

Injecting (3.2) into (3.1) yields:

$$\boxed{K_{\{K\}} = \left[\begin{array}{c|c} k_i\hat{s}_i\hat{s}_i^T & k_i\hat{s}_i(b_{i,x}\hat{s}_{i,y} - b_{i,y}\hat{s}_{i,x}) \\ \hline k_i\hat{s}_i(b_{i,x}\hat{s}_{i,y} - b_{i,y}\hat{s}_{i,x}) & k_i(b_{i,x}\hat{s}_{i,y} - b_{i,y}\hat{s}_{i,x})^2 \end{array}\right]} \tag{3.3}$$

In order to have a decoupled stiffness matrix, we have the following two conditions:

$$k_i\hat{s}_i\hat{s}_i^T = \text{diag. matrix} \tag{3.4}$$

$$k_i\hat{s}_i(b_{i,x}\hat{s}_{i,y} - b_{i,y}\hat{s}_{i,x}) = 0 \tag{3.5}$$

Note that we don't have any condition on the term $k_i(b_{i,x}\hat{s}_{i,y} - b_{i,y}\hat{s}_{i,x})^2$ as it is only a scalar.

Condition (3.4):

- represents the coupling between translations and forces
- does only depends on the orientation of the struts and the stiffnesses and not on the choice of frame

- it is therefore a intrinsic property of the chosen geometry

Condition (3.5):

- represents the coupling between forces/rotations and torques/translation
- it does depend on the positions of the joints b_i in the frame $\{K\}$

Let's make a change of frame from the initial frame $\{M\}$ to the frame $\{K\}$:

$$^{K}b_i = {}^{M}b_i - {}^{M}O_K \tag{3.6}$$

$$^{K}\hat{s}_i = {}^{M}\hat{s}_i \tag{3.7}$$

And the goal is to find ${}^{M}O_K$ such that (3.5) is fulfilled:

$$k_i({}^{M}b_{i,x}\hat{s}_{i,y} - {}^{M}b_{i,y}\hat{s}_{i,x} - {}^{M}O_{K,x}\hat{s}_{i,y} + {}^{M}O_{K,y}\hat{s}_{i,x})\hat{s}_i = 0 \tag{3.8}$$

$$k_i({}^{M}b_{i,x}\hat{s}_{i,y} - {}^{M}b_{i,y}\hat{s}_{i,x})\hat{s}_i = {}^{M}O_{K,x}k_i\hat{s}_{i,y}\hat{s}_i - {}^{M}O_{K,y}k_i\hat{s}_{i,x}\hat{s}_i \tag{3.9}$$

And we have two sets of linear equations of two unknowns.

This can be easily solved by writing the equations in a matrix form:

$$\underbrace{k_i({}^{M}b_{i,x}\hat{s}_{i,y} - {}^{M}b_{i,y}\hat{s}_{i,x})\hat{s}_i}_{2\times 1} = \underbrace{\begin{bmatrix} k_i\hat{s}_{i,y}\hat{s}_i & -k_i\hat{s}_{i,x}\hat{s}_i \end{bmatrix}}_{2\times 2} \underbrace{\begin{bmatrix} {}^{M}O_{K,x} \\ {}^{M}O_{K,y} \end{bmatrix}}_{2\times 1} \tag{3.10}$$

And finally, if the matrix is invertible:

$$\boxed{{}^{M}O_K = \begin{bmatrix} k_i\hat{s}_{i,y}\hat{s}_i & -k_i\hat{s}_{i,x}\hat{s}_i \end{bmatrix}^{-1} k_i({}^{M}b_{i,x}\hat{s}_{i,y} - {}^{M}b_{i,y}\hat{s}_{i,x})\hat{s}_i} \tag{3.11}$$

Note that a rotation of the frame $\{K\}$ with respect to frame $\{M\}$ would make not change on the "diagonality" of $K_{\{K\}}$.

3.4 Example 1 - Planar manipulator with 3 actuators

Consider system of Figure 3.2.

The stiffnesses k_i, the joint positions ${}^{M}b_i$ and joint unit vectors ${}^{M}\hat{s}_i$ are defined below:

Matlab

```
ki = [5,1,2]; % Stiffnesses [N/m]
si = [[1;0],[0;1],[0;1]]; si = si./vecnorm(si); % Unit Vectors
bi = [[-1;0.5],[-2;-1],[0;-1]]; % Joint's positions in frame {M}
```

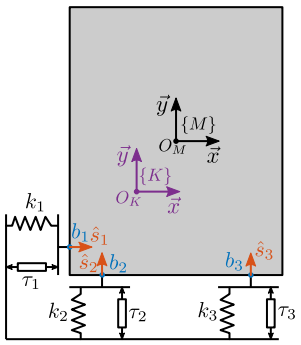

Figure 3.2: Example of 3DoF parallel platform

Let's first verify that condition (3.4) is true:

Matlab
```
ki.*si*si'
```

$$\begin{matrix} 5 & 0 \\ 0 & 2 \end{matrix}$$

Now, compute ${}^{M}O_K$:

Matlab
```
Ok = inv([sum(ki.*si(2,:).*si, 2), -sum(ki.*si(1,:).*si, 2)])*sum(ki.*(bi(1,:).*si(2,:) - bi(2,:).*si(1,:)).*si, 2);
```

Let's compute the new coordinates ${}^{K}b_i$ after the change of frame:

Matlab
```
Kbi = bi - Ok;
```

In order to verify that the new frame $\{K\}$ indeed yields a diagonal stiffness matrix, we first compute the Jacobian $J_{\{K\}}$:

Matlab
```
Jk = [si', (Kbi(1,:).*si(2,:) - Kbi(2,:).*si(1,:))'];
```

And the stiffness matrix:

Matlab
```
K = Jk'*diag(ki)*Jk
```

3.5 Example 2 - Planar manipulator with 4 actuators

Now consider the planar manipulator of Figure 3.3.

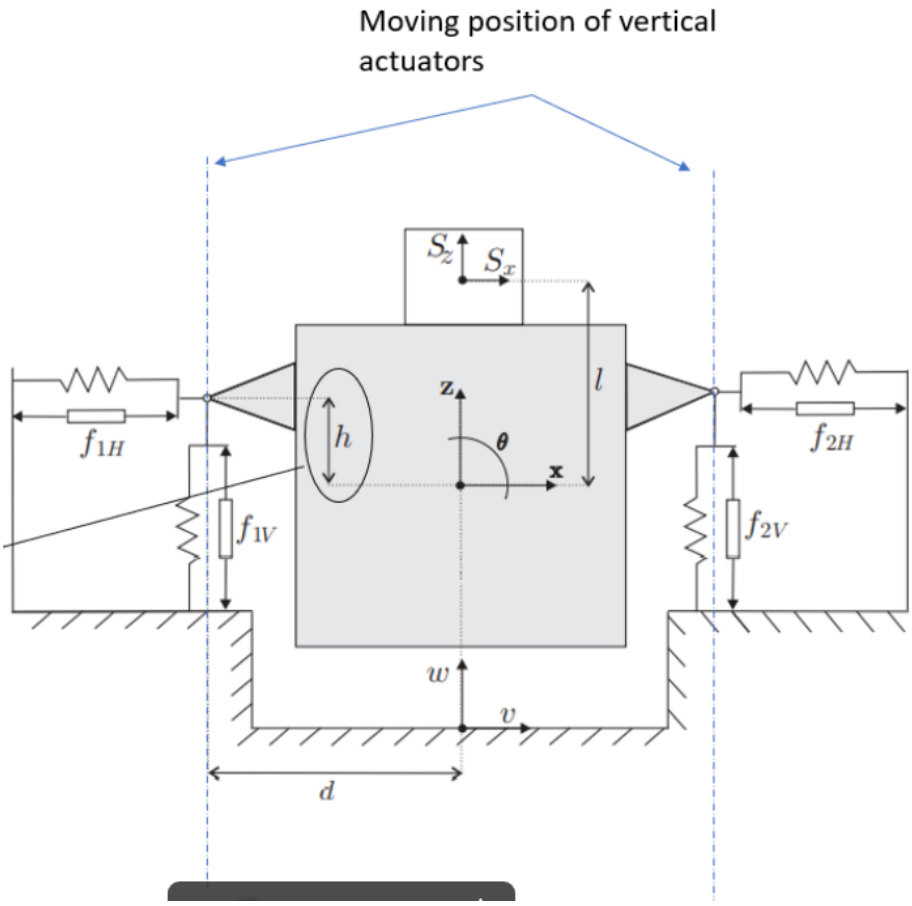

Figure 3.3: Planar Manipulator

The stiffnesses k_i, the joint positions ${}^M b_i$ and joint unit vectors ${}^M \hat{s}_i$ are defined below:

Matlab
```
ki = [1,2,1,1];
si = [[1;0],[0;1],[-1;0],[0;1]];
si = si./vecnorm(si);
h = 0.2;
L = 2;
bi = [[-L/2;h],[-L/2;-h],[L/2;h],[L/2;h]];
```

Let's first verify that condition (3.4) is true:

Matlab
```
ki.*si*si'
```

$$\begin{matrix} 2 & 0 \\ 0 & 3 \end{matrix}$$

Now, compute ${}^M O_K$:

Matlab
```
Ok = inv([sum(ki.*si(2,:).*si, 2), -sum(ki.*si(1,:).*si, 2)])*sum(ki.*(bi(1,:).*si(2,:) - bi(2,:).*si(1,:)).*si, 2);
```

Let's compute the new coordinates ${}^K b_i$ after the change of frame:

Matlab
```
Kbi = bi - Ok;
```

In order to verify that the new frame $\{K\}$ indeed yields a diagonal stiffness matrix, we first compute the Jacobian $J_{\{K\}}$:

Matlab
```
Jk = [si', (Kbi(1,:).*si(2,:) - Kbi(2,:).*si(1,:))'];
```

And the stiffness matrix:

Matlab
```
K = Jk'*diag(ki)*Jk
```

4 Diagonal Stiffness Matrix for a general parallel manipulator

4.1 Model and Assumptions

Let's consider a 6dof parallel manipulator with:

- b_i: location of the joints on the top platform are called b_i
- $\hat{s}_i$: unit vector corresponding to the struts
- k_i: stiffness of the struts
- τ_i: actuator forces
- O_M: center of mass of the solid body

Consider two frames:

- $\{M\}$ with origin O_M
- $\{K\}$ with origin O_K

An example is shown in Figure 4.1.

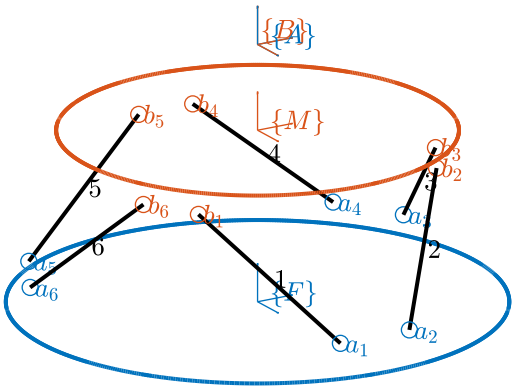

Figure 4.1: Parallel manipulator Example

4.2 Objective

The objective is to find conditions for the existence of a frame $\{K\}$ in which the Stiffness matrix of the manipulator is diagonal. If the conditions are fulfilled, a second objective is to fine the location of the frame $\{K\}$ analytically.

4.3 Analytical formula of the stiffness matrix

For a fully parallel manipulator, the stiffness matrix $K_{\{K\}}$ expressed in a frame $\{K\}$ is:

$$K_{\{K\}} = J_{\{K\}}^T \mathcal{K} J_{\{K\}} \tag{4.1}$$

where:

- $K_{\{K\}}$ is the Jacobian transformation from the struts to the frame $\{K\}$
- $\mathcal{K}$ is a diagonal matrix with the strut stiffnesses on the diagonal

The analytical expression of $J_{\{K\}}$ is:

$$J_{\{K\}} = \begin{bmatrix} {}^K\hat{s}_1^T & ({}^Kb_1 \times {}^K\hat{s}_1)^T \\ {}^K\hat{s}_2^T & ({}^Kb_2 \times {}^K\hat{s}_2)^T \\ \vdots & \vdots \\ {}^K\hat{s}_n^T & ({}^Kb_n \times {}^K\hat{s}_n)^T \end{bmatrix} \tag{4.2}$$

To simplify, we ignore the superscript K and we assume that all vectors / positions are expressed in this frame $\{K\}$. Otherwise, it is explicitly written.

Let's now write the analytical expressing of the stiffness matrix $K_{\{K\}}$:

$$K_{\{K\}} = \begin{bmatrix} \hat{s}_1 & \dots & \hat{s}_n \\ (b_1 \times \hat{s}_1) & \dots & (b_n \times \hat{s}_n) \end{bmatrix} \begin{bmatrix} k_1 & & \\ & \ddots & \\ & & k_n \end{bmatrix} \begin{bmatrix} \hat{s}_1^T & (b_1 \times \hat{s}_1)^T \\ \hat{s}_2^T & (b_2 \times \hat{s}_2)^T \\ \vdots & \dots \\ \hat{s}_n^T & (b_n \times \hat{s}_n)^T \end{bmatrix} \tag{4.3}$$

And we finally obtain:

$$\boxed{K_{\{K\}} = \left[\begin{array}{c|c} k_i\hat{s}_i\hat{s}_i^T & k_i\hat{s}_i(b_i \times \hat{s}_i)^T \\ \hline k_i\hat{s}_i(b_i \times \hat{s}_i)^T & k_i(b_i \times \hat{s}_i)(b_i \times \hat{s}_i)^T \end{array}\right]} \tag{4.4}$$

We want the stiffness matrix to be diagonal, therefore, we have the following conditions:

$$k_i\hat{s}_i\hat{s}_i^T = \text{diag. matrix} \tag{4.5}$$

$$k_i(b_i \times \hat{s}_i)(b_i \times \hat{s}_i)^T = \text{diag. matrix} \tag{4.6}$$

$$k_i\hat{s}_i(b_i \times \hat{s}_i)^T = 0 \tag{4.7}$$

Note that:

- condition (4.5) corresponds to coupling between forces applied on O_K to translations of the payload. It does not depend on the choice of $\{K\}$, it only depends on the orientation of the struts and the stiffnesses. It is therefore an intrinsic property of the manipulator.

- condition (4.6) corresponds to the coupling between forces applied on O_K and rotation of the payload. Similarly, it does also correspond to the coupling between torques applied on O_K to translations of the payload.

- condition (4.7) corresponds to the coupling between torques applied on O_K to rotation of the payload.

- conditions (4.6) and (4.7) do depend on the positions ${}^K b_i$ and therefore depend on the choice of $\{K\}$.

Note that if we find a frame $\{K\}$ in which the stiffness matrix $K_{\{K\}}$ is diagonal, it will still be diagonal for any rotation of the frame $\{K\}$. Therefore, we here suppose that the frame $\{K\}$ is aligned with the initial frame $\{M\}$.

Let's make a change of frame from the initial frame $\{M\}$ to the frame $\{K\}$:

$$ {}^K b_i = {}^M b_i - {}^M O_K \tag{4.8} $$

$$ {}^K \hat{s}_i = {}^M \hat{s}_i \tag{4.9} $$

The goal is to find ${}^M O_K$ such that conditions (4.6) and (4.7) are fulfilled.

Let's first solve equation (4.7) that corresponds to the coupling between forces and rotations:

$$ k_i \hat{s}_i (({}^M b_i - {}^M O_K) \times \hat{s}_i)^T = 0 \tag{4.10} $$

Taking the transpose and re-arranging:

$$ k_i ({}^M b_i \times \hat{s}_i) \hat{s}_i^T = k_i ({}^M O_K \times \hat{s}_i) \hat{s}_i^T \tag{4.11} $$

As the vector cross product also can be expressed as the product of a skew-symmetric matrix and a vector, we obtain:

$$ k_i ({}^M b_i \times \hat{s}_i) \hat{s}_i^T = {}^M \boldsymbol{O}_K (k_i \hat{s}_i \hat{s}_i^T) \tag{4.12} $$

with:

$$ {}^M \boldsymbol{O}_K = \begin{bmatrix} 0 & -{}^M O_{K,z} & {}^M O_{K,y} \\ {}^M O_{K,z} & 0 & -{}^M O_{K,x} \\ -{}^M O_{K,y} & {}^M O_{K,x} & 0 \end{bmatrix} \tag{4.13} $$

We suppose $k_i \hat{s}_i \hat{s}_i^T$ invertible as it is diagonal from (4.5).

And finally, we find:

$$ \boxed{ {}^M \boldsymbol{O}_K = \left(k_i ({}^M b_i \times \hat{s}_i) \hat{s}_i^T \right) \cdot \left(k_i \hat{s}_i \hat{s}_i^T \right)^{-1} } \tag{4.14} $$

If the obtained ${}^M \boldsymbol{O}_K$ is a skew-symmetric matrix, we can easily determine the corresponding vector ${}^M O_K$ from (4.13).

In such case, condition (4.6) is fulfilled and there is no coupling between translations and rotations in the frame $\{K\}$.

Then, we can only verify if condition (4.7) is verified or not.

Note

If there is no frame $\{K\}$ such that conditions (4.6) and (4.7) are valid, it would be interesting to be able to determine the frame $\{K\}$ in which is coupling is minimal.

4.4 Example 1 - 6DoF manipulator (3D)

Let's define the geometry of the manipulator ($^M b_i$, $^M s_i$ and k_i):

Matlab
```
ki = [2,2,1,1,3,3,1,1,1,1,2,2];
si = [[-1;0;0],[-1;0;0],[-1;0;0],[-1;0;0],[0;0;1],[0;0;1],[0;0;1],[0;0;1],[0;-1;0],[0;-1;0],[0;-1;0],[0;-1;0]];
bi =
↪  [[1;-1;1],[1;1;-1],[1;1;1],[1;-1;-1],[1;-1;-1],[-1;1;-1],[1;1;-1],[-1;-1;-1],[1;1;-1],[-1;1;1],[-1;1;-1],[1;1;1]]-[0;2;-1];
```

Cond 1:

Matlab
```
ki.*si*si'
```

$$\begin{matrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 8 \end{matrix}$$

Find Ok

Matlab
```
OkX = (ki.*cross(bi, si)*si')/(ki.*si*si');

if all(diag(OkX) == 0) && all(all((OkX + OkX') == 0))
    disp('OkX is skew symmetric')
    Ok = [OkX(3,2);OkX(1,3);OkX(2,1)]
else
    error('OkX is *not* skew symmetric')
end
```

$$\begin{matrix} 0 \\ -2 \\ 1 \end{matrix}$$

Matlab
```
% Verification of second condition
si*cross(bi-Ok, si)'
```

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Verification of third condition

Matlab
```
ki.*cross(bi-Ok, si)*cross(bi-Ok, si)'
```

$$\begin{matrix} 14 & 4 & -2 \\ 4 & 14 & 2 \\ -2 & 2 & 12 \end{matrix}$$

Let's compute the Jacobian:

Matlab
```
Jk = [si', cross(bi - Ok, si)'];
```

And the stiffness matrix:

Matlab
```
Jk'*diag(ki)*Jk
```

$$\begin{matrix} 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 4 & -2 \\ 0 & 0 & 0 & 4 & 14 & 2 \\ 0 & 0 & 0 & -2 & 2 & 12 \end{matrix}$$

Matlab
```
figure;
hold on;
set(gca,'ColorOrderIndex',1)
plot(b1(1), b1(2), 'o');
set(gca,'ColorOrderIndex',2)
plot(b2(1), b2(2), 'o');
set(gca,'ColorOrderIndex',3)
plot(b3(1), b3(2), 'o');
set(gca,'ColorOrderIndex',1)
quiver(b1(1),b1(2),0.1*s1(1),0.1*s1(2))
set(gca,'ColorOrderIndex',2)
quiver(b2(1),b2(2),0.1*s2(1),0.1*s2(2))
set(gca,'ColorOrderIndex',3)
quiver(b3(1),b3(2),0.1*s3(1),0.1*s3(2))

plot(0, 0, 'ko');
quiver([0,0],[0,0],[0.1,0],[0,0.1], 'k')

plot(Ok(1), Ok(2), 'ro');
quiver([Ok(1),Ok(1)],[Ok(2),Ok(2)],[0.1,0],[0,0.1], 'r')

hold off;
axis equal;
```

4.5 Example 2 - Stewart Platform

5 Stewart Platform - Simscape Model

In this analysis, we wish to applied SVD control to the Stewart Platform shown in Figure 5.1.

Some notes about the system:

- 6 voice coils actuators are used to control the motion of the top platform.
- the motion of the top platform is measured with a 6-axis inertial unit (3 acceleration + 3 angular accelerations)
- the control objective is to isolate the top platform from vibrations coming from the bottom platform

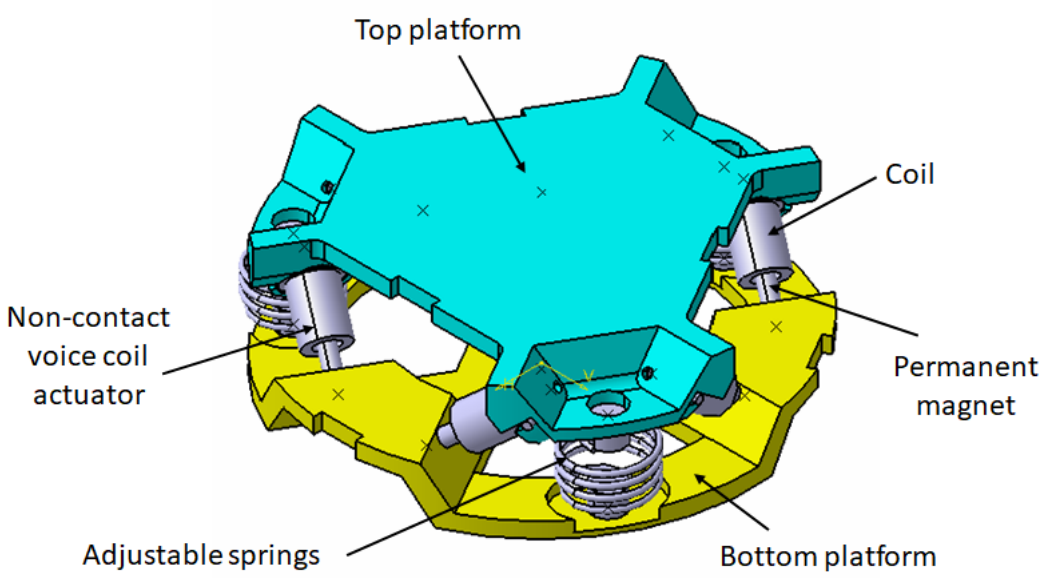

Figure 5.1: Stewart Platform CAD View

The analysis of the SVD/Jacobian control applied to the Stewart platform is performed in the following sections:

- Section 5.1: The parameters of the Simscape model of the Stewart platform are defined
- Section 5.2: The plant is identified from the Simscape model and the system coupling is shown
- Section 5.3: The plant is first decoupled using the Jacobian
- Section 5.4: The decoupling is performed thanks to the SVD. To do so a real approximation of

the plant is computed.

- Section 5.5: The effectiveness of the decoupling with the Jacobian and SVD are compared using the Gershorin Radii
- Section 5.6:
- Section 5.7: The dynamics of the decoupled plants are shown
- Section 5.8: A diagonal controller is defined to control the decoupled plant
- Section 5.9: Finally, the closed loop system properties are studied

5.1 Simscape Model - Parameters

```Matlab
open('drone_platform.slx');
```

Definition of spring parameters:

```Matlab
kx = 0.5*1e3/3; % [N/m]
ky = 0.5*1e3/3;
kz = 1e3/3;

cx = 0.025; % [Nm/rad]
cy = 0.025;
cz = 0.025;
```

We suppose the sensor is perfectly positioned.

```Matlab
sens_pos_error = zeros(3,1);
```

Gravity:

```Matlab
g = 0;
```

We load the Jacobian (previously computed from the geometry):

```Matlab
load('jacobian.mat', 'Aa', 'Ab', 'As', 'l', 'J');
```

We initialize other parameters:

```Matlab
U = eye(6);
V = eye(6);
Kc = tf(zeros(6));
```

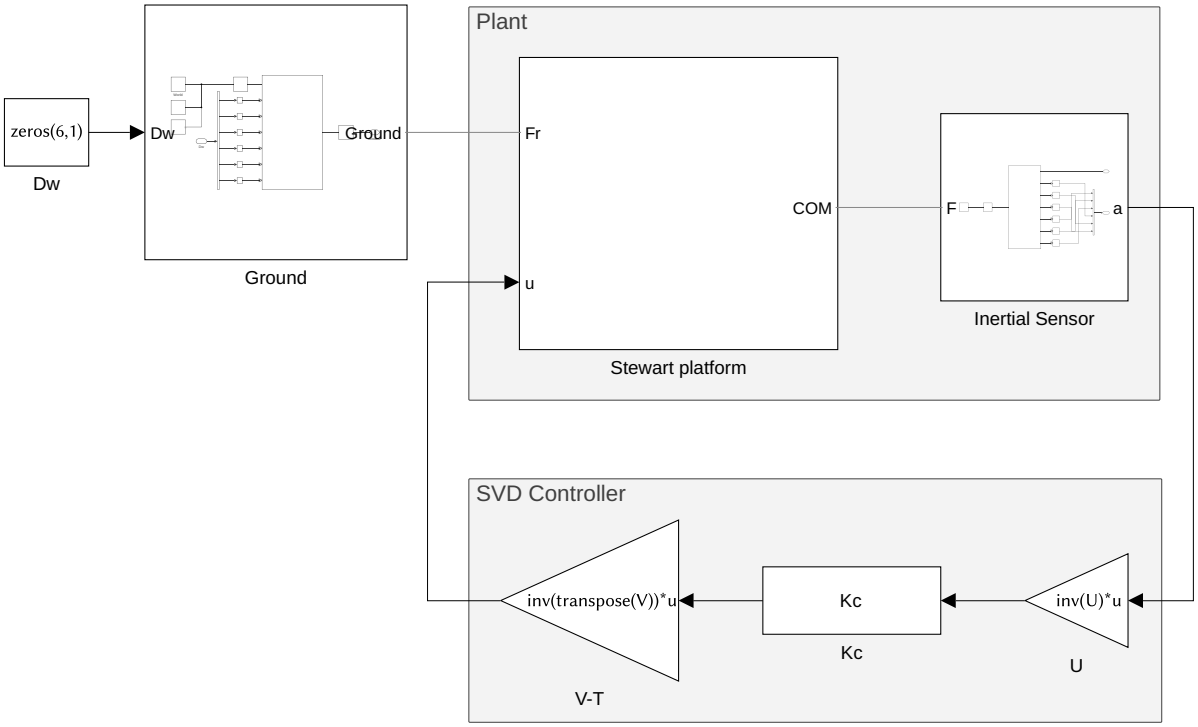

Figure 5.2: General view of the Simscape Model

5.2 Identification of the plant

The plant shown in Figure 5.4 is identified from the Simscape model.

The inputs are:

- D_w translation and rotation of the bottom platform (with respect to the center of mass of the top platform)
- τ the 6 forces applied by the voice coils

The outputs are the 6 accelerations measured by the inertial unit.

```Matlab
%% Name of the Simulink File
mdl = 'drone_platform';

%% Input/Output definition
clear io; io_i = 1;
io(io_i) = linio([mdl, '/Dw'],              1, 'openinput');  io_i = io_i + 1; % Ground Motion
io(io_i) = linio([mdl, '/V-T'],             1, 'openinput');  io_i = io_i + 1; % Actuator Forces
io(io_i) = linio([mdl, '/Inertial Sensor'], 1, 'openoutput'); io_i = io_i + 1; % Top platform acceleration
```

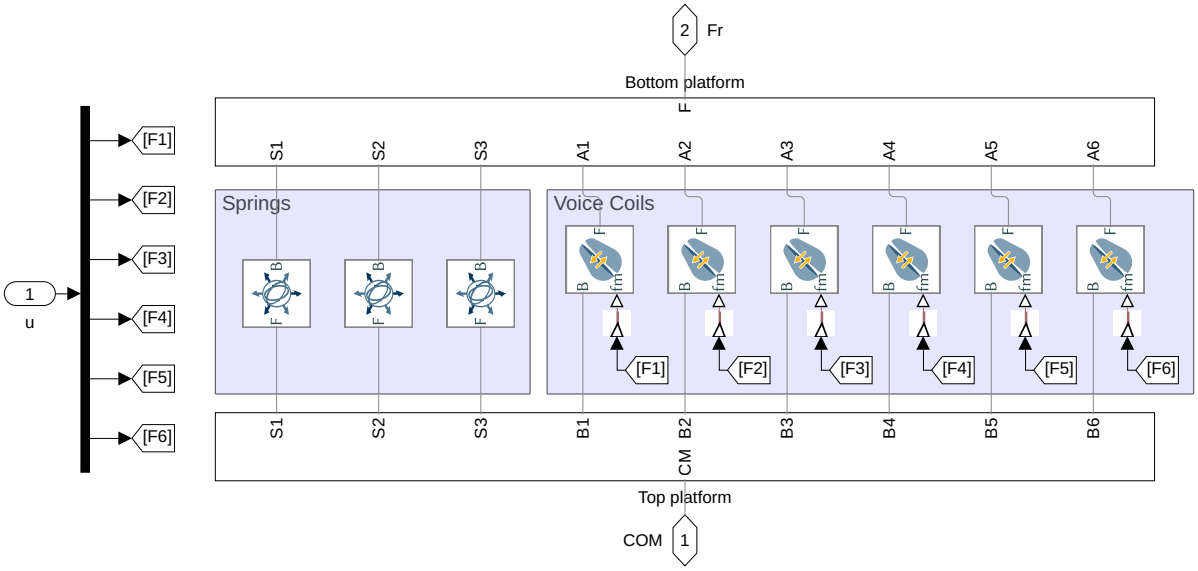

Figure 5.3: Simscape model of the Stewart platform

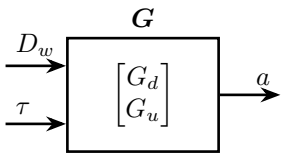

Figure 5.4: Considered plant $\boldsymbol{G} = \begin{bmatrix} G_d \\ G_u \end{bmatrix}$. D_w is the translation/rotation of the support, τ the actuator forces, a the acceleration/angular acceleration of the top platform

```
G = linearize(mdl, io);
G.InputName  = {'Dwx', 'Dwy', 'Dwz', 'Rwx', 'Rwy', 'Rwz', ...
                'F1', 'F2', 'F3', 'F4', 'F5', 'F6'};
G.OutputName = {'Ax', 'Ay', 'Az', 'Arx', 'Ary', 'Arz'};

% Plant
Gu = G(:, {'F1', 'F2', 'F3', 'F4', 'F5', 'F6'});
% Disturbance dynamics
Gd = G(:, {'Dwx', 'Dwy', 'Dwz', 'Rwx', 'Rwy', 'Rwz'});
```

There are 24 states (6dof for the bottom platform + 6dof for the top platform).

Matlab
```
size(G)
```

Results
```
State-space model with 6 outputs, 12 inputs, and 24 states.
```

The elements of the transfer matrix $\boldsymbol{G}$ corresponding to the transfer function from actuator forces τ to the measured acceleration a are shown in Figure 5.5.

One can easily see that the system is strongly coupled.

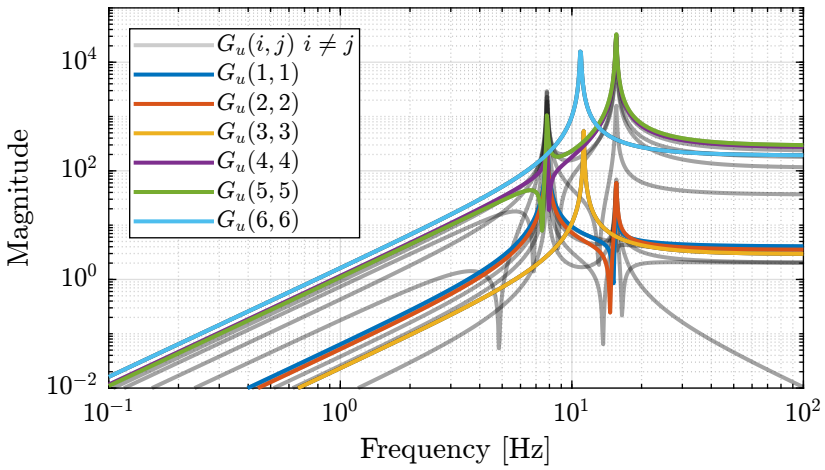

Figure 5.5: Magnitude of all 36 elements of the transfer function matrix G_u

5.3 Decoupling using the Jacobian

Consider the control architecture shown in Figure 5.6. The Jacobian matrix is used to transform forces/torques applied on the top platform to the equivalent forces applied by each actuator.

The Jacobian matrix is computed from the geometry of the platform (position and orientation of the actuators).

Table 5.1: Computed Jacobian Matrix

0.811	0.0	0.584	-0.018	-0.008	0.025
-0.406	-0.703	0.584	-0.016	-0.012	-0.025
-0.406	0.703	0.584	0.016	-0.012	0.025
0.811	0.0	0.584	0.018	-0.008	-0.025
-0.406	-0.703	0.584	0.002	0.019	0.025
-0.406	0.703	0.584	-0.002	0.019	-0.025

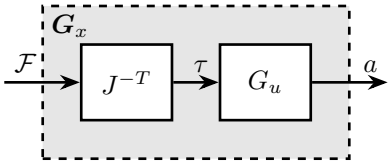

Figure 5.6: Decoupled plant $\boldsymbol{G}_x$ using the Jacobian matrix J

We define a new plant:

$$G_x(s) = G(s)J^{-T}$$

$G_x(s)$ correspond to the transfer function from forces and torques applied to the top platform to the absolute acceleration of the top platform.

Matlab
```
Gx = Gu*inv(J');
Gx.InputName  = {'Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz'};
```

5.4 Decoupling using the SVD

In order to decouple the plant using the SVD, first a real approximation of the plant transfer function matrix as the crossover frequency is required.

Let's compute a real approximation of the complex matrix H_1 which corresponds to the the transfer function $G_u(j\omega_c)$ from forces applied by the actuators to the measured acceleration of the top platform evaluated at the frequency ω_c.

Matlab
```
wc = 2*pi*30; % Decoupling frequency [rad/s]

H1 = evalfr(Gu, j*wc);
```

The real approximation is computed as follows:

Matlab
```
D = pinv(real(H1'*H1));
H1 = inv(D*real(H1'*diag(exp(j*angle(diag(H1*D*H1.'))/2))));
```

Note that the plant G_u at ω_c is already an almost real matrix. This can be seen on the Bode plots where the phase is close to 1. This can be verified below where only the real value of $G_u(\omega_c)$ is shown

Table 5.2: Real approximate of G at the decoupling frequency ω_c

4.4	-2.1	-2.1	4.4	-2.4	-2.4
-0.2	-3.9	3.9	0.2	-3.8	3.8
3.4	3.4	3.4	3.4	3.4	3.4
-367.1	-323.8	323.8	367.1	43.3	-43.3
-162.0	-237.0	-237.0	-162.0	398.9	398.9
220.6	-220.6	220.6	-220.6	220.6	-220.6

Table 5.3: Real part of G at the decoupling frequency ω_c

4.4	-2.1	-2.1	4.4	-2.4	-2.4
-0.2	-3.9	3.9	0.2	-3.8	3.8
3.4	3.4	3.4	3.4	3.4	3.4
-367.1	-323.8	323.8	367.1	43.3	-43.3
-162.0	-237.0	-237.0	-162.0	398.9	398.9
220.6	-220.6	220.6	-220.6	220.6	-220.6

Now, the Singular Value Decomposition of H_1 is performed:

$$H_1 = U\Sigma V^H$$

Matlab
```
[U,~,V] = svd(H1);
```

Table 5.4: Obtained matrix U

-0.005	7e-06	6e-11	-3e-06	-1	0.1
-7e-06	-0.005	-9e-09	-5e-09	-0.1	-1
4e-08	-2e-10	-6e-11	-1	3e-06	-3e-07
-0.002	-1	-5e-06	2e-10	0.0006	0.005
1	-0.002	-1e-08	2e-08	-0.005	0.0006
-4e-09	5e-06	-1	6e-11	-2e-09	-1e-08

The obtained matrices U and V are used to decouple the system as shown in Figure 5.7.

The decoupled plant is then:

$$G_{SVD}(s) = U^{-1}G_u(s)V^{-H}$$

Matlab
```
Gsvd = inv(U)*Gu*inv(V');
```

5.5 Verification of the decoupling using the "Gershgorin Radii"

The "Gershgorin Radii" is computed for the coupled plant $G(s)$, for the "Jacobian plant" $G_x(s)$ and the "SVD Decoupled Plant" $G_{SVD}(s)$:

Table 5.5: Obtained matrix V

-0.2	0.5	-0.4	-0.4	-0.6	-0.2
-0.3	0.5	0.4	-0.4	0.5	0.3
-0.3	-0.5	-0.4	-0.4	0.4	-0.4
-0.2	-0.5	0.4	-0.4	-0.5	0.3
0.6	-0.06	-0.4	-0.4	0.1	0.6
0.6	0.06	0.4	-0.4	-0.006	-0.6

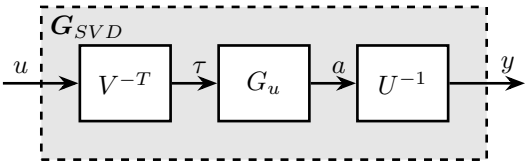

Figure 5.7: Decoupled plant G_{SVD} using the Singular Value Decomposition

The "Gershgorin Radii" of a matrix S is defined by:

$$\zeta_i(j\omega) = \frac{\sum\limits_{j \neq i} |S_{ij}(j\omega)|}{|S_{ii}(j\omega)|}$$

This is computed over the following frequencies.

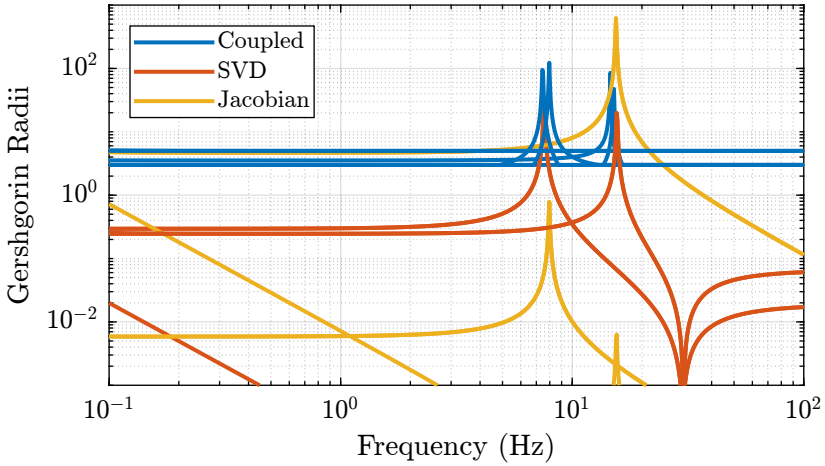

Figure 5.8: Gershgorin Radii of the Coupled and Decoupled plants

5.6 Verification of the decoupling using the "Relative Gain Array"

The relative gain array (RGA) is defined as:

$$\Lambda\big(G(s)\big) = G(s) \times \big(G(s)^{-1}\big)^T \tag{5.1}$$

where $\times$ denotes an element by element multiplication and $G(s)$ is an $n \times n$ square transfer matrix.

The obtained RGA elements are shown in Figure 5.9.

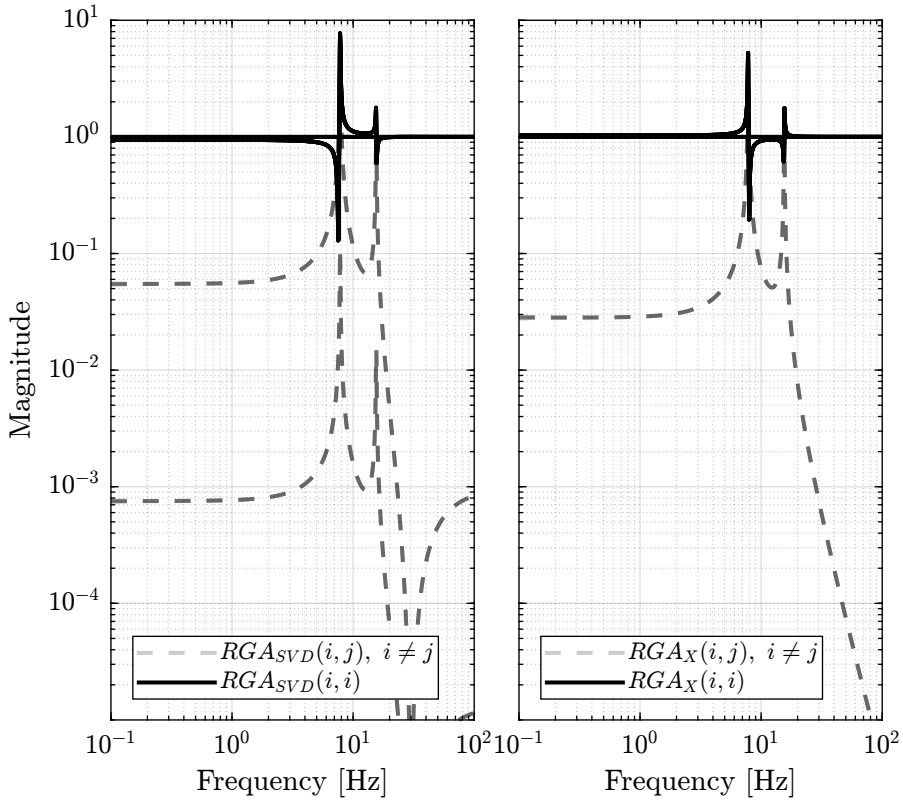

Figure 5.9: Obtained norm of RGA elements for the SVD decoupled plant and the Jacobian decoupled plant

5.7 Obtained Decoupled Plants

The bode plot of the diagonal and off-diagonal elements of G_{SVD} are shown in Figure 5.10.

Similarly, the bode plots of the diagonal elements and off-diagonal elements of the decoupled plant $G_x(s)$ using the Jacobian are shown in Figure 5.11.

5.8 Diagonal Controller

The control diagram for the centralized control is shown in Figure 5.12.

The controller K_c is "working" in an cartesian frame. The Jacobian is used to convert forces in the cartesian frame to forces applied by the actuators.

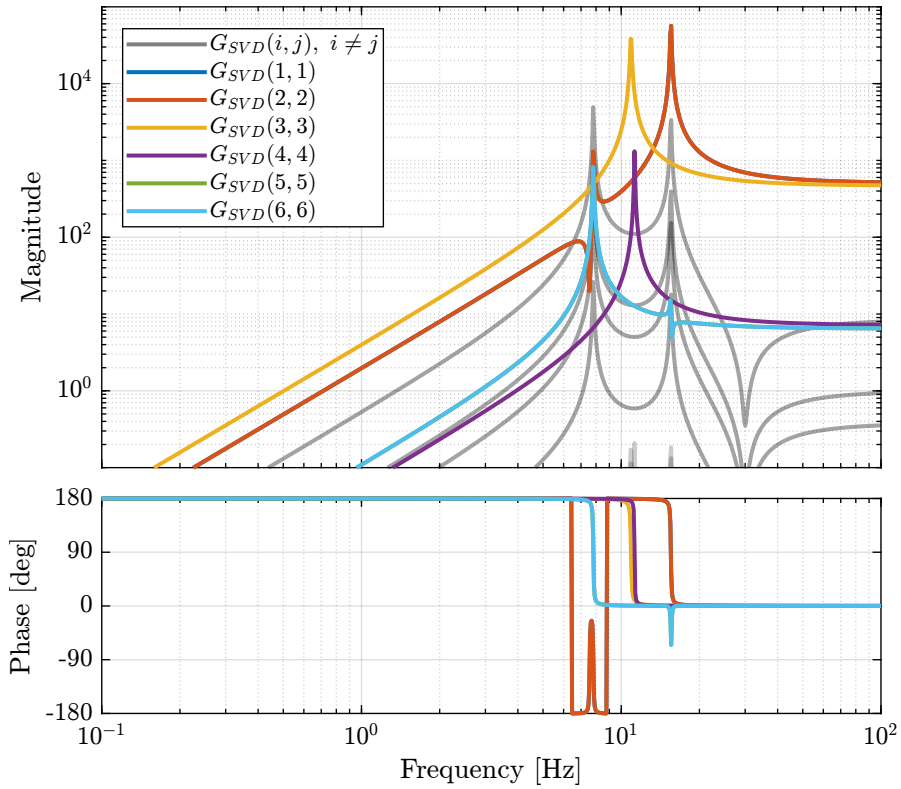

Figure 5.10: Decoupled Plant using SVD

The SVD control architecture is shown in Figure 5.13. The matrices U and V are used to decoupled the plant G.

We choose the controller to be a low pass filter:

$$K_c(s) = \frac{G_0}{1 + \frac{s}{\omega_0}}$$

G_0 is tuned such that the crossover frequency corresponding to the diagonal terms of the loop gain is equal to ω_c

Matlab
```
wc = 2*pi*80;  % Crossover Frequency [rad/s]
w0 = 2*pi*0.1; % Controller Pole [rad/s]
```

Matlab
```
K_cen = diag(1./diag(abs(evalfr(Gx, j*wc))))*(1/abs(evalfr(1/(1 + s/w0), j*wc)))/(1 + s/w0);
L_cen = K_cen*Gx;
G_cen = feedback(G, pinv(J')*K_cen, [7:12], [1:6]);
```

Matlab
```
K_svd = diag(1./diag(abs(evalfr(Gsvd, j*wc))))*(1/abs(evalfr(1/(1 + s/w0), j*wc)))/(1 + s/w0);
L_svd = K_svd*Gsvd;
G_svd = feedback(G, inv(V')*K_svd*inv(U), [7:12], [1:6]);
```

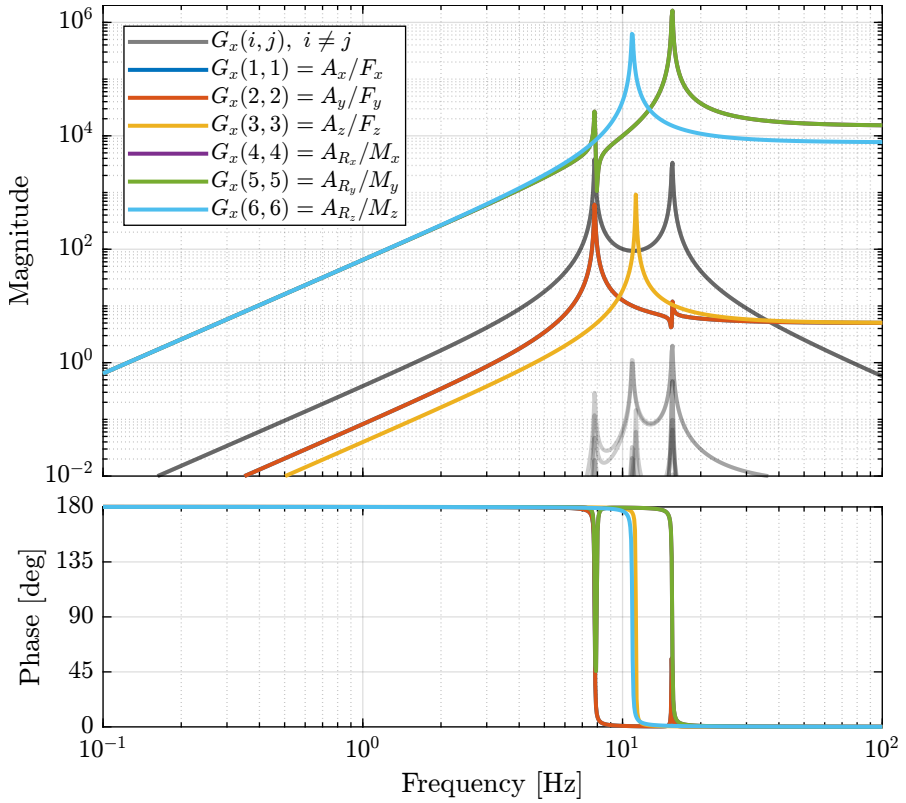

Figure 5.11: Stewart Platform Plant from forces (resp. torques) applied by the legs to the acceleration (resp. angular acceleration) of the platform as well as all the coupling terms between the two (non-diagonal terms of the transfer function matrix)

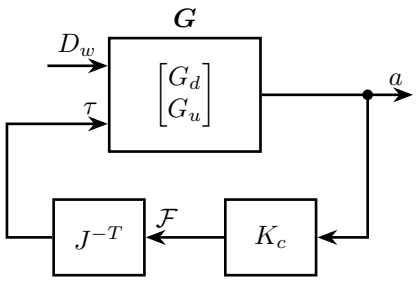

Figure 5.12: Control Diagram for the Centralized control

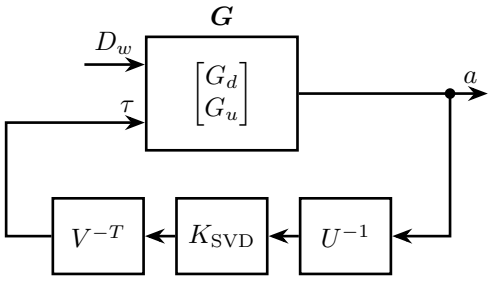

Figure 5.13: Control Diagram for the SVD control

The obtained diagonal elements of the loop gains are shown in Figure 5.14.

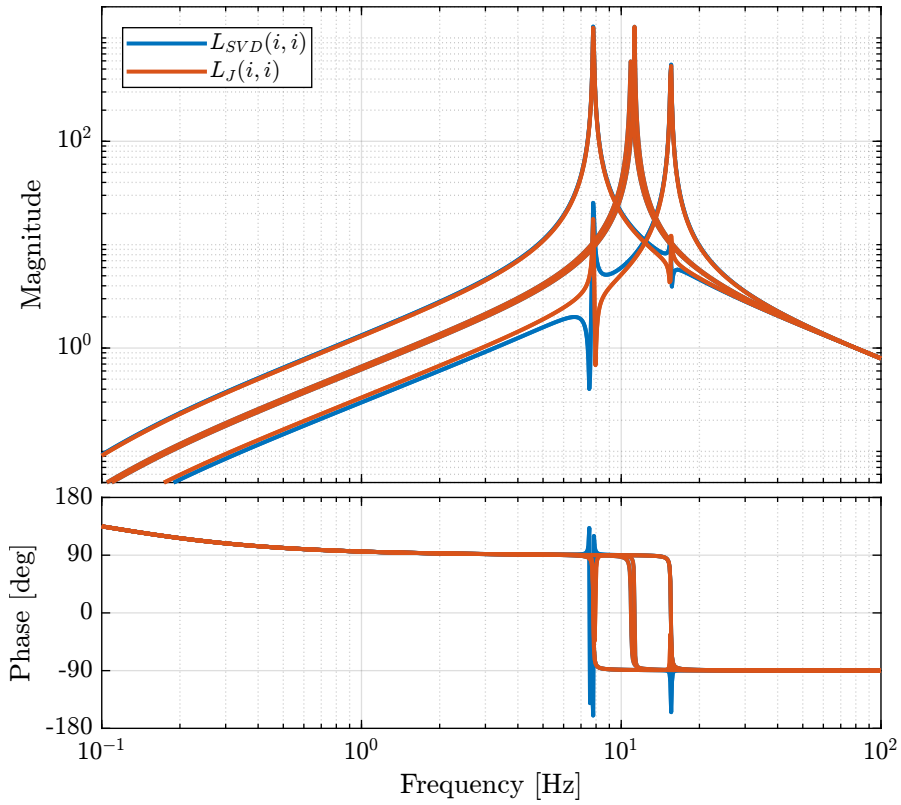

Figure 5.14: Comparison of the diagonal elements of the loop gains for the SVD control architecture and the Jacobian one

5.9 Closed-Loop system Performances

Let's first verify the stability of the closed-loop systems:

Matlab
```
isstable(G_cen)
```

Results
```
ans =
  logical
   1
```

Matlab
```
isstable(G_svd)
```

```
Results
ans =
  logical
   1
```

The obtained transmissibility in Open-loop, for the centralized control as well as for the SVD control are shown in Figure 5.15.

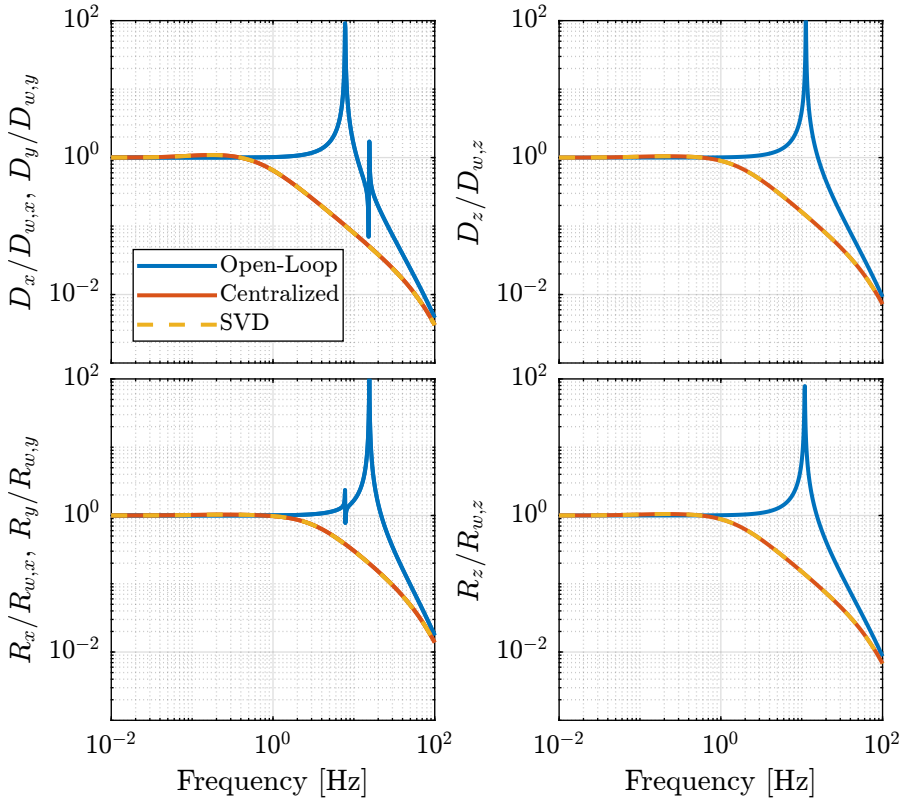

Figure 5.15: Obtained Transmissibility