Internship report: Design of a Matlab based graphical user interface for visualization and enhancement of floor dynamics

F.B.Koopman
s0166626
Document name: Report_fkoo.docx
Date: 23 July 2018
MECAL, Enschede, The Netherlands
Supervisor Mecal: Servaas Bank
Internship period: 1/5/2018-31/7/2018
University of Twente
Faculty of Engineering Technology
Departments of Mechanics of Solids, Surfaces and Systems
Supervisor UT: Prof.dr.ir. A. de Boer

Internship report: Design of a Matlab based graphical user interface for visualization and enhancement of floor dynamics

Document name:	Report_fkoo.docx
Date:	23/07/18
Written by:	Frederik Koopman, MECAL, Enschede, The Netherlands
Reviewed by:	Servaas Bank, MECAL, Enschede, The Netherlands
Approved by:	Servaas Bank, MECAL, Enschede, The Netherlands
Document by:	MECAL, Enschede, The Netherlands
Mailing list:	MECAL (1x)
Pages incl. cover:	151
Keywords:	Floor vibrations, floor vibrations reduction, floor dynamics
Language:	English
Software version:	Microsoft Word 2016

Contributing Authors	
Name Author	**Part(s) of this document**
Frederik Koopman	All

Document change and history record			
Version	**Date**	**Page (s)**	**Short description**
1.0	23/07/18	151	First Issue

©MECAL 2018. All rights reserved.
Reproduction in whole or in part is prohibited without the prior written consent of MECAL

Summary

The research documented in this report is about gaining insight into the dynamics of floor and doing simulations of existing floors in altered situations. To get better insight into the dynamics of floor first a floor was modelled in more and more complex ways but more and more lifelike by means of a Matlab tools. This is done in Chapter 2. Next it was tried to get a good simple model from a measured floor, see Chapter 3, by estimating the mass, damping and stiffness matrix by a variety of techniques. This was not done successful on a real floor as will come to light in Chapter 5. From the estimation method tried, a simple method came to light for altering the floor dynamics to new situations. This method relies on inverting a system of frequency response functions to get the dynamic matrix. Next elements can be added to this matrix and inverted back to get the new frequency response functions. A graphical user interface based on this method was made, as will be addressed in Chapter 4.

©MECAL 2018 Confidential

Table of contents

©MECAL 2018 Confidential

©MECAL 2018 Confidential

Preface and acknowledgements

This report is drafted for MECAL and University of Twente, as a conclusion of my 3 months internship at MECAL from May 1, 2018 until July 31, 2018. During the final year of the master Mechanical Engineering (Mechanics of Solids, Surfaces and Systems) at the University of Twente, an internship at an external party must be done. The goal of this internship is to acquiring work experience outside University of Twente of a trainee or recently graduated engineer level with the main objective of putting acquired knowledge and skills into practice while the experiences of applying and writing reports as well as working in expert teams play an important role too.

I chose to do my internship at MECAL since my internship options abroad were cancelled and I needed an internship fast, this was possible at MECAL. Nevertheless I had a great time during my internship. While my master was focused mainly on mechatronics this internship was mostly dynamics.

Information on MECAL can be found in Appendix A.

Acknowledgments

The opportunity for doing my internship at MECAL was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

I would like to thank Servaas Bank in particular for not only being my daily supervisor but also learning me new a different way of looking at dynamics. Also I would like to thank Ronald Rijkers for thinking up extra features to add the graphical user interface.

©MECAL 2018 Confidential

Chapter 1 Introduction

Mecal High-tech/Systems solves vibration problems for the semiconductor fabrication industry. Mostly the vibration problems are solved with machine support frames or pedestals, structural building improvements or by use of EQUALIZER(s) [1] (EQ). The EQ is an active inertial damper system. A big disadvantage of this method is that it is very time consuming [2].

To get the best results with the EQ the floor needs to be modeled. At the moment this is done based on fitted FEM calculations, or the EQ are placed using common sense. Previous interns have tried to using the modal description [3] to get an model for the floor. This method works by getting the mode shapes at the corresponding eigenfrequencies. The more eigenmodes are known the more accurate the model becomes. A big advantage of this method is that not all transfer needs to be known just all transfer from all inputs to a single output. The eigenfrequencies are found by looking for peaks in the absolute value of the transfer or points were the phase of the transfer passes the points -90 or 90 degrees with a negative slope. The corresponding mode shapes are gotten by taking the imaginary part of the transfer at these frequency. This method is known under the name quadrature picking.

While this method works well in theory, in practice there are a few issues. Firstly with systems that have a lot of damping or having eigenfrequencies that are close together the eigenfrequencies are not easily identified which can lead to missing eigenmodes. Also the transfers need to have enough eigenfrequencies to get enough eigenmodes, so measurements needs to have high frequency data.

1.1 Assignment description

At the starting of this internship the assignment was only a global descripting because it was yet unsure which direction this internship will take. The general idea was to get an even better understanding of vibrations in floors, how to model it and to be able to measure these vibrations and process the results more efficiently. This last step is still done with custom Matlab scripts, FEM calculations and simple tests for each case.

1.2 Outline of the report

At MECAL, a Matlab based insight tool existed to get an better understanding of floor vibrations. This tool was very limited, therefore one of the first things done for this internship was expanding this tool. What was done is explained in detail in Chapter 2. For the next part of this internship it was tried to estimate a good, simple model of a floor from measurements. How this was done can be found in Chapter 3. Although this worked very well in theory, for a real floor it was not successful. A lesson learned from the estimation method is the base for the Matlab based GUI for visualization and enhancement of floor dynamics. What this GUI can do and how it works is described in Chapter 4. A test case was setup up to test ideas from Chapter 3 and Chapter 4. In Chapter 6 conclusions and recommendations will be addressed. Appendix B contains a reflection of my functioning within MECAL.

©MECAL 2018 Confidential

Chapter 2 Floor insight GUI

2.1 Existing version: 1D

The original floor GUI, see Figure 1, made by Servaas Bank is an insight tool for vibrations in a floor. This GUI approximates the floor as a series of masses connected by springs that only move in the vertical direction, see Figure 2. The floor is assumed to be homogeneous in mass and stiffness. The floor is connected to the fixed world by support springs. The tool asks the user for the number of masses it should use, the stiffness in the middle of the floor the support spring stiffness, the total mass of the floor and the damping coefficient for both the floor and support.

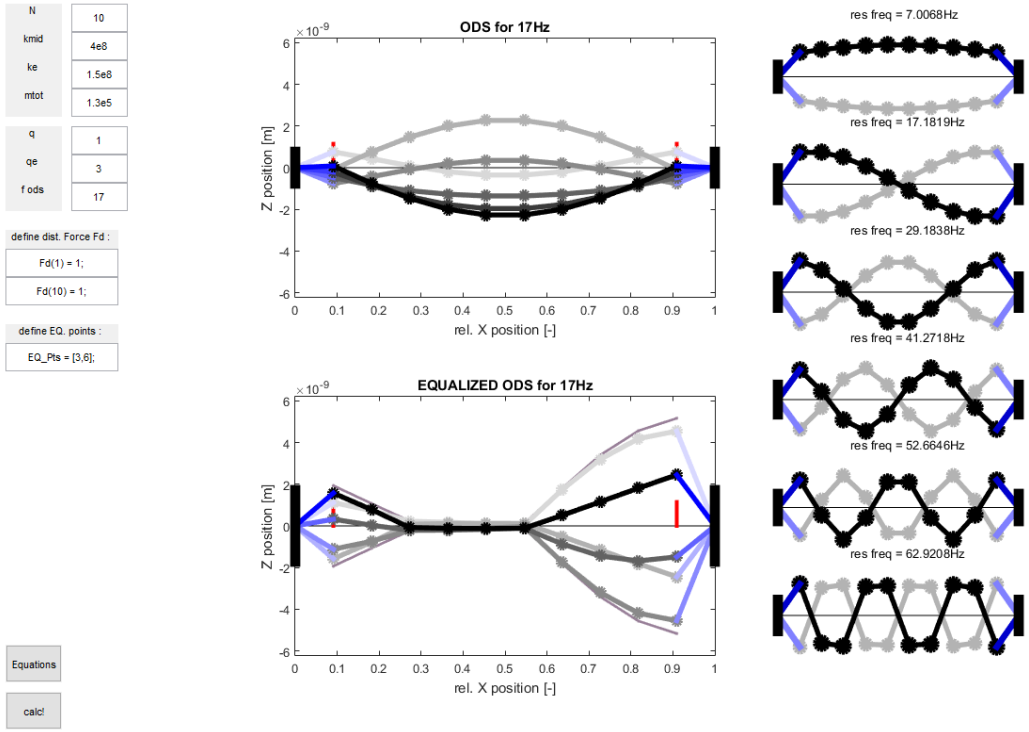


Figure 1: 1D floor GUI

The tool calculates the first six eigenmodes of the system and displays them on the right. Also the user can give 1 or more harmonic forces on the masses and the response is calculated and displayed in the middle top figure. The user can also give the locations of EQs and the response with the EQs is calculated in the middle bottom figure.

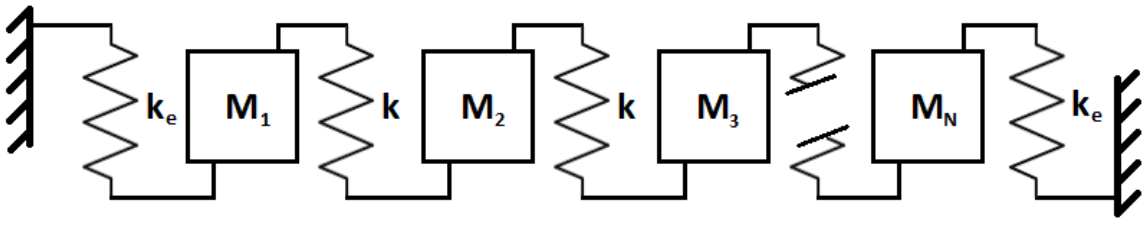


Figure 2: 1D floor system

©MECAL 2018 Confidential

The tool works by making a global stiffness, damping and mass matrices from the inputs. Then the eigenmodes and eigenfrequencies are calculated by the function eig from Matlab from the mass and stiffness matrix. The harmonic response is calculated by making the dynamic matrix from equation:
$D(\omega) = -\omega^2 M + i\omega C + K$
Since the user has given the frequency this can be solved. Next the external force vector is made from the user input and the response is calculated from D\F and animated. The same is done for the animation for the response with EQs. For this it is assumed that the stiffness of a point where an EQ is placed will become 10 times as stiff as before. Therefore only the stiffness matrix is altered. The EQ can in this case be thought of as an extra spring connecting the point it is placed on to the fixed world.

2.2 First new version: 2D

To get even better insight in the vibrations of floor this tool was adjusted to be 2D. A rotation was added. For this the floor is assumed to be a series of beam elements connected to the fixed world by translation and rotation springs, see Figure 3.

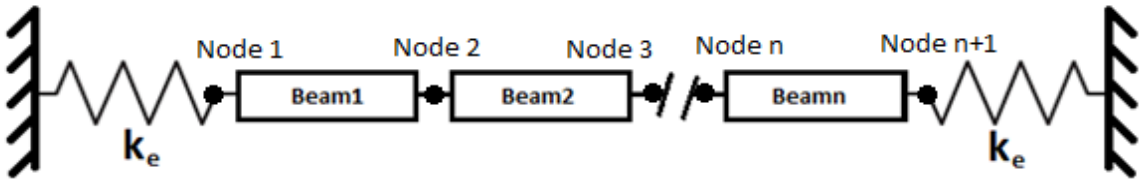


Figure 3: 2D floor system

The beam elements has the element stiffness, mass matrix [4] with coordinate system where the beam is connected between node i and j:

$$K_e = \frac{EI}{L_e^3}\begin{bmatrix} 12 & 6 & -12 & 6 \\ 6 & 4 & -6 & 2 \\ -12 & -6 & 12 & -6 \\ 6 & 2 & -6 & 4 \end{bmatrix}$$

$$M_e = \frac{m_e}{420}\begin{bmatrix} 156 & 22 & 54 & -13 \\ 22 & 4 & 13 & -3 \\ 54 & 13 & 156 & -22 \\ -13 & -3 & -22 & 4 \end{bmatrix}$$

$$x = \begin{pmatrix} z_i \\ \theta_i \\ z_j \\ \theta_j \end{pmatrix}$$

The global matrices can be constructed from this by adding each contribution of the individual elements to the global nodes it is connected to in the global matrices. The GUI, see Figure 4, is mostly kept the same only a few more parameters for the rotational support stiffness and damping and the disturbance forces can also be moments. Also a fictional rotational EQ was added. The tool still works on the same principles. The global matrices have become larger due to extra degree of freedom (DOF) per element. All plotted data shows only the vertical response.

©MECAL 2018 Confidential

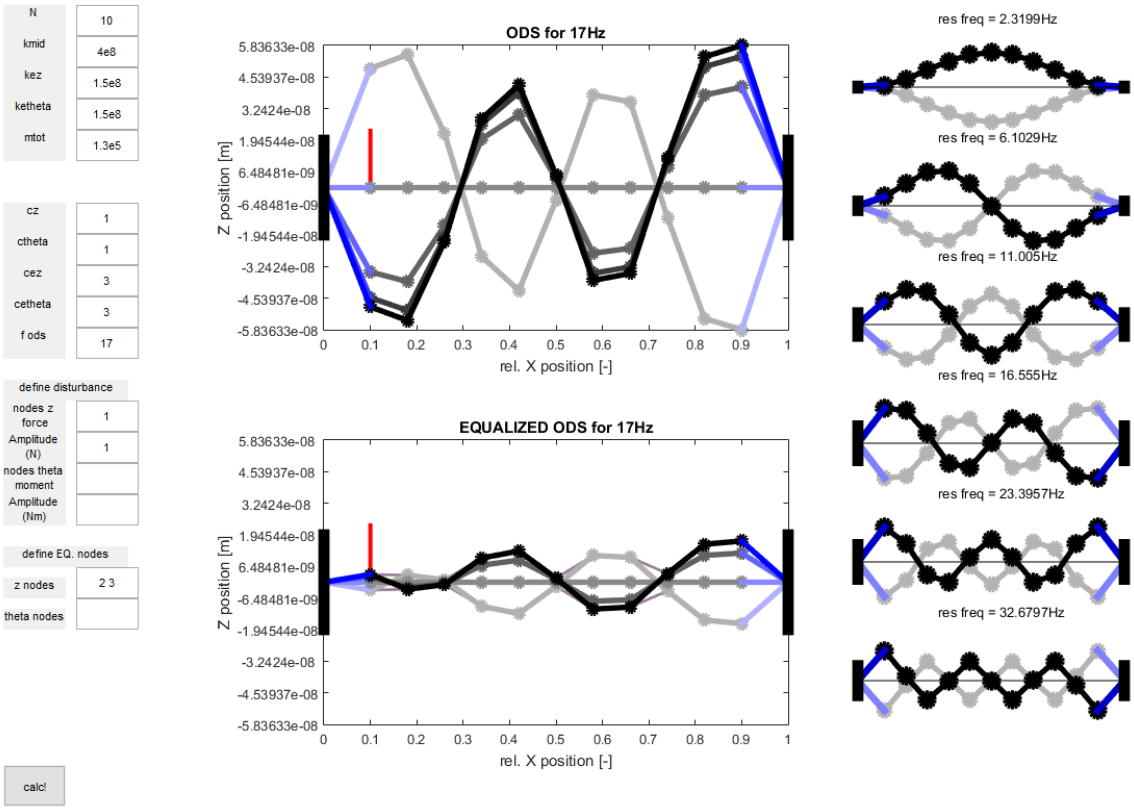


Figure 4: 2D floor GUI

It is assumed the floor can be approximated by a beam supporting in the way shown in Figure 5. For this case the factor EI is calculated from the stiffness in the middle by using the beam bending formula

$$v_{max} = \frac{-PL^3}{48EI} \rightarrow EI = \frac{-PL^3}{48v_{max}} = \frac{k_{mid}L^3}{48}$$

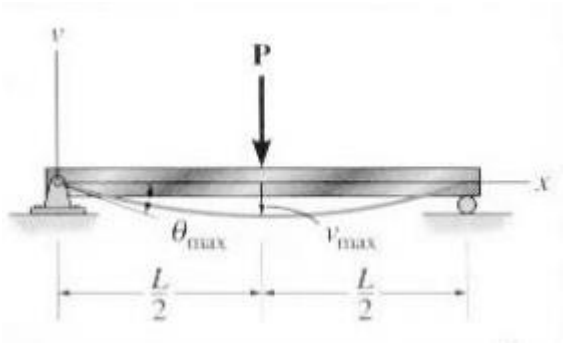


Figure 5: floor support assumption

Filling the found expression into the beam element stiffness matrix and using $L_e = L/N$ the leading factor of the stiffness matrix becomes $k_{mid}N^3/48$. The code for this tool is placed in Appendix C.
From this tool a few conclusion can be drawn. The location of the EQs is very important since damping it on one location thus no longer means damping along the chain. Damping along the chain can still be reached by using 2 translational EQs, however the placing and spacing between them depends on the frequency of

©MECAL 2018 Confidential

the disturbance force. Since both need to be placed outside a zero vertical movement point. The location of these points depends on the frequency of the disturbance force.

2.3 Second new version: 2D with floor parameters

The next version of the tool, see Figure 6, was altered by no longer using the stiffness in the middle of the floor but the user should give values for the parameters E (Youngs modulus), I (area moment of inertia) and L (floor length). No other adjustments were made. Although this tool does not give much more insight then the previous versions of the tool, it can be used for a real floor for a first estimate.

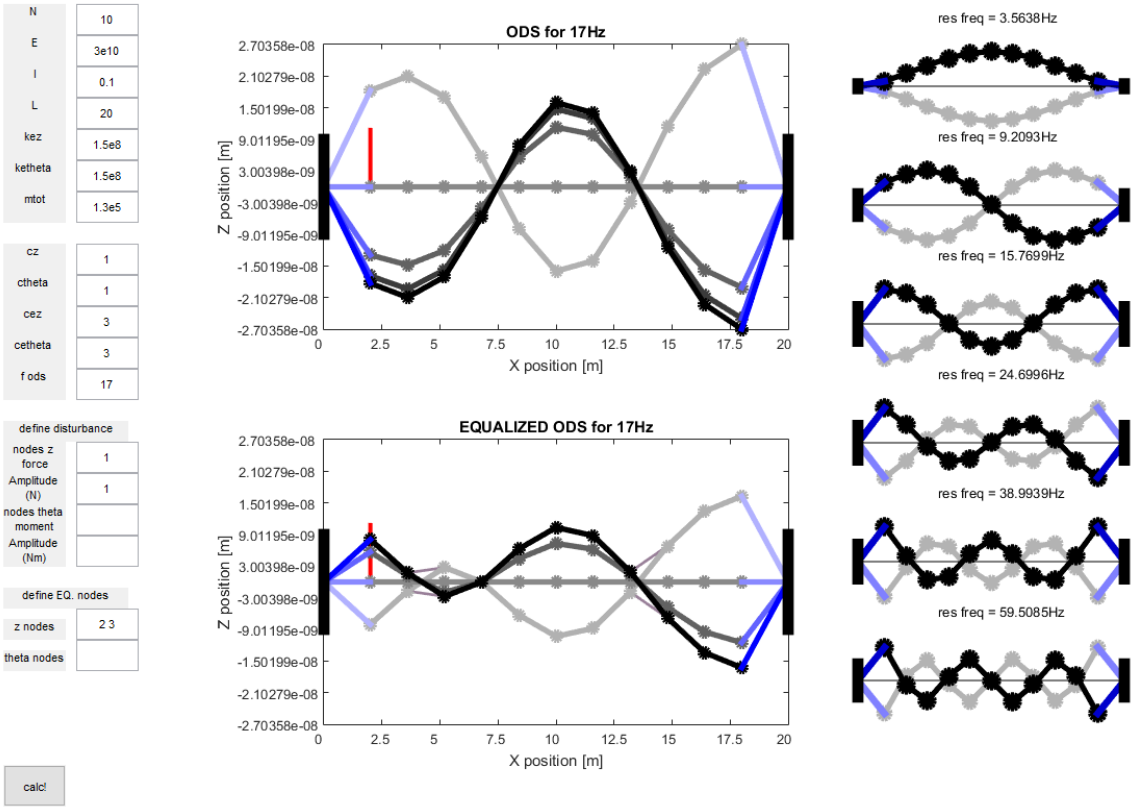


Figure 6: 2D floor GUI with floor parameters

2.4 Third new version: 3D with floor parameters

To get even better insight, the tool, see Figure 7, was altered by making it 3D by adding width. For this it is assumed the floor can be approximated by a grid of beams. The previous chosen 2D beam elements were made 3D by only allowing bending in plane. The GUI of the tool was altered by also showing the node and element grid on the right side. Also all plot had to be made 3D and the user needs to give a lot more parameters for both rotations. Again only the Z response is plotted. The code is placed in Appendix D.

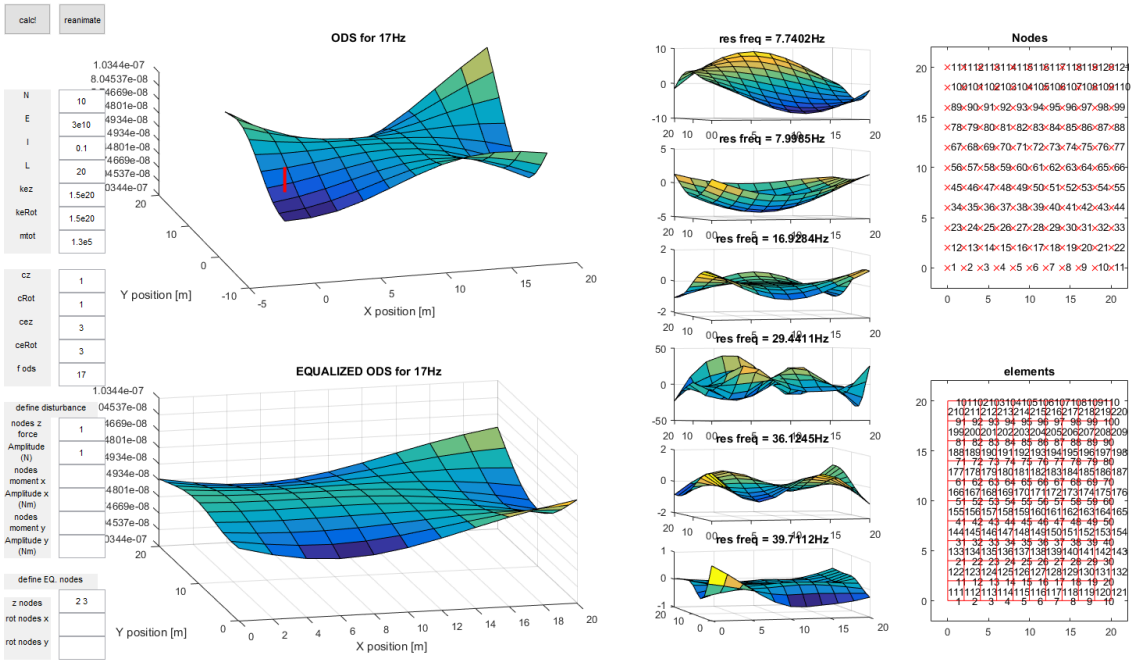


Figure 7: 3D floor GUI

2.5 Fourth new version: 3D with floor parameters and not square floors

The last adjustment to the tool, see Figure 8, was made by making it capable of handling not square floors. The code for this tool is placed in Appendix G.

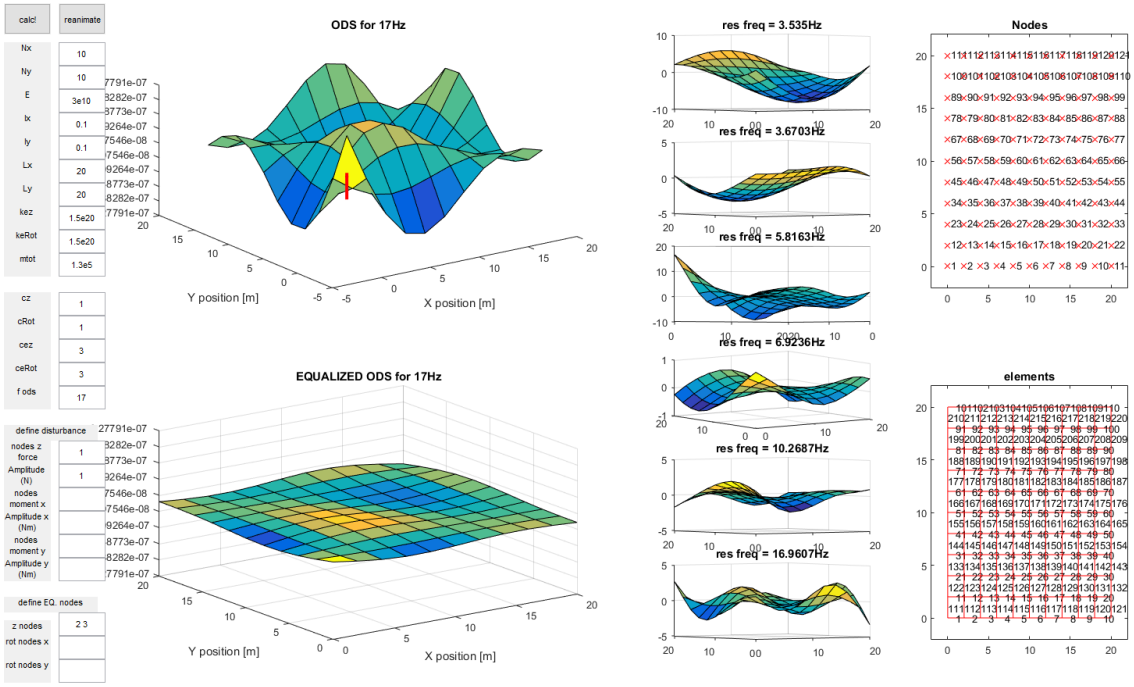


Figure 8: 3D floor GUI for not square floors

Chapter 3 Floor model estimation

As already stated in Chapter 1 previous interns had tried to get a floor model by using modal analysis. This method had problems if floors have a lot of damping and/or when the eigenfrequencies are close together. Therefore, either this method needs to be adjusted to be able to handle this, or another method needs to used. A new method of estimating mass, stiffness and damping matrices directly from the frequency response functions (frf) was derived. For this method to work, it could be necessary that a higher DOF system can be estimated by one of a lower. Since not all DOF are always measured. For instance of the floor it would be very handy if only a system of vertical forces and displacements can accurately describe the system since these are easily measured. However the forces will also result in rotations which are now ignored.

3.1 Higher DOF systems

To test if a higher DOF system can be accurately describe by a lower DOF system a theoretical system was used consisting of two 2D beams which can translate in vertical direction and rotate, see Figure 9. The beams are connected to each other and supported by vertical and rotational springs. For this system the global mass and stiffness matrices were derived and from this the frfs were constructed. Next, only the force to translation frfs where kept and used in a model estimation technique, that will be described in 3.2. From this, new frfs were constructed and compared to the original. The Matlab script can be found in Appendix I. From the results, see the Figure 1, it show that this works very well, for low frequencies the estimated system follows the original system very close. At higher frequencies it does not, due to missing eigenmodes. But this was to be expected since the system is of lower DOF then the original system. Also it can be noted that the matrices become less sparse, this is most likely due to missing coupling from the ignored rotational element.

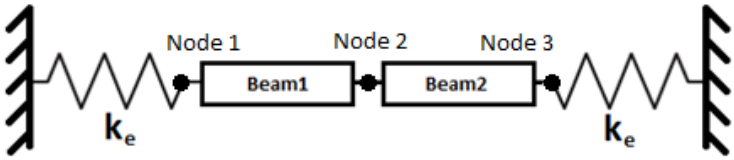


Figure 9: Dual beam element system

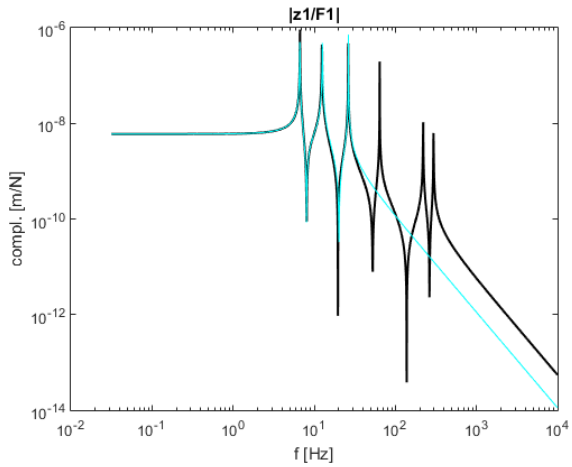


Figure 10: high to low DOF system

©MECAL 2018 Confidential

3.2 Matrices estimation

Theory

Another method than the modal analysis was though out to get a model. While the modal analysis has the advantage that only a single row or column of the frf matrix needs to be known this will not be the case with the following methods.
To get a model of the floor the stiffness and mass matrix were estimated by taking the transfer equation

$$frf(s) = (Ms^2 + Cs + K)^{-1}$$

And inverting it

$$frf(s)^{-1} = Ms^2 + Cs + K$$

At low frequencies (s≈0) the stiffness matrix is dominating the equation. Therefore it can be estimated by

$$K \approx |frf(f_l)|^{-1}$$

Where F_l is a low frequency. A similar trick can be done for estimating the mass matrix. The first equation is multiplied with s^2 resulting in

$$s^2 frf(s) = (M + \frac{C}{s} + \frac{K}{s^2})^{-1}$$

Inverting gives

$$(s^2 frf(s))^{-1} = M + \frac{C}{s} + \frac{K}{s^2}$$

At high frequencies (s≈i∞) the mass matrix will become dominant. Therefore the it can be estimated by

$$M \approx |-(2\pi f_h)^2 * frf(f_h)|^{-1}$$

Where f_h is at high frequency. This method works again well for a theoretical system as can be seen in the example but not in practice, as will be addressed in Chapter 5, due to missing high frequency data. Therefore this method is adjusted to work with low frequency data. This was done by still using the same method to get the stiffness matrix at f_l and using the result to estimate the damping matrix at a slightly higher frequency f_2 with

$$C = \frac{frf(f_2)^{-1} - K}{2\pi i f_2}$$

Example

The methods explained here are tested on a theoretical system. The Matlab code can be found in Appendix J. This theoretical system has the following matrices:

$$K = 10^8 \begin{bmatrix} 3.5 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 3.5 \end{bmatrix}$$

$$M = 10^4 \begin{bmatrix} 2.4143 & 0.8357 & 0 \\ 0.8357 & 4.8286 & 0.8357 \\ 0 & 0.8357 & 2.4143 \end{bmatrix}$$

$$C = 10^5 \begin{bmatrix} 6 & -3 & 0 \\ -3 & 6 & -3 \\ 0 & -3 & 6 \end{bmatrix}$$

For F_l=0.7463Hz the stiffness matrix is estimated at

$$K = 10^8 \begin{bmatrix} 3.4948 & -2.0019 & 0 \\ -2.0019 & 3.9895 & -2.0019 \\ 0 & -2.0019 & 3.4948 \end{bmatrix}$$

While the chosen frequency may seems like a low frequency to get accurate results on, one should keep in mind the chosen example system has a low first eigenfrequency of 7Hz thus this the stiffness matrix is estimated at approximant 1/10 of it and has proven to be accurate.
The mass matrix at f_h=118.5870Hz, which is approximant 4 times the highest eigenmode, is estimate to be

$$M = 10^4 \begin{bmatrix} 2.3550 & -0.8735 & 0 \\ -0.8735 & 4.7655 & -0.8735 \\ 0 & -0.8735 & 2.355 \end{bmatrix}$$

This is close in value but the cross terms do not have the correct sign. When not the absolute value is used but the real value instead this problem does not show and the matrix becomes

$$M = 10^4 \begin{bmatrix} 2.3552 & 0.8685 & 0.0012 \\ 0.8685 & 4.7609 & 0.8685 \\ 0.0012 & 0.8685 & 2.3552 \end{bmatrix}$$

For f_2=2.6498Hz the damping matrix becomes

$$C = 10^5 * \begin{bmatrix} 6 + 3.7078i & -3 + 1.2779i & 0 \\ -3 + 1.2779i & 6 + 7.4077i & -3 + 1.2779i \\ 0 & -3 + 1.2779i & 6 + 3.7078i \end{bmatrix}$$

©MECAL 2018 Confidential

From this the mass matrix is estimated at again a higher frequency f_3 with

$$M = \frac{\frac{frf(f_3)^{-1} - K}{2\pi i f_3} - C}{2\pi i f_3}$$

For a theoretical system this did gives acceptable results, however for a real system the choices for the frequencies to estimate the matrices influence the result. This method is adjusted once again by using a least square estimate (LSE) method to estimate the matrices, so the values for the matrices are estimated over a frequency range instead of at three points. This was done by using the equation

$$frf(s)^{-1} = Ms^2 + Cs + K$$

And rewriting the series of matrices frf(s)$^{-1}$ to the series of vector

$$Y(s) = [frf(s)_{11}^{-1}: frf(s)_{1n}^{-1}$$
$$frf(s)_{21}^{-1}: frf(s)_{2n}^{-1} \quad \ldots . frf(s)_{n1}^{-1}: frf(s)_{nn}^{-1}]$$

Next the series of vectors are made in a matrix by placing each vector for a s in a row. The matrix

$$\Phi = [-(2\pi f)^2 \quad 2\pi f i \quad \mathbf{1}]$$

Where **1** is a column vector of ones. Next using LSE give the values for matrices by

$$\theta = \Phi \backslash Y$$

Where $[M_{11}: M_{1n} \quad M_{21}: M_{2n} \quad \ldots \quad M_{n1}: M_{nn}] = \theta_{11}: \theta_{1n}$,
$[C_{11}: C_{1n} \quad C_{21}: C_{2n} \quad \ldots \quad C_{n1}: C_{nn}] = \theta_{21}: \theta_{2n}$
and $[K_{11}: K_{1n} \quad K_{21}: K_{2n} \quad \ldots \quad K_{n1}: K_{nn}] = \theta_{31}: \theta_{3n}$

While this method works very well on the theoretical system. On a real system it does not give useful results, see Chapter 5, since all the matrices had become complex and when looking at the eigenmodes showed no expected modes. However the idea of this method by first inverting the system of frfs to get the dynamic matrices is useful since the result can be used to add elements to the existing measured system. On this principle the Matlab based GUI for visualization and enhancement of floor dynamics is based, as will be explained in detail in Chapter 4.

The matrix has become complex the real part is exactly the original matrix thus this method works well in theory.
For f_3=17.7266Hz the mass matrix is estimated to be

$$M = 10^4 \begin{bmatrix} 2.0772 & 0.7195 & 0 \\ 0.7195 & 4.1550 & 0.7195 \\ 0 & 0.7195 & 2.0772 \end{bmatrix}$$

Which is acceptable but could be better.

The LSE method resulted in the matrices

$$M = 10^4 \begin{bmatrix} 2.4143 & 0.8357 & 0 \\ 0.8357 & 4.8286 & 0.8357 \\ 0 & 0.8357 & 2.4143 \end{bmatrix}$$
$$C = 10^5 \begin{bmatrix} 6 & -3 & 0 \\ -3 & 6 & -3 \\ 0 & -3 & 6 \end{bmatrix}$$
$$K = 10^8 \begin{bmatrix} 3.5 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 3.5 \end{bmatrix}$$

These matrices are the same as the starting system, therefor it can be concluded that this method works very well on a theoretical system.

©MECAL 2018 Confidential

Chapter 4 Matlab based GUI

The biggest contribution of this internship is the Matlab based GUI for visualization and enhancement of floor dynamics. This GUI is based on the principle of inverting a system of frfs to get the corresponding dynamic matrix. Next this matrix can be altered to simulate alterations of the system. The GUI should have the following features and/or options:

- Load measurement data from 3 sources:
  - Mat file
  - SpecTest measurements
  - Workspace
- Drawing of a map with measurement points and displaying added elements
- Allow user to draw and write on map
- Plot user chosen frfs for z, v and/or a with coherence
- Show user which frfs is the largest before and after altering the frfs over a chosen frequency range
- Run Operating Deflection Shapes (ODS) for a user chosen external force vector and frequency
- Allow user to add elements (point masses, point stiffness, frames and EQs)
- Adding of simple EQs
- Adding of tunable EQs the tuning should have the same interface as currently used interface
- Plot sensitivity
- Plot vibration forces
- Plot vibrations
- Plot EQs forces
- Save session

4.1 Loading of data

As stated the GUI should be able to load data from 3 sources. From these sources the Mat file and load from workspace are the simplest. Both rely on the user to supply name(s) of either the file or variables and the GUI can then load the data. The data should be in the correct format, the chosen formats can be found in Table 1 with N the number of measurement points, Nf the number of frequency points, Ny the number of measurements points in the y direction, Nx the number of measurements points in the x direction, Neq the number of EQs and Nfloor the number of floor drawings. The optional variables only work with a Mat file loading not with the loading from workspace. The frf and Coh should be in the form y*x*f, where y is the output, x the input points and f the frequency, all frf should be x/F. The measurement points should always be numbered by starting over the x axis and going up from there, an example is given in Figure 11. The variables X and Y can best be gotten from the Matlab function meshgrid. The GUI cannot work with an incomplete grid, but there is not need for the grid to be evenly spaced.

©MECAL 2018 Confidential

Variable name	Function	Data format
frf	Frf data	N*N*Nf complex double
Coh	Coherence data	N*N*Nf complex double
f	Frequency array	Nf double
X	X values of the measurement points	Ny*Nx double
Y	Y values of the measurement points	Ny*Nx double
VibF	Vibration force levels	N*Nf complex double
Optional		
EQ	EQ data	Neq struct; EQ.point double; EQ.transfer Nf complex double or EQ.stiffness double
floorVar	Floor drawings data	Nfloor struct; floorVar.x double array; floorVar.x double array; floorVar.text string
massPoints	Point mass locations	Char array with values "*loc1 loc2 … locm*"
mass	Point mass	Char array with values "*mass1 mass2 … massm*"
frameLoc1	Frame location 1	Char array with values "*loc11 loc12 … loc1f*"
frameLoc2	Frame location 2	Char array with values "*loc21 loc22 … loc2f*"
frameMass	Frame mass	Char array with values "*mass1 mass2 … massf*"
frameStiff	Frame stiffness	Char array with values "*stiff1 stiff2 … stifff*"
stiffPoint	Point stiffness location	Char array with values "*loc1 loc2 … lock*"
stiffPointValue	Point stiffness	Char array with values "*stiff1 stiff2 … stiffk*"

Table 1: Data format GUI

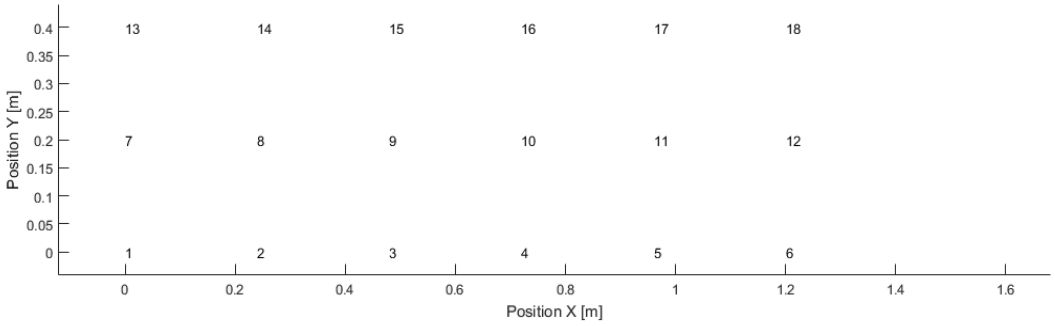


Figure 11: Measurement points example

The last option is loading from SpecTest measurements. SpecTest is a MECAL developed Matlab based GUI for doing measurements on floors, this GUI measure both frfs and vibration levels. SpecTest works with a velocity sensor(s) and a hammer with an accelerometer. For the vibration level measurement only the velocity sensors are used. For the frfs measurements an impact experiment is done to derive the transfer. It was chosen that the data from SpecTest can be loaded by placing the measurement data in a folder with two subfolders, one with the frf measurements and one with vibration level data. The folders need to be named 'Stiff' and 'Vib' respectively. The files in these folders need to have the correct name with the locations of the measurement points, since the written code uses the names to read the measurement points. The correct name is 'F(%s)_s(%s),(%s),(%s).mat' and 's(%s),(%s),(%s).mat' for the frf data and vibration level data respectively. From the SpecTest files the

variables frf, Coh and f are calculated. The variables X and Y are calculated with the function meshgrid and user given parameters of the floor. It assumed that the measurement points are evenly spaced. VibF is calculated from the frf and the vibration levels by use of

$$x_d = frf * F_d \rightarrow frf^{-1} * x_d = frf^{-1} * frf * F_d = F_d \rightarrow F_d = frf^{-1} * x_d$$

Where F_d is the disturbance forces and x_d the disturbance displacement or the vibrations level.

4.2 GUI functions

In this section the inner workings of some GUI functions is explained. Only the more complicated functions are elaborated. A complete user manual exist [5]. In this manual it is explained what the GUI can do and how it should be used.

4.2.1 Plotting frf for z, v and/or a with coherence

The frf(s) can be convered from displacement to velocity or acceleration by multiplying with $2\pi if$ or $-(2\pi f)^2$ respectively.

4.2.2 Run ODS

The ODS is calculated with $ODS(s) = frf(s) * F$. Where F is the external force vector. The result is a complex number. From this the displacement is calculated with $|ODS(s)|\cos(\angle ODS(s) + \varphi)$. The simulation is run by running φ from 0 till 2π and plotting the result.

4.2.3 Adding a point mass, frames or point stiffness

A point mass, frames or point stiffness can be added by inverting a system of frfs to get the dynamic matrices and adjusting it to the added elements and inverting the result back to get a new system of frfs as already stated in Chapter 3. This is the easiest for a point stiffness, the value of the stiffness is simply added to the collocated entry in the dynamic matrix for all frequencies. For the point was this is only slightly more complicated since the value that needs to be added to the collocated entry on the dynamic matrix is not the same for all frequencies. This value is easily calculated by:

$$D_{add}(f) = -m(2\pi f)^2$$

The frame is the most complicated since it adds both mass and stiffness. For the mass a simple lumped mass assumption is made and thus the mass is evenly split on both points from the frame. This results into two point masses which can be added. For the stiffness it is slight more complicated than the point stiffness. The stiffness is still added to both collocated entries but on the cross terms it is subtracted. This is the same as with the well know element stiffness matrix:

$$k_e = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix}$$

4.2.4 Add of EQs

As stated in the GUI function list at the beginning of this chapter two types of EQs need to be able to be added, namely a simple one and a tunable. The simple one assumes that adding an EQ on a point will increase the local stiffness tenfold. This is implemented by calculating the stiffness of that point with the all already added elements (point masses/stiffnesses, frames and other EQs) and adding ten times the real value as a point stiffness. For the tunable EQs the collocated transfer is loaded and

©MECAL 2018 Confidential

converted to velocity by multiplying with $2\pi if$. This is done since the sensor used by the EQs is a velocity sensor. Next the user can design a controller by adjusting gain and adding filters. The following filter types can be added: lag, lead, 1st order low pass, 2nd order low pass, 1st order high pass, notch and 2nd order filter. For most filter types five filters can be added with exception of the notch of which 20 can be added. While the user is designing the controller the open-loop transfer, sensitivity and controller are plotted and kept up to date. After the user is finished, he can save the controller. The controller settings are saved to a text file which can be used by the EQ software. This GUI uses the controller transfer to be added to the dynamic matrix after multiplication with $2\pi if$ to compensate for the fact the controller was designed for the velocity and not for the displacement. Just like it is done on real systems the EQs are placed and tuned sequential. Thus the order of placement is of importance.

4.2.5 Plot sensitivity

For a closed-loop system, see Figure 12, the sensitivity can be calculated with

$$S(s) = (\mathbf{1} + C(s)P(s))^{-1}$$

And the closed-loop transfer is

$$frf(s) = \left(\mathbf{1} + C(s)P(s)\right)^{-1}P(s)$$

Therefore

$$frf(s) = S(s)P(s) \rightarrow S(s) = frf(s)P^{-1}(s)$$

Or in words the sensitivity is the new transfer or transmissibility multiplied with the inverse of the old transfer or plant.

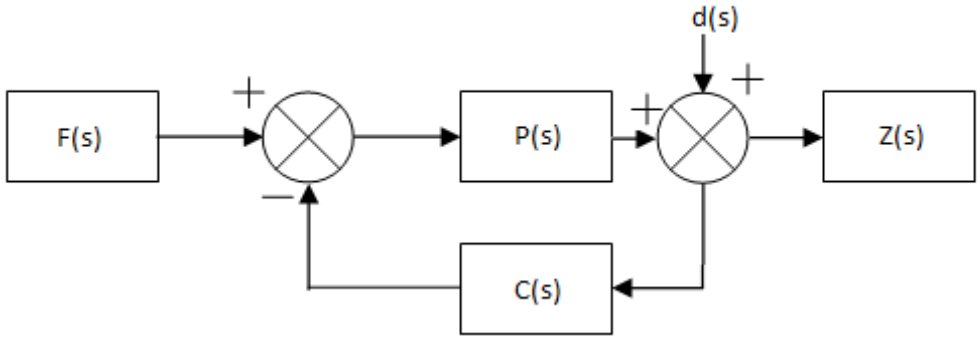


Figure 12: Closed-loop system, altered from [6]

4.2.6 Plot vibrations

The vibrations can easily be plotted since the disturbances are calculated to forces so the new vibration levels are simply

$$Z(s) = frf(s)F_d$$

4.2.7 Plot EQ forces

From Figure 12 it can be derived the EQ forces can be calculated with (first deriving the transfer from F(s) to the output of C(s) and then multiplying with the vibration forces)

$$F_{EQ} = \left(\mathbf{1} + C(s)P(s)\right)^{-1}C(s)P(s)F_d$$

©MECAL 2018 Confidential

4.2.8 Save session

When saving a session a Mat file is made with data as described in Table 1. However open plots are not saved.

4.3 Code

In Appendix K the code for the GUI can be found. Since there is a lot of code is split into several files. Appendix K has an index to keep track of these files. The code was tested with a test case described in Chapter 5.

©MECAL 2018 Confidential

Chapter 5 Test case

As a test case a table, see Figure 13, in the lab was used. On this table 18 locations where marked according the grid in Figure 14. For this setup all frfs and vibration levels where measured using SpecTest. From these measurements the matrices where tried to be derived using the methods from Chapter 3. The Matlab script for this can be found in Appendix L.

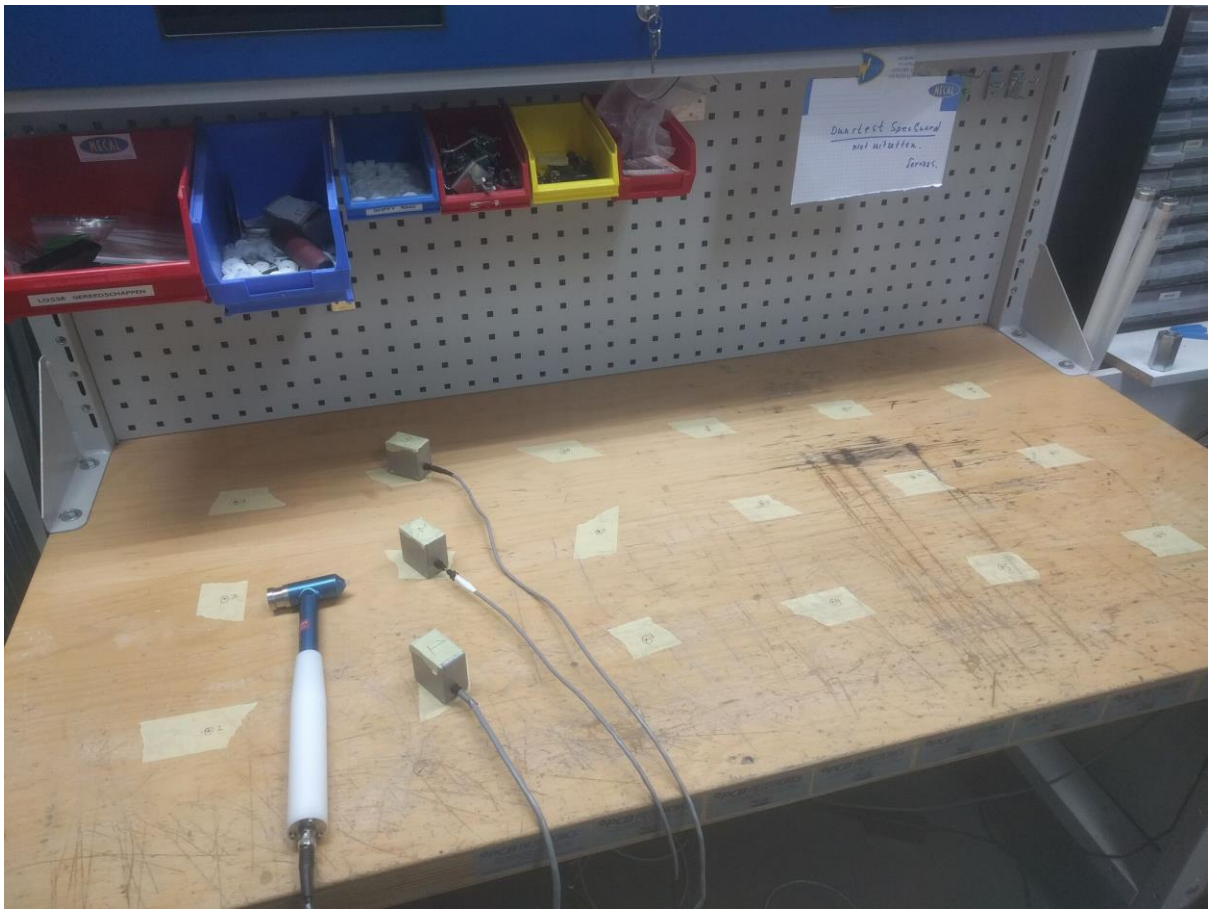


Figure 13: Test case table with SpecTest hammer and sensors

©MECAL 2018 Confidential

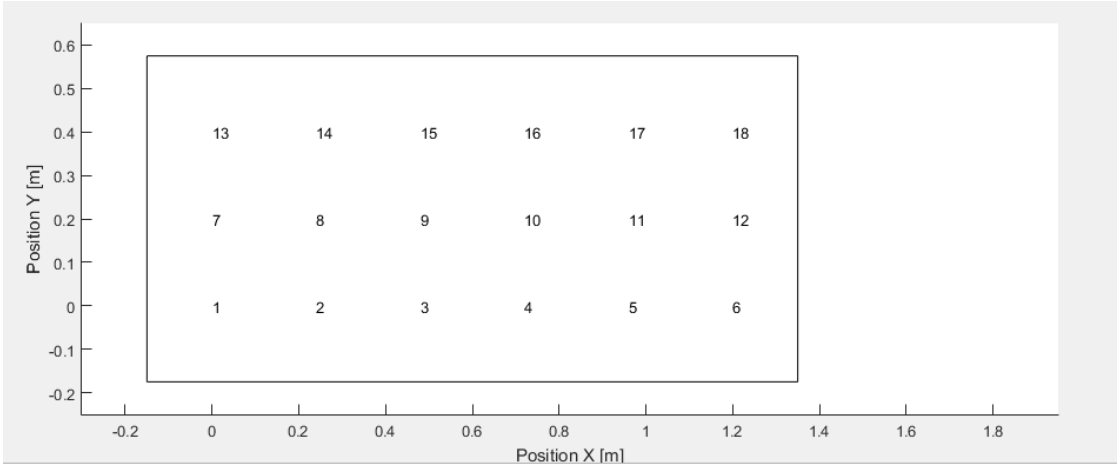


Figure 14: Table grid

In Figure 15 top left the co-located transfers are plotted from the measurements of the table. One can see a distinctive resonance peak at 58Hz. This peak should therefore be visible in all estimation methods or else it is not a good method.

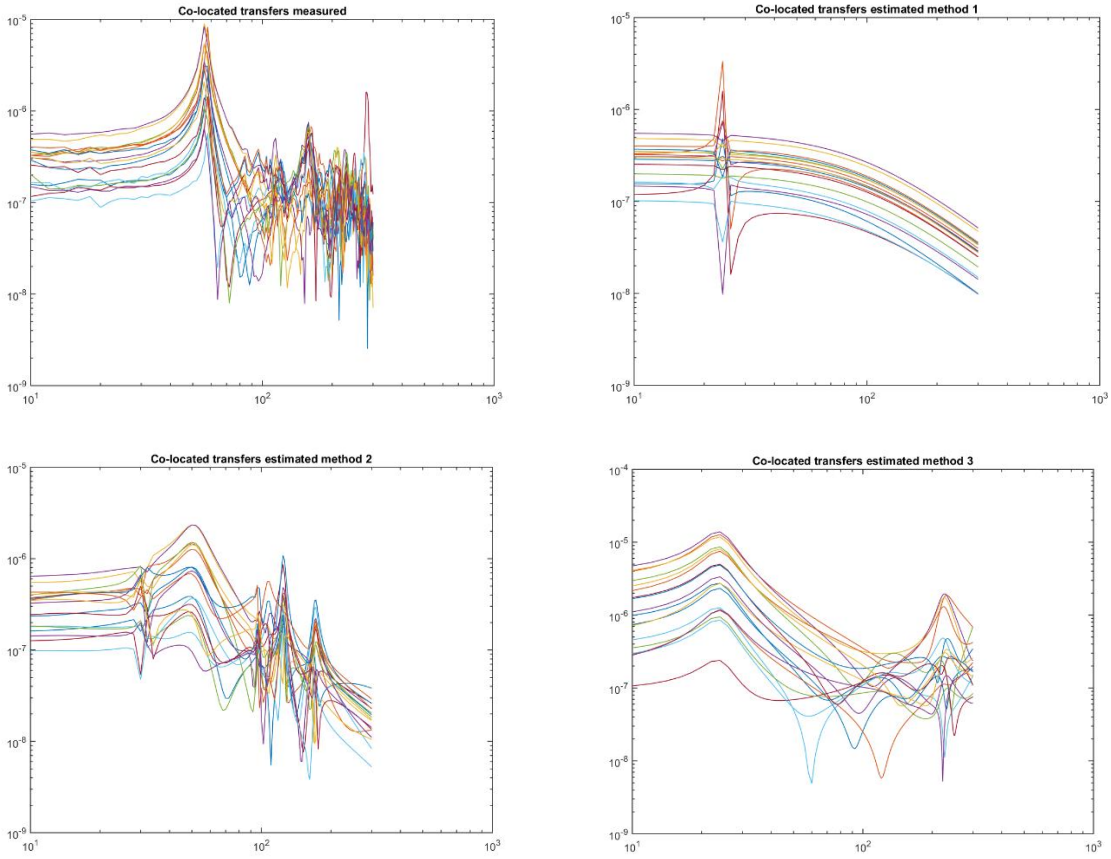


Figure 15: Table Co-located transfers

With the first estimation method from Chapter 3 only a stiffness and mass matrix where estimated, while the co-located transfers, see Figure 15 top right, show a very neat second order dynamic system

the resonance peak is at the wrong location. Therefore it can be concluded that this method does not work on a real system.

The second method works significantly better as can be seen from the co-located transfer in Figure 15 bottom left. However the peak is still not at the correct location and also there is a lot unexpected dynamic behaviour. Thus is concluded that this method does not lead to good results on a real system.

With final method, see Figure 15 bottom right**Error! Reference source not found.**, shows less u nexpected dynamic behaviour. But the peak is still not at the correct location so this method also not works on a real system.

This system is also used in the GUI from Chapter 4. From this the function were tested and it was concluded that the functions of the GUI behave like one would expect. This will not be addressed, since it is highly encouraged to try and play around with the GUI by yourself.

©MECAL 2018 Confidential

Chapter 6 Conclusions and recommendations

6.1.1 Conclusions

The GUI made works well, the user can easily see how the floor would react to an altered situation. Also the vibration levels can be obtained for the altered situation. Although the GUI is a great insight tool, the placement for the EQ is still difficult since it is still not automated and the results dependent on a lot of variables, e.g. disturbance source locations and local stiffness. When placing EQs one will want to place as few as possible to cut cost, but also one needs to get the vibration levels within specification.

6.1.2 Recommendations

The main recommendations for are

1. Run an experiment to test if the disturbance location estimation works
2. Retuning of tunable EQs
3. Automatization of EQ placement or a procedure
4. Loading of vibration levels specification
5. Adding of point damper and damper parameter to frame

6.1.3 Experiment

In the GUI the disturbance forces are estimated from the frfs and vibration levels with

$$F_d = frf^{-1} * x_d$$

It would be best to check if this idea also works on a real system. A simple way to do this is by making a system with a known disturbance source, for instance a shaker. From the measurements of this real system one should be able to reconstruct the disturbance force. A beginning was made with this experiment but it has not proven to be successful. This is most likely due to not syncing of the phase. Since there are more measurement points then sensors the vibration level measurement have to take place in multiple measurements. To sync the phase the multiple experiments should have a common sensor the measure the phase shift between measurements.

6.1.4 Retuning of tunable EQs

At the moment when a tunable EQ is placed it cannot be changed but it can be deleted. So if one wants to adjust a tunable EQ one has to delete it and rebuild it from scratch. Also one cannot see how the original EQ was constructed. Therefore it would be best if the original EQ could be loaded and adjusted.

6.1.5 Automatization of EQ placement or a procedure

As stated in 6.1.1 the placement of EQ is still done manual, it would be best if this could be automated or at least a good procedure would be drafted. A good starting point for this algorithm or procedure would be the location of large disturbance forces, also known as vibration entry points, and locations

©MECAL 2018 Confidential

where the floor vibration levels are above specification. Although the last points does not need to have to any vibration entry points but due to large transfers can become above specification.

6.1.6 Loading of vibration levels specification

At the moment it is up to the user if the vibration levels specification are met. Therefore it would be best to be able to load the specifications into the GUI. Also when this is added it would be best to also give the user cues when the specifications are met and which locations are out of bounds.

6.1.7 Adding of point damper and damper parameter to frame

In the GUI a point damper is not yet implemented while this could be done. Also the frame can also have a damper parameter implemented.

©MECAL 2018 Confidential

References

[1] S. Bank, Active Vibration Solutions, Eindhoven: Precisiebeurs, 2017.

[2] R. Giesen, “Modal analysis techniques for (damped) floors,” 2017.

[3] C. Felippa, “Introduction to Aerospace Structures,” 16 August 2017. [Online]. Available: https://www.colorado.edu/engineering/CAS/courses.d/Structures.d/Home.html. [Accessed 2 Juli 2018].

[4] J. Schilder, “Reader Dynamics 3 Flexible multibody dynamics,” p. 98.

[5] F. Koopman, Manual: Floor dynamics and vibration GUI V2, Enschede: MECAL, 2018.

[6] Skorkmaz, “wikimedia,” 9 December 2011. [Online]. Available: https://commons.wikimedia.org/wiki/File:Closed_Loop_Block_Deriv.png. [Accessed 9 July 2018].

[7] “MECAL,” 2015. [Online]. Available: mecal.eu. [Accessed 9 July 2018].

[8] “MECAL High-tech/Systems,” 2015. [Online]. Available: http://www.mecal.eu/high-tech#hightechsystemdevelopment. [Accessed 9 July 2018].

[9] Z. Giles, “MathWorks File Exchange circular_arrow,” 24 October 2016. [Online]. Available: https://nl.mathworks.com/matlabcentral/fileexchange/59917-circular-arrow. [Accessed 3 July 2018].

©MECAL 2018 Confidential

Appendix A MECAL

MECAL [7] is founded in 1989 and the name MECAL comes from (**ME**chanical **CAL**culation). MECAL focussed in the past on providing strength calculations and dynamic simulations performed on structures The plan was to build a company with at most a dozen employees working in the consultancy sector. The company was very focused on the competency and didn't have a specific market orientation, so they would take on any assignments or projects. But MECAL grew and became aware that it wasn't possible to be specialized in every field, so MECAL needed to specify. The conclusion was that MECAL strongly manifested with its knowledge in two segments, namely the Wind and Semiconductor Market.

MECAL organogram is displayed in Figure 16. This internship took place in Enschede under Hightech/ Systems [8].

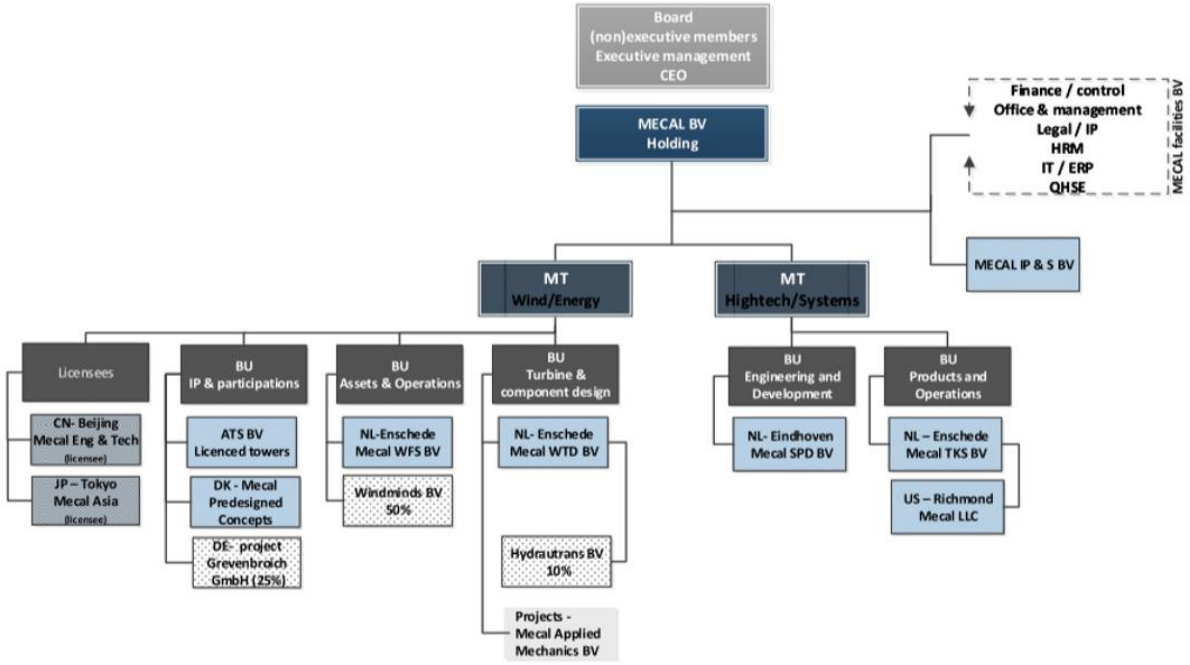


Figure 16: MECAL organogram

Appendix B Reflection

The time I spent at MECAL I have gained first-hand experience of the many aspects going in a project and the general process. From this I learned what to expect later in my working life and what I want in it. This is a valuable lesson that I learned during my 3 month internship.

Looking back at the beginning of my internship it was difficult to start since the assignment was not yet fixed. Since it was unclear at that moment which direction this internship would take. However starting from the work of previous intern I had several ideas to add my part in a better understanding and modelling the dynamics of floors.

From my supervisor I learned a lot about dynamics, being more distrustful to strange results, and the importance of good visualization.

During this internship I had the chance to put several things I learned during my study into practice. For instance the getting the equation of motion for a system, system identification and Matlab programming. Most of my time I spent in Matlab either programming or processing results, but I also spent some time in the lab doing experiments and doing research.

Although there was a positive working atmosphere, my project was highly individual. Therefore I missed working within a team. This would be something I do not want in my working life later.

I think I functioned well despite my personal problem of suffering from depression. I do not think that this had a vast impact on my internship as in it was mostly only noticeable when I was away for counselling.

Overall, I had a positive working experience at MECAL. I am grateful for everything I learned during this internship and I think having done this I gained experience that will benefit me for the rest of my life. Also having a internship adds a lot to my study as it gives a possibility to use skills learned in practice.

©MECAL 2018 Confidential

Appendix C Matlab floor insight GUI code 2D Version

The function below uses the function circular_arrow [9].

```
function twoD_floor_gui()
% twoD_floor_gui()
%
% UI function to calculate mode shapes and
% Operating Deflection Shapes(ODS)of a two-dimensional floor
% under force loading and with Equalizer
% The floor has only a length, not width, and can move only in z and
% theta. Forces only in z direction and stiffness in theta
%
% INPUT:
% for modal plots needed
%  N     : number of beam elements
%  kmid  : approx. stiffness mid floor of only floor = 4k/N [N/m]
%  kez   : edge stiffness between wall an first/last mass in z direction [N/m]
%  ketheta : edge stiffness between wall an first/last mass in theta direction [Nm/rad]
%  mtot  : total mass floor [kg]
%
% derived param's
%  Ke    : element stiffness matrix
%  m     : mass of each indiv. mass element = mtot/N [kg]
%  Me    : element mass matrix
%  K     : global stiffness matrix
%  M     : global mass matrix
%
% for ODS also needed
%  cez   : dampingconstant between edge and first element in z [-]
%  cetheta : dampingconstant between edge and first element in theta [-]
%  cz    : damping between mass elements [-] cz = sqrt(m*k)/q [N/(m/s)]
%  ctheta : damping between mass elements [-] ctheta = sqrt(m*k)/q [Nm/(rad/s)]
%  fods  : disturbance frequency [Hz]
%  Fdp   : array with applied disturbance forces nodes (for ODS only)
%  FdA   : array with applied disturbance forces amplitude (for ODS only)
%  Mdp   : array with applied disturbance moments nodes (for ODS only)
%  MdA   : array with applied disturbance moments amplitude (for ODS only)
%
% for EQUALIZED ODS also needed
%  EQ_Ptsz : z nodes which are equalized
%  EQ_Ptstheta : theta nodes which are equalized
%
% MODEL:
%
```


©MECAL 2018 Confidential

```
% || - - - ... - - - ||
%
% with || = wall, - = element
%
% Z = array with Z positions for mass 1..N
% only Z movement is allowed by the springs
%
%
% OUTPUT:
%  resonance frequencies
%  mode shapes visualized in a plot
%  ODS
%
% METHOD
%   (-1*lbd*M + K)*Z = F
% with lbd = w^2 and -1 = i^2
% eigen freq and mode shapes follow from the determinant
% of the dynamic matrix: D = K-lbd*M
% this can be computed with the eig(K,M) function in Octave/Matlab
%
% For the ODS the follwing formula holds:
%  (-w^2*M+iw*C +K)*Z = F
% with C being the damping matrix
% Z movements can be calculated by Z = (   )\F
%
% For the Equalized ODS:
% - the collocated percieved stiffness is calculated
%  compliance = K\F, stiffn = 1/compliance(i)
% - A stiffness (to earth) is added to the stiffness marix K
%  which is 10 times greater than the perceived stiffness at that point
%  and frequency.
%  this is the same as Sensitivity = 10
%  K(i,i) = K(i,i) + stiffn
% Mark that the order in which points are being equalized is not trivial
%
% ver 1 fkoo 3-7-2018

close all
global h;
% Define input parameters and UI
h.fig1 = figure('position',[10 10 1200 800],'name','TwoD floor GUI');
% button Calc
h.but= uicontrol('style','pushbutton','position', [10 10 60 40 ],'string','calc!');
set(h.but,'Callback',@CalcCallback);
% edit box N and text 'N'
h.t1 = uicontrol('style', 'text','position',[ 10 750,60,30],'string','N ');
h.edN = uicontrol('style','edit', 'position',[ 80,750,60,30],'string','10');
% edit box and text kmid
h.t2 = uicontrol('style', 'text','position',[ 10 720,60,30],'string','kmid');
h.edkmid = uicontrol('style','edit', 'position',[ 80,720,60,30],'string','4e8');
% edit box and text kez
h.t3 = uicontrol('style', 'text','position',[ 10 690,60,30],'string','kez ');
```


©MECAL 2018 Confidential

```
h.edkez = uicontrol('style','edit', 'position',[ 80,690,60,30],'string','1.5e8');
% edit box and text ketheta
h.t4 = uicontrol('style', 'text','position',[ 10 660,60,30],'string','ketheta ');
h.edketheta = uicontrol('style','edit', 'position',[ 80,660,60,30],'string','1.5e8');
% edit box and text m
h.t5 = uicontrol('style', 'text','position',[ 10 630,60,30],'string','mtot');
h.edm = uicontrol('style','edit', 'position',[ 80,630,60,30],'string','1.3e5');
% axes for modal plot
%h.ax1 = axes('units','pixels','position',[ 180,50, 1000,700]);
%get(h.ax1,'position')
h.t6 = uicontrol('style', 'text','position',[ 10 560,60,30],'string','cz ');
h.edqz = uicontrol('style','edit', 'position',[ 80,560,60,30],'string','1');
h.t7 = uicontrol('style', 'text','position',[ 10 530,60,30],'string','ctheta ');
h.edqtheta = uicontrol('style','edit', 'position',[ 80,530,60,30],'string','1');
h.t8 = uicontrol('style', 'text','position',[ 10 500,60,30],'string','cez ');
h.edqez = uicontrol('style','edit', 'position',[ 80,500,60,30],'string','3');
h.t9 = uicontrol('style', 'text','position',[ 10 470,60,30],'string','cetheta ');
h.edqetheta = uicontrol('style','edit', 'position',[ 80,470,60,30],'string','3');
h.t10 = uicontrol('style', 'text','position',[ 10 440,60,30],'string','f ods ');
h.edfods = uicontrol('style','edit', 'position',[ 80,440,60,30],'string','17');
% disturbance forces
h.t11 = uicontrol('style', 'text','position',[ 10 400,120,20],'string','define disturbance');
h.t12 = uicontrol('style', 'text','position',[ 10 370,60,30],'string','nodes z force');
h.edFdp = uicontrol('style','edit', 'position',[ 80,370,60,30],'string','1');
h.t13 = uicontrol('style', 'text','position',[ 10 340,60,30],'string','Amplitude (N) ');
h.edFdA = uicontrol('style','edit', 'position',[ 80,340,60,30],'string','1');
h.t14 = uicontrol('style', 'text','position',[ 10 310,60,30],'string','nodes theta moment');
h.edMdp = uicontrol('style','edit', 'position',[ 80,310,60,30],'string','');
h.t15 = uicontrol('style', 'text','position',[ 10 280,60,30],'string','Amplitude (Nm)');
h.edMdA = uicontrol('style','edit', 'position',[ 80,280,60,30],'string','');
% EQUALIZER points
h.t16 = uicontrol('style', 'text','position',[ 10 240,120,20],'string','define EQ. nodes');
h.t17 = uicontrol('style', 'text','position',[ 10 210,60,20],'string','z nodes');
h.edPtsz = uicontrol('style','edit', 'position',[ 80,210,60,30],'string','2 3');
h.t18 = uicontrol('style', 'text','position',[ 10 180,60,20],'string','theta nodes');
h.edPtstheta = uicontrol('style','edit', 'position',[ 80,180,60,30],'string','');

% calculation is done in the function CalcCallback

end % function OneD_floor_gui



function CalcCallback(~,~)
global h;       % handle to figure


% input for modeshapes
N = str2num(get(h.edN,'string'));
kmid = str2num(get(h.edkmid,'string'));
kez = str2num(get(h.edkez,'string'));
ketheta = str2num(get(h.edketheta,'string'));
```

```
m = str2num(get(h.edm,'string'));
me = m/N;

% Element matricies
Ke=kmid*N^3/48*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=me/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];

% input for ODS
cz = str2num(get(h.edqz,'string'));
ctheta = str2num(get(h.edqtheta,'string'));
cez = str2num(get(h.edqez,'string'));
cetheta = str2num(get(h.edqetheta,'string'));
fods = str2num(get(h.edfods,'string'));

% Distrubance forces
Fd = zeros(2*N+2,1);
edFdp = str2num(get(h.edFdp,'string'));
edFdA = str2num(get(h.edFdA,'string'));
Fd(2*edFdp-1)=edFdA;

% Distrubance moments
edMdp = str2num(get(h.edMdp,'string'));
edMdA = str2num(get(h.edMdA,'string'));
Fd(2*edMdp)=edMdA;

% check dat Fd niet per ongeluk te groot wordt
if length(Fd) > 2*N+2
  Fd = Fd(1:2*N);
end

% EQ points
edPtsz = str2num(get(h.edPtsz,'string'));
edPtstheta = str2num(get(h.edPtstheta,'string'));

EQ_Pts = sort([2*edPtsz-1 2*edPtstheta]);
dx = 1/(N);
x = linspace(dx,1-dx,N+1);
xleft = [0 dx];
xright = [1-dx 1];

% FILL in the modal masses M and K
M= zeros(2*N+2,2*N+2);
K=M;
for i = 1:N
  M(2*i-1:2*i+2,2*i-1:2*i+2) = M(2*i-1:2*i+2,2*i-1:2*i+2) + Me;
  K(2*i-1:2*i+2,2*i-1:2*i+2) = K(2*i-1:2*i+2,2*i-1:2*i+2) + Ke;
end

K(1,1) = K(1,1)+kez;
K(2,2) = K(2,2)+ketheta;
K(end-1,end-1) = K(end-1,end-1)+kez;
K(end,end) = K(end,end)+ketheta;
```


©MECAL 2018 Confidential

```
% compute eigen freq and mode shapes with magical eig function
[v,lbd] = eig(K,M);
fres=zeros(1,length(lbd));
for i = 1:length(lbd)
  fres(i) = sqrt(lbd(i,i))/(2*pi);
end

% plot first 6 (if available) modes on separate figure
nplt = min(6,length(lbd));

for i = 1:nplt
  subplot(nplt,3,3+3*(i-1));
  hold off;
  a = v(:,i)/v(1,i);
  % eerst andere kant op in grijs
  plot(x,-a(1:2:end-1),'-o','color',
[0.7,0.7,0.7],'linewidth',4,'markersize',8,'markerfacecolor',[0.7,0.7,0.7]);
  hold on; % the masses with k's
  plot(xleft,[0 -a(1)],'color',[.5,0.5,1],'linewidth',5);  % left end
  plot(xright,[-a(end-1),0],'color',[.5,0.5,1],'linewidth',5); % right end
  % dan ene kant op
  plot(x,a(1:2:end-1),'-ok','linewidth',4,'markersize',10,'markerfacecolor','k');
  hold on; % the masses with k's
  plot(xleft,[0 a(1)],'color',[0,0,0.8],'linewidth',5);  % left end
  plot(xright,[a(end-1),0],'color',[0,0,0.8],'linewidth',5); % right end
  plot ([0,0],[-.5 .5],'k','linewidth',8); % wall
  plot ([1,1],[-.5 .5],'k','linewidth',8); % wall
  plot([0 1],[0 0],'k');
  xlim([0 1]);ylim([-1.2*max(abs(a)) 1.2*max(abs(a))]);
  text(0.3,2*max(abs(a)),['res freq = ' num2str(fres(i)) 'Hz']);
  set(gca,'visible','off');
end

% ODS
% Fill in the C matrix

% elements C matrix
Ce=zeros(4);
Ce(1:2:3,1:2:3)=[cz -cz;-cz cz];
Ce(2:2:4,2:2:4)=[ctheta -ctheta;-ctheta ctheta];

C=zeros(2*N+2,2*N+2);
for i = 1:N
  C(2*i-1:2*i+2,2*i-1:2*i+2) = C(2*i-1:2*i+2,2*i-1:2*i+2) + Ce;
end

C(1,1) = C(1,1)+cez;
C(2,2) = C(2,2)+cetheta;
C(end-1,end-1) = C(end-1,end-1)+cez;
C(end,end) = C(end,end)+cetheta;
```

```
% Now calculate the deflection shape
w = 2*pi*fods;
D = -w^2*M + 1i*w*C+K;
Z = D\Fd;

% Plot the ODS
% because not all masses move either the same way or the opposite way
% we need to take into account the phase.
% We do this by plotting different points in time ( = different values for the phase)
hods(1)= subplot(2,6,[2,3,4]);
hold off;
scalingy = 0.5/max(abs(Z(1:2:end-1)));
scalingF = 0.5/max(Fd);
plot([0 1],[0 0],'k'); hold on; % 0 -line
for i = 1:N+1
  % plot applied forces
  if Fd(2*i-1) ~= 0
    plot([x(i) x(i)],[0 0.4*scalingF*Fd(2*i-1)],'r','linewidth',3);
  end % if

  % plot applied moments
  if Fd(2*i) ~= 0
    circular_arrow(gcf,0.1*scalingF*Fd(2*i),[x(i) 0],0,170,-1,'r');
  end % if

end % for
% boundary
% plot(x,abs(Z),'.-','color',[0.6 0.5 0.6],'linewidth',2);
% plot(x,-abs(Z),'.-','color',[0.6 0.5 0.6],'linewidth',2);

Nl = 5; % number of lines in the plot 36 for each 10 degrees
phi = [-1.5*pi,-pi,-pi/2,-pi/4,-pi/8];
for i = 1: Nl
  plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi(i)+angle(Z(1:2:end-1))),'-o','color', [1-i/(1.3*Nl),1-
i/(1.3*Nl),1-i/(1.3*Nl)],'linewidth',4);
  hold on;
  plot([0 dx],scalingy*[0 abs(Z(1)).*cos(phi(i)+angle(Z(1)))],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);
  plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(phi(i)+angle(Z(end-1))) 0],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);hold on;
end
%plot(x,abs(Z).*cos(angle(Z)),'-o','color', [0 0 0 ],'linewidth',4);
%plot([1-dx 1],[abs(Z(N)).*cos(angle(Z(N))) 0],'color',[0,0,1],'linewidth',4);
%plot([0 dx],[0 abs(Z(1)).*cos(angle(Z(1)))],'color',[0,0, 1],'linewidth',4);

% test with animation, plot only first moment, phi = 0
phi = 0;
h1= plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi+angle(Z(1:2:end-1))),'-ok', 'linewidth',4);
h2 = plot([0 x(1)],scalingy*[0 abs(Z(1))*cos(phi+angle(Z(1)))],'-b','linewidth',4);
h3 = plot([x(end) 1],scalingy*[ abs(Z(end-1))*cos(phi+angle(Z(end-1))) 0],'-b','linewidth',4);
Zods = Z;
```


©MECAL 2018 Confidential

```
ylim([-0.5,0.5]);
plot ([0,0],[-0.5/3 0.5/3],'k','linewidth',8); % wall
plot ([1,1],[-0.5/3 0.5/3],'k','linewidth',8); % wall

title(['ODS for ' num2str(fods) 'Hz']);
ylabel('Z position [m]'); xlabel('rel. X position [-]');
set(gca,'YTick',linspace(-0.5,0.5,10));
set(gca,'YTickLabel',linspace(-1/scalingy,1/scalingy,10));
Zorig= Z; % keep track of original movement
% Now calc forces needed for EQ
for i =1: length(EQ_Pts)
  pt= EQ_Pts(i); % point to equalize
  %zpt = Z(pt); % movement of that point
  Fhlp = zeros(2*N+2,1); Fhlp(pt) =1; % F for collocated stiffness
  Zcol = D\Fhlp; % collocated compl. z/F for point pt for f=fods;
  keq = 10/Zcol(pt); % 10 times complex stiffness %10/abs(Zcol(pt)); % equivalent stiffness of
active eq
  K(pt,pt) = K(pt,pt) +keq;
end
% calculate new system ( with changed K matrix)
D = -w^2*M + 1i*w*C+K;
Z = D\Fd; % calc new resulting movement

% now plot
hods(2) = subplot(2,6,[8,9,10]);
linkaxes(hods,'y');
hold off;
plot([0 1],[0 0],'k'); hold on;% 0 -line
for i = 1:N+1
  % plot applied forces
  if Fd(2*i-1) ~= 0
    plot([x(i) x(i)],[0 0.4*scalingF*Fd(2*i-1)],'r','linewidth',3);
  end % if

  % plot applied moments
  if Fd(2*i) ~= 0
    circular_arrow(gcf,0.1*scalingF*Fd(2*i),[x(i) 0],0,170,-1,'r');
  end % if

end % for
scalingy = 0.5/max(abs([Z(1:2:end-1);Zorig(1:2:end-1)]));
% boundary
plot(x,scalingy*abs(Z(1:2:end-1)),'.-','color',[0.6 0.5 0.6],'linewidth',2);
plot(x,-scalingy*abs(Z(1:2:end-1)),'.-','color',[0.6 0.5 0.6],'linewidth',2);
Nl = 5; % number of lines in the plot 36 for each 10 degrees
phi = [-1.5*pi,-pi,-pi/2,-pi/4,-pi/8];
for i = 1: Nl
  h5= plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi(i)+angle(Z(1:2:end-1))),'-o','color', [1-
i/(1.3*Nl),1-i/(1.3*Nl),1-i/(1.3*Nl)],'linewidth',4);
  hold on;
  h6 =plot([0 dx],scalingy*[0 abs(Z(1)).*cos(phi(i)+angle(Z(1)))],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);
```


©MECAL 2018 Confidential

```
  h7 =plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(phi(i)+angle(Z(end-1))) 0],'color',[1-
i/(1.3*Nl),1-i/(1.3*Nl),1],'linewidth',4);hold on;
end
plot(x,scalingy*abs(Z(1:2:end-1)).*cos(angle(Z(1:2:end-1))),'-o','color', [0 0 0 ],'linewidth',4);
plot([0 dx],scalingy*[0 abs(Z(1)).*cos(angle(Z(1)))],'color',[0,0, 1],'linewidth',4);
plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(angle(Z(end-1))) 0],'color',[0,0,1],'linewidth',4);

ylim([-0.5,0.5]);
plot ([0,0],[-0.5/3 0.5/3],'k','linewidth',8); % wall
plot ([1,1],[-0.5/3 0.5/3],'k','linewidth',8); % wall

title(['EQUALIZED ODS for ' num2str(fods) 'Hz']);
ylabel('Z position [m]'); xlabel('rel. X position [-]');
set(gca,'YTick',linspace(-0.5,0.5,10));
set(gca,'YTickLabel',linspace(-1/scalingy,1/scalingy,10));

% Now animate!
for i = 1: 80
  phi = i*2*pi/40;
  pause(0.15);
  set(h1,'ydata',scalingy*abs(Zods(1:2:end-1)).*cos(phi+angle(Zods(1:2:end-1))));
  set(h2 ,'ydata',scalingy*[ 0 abs(Zods(1)).*cos(phi+angle(Zods(1)))]);
  set(h3 ,'ydata',scalingy*[ abs(Zods(end-1)).*cos(phi+angle(Zods(end-1))) 0]);

  set(h5,'ydata', scalingy*abs(Z(1:2:end-1)).*cos(phi+angle(Z(1:2:end-1))));
  set(h6, 'ydata',scalingy*[0 abs(Z(1)).*cos(phi+angle(Z(1)))]);
  set(h7, 'ydata',scalingy*[abs(Z(end-1)).*cos(phi+angle(Z(end-1))) 0]);
end

end % function calcCallback()
```

Appendix D Matlab floor insight GUI code 2D Version with floor parameters

The function below uses the function circular_arrow [9].

```
function twoD_floor_gui_floorParameters()
% twoD_floor_gui()
%
% UI function to calculate mode shapes and
% Operating Deflection Shapes(ODS)of a two-dimensional floor
% under force loading and with Equalizer
% The floor has only a length, not width, and can move only in z and
% theta. Forces only in z direction and stiffness in theta
%
% INPUT:
% for modal plots needed
%  N    : number of beam elements
%  E    : yield modulus [Pa]
%  I    : second moment of area [m^4]
%  L    : floor length [m]
%  kez   : edge stiffness between wall an first/last mass in z direction [N/m]
%  ketheta : edge stiffness between wall an first/last mass in theta direction [Nm/rad]
%  mtot  : total mass floor [kg]
%
% derived param's
%  Ke   : element stiffness matrix
%  m    : mass of each indiv. mass element = mtot/N [kg]
%  Me   : element mass matrix
%  K    : global stiffness matrix
%  M    : global mass matrix
%
% for ODS also needed
%  cez   : dampingconstant between edge and first element in z [-]
%  cetheta : dampingconstant between edge and first element in theta [-]
%  cz   : damping between mass elements [-] cz = sqrt(m*k)/q [N/(m/s)]
%  ctheta : damping between mass elements [-] ctheta = sqrt(m*k)/q [Nm/(rad/s)]
%  fods  : disturbance frequency [Hz]
%  Fdp   : array with applied disturbance forces nodes (for ODS only)
%  FdA   : array with applied disturbance forces amplitude (for ODS only)
%  Mdp   : array with applied disturbance moments nodes (for ODS only)
%  MdA   : array with applied disturbance moments amplitude (for ODS only)
%
% for EQUALIZED ODS also needed
%  EQ_Ptsz : z nodes which are equalized
%  EQ_Ptstheta : theta nodes which are equalized
%
% MODEL:
```


©MECAL 2018 Confidential

```
%
% || - - - ... - - - ||
%
% with || = wall, - = element
%
% Z = array with Z positions for mass 1..N
% only Z movement is allowed by the springs
%
%
% OUTPUT:
%  resonance frequencies
%  mode shapes visualized in a plot
%  ODS
%
% METHOD
%   (-1*lbd*M + K)*Z = F
% with lbd = w^2 and -1 = i^2
% eigen freq and mode shapes follow from the determinant
% of the dynamic matrix: D = K-lbd*M
% this can be computed with the eig(K,M) function in Octave/Matlab
%
% For the ODS the follwing formula holds:
%  (-w^2*M+iw*C +K)*Z = F
% with C being the damping matrix
% Z movements can be calculated by Z = (   )\F
%
% For the Equalized ODS:
% - the collocated percieved stiffness is calculated
%  compliance = K\F, stiffn = 1/compliance(i)
% - A stiffness (to earth) is added to the stiffness marix K
%  which is 10 times greater than the perceived stiffness at that point
%  and frequency.
%  this is the same as Sensitivity = 10
%  K(i,i) = K(i,i) + stiffn
% Mark that the order in which points are being equalized is not trivial
%
% ver 1 fkoo 3-7-2018

close all
global h;
% Define input parameters and UI
h.fig1 = figure('position',[10 10 1200 800],'name','TwoD floor GUI with parameters');
% button Calc
h.but= uicontrol('style','pushbutton','position', [10 10 60 40 ],'string','calc!');
set(h.but,'Callback',@CalcCallback);
% edit box N and text 'N'
h.t1 = uicontrol('style', 'text','position',[ 10 750,60,30],'string','N ');
h.edN = uicontrol('style','edit', 'position',[ 80,750,60,30],'string','10');
% edit box and text E
h.t2 = uicontrol('style', 'text','position',[ 10 720,60,30],'string','E');
h.E = uicontrol('style','edit', 'position',[ 80,720,60,30],'string','3e10');
% edit box and text I
```


©MECAL 2018 Confidential

```
h.t3 = uicontrol('style', 'text','position',[ 10 690,60,30],'string','I');
h.I = uicontrol('style','edit', 'position',[ 80,690,60,30],'string','0.1');
% edit box and text L
h.t4 = uicontrol('style', 'text','position',[ 10 660,60,30],'string','L');
h.L = uicontrol('style','edit', 'position',[ 80,660,60,30],'string','20');
% edit box and text kez
h.t5 = uicontrol('style', 'text','position',[ 10 630,60,30],'string','kez ');
h.edkez = uicontrol('style','edit', 'position',[ 80,630,60,30],'string','1.5e8');
% edit box and text ketheta
h.t6 = uicontrol('style', 'text','position',[ 10 600,60,30],'string','ketheta ');
h.edketheta = uicontrol('style','edit', 'position',[ 80,600,60,30],'string','1.5e8');
% edit box and text m
h.t7 = uicontrol('style', 'text','position',[ 10 570,60,30],'string','mtot');
h.edm = uicontrol('style','edit', 'position',[ 80,570,60,30],'string','1.3e5');
% axes for modal plot
%h.ax1 = axes('units','pixels','position',[ 180,50, 1000,700]);
%get(h.ax1,'position')
h.t8 = uicontrol('style', 'text','position',[ 10 520,60,30],'string','cz ');
h.edqz = uicontrol('style','edit', 'position',[ 80,520,60,30],'string','1');
h.t9 = uicontrol('style', 'text','position',[ 10 490,60,30],'string','ctheta ');
h.edqtheta = uicontrol('style','edit', 'position',[ 80,490,60,30],'string','1');
h.t10 = uicontrol('style', 'text','position',[ 10 460,60,30],'string','cez ');
h.edqez = uicontrol('style','edit', 'position',[ 80,460,60,30],'string','3');
h.t11 = uicontrol('style', 'text','position',[ 10 430,60,30],'string','cetheta ');
h.edqetheta = uicontrol('style','edit', 'position',[ 80,430,60,30],'string','3');
h.t12 = uicontrol('style', 'text','position',[ 10 400,60,30],'string','f ods ');
h.edfods = uicontrol('style','edit', 'position',[ 80,400,60,30],'string','17');
% disturbance forces
h.t13 = uicontrol('style', 'text','position',[ 10 360,120,20],'string','define disturbance');
h.t14 = uicontrol('style', 'text','position',[ 10 330,60,30],'string','nodes z force');
h.edFdp = uicontrol('style','edit', 'position',[ 80,330,60,30],'string','1');
h.t15 = uicontrol('style', 'text','position',[ 10 300,60,30],'string','Amplitude (N) ');
h.edFdA = uicontrol('style','edit', 'position',[ 80,300,60,30],'string','1');
h.t16 = uicontrol('style', 'text','position',[ 10 270,60,30],'string','nodes theta moment');
h.edMdp = uicontrol('style','edit', 'position',[ 80,270,60,30],'string','');
h.t17 = uicontrol('style', 'text','position',[ 10 240,60,30],'string','Amplitude (Nm)');
h.edMdA = uicontrol('style','edit', 'position',[ 80,240,60,30],'string','');
% EQUALIZER points
h.t18 = uicontrol('style', 'text','position',[ 10 200,120,20],'string','define EQ. nodes');
h.t18 = uicontrol('style', 'text','position',[ 10 170,60,20],'string','z nodes');
h.edPtsz = uicontrol('style','edit', 'position',[ 80,170,60,30],'string','2 3');
h.t19 = uicontrol('style', 'text','position',[ 10 140,60,20],'string','theta nodes');
h.edPtstheta = uicontrol('style','edit', 'position',[ 80,140,60,30],'string','');

% calculation is done in the function CalcCallback

end % function OneD_floor_gui



function CalcCallback(~,~)
global h;       % handle to figure
```


©MECAL 2018 Confidential

```
% input for modeshapes
N = str2num(get(h.edN,'string'));
E = str2num(get(h.E,'string'));
I = str2num(get(h.I,'string'));
L = str2num(get(h.L,'string'));

kez = str2num(get(h.edkez,'string'));
ketheta = str2num(get(h.edketheta,'string'));
m = str2num(get(h.edm,'string'));
me = m/N;
le=L/N;

% Element matricies
Ke=E*I/le^3*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=me/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];

% input for ODS
cz = str2num(get(h.edqz,'string'));
ctheta = str2num(get(h.edqtheta,'string'));
cez = str2num(get(h.edqez,'string'));
cetheta = str2num(get(h.edqetheta,'string'));
fods = str2num(get(h.edfods,'string'));

% Distrubance forces
Fd = zeros(2*N+2,1);
edFdp = str2num(get(h.edFdp,'string'));
edFdA = str2num(get(h.edFdA,'string'));
Fd(2*edFdp-1)=edFdA;

% Distrubance moments
edMdp = str2num(get(h.edMdp,'string'));
edMdA = str2num(get(h.edMdA,'string'));
Fd(2*edMdp)=edMdA;

% check dat Fd niet per ongeluk te groot wordt
if length(Fd) > 2*N+2
  Fd = Fd(1:2*N+2);
end

% EQ points
edPtsz = str2num(get(h.edPtsz,'string'));
edPtstheta = str2num(get(h.edPtstheta,'string'));

EQ_Pts = sort([2*edPtsz-1 2*edPtstheta]);
dx = 1/(N);
x = linspace(dx,1-dx,N+1);
xleft = [0 dx];
xright = [1-dx 1];

% FILL in the modal masses M and K
```


©MECAL 2018 Confidential

```
M= zeros(2*N+2,2*N+2);
K=M;
for i = 1:N
  M(2*i-1:2*i+2,2*i-1:2*i+2) = M(2*i-1:2*i+2,2*i-1:2*i+2) + Me;
  K(2*i-1:2*i+2,2*i-1:2*i+2) = K(2*i-1:2*i+2,2*i-1:2*i+2) + Ke;
end

K(1,1) = K(1,1)+kez;
K(2,2) = K(2,2)+ketheta;
K(end-1,end-1) = K(end-1,end-1)+kez;
K(end,end) = K(end,end)+ketheta;

% compute eigen freq and mode shapes with magical eig function
[v,lbd] = eig(K,M);
fres=zeros(1,length(lbd));
for i = 1:length(lbd)
  fres(i) = sqrt(lbd(i,i))/(2*pi);
end

% plot first 6 (if available) modes on separate figure
nplt = min(6,length(lbd));

for i = 1:nplt
  subplot(nplt,3,3+3*(i-1));
  hold off;
  a = v(:,i)/v(1,i);
  % eerst andere kant op in grijs
  plot(x,-a(1:2:end-1),'-o','color',
[0.7,0.7,0.7],'linewidth',4,'markersize',8,'markerfacecolor',[0.7,0.7,0.7]);
  hold on; % the masses with k's
  plot(xleft,[0 -a(1)],'color',[.5,0.5,1],'linewidth',5);  % left end
  plot(xright,[-a(end-1),0],'color',[.5,0.5,1],'linewidth',5); % right end
  % dan ene kant op
  plot(x,a(1:2:end-1),'-ok','linewidth',4,'markersize',10,'markerfacecolor','k');
  hold on; % the masses with k's
  plot(xleft,[0 a(1)],'color',[0,0,0.8],'linewidth',5);  % left end
  plot(xright,[a(end-1),0],'color',[0,0,0.8],'linewidth',5); % right end
  plot ([0,0],[-.5 .5],'k','linewidth',8); % wall
  plot ([1,1],[-.5 .5],'k','linewidth',8); % wall
  plot([0 1],[0 0],'k');
  xlim([0 1]);ylim([-1.2*max(abs(a)) 1.2*max(abs(a))]);
  text(0.3,2*max(abs(a)),['res freq = ' num2str(fres(i)) 'Hz']);
  set(gca,'visible','off');
end

% ODS
% Fill in the C matrix

% elements C matrix
Ce=zeros(4);
Ce(1:2:3,1:2:3)=[cz -cz;-cz cz];
Ce(2:2:4,2:2:4)=[ctheta -ctheta;-ctheta ctheta];
```


©MECAL 2018 Confidential

```
C=zeros(2*N+2,2*N+2);
for i = 1:N
  C(2*i-1:2*i+2,2*i-1:2*i+2) = C(2*i-1:2*i+2,2*i-1:2*i+2) + Ce;
end

C(1,1) = C(1,1)+cez;
C(2,2) = C(2,2)+cetheta;
C(end-1,end-1) = C(end-1,end-1)+cez;
C(end,end) = C(end,end)+cetheta;

% Now calculate the deflection shape
w = 2*pi*fods;
D = -w^2*M + 1i*w*C+K;
Z = D\Fd;

% Plot the ODS
% because not all masses move either the same way or the opposite way
% we need to take into account the phase.
% We do this by plotting different points in time ( = different values for the phase)
hods(1)= subplot(2,6,[2,3,4]);
hold off;
scalingy = 0.5/max(abs(Z(1:2:end-1)));
scalingF = 0.5/max(Fd);
plot([0 1],[0 0],'k'); hold on; % 0 -line
for i = 1:N+1
  % plot applied forces
  if Fd(2*i-1) ~= 0
    plot([x(i) x(i)],[0 0.4*scalingF*Fd(2*i-1)],'r','linewidth',3);
  end % if

  % plot applied moments
  if Fd(2*i) ~= 0
    circular_arrow(gcf,0.1*scalingF*Fd(2*i),[x(i) 0],0,170,-1,'r');
  end % if

end % for
% boundary
% plot(x,abs(Z),'.-','color',[0.6 0.5 0.6],'linewidth',2);
% plot(x,-abs(Z),'.-','color',[0.6 0.5 0.6],'linewidth',2);

Nl = 5; % number of lines in the plot 36 for each 10 degrees
phi = [-1.5*pi,-pi,-pi/2,-pi/4,-pi/8];
for i = 1: Nl
  plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi(i)+angle(Z(1:2:end-1))),'-o','color', [1-i/(1.3*Nl),1-
i/(1.3*Nl),1-i/(1.3*Nl)],'linewidth',4);
  hold on;
  plot([0 dx],scalingy*[0 abs(Z(1)).*cos(phi(i)+angle(Z(1)))],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);
  plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(phi(i)+angle(Z(end-1))) 0],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);hold on;
end
```


©MECAL 2018 Confidential

```
%plot(x,abs(Z).*cos(angle(Z)),'-o','color', [0 0 0 ],'linewidth',4);
%plot([1-dx 1],[abs(Z(N)).*cos(angle(Z(N))) 0],'color',[0,0,1],'linewidth',4);
%plot([0 dx],[0 abs(Z(1)).*cos(angle(Z(1)))],'color',[0,0, 1],'linewidth',4);

% test with animation, plot only first moment, phi = 0
phi = 0;
h1= plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi+angle(Z(1:2:end-1))),'-ok', 'linewidth',4);
h2 = plot([0 x(1)],scalingy*[0 abs(Z(1))*cos(phi+angle(Z(1)))],'-b','linewidth',4);
h3 = plot([x(end) 1],scalingy*[ abs(Z(end-1))*cos(phi+angle(Z(end-1))) 0],'-b','linewidth',4);
Zods = Z;

ylim([-0.5,0.5]);
plot ([0,0],[-0.5/3 0.5/3],'k','linewidth',8); % wall
plot ([1,1],[-0.5/3 0.5/3],'k','linewidth',8); % wall

title(['ODS for ' num2str(fods) 'Hz']);
ylabel('Z position [m]'); xlabel('X position [m]');
set(gca,'YTick',linspace(-0.5,0.5,10));
set(gca,'YTickLabel',linspace(-1/scalingy,1/scalingy,10));

set(gca,'XTick',linspace(0,1,9));
set(gca,'XTickLabel',linspace(0,L,9));
Zorig= Z; % keep track of original movement
% Now calc forces needed for EQ
for i =1: length(EQ_Pts)
  pt= EQ_Pts(i); % point to equalize
  %zpt = Z(pt); % movement of that point
  Fhlp = zeros(2*N+2,1); Fhlp(pt) =1; % F for collocated stiffness
  Zcol = D\Fhlp; % collocated compl. z/F for point pt for f=fods;
  keq = 10/Zcol(pt); % 10 times complex stiffness %10/abs(Zcol(pt)); % equivalent stiffness of
active eq
  K(pt,pt) = K(pt,pt) +keq;
end
% calculate new system ( with changed K matrix)
D = -w^2*M + 1i*w*C+K;
Z = D\Fd; % calc new resulting movement

% now plot
hods(2) = subplot(2,6,[8,9,10]);
linkaxes(hods,'y');
hold off;
plot([0 1],[0 0],'k'); hold on;% 0 -line
for i = 1:N+1
  % plot applied forces
  if Fd(2*i-1) ~= 0
    plot([x(i) x(i)],[0 0.4*scalingF*Fd(2*i-1)],'r','linewidth',3);
  end % if

  % plot applied moments
  if Fd(2*i) ~= 0
    circular_arrow(gcf,0.1*scalingF*Fd(2*i),[x(i) 0],0,170,-1,'r');
  end % if
```

```
end % for
scalingy = 0.5/max(abs([Z(1:2:end-1);Zorig(1:2:end-1)]));
% boundary
plot(x,scalingy*abs(Z(1:2:end-1)),'.-','color',[0.6 0.5 0.6],'linewidth',2);
plot(x,-scalingy*abs(Z(1:2:end-1)),'.-','color',[0.6 0.5 0.6],'linewidth',2);
Nl = 5; % number of lines in the plot 36 for each 10 degrees
phi = [-1.5*pi,-pi,-pi/2,-pi/4,-pi/8];
for i = 1: Nl
  h5= plot(x,scalingy*abs(Z(1:2:end-1)).*cos(phi(i)+angle(Z(1:2:end-1))),'-o','color', [1-
i/(1.3*Nl),1-i/(1.3*Nl),1-i/(1.3*Nl)],'linewidth',4);
  hold on;
  h6 =plot([0 dx],scalingy*[0 abs(Z(1)).*cos(phi(i)+angle(Z(1)))],'color',[1-i/(1.3*Nl),1-
i/(1.3*Nl),1],'linewidth',4);
  h7 =plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(phi(i)+angle(Z(end-1))) 0],'color',[1-
i/(1.3*Nl),1-i/(1.3*Nl),1],'linewidth',4);hold on;
end
plot(x,scalingy*abs(Z(1:2:end-1)).*cos(angle(Z(1:2:end-1))),'-o','color', [0 0 0 ],'linewidth',4);
plot([0 dx],scalingy*[0 abs(Z(1)).*cos(angle(Z(1)))],'color',[0,0, 1],'linewidth',4);
plot([1-dx 1],scalingy*[abs(Z(end-1)).*cos(angle(Z(end-1))) 0],'color',[0,0,1],'linewidth',4);

ylim([-0.5,0.5]);
plot ([0,0],[-0.5/3 0.5/3],'k','linewidth',8); % wall
plot ([1,1],[-0.5/3 0.5/3],'k','linewidth',8); % wall

title(['EQUALIZED ODS for ' num2str(fods) 'Hz']);
ylabel('Z position [m]'); xlabel('X position [m]');
set(gca,'YTick',linspace(-0.5,0.5,10));
set(gca,'YTickLabel',linspace(-1/scalingy,1/scalingy,10));
set(gca,'XTick',linspace(0,1,9));
set(gca,'XTickLabel',linspace(0,L,9));

% Now animate!
for i = 1: 80
  phi = i*2*pi/40;
  pause(0.15);
  set(h1,'ydata',scalingy*abs(Zods(1:2:end-1)).*cos(phi+angle(Zods(1:2:end-1))));
  set(h2 ,'ydata',scalingy*[ 0 abs(Zods(1)).*cos(phi+angle(Zods(1)))]);
  set(h3 ,'ydata',scalingy*[ abs(Zods(end-1)).*cos(phi+angle(Zods(end-1))) 0]);

  set(h5,'ydata', scalingy*abs(Z(1:2:end-1)).*cos(phi+angle(Z(1:2:end-1))));
  set(h6, 'ydata',scalingy*[0 abs(Z(1)).*cos(phi+angle(Z(1)))]);
  set(h7, 'ydata',scalingy*[abs(Z(end-1)).*cos(phi+angle(Z(end-1))) 0]);
end

end % function calcCallback()
```

Appendix E Matlab floor insight GUI code 3D Version with floor parameters

The function below uses the function globalMatrixBuilder, see Appendix F.

```
function threeD_floor_gui_floorParameters()
% threeD_floor_gui_floorParameters()
%
% UI function to calculate mode shapes and
% Operating Deflection Shapes(ODS)of a 3DOF floor
% under force loading and with Equalizer
% The floor has length and can translate only in z and
% rotate around x and y. Forces only in z direction and moments around x
% and y
%
% INPUT:
% for modal plots needed
%  N    : number of beam elements per dimension
%  E    : yield modulus [Pa]
%  I    : second moment of area vector [m^4]
%  L    : floor length [m]
%  kez   : edge stiffness between wall an first/last mass in z direction [N/m]
%  keRot  : edge stiffness vector [rotX,rotY] between wall an first/last mass in theta direction
[Nm/rad]
%  mtot  : total mass floor [kg]
%
% derived param's
%  Ke   : element stiffness matrix
%  m    : mass of each indiv. mass element = mtot/N [kg]
%  Me   : element mass matrix
%  K    : global stiffness matrix
%  M    : global mass matrix
%
% for ODS also needed
%  qez   : dampingconstant between edge and first element in z [N/(m/s)]
%  qeRot  : dampingconstant vector [rotX,rotY] between edge and first element in theta [Nm/(rad/s)]
%  qz   : damping between mass elements [N/(m/s)]
%  qRot  : damping vector [rotX,rotY] between mass elements [Nm/(rad/s)]
%  fods  : disturbance frequency [Hz]
%  Fdp   : array with applied disturbance forces nodes (for ODS only)
%  FdA   : array with applied disturbance forces amplitude (for ODS only)
%  Mxdp  : array with applied disturbance moments around x nodes (for ODS only)
%  MxdA  : array with applied disturbance moments around x amplitude (for ODS only)
%  Mydp  : array with applied disturbance moments around y nodes (for ODS only)
%  MydA  : array with applied disturbance moments around y amplitude (for ODS only)
%
% for EQUALIZED ODS also needed
```

```
%  EQ_Ptsz : z nodes which are equalized
%  EQ_PtsrotX : rotational nodes around x which are equalized
%  EQ_PtsrotY : rotational nodes around x which are equalized
%
% MODEL:
%
% --------- ... --------
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% ......................
% ......................
% ......................
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% --------- ... --------
%
% with || or - = wall, ~ or l = elements and , = nodes
%
% Z = array with Z translation and rotations around x and y for all nodes
%
%
% OUTPUT:
%  resonance frequencies
%  mode shapes visualized in a plot
%  ODS
%
% METHOD
%   (-1*lbd*M + K)*Z = F
% with lbd = w^2 and -1 = i^2
% eigen freq and mode shapes follow from the determinant
% of the dynamic matrix: D = K-lbd*M
% this can be computed with the eig(K,M) function in Octave/Matlab
%
% For the ODS the follwing formula holds:
%  (-w^2*M+iw*C +K)*Z = F
% with C being the damping matrix
% Z movements can be calculated by Z = (   )\F
%
% For the Equalized ODS:
% - the collocated percieved stiffness is calculated
%  compliance = K\F, stiffn = 1/compliance(i)
% - A stiffness (to earth) is added to the stiffness marix K
%  which is 10 times greater than the perceived stiffness at that point
%  and frequency.
%  this is the same as Sensitivity = 10
%  K(i,i) = K(i,i) + stiffn
% Mark that the order in which points are being equalized is not trivial
%
```


©MECAL 2018 Confidential

```
% ver 1 fkoo 3-7-2018

close all

% Open figure on secondairy screen if aviable
secondScreenInfo=get(0,'MonitorPositions');
offset=0;
if size(secondScreenInfo,1)>1
  offset=secondScreenInfo(2,1);
end

global h;
% Define input parameters and UI
h.fig1 = figure('position',[80+offset 80 1600 900],'name','TwoD floor GUI with parameters');
% button Calc
h.but1= uicontrol('style','pushbutton','position', [10 850 60 40 ],'string','calc!');
set(h.but1,'Callback',@CalcCallback);
% button reanimate
h.but2= uicontrol('style','pushbutton','position', [80 850 60 40 ],'string','reanimate');
set(h.but2,'Callback',@animate);
set(h.but2,'enable','off');
% edit box N and text 'N'
h.t1 = uicontrol('style', 'text','position',[ 10 750,60,30],'string','N ');
h.edN = uicontrol('style','edit', 'position',[ 80,750,60,30],'string','10');
% edit box and text E
h.t2 = uicontrol('style', 'text','position',[ 10 720,60,30],'string','E');
h.E = uicontrol('style','edit', 'position',[ 80,720,60,30],'string','3e10');
% edit box and text I
h.t3 = uicontrol('style', 'text','position',[ 10 690,60,30],'string','I');
h.I = uicontrol('style','edit', 'position',[ 80,690,60,30],'string','0.1');
% edit box and text L
h.t4 = uicontrol('style', 'text','position',[ 10 660,60,30],'string','L');
h.L = uicontrol('style','edit', 'position',[ 80,660,60,30],'string','20');
% edit box and text kez
h.t5 = uicontrol('style', 'text','position',[ 10 630,60,30],'string','kez ');
h.edkez = uicontrol('style','edit', 'position',[ 80,630,60,30],'string','1.5e20');
% edit box and text ketheta
h.t6 = uicontrol('style', 'text','position',[ 10 600,60,30],'string','keRot ');
h.edkeRot = uicontrol('style','edit', 'position',[ 80,600,60,30],'string','1.5e20');
% edit box and text m
h.t7 = uicontrol('style', 'text','position',[ 10 570,60,30],'string','mtot');
h.edm = uicontrol('style','edit', 'position',[ 80,570,60,30],'string','1.3e5');
% axes for modal plot
%h.ax1 = axes('units','pixels','position',[ 180,50, 1000,700]);
%get(h.ax1,'position')
h.t8 = uicontrol('style', 'text','position',[ 10 520,60,30],'string','cz ');
h.edqz = uicontrol('style','edit', 'position',[ 80,520,60,30],'string','1');
h.t9 = uicontrol('style', 'text','position',[ 10 490,60,30],'string','cRot ');
h.edqRot = uicontrol('style','edit', 'position',[ 80,490,60,30],'string','1');
h.t10 = uicontrol('style', 'text','position',[ 10 460,60,30],'string','cez ');
h.edqez = uicontrol('style','edit', 'position',[ 80,460,60,30],'string','3');
h.t11 = uicontrol('style', 'text','position',[ 10 430,60,30],'string','ceRot ');
```


©MECAL 2018 Confidential

```
h.edqeRot = uicontrol('style','edit', 'position',[ 80,430,60,30],'string','3');
h.t12 = uicontrol('style', 'text','position',[ 10 400,60,30],'string','f ods ');
h.edfods = uicontrol('style','edit', 'position',[ 80,400,60,30],'string','17');
% disturbance forces
h.t13 = uicontrol('style', 'text','position',[ 10 360,120,20],'string','define disturbance');
h.t14 = uicontrol('style', 'text','position',[ 10 330,60,30],'string','nodes z force');
h.edFdp = uicontrol('style','edit', 'position',[ 80,330,60,30],'string','1');
h.t15 = uicontrol('style', 'text','position',[ 10 300,60,30],'string','Amplitude (N) ');
h.edFdA = uicontrol('style','edit', 'position',[ 80,300,60,30],'string','1');
h.t16 = uicontrol('style', 'text','position',[ 10 270,60,30],'string','nodes moment x');
h.edMxdp = uicontrol('style','edit', 'position',[ 80,270,60,30],'string','');
h.t17 = uicontrol('style', 'text','position',[ 10 240,60,30],'string','Amplitude x (Nm)');
h.edMxdA = uicontrol('style','edit', 'position',[ 80,240,60,30],'string','');
h.t18 = uicontrol('style', 'text','position',[ 10 210,60,30],'string','nodes moment y');
h.edMydp = uicontrol('style','edit', 'position',[ 80,210,60,30],'string','');
h.t19 = uicontrol('style', 'text','position',[ 10 180,60,30],'string','Amplitude y (Nm)');
h.edMydA = uicontrol('style','edit', 'position',[ 80,180,60,30],'string','');
% EQUALIZER points
h.t20 = uicontrol('style', 'text','position',[ 10 140,120,20],'string','define EQ. nodes');
h.t21 = uicontrol('style', 'text','position',[ 10 110,60,20],'string','z nodes');
h.edPtsz = uicontrol('style','edit', 'position',[ 80,110,60,30],'string','2 3');
h.t22 = uicontrol('style', 'text','position',[ 10 80,60,30],'string','rot nodes x');
h.edPtsRotx = uicontrol('style','edit', 'position',[ 80,80,60,30],'string','');
h.t23 = uicontrol('style', 'text','position',[ 10 50,60,30],'string','rot nodes y');
h.edPtsRoty = uicontrol('style','edit', 'position',[ 80,50,60,30],'string','');
% calculation is done in the function CalcCallback

end % function OneD_floor_gui



function CalcCallback(~,~)
global h N;       % handle to figure
set(h.but2,'enable','off');
set(h.but1,'enable','off');

% input for modeshapes
N = str2num(get(h.edN,'string'));
E = str2num(get(h.E,'string'));
I = str2num(get(h.I,'string'));
L = str2num(get(h.L,'string'));

kez = str2num(get(h.edkez,'string'));
keRot = str2num(get(h.edkeRot,'string'));
m = str2num(get(h.edm,'string'));
me = m/N^2;
le=L/N;

% input for ODS
cz = str2num(get(h.edqz,'string'));
cRot = str2num(get(h.edqRot,'string'));
cez = str2num(get(h.edqez,'string'));
```


©MECAL 2018 Confidential

```
ceRot = str2num(get(h.edqeRot,'string'));
fods = str2num(get(h.edfods,'string'));

% Element matricies
Ke=E*I/le^3*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=me/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];
Ce=[cz 0 -cz 0 ; 0 cRot 0 -cRot ; -cz 0 cz 0 ; 0 -cRot 0 cRot];

Kex=zeros(6);Key=Kex;Mex=Kex;Mey=Kex;Cex=Kex;Cey=Kex;
Kex([1 3:4 6],[1 3:4 6])=Ke;
Key([1:2 4:5],[1:2 4:5])=Ke;
Mex([1 3:4 6],[1 3:4 6])=Me;
Mey([1:2 4:5],[1:2 4:5])=Me;
Cex([1 3:4 6],[1 3:4 6])=Ce;
Cey([1:2 4:5],[1:2 4:5])=Ce;

% Distrubance forces
Fd = zeros(3*(N+1)^2,1);
edFdp = str2num(get(h.edFdp,'string'));
edFdA = str2num(get(h.edFdA,'string'));
Fd(3*edFdp-2)=edFdA;

% Distrubance moments
edMdp = str2num(get(h.edMxdp,'string'));
edMdA = str2num(get(h.edMxdA,'string'));
Fd(3*edMdp-1)=edMdA;

edMdp = str2num(get(h.edMydp,'string'));
edMdA = str2num(get(h.edMydA,'string')); %#ok<*ST2NM>
Fd(3*edMdp)=edMdA;

% check dat Fd niet per ongeluk te groot wordt
if length(Fd) > 3*(N+1)^2
  Fd = Fd(1:3*(N+1)^2);
end

% EQ points
edPtsz = str2num(get(h.edPtsz,'string'));
edPtsRotx = str2num(get(h.edPtsRotx,'string'));
edPtsRoty = str2num(get(h.edPtsRoty,'string'));

EQ_Pts = sort([3*edPtsz-2 3*edPtsRotx-1 3*edPtsRoty]);

% FILL in the modal masses M and K

% Get nodal list
[nodal{1:(N+1)^2}]=deal([0 0]); % list of nodal points [X Y] (preallocation)
for i=0:(N+1)^2-1
  nodal{i+1}=L*[mod(i,N+1)/N floor(i/(N+1))/N]; % Nodal list construction. numbered left to right
then down to up
end
% Plot nodes
```


©MECAL 2018 Confidential

```
subplot(2,4,4);
cla
for i=1:length(nodal)
  plot(nodal{i}(1),nodal{i}(2),'rx');
  hold on
  text(nodal{i}(1)+8/L,nodal{i}(2)+3/L,num2str(i));
end
title('Nodes')
xlim([-0.1*L 1.1*L])
ylim([-0.1*L 1.1*L])

% horizontale elements
[helement(1:(N*(N+1))).k]=deal(Kex);
[helement(:).m]=deal(Mex);
[helement(:).c]=deal(Cex);
nodali=1:(N+1)^2;
nodali(N+1:N+1:(N+1)^2)=[];
nodalj=nodali+1;

nodali=num2cell(nodali);
nodalj=num2cell(nodalj);
[helement(:).i]=deal(nodali{:});
[helement(:).j]=deal(nodalj{:});
clear nodali nodalj % remove temp variable

% vertical elements
[velement(1:(N*(N+1))).k]=deal(Key);
[velement(:).m]=deal(Mey);
[velement(:).c]=deal(Cey);
nodali=num2cell(1:(N+1)^2-N-1);
nodalj=num2cell(N+2:(N+1)^2);
[velement(:).i]=deal(nodali{:});
[velement(:).j]=deal(nodalj{:});
clear nodali nodalj % remove temp variable

element=[helement velement];

% Plot beams

subplot(2,4,8);
cla
for i=1:length(element)
  plot([nodal{element(i).i}(1) nodal{element(i).j}(1)],[nodal{element(i).i}(2)
nodal{element(i).j}(2)],'r')
  hold on
  text(mean([nodal{element(i).i}(1) nodal{element(i).j}(1)]),mean([nodal{element(i).i}(2)
nodal{element(i).j}(2)]),num2str(i))
end
title('elements')
xlim([-0.1*L 1.1*L])
ylim([-0.1*L 1.1*L])
```

```
% get global K and M without boundairy conditions
[K,M,C] = globalMatrixBuilder(nodal,element);

% add boundairy
% get nodes nodes
% boundNodes=sort(unique(bottom nodes + top nodes + left nodes
boundNodes=sort(unique([1:(N+1) (N+1)^2-N:(N+1)^2 1:N+1:(N+1)^2-N N+1:N+1:(N+1)^2]));

K(3*boundNodes-2,3*boundNodes-2) = K(3*boundNodes-2,3*boundNodes-2)+kez;
K(3*boundNodes-1,3*boundNodes-1) = K(3*boundNodes-1,3*boundNodes-1)+keRot;
K(3*boundNodes,3*boundNodes) = K(3*boundNodes,3*boundNodes)+keRot;

C(3*boundNodes-2,3*boundNodes-2) = C(3*boundNodes-2,3*boundNodes-2)+cez;
C(3*boundNodes-1,3*boundNodes-1) = C(3*boundNodes-1,3*boundNodes-1)+ceRot;
C(3*boundNodes,3*boundNodes) = C(3*boundNodes,3*boundNodes)+ceRot;


% compute eigen freq and mode shapes with magical eig function
[v,lbd] = eig(K,M);
fres = sqrt(diag(lbd))./(2*pi);


% plot first 6 (if available) modes on separate figure
nplt = min(6,length(lbd));

[X,Y]=meshgrid(0:L/N:L,0:L/N:L);

hmodes(1:nplt)=subplot(nplt,4,3:4:4*nplt-1);

global Zods Z hods heqOds scalingz

for i = 1:nplt
  hmodes(i)=subplot(nplt,4,3+4*(i-1));
  a = v(:,i)/v(1,i);
  Z=reshape(a(1:3:end),N+1,N+1);
  surf(X,Y,Z)
  title(['res freq = ' num2str(fres(i)) 'Hz']);
  %set(gca,'visible','off');
end

% ODS


% Now calculate the deflection shape
w = 2*pi*fods;
D = -w^2*M + 1i*w*C+K;
Z = D\Fd;

% Plot the ODS
% because not all masses move either the same way or the opposite way
% we need to take into account the phase.
% We do this by plotting different points in time ( = different values for the phase)
```


©MECAL 2018 Confidential

```
hsim(1)=subplot(2,2,1);
cla
hold on;
scalingz = 0.5/max(abs(Z(1:3:end-2)));
scalingF = 0.2/max(Fd);
for i = 1:length(nodal)
  % plot applied forces
  if Fd(3*i-2) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,0,scalingF*Fd(3*i-2),'r','linewidth',3);
  end % if

  % plot applied moments x
  if Fd(3*i-1) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,L*scalingF*Fd(3*i-1),0,0,'r','linewidth',3);
  end % if

  % plot applied moments y
  if Fd(3*i) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,L*scalingF*Fd(3*i),0,'r','linewidth',3);
  end % if

end % for


% test with animation, plot only first moment, phi = 0
phi = 0;
hods= surf(X,Y,reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),N+1,N+1));
Zods = Z;

title(['ODS for ' num2str(fods) 'Hz']);
ylabel('Y position [m]'); xlabel('X position [m]'); zlabel('Z position [m]');
set(gca,'ZTick',linspace(-0.5,0.5,10));
set(gca,'ZTickLabel',linspace(-1/scalingz,1/scalingz,10));

Zorig= Z; % keep track of original movement
% Now calc forces needed for EQ
for i =1: length(EQ_Pts)
  pt= EQ_Pts(i); % point to equalize
  %zpt = Z(pt); % movement of that point
  Fhlp = zeros(3*(N+1)^2,1); Fhlp(pt) =1; % F for collocated stiffness
  Zcol = D\Fhlp; % collocated compl. z/F for point pt for f=fods;
  keq = 10/Zcol(pt); % 10 times complex stiffness %10/abs(Zcol(pt)); % equivalent stiffness of
active eq
  K(pt,pt) = K(pt,pt) +keq;
end
% calculate new system ( with changed K matrix)
D = -w^2*M + 1i*w*C+K;
Z = D\Fd; % calc new resulting movement

% now plot
hsim(2)=subplot(2,2,3);
cla
```


©MECAL 2018 Confidential

```
for i = 1:length(nodal)
  % plot applied forces
  if Fd(3*i-2) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,0,scalingF*Fd(3*i-2),'r','linewidth',3);
  end % if

  % plot applied moments x
  if Fd(3*i-1) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,L*scalingF*Fd(3*i-1),0,0,'g','linewidth',3);
  end % if

  % plot applied moments y
  if Fd(3*i) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,L*scalingF*Fd(3*i),0,'g','linewidth',3);
  end % if

end % for
scalingz = 0.5/max(abs([Z(1:3:end-2);Zorig(1:3:end-2)]));

heqOds= surf(X,Y,reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),N+1,N+1));

title(['EQUALIZED ODS for ' num2str(fods) 'Hz']);
ylabel('Y position [m]'); xlabel('X position [m]');zlabel('Z position [m]');
set(gca,'ZTick',linspace(-0.5,0.5,10));
set(gca,'ZTickLabel',linspace(-1/scalingz,1/scalingz,10));

global hlinkCamera hlinkZaxis
hlinkCamera=linkprop([hmodes hsim],{'CameraPosition','CameraUpVector'});
hlinkZaxis=linkprop(hsim,'ZLim');
rotate3d on

animate
set(h.but2,'enable','on');
set(h.but1,'enable','on');
end % function calcCallback()

function animate(~,~)
global h Zods Z hods heqOds scalingz N
% Now animate!
set(h.but2,'enable','off');
set(h.but1,'enable','off');
for i = 1: 80
  phi = i*2*pi/40;
  pause(0.15);
  set(hods,'zdata',reshape(scalingz*abs(Zods(1:3:end-2)).*cos(phi+angle(Zods(1:3:end-
2))),N+1,N+1));

  set(heqOds,'zdata',reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),N+1,N+1));
end
set(h.but2,'enable','on');
set(h.but1,'enable','on');
end
```


©MECAL 2018 Confidential

Appendix F Function globalMatrixBuilder code

```
function [K,M,C] = globalMatrixBuilder(nodes,element)
%[K,M] = globalMatrixBuilder(nodes,element)
  % Function for building a global stiffness or mass matricies from an
  % element struct array. Element should be in the form element.k for the
  % element matrix, element.m for the element mass matrix and
  % element.i and element.j should have the node numbers the element is
  % connected to.

% Get number of DOF
DOF=length(element(1).k)/2;

% Preallocate global matricies
K=zeros(length(nodes)*DOF);
M=K;
C=K;

% Superimposing elements on global matericies
for i=1:length(element)
  % Split element matricies in matricies for ii,ij,ji and jj matericies
  % matrix
  kii=element(i).k(1:DOF,1:DOF);
  kij=element(i).k(1:DOF,DOF+1:end);
  kji=element(i).k(DOF+1:end,1:DOF);
  kjj=element(i).k(DOF+1:end,DOF+1:end);

  mii=element(i).m(1:DOF,1:DOF);
  mij=element(i).m(1:DOF,DOF+1:end);
  mji=element(i).m(DOF+1:end,1:DOF);
  mjj=element(i).m(DOF+1:end,DOF+1:end);

  cii=element(i).c(1:DOF,1:DOF);
  cij=element(i).c(1:DOF,DOF+1:end);
  cji=element(i).c(DOF+1:end,1:DOF);
  cjj=element(i).c(DOF+1:end,DOF+1:end);

  % Superimposing
  irang=element(i).i*DOF-(DOF-1):element(i).i*DOF;
  jrang=element(i).j*DOF-(DOF-1):element(i).j*DOF;

  K(irang,irang)=K(irang,irang)+kii;
  K(irang,jrang)=K(irang,jrang)+kij;
  K(jrang,irang)=K(jrang,irang)+kji;
  K(jrang,jrang)=K(jrang,jrang)+kjj;

  M(irang,irang)=M(irang,irang)+mii;
  M(irang,jrang)=M(irang,jrang)+mij;
```


©MECAL 2018 Confidential

```
    M(jrang,irang)=M(jrang,irang)+mji;
    M(jrang,jrang)=M(jrang,jrang)+mjj;

    C(irang,irang)=C(irang,irang)+cii;
    C(irang,jrang)=C(irang,jrang)+cij;
    C(jrang,irang)=C(jrang,irang)+cji;
    C(jrang,jrang)=C(jrang,jrang)+cjj;
  end
```


©MECAL 2018 Confidential

Appendix G Matlab floor insight GUI code 3D Version with floor parameters and variable floor size

The function below uses the function globalMatrixBuilder and “equations.pdf”, see Appendix F and Appendix H.

```
function threeD_floor_gui_floorParameters_variFloor()
% threeD_floor_gui_floorParameters_variFloor()
%
% UI function to calculate mode shapes and
% Operating Deflection Shapes(ODS)of a 3DOF floor
% under force loading and with Equalizer
% The floor has length and width, and can translate only in z and
% rotate around x and y. Forces only in z direction and moments around x
% and y
%
% INPUT:
% for modal plots needed
%  Nx    : number of beam elements in x direction
%  Ny    : number of beam elements in y direction
%  E     : yield modulus [Pa]
%  Ix    : second moment of area vector in x direction [m^4]
%  Iy    : second moment of area vector in y direction [m^4]
%  Lx    : floor length in x direction [m]
%  Ly    : floor length in y direction [m]
%  kez    : edge stiffness between wall an first/last mass in z direction [N/m]
%  keRot  : edge stiffness vector [rotX,rotY] between wall an first/last mass in x and y rotation
[Nm/rad]
%  mtot  : total mass floor [kg]
%
% derived param's
%  Ke    : element stiffness matrix
%  m     : mass of each indiv. mass element = mtot/N [kg]
%  Me    : element mass matrix
%  K     : global stiffness matrix
%  M     : global mass matrix
%
% for ODS also needed
%  cz    : damping between mass elements in z [N/(m/s)]
%  cRot  : damping between mass elements around x and y[Nm/(rad/s)]
%  cez   : dampingconstant between edge and first/last element in z [N/(m/s)]
%  ceRot  : dampingconstant vector [rotX,rotY] between edge and first/last element around x and y
[Nm/(rad/s)]
%  fods  : disturbance frequency [Hz]
%
```


©MECAL 2018 Confidential

```
%  Fdp   : array with applied disturbance forces nodes (for ODS only)
%  FdA   : array with applied disturbance forces amplitude (for ODS only)
%  Mxdp  : array with applied disturbance moments around x nodes (for ODS only)
%  MxdA  : array with applied disturbance moments around x amplitude (for ODS only)
%  Mydp  : array with applied disturbance moments around y nodes (for ODS only)
%  MydA  : array with applied disturbance moments around y amplitude (for ODS only)
%
% for EQUALIZED ODS also needed
%  EQ_Ptsz : z nodes which are equalized
%  EQ_PtsrotX : rotational nodes around x which are equalized
%  EQ_PtsrotY : rotational nodes around x which are equalized
%
% MODEL:
%
% --------- ... --------
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% .....................
% .....................
% .....................
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% || ,~,~,~ ... ,~,~, ||
% || l l l ... l l l ||
% --------- ... --------
%
% with || or - = wall, ~ or l = elements and , = nodes
%
% Z = array with Z translation and rotations around x and y for all nodes
%
%
% OUTPUT:
%  resonance frequencies
%  mode shapes visualized in a plot by only translations
%  ODS
%
% METHOD
%   (-1*lbd*M + K)*Z = F
% with lbd = w^2 and -1 = i^2
% eigen freq and mode shapes follow from the determinant
% of the dynamic matrix: D = K-lbd*M
% this can be computed with the eig(K,M) function in Octave/Matlab
%
% For the ODS the follwing formula holds:
%  (-w^2*M+iw*C +K)*Z = F
% with C being the damping matrix
% Z movements can be calculated by Z = (   )\F
%
% For the Equalized ODS:
% - the collocated percieved stiffness is calculated
```


©MECAL 2018 Confidential

```
%  compliance = K\F, stiffn = 1/compliance(i)
% - A stiffness (to earth) is added to the stiffness marix K
%  which is 10 times greater than the perceived stiffness at that point
%  and frequency.
%  this is the same as Sensitivity = 10
%  K(i,i) = K(i,i) + stiffn
% Mark that the order in which points are being equalized is not trivial
%
% ver 1 fkoo 3-7-2018

close all

% Open figure on secondairy screen if aviable
secondScreenInfo=get(0,'MonitorPositions');
offset=0;
if size(secondScreenInfo,1)>1
  offset=secondScreenInfo(2,1);
end

global h;
% Define input parameters and UI
h.fig1 = figure('position',[80+offset 50 1600 950],'name','TwoD floor GUI with parameters');
% button Calc
h.but1= uicontrol('style','pushbutton','position', [10 880 60 40 ],'string','calc!');
set(h.but1,'Callback',@CalcCallback);
% button reanimate
h.but2= uicontrol('style','pushbutton','position', [80 880 60 40 ],'string','reanimate');
% button equations
h.but3= uicontrol('style','pushbutton','position', [150 880 60 40
],'string','equations','Callback',@showEquations);
set(h.but2,'Callback',@animate);
set(h.but2,'enable','off');
% edit box Nx and text 'Nx'
h.t1 = uicontrol('style', 'text','position',[ 10 840,60,30],'string','Nx ');
h.edNx = uicontrol('style','edit', 'position',[ 80,840,60,30],'string','10');
% edit box Ny and text 'Ny'
h.t2 = uicontrol('style', 'text','position',[ 10 810,60,30],'string','Ny ');
h.edNy = uicontrol('style','edit', 'position',[ 80,810,60,30],'string','10');
% edit box and text E
h.t3 = uicontrol('style', 'text','position',[ 10 780,60,30],'string','E');
h.E = uicontrol('style','edit', 'position',[ 80,780,60,30],'string','3e10');
% edit box and text Ix
h.t4 = uicontrol('style', 'text','position',[ 10 750,60,30],'string','Ix');
h.Ix = uicontrol('style','edit', 'position',[ 80,750,60,30],'string','0.1');
% edit box and text Iy
h.t5 = uicontrol('style', 'text','position',[ 10 720,60,30],'string','Iy');
h.Iy = uicontrol('style','edit', 'position',[ 80,720,60,30],'string','0.1');
% edit box and text Lx
h.t6 = uicontrol('style', 'text','position',[ 10 690,60,30],'string','Lx');
h.Lx = uicontrol('style','edit', 'position',[ 80,690,60,30],'string','20');
% edit box and text Ly
h.t7 = uicontrol('style', 'text','position',[ 10 660,60,30],'string','Ly');
```


©MECAL 2018 Confidential

```
h.Ly = uicontrol('style','edit', 'position',[ 80,660,60,30],'string','20');
% edit box and text kez
h.t8 = uicontrol('style', 'text','position',[ 10 630,60,30],'string','kez ');
h.edkez = uicontrol('style','edit', 'position',[ 80,630,60,30],'string','1.5e20');
% edit box and text ketheta
h.t9 = uicontrol('style', 'text','position',[ 10 600,60,30],'string','keRot ');
h.edkeRot = uicontrol('style','edit', 'position',[ 80,600,60,30],'string','1.5e20');
% edit box and text m
h.t10 = uicontrol('style', 'text','position',[ 10 570,60,30],'string','mtot');
h.edm = uicontrol('style','edit', 'position',[ 80,570,60,30],'string','1.3e5');
% axes for modal plot
%h.ax1 = axes('units','pixels','position',[ 180,50, 1000,700]);
%get(h.ax1,'position')
h.t11 = uicontrol('style', 'text','position',[ 10 520,60,30],'string','cz ');
h.edqz = uicontrol('style','edit', 'position',[ 80,520,60,30],'string','1');
h.t12 = uicontrol('style', 'text','position',[ 10 490,60,30],'string','cRot ');
h.edqRot = uicontrol('style','edit', 'position',[ 80,490,60,30],'string','1');
h.t13 = uicontrol('style', 'text','position',[ 10 460,60,30],'string','cez ');
h.edqez = uicontrol('style','edit', 'position',[ 80,460,60,30],'string','3');
h.t14 = uicontrol('style', 'text','position',[ 10 430,60,30],'string','ceRot ');
h.edqeRot = uicontrol('style','edit', 'position',[ 80,430,60,30],'string','3');
h.t15 = uicontrol('style', 'text','position',[ 10 400,60,30],'string','f ods ');
h.edfods = uicontrol('style','edit', 'position',[ 80,400,60,30],'string','17');
% disturbance forces
h.t16 = uicontrol('style', 'text','position',[ 10 360,120,20],'string','define disturbance');
h.t17 = uicontrol('style', 'text','position',[ 10 330,60,30],'string','nodes z force');
h.edFdp = uicontrol('style','edit', 'position',[ 80,330,60,30],'string','1');
h.t18 = uicontrol('style', 'text','position',[ 10 300,60,30],'string','Amplitude (N) ');
h.edFdA = uicontrol('style','edit', 'position',[ 80,300,60,30],'string','1');
h.t19 = uicontrol('style', 'text','position',[ 10 270,60,30],'string','nodes moment x');
h.edMxdp = uicontrol('style','edit', 'position',[ 80,270,60,30],'string','');
h.t20 = uicontrol('style', 'text','position',[ 10 240,60,30],'string','Amplitude x (Nm)');
h.edMxdA = uicontrol('style','edit', 'position',[ 80,240,60,30],'string','');
h.t21 = uicontrol('style', 'text','position',[ 10 210,60,30],'string','nodes moment y');
h.edMydp = uicontrol('style','edit', 'position',[ 80,210,60,30],'string','');
h.t22 = uicontrol('style', 'text','position',[ 10 180,60,30],'string','Amplitude y (Nm)');
h.edMydA = uicontrol('style','edit', 'position',[ 80,180,60,30],'string','');
% EQUALIZER points
h.t23 = uicontrol('style', 'text','position',[ 10 140,120,20],'string','define EQ. nodes');
h.t24 = uicontrol('style', 'text','position',[ 10 110,60,20],'string','z nodes');
h.edPtsz = uicontrol('style','edit', 'position',[ 80,110,60,30],'string','2 3');
h.t25 = uicontrol('style', 'text','position',[ 10 80,60,30],'string','rot nodes x');
h.edPtsRotx = uicontrol('style','edit', 'position',[ 80,80,60,30],'string','');
h.t26 = uicontrol('style', 'text','position',[ 10 50,60,30],'string','rot nodes y');
h.edPtsRoty = uicontrol('style','edit', 'position',[ 80,50,60,30],'string','');
% calculation is done in the function CalcCallback

end % function OneD_floor_gui



function CalcCallback(~,~)
```


©MECAL 2018 Confidential

```
global h Nx Ny;       % handle to figure
set(h.but2,'enable','off');
set(h.but1,'enable','off');

% input for modeshapes
Nx = str2num(get(h.edNx,'string'));
Ny = str2num(get(h.edNy,'string'));
E = str2num(get(h.E,'string'));
Ix = str2num(get(h.Ix,'string'));
Iy = str2num(get(h.Iy,'string'));
Lx = str2num(get(h.Lx,'string'));
Ly = str2num(get(h.Ly,'string'));

kez = str2num(get(h.edkez,'string'));
keRot = str2num(get(h.edkeRot,'string'));
m = str2num(get(h.edm,'string'));
lex=Lx/Nx;
ley=Ly/Ny;

mex = (m/ley)/Nx;
mey = (m/lex)/Ny;


% input for ODS
cz = str2num(get(h.edqz,'string'));
cRot = str2num(get(h.edqRot,'string'));
cez = str2num(get(h.edqez,'string'));
ceRot = str2num(get(h.edqeRot,'string'));
fods = str2num(get(h.edfods,'string'));

% Element matricies
Ke=E*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=1/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];
Ce=[cz 0 -cz 0 ; 0 cRot 0 -cRot ; -cz 0 cz 0 ; 0 -cRot 0 cRot];

Kex=zeros(6);Key=Kex;Mex=Kex;Mey=Kex;Cex=Kex;Cey=Kex;
Kex([1 3:4 6],[1 3:4 6])=Ke*Ix/lex^3;
Key([1:2 4:5],[1:2 4:5])=Ke*Iy/ley^3;
Mex([1 3:4 6],[1 3:4 6])=Me*mex;
Mey([1:2 4:5],[1:2 4:5])=Me*mey;
Cex([1 3:4 6],[1 3:4 6])=Ce;
Cey([1:2 4:5],[1:2 4:5])=Ce;

% Distrubance forces
Fd = zeros(3*(Nx+1)*(Ny+1),1);
edFdp = str2num(get(h.edFdp,'string'));
edFdA = str2num(get(h.edFdA,'string'));
Fd(3*edFdp-2)=edFdA;

% Distrubance moments
edMdp = str2num(get(h.edMxdp,'string'));
edMdA = str2num(get(h.edMxdA,'string'));
```


©MECAL 2018 Confidential

```
Fd(3*edMdp-1)=edMdA;

edMdp = str2num(get(h.edMydp,'string'));
edMdA = str2num(get(h.edMydA,'string')); %#ok<*ST2NM>
Fd(3*edMdp)=edMdA;

% check dat Fd niet per ongeluk te groot wordt
if length(Fd) > 3*(Nx+1)*(Ny+1)
  Fd = Fd(1:3*3*(Nx+1)*(Ny+1));
end

% EQ points
edPtsz = str2num(get(h.edPtsz,'string'));
edPtsRotx = str2num(get(h.edPtsRotx,'string'));
edPtsRoty = str2num(get(h.edPtsRoty,'string'));

EQ_Pts = sort([3*edPtsz-2 3*edPtsRotx-1 3*edPtsRoty]);

% FILL in the modal masses M and K

% Get nodal list
[nodal{1:(Nx+1)*(Ny+1)}]=deal([0 0]); % list of nodal points [X Y] (preallocation)
for i=0:(Nx+1)*(Ny+1)-1
  nodal{i+1}=[Lx*mod(i,Nx+1)/Nx Ly*floor(i/(Nx+1))/Ny]; % Nodal list construction. numbered left to
right then down to up
end
% Plot nodes
subplot(2,4,4);
cla
for i=1:length(nodal)
  plot(nodal{i}(1),nodal{i}(2),'rx');
  hold on
  text(nodal{i}(1)+8/Lx,nodal{i}(2)+3/Ly,num2str(i));
end
title('Nodes')
xlim([-0.1*Lx 1.1*Lx])
ylim([-0.1*Ly 1.1*Ly])

% horizontale elements
[helement(1:(Nx*(Ny+1))).k]=deal(Kex);
[helement(:).m]=deal(Mex);
[helement(:).c]=deal(Cex);
nodali=1:(Nx+1)*(Ny+1);
nodali(Nx+1:Nx+1:(Nx+1)*(Ny+1))=[];
nodalj=nodali+1;

nodali=num2cell(nodali);
nodalj=num2cell(nodalj);
[helement(:).i]=deal(nodali{:});
[helement(:).j]=deal(nodalj{:});
clear nodali nodalj % remove temp variable
```


©MECAL 2018 Confidential

```
% vertical elements
[velement(1:(Ny*(Nx+1))).k]=deal(Key);
[velement(:).m]=deal(Mey);
[velement(:).c]=deal(Cey);
nodali=num2cell(1:(Nx+1)*(Ny+1)-Nx-1);
nodalj=num2cell(Nx+2:(Nx+1)*(Ny+1));
[velement(:).i]=deal(nodali{:});
[velement(:).j]=deal(nodalj{:});
clear nodali nodalj % remove temp variable

element=[helement velement];

% Plot beams

subplot(2,4,8);
cla
for i=1:length(element)
  plot([nodal{element(i).i}(1) nodal{element(i).j}(1)],[nodal{element(i).i}(2)
nodal{element(i).j}(2)],'r')
  hold on
  text(mean([nodal{element(i).i}(1) nodal{element(i).j}(1)]),mean([nodal{element(i).i}(2)
nodal{element(i).j}(2)]),num2str(i))
end
title('elements')
xlim([-0.1*Lx 1.1*Lx])
ylim([-0.1*Ly 1.1*Ly])

% get global K and M without boundairy conditions
[K,M,C] = globalMatrixBuilder(nodal,element);

% add boundairy
% get nodes nodes
% boundNodes=sort(unique(bottom nodes + top nodes + left nodes
boundNodes=sort(unique([1:(Nx+1) (Nx+1)*(Ny+1)-Nx:(Nx+1)*(Ny+1) 1:Nx+1:(Nx+1)*(Ny+1)-Nx
Nx+1:Nx+1:(Nx+1)*(Ny+1)]));

K(3*boundNodes-2,3*boundNodes-2) = K(3*boundNodes-2,3*boundNodes-2)+kez;
K(3*boundNodes-1,3*boundNodes-1) = K(3*boundNodes-1,3*boundNodes-1)+keRot;
K(3*boundNodes,3*boundNodes) = K(3*boundNodes,3*boundNodes)+keRot;

C(3*boundNodes-2,3*boundNodes-2) = C(3*boundNodes-2,3*boundNodes-2)+cez;
C(3*boundNodes-1,3*boundNodes-1) = C(3*boundNodes-1,3*boundNodes-1)+ceRot;
C(3*boundNodes,3*boundNodes) = C(3*boundNodes,3*boundNodes)+ceRot;


% compute eigen freq and mode shapes with magical eig function
[v,lbd] = eig(K,M);
fres = sqrt(diag(lbd))./(2*pi);


% plot first 6 (if available) modes on separate figure
nplt = min(6,length(lbd));
```

```
[X,Y]=meshgrid(0:Lx/Nx:Lx,0:Ly/Ny:Ly);

hmodes(1:nplt)=subplot(nplt,4,3:4:4*nplt-1);

global Zods Z hods heqOds scalingz

for i = 1:nplt
  hmodes(i)=subplot(nplt,4,3+4*(i-1));
  a = v(:,i)/v(1,i);
  Z=reshape(a(1:3:end),Ny+1,Nx+1);
  surf(X,Y,Z)
  title(['res freq = ' num2str(fres(i)) 'Hz']);
  %set(gca,'visible','off');
end

% ODS


% Now calculate the deflection shape
w = 2*pi*fods;
D = -w^2*M + 1i*w*C+K;
Z = D\Fd;

% Plot the ODS
% because not all masses move either the same way or the opposite way
% we need to take into account the phase.
% We do this by plotting different points in time ( = different values for the phase)
hsim(1)=subplot(2,2,1);
cla
hold on;
scalingz = 0.5/max(abs(Z(1:3:end-2)));
scalingF = 0.2/max(Fd);
for i = 1:length(nodal)
  % plot applied forces
  if Fd(3*i-2) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,0,scalingF*Fd(3*i-2),'r','linewidth',3);
  end % if

  % plot applied moments x
  if Fd(3*i-1) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,Lx*scalingF*Fd(3*i-1),0,0,'r','linewidth',3);
  end % if

  % plot applied moments y
  if Fd(3*i) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,Ly*scalingF*Fd(3*i),0,'r','linewidth',3);
  end % if

end % for
```


©MECAL 2018 Confidential

```
% test with animation, plot only first moment, phi = 0
phi = 0;
hods= surf(X,Y,reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),Ny+1,Nx+1));
Zods = Z;

title(['ODS for ' num2str(fods) 'Hz']);
ylabel('Y position [m]'); xlabel('X position [m]'); zlabel('Z position [m]');
set(gca,'ZTick',linspace(-0.5,0.5,10));
set(gca,'ZTickLabel',linspace(-1/scalingz,1/scalingz,10));

Zorig= Z; % keep track of original movement
% Now calc forces needed for EQ
for i =1: length(EQ_Pts)
  pt= EQ_Pts(i); % point to equalize
  %zpt = Z(pt); % movement of that point
  Fhlp = zeros(3*(Nx+1)*(Ny+1),1); Fhlp(pt) =1; % F for collocated stiffness
  Zcol = D\Fhlp; % collocated compl. z/F for point pt for f=fods;
  keq = 10/Zcol(pt); % 10 times complex stiffness %10/abs(Zcol(pt)); % equivalent stiffness of
active eq
  K(pt,pt) = K(pt,pt) +keq;
end
% calculate new system ( with changed K matrix)
D = -w^2*M + 1i*w*C+K;
Z = D\Fd; % calc new resulting movement

% now plot
hsim(2)=subplot(2,2,3);
cla
for i = 1:length(nodal)
  % plot applied forces
  if Fd(3*i-2) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,0,scalingF*Fd(3*i-2),'r','linewidth',3);
  end % if

  % plot applied moments x
  if Fd(3*i-1) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,Lx*scalingF*Fd(3*i-1),0,0,'g','linewidth',3);
  end % if

  % plot applied moments y
  if Fd(3*i) ~= 0
    quiver3(nodal{i}(1),nodal{i}(2),0,0,Ly*scalingF*Fd(3*i),0,'g','linewidth',3);
  end % if

end % for
scalingz = 0.5/max(abs([Z(1:3:end-2);Zorig(1:3:end-2)]));

heqOds= surf(X,Y,reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),Ny+1,Nx+1));

title(['EQUALIZED ODS for ' num2str(fods) 'Hz']);
ylabel('Y position [m]'); xlabel('X position [m]');zlabel('Z position [m]');
set(gca,'ZTick',linspace(-0.5,0.5,10));
```

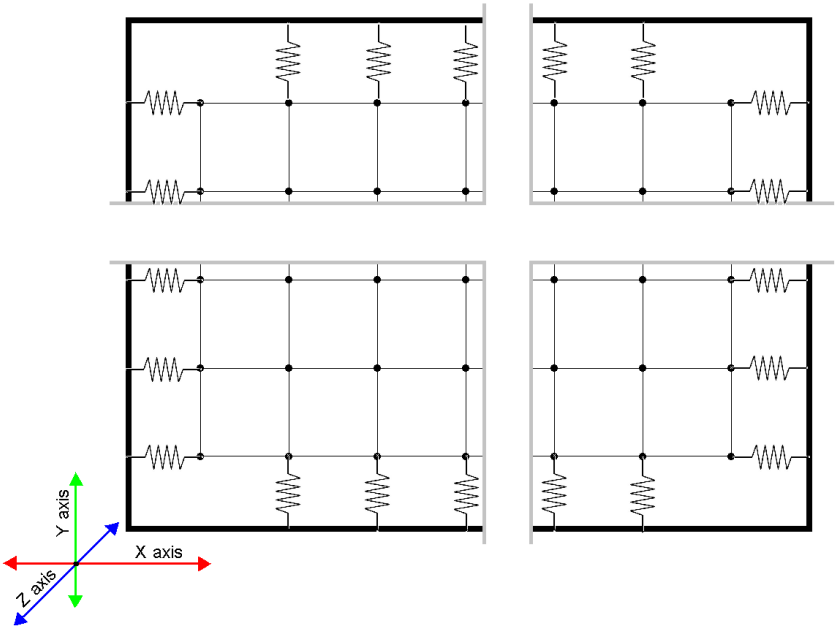

©MECAL 2018 Confidential

```
set(gca,'ZTickLabel',linspace(-1/scalingz,1/scalingz,10));

global hlinkCamera hlinkZaxis
hlinkCamera=linkprop([hmodes hsim],{'CameraPosition','CameraUpVector'});
hlinkZaxis=linkprop(hsim,'ZLim');
rotate3d on

animate
set(h.but2,'enable','on');
set(h.but1,'enable','on');
end % function calcCallback()

function animate(~,~)
global h Zods Z hods heqOds scalingz Nx Ny
% Now animate!
set(h.but2,'enable','off');
set(h.but1,'enable','off');
for i = 1: 80
  phi = i*2*pi/40;
  pause(0.15);
  set(hods,'zdata',reshape(scalingz*abs(Zods(1:3:end-2)).*cos(phi+angle(Zods(1:3:end-
2))),Ny+1,Nx+1));

  set(heqOds,'zdata',reshape(scalingz*abs(Z(1:3:end-2)).*cos(phi+angle(Z(1:3:end-2))),Ny+1,Nx+1));
end
set(h.but2,'enable','on');
set(h.but1,'enable','on');
end

function showEquations(~,~)
system('equations.pdf');
end
```


©MECAL 2018 Confidential

Appendix H Equations.pdf

This GUI builds from the inputs the following system of beams:



Each node (round connection points) has 3DOFs, one translation in z and 2 rotations around x and y. The beams (thin lines) have the element stiffness, mass and damping matrices

$$K_e = \frac{EI}{L_e^3}\begin{bmatrix} 12 & 6 & -12 & 6 \\ 6 & 4 & -6 & 2 \\ -12 & -6 & 12 & -6 \\ 6 & 2 & -6 & 4 \end{bmatrix} M_e = \frac{m_e}{420}\begin{bmatrix} 156 & 22 & 54 & -13 \\ 22 & 4 & 13 & -3 \\ 54 & 13 & 156 & -22 \\ -13 & -3 & -22 & 4 \end{bmatrix} C_e = \begin{bmatrix} c_z & 0 & -c_z & 0 \\ 0 & c_\theta & 0 & -c_\theta \\ -c_z & 0 & c_z & 0 \\ 0 & -c_\theta & 0 & c_\theta \end{bmatrix}$$

These matrices are 2DOF and are made 3DOF by only allowing bending in plane, thus ignoring torsion. From these and the boundary springs the global matrices can be constructed. The eigenmodes can be calculated from the stiffness and mass matrix by solving: K*V = M*V*E, where E is a diagonal matrix with eigenvalues and the columns of V contain the eigenvectors. The Operating Deflection Shapes (ODS) are calculated from the dynamic matrix (D) and the external force vector (Fd) by ODS=D\Fd. The dynamic matrix is calculated with $D(\omega) = -\omega^2 M + \omega i C + K$. The ODS solution is stripped to contain only the translation z and this result is animated. The ODS solution with equalizers (EQ) the dynamic matrix is altered to have a 10 times higher stiffness at these points equalizer points. The rest of the solution for the ODS with EQ is the same as without EQ.

©MECAL 2018 Confidential

Appendix I High to low order estimation Matlab code

```
% Clean matlab
clear
close all
clc

% Parameters
N=2;        % Number of elements [-]
L=10;        % Length of floor [m]
kez=1.5e8;     % Stiffness of support [N/m]
ketheta=1.5e8;   % Rotational stiffness of support [Nm/rad]
kmid=4e8;      % Stiffness in middle of the floor [N/m]
m=1.3e5;      % Total mass of floor [kg]
n = 2000;      % Number of frequency points
fRangeLog=[-1.5 4]; % Log space range of frf's

% Calc parameters
me = m/N;      % Element mass [kg]
f = logspace(fRangeLog(1),fRangeLog(2),n)'; % Frequency points [Hz]

% Element matricies
Ke=kmid*N/48*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=me/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];

% Global matricies
M=zeros(2*N+2,2*N+2);
K=M;
for i = 1:N
  M(2*i-1:2*i+2,2*i-1:2*i+2) = M(2*i-1:2*i+2,2*i-1:2*i+2) + Me;
  K(2*i-1:2*i+2,2*i-1:2*i+2) = K(2*i-1:2*i+2,2*i-1:2*i+2) + Ke;
end

% Apply boundairy conditions
K(1,1) = K(1,1)+kez;
K(2,2) = K(2,2)+ketheta;
K(end-1,end-1) = K(end-1,end-1)+kez;
K(end,end) = K(end,end)+ketheta;

w = 2*pi*f; % Frequency points [rad/s]

% Calc transfers
% calculate the frf's for each frequency
% Force on mass k
F = eye(length(M));
```


©MECAL 2018 Confidential

```
% Preallocate frf
frf=zeros(length(M),length(M),n);

% get frf
for k=1:length(F)
  for i = 1:n
    D = -w(i)^2*M + K;
    Z = D\F(:,k);
    frf(k,:,i)=Z';
  end
end

% Get K and M matrix from stripped frf
frfStripped=frf(1:2:end-1,1:2:end-1,:); % Get stripped frfs

% estimate K at 10
Kestimate=inv(abs(frfStripped(:,:,10)));

% estimate M by the difference between s1 and s2
s1=100;s2=895;
Mestimate=((inv(frfStripped(:,:,s1))-Kestimate)/(w(s1)*1i)-(inv(frfStripped(:,:,s2))-
Kestimate)/(w(s2)*1i))/(w(s1)*1i-w(s2)*1i);

% Check responce
F2 = eye(length(Kestimate));
frfResponce=complex(zeros(size(Kestimate,1),size(Kestimate,1),n));
for k=1:length(F2)
  for i = 1:n
    D = -w(i)^2*Mestimate + Kestimate;
    Z = D\F2(:,k);
    frfResponce(k,:,i)=transpose(Z);
  end
end

% Plot frf
for i=1:size(frfStripped,1)
  for j=1:size(frfStripped,2)
    figure % all transfer in seprate figure
    % Magnitude
    subplot(2,1,1)
    loglog(f,squeeze(abs(frfStripped(i,j,:))),'k','linewidth',1.5);
    title(['|z' num2str(j) '/F' num2str(i) '|'])
    xlabel('f [Hz]'); ylabel('compl. [m/N]');
    hold on
    loglog(f,squeeze(abs(frfResponce(i,j,:))),'c')

    % Angle
    subplot(2,1,2)
    semilogx(f,squeeze(angle(frfStripped(i,j,:)))*180/pi,'k','linewidth',1.5);
    title(['|z' num2str(j) '/F' num2str(i) '|'])
    xlabel('f [Hz]'); ylabel('phase [deg.]');
    hold on
```


©MECAL 2018 Confidential

```
    semilogx(f,squeeze(angle(frfResponce(i,j,:)))*180/pi,'c')
  end
end
```


©MECAL 2018 Confidential

Appendix J Matrices estimation Matlab code

```
% Clean matlab
clear
close all
clc

% Parameters
N=2; % Number of elements [-]
L=10; % Length of floor [m]
kez=1.5e8; % Stiffness of support [N/m]
ketheta=1.5e8; % Rotational stiffness of support [Nm/rad]
c= 300000; % Damping internal [Ns/m or Nms/rad]
ce=300000; % Damping boundairy [Ns/m or Nms/rad]
kmid=4e8; % Stiffness in middle of the floor [N/m]
m=1.3e5; % Total mass of floor [kg]
n = 2000; % Number of frequency points
fRangeLog=[-1.5 4]; % Log space range of frf's

% Calc parameters
me = m/N; % Element mass [kg]
f = logspace(fRangeLog(1),fRangeLog(2),n)'; % Frequency points [Hz]

% Element matricies
Ke=kmid*N/48*[12 6 -12 6 ; 6 4 -6 2 ; -12 -6 12 -6 ; 6 2 -6 4];
Me=me/420*[156 22 54 -13 ; 22 4 13 -3 ; 54 13 156 -22 ; -13 -3 -22 4];
Ce=c*[1 0 -1 0;0 1 0 -1;-1 0 1 0;0 -1 0 1];

% Global matricies
M= zeros(2*N+2,2*N+2);
K=M;C=M;
for i = 1:N
  M(2*i-1:2*i+2,2*i-1:2*i+2) = M(2*i-1:2*i+2,2*i-1:2*i+2) + Me;
  K(2*i-1:2*i+2,2*i-1:2*i+2) = K(2*i-1:2*i+2,2*i-1:2*i+2) + Ke;
  C(2*i-1:2*i+2,2*i-1:2*i+2) = C(2*i-1:2*i+2,2*i-1:2*i+2) + Ce;
end

% Apply boundairy conditions
K(1,1) = K(1,1)+kez;
K(2,2) = K(2,2)+ketheta;
K(end-1,end-1) = K(end-1,end-1)+kez;
K(end,end) = K(end,end)+ketheta;
C(1,1) = C(1,1)+ce;
C(2,2) = C(2,2)+ce;
C(end-1,end-1) = C(end-1,end-1)+ce;
C(end,end) = C(end,end)+ce;

% Reduce order for simple testing of procedure
```


©MECAL 2018 Confidential

```
M=M(1:2:end-1,1:2:end-1);
K=K(1:2:end-1,1:2:end-1);
C=C(1:2:end-1,1:2:end-1);

w = 2*pi*f; % Frequency points [rad/s]

% Calc transfers
% calculate the frf's for each frequency
% Force on mass k
F = eye(length(M));

% Preallocate frf
frf=zeros(length(M),length(M),n);

% get frfs
for i = 1:n
  Dvec = -w(i)^2*M + C*w(i)*1i + K;
  for k=1:length(F)
    Z = Dvec\F(:,k);
    frf(k,:,i)=transpose(Z);
  end
end

% first estimate method

% estimate K
fli=500;
Kestmate1=inv(abs(frf(:,:,fli)));

% estimate M
fhi=1300;
Mestimate1{1}=inv(abs(-w(fhi)^2*frf(:,:,fhi)));

% better estimation M
Mestimate1{2}=inv(real(-w(fhi)^2*frf(:,:,fhi)));

% second estimate method

% estimate K
Kestmate2=Kestmate1;

% estimate C
f2i=700;
Cestmate2=(inv(frf(:,:,f2i))-Kestmate2)/(w(f2i)*1i);

% estimate M
f3i=1000;
Mestmate2=((inv(frf(:,:,f3i))-Kestmate2)/(w(f3i)*1i)-Cestmate2)/(w(f3i)*1i);

% LSE

% get dynamic matrix in a series of vectors form
```


©MECAL 2018 Confidential

```
Dvec=complex(zeros(size(frf,3),size(frf,1)*size(frf,2)));
for i=1:size(frf,3)
  dummy=inv(frf(:,:,i));
  Dvec(i,:)=dummy(:);
end

% get phi matrix for LSE
phi=[-w.^2 w*1i ones(size(w))];

% estimate matrices
matericies=(phi\Dvec);

% split and reshape estimates
Mestmate3=real(reshape(matericies(1,:),size(frf,1),size(frf,2)));
Cestmate3=real(reshape(matericies(2,:),size(frf,1),size(frf,2)));
Kestmate3=real(reshape(matericies(3,:),size(frf,1),size(frf,2)));
```


©MECAL 2018 Confidential

Appendix K GUI code

©MECAL 2018 Confidential

FloorDynamicsAndVibrationGUIV2

```
% GUI start
% GUI for floor dynamic analysis with EQ tuning and vibration levels
% 28-6-2018 fkoo
% V2 4-7-2018 fkoo
% following had been added:
% - continious ODS animation
% - free drawing tool
% - removed seperate transfer field
% - 2 methods for adding EQs, one which assumes a local stiffness becomes
%    10 times higher as before and the old on with EQ adder with tuning
% - if user selects a point which already has a EQ the user is asked what
% he wants to do
% - EQ forces estimation
% - beep added when a new messages is written to console

% Starts a GUI for floor analysis
function FloorDynamicsAndVibrationGUIV2()
close all % close old figures (including an already running FloorDynamicsAndVibrationGUIV2)
global h

% Clear old data
h=[];

% get screen size
screenSize=get(0,'MonitorPositions');
screenX=screenSize(1,3);
screenY=screenSize(1,4);

% Open figure
```

```
h.mainScreen = figure('position',round([0.1*screenX 0.1*screenY 0.8*screenX
0.8*screenY]),'name','Floor analysis GUI with advance EQ');

% build console
h.Console = uipanel('Title','Console','Position',[0.8 0.05 0.2 0.95]);
h.ConsoleUI = uicontrol('Parent',h.Console,'Style','text',...
  'Units','normalized','Position', [0 0 1 1],...
  'HorizontalAlignment','left');

% Load screen
source.Value=1;
h.loadScreen=uipanel('Title','Load data','Position',[0.43 0.3 0.14 0.4]); % only build to be able
to be deleted
setInputSelector(source,[]) % build load screen

% Restart button
h.clean = uicontrol('style','pushbutton','Units','normalized',...
  'Position', [0.8 0 0.05 0.03],...
  'string','Restart','Callback',@restart);

% Clean console button
h.cleanConsole = uicontrol('style','pushbutton','Units','normalized',...
  'Position', [0.9 0 0.05 0.03],...
  'string','Clean console','Callback',@cleanConsole);

end
```

setInputSelector

```
function setInputSelector(source,~)
% Load screen builder
global h

% Store source dropdown value before it is delete
val=source.Value;

% clear old load screen
delete(h.loadScreen)

% Build new load screen
h.loadScreen = uipanel('Title','Load data','Position',[0.43 0.3 0.14 0.4]);

% dropdown menu for input method ('Matfile','Experiment','Workspace')
% With callback to rebuild loadscreen to new method
h.loadScreenUI = uicontrol('Parent',h.loadScreen,'Style', 'popup',...
  'String', {'Matfile','Experiment','Workspace'},...
  'Units','normalized','Position', [0 0.9 1 0.08],...
  'Callback', @setInputSelector);

% Set dropdown to chosen method
h.loadScreenUI.Value=val;
```

```
% Build rest of load screen to chosen method
switch val
  case 1
    % Method == Matfile
    h.loadScreenUIText = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.85 1 0.05],'string','Matfile',...
      'HorizontalAlignment','left');
    h.loadScreenMatfile = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.8 1 0.05]);
    h.loadScreenLoadBut = uicontrol('Parent',h.loadScreen,...
      'style','pushbutton','Units','normalized',...
      'Position', [0 0 1 0.1],'string','Load',...
      'Callback',@loadMatFile);
  case 2
    % Method == Experiment
    h.loadScreenUIText = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.85 1 0.05],'string','Experiment directory',...
      'HorizontalAlignment','left');
    h.loadScreenFileDir = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.8 1 0.05]);
    h.loadScreenUIText2 = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.74 1 0.05],'string','Reversed sensors',...
      'HorizontalAlignment','left');
    h.loadScreen_sRev = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.69 1 0.05]);
    h.loadScreenUIText3 = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.63 1 0.05],'string','Coherence tolerance check',...
      'HorizontalAlignment','left');
    h.loadScreenCohTol = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.58 1 0.05]);
    h.loadScreenUITex4 = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.52 1 0.05],'string','Coherence tolerance check range',...
      'HorizontalAlignment','left');
    h.loadScreenCohTolRange = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.47 1 0.05]);
    h.loadScreenUITex5 = uicontrol('Parent',h.loadScreen,...
      'style','text','Units','normalized',...
      'Position', [0 0.41 1 0.05],'string','Floor size',...
      'HorizontalAlignment','left');
    h.FloorSize = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
```


©MECAL 2018 Confidential

```
        'Position', [0 0.36 1 0.05]);
      h.loadScreenUITex6 = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.3 1 0.05],'string','Floor grid size',...
        'HorizontalAlignment','left');
      h.GridSize = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.25 1 0.05]);
      h.loadScreenLoadExp = uicontrol('Parent',h.loadScreen,...
        'style','pushbutton','Units','normalized',...
        'Position', [0 0 1 0.1],'string','Load',...
        'Callback',@LoadExp);
    case 3
      % Method == Workspace
      h.loadScreenUIText = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.85 1 0.05],'string','frf name',...
        'HorizontalAlignment','left');
      h.loadScreen_frfName = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.8 1 0.05]);
      h.loadScreenUIText2 = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.74 1 0.05],'string','f name',...
        'HorizontalAlignment','left');
      h.loadScreen_fName = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.69 1 0.05]);
      h.loadScreenUIText3 = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.63 1 0.05],'string','coherence name',...
        'HorizontalAlignment','left');
      h.loadScreen_CohName = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.58 1 0.05]);
      h.loadScreenUIText3b = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.52 1 0.05],'string','Vibration force levels name',...
        'HorizontalAlignment','left');
      h.loadScreen_VibFName = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.47 1 0.05]);
      h.loadScreenUIText4 = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.41 1 0.05],'string','X grid mesh name',...
        'HorizontalAlignment','left');
      h.XgridName = uicontrol('Parent',h.loadScreen,...
        'style','edit','Units','normalized',...
        'Position', [0 0.36 1 0.05]);
      h.loadScreenUIText5 = uicontrol('Parent',h.loadScreen,...
        'style','text','Units','normalized',...
        'Position', [0 0.3 1 0.05],'string','Y grid mesh name',...
```


©MECAL 2018 Confidential

```
      'HorizontalAlignment','left');
    h.YgridName = uicontrol('Parent',h.loadScreen,...
      'style','edit','Units','normalized',...
      'Position', [0 0.25 1 0.05]);
    h.loadScreenLoadExp = uicontrol('Parent',h.loadScreen,...
      'style','pushbutton','Units','normalized',...
      'Position', [0 0 1 0.1],'string','Load',...
      'Callback',@LoadWorkSpace);
end
end
```

restart

```
function restart(~,~)
% restart GUI by calling the the original function (this GUI will be
% stopped by the close all command at the beginning of the function)
FloorDynamicsAndVibrationGUIV2();
end
```

cleanConsole

```
function cleanConsole(~,~)
global h
% empty console
h.ConsoleUI.String=[];
end
```

consoleWrite

```
function consoleWrite(message)
% Write function to the console

% sound beep to let user take note
beep;

% add new line to message
message(end+1:end+2)='\n';
global h
% get old console writes (last line is empty)
oldMessage=h.ConsoleUI.String(1:end-1,:);
% if nothing is yet written to concole do nothing with old message
if ~strcmp(oldMessage,'')
  % if a single line was written add new line
  if size(oldMessage,1)==1
    oldMessage(end+1:end+2)='\n';
    % Remove console writes longer then 25 lines
  elseif size(oldMessage,1)==25
    oldMessage(1,:)=[]; % remove first line
    oldMessage=oldMessageRewritten(oldMessage);
  else
```

```
    oldMessage=oldMessageRewritten(oldMessage);
  end
  message=[oldMessage message]; % Add message to old messages
end
h.ConsoleUI.String=sprintf(message); % Set console
end
```

oldMessageRewritten

```
function oldMessage=oldMessageRewritten(oldMessage)
% convert matrix to array with rows separeted by new lines
oldMessage(:,end+1:end+2)=repmat('\n',size(oldMessage,1),1); % add new line to all rows
oldMessage=oldMessage';
oldMessage=oldMessage(:);
oldMessage=oldMessage';
end
```

loadMatFile

```
function loadMatFile(~,~)
% Function for loading matfile to globals
global h glo_frf glo_f glo_Coh glo_VibF glo_X glo_Y glo_floor glo_EQ

% Set texts for input choices (if these are in the matfile these will be
% overwritting to the given texts
h.text.massPoints='';
h.text.mass='';
h.text.frameLoc1='';
h.text.frameLoc2='';
h.text.frameMass='';
h.text.frameStiff='';
h.text.stiffPoint='';
h.text.stiffPointValue='';

% Start with a empty global floor
glo_floor=[];
glo_EQ=[];

% Try to load the file and get data
try
  load(h.loadScreenMatfile.String); % load file
  glo_frf=frf; % store frf from file to global
  glo_f=f; % store f from file to global
  glo_Coh=Coh; % store Coh from file to global
  glo_VibF=VibF; % Store VibF from file to global
  glo_X=X; % store X from file to global
  glo_Y=Y; % store Y from file to global

  % if the variable floorVar is in matfile store it to global
  if exist('floorVar','var')
    glo_floor=floorVar;
```

```
    end
    % if the variable EQ is in matfile store it to global
    if exist('EQ','var')
      glo_EQ=EQ;
    end
    % if the variable massPoints is in matfile store it to global
    if exist('massPoints','var')
      h.text.massPoints=massPoints;
    end
    % if the variable mass is in matfile store it to global
    if exist('mass','var')
      h.text.mass=mass;
    end
    % if the variable frameLoc1 is in matfile store it to global
    if exist('frameLoc1','var')
      h.text.frameLoc1=frameLoc1;
    end
    % if the variable frameLoc2 is in matfile store it to global
    if exist('frameLoc2','var')
      h.text.frameLoc2=frameLoc2;
    end
    % if the variable frameMass is in matfile store it to global
    if exist('frameMass','var')
      h.text.frameMass=frameMass;
    end
    % if the variable frameStiff is in matfile store it to global
    if exist('frameStiff','var')
      h.text.frameStiff=frameStiff;
    end
    % if the variable stiffPoint is in matfile store it to global
    if exist('stiffPoint','var')
      h.text.stiffPoint=stiffPoint;
    end
    % if the variable stiffPointValue is in matfile store it to global
    if exist('stiffPointValue','var')
      h.text.stiffPointValue=stiffPointValue;
    end

    % if the try got to here it was succesfull so this can be written in
    % console
    consoleWrite(['Matfile: ' h.loadScreenMatfile.String ' loaded.']);

    % clear load screen away
    delete(h.loadScreen)

    % load main menu
    buildMainMenu
  catch me
    % Write in console that the load failed and why
    if isfield(h,'loadScreenMatfile.String')
      consoleWrite(['Matfile: ' h.loadScreenMatfile.String ' failed to load. Due to: ' me.message]);
    else
```

```
    consoleWrite('Failed to build main menu.');
  end
end
end
```

LoadExp

```
function LoadExp(~,~)
% Function for experiment data to globals
global h glo_frf glo_f glo_Coh glo_VibF glo_X glo_Y glo_floor glo_EQ

% Get user given info about the experiment to load
fileDir=h.loadScreenFileDir.String;
sReversed=str2num(h.loadScreen_sRev.String); %#ok<ST2NM>
CohTol=str2double(h.loadScreenCohTol.String);
CohTolRange=str2num(h.loadScreenCohTolRange.String); %#ok<ST2NM>
floorSize=str2num(h.FloorSize.String); %#ok<ST2NM>
gridSize=str2num(h.GridSize.String); %#ok<ST2NM>

% Start with a empty global floor
glo_floor=[];
glo_EQ=[];

h.text.massPoints='';
h.text.mass='';
h.text.frameLoc1='';
h.text.frameLoc2='';
h.text.frameMass='';
h.text.frameStiff='';
h.text.stiffPoint='';
h.text.stiffPointValue='';

clc % clear matlab command line (to capture Coh warnings)

% try to load experiment to globals
try
  % load
  [glo_frf,glo_f,glo_Coh,glo_VibF,glo_X,glo_Y]=
specTestStiffAndVib2data(fileDir,sReversed,CohTol,CohTolRange,floorSize,gridSize);

  % Get warning messeages from command line viewer to display in console
  [cmdWin]=com.mathworks.mde.cmdwin.CmdWin.getInstance;
  cmdWin_comps=get(cmdWin,'Components');
  subcomps=get(cmdWin_comps(1),'Components');
  text_container=get(subcomps(1),'Components');
  output_string=get(text_container(1),'text');

  % Display warnings
  if ~strcmp(output_string,'>> ') % in case there are no warnings
    % find Coh warnings
    warningStart=find(output_string==':')+2;
```

```
    warningEnd=find(output_string=='.');
    for i=1:length(warningStart)
      % Write all warnings
      consoleWrite(output_string(warningStart(i):warningEnd(i)));
    end
  end

  % if the try got to here the load was succesfull
  consoleWrite(['Experiment: ' fileDir ' loaded.']);

  % clear load screen away
  delete(h.loadScreen)

  % load main menu
  buildMainMenu

catch me
  % Write in console that the load failed and why
  consoleWrite(['Failed to load experiment due to: ' me.message]);
end

end
```

specTestStiffAndVib2data

```
function [frf,f,Coh,VibF,X,Y] =
specTestStiffAndVib2data(fileDir,sReversed,CohTol,CohTolRange,floorSize,gridSize,matFile)
% [frf,f,Coh,Vib,X,Y] =
stiffAndVib2data(fileDir,sReversed,CohTol,CohTolRange,floorSize,gridSize,matFile)
%   Function for converting specTest experiments into frf's and vibration
%   Data Vib. Give the folder fileDir where the specTest data is stored.
%   SpecTest data filenames for the stiffness should be in the form
%   'F(%s)_s(%s),(%s),(%s).mat' where (%s) are numbers for the in and
%   outputs and be stored in the folder 'Stiff'. For the vibration data
%   file names should be in the form 's(%s),(%s),(%s).mat' and stored in
%   the folder 'Vib'. This function can only use 1 input (F) and 1:3
%   outputs (s). If 1 or more of the outputs are inverted this can be
%   adjusted by giving the sensor number in sReversed. If this is not
%   needed pass an empty array ([]). The data is checked for coherence is
%   above CohTol in the CohTolRange. If this is not the case the function
%   will still run but give an warning. A floor grid is made from the floor
%   and grid size [x y]. The frf's can be stored to mat-file matFile.
% Update 11-7-2018 fkoo to handle variable number (1:3) sensor inputs, and
% remove of bug to calculate vibration forces

% Function input check
if nargin<6
    error('Not enough inputs for stiffAndVib2data')
elseif nargin>7
    error('To many inputs for stiffAndVib2data')
elseif ~isdir(fileDir)
```

```
    error('Experiment folder does not exist')
end

% get frfs from other function
[frf,f,Coh,X,Y] = specTest2frf([fileDir '/Stiff'],sReversed,CohTol,CohTolRange,floorSize,gridSize);

% get vibration data
% Get locations and error checking
files = dir([fileDir '/Vib']); % read all files in dir

% Check if all data is there and no duplicated exist

% Get s locations and remove file from list that can not be used
kfile=1;
while kfile <=length(files)
    fname = files(kfile).name; % filename
    if length(fname)>4 % skip files to small to have .mat extention
        if strcmp(fname(end-3:end),'.mat') % check if the file of .mat type
            % Get sensor locations
            dummy=fname(find(fname=='s',1)+1:find(fname=='.',1)-1);
            dummy(dummy==',')=' ';
            locS{kfile}=str2num(dummy); %#ok<,ST2NM>
            kfile=kfile+1;
        else
            files(kfile)=[]; % remove entry
        end
    else
        files(kfile)=[]; % remove entry
    end
end

% Create vector of locations corresponding
locScheck=cell2mat(locS);

% Get size of system
sizeS=max(locScheck);

% Check for duplicates
if any(histcounts(locScheck)>1)
    error('The vibration data has duplicates')
end

% Check for missing data
if sizeS~=length(locScheck)
    error('The vibration data is incomplete')
end

% Get data and process
% Get fnew vector
load([fileDir '/Vib/' files(1).name]); % Load data
fs=VibGlob.acq.VIB.Fs;
nyfreq = fs/2;
```


©MECAL 2018 Confidential

```
df = nyfreq /(size(VibGlob.x,1)/2);
fnew = df:df:nyfreq;
% Preallocate data
Vib=complex(zeros(sizeS,length(fnew)));

% loop over all files
for kfile = 1:length(files)
    fname = files(kfile).name; % filename
    load([fileDir '/Vib/' fname]); % Load data

    % Get fs or check if it is the same as the others
    if fs~=VibGlob.acq.VIB.Fs;
        error('Sample frequencies between vibration experiments not equal');
    end

    % Get data

    % Sensor
    for i=1:length(locS{kfile})
        switch i
            case 1
                siganlSensor=VibGlob.x;
            case 2
                siganlSensor=VibGlob.y;
            case 3
                siganlSensor=VibGlob.z;
            otherwise
                error('Requested sensor data does not exist.')
        end

        % Inverse data if sensor was reversed
        if ismember(i,sReversed)
            siganlSensor=-siganlSensor;
        end

        % Sensor meta data
        chans=VibGlob.chan(i+1);

        % Get data
        [Vib(locS{kfile}(i),:),fnew] = data2Vib(siganlSensor,chans,fs,0);


        % Check if the sample frequency is the same
        if fnew(end)~=f(end)
            error('Sample frequencies between stiffness and vibration experiments not the same')
        end

    end
end

% Adjust results to have the same frequency range
```


©MECAL 2018 Confidential

```
if length(f)~=length(fnew)
    if length(fnew)>length(f)
        factor=length(fnew)/length(f);
        dummy=complex(zeros(size(Vib,1),length(f)));
        for i=1:size(frf,1)
            dummy(i,:)=Vib(i,factor:factor:end);
        end
        Vib=dummy;
    else
        factor=length(f)/length(fnew);
        f=fnew;
        dummy=complex(zeros(size(frf,1),size(frf,2),length(f)));
        for i=1:size(frf,1)
            for j=size(frf,2)
                dummy(i,j,:)=frf(i,j,factor:factor:end);
            end
        end
        frf=dummy;
    end
end

% convert vibration levels from displacement to forces
VibF=complex(zeros(size(Vib)));
for i=1:size(Vib,2)
    VibF(:,i)=frf(:,:,i)\Vib(:,i);
end

% Save data
if nargin==7
    save(matFile,'frf','f','Coh','VibF','X','Y');
end
end
```

specTest2frf

```
function [frf,f,Coh,X,Y] =
specTest2frf(fileDir,sReversed,CohTol,CohTolRange,floorSize,gridSize,matFile)
% [frf,f,Coh,X,Y]= specTest2frf(fileDir,sReversed,CohTol,CohTolRange,floorSize,gridSize,matFile)
%   Function for converting specTest experiments into frf's. Give the
%   folder fileDir where the specTest data is stored. SpecTest data
%   filenames should be in the form 'F(%s)_s(%s),(%s),(%s).mat' where (%s)
%   are numbers for the in and outputs. This function can only use 1 input
%   (F) and 1:3 outputs (s). If 1 or more of the outputs are inverted this
%   can be adjusted by giving the sensor number in sReversed. If this is
%   not needed pass an empty array ([]). The data is checked for coherence
%   is above CohTol in the CohTolRange. If this is not the case the
%   function will still run but give an warning. A floor grid is made from
%   the floor and grid size [x y]. The frf's can be stored to mat-file
%   matFile.
%   1-6-2018 fkoo
% Version 2 to handle new variable formats 26-6-2018 fkoo
```

```
% Update 11-7-2018 fkoo to handle variable number (1:3) sensor inputs

% Function input check
if nargin<6
    error('Not enough inputs for specTest2frf')
elseif nargin>7
    error('To many inputs for specTest2frf')
end

% Get locations and error checking
files = dir(fileDir); % read all files in dir

% Check in and output size and if all data is there and no duplicated exist

% Get F and s locations and remove file from list that can not be used
kfile=1;
while kfile <=length(files)
    fname = files(kfile).name; % filename
    if length(fname)>4 % skip files to small to have .mat extention
        if strcmp(fname(end-3:end),'.mat') % check if the file of .mat type
            % Get hammer and sensor locations

            dummy=fname(find(fname=='s',1)+1:find(fname=='.',1)-1);
            dummy(dummy==',')=' ';
            locS{kfile}=str2num(dummy); %#ok<,ST2NM>
            locF{kfile}=repmat(str2double(fname(2:find(fname=='_',1)-1)),1,length(locS{kfile}));
%#ok<*AGROW>
            kfile=kfile+1;
        else
            files(kfile)=[]; % remove entry
        end
    else
        files(kfile)=[]; % remove entry
    end
end

% Create vector of locations corresponding
locFcheck=cell2mat(locF);
locScheck=cell2mat(locS);

% Get size of system
sizeF=max(locFcheck);
sizeS=max(locScheck);

% Check if size is the same for inputs and outputs
if sizeF~=sizeS
    error('In and outputs do not have same size.')
end

% Check for duplicates
checkDuplicate=zeros(sizeF); % Preallocate
for i=1:length(locFcheck)
```


©MECAL 2018 Confidential

```
    if checkDuplicate(locFcheck(i),locScheck(i))~=0
        error(['Input ' num2str(locFcheck(i)) ' to output ' num2str(locScheck(i)) ' has
duplicates.'])
    end
    checkDuplicate(locFcheck(i),locScheck(i))=1;
end

% Check for missing data
if any(any(~checkDuplicate))
    errorText='Not data for all transfer is known. Data is missing for:\n';
    for i=1:sizeF
        for j=1:sizeF
            if checkDuplicate(i,j)==0
                errorText=[errorText num2str(i) ' to ' num2str(j) '\n'];
            end
        end
    end
    error(sprintf(errorText)) %#ok<*SPERR>
end

% Get data and process
% Get f vector
load([fileDir '/' files(1).name]); % Load data
fs=StiffGlob.acq.STIFF.Fs;
nyfreq = fs/2;
df = nyfreq /(size(StiffGlob.meas(2).Hamt,1)/2);
f = df:df:nyfreq;

% Preallocate data
frf=complex(zeros(sizeF,sizeF,length(f)));
Coh=frf;


% loop over all files
for kfile = 1:length(files)
    fname = files(kfile).name; % filename
    load([fileDir '/' fname]); % Load data

    % check if fs is the same as the others
    if fs~=StiffGlob.acq.STIFF.Fs;
        error('Sample frequencies between Stiffness experiments not equal');
    end

    % Get data
    % Hammer
    siganlHammer=zeros(length(StiffGlob.meas(1).Hamt),length(StiffGlob.chosen));
    for i=1:length(StiffGlob.chosen)
        siganlHammer(:,i)=StiffGlob.meas(i+1).Hamt*StiffGlob.chosen(i);
    end
    siganlHammer(:,~StiffGlob.chosen)=[]; % remove not chosen
    chanh=StiffGlob.chan(1);
```


©MECAL 2018 Confidential

```
    % Sensor
    for i=1:length(locS{kfile})
        siganlSensor=zeros(length(StiffGlob.meas(1).Hamt),length(StiffGlob.chosen));
        for j=1:length(StiffGlob.chosen)
            switch i
                case 1
                    siganlSensor(:,j)=StiffGlob.meas(j+1).xt*StiffGlob.chosen(j);
                case 2
                    siganlSensor(:,j)=StiffGlob.meas(j+1).yt*StiffGlob.chosen(j);
                case 3
                    siganlSensor(:,j)=StiffGlob.meas(j+1).zt*StiffGlob.chosen(j);
                otherwise
                    error('Requested sensor data does not exist.')
            end
        end
        siganlSensor(:,~StiffGlob.chosen)=[];

        % Inverse data if sensor was reversed
        if ismember(i,sReversed)
            siganlSensor=-siganlSensor;
        end

        chans=StiffGlob.chan(i+1);

        % Get transfer
        [f,frf(locS{kfile}(i),locF{kfile}(i),:),Coh(locS{kfile}(i),locF{kfile}(i),:)] =
data2frf(siganlHammer,chanh,siganlSensor,chans,fs,0);

        % Get f Coh range
        CohTolRangei=[find(f>=CohTolRange(1),1) find(f>=CohTolRange(2),1)];

        % check Coh
        if mean(squeeze(Coh(locS{kfile}(i),locF{kfile}(i),CohTolRangei(1):CohTolRangei(2))))<CohTol
            warning(['Experiment data for the transfer from ' num2str(locF{kfile}(i)) ' to '
num2str(locS{kfile}(i)) ' has low coherence.'])
        end

    end
end

% Get floor grid
[X,Y]=meshgrid(linspace(0,floorSize(1),gridSize(1)),linspace(0,floorSize(2),gridSize(2)));

% Save data
if nargin==7
    save(matFile,'frf','f','Coh','X','Y');
end
end
```

data2frf

```
function [f,frf,Coh] = data2frf(hammer_data,chanh, sensor_data,chans, fs,window)
% [f,cStiff,Coh] = data2frf(hammer_data,chanh, sensor_data,chans, fs,window)
%  Function for specTest2frf based on the function StiffAvgAndCoh by sban.

% Get number of samples n and number of experiments m
[n,m] = size(hammer_data);


% calculate individual fft's & average
% Preallocated fft data
ft_ham = zeros (n/2,m);
ft_sens = zeros (n/2,m);

% run first experiment data
i =1;
[~,ft_ham(:,i)] = fft_cspectrum(hammer_data(:,i), fs, window);
[f,ft_sens(:,i)] = fft_cspectrum(sensor_data(:,i), fs, window);
unith = chanh.unit;
ft_ham(:,i) = ft_ham(:,i).*((1i*2*pi*f).^unith);
units = chans.unit;
ft_sens(:,i) = ft_sens(:,i).*((1i*2*pi*f).^units);

% Get adjustment for hammer sensor frf
if chanh.FOLink > 0
  FOLink = chanh.FOLink;
  QLink = chanh.QLink;
  geocorrh = (1-f.^2/FOLink^2+1i*f/(FOLink*QLink))./(f.^2/FOLink^2);
else
  geocorrh = ones(size(f));
end

% Get adjustment for sensors frf
if chans.FOLink > 0
  FOLink = chans.FOLink;
  QLink = chans.QLink;
  geocorrs = (1-f.^2/FOLink^2+1i*f/(FOLink*QLink))./(f.^2/FOLink^2);
else
  geocorrs = ones(size(f));
end

% Correct signal
ft_ham(:,i) = ft_ham(:,i).*geocorrh;
ft_sens(:,i) = ft_sens(:,i).*geocorrs;
% do the same for the rest of the experiment data
for i = 2:m
  [~,ft_ham(:,i)] = fft_cspectrum(hammer_data(:,i), fs, window);
  [~,ft_sens(:,i)] = fft_cspectrum(sensor_data(:,i), fs, window);
  ft_ham(:,i) = ft_ham(:,i).*((1i*2*pi*f).^unith);  %r5
  ft_sens(:,i) = ft_sens(:,i).*((1i*2*pi*f).^units); %r5
  ft_ham(:,i) = ft_ham(:,i).*geocorrh;
  ft_sens(:,i) = ft_sens(:,i).*geocorrs;
```

```
end


frf=(sum(ft_sens,2)./sum(ft_ham,2))./(-2*pi.*f).^2 ;

% calculate coherence

part1 = sum(ft_sens.*conj(ft_ham) ,2);
part2 = sum( ft_ham.*conj(ft_ham) , 2);
part3 = sum( ft_ham.*conj(ft_sens) , 2);
part4 = sum(ft_sens.*conj(ft_sens) ,2);

nom = part1 .* part3;
denom = part2 .* part4;
Coh = nom ./ denom;

% End of function
```

data2Vib

```
function [Vib,f] = data2Vib(sensor_data,chans, fs,window)
% Vib = data2Vib(sensor_data,chans, fs,window)
%  Function for stiffAndVib2data based on the function StiffAvgAndCoh by sban.

% Get number of samples n and number of experiments m
[n,m] = size(sensor_data);


% calculate individual fft's & average
% Preallocated fft data
ft_sens = zeros (n/2,m);

% run first experiment data
i =1;
[f,ft_sens(:,i)] = fft_cspectrum(sensor_data(:,i), fs, window);
units = chans.unit;
ft_sens(:,i) = ft_sens(:,i).*((1i*2*pi*f).^units);

% Get adjustment for sensors frf
if chans.FOLink > 0
  FOLink = chans.FOLink;
  QLink = chans.QLink;
  geocorrs = (1-f.^2/FOLink^2+1i*f/(FOLink*QLink))./(f.^2/FOLink^2);
else
  geocorrs = ones(size(f));
end

% Correct signal
ft_sens(:,i) = ft_sens(:,i).*geocorrs;
% do the same for the rest of the experiment data
for i = 2:m
```

```
  [~,ft_sens(:,i)] = fft_cspectrum(sensor_data(:,i), fs, window);
  ft_sens(:,i) = ft_sens(:,i).*((1i*2*pi*f).^units); %r5
  ft_sens(:,i) = ft_sens(:,i).*geocorrs;
end


Vib=sum(ft_sens,2)./(-2*pi.*f).^2 ;


% End of function
```

fft_cspectrum

```
% function [f cspect] = fft_cspectrum(data, fs, window)
function [f cspect varargout] = fft_cspectrum(data, fs,window)
% calculates the complex fourierspectrum of data for
% freq's > 0
% inputs
%      data: array(column) of real data representing a time signal
%         preferrably the length of the array is a power of 2
%      fs:  sample frequency [S/s] applicable to the data array
%      window: marker if a (hanning) window must be applied: 0 = no
%      window, 1 = use a hanning window
%
% outputs
%      f :  array containing the positive frequencies for the
%          spectrum
%      cspect: complex spectrum scaled as an amplitude spectrum for
%      the positive frequencies
%
% sban sept 2007
% sban added correction for offset and linear trend
% sban 2015 added comment
% sban 2017 warning on non power of 2 removed


% check data range is a column and not a row (convention)
[n m] = size(data);
if ~(m ==1)
   disp('fft_cspectrum > data is not of format [n,1]');
end

% apply window to the data
if window ==1 % a hanning window must be applied
  % correction for offset and linear trend only when a window is applied
  x = 1:n;x = x';
  p = polyfit(x,data,1); % fit to function y =p(1)*x+p(2)
  data = data-polyval(p,x);

  t = [0: 2*pi/(n-1):2*pi]';
  data = data .* (0.5-0.5*cos(t));
```


©MECAL 2018 Confidential

```
  % mark the following !
  % for corect amplitude's apply factor *2 on amplitudes
  % for correct power estimation (eg psd conversion)
  % apply *sqrt(8/3) on amplitude (or 8/3 on psd)
  % this is left to the user!
end

% make the freq vector
nyfreq = fs/2;   % the highest freq is the nyquist freq = fs/2
df = nyfreq /(n/2);
f = [df:df:nyfreq]';   % do not use freq = 0

% Now the fft
fft_raw = fft(data);
cspect = fft_raw(2:n/2+1)*2/n;
cspect(end) = cspect(end)/2; % highest freq is differently scaled
DC_value = fft_raw(1)/n;
if nargout >= 3
  varargout{1} = DC_value
end

% % testcode for fft_cspectrum
% % first without window
% fs = 4096;
% t = [1/fs:1/fs:1]';
% y = sin(2*pi*50*t);   % 50 sinus's fit in the signal
% [f A] = fft_cspectrum(y,fs,0);
% plot(f,abs(A))
% % 50 Hz amplitude must be 1
% disp (['f(50) = ' num2str(f(50))]);
% disp(['A(50) = ' num2str(A(50))]);
% disp(['abs(A(50)) = ' num2str(abs(A(50)))]);
% % idem with window
% [f Aw] = fft_cspectrum(y,fs,1);
% Aw_corr_for_A = Aw*2;
% % 50 Hz amplitude must be 1
% disp (['f(50) = ' num2str(f(50))]);
% disp(['A with window amp corr.(50) = ' num2str(Aw_corr_for_A(50))]);
% % SBAN jan 2008/ 2015
```

LoadWorkSpace

```
function LoadWorkSpace(~,~)
% Function for workspace to globals
global h glo_frf glo_f glo_Coh glo_VibF glo_X glo_Y glo_floor glo_EQ

% Get workspace variable names
frfName=h.loadScreen_frfName.String;
fName=h.loadScreen_fName.String;
CohName=h.loadScreen_CohName.String;
VibFName=h.loadScreen_VibFName.String;
```


©MECAL 2018 Confidential

```
XgridName=h.XgridName.String;
YgridName=h.YgridName.String;

% Start with a empty global floor
glo_floor=[];
glo_EQ=[];

h.text.massPoints='';
h.text.mass='';
h.text.frameLoc1='';
h.text.frameLoc2='';
h.text.frameMass='';
h.text.frameStiff='';
h.text.stiffPoint='';
h.text.stiffPointValue='';

% try to load workspace variable to globals
try
  glo_frf = evalin('base',frfName);
  glo_f = evalin('base',fName);
  glo_Coh = evalin('base',CohName);
  glo_VibF = evalin('base',VibFName);
  glo_X = evalin('base',XgridName);
  glo_Y = evalin('base',YgridName);

  % if the try got to here the load was succesfull
  consoleWrite('Workspace variables loaded.');

  % clear load screen away
  delete(h.loadScreen)

  % load main menu
  buildMainMenu

catch me
  % Write in console that the load failed and why
  consoleWrite(['Failed to load experiment due to: ' me.message]);
end
end
```

buildMainMenu

```
function buildMainMenu
global h glo_frf glo_f glo_EQ

% main menu screen
h.mainMenu = uipanel('Parent',h.mainScreen,'Title','Main Menu','Position',[0 0 0.8 1]);

% floor plan plot screen without added elements
h.plan = uipanel('Parent',h.mainMenu,'Title','Floor plan','Position',[0 0 1 0.5]);
h.planAxes = axes('Parent',h.plan);
```

```
buildPlan % Plot floor plan


% Plot transfer part
h.plotTranferScreen=uipanel('Parent',h.mainMenu,'Title','Plot transfer','Position',[0 0.85 0.5
0.13]);
h.plotTransferText0 = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0.34 0.6 0.33],'string','Select in and output(s)',...
    'HorizontalAlignment','left');
h.plotTransferText0b = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0.3 0.34 0.6 0.33],'string','Maxium transfer range',...
    'HorizontalAlignment','left');
h.maxTransferRange = uicontrol('Parent',h.plotTranferScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.5 0.34 0.09 0.33],...
    'String','10 100','Callback',@editMaxTransfer);

maxTransferLoc = maxTransfer(glo_frf,glo_f,[10 100]);

h.plotTransferText1 = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0.68 0.6 0.32],...
    'string',['Maxium transfer of the unaltered system is z' num2str(maxTransferLoc(1)) '/F'
num2str(maxTransferLoc(2))],...
    'HorizontalAlignment','left');
h.plotTransferText1b = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0.5 0.68 0.6 0.32],...
    'string',['Maxium transfer of the altered system is z' num2str(maxTransferLoc(1)) '/F'
num2str(maxTransferLoc(2))],...
    'HorizontalAlignment','left');
h.plotTransferText2 = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 0.1 0.33],'string','Inputs',...
    'HorizontalAlignment','left');
h.plotTransferF = uicontrol('Parent',h.plotTranferScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.07 0 0.2 0.33]);
h.plotTransferText3 = uicontrol('Parent',h.plotTranferScreen,...
    'style','text','Units','normalized',...
    'Position', [0.28 0 0.1 0.33],'string','Outputs',...
    'HorizontalAlignment','left');
h.plotTransferS = uicontrol('Parent',h.plotTranferScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.39 0 0.2 0.33]);
h.plotTransfer = uicontrol('Parent',h.plotTranferScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.6 0 0.2 0.33],'string','Plot transfer(s)',...
    'Callback',@plotTransfer);
h.plotTransferCo = uicontrol('Parent',h.plotTranferScreen,...
```


©MECAL 2018 Confidential

```
    'style','pushbutton','Units','normalized',...
    'Position', [0.8 0 0.2 0.33],'string','Plot Colocated transfer',...
    'Callback',@plotTransferCo);
h.plotTransferSelctor = uicontrol('Parent',h.plotTranferScreen,...
    'style','popup','String',{'z','v','a'},...
    'Units','normalized','Position', [0.6 0.34 0.1 0.33]);

% vibration forces
h.vibFScreen=uipanel('Parent',h.mainMenu,'Title','Vibrations forces','Position',[0.5 0.94 0.5
0.04]);
h.vibFBut = uicontrol('Parent',h.vibFScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0 0 0.3 1],'string','Plot Vibrations forces',...
    'Callback',@plotVibF);

% vibration
h.vibScreen=uipanel('Parent',h.mainMenu,'Title','Vibrations','Position',[0.5 0.895 0.5 0.04]);
uicontrol('Parent',h.vibScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 .1 1],'String','Location(s)',...
    'HorizontalAlignment','left');
h.vibLoc = uicontrol('Parent',h.vibScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.1 0 0.5 1]);
h.vibType = uicontrol('Parent',h.vibScreen,...
    'style','popup','Units','normalized','String',{'z','v','a'},...
    'Position', [0.65 0 0.1 1]);
h.vib = uicontrol('Parent',h.vibScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.8 0 0.15 1],'string','Plot Vibrations',...
    'Callback',@plotVib);

% Sensitivity
h.sensitivityScreen=uipanel('Parent',h.mainMenu,'Title','Sensitivity','Position',[0.5 0.85 0.5
0.04]);
uicontrol('Parent',h.sensitivityScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 .2 1],'String','Inputs',...
    'HorizontalAlignment','left');
h.sensitivityF = uicontrol('Parent',h.sensitivityScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.07 0 0.3 1]);
uicontrol('Parent',h.sensitivityScreen,...
    'style','text','Units','normalized',...
    'Position', [0.38 0 .2 1],'String','Outputs',...
    'HorizontalAlignment','left');
h.sensitivityS = uicontrol('Parent',h.sensitivityScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.47 0 0.3 1]);
h.sensitivity = uicontrol('Parent',h.sensitivityScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.8 0 0.15 1],'string','Plot Sensitivity',...
```

```
    'Callback',@plotSensitivity);

%ODS (2x)
for i=1:2
    if i==1
        h.ODSScreen(i)=uipanel('Parent',h.mainMenu,'Title','ODS without extra
elements','Position',[0 0.8 0.5 0.04]);
    else
        h.ODSScreen(i)=uipanel('Parent',h.mainMenu,'Title','ODS with extra
elements','Position',[0.5 0.8 0.5 0.04]);
    end

    h.ODSText2(i) = uicontrol('Parent',h.ODSScreen(i),...
        'style','text','Units','normalized',...
        'Position', [.01 0 0.17 1],...
        'string','Force points [-]',...
        'HorizontalAlignment','left');
    h.ODSF(i) = uicontrol('Parent',h.ODSScreen(i),...
        'style','edit','Units','normalized',...
        'Position', [0.18 0 0.2 1]);
    h.ODSText3(i) = uicontrol('Parent',h.ODSScreen(i),...
        'style','text','Units','normalized',...
        'Position', [0.39 0 0.21 1],...
        'string','Force amplitude [N]',...
        'HorizontalAlignment','left');
    h.ODSFa(i) = uicontrol('Parent',h.ODSScreen(i),...
        'style','edit','Units','normalized',...
        'Position', [0.6 0 0.2 1]);
    h.runODS(i) = uicontrol('Parent',h.ODSScreen(i),...
        'style','pushbutton','Units','normalized',...
        'Position', [0.81 0 0.2 1],'string','Run ODS',...
        'Tag',num2str(i),'Callback',@runODS);
end

%EQ ADD
h.EQScreen=uipanel('Parent',h.mainMenu,'Title','EQ locations','Position',[0 0.75 0.5 0.04]);
h.EQLoc = uicontrol('Parent',h.EQScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 .3 1],'String','No EQs');
if ~isempty(glo_EQ)
    EQpoints='Equalizers placed on: ';
    for i=1:length(glo_EQ)
        EQpoints=[EQpoints num2str(glo_EQ(i).point) ', ']; %#ok<AGROW>
    end
    h.EQLoc.String=EQpoints(1:end-2);
end
h.addEQ = uicontrol('Parent',h.EQScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.3 0 0.2 1],'string','Add EQ',...
    'Tag',num2str(i),'Callback',@addEQ);
h.removeEQedit = uicontrol('Parent',h.EQScreen,...
    'style','edit','Units','normalized',...
```

```
    'Position', [0.55 0 0.2 1]);
h.removeEQ = uicontrol('Parent',h.EQScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.78 0 0.2 1],'string','Remove EQ',...
    'Tag',num2str(i),'Callback',@removeEQ);

%EQ force estimation
h.EQScreenF=uipanel('Parent',h.mainMenu,'Title','EQ force estimation','Position',[0.5 0.75 0.5
0.04]);
uicontrol('Parent',h.EQScreenF,...
    'style','text','Units','normalized',...
    'Position', [0 0 .2 1],'String','Plot range',...
    'HorizontalAlignment','left');
h.EQFrange = uicontrol('Parent',h.EQScreenF,...
    'style','edit','Units','normalized',...
    'Position', [0.2 0 0.28 1],'String','10 100');
h.getEQF = uicontrol('Parent',h.EQScreenF,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.5 0 0.2 1],'string','Calculate EQ forces',...
    'Tag',num2str(i),'Callback',@getEQF);

%MASS ADD
h.massScreen=uipanel('Parent',h.mainMenu,'Title','Point masses','Position',[0 0.65 0.5 0.08]);
h.massText1 = uicontrol('Parent',h.massScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0.5 0.2 0.5],...
    'string','Point mass locations',...
    'HorizontalAlignment','left');
h.massLoc = uicontrol('Parent',h.massScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.2 0.5 0.8 0.5],'Callback',@editPlan2newElements,'String',h.text.massPoints);
h.massText2 = uicontrol('Parent',h.massScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 0.2 0.5],...
    'string','Point mass [kg]',...
    'HorizontalAlignment','left');
h.mass = uicontrol('Parent',h.massScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.2 0 0.8 0.5],'Callback',@editPlan2newElements,'String',h.text.mass);

%Frame add
h.frameScreen=uipanel('Parent',h.mainMenu,'Title','Frames','Position',[0 0.55 0.5 0.08]);
h.frameText1 = uicontrol('Parent',h.frameScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0.5 0.2 0.5],...
    'string','Point 1',...
    'HorizontalAlignment','left');
h.frameLoc1 = uicontrol('Parent',h.frameScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.07 0.5 0.4 0.5],'Callback',@editPlan2newElements,'String',h.text.frameLoc1);
h.frameText2 = uicontrol('Parent',h.frameScreen,...
    'style','text','Units','normalized',...
```

```
    'Position', [0 0 0.2 0.5],...
    'string','Point 2',...
    'HorizontalAlignment','left');
h.frameLoc2 = uicontrol('Parent',h.frameScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.07 0 0.4 0.5],'Callback',@editPlan2newElements,'String',h.text.frameLoc2);
h.frameText3 = uicontrol('Parent',h.frameScreen,...
    'style','text','Units','normalized',...
    'Position', [0.5 0.5 0.2 0.5],...
    'string','Mass',...
    'HorizontalAlignment','left');
h.frameMass = uicontrol('Parent',h.frameScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.6 0.5 0.4 0.5],'Callback',@editPlan2newElements,'String',h.text.frameMass);
h.frameText4 = uicontrol('Parent',h.frameScreen,...
    'style','text','Units','normalized',...
    'Position', [0.5 0 0.2 0.5],...
    'string','Stiffness',...
    'HorizontalAlignment','left');
h.frameStiff = uicontrol('Parent',h.frameScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.6 0 0.4 0.5],'Callback',@editPlan2newElements,'String',h.text.frameStiff);

% stiffness screen
h.stiffScreen=uipanel('Parent',h.mainMenu,'Title','Point stiffness','Position',[0 0.50 0.5 0.04]);
uicontrol('Parent',h.stiffScreen,...
    'style','text','Units','normalized',...
    'Position', [0 0 0.2 1],...
    'string','Point',...
    'HorizontalAlignment','left');
h.stiffPoint = uicontrol('Parent',h.stiffScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.07 0 0.4 1],'Callback',@editPlan2newElements,'String',h.text.stiffPoint);
uicontrol('Parent',h.stiffScreen,...
    'style','text','Units','normalized',...
    'Position', [0.5 0 0.2 1],...
    'string','Stiffness',...
    'HorizontalAlignment','left');
h.stiffPointValue = uicontrol('Parent',h.stiffScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.6 0 0.4 1],'Callback',@editPlan2newElements,'String',h.text.stiffPointValue);

%Draw tool
h.DrawToolScreen=uipanel('Parent',h.mainMenu,'Title','Draw tool','Position',[0.5 0.65 0.5 0.08]); %
to be deleted
source.Value=1;
setDrawTool(source); % Build draw tool

%save screen
h.saveScreen=uipanel('Parent',h.mainMenu,'Title','Save','Position',[0.5 0.55 0.5 0.04]);
h.saveText = uicontrol('Parent',h.saveScreen,...
    'style','text','Units','normalized',...
```


©MECAL 2018 Confidential

```
    'Position', [0 0 0.2 1],...
    'string','File name',...
    'HorizontalAlignment','left');
h.saveFileName = uicontrol('Parent',h.saveScreen,...
    'style','edit','Units','normalized',...
    'Position', [0.1 0 0.4 1]);
h.runSave = uicontrol('Parent',h.saveScreen,...
    'style','pushbutton','Units','normalized',...
    'Position', [0.51 0 0.2 1],'string','Save',...
    'Callback',@runSave);

% plot floor plan with added elements and update maxiumum transfers
editPlan2newElements;
end
```

buildPlan

```
function buildPlan
% function for plotting the bareback floor plan
global glo_X glo_Y glo_floor

% plot nodes
X=glo_X';
Y=glo_Y';
text(X(:),Y(:),num2cell(1:length(X(:))));
hold on

% smallest and largest values in plot
maxVal=zeros(1,2);
minVal=maxVal;

% plot floor drawing if exist
if ~isempty(glo_floor)
  % plot all floor drawings
  for i=1:length(glo_floor)
    % check if text or other drawing
    if isempty(glo_floor(i).text)
      % plot other style drawing
      P=plot(glo_floor(i).x,glo_floor(i).y,'k');
      % make sure it is not a legend entry
      P.Annotation.LegendInformation.IconDisplayStyle = 'off';
    else
      % plot text
      text(glo_floor(i).x,glo_floor(i).y,glo_floor(i).text);
    end
    % get largest and smallest value of the floor drawing part
    maxVal(1)=max([maxVal(1) glo_floor(i).x]);
    maxVal(2)=max([maxVal(2) glo_floor(i).y]);
    minVal(1)=min([minVal(1) glo_floor(i).x]);
    minVal(2)=min([minVal(2) glo_floor(i).y]);
  end
```


©MECAL 2018 Confidential

```
end

% get smallest and largest values in plot and use it to have the correct
% scaling in plot
maxVal(1)=max([maxVal(1) max(max(glo_X))]);
maxVal(2)=max([maxVal(2) max(max(glo_Y))]);
minVal(1)=min([minVal(1) min(min(glo_X))]);
minVal(2)=min([minVal(2) min(min(glo_Y))]);
axis equal
% extra space on x axes is for legend
xlim([minVal(1)-0.1*(maxVal(1)-minVal(1)) maxVal(1)+0.4*(maxVal(1)-minVal(1))])
ylim([minVal(2)-0.1*(maxVal(2)-minVal(2)) maxVal(2)+0.1*(maxVal(2)-minVal(2))])

% set labels
xlabel('Position X [m]')
ylabel('Position Y [m]')
end
```

setDrawTool

```
function setDrawTool(source,~)
global h
% Function for setting draw tool screen

% get source
val=source.Value;

% clear old load screen
delete(h.DrawToolScreen)

% Build new load screen
h.DrawToolScreen=uipanel('Parent',h.mainMenu,'Title','Draw tool','Position',[0.5 0.65 0.5 0.08]);
h.DrawToolUI = uicontrol('Parent',h.DrawToolScreen,'Style', 'popup',...
  'String', {'Rectangle','Circle','Text'},...
  'Units','normalized','Position', [0 0.5 0.18 0.5],...
  'Callback', @setDrawTool);
h.clearFloor = uicontrol('Parent',h.DrawToolScreen,...
  'style','pushbutton','Units','normalized',...
  'Position', [0 0 0.18 0.5],'string','Clear floor',...
  'Callback',@ClearFloor);
h.deleteFloorParts = uicontrol('Parent',h.DrawToolScreen,...
  'style','pushbutton','Units','normalized',...
  'Position', [0.7 0 0.2 0.5],'string','Delete floor parts',...
  'Callback',@deleteFloorParts);
h.freeDraw = uicontrol('Parent',h.DrawToolScreen,...
  'style','pushbutton','Units','normalized',...
  'Position', [0.7 0.5 0.2 0.5],'string','Free draw tool',...
  'Callback',@addDraw);

% Set dropdown to chosen tool
h.DrawToolUI.Value=val;
```

```
% build screen to chosen tool
switch val
  case 1
    h.DrawToolText1 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.2 0.5 0.1 0.5],...
      'string','x mid',...
      'HorizontalAlignment','left');
    h.DrawToolX = uicontrol('Parent',h.DrawToolScreen,...
      'style','edit','Units','normalized',...
      'Position', [0.25 0.5 0.1 0.5]);
    h.DrawToolText2 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.2 0 0.1 0.5],...
      'string','y mid',...
      'HorizontalAlignment','left');
    h.DrawToolY = uicontrol('Parent',h.DrawToolScreen,...
      'style','edit','Units','normalized',...
      'Position', [0.25 0 0.1 0.5]);
    h.DrawToolText3 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.36 0.5 0.1 0.5],...
      'string','Width',...
      'HorizontalAlignment','left');
    h.DrawToolWidth = uicontrol('Parent',h.DrawToolScreen,...
      'style','edit','Units','normalized',...
      'Position', [0.42 0.5 0.1 0.5]);
    h.DrawToolText4 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.36 0 0.1 0.5],...
      'string','Length',...
      'HorizontalAlignment','left');
    h.DrawToolLength = uicontrol('Parent',h.DrawToolScreen,...
      'style','edit','Units','normalized',...
      'Position', [0.42 0 0.1 0.5]);
    h.addRectangle = uicontrol('Parent',h.DrawToolScreen,...
      'style','pushbutton','Units','normalized',...
      'Position', [0.53 0 0.15 0.5],'string','Add rectangle',...
      'Callback',@addRectangle);
  case 2
    h.DrawToolText1 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.2 0.5 0.1 0.5],...
      'string','x mid',...
      'HorizontalAlignment','left');
    h.DrawToolX = uicontrol('Parent',h.DrawToolScreen,...
      'style','edit','Units','normalized',...
      'Position', [0.25 0.5 0.1 0.5]);
    h.DrawToolText2 = uicontrol('Parent',h.DrawToolScreen,...
      'style','text','Units','normalized',...
      'Position', [0.2 0 0.1 0.5],...
```

```
        'string','y mid',...
        'HorizontalAlignment','left');
      h.DrawToolY = uicontrol('Parent',h.DrawToolScreen,...
        'style','edit','Units','normalized',...
        'Position', [0.25 0 0.1 0.5]);
      h.DrawToolText3 = uicontrol('Parent',h.DrawToolScreen,...
        'style','text','Units','normalized',...
        'Position', [0.36 0.5 0.1 0.5],...
        'string','Radius',...
        'HorizontalAlignment','left');
      h.DrawToolRadius = uicontrol('Parent',h.DrawToolScreen,...
        'style','edit','Units','normalized',...
        'Position', [0.42 0.5 0.1 0.5]);
      h.addCircle = uicontrol('Parent',h.DrawToolScreen,...
        'style','pushbutton','Units','normalized',...
        'Position', [0.53 0 0.15 0.5],'string','Add circle',...
        'Callback',@addCircle);
    case 3
      h.DrawToolText1 = uicontrol('Parent',h.DrawToolScreen,...
        'style','text','Units','normalized',...
        'Position', [0.2 0.5 0.1 0.5],...
        'string','x 1',...
        'HorizontalAlignment','left');
      h.DrawToolX = uicontrol('Parent',h.DrawToolScreen,...
        'style','edit','Units','normalized',...
        'Position', [0.25 0.5 0.1 0.5]);
      h.DrawToolText2 = uicontrol('Parent',h.DrawToolScreen,...
        'style','text','Units','normalized',...
        'Position', [0.2 0 0.1 0.5],...
        'string','y b',...
        'HorizontalAlignment','left');
      h.DrawToolY = uicontrol('Parent',h.DrawToolScreen,...
        'style','edit','Units','normalized',...
        'Position', [0.25 0 0.1 0.5]);
      h.DrawToolText3 = uicontrol('Parent',h.DrawToolScreen,...
        'style','text','Units','normalized',...
        'Position', [0.36 0.5 0.1 0.5],...
        'string','Text',...
        'HorizontalAlignment','left');
      h.DrawToolText = uicontrol('Parent',h.DrawToolScreen,...
        'style','edit','Units','normalized',...
        'Position', [0.42 0.5 0.1 0.5]);
      h.addText = uicontrol('Parent',h.DrawToolScreen,...
        'style','pushbutton','Units','normalized',...
        'Position', [0.53 0 0.15 0.5],'string','Add text',...
        'Callback',@addText);
  end
  end
```

ClearFloor

```
function ClearFloor(~,~)
global glo_floor
% function for clearing the floor plan drawings

% clear global
glo_floor=[];

% redraw floor plan to new entry
editPlan;
end
```

deleteFloorParts

```
function deleteFloorParts(~,~)
global glo_floor
% function for deleting onlt part(s) of the floor plan drawing

% Only run if there are part to be able to delete
if ~isempty(glo_floor)
  % start new GUI figure screen with handle so it can be deleted
  p=figure('Units','normalized','position',[0.1 0.1 0.8 0.8]);

  % plot floor plan
  buildPlan;
  title(['Select items to be deleted (turns red), to undo use right'...
    ' mouse button, apply changes by use of middle mouse button. '...
    'For drawings click close to the middle and for text'...
    ' close on the left side.'])

  % preset loop
  button=1;

  % variable storing which floor elements should be deleted
  toBeDel=false(length(glo_floor),1);

  % loop till a middle mouse button is detected
  while button~=2
    % Get single user input
    [x,y,button]=ginput(1);

    % get distances from mouse click to drawings point
    dis=zeros(1,length(glo_floor));
    for i=1:length(glo_floor)
      dis(i)=norm([x-glo_floor(i).x y-glo_floor(i).y]);
    end

    if button==1&&any(~toBeDel)
      % find closest entry to be deleted
      i=find(dis==min(dis(~toBeDel)));

      % Make it red
```

```
      if isempty(glo_floor(i).text)
        plot(glo_floor(i).x,glo_floor(i).y,'r');
      else
        text(glo_floor(i).x,glo_floor(i).y,glo_floor(i).text,'Color','red');
      end

      % store it to be deleted
      toBeDel(i)=true;

    elseif button==3&&any(toBeDel)
      % find closest entry to not be longer deleted
      i=find(dis==min(dis(toBeDel)));

      % Make it black again
      if isempty(glo_floor(i).text)
        plot(glo_floor(i).x,glo_floor(i).y,'k');
      else
        text(glo_floor(i).x,glo_floor(i).y,glo_floor(i).text);
      end

      % store it to not be longer deleted
      toBeDel(i)=false;
    end
  end

  % close GUI
  close(p);

  % delete to be deleted entries
  glo_floor(toBeDel)=[];

  % redraw floor plan
  editPlan;
end
end
```

addDraw

```
function addDraw(~,~)
global h glo_floor
% function for adding a free drawings to floor plan

% Get floor index for this new entry
n=length(glo_floor)+1;

fig=figure('position',h.mainScreen.Position);
buildPlan;
title('Please draw lines. Finnish with the middle mouse button and restart with the right')

% preset loop
button=1;
```

```
x=[];
y=[];


while button~=2
  % Get single user input
  [xnew,ynew,button]=ginput(1);
  switch button
    case 1
      x(end+1)=xnew; %#ok<AGROW>
      y(end+1)=ynew; %#ok<AGROW>
      switch length(x)
        case 1
        case 2
          newLines=plot(x,y,'k');
        otherwise
          newLines.XData=x;
          newLines.YData=y;
      end
    case 3
      x=[];
      y=[];
      delete(newLines);
  end
end

% set floor parameters
glo_floor(n).x=x;
glo_floor(n).y=y;
glo_floor(n).text=[];

% exit figure
close(fig);

% redraw floor plan to new entry
editPlan;
end
```

addRectangle

```
function addRectangle(~,~)
global h glo_floor
% function for adding a rectangle to floor plan

% Get floor index for this new entry
n=length(glo_floor)+1;

% get rectangle parameters
xMid=str2double(h.DrawToolX.String);
yMid=str2double(h.DrawToolY.String);
floorWidth=str2double(h.DrawToolWidth.String);
```

```
floorLength=str2double(h.DrawToolLength.String);

% set floor parameters
glo_floor(n).x=floorWidth*[-0.5 0.5 0.5 -0.5 -.5]+xMid;
glo_floor(n).y=floorLength*[-0.5 -0.5 0.5 0.5 -0.5]+yMid;
glo_floor(n).text=[];

% redraw floor plan to new entry

editPlan;
end
```

addCircle

```
function addCircle(~,~)
global h glo_floor
% function for adding a circle to floor plan

% Get floor index for this new entry
n=length(glo_floor)+1;

% get circle parameters
xMid=str2double(h.DrawToolX.String);
yMid=str2double(h.DrawToolY.String);
FloorRadius=str2double(h.DrawToolRadius.String);

% draw loop variable
t=linspace(0,2*pi,30);

% set floor parameters
glo_floor(n).x=FloorRadius*cos(t)+xMid;
glo_floor(n).y=FloorRadius*sin(t)+yMid;
glo_floor(n).text=[];

% redraw floor plan to new entry

editPlan;
end
```

addText

```
function addText(~,~)
global h glo_floor
% function for adding a text to floor plan

% Get floor index for this new entry
n=length(glo_floor)+1;

% get text parameters
xMid=str2double(h.DrawToolX.String);
yMid=str2double(h.DrawToolY.String);
```

```
FloorText=h.DrawToolText.String;

% set floor parameters
glo_floor(n).x=xMid;
glo_floor(n).y=yMid;
glo_floor(n).text=FloorText;

% redraw floor plan to new entry
editPlan;
end
```

maxTransfer

```
function maxTransfer = maxTransfer(frf,f,frange)
% maxTransfer = maxTransfer(frf,f,frange)
%  Function for finding the worst transfer of all frf's over a range of f.
%  if no range is given the range is the complete range of the frf.
%  5-6-2018 fkoo
% Version 2 to handle new frf format 26-6-2018 fkoo

if nargin~=3
  frange=1:length(frf{1,1});
else
  frange=find(f>=frange(1),1):find(f>=frange(1),1);
end

% Get maximum value for all frf's
maxVal=max(max(max(abs(frf(:,:,frange)))));

% Find corresponding frf
[maxTransfer(1),maxTransfer(2)]=find(abs(frf)==maxVal);
maxTransfer(2)=mod(maxTransfer(2),size(frf,1));
end
```

editPlan2newElements

```
function editPlan2newElements(~,~)
% update floor plan to new elements
editPlan;

% update maximum transfer
frfAddedElements;
end
```

editPlan

```
function editPlan(~,~)
global h glo_X glo_Y glo_EQ
% Redraw of the plan
```

```
% get nodes
X=glo_X';
X=X(:);
Y=glo_Y';
Y=Y(:);

% Select the correct axes to draw on
axes(h.planAxes);

% clear axes
cla

% redraw bareback floor plan
buildPlan
hold on

% draw eq points
if ~isempty(glo_EQ)
  eq=[glo_EQ.point];
  plot(X(eq),Y(eq),'ob','MarkerSize',26)
end

% draw point masses
mass=str2num(h.massLoc.String); %#ok<ST2NM>
plot(X(mass),Y(mass),'or','MarkerSize',30)

% draw frames
frameLoc1=str2num(h.frameLoc1.String); %#ok<ST2NM>
frameLoc2=str2num(h.frameLoc2.String); %#ok<ST2NM>
plotFrame=[];
% only draw frames if the number of entries is the same for the locations
if length(frameLoc1)==length(frameLoc2)&&~isempty(frameLoc2)
  plot([X(frameLoc1) X(frameLoc2)],[Y(frameLoc1) Y(frameLoc2)],'b')
  plotFrame=1;
end

% draw stiffness points
stiffloc=str2num(h.stiffPoint.String); %#ok<ST2NM>
plot(X(stiffloc),Y(stiffloc),'om','MarkerSize',34)

% set legend if needed
if any([~isempty(glo_EQ) mass plotFrame])
  leg={'EQs','Masses','Frames','Stiffness'};
  legend(leg([~isempty(glo_EQ) any(mass) any(plotFrame) any(stiffloc)]));
else
  legend('off')
end
end
```

frfAddedElements

```
function [frf,sen,frf_old]=frfAddedElements
global h glo_frf glo_f glo_EQ
% function for getting frf with added elements and for update maxiumum
% transfer with added elements

% mass points
massPointsLoc = str2num(h.massLoc.String); %#ok<ST2NM>
mass = str2num(h.mass.String); %#ok<ST2NM>
massPoints=[];
if length(massPointsLoc)==length(mass)
  if ~isempty(massPointsLoc)
    massPoints=struct('loc',num2cell(massPointsLoc),'mass',num2cell(mass));
  end
elseif nargout~=0
  % if the frf is needed for a calculation or plot and the number of mass
  % points and masses are not the same report the user that all mass
  % points will be igored in the calculation
  consoleWrite('Mass point locations and point masses not the same length, ALL point masses will be
ignored.');
end

% frames
frameLoc1=str2num(h.frameLoc1.String); %#ok<ST2NM>
frameLoc2=str2num(h.frameLoc2.String); %#ok<ST2NM>
frameMass=str2num(h.frameMass.String); %#ok<ST2NM>
frameStiff=str2num(h.frameStiff.String); %#ok<ST2NM>
frames=[];
if length(frameLoc1)==length(frameLoc2)&&...
    length(frameLoc2)==length(frameMass)&&...
    length(frameLoc2)==length(frameStiff)
  if ~isempty(frameLoc1)

frames=struct('loc1',num2cell(frameLoc1),'loc2',num2cell(frameLoc2),'mass',num2cell(frameMass),'sti
ffness',num2cell(frameStiff));
  end
elseif nargout~=0
  % if the frf is needed for a calculation or plot and the number of
  % locations, masses and stiffness are not the same report the user that
  % all frames will be igored in the calculation
  consoleWrite('Frame parameters not the same length, ALL frames will be ignored.');
end

% stiffnesspoints
stiffPoint=str2num(h.stiffPoint.String); %#ok<ST2NM>
stiffVal=str2num(h.stiffPointValue.String); %#ok<ST2NM>
stiffnessPoints=[];
if length(stiffPoint)==length(stiffVal)
  if ~isempty(stiffPoint)
    stiffnessPoints=struct('loc',num2cell(stiffPoint),'stiffness',num2cell(stiffVal));
  end
elseif nargout~=0
```


©MECAL 2018 Confidential

```
  % if the frf is needed for a calculation or plot and the number of
  % locations and stiffness are not the same report the user that
  % all stiffness will be igored in the calculation
  consoleWrite('Point stiffness parameters not the same length, ALL point stiffness will be
ignored.');
end

[frf,sen,frf_old] = addElements(glo_frf,glo_f,massPoints,frames,stiffnessPoints,glo_EQ);

% update maximum transfer
maxTransferLoc = maxTransfer(frf,glo_f,str2num(h.maxTransferRange.String)); %#ok<ST2NM>
h.plotTransferText1b.String=['Maxium transfer of the altered system is z'
num2str(maxTransferLoc(1)) '/F' num2str(maxTransferLoc(2))];
end
```

addElements

```
function [frf,sen,frf_old] = addElements(frf,f,massPoints,frames,stiffnessPoints,EQs)
% [frf,sen] = addElements(frf,f,massPoints,frames,stiffnessPoints,EQs)
%  Function for adding elements to frfs. Mass points, frames, stiffness
%  points and equalizers can be added.
% 26-6-2018 fkoo

% Calc standard sen
sen=zeros(size(frf,1),size(frf,1),size(frf,3));
for i=1:size(frf,3)
  sen(:,:,i)=eye(size(frf,1));
end

% if no added elements exist skip rest function
if isempty(massPoints)&&isempty(frames)&&isempty(stiffnessPoints)&&length(EQs)==0 %#ok<ISMT>
  frf_old=frf;
  return;
end

% get omega
w=2*pi*f;

% Invert frf to get dynamic matrix
D=complex(zeros(size(frf))); % Preallocated
for i=1:size(frf,3)
  D(:,:,i)=inv(frf(:,:,i));
end

%add masspoints
if ~isempty(massPoints)
  for i=1:length(massPoints)
    D(massPoints(i).loc,massPoints(i).loc,:)=squeeze(D(massPoints(i).loc,massPoints(i).loc,:))-
w.^2*massPoints(i).mass';
  end
end
```

```
% add frames
if ~isempty(frames)
  for i=1:length(frames)
    % add half mass to points 1
    D(frames(i).loc1,frames(i).loc1,:)=squeeze(D(frames(i).loc1,frames(i).loc1,:))-
w.^2*frames(i).mass'/2;
    % add half mass to points 2
    D(frames(i).loc2,frames(i).loc2,:)=squeeze(D(frames(i).loc2,frames(i).loc2,:))-
w.^2*frames(i).mass'/2;
    % add stiffness

D(frames(i).loc1,frames(i).loc1,:)=squeeze(D(frames(i).loc1,frames(i).loc1,:))+frames(i).stiffness;

D(frames(i).loc2,frames(i).loc2,:)=squeeze(D(frames(i).loc2,frames(i).loc2,:))+frames(i).stiffness;
    D(frames(i).loc1,frames(i).loc2,:)=squeeze(D(frames(i).loc1,frames(i).loc2,:))-
frames(i).stiffness;
    D(frames(i).loc2,frames(i).loc1,:)=squeeze(D(frames(i).loc2,frames(i).loc1,:))-
frames(i).stiffness;
  end
end

% add stiffnesspoints
if ~isempty(stiffnessPoints)
  for i=1:length(stiffnessPoints)

D(stiffnessPoints(i).loc,stiffnessPoints(i).loc,:)=squeeze(D(stiffnessPoints(i).loc,stiffnessPoints
(i).loc,:))+stiffnessPoints(i).stiffness;
  end
end

% add EQ
if length(EQs)~=0 %#ok<ISMT>
  % get transfer before adding
  frf_old=complex(zeros(size(D)));
  for i=1:size(frf,3)
    frf_old(:,:,i)=inv(D(:,:,i));
  end

  % add EQ
  for i=1:length(EQs)
    if ~isempty(EQs(i).transfer)

D(EQs(i).point,EQs(i).point,:)=squeeze(D(EQs(i).point,EQs(i).point,:))+EQs(i).transfer.*(w*1i);
    else
      D(EQs(i).point,EQs(i).point,:)=squeeze(D(EQs(i).point,EQs(i).point,:))+EQs(i).stiffness;
    end
  end

  % get transfer and sen
  for j=1:size(frf,3)
    frf(:,:,j)=inv(D(:,:,j));
```


©MECAL 2018 Confidential

```
    sen(:,:,j)=frf(:,:,j)/(frf_old(:,:,j));
  end
else
  % Invert dynamic matrix to get updated frf
  for i=1:size(frf,3)
    frf(:,:,i)=inv(D(:,:,i));
  end
  frf_old=frf;
end
end
```

editMaxTransfer

```
function editMaxTransfer(~,~)
% function for editing max transfer notification due to change in range

global h glo_frf glo_f
range=str2num(h.maxTransferRange.String);%#ok<ST2NM>

try
  maxTransferLoc = maxTransfer(glo_frf,glo_f,range);
  h.plotTransferText1.String=['Maxium transfer of the unaltered system is z'
num2str(maxTransferLoc(1)) '/F' num2str(maxTransferLoc(2))];
  maxTransferLoc = maxTransfer(frfAddedElements,glo_f,range);
  h.plotTransferText1b.String=['Maxium transfer of the altered system is z'
num2str(maxTransferLoc(1)) '/F' num2str(maxTransferLoc(2))];
catch
end
end
```

plotTransfer

```
function plotTransfer(~,~)
global h glo_frf glo_f glo_Coh
% function for plotting the selected transfers

% get i/o
F=str2num(h.plotTransferF.String); %#ok<ST2NM>
S=str2num(h.plotTransferS.String); %#ok<ST2NM>

% check if there are values
if isempty(F)&&isempty(S)
  consoleWrite('No in and outputs chosen.')
elseif isempty(F)
  consoleWrite('No inputs chosen.')
elseif isempty(S)
  consoleWrite('No outputs chosen.')
else
  % in case there are values plot transfers
  figure('position',h.mainScreen.Position) % new figure
  leg=cell(0); % legend cell array
```


©MECAL 2018 Confidential

```
  % get new transfer
  frfNew=frfAddedElements;

  % get target type (x,v,a)
  type=h.plotTransferSelctor.Value;
  typeConvert={'z','v','a'};
  type=typeConvert{type};

  % plot transfer and add the entry to legend
  for f=F
    for s=S
      switch type
        case 'z'
          % original
          plotSpectrumCoH(glo_f,squeeze(glo_frf(s,f,:)),squeeze(glo_Coh(s,f,:)),type,[],'--');
          setLineColorOneBack(gcf);
          % altered
          plotSpectrumCoH(glo_f,squeeze(frfNew(s,f,:)),squeeze(glo_Coh(s,f,:)),type);
        case 'v'
          % original

plotSpectrumCoH(glo_f,squeeze(glo_frf(s,f,:)).*(glo_f*2*pi*1i),squeeze(glo_Coh(s,f,:)),type,[],'--
');
          setLineColorOneBack(gcf);
          % altered

plotSpectrumCoH(glo_f,squeeze(frfNew(s,f,:)).*(glo_f*2*pi*1i),squeeze(glo_Coh(s,f,:)),type);
        case 'a'
          % original
          plotSpectrumCoH(glo_f,squeeze(glo_frf(s,f,:)).*-
(glo_f*2*pi).^2,squeeze(glo_Coh(s,f,:)),type,[],'--');
          setLineColorOneBack(gcf);
          % altered
          plotSpectrumCoH(glo_f,squeeze(frfNew(s,f,:)).*-
(glo_f*2*pi).^2,squeeze(glo_Coh(s,f,:)),type);
      end
      % add legend entry
      leg{end+1}=[type num2str(s) '/F' num2str(f) ' original']; %#ok<AGROW>
      leg{end+1}=[type num2str(s) '/F' num2str(f) ' altered']; %#ok<AGROW>
    end
  end

  % set legend
  subplot(3,1,1)
  legend(leg);

  % write added elements
  writeAdded;
end
end
```


©MECAL 2018 Confidential

plotTransferCo

```
function plotTransferCo(~,~)
global h glo_frf glo_f glo_Coh
% function for plotting colocated transfers

% get target type (x,v,a)
type=h.plotTransferSelctor.Value;
typeConvert={'z','v','a'};
type=typeConvert{type};

figure('position',h.mainScreen.Position) % new figure
leg=cell(0); % legend cell array

% get transfer
frfNew=frfAddedElements;

% plot transfer and add the entry to legend
for i=1:size(glo_frf,1)
  switch type
    case 'z'
      % original
      plotSpectrumCoH(glo_f,squeeze(glo_frf(i,i,:)),squeeze(glo_Coh(i,i,:)),type,[],'--');
      setLineColorOneBack(gcf);
      % altered
      plotSpectrumCoH(glo_f,squeeze(frfNew(i,i,:)),squeeze(glo_Coh(i,i,:)),type);
    case 'v'
      % original

plotSpectrumCoH(glo_f,squeeze(glo_frf(i,i,:)).*(glo_f*2*pi*1i),squeeze(glo_Coh(i,i,:)),type,[],'--
');
      setLineColorOneBack(gcf);
      % altered
      plotSpectrumCoH(glo_f,squeeze(frfNew(i,i,:)).*(glo_f*2*pi*1i),squeeze(glo_Coh(i,i,:)),type);
    case 'a'
      % original
      plotSpectrumCoH(glo_f,squeeze(glo_frf(i,i,:)).*-
(glo_f*2*pi).^2,squeeze(glo_Coh(i,i,:)),type,[],'--');
      setLineColorOneBack(gcf);
      % altered
      plotSpectrumCoH(glo_f,squeeze(frfNew(i,i,:)).*-(glo_f*2*pi).^2,squeeze(glo_Coh(i,i,:)),type);
  end
  % add legend entry
  leg{end+1}=[type num2str(i) '/F' num2str(i) ' original']; %#ok<AGROW>
  leg{end+1}=[type num2str(i) '/F' num2str(i) ' altered']; %#ok<AGROW>
end

% set legend
subplot(3,1,1)
legend(leg);

% write added elements
```


©MECAL 2018 Confidential

```
writeAdded;
end
```

plotSpectrumCoH

```
function plotSpectrumCoH(f,frf,CoH,type,Title,plotOptions)
% plotSpectrumCoH(f,frf,CoH,type,Title,plotOptions)
%  Function for plotting a frf spectrum in a figure. Needs to have the
%  frf(s), frequency domain f and coherence CoH. Can take the optional
%  parameter type and Title to set the correct xlabel and a title.
%  25-5-2018 fkoo
% Version 2 to handle other xlabels then x 26-6-2018 fkoo
% Version 3 to pass plot options 4-7-2018 fkoo

% Input check
if nargin<3
  error('Not enough inputs for plotSpectrum.')
elseif nargin==3
  type='x';
  Title='';
  plotOptions=[];
elseif nargin==4
  Title='';
  plotOptions=[];
elseif nargin==5
  plotOptions=[];
elseif nargin>6
  error('To many inputs given for plotSpectrum.')
end

subplot(3,1,1) % Open subplot for abs plot
title(Title) % Set title
if isempty(plotOptions)
  loglog(f,abs(frf)) % Plot
else
  loglog(f,abs(frf),plotOptions) % Plot
end
xlabel('Frequency [Hz]'),ylabel([type '/F [m/N]']) % Set labels
grid on
hold on

subplot(3,1,2) % Open subplot for phase plot
if isempty(plotOptions)
  semilogx(f,180/pi*angle(frf)) % Plot
else
  semilogx(f,180/pi*angle(frf),plotOptions) % Plot
end
xlabel('Frequency [Hz]'),ylabel('Phase [Deg]') % Set labels
grid on
hold on
ylim([-180,180]); % Limit phase plot to stay between -180 and 180
```


©MECAL 2018 Confidential

```
set(gca,'ytick', -180:90:180); % Set ylabels to usefull increments

subplot(3,1,3) % Open subplot for phase plot
if isempty(plotOptions)
  semilogx(f,CoH) % Plot
else
  semilogx(f,CoH,plotOptions) % Plot
end
xlabel('Frequency [Hz]'),ylabel('Coherence [-]') % Set labels
grid on
hold on
end
```

setLineColorOneBack

```
function setLineColorOneBack(figHan)
% setLineColorOneBack(figHan)
%  Function for setting for all axis for a figure hangle figHan all colors
%  1 step back
% 4-7-2018 fkoo

% Get all axes from fig
ax=findall(figHan,'type','axes');

for i=1:length(ax)
  ax(i).ColorOrderIndex=ax(i).ColorOrderIndex-1;
end
end
```

writeAdded

```
function writeAdded
global h glo_EQ
% Function for writting what added elements are added to transfer or ODS

% start with empty added elements
writeString=[];

% check if EQ are added and if add to writeString
if ~isempty(glo_EQ)
  writeString=[writeString 'EQ placed on points: ' num2str([glo_EQ.point]) ' '];
end

% check if point masses are added and if add to writeString
if length(str2num(h.massLoc.String))==length(str2num(h.mass.String))&&~isempty(h.massLoc.String)
%#ok<ST2NM>
  writeString=[writeString 'Point masses on: ' h.massLoc.String ' with mass(es): ' h.mass.String '
'];
end

% check if frames are added and if add to writeString
```

```
if length(str2num(h.frameLoc1.String))==length(str2num(h.frameLoc2.String))&&...
    length(str2num(h.frameLoc1.String))==length(str2num(h.frameMass.String))&&...
    length(str2num(h.frameLoc1.String))==length(str2num(h.frameStiff.String))&&...
    ~isempty(h.frameLoc1.String)%#ok<ST2NM>
  writeString=[writeString 'Frames placed between points: ['...
    h.frameLoc1.String '] and [' h.frameLoc2.String...
    '] with mass(es): ' h.frameMass.String ' and stiffness(es): '...
    h.frameStiff.String];
end

% check if point stiffness are added and if add to writeString
if
length(str2num(h.stiffPoint.String))==length(str2num(h.stiffPointValue.String))&&~isempty(h.stiffPo
int.String) %#ok<ST2NM>
  writeString=[writeString 'Point stiffness(es) on: ' h.stiffPoint.String ' with stiffness(es): '
h.stiffPointValue.String ' '];
end

% write writeString to the bottom of figure
uicontrol('style','text','Units','normalized', 'Position', [0 0 1 0.05],...
  'string',writeString,'HorizontalAlignment','left');

end
```

plotVibF

```
function plotVibF(~,~)
% Function for plotting vibrations
global h glo_VibF glo_f glo_X glo_Y

Xt=glo_X';
Xt=Xt(:);
Yt=glo_Y';
Yt=Yt(:);

h.VibF.fig=figure('position',h.mainScreen.Position,'name','Vibration forces'); % new figure
h.VibF.sld = uicontrol('Style', 'slider','Units','normalized',...
        'Min',1,'Max',size(glo_VibF,2),'Value',1,...
        'Position', [.1 .95 .2 .04],...
        'SliderStep',[1/length(glo_f) 10/length(glo_f)],...
        'Callback', @adjustFVibF);
h.VibF.fText=uicontrol('style', 'text','Units','normalized',...
    'position',[.31 .95 .2 .04],...
    'String',[num2str(glo_f(1)) 'Hz'],'HorizontalAlignment','left',...
    'FontUnits','normalized','FontSize',1);

h.VibF.F=quiver3(Xt,Yt,zeros(size(glo_VibF,1),1),zeros(size(glo_VibF,1),1),zeros(size(glo_VibF,1),1
),abs(glo_VibF(:,1)),'lineWidth',3,'AutoScale','off');

ylabel('Y position [m]'); xlabel('X position [m]');zlabel('Disturbance forces [N]');
title('Vibration forces')
```

```
rotate3d on

end

function adjustFVibF(~,~)
global h glo_VibF glo_f
k=round(h.VibF.sld.Value);
h.VibF.fText.String=[num2str(glo_f(k)) 'Hz'];
h.VibF.F.WData=abs(glo_VibF(:,k));
end
```

plotVib

```
function plotVib(~,~)
% Function for plotting vibrations
global h glo_frf glo_f glo_VibF

% get locations
loc=str2num(h.vibLoc.String); %#ok<ST2NM>

% get type
type=h.vibType.String;
typeVal=h.vibType.Value;

% check if there are values
if isempty(loc)
    consoleWrite('No locations selected');
    return;
end

% get frf with added elements
frf=frfAddedElements;

figure('position',h.mainScreen.Position) % new figure
leg=cell(0); % legend cell array

% prealloact vibrations
vib=complex(zeros(size(glo_VibF)));
vibAddedElements=vib;

% calc vibrations
for j=1:size(frf,3)
    vib(:,j)=squeeze(glo_frf(:,:,j))*glo_VibF(:,j);
    vibAddedElements(:,j)=squeeze(frf(:,:,j))*glo_VibF(:,j);
    % adjust for type
    for i=1:typeVal
        vib(:,j)=vib(:,j).*glo_f(j)*1i;
        vibAddedElements(:,j)=vibAddedElements(:,j).*glo_f(j)*1i;
    end
end
```


©MECAL 2018 Confidential

```
for i=1:length(loc)
    % plot vibrations
    loglog(glo_f,abs(vib(i,:)),'--');
    hold on;
    setLineColorOneBack(gcf)
    loglog(glo_f,abs(vibAddedElements(i,:)),'LineWidth',2);

    % add legend entry
    leg{end+1}=[type{typeVal} num2str(loc(i))]; %#ok<AGROW>
    leg{end+1}=[type{typeVal} num2str(loc(i)) ' added elements']; %#ok<AGROW>

end

xlabel('Frequency [Hz]'),ylabel([type{typeVal} ' [m]']),title('Vibration levels') % Set labels

% set legend
legend(leg);

writeAdded;

end
```

plotSensitivity

```
function plotSensitivity(~,~)
global h glo_f

F=str2num(h.sensitivityF.String); %#ok<ST2NM>
S=str2num(h.sensitivityS.String); %#ok<ST2NM>


% check if there are values
if isempty(F)&&isempty(S)
  consoleWrite('No in and outputs chosen.')
elseif isempty(F)
  consoleWrite('No inputs chosen.')
elseif isempty(S)
  consoleWrite('No outputs chosen.')
else
  % Get sen
  [~,sen]=frfAddedElements;
  % in case there are values plot transfers
  figure('position',h.mainScreen.Position) % new figure
  leg=cell(0); % legend cell array

  for f=F
    for s=S
      plotSpectrum(glo_f,squeeze(sen(s,f,:)),'Sensitivity');
    % add legend entry
      leg{end+1}=['z' num2str(s) '/z' num2str(f)]; %#ok<AGROW>
    end
```


©MECAL 2018 Confidential

```
  end

  % set legend
  subplot(2,1,1)
  ylabel('Gain [m/m]')
  legend(leg);
  writeAdded;
end
end
```

plotSpectrum

```
function plotSpectrum(f,frf,Title)
% plotSpectrum(f,frf,vargin)
%  Function for plotting a frf spectrum in a figure. Needs to have the
%  frf(s), frequency domain f. Can take the optional parameter Title to
%  set a title.
%  25-5-2018 fkoo

% Input check
if nargin<2
  error('Not enough inputs for plotSpectrum.')
elseif nargin==2
  Title='';
elseif nargin>3
  error('To many inputs given for plotSpectrum.')
end

subplot(2,1,1) % Open subplot for abs plot
title(Title) % Set title
loglog(f,abs(frf)) % Plot
xlabel('Frequency [Hz]'),ylabel('x/F [m/N]') % Set labels
grid on
hold on

subplot(2,1,2) % Open subplot for phase plot
semilogx(f,180/pi*angle(frf)) % Plot
xlabel('Frequency [Hz]'),ylabel('Phase [Deg]') % Set labels
grid on
hold on
ylim([-180,180]); % Limit phase plot to stay between -180 and 180
set(gca,'ytick', -180:90:180); % Set ylabels to usefull increments
end
```

runODS

```
function runODS(source,~)
global h glo_frf glo_X glo_Y
% function to run a ODS

% get source
```

```
val=str2double(source.Tag);

% get user inputs
ODSF=str2num(h.ODSF(val).String); %#ok<ST2NM>
ODSFa=str2num(h.ODSFa(val).String); %#ok<ST2NM>

% check if values for the forces are given
if isempty(ODSF)||isempty(ODSFa)
  consoleWrite('Not enough parameters given for ODS.');
  return;
  % check if the force locations and forces have the the same number of
  % entries
elseif length(ODSF)~=length(ODSFa)
  consoleWrite('Force points and amplitude not the same length.');
  return;
end

% create external force vector part 1
F=zeros(size(glo_frf,1),1);
try
  % create external force vector part 2
  F(ODSF)=ODSFa;

  % run ODS for the correct source
  if val==1
    % get frf with name frf
    frf=glo_frf;
    % run ODS
    ODS(frf,glo_X,glo_Y,F);
  else
    % get frf with name frfWithAddedElements
    frfWithAddedElements=frfAddedElements;
    % run ODS
    ODS(frfWithAddedElements,glo_X,glo_Y,F);
    % write added elements
    writeAdded
  end
catch me
  consoleWrite(me.message);
end
end
```

ODS

```
function ODS(frf,X,Y,F)
% ODS(frf,X,Y,F)
%  Function for animating a ODS (Operating Deflection Shapes) of a frf
%  with external force vector F. Is also capble of recording the
%  animation to a avi movie.
%  5-6-2018 fkoo
% Version 2 to handle new frf format and slider use to set fods 26-6-2018 fkoo
```


©MECAL 2018 Confidential

```
% Version 3 added contious animation button 4-7-2018 fkoo

% get or allocated globals
global Z odsFig hods N glo_f maxZ h

if isempty(N)
  N=1;
else
  N=N+1;
end

% get gridsize
gridSize=size(X);

% get responce
maxResponce=zeros(1,size(frf,3));
for i=1:size(frf,3)
  responce=frf(:,:,i)*F;
  maxResponce(i)=max(abs(responce));
  Z{N,i}=reshape(responce,gridSize(1),gridSize(2));
end
maxZ(N)=max(maxResponce);

% animate
% Preset animation
% Define input parameters and UI
odsFig.fig(N) = figure('position',h.mainScreen.Position,'name','ODS');
% button reanimate
odsFig.but(N)= uicontrol('style','pushbutton','Units','normalized',...
  'position', [.01 .95 .04 .04],...
  'string','Reanimate','Tag',num2str(N),'Callback',@animate);
odsFig.sld(N) = uicontrol('Style', 'slider','Units','normalized',...
  'Min',1,'Max',length(glo_f),'Value',1,...
  'Position', [.1 .95 .2 .04],...
  'SliderStep',[1/length(glo_f) 10/length(glo_f)],...
  'Tag',num2str(N),'Callback', @adjustFods);
odsFig.fText(N)=uicontrol('style', 'text','Units','normalized',...
  'position',[.31 .95 .2 .04],...
  'String',[num2str(glo_f(1)) 'Hz'],'HorizontalAlignment','left',...
  'FontUnits','normalized','FontSize',1);
uicontrol('style', 'text','Units','normalized',...
  'position',[.01 .9 .2 .04],'string','Movie name:','HorizontalAlignment','left');
odsFig.radBut(N)=uicontrol('style','popup','Units','normalized',...
  'position',[.1 .9 .1 .04],...
  'String',{'Absolute scaling','Relative scaling'},'Tag',num2str(N),...
  'Callback',@scalingAdjust);
odsFig.movieName(N) = uicontrol('style','edit','Units','normalized',...
  'position',[.01 .85 .08 .04],'string','movie.avi');
odsFig.but2(N)= uicontrol('style','pushbutton','Units','normalized',...
  'position', [.01 .8 .04 .04],...
  'Callback',@recordMovie,'string','Record','Tag',num2str(N));
odsFig.but3(N)= uicontrol('style','pushbutton','Units','normalized',...
```

```
  'position', [.01 .75 .04 .04],...
  'Callback',@animateAll,'string','Animate all');
odsFig.but4(N)= uicontrol('style','pushbutton','Units','normalized',...
  'position', [.01 .7 .08 .04],...
  'Callback',@animateConAll,'string','Animate all continous');

hods(N)= surf(X,Y,abs(Z{N,1}).*cos(angle(Z{N,1})));

maxF=max(F);
F=0.8*F*maxZ(N)/maxF;
Xt=X';
Xt=Xt(:);
Yt=Y';
Yt=Yt(:);

hold on
quiver3(Xt,Yt,zeros(size(F)),zeros(size(F)),zeros(size(F)),F,'lineWidth',3);

zlim(1.1*[-maxZ(N) maxZ(N)])
ylabel('Y position [m]'); xlabel('X position [m]');zlabel('Z position [m]');
title(['ODS for ' inputname(1)])
rotate3d on

% Run animation
animate(odsFig.but(N));
end

function animate(source,~,movieName)
global odsFig Z hods
i=str2double(source.Tag);

% Now animate!

[odsFig.but(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but4(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.t(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('off');

if nargin==3
  v = VideoWriter(movieName);
  open(v);
end

for phi = pi/80:pi/80:2*pi
  if nargin~=3
    pause(0.02);
  end
  j=round(odsFig.sld(i).Value);
  set(hods(i),'zdata',abs(Z{i,j}).*cos(angle(Z{i,j})+phi));
  if nargin==3
```


©MECAL 2018 Confidential

```
    writeVideo(v,getframe(gcf));
  end
end

if nargin==3
  close(v);
end

[odsFig.but(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but4(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.t(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('on');


end

function recordMovie(source,~)
global odsFig
animate(source,[],get(odsFig.movieName(source.Value),'string'));
end

function animateAll(~,~)
global odsFig Z hods

% Now animate!
[odsFig.but(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.but4(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.t(isvalid(odsFig.fig)).Visible]=deal('off');
[odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('off');


for phi = 1:pi/80:4*pi
  pause(0.02);
  for i=1:size(Z,1)
    if isvalid(odsFig.fig(i))
      j=round(odsFig.sld(i).Value);
      set(hods(i),'zdata',abs(Z{i,j}).*cos(angle(Z{i,j})+phi));
    end
  end
end

[odsFig.but(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.but4(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.t(isvalid(odsFig.fig)).Visible]=deal('on');
[odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('on');
end
```


©MECAL 2018 Confidential

```
function adjustFods(source,~)
global odsFig glo_f Z
i=str2double(source.Tag);
k=round(odsFig.sld(i).Value);
odsFig.fText(i).String=[num2str(glo_f(k)) 'Hz'];
if odsFig.radBut(i).Value==2
  maxZ=max(max(abs(Z{i,k})));
  zlim(1.1*[-maxZ maxZ])
end
end

function scalingAdjust(source,~)
i=str2double(source.Tag);
global odsFig Z maxZ
figure(odsFig.fig(i));
j=source.Value;
k=round(odsFig.sld(i).Value);
if j==1
  zlim(1.1*[-maxZ(i) maxZ(i)])
else
  maxZloc=max(max(abs(Z{i,k})));
  zlim(1.1*[-maxZloc maxZloc])
end
end

function animateConAll(~,~)
global odsFig Z hods animateRun
if isempty(animateRun)
  animateRun=1;
  [odsFig.but4(isvalid(odsFig.fig)).String]=deal('Stop animation');
  [odsFig.but(isvalid(odsFig.fig)).Visible]=deal('off');
  [odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('off');
  [odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('off');
  [odsFig.t(isvalid(odsFig.fig)).Visible]=deal('off');
  [odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('off');
  phi = 1;
  while ~isempty(animateRun)
    % Now animate!


    pause(0.02);
    for i=1:size(Z,1)
      if isvalid(odsFig.fig(i))
        j=round(odsFig.sld(i).Value);
        set(hods(i),'zdata',abs(Z{i,j}).*cos(angle(Z{i,j})+phi));
      end
    end


    phi=phi+pi/80;
  end
```


©MECAL 2018 Confidential

```
  [odsFig.but(isvalid(odsFig.fig)).Visible]=deal('on');
  [odsFig.but2(isvalid(odsFig.fig)).Visible]=deal('on');
  [odsFig.but3(isvalid(odsFig.fig)).Visible]=deal('on');
  [odsFig.t(isvalid(odsFig.fig)).Visible]=deal('on');
  [odsFig.movieName(isvalid(odsFig.fig)).Visible]=deal('on');
  [odsFig.but4(isvalid(odsFig.fig)).String]=deal('Animate all continous');
else
  animateRun=[];
end
end
```

addEQ

```
function addEQ(~,~)
global p C glo_X glo_Y glo_f glo_EQ plant h glo_Coh
% function for adding EQ to a point

% start new GUI figure screen with handle so it can be deleted
p.fig=figure('position',h.mainScreen.Position);
clf

% plot floor plan
buildPlan;
title('Select point where to add EQ')
[x,y]=ginput(1);

% get nodes
X=glo_X';
X=X(:);
Y=glo_Y';
Y=Y(:);


% get distances from mouse click to drawings point
dis=zeros(size(X));
for i=1:length(X)
  dis(i)=norm([x-X(i) y-Y(i)]);
end
p.node=find(min(dis)==dis);

% Check if this node had a EQ already and if so ask what the user wants
if length(glo_EQ)~=0 %#ok<ISMT>
  if ismember(p.node,[glo_EQ.point])
    clf;
    text([0.1 0.1 0.4 0.7],[.8 0.1 0.1 0.1],...
      {['Note ' num2str(p.node) ' has a EQ already. Do you want to override it?'],...
      'Overide','choose another point','Stop adding EQ'});
    [x,y]=ginput(1);
    disSelector=[norm([0.1-x 0.1-y]) norm([0.4-x 0.1-y]) norm([0.7-x 0.1-y])];
    chosenOption=find(disSelector==min(disSelector));
```

```
    % do want the user has chosen
    switch chosenOption
      case 1
        % remove EQ and continue with function
        glo_EQ([glo_EQ.point]==p.node)=[];
      case 2
        % close figure and restart function
        close(p.fig);
        addEQ;
      case 3
        % close figure and % stop function
        close(p.fig);
        return;
    end
  end
end

clf;
text([0.1 0.1 .6],[.8 0.1 0.1],{'What kind of EQ do you want to add?','Simple','Tunable'});
[x,y]=ginput(1);
disSelector=[norm([0.1-x 0.1-y]) norm([0.6-x 0.1-y])];
chosenOption=find(disSelector==min(disSelector));

% close figure
close(p.fig);

if chosenOption==1
  % Simple EQ chosen

  % Get frf with the already added elements
  frf=frfAddedElements;

  n=length(glo_EQ)+1;
  glo_EQ(n).point=p.node;
  glo_EQ(n).transfer=[];

  % get Stiffness matrix
  K=inv(abs(frf(:,:,find(glo_Coh>=0.95,1))));

  % get stiffness to add (10 times)
  glo_EQ(n).stiffness=10*K(p.node,p.node);

  editPlan;
  % update eq text
  if length(glo_EQ)==0 %#ok<ISMT>
    h.EQLoc.String='No EQs';
  else
    EQpoints='Equalizers placed on: ';
    for i=1:length(glo_EQ)
      EQpoints=[EQpoints num2str(glo_EQ(i).point) ', ']; %#ok<AGROW>
    end
    h.EQLoc.String=EQpoints(1:end-2);
```


©MECAL 2018 Confidential

```
  end
  frfAddedElements;


else
  % Tunable EQ chosen

  % Get frf with the already added elements and convert position to velocity
  frf=frfAddedElements;
  frf=squeeze(frf(p.node,p.node,:)).*glo_f*2*pi*1i;

  A = textread('EQ_firmware_used.txt','%s','bufsize',1e5); %#ok<DTXTRD>

  try
    eval(['p.' strcat(A{1:end})])
  catch
    consoleWrite('failed evaluation of default_setting.txt, please check if textfile does not
contain matlab coding errors!');
  end

  % copy to global
  p.settings=eval('settings');

  % initialize tuning stuff
  p.tune.filter_names = {'lag','lead','1st order lowpass','2nd order lowpass','1st order
highpass','notch','2nd order filter'};
  p.tune.filter_fields = {'lag','lead','lowpass1','lowpass2','highpass1','notch','filter2'};
  p.tune.filter_nedit = [2 2 1 2 1 4 6];
  p.tune.nmax = [5 5 5 5 5 20 5];
  p.tune.orders = [1 1 1 2 1 2 2];
  p.tune.ordermax = sum(p.tune.orders.*p.tune.nmax);
  p.Ts = p.settings.Ts;
  p.fs = 1/p.Ts;

  % plot controller performance plot in window
  p.tune.init=1;
  p.tune.xlims=[glo_f(1) glo_f(end)];
  mainScreenSize=h.mainScreen.Position;

  p.tune.fg = figure('position',mainScreenSize+mainScreenSize(1,3)/3*[1 0 -1 0],'name',['Controller
tuning node ' num2str(p.node)]);
  C.ch=1;

  % init C
  vcs_tune_init_C_struct;

  %setup figure handles for plant, sensitivity and controller plot
  subplot(2,3,[1 2]),%mag
  p.tune.handles.Pmag= loglog([1 500],[1 1],'b'); grid on,hold on
  p.tune.handles.Lmag= loglog([1 500],[1 1],'k');
  p.tune.handles.onemag= loglog([1 500],[1 1],'m--');
  legend('Plant','Plant*Control','1','location','NorthEast')
```


©MECAL 2018 Confidential

```
xlabel('Frequency [Hz]','fontsize',12)
ylabel('Magnitude [(m/s)/N]','fontsize',12)
p.tune.title_Pmag = title(['Controller node ' num2str(p.node)],'fontsize',12);
xlim(p.tune.xlims)
p.tune.ax1=gca;

subplot(2,3,[4 5]),
p.tune.handles.Pphs= semilogx([1 500],[0 0],'b'); grid on, hold on
p.tune.handles.Lphs= semilogx([1 500],[0 0],'k');
semilogx(p.tune.xlims,[1 1]*180,'r--')
semilogx(p.tune.xlims,[1 1]*-180,'r--')
xlabel('Frequency [Hz]','fontsize',12)
ylabel('Phase [deg]','fontsize',12)
xlim(p.tune.xlims)
p.tune.ax2=gca;

subplot(4,3,3),
p.tune.handles.Cmag=loglog([1 500],[0 0],'k'); grid on, xlim(p.tune.xlims)
p.tune.handles.Ctitle = title('Controller (order = 1)','fontsize',12);
xlabel('Frequency [Hz]','fontsize',12)
ylabel('Magnitude [N/(m/s)]','fontsize',12)

p.tune.ax3=gca;

subplot(4,3,6),
p.tune.handles.Cphs=semilogx([1 500],[0 0],'k'); grid on, xlim(p.tune.xlims)
xlabel('Frequency [Hz]','fontsize',12),ylim([-200 200])
ylabel('Phase [deg]','fontsize',12)

p.tune.ax4=gca;

subplot(2,3,6),
p.tune.handles.Smag=semilogx([1 500],20*log10(abs([1 1]))); grid on, hold on
semilogx(p.tune.xlims,[1 1]*6,'r--')
legend('S1','6 dB','location','best'), xlim(p.tune.xlims)
xlabel('Frequency [Hz]','fontsize',12)
ylabel('Magnitude [dB]','fontsize',12)
title('Sensitivity','fontsize',12)
p.tune.ax5=gca;
linkaxes([p.tune.ax1 p.tune.ax2 p.tune.ax3 p.tune.ax4 p.tune.ax5],'x');

plant=frf;

p = vcs_tune_plot_control_performance(p,glo_f,plant,C);

% add buttons to tune/save/close
button_names = {'Control filters','Save controller','Close and add EQ','Close'};
callbacks ={@change_filters,@saveController,@closeTuneAndSave,@closeTune};
for kb = 1:length(button_names)
  p.tune.pb_button(kb) = uicontrol(p.tune.fg,'Style','PushButton','Units','pixels',...
    'String',button_names{kb},'Position',[30+100*(kb-1) 20 100 25],'Callback', callbacks{kb});
end
```


©MECAL 2018 Confidential

```
  vcs_tune_change_filters;

end
end
```

vcs_tune_init_C_struct

```
% function C = vcs_tune_init_C_struct(h)

C=struct;
C.ss = ss(1);
C.gain = 1; %[N/m/s];
C.Ts=p.Ts; %[s]
C.fbw = 30; %[Hz]
C.gain_at_bw = 10; %[N/m/s]
C.nfilters = 0;

C.frf = calc_frf_data(c2d(C.ss,C.Ts,'tustin'),glo_f*2*pi);


p.tune.filter_fields = p.settings.Cfilter_fields;
p.tune.filter_nedit = p.settings.C_filter_nedit;
p.tune.nmax = [5 5 5 5 5 20 5];
for filtertype = 1:7
  eval(['C.',p.tune.filter_fields{filtertype},' =
zeros(p.tune.nmax(filtertype),p.tune.filter_nedit(filtertype));']);
end
% C.plant_filename = h.plant_fname;
```

change_filters

```
function change_filters(~,~)
global p C %#ok<NUSED>
try %#ok<TRYNC>
  close(p.tune.filter.fg)
end
vcs_tune_change_filters;
end
```

vcs_tune_change_filters

```
% vcs_tune_change_filters
%
% Description: This function enables user to see, change and update
% control filter settings
%
%
%
% file      : vcs_tune_change_filters.m
```


©MECAL 2018 Confidential

```
% last changes  : 25-07-2017
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file
% function h = vcs_tune_change_filters(h,C);



p.tune.filter.fg = figure('name','filter_input','position',mainScreenSize+[0 0 -
mainScreenSize(1,3)*2/3 -mainScreenSize(1,4)/2]);
dposy=0;
%controller gain
texts = {'bandwidth [Hz]','loopgain at bw [N/m/s]'};
defaults = {num2str(C.fbw),num2str(C.gain)};
for kedit = 1:2

  p.tune.filter.txt(kedit) = uicontrol(p.tune.filter.fg,'style','text','units','pixels',...
    'position',[10,200-20*(kedit-
1)+dposy,100,15],'string',texts{kedit},'HorizontalAlignment','Left');
    p.tune.filter.edit(kedit) = uicontrol(p.tune.filter.fg,'style','edit','units','pixels',...
    'position',[120,200-20*(kedit-1)+dposy,50,15],'string',defaults{kedit});
  p.tune.filter.pb(kedit) = uicontrol(p.tune.filter.fg,'Style','PushButton','Units','pixels',...
        'String','Set','Position',[180,200-20*(kedit-
1)+dposy,70,15],'Tag',num2str(kedit),'Callback',@setFilter);
end

% filter window


for kb = 1:length(p.tune.filter_names)
p.tune.filter.pb(kb+kedit) = uicontrol(p.tune.filter.fg,'Style','PushButton','Units','pixels',...
        'String',p.tune.filter_names{kb},'Position',[10,200-20*(kedit+kb-
1)+dposy,100,15],'Tag',num2str(kb),'Callback', @filterAdd);
  p.tune.filter.txtpb(kb) = uicontrol(p.tune.filter.fg,'Style','text','Units','pixels',...

'String',[num2str(p.settings.C_nfilters(kb)),'/',num2str(p.tune.nmax(kb))],'Position',[120,200-
20*(kedit+kb-1)+dposy,30,15]);

end
p.tune.filter.txtpb(kb+1) = uicontrol(p.tune.filter.fg,'Style','text','Units','pixels',...
        'String',['Order = ',num2str(order(C.ss)),'/',num2str(p.tune.ordermax)],'Position',[65,200-
20*(kedit+kb+1-1)+dposy,100,15]);
p.tune.filter_popup = [];
```


©MECAL 2018 Confidential

setFilter

```
function setFilter(source,~)
global p C glo_f plant %#ok<NUSED>
p.gain_done=0;
p.Cupdate = str2double(source.Tag);
vcs_tune_filter_update;
end
```

vcs_tune_filter_update

```
% vcs_tune_filter_update
%
% Description: This script checks all input figures for values and updates
% the controller structure 'C', then replots the controller in the
% performance figure
%
%
%
% file      : vcs_tune_filter_update.m
% last changes  : 07-08-2017
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file


%check all open windows and update filter struct C
for filtertype = 1:7


  if length(p.tune.filter_popup) >= filtertype
    if ishandle(p.tune.filter_popup(filtertype).fg)
      vals=[];
      for kfilt = 1:p.settings.C_nfilters(filtertype)
        for kedit = 1:p.settings.C_filter_nedit(filtertype)


          vals(kfilt,kedit) =
str2num(get(p.tune.filter_popup(filtertype).filter(kfilt).edit(kedit),'string'));
        end
      end
      %update C struct
      eval(['C.',p.settings.Cfilter_fields{filtertype},' = vals;']);
```


©MECAL 2018 Confidential

```
    end
  end

end
C.Ts = p.Ts;C.gain = 1;
vcs_compute_C_from_struct


%update order
set(p.tune.filter.txtpb(8),'String',['Order =
',num2str(sum(p.tune.orders.*C.nfilters)),'/',num2str(p.tune.ordermax)]);
C.fbw = str2num(get(p.tune.filter.edit(1),'string'));
C.gain_at_bw = str2num(get(p.tune.filter.edit(2),'string'));


%set bandwidth using measurement data
kfbw = find(min(abs(glo_f-C.fbw)) == abs(glo_f-C.fbw),1,'first');

Pmag = abs(plant(kfbw));

C.frf = calc_frf_data(c2d(C.ss,C.Ts,'tustin'),glo_f*2*pi);
Cmag = abs(C.frf(kfbw));
C.gain = C.gain_at_bw/Pmag/Cmag;
C.ss = C.ss*C.gain;


C.ssd = c2d(C.ss,C.Ts,'tustin');
C.frf = calc_frf_data(C.ssd,glo_f*2*pi);
% make controller filter from C struct


h = vcs_tune_plot_control_performance(p,glo_f,plant,C);
```

vcs_compute_C_from_struct

```
%  vcs_compute_C_from_struct
%
% Description: This script checks all input figures for values and updates
% the controller structure 'C', then replots the controller in the
% performance figure
%
%
%
% file      :  vcs_compute_C_from_struct.m
% last changes  : 28-09-2017
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
```


©MECAL 2018 Confidential

```
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file
C.ss=ss(1);


for filtertype = 1:length(p.settings.C_nfilters)

  for kfilt = 1:p.settings.C_nfilters(filtertype)

    val = eval(['C.',p.settings.Cfilter_fields{filtertype},'(kfilt,:);']);
    if sum(abs(val)) == 0 %make empty filter of order h.tune.orders(filtertype)
      order1 =p.tune.orders(filtertype);
      Ass = blkdiag(C.ss.A,zeros(order1,order1));
      Bss = [C.ss.B;zeros(order1,1)];
      Css = [C.ss.C zeros(1,order1)];
      Dss = C.ss.D;
      C.ss = ss(Ass,Bss,Css,Dss);
      clear Ass Bss Css Dss
    else



      if filtertype == 1 %lag
        filt = lag(val(2),val(1));
      elseif filtertype == 2 %lead
        filt = lead(val(2),val(1));
      elseif filtertype == 3 %1st order lowpass
        filt = tf([val(1)*2*pi],[1 val(1)*2*pi]);
      elseif filtertype == 4 %2nd order lowpass
        wlp =val(1)*2*pi;
        filt = tf([wlp^2],[1 2*val(2)*wlp wlp^2]);
      elseif filtertype == 5 %1st order highpass
        filt = 1-tf([val(1)*2*pi],[1 val(1)*2*pi]);
      elseif filtertype == 6 %notch
        filt = notch(val(1),val(2),val(3),val(4));
      elseif filtertype == 7 %2nd order filter
        filt = tf(val(1:3),val(4:6));
      else
        filt = ss(1);
      end
      C.ss=C.ss*filt;
    end
  end

end
```


©MECAL 2018 Confidential

```
% set bandwidth

C.ss = C.ss*C.gain;
C.ssd = c2d(C.ss,C.Ts,'tustin');
```

lag

```
% lag
%
% Description: generates lag filter
%
%
%
% file      : lag.m
% last changes  : 31-12-2015
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

function [D,alpha] = lag(phi_max,omega_max)
% D = lag(phi_max,omega_max)
%
% In: phi_max:  maximum phase lead in degrees
%   omega_max: frequency for maximum phase lead in Hz
% Out: lead compensator transfer function (continuous)


phi_rad = phi_max*pi/180;
omega_rad = omega_max*2*pi;

alpha = (1-sin(phi_rad))/(1+sin(phi_rad));
T = 1/(omega_rad*sqrt(alpha));

D = balreal(ss(tf([T 1/alpha],[T 1])));
```

lead

```
% lead
%
% Description: generates lead filter
%
%
```


©MECAL 2018 Confidential

```
%
% file      : lead.m
% last changes  : 31-12-2015
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

function [D,alpha] = lead(phi_max,omega_max)
% D = lead(phi_max,omega_max)
%
% In: phi_max:  maximum phase lead in degrees
%   omega_max: frequency for maximum phase lead in Hz
% Out: lead compensator transfer function (continuous)


phi_rad = phi_max*pi/180;
omega_rad = omega_max*2*pi;

alpha = (1-sin(phi_rad))/(1+sin(phi_rad));
T = 1/(omega_rad*sqrt(alpha));

D = balreal(ss(tf([T 1],[alpha*T 1])));
```

notch

```
% notch
%
% Description: generates notch filter
%
%
%
% file      : notch.m
% last changes  : 11-11-2015
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
```

```
% track changes  : v001: initial file

function G_notch = notch(omega_1,omega_2,beta_1,beta_2)

s = tf('s');

omega_1 = omega_1*2*pi;
omega_2 = omega_2*2*pi;

G_notch = balreal(ss((s^2/omega_1^2 + 2*beta_1*s/omega_1 + 1) / (s^2/omega_2^2 + 2*beta_2*s/omega_2
+ 1)));
```

calc_frf_data

```
% calc_frf_data
%
% Description: This function calculates frd data of model
%
%
%
% file      : calc_frf_data.m
% last changes  : 31-12-2015
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

function H_frf = calc_frf_data(H,omega)

[H_mag,H_phase] = bode(H,omega);
H_mag = squeeze(H_mag);
H_phase = squeeze(H_phase);
H_frf = H_mag.*exp(1i*H_phase*2*pi/360);
```

vcs_tune_plot_control_performance

```
% vcs_tune_plot_control_performance
%
% Description:
%
%
%
% file      : vcs_tune_plot_control_performance.m
% last changes  : 25-07-2017
```


©MECAL 2018 Confidential

```
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

function h = vcs_tune_plot_control_performance(h,f,P,C)

%plant and open loop
set(h.tune.handles.Pmag,'XData',f,'YData',abs(P.*C.gain));
set(h.tune.handles.Lmag,'XData',f,'YData',abs(P.*C.frf));
set(h.tune.handles.Pphs,'XData',f,'YData',angle(P.*C.gain)*360/2/pi);
set(h.tune.handles.Lphs,'XData',f,'YData',angle(P.*C.frf)*360/2/pi);
%set(h.tune.title_Pmag,'string',strrep(['Controller ch',num2str(h.tune.ch ),'
('',h.plant_fname,')'],'_','\_'));
%controller
set(h.tune.handles.Cmag,'XData',f,'YData',abs(C.frf));
set(h.tune.handles.Ctitle,'string',['Controller (order = ',num2str(order(C.ss)),')']);
set(h.tune.handles.Cphs,'XData',f,'YData',angle(C.frf)*360/2/pi);

%sensitivity
set(h.tune.handles.Smag,'XData',f,'YData',20*log10(abs(1./(1+P.*C.frf))));
%set(h.tune.handles.mag_local_iso_spec,'XData',h.tune.local_iso_specs(:,1),'YData',20*log10(abs(1./
h.tune.local_iso_specs(:,2))))
```

filterAdd

```
function filterAdd(source,~)
global p C glo_f plant %#ok<NUSED>
filtertype =str2double(source.Tag); %#ok<NASGU>
vcs_tune_filter_popup;
end
```

vcs_tune_filter_popup

```
% vcs_tune_filter_popup
%
% Description: This function opens a window depending on the slected
% filter type and lets the user update filters.
%
%
%
% file      : vcs_tune_filter_popup.m
% last changes  : 25-07-2017
% author     : Ronald Rijkers
```


©MECAL 2018 Confidential

```
% used functions :
% version     : v001
% notes       :
% source      : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file
% function h = vcs_tune_change_filters(h,C);
type_str={'lag','lead','1st order lowpass','2nd order lowpass','1st order highpass','notch','2nd
order filter'};
% filtertype = ; %selected before opening

%check if filter popup window is already open
p.init=1;
if isfield(p.tune,'filter_popup');
  if length(p.tune.filter_popup) >= filtertype
    if ishandle(p.tune.filter_popup(filtertype).fg)
      figure(p.tune.filter_popup(filtertype).fg);
      p.init=0;
    end
  end
end

if p.init
  %   rmfield(h.tune,'filter_popup')
  %   for kfilt = 1:7
  %     h.tune.filter_popup(kfilt).fg=[];
  %   end
  nmax = p.tune.nmax(filtertype);
  ncol = ceil(nmax/10);
  p.tune.filter_popup(filtertype).fg =
figure('name',[type_str{filtertype}],'position',p.tune.filter.fg.Position);
  p.tune.filter_popup(filtertype).init = 1;

  %   {'lag','lead','1st order lowpass','2nd order lowpass','1st order highpass','notch','2nd order
filter'};
  if filtertype == 1 | filtertype == 2 %lag /lead
    texts = {'f','deg'};
  elseif filtertype == 3 | filtertype ==5 %1st order lowpass/highpass
    texts = {'flp'};
  elseif filtertype == 4 %2nd order lowpass
    texts = {'flp','damp'};
  elseif filtertype == 6 %notch
    texts = {'f1','f2','b1','b2'};
  elseif filtertype == 7 %2nd order filter
    texts = {'a1','a2','a3','b1','b2','b3'};
  end
```

```
  p.tune.filter_popup(filtertype).txt1 =
uicontrol(p.tune.filter_popup(filtertype).fg,'style','text','units','pixels',...
    'position',[120 260 150 30],'string',[p.tune.filter_names{filtertype},'
Filters'],'fontsize',16,'HorizontalAlignment','Left');
  kcol =1;krow=0;
  for kfilt = 1:nmax
    krow = krow+1;
    if krow>10; krow= 1;kcol=kcol+1;end


    for kedit = 1:length(texts)
      pos = [(kcol-1)*320+10+(65*kedit-1) 240-20*(krow-1) 30 15];
      txt = uicontrol(p.tune.filter_popup(filtertype).fg,'style','text','units','pixels',...
        'position',pos,'string',texts{kedit},'HorizontalAlignment','Left');
      p.tune.filter_popup(filtertype).filter(kfilt).edit(kedit) =
uicontrol(p.tune.filter_popup(filtertype).fg,'style','edit','units','pixels',...
        'position',pos+[30 0 0
0],'string',num2str(eval(['C.',p.tune.filter_fields{filtertype},'(kfilt,kedit)'])));
    end
    p.tune.filter_popup(filtertype).pb_plot(kfilt) =
uicontrol(p.tune.filter_popup(filtertype).fg,'style','pushbutton','units','pixels',...
      'position',pos+[60 0 0 0],'string','plot','Tag',num2str([filtertype
kfilt]),'callback',@plotFilter);

  end
  p.tune.filter_popup(filtertype).pb_update =
uicontrol(p.tune.filter_popup(filtertype).fg,'style','pushbutton','units','pixels',...
    'position',[100,30,100,25],'string','update','callback',@updateFilter);


end
```

plotFilter

```
function plotFilter(source,~)
global p C glo_f plant %#ok<NUSED>
dummy=str2num(source.Tag); %#ok<ST2NM>
filtertype=dummy(1); %#ok<NASGU>
kplot=dummy(2); %#ok<NASGU>
vcs_tune_filter_plot;
end
```

vcs_tune_filter_plot

```
% vcs_tune_filter_plot
%
% Description: This function plots a bode diagram of a filter
% inputs: -kplot = filter number of this type (1-..)
%     -filtertype = 1-7, type
%
%
```


©MECAL 2018 Confidential

```
% file      : vcs_tune_filter_plot.m
% last changes  : 25-07-2017
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

% kplot = ...
% filtertype = ...


%get filter
clear val
val_str=[];

for kval = 1:length(p.tune.filter_popup(filtertype).filter(kplot).edit)
  val(kval) = str2num(get(p.tune.filter_popup(filtertype).filter(kplot).edit(kval),'string'));
  val_str=[val_str,' ',num2str(val(kval))];
end


if filtertype == 1 %lag
   filt = lag(val(2),val(1));
elseif filtertype == 2 %lead
  filt = lead(val(2),val(1));

 elseif filtertype == 3 %1st order lowpass
  filt = tf([val(1)*2*pi],[1 val(1)*2*pi]);
elseif filtertype == 4 %2nd order lowpass
  wlp =val(1)*2*pi;
  filt = tf([wlp^2],[1 2*val(2)*wlp wlp^2]);
elseif filtertype == 5 %1st order highpass
  filt = 1-tf([val(1)*2*pi],[1 val(1)*2*pi]);
  elseif filtertype == 6 %notch
  filt = notch(val(1),val(2),val(3),val(4));
    elseif filtertype == 7 %2nd order filter
  filt = tf(val(1:3),val(4:6));
else
  filt = ss(1);
end

global h
mainScreenSize=h.mainScreen.Position;
if exist('p.tune.filterPlots.fg','var')
  p.tune.filterPlots.fg(end+1)=figure('position',mainScreenSize+[0 mainScreenSize(1,4)/2 -
```


©MECAL 2018 Confidential

```
mainScreenSize(1,3)*2/3 -mainScreenSize(1,4)/2]);
else
  p.tune.filterPlots.fg=figure('position',mainScreenSize+[0 mainScreenSize(1,4)/2 -
mainScreenSize(1,3)*2/3 -mainScreenSize(1,4)/2]);
end
bode(filt), grid on
title([p.tune.filter_names{filtertype},' filter with values = [',val_str,']'],'fontsize',12)
```

updateFilter

```
function updateFilter(~,~)
global p C glo_f plant %#ok<NUSED>
vcs_tune_filter_update;
end
```

saveController

```
function saveController(~,~)
global p C glo_f plant %#ok<NUSED>
vcs_tune_save_controller;
end
```

vcs_tune_save_controller

```
% vcs_tune_save_controller
% Description: save current controller structure 'C' to file
%
%
%
% file      : vcs_tune_save_controller.m
% last changes  : 07-08-2017
% author     : Ronald Rijkers
% used functions :
% version     : v001
% notes      :
% source     : MECAL SPD
% modified by   : -
% copyright    : This file has been developed by MECAL and can not
%          be used, copied of otherwise distributed without written
%          approval from MECAL.
%
% track changes  : v001: initial file

% add extra info in header

%ask user for extra details in controller file

% filename and save
c=clock;
date_str = [date,'_',sprintf('%02.0fh%02.0f',c([4 5]))];
```


©MECAL 2018 Confidential

```
control_fname = ['control_filter_node',num2str(p.node),'_',date_str,'.txt'];

if ~isdir('Controllers')
  mkdir Controllers;
end

%save to text file
fid = fopen(['Controllers\',control_fname],'w');
A=fieldnames(C);
for k = 1:length(A)
  value = eval(['C.',A{k}]);
  if ~strcmp(A{k},'ss') & ~strcmp(A{k},'ssd')
    if isstr(value)
      value_str = value;
    else
      size_val = size(value);
      if size_val(1)==1 & size_val(2)==1
        value_str = num2str(value);
      elseif size_val(1)>1 & size_val(2)==1
        value_str='[';
        for krow = 1:size_val(1);
          value_str = [value_str,num2str(value(krow,1)),';'];
        end
        value_str(end)=']';value_str=[value_str,';'];
      elseif size_val(1)>1 & size_val(2)>1
        value_str='[';
        for krow = 1:size_val(1);
          for kcol = 1:size_val(2);

            value_str = [value_str,num2str(value(krow,kcol)),','];
          end
          value_str(end) = ';';
        end
        value_str(end)=']';value_str=[value_str,';'];
      end
    end

    fprintf(fid,'%s;\n',['C.',A{k},' = ',value_str]);
  end

end
fclose(fid);
```

closeTuneAndSave

```
function closeTuneAndSave(~,~)
global p C h glo_EQ
n=length(glo_EQ)+1;
glo_EQ(n).point=p.node;
glo_EQ(n).transfer=C.frf;
saveController;
```


©MECAL 2018 Confidential

```
closeTune;
editPlan;
% update eq text
if length(glo_EQ)==0 %#ok<ISMT>
  h.EQLoc.String='No EQs';
else
  EQpoints='Equalizers placed on: ';
  for i=1:length(glo_EQ)
    EQpoints=[EQpoints num2str(glo_EQ(i).point) ', ']; %#ok<AGROW>
  end
  h.EQLoc.String=EQpoints(1:end-2);
end
frfAddedElements;
end
```

closeTune

```
function closeTune(~,~)
global p
try %#ok<TRYNC>
  close(p.tune.fg)
  close(p.tune.filter.fg(isvalid(p.tune.filter.fg)))
  close(p.tune.filter_popup.fg(isvalid(p.tune.filter_popup.fg)))
  close(p.tune.filterPlots.fg(isvalid(p.tune.filterPlots.fg)))
end
p=[];
end
```

removeEQ

```
function removeEQ(~,~)
% function to remove EQs
global h glo_EQ

% Stop function if there are no eq to be deleted
if length(glo_EQ)==0 %#ok<ISMT>
  return;
end

% Delete EQ
toDelEQ=str2num(h.removeEQedit.String); %#ok<ST2NM>
glo_EQ(ismember([glo_EQ.point],toDelEQ))=[];

% redraw floor plan
editPlan;

% update eq text
if length(glo_EQ)==0 %#ok<ISMT>
  h.EQLoc.String='No EQs';
else
  EQpoints='Equalizers placed on: ';
```

```
  for i=1:length(glo_EQ)
    EQpoints=[EQpoints num2str(glo_EQ(i).point) ', ']; %#ok<AGROW>
  end
  h.EQLoc.String=EQpoints(1:end-2);
end

frfAddedElements;
end
```

getEQF

```
function EQForce = getEQF(~,~)
% function for plotting EQ forces
global h glo_EQ glo_frf glo_f glo_VibF

if length(glo_EQ)==0 %#ok<ISMT>
  % no EQ exist so no forces can happen
  consoleWrite('No EQs exist, add EQs to use this function');
  return;
else
  % Get sen
  [~,~,frf_old]=frfAddedElements;

  % Calculate controller
  C=complex(zeros(size(glo_frf)));
  for i=1:length(glo_EQ)
    if ~isempty(glo_EQ(i).transfer)
      C(glo_EQ(i).point,glo_EQ(i).point,:)=glo_EQ(i).transfer.*(glo_f*2*pi*1i);
    else
      C(glo_EQ(i).point,glo_EQ(i).point,:)=glo_EQ(i).stiffness;
    end
  end

  % calculate forces
  EQForce=complex(zeros(size(glo_VibF)));
  for i=1:size(glo_VibF,2)

EQForce(:,i)=(eye(size(glo_VibF,1))+C(:,:,i)*frf_old(:,:,i))\C(:,:,i)*frf_old(:,:,i)*glo_VibF(:,i);
  end

  % get plot range
  fRange=str2num(h.EQFrange.String); %#ok<ST2NM>
  fi=find(fRange(1)<=glo_f,1):find(glo_f>=fRange(2),1);

  % plot forces
  for i=1:length(glo_EQ)
    figure('position',h.mainScreen.Position) % new figure
    loglog(glo_f(fi),abs(EQForce(glo_EQ(i).point,fi)))
    title(['EQ forces of EQ on point ' num2str(glo_EQ(i).point)])
    xlabel('Frequency [Hz]');ylabel('EQ Force [N]'); grid on;
  end
```


©MECAL 2018 Confidential

```
end
end
```

runSave

```
function runSave(~,~)
global h glo_floor glo_frf glo_VibF glo_f glo_X glo_Y glo_Coh glo_EQ
% Function for storing current session (system and added elements)

% Get system and added elements
frf=glo_frf; %#ok<NASGU>
f=glo_f; %#ok<NASGU>
X=glo_X; %#ok<NASGU>
Y=glo_Y; %#ok<NASGU>
Coh=glo_Coh; %#ok<NASGU>
VibF=glo_VibF; %#ok<NASGU>
floorVar=glo_floor; %#ok<NASGU>
EQ=glo_EQ; %#ok<NASGU>
massPoints = h.massLoc.String; %#ok<NASGU>
mass = h.mass.String; %#ok<NASGU>
frameLoc1=h.frameLoc1.String; %#ok<NASGU>
frameLoc2=h.frameLoc2.String; %#ok<NASGU>
frameMass=h.frameMass.String; %#ok<NASGU>
frameStiff=h.frameStiff.String; %#ok<NASGU>
stiffPoint=h.stiffPoint.String; %#ok<NASGU>
stiffPointValue=h.stiffPointValue.String; %#ok<NASGU>

% get file name to save to
matfileName=h.saveFileName.String;

% save session
save(matfileName,'frf','f','X','Y','Coh','VibF','floorVar','EQ','massPoints',...
  'mass','frameLoc1','frameLoc2','frameMass','frameStiff','stiffPoint','stiffPointValue');

% Write succes
consoleWrite([matfileName '.mat saved']);
end
```

Appendix L Test case table Matrices estimation Matlab script

```
% Clean matlab
clear
close all
clc

% Parameters
dataFile='tafel3';  % Name of the file the frf and f are stored
floorSize=[1.1 0.4];  % Size of the floor [x y] [m]
gridSize=[6 3];     % Number of grid points [x y] [-]
fRange=[10 300];

% Load data
load(dataFile);

fRangeIndex=find(f>=fRange(1)&f<=fRange(2));

% plot co-located
figure
for i=1:size(frf,1)
    loglog(f(fRangeIndex),abs(frf{i,i}(fRangeIndex)));
    hold on
end
title('Co-located transfers measured');
saveas(gcf,'measured.png')

% matrices estimation method 1

% K
fl=10; % frequency to guess K
fli=find(f>=fl,1);
% get frf matrix at fli
frfl=complex(zeros(size(frf,1)));
for i=1:size(frf,1)
    for j=1:size(frf,2)
        frfl(i,j)=frf{i,j}(fli);
    end
end
K1=inv(abs(frfl));

% M
fh=100; % frequency to guess K
fhi=find(f>=fh,1);
% get frf matrix at fhi
frfh=complex(zeros(size(frf,1)));
```


©MECAL 2018 Confidential

```
for i=1:size(frf,1)
    for j=1:size(frf,2)
        frfh(i,j)=frf{i,j}(fhi);
    end
end
M1=inv(real(-(2*pi*fh)^2*frfl));

% C
% this method does not estimate C so zeros are usesed
C1=zeros(size(K1));

% calc transfer from estimated materices
frfEst1 = matricies2frf(K1,M1,C1,f);

% plot co-located
figure
for i=1:size(frf,1)
    loglog(f(fRangeIndex),abs(squeeze(frfEst1(i,i,fRangeIndex))));
    hold on
end
title('Co-located transfers estimated method 1');
saveas(gcf,'estimated1.png')

% matrices estimation method 2

% K
K2=K1; % same method

% C
f2=20; % frequency to guess C
f2i=find(f>=f2,1);
% get frf matrix at fli
frf2=complex(zeros(size(frf,1)));
for i=1:size(frf,1)
    for j=1:size(frf,2)
        frf2(i,j)=frf{i,j}(f2i);
    end
end
C2=(inv(frf2)-K2)./(2*pi*1i*f2);

% M
f3=100;
frf3=frfh; % same f
M2=((inv(frf3)-K2)./(2*pi*1i*f3)-C2)./(2*pi*1i*f3);

% calc transfer from estimated materices
frfEst2 = matricies2frf(K2,M2,C2,f);

% plot co-located
figure
for i=1:size(frf,1)
    loglog(f(fRangeIndex),abs(squeeze(frfEst2(i,i,fRangeIndex))));
```


©MECAL 2018 Confidential

```
    hold on
end
title('Co-located transfers estimated method 2');
saveas(gcf,'estimated2.png')

% matrices estimation method 2

% get dynamic matrix in a serie of vectors form
Dvec=complex(zeros(length(f),size(frf,1)*size(frf,2)));
for k=1:length(f)
    frff=complex(zeros(size(frf)));
    for i=1:size(frf,1)
        for j=1:size(frf,2)
            frff(i,j)=frf{i,j}(k);
        end
    end
    D=inv(frff);
    Dvec(k,:)=D(:);
end

% get phi for LSE
phi=[-(f*2*pi).^2 (f*2*pi)*1i ones(size(f))];

% Get matrices
materices=(phi\Dvec);

% Reshape materices to correct form
M3=reshape(materices(1,:),size(frf,1),size(frf,2));
C3=reshape(materices(2,:),size(frf,1),size(frf,2));
K3=reshape(materices(3,:),size(frf,1),size(frf,2));

% calc transfer from estimated materices
frfEst3 = matricies2frf(K3,M3,C3,f);

% plot co-located
figure
for i=1:size(frf,1)
    loglog(f(fRangeIndex),abs(squeeze(frfEst3(i,i,fRangeIndex))));
    hold on
end
title('Co-located transfers estimated method 3');
saveas(gcf,'estimated3.png')
```


©MECAL 2018 Confidential