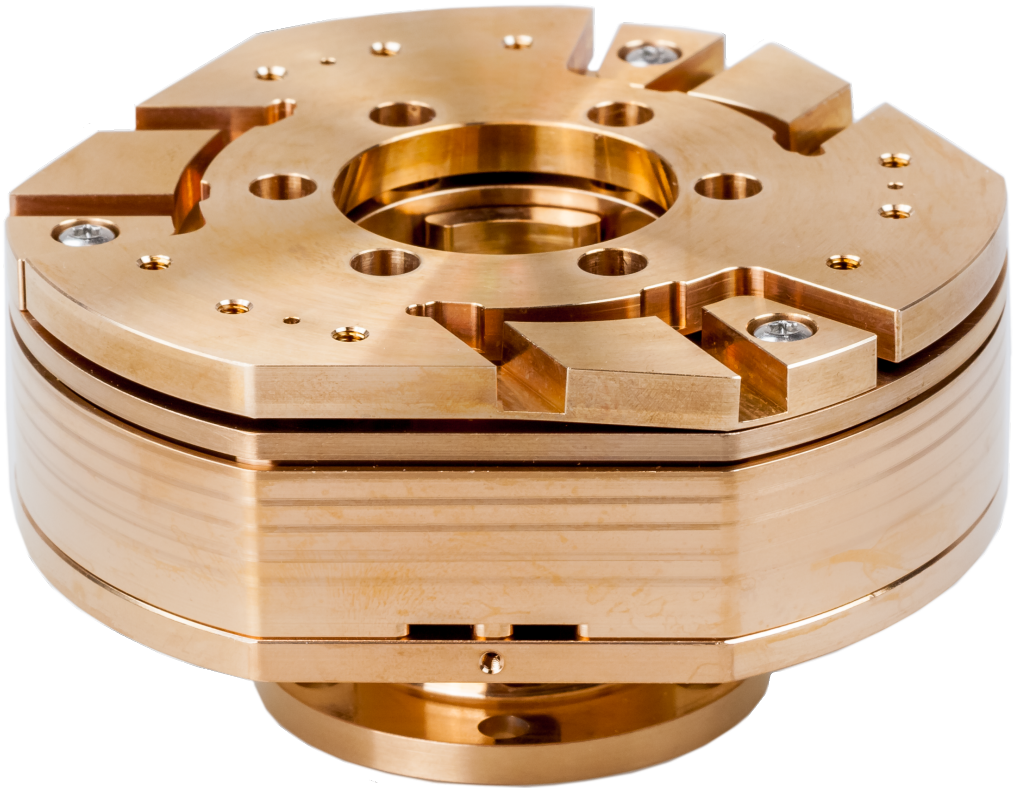
Control for a Cryo Vibration Isolation Platform

Active vibration isolation in 3 degrees of freedom

P.J. Prins

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Control for a Cryo Vibration Isolation Platform

Active vibration isolation in 3 degrees of freedom

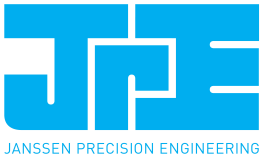
Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

P.J. Prins

March 27, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology



Delft University of Technology
Department of
Delft Center for Systems and Control (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Control for a Cryo Vibration Isolation Platform

by

P.J. Prins

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: March 27, 2017

Supervisor(s):

prof.dr.ir. M. Verhaegen

ir. B.C.T. van Bree

Reader(s):

prof.dr.ir. J.L. Herder

dr.-ing. S. Wahls

dr. S.H. HosseinNia

ir. B. Sinquin

Abstract

Experiments in a closed-cycle cryostat suffer from vibrations caused by its pulse tube. To reduce this disturbance, a vibration isolator needs to be placed inside the cryostat. Since mechanical dampers do not work in a cryogenic environment, it requires active damping. This thesis is about the design of a controller for such an active vibration isolator.

To model the vibration isolator with a limited amount of sensors, sensor roving is applied. The PO-MOESP algorithm for data-driven subspace identification is extended to build a single black box model from the input-output data of a series of sensor roving experiments.

A cautious Wiener filter is proposed to control the vibration isolator.

Table of Contents

1 Introduction **1**

2 System identification **3**

2-1 Noise generating model . . . 3

2-1-1 Available data . . . 3

2-1-2 Noise shaping model . . . 4

2-1-3 Envelope detection . . . 6

2-2 Shaker table and Cryo Vibration Isolation Platform . . . 9

2-2-1 Measurement set-up . . . 9

2-2-2 Model structure . . . 11

2-2-3 Data equation . . . 12

2-2-4 The PO-MOESP method . . . 13

2-2-5 Sensor roving . . . 15

2-2-6 Identification of $(\mathbf{A}, \mathbf{C})$ . . . 22

2-2-7 Identification of $(\mathbf{B}, \mathbf{D})$ . . . 23

2-2-8 From sensor readings to rigid body motion . . . 29

2-2-9 Identification results . . . 33

2-2-10 Proof of concept for PO-MOESP with sensor roving . . . 39

3 Controller design **43**

3-1 Shaker table controller . . . 43

3-2 Cryo Vibration Isolation Platform controller . . . 44

3-2-1 Nominal controller . . . 44

3-2-2 Robust controller without feedback uncertainty . . . 46

3-2-3 Robust feedback controller with feedback uncertainty . . . 48

4 Conclusion and recommendations **51**
4-1 Conclusion . . . 51
4-2 Recommendations . . . 52
4-2-1 Vibration isolator . . . 52
4-2-2 PO-MOESP with sensor roving . . . 53

A Lemmas **55**
A-1 General lemmas . . . 55
A-1-1 General lemmas involving the Moore-Penrose pseudo-inverse . . . 55
A-1-2 General lemmas involving the Kronecker product . . . 55
A-1-3 General lemmas involving Hankel matrices . . . 56
A-1-4 Other general lemmas . . . 60
A-2 Derivations for PO-MOESP with sensor roving . . . 60
A-3 Derivations for Section 2-2-8 . . . 63

B Matlab files **67**
B-1 Supporting Matlab functions . . . 67
B-1-1 Identification . . . 67
B-1-2 State-space algebra . . . 77
B-2 System identification . . . 81
B-2-1 Noise generating model . . . 81
B-2-2 Shaker table and Cryo Vibration Isolation Platform . . . 96

Bibliography **115**

Glossary **117**
List of Acronyms . . . 117
List of Symbols . . . 117

List of Figures

2-1 Block diagram for noise shaping model identification. 4
2-2 First s singular values of the rank-deficient $\mathbf{R}_{32}$ matrix. 5
2-3 Acceleration spectra in x direction. 6
2-4 Autocorrelation and cross correlation of the reconstructed input noise sequences. 7
2-5 Block diagram of the envelope modelling algorithm. 7
2-6 Input noise sequence. 8
2-7 Measurement set-up. 10

3-1 Block diagram for voice coil controller synthesis. 44
3-2 Block diagram of the piezo feedback interconnection. 45
3-3 Block diagram for nominal piezo actuator controller synthesis. 45
3-4 Block diagram for robust piezo actuator controller synthesis. 46
3-5 Equivalent block diagram for robust piezo actuator controller synthesis without feedback uncertainty. 47
3-6 Block diagram for robust piezo actuator controller synthesis with artificial feedback uncertainty paths. 48

List of Tables

2-1 Variance accounted for on validation data when the model is identified using ordinary PO-MOESP with Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$) on only the reference data of the experiment for roving sensor position 8, without voice coil actuation. . . . 34

2-2 Variance accounted for on validation data when the model is identified using ordinary PO-MOESP with Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$) on only the reference data of all sensor roving experiments, without voice coil actuation. 35

2-3 Variance accounted for on validation data when the model is identified with both the reference and roving data of the experiment for roving sensor position 8, without voice coil actuation. 36

2-4 Variance accounted for on validation data when the model is identified using sensor roving PO-MOESP Algorithm IIb with Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$) on all sensor roving experiments, without voice coil actuation. 37

2-5 Variance accounted for on validation data when the model is identified using sensor roving PO-MOESP Algorithm IIb with Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$) on all sensor roving experiments, without voice coil actuation. 38

2-6 Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 2 as a basis and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$). 40

2-7 Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 3 as a basis and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$). 40

2-8 Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 2 as a basis and Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$). 41

2-9 Variance accounted for on validation data when the model is identified using Algorithm IIb (or Algorithm IIa) for ($\mathbf{A}$,$\mathbf{C}$) and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$). . . . 42

2-10 Variance accounted for on validation data when the model is identified using Algorithm IIb (or Algorithm IIa) for ($\mathbf{A}$,$\mathbf{C}$) and Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$). . . . 42

List of Lemmas

Lemma 1 Pseudo-inverse of a product (1) 55
Lemma 2 Pseudo-inverse of a product (2) 55
Lemma 3 Mixed-product property 55
Lemma 4 Transpose of a Kronecker product 56
Lemma 5 Inverse of a Kronecker product 56
Lemma 6 Reshaping a Kronecker product 56
Lemma 7 Rank of a Kronecker product 56
Lemma 8 Pseudo-inverse of a Kronecker product 56
Lemma 9 Product of triangular Hankel matrices 56
Lemma 10 Product of circulant matrices 57
Lemma 11 Product of Hankel matrices 58
Lemma 12 Sylvester's inequality 60
Lemma 13 Instrumental variable matrix for sensor roving PO-MOESP 60
Lemma 14 Derivation of Eq. (2-66) on page 25 61
Lemma 15 Derivation of a pseudo-inverse required in Lemma 16 63
Lemma 16 Derivation of Eq. (2-84) on page 30 64
Lemma 17 Coordinate transformation of an acceleration matrix 65

List of Listings

B.1 blockhankel.m 67
B.2 block_hankel_permutation.m 69
B.3 block_vectorise.m 69
B.4 circulant_multiply.m 70
B.5 dinit_batch.m 71
B.6 filter_batch.m 71
B.7 global_obsv_permutation.m 72
B.8 hankel_gram.m 73
B.9 local2global.m 74
B.10 lsim_batch.m 75
B.11 obsv_ext.m 76
B.12 perfect_shuffle.m 76
B.13 vaf_batch.m 77
B.14 sscausal_adleft.m 78
B.15 sscausal_adright.m 78
B.16 ssIO.m 79
B.17 ssOI.m 79
B.18 sspinv.m 80
B.19 cold_plate_data_identification.m 81

B.20 artificial_cold_plate_noise.m 95
B.21 compress_PO_MOESP.m 96
B.22 svd_PO_MOESP.m 97
B.23 algorithm_Ia.m 99
B.24 algorithm_IIa.m 100
B.25 algorithm_IIb.m 101
B.26 mdac2bd.m 102
B.27 identification_step_by_step_pa.m 109
B.28 identification_proof_of_concept.m 111

Chapter 1

Introduction

Janssen Precision Engineering (JPE) develops motion stages for cryogenic applications for which nanometer level accuracy and stability is required. When these motion stages are mounted in the cryostat on the cold plate, vibrations of the cold plate have a negative impact on the stability of the positioning. The cryostat can be mounted on top of a vibration isolator to prevent vibrations caused by the outside world from reaching the cold plate. For a continuous flow cryostat this may suffice, but this type of cryostat is costly to operate, because liquid cryogens evaporate and need replenishment. Closed cycle cryostats work similarly to an ordinary refrigerator and are cheaper in use, as the cryogen is continuously reused. Unfortunately, the pulse tube that pumps the cryogen through the closed cycle, vibrates. Because the cold plate needs to have a rigorous mechanical connection to the cryogen circuit, the cold plate cannot be isolated from these vibrations.

As an alternative, JPE has designed vibration isolators that can be placed in a cryostat, between the cold plate and a positioning stage. A problem with this approach is that there are no mechanical dampers that work in a cryogenic environment. Hence, a passive vibration isolator in a cryostat, basically a mass-spring system, suffers from resonating behaviour. To solve this issue, the Cryo Vibration Isolation Platform 2 (CVIP 2) is equipped with piezo sensors and actuators. The aim is to control these electro mechanical transducers actively to behave like a mechanical damper in order to damp the resonant modes.

A CVIP 2 consists of two layers. The bottom layer is a purely passive mechanical structure with three springs. Together, these form a joint with three degrees of freedom: A vertical piston motion and both joystick motions. The second layer consists of three stacks of two piezo elements each, where one is used as a sensor and the other one as an actuator. This layer can be seen as a purely active vibration isolator that allows the same degrees of freedom as the first layer. If the mass between the layers is neglected and the mechanically collocated sensor actuator pairs are controlled to behave like a mechanical damper, the

result is a mass-spring-damper system with the dampers in series with the springs. The reason for the latter is that the stiffness and limited range of the piezo elements do not allow mounting them in parallel to the springs. The idea behind this design is that the passive layer has a transmissibility like a second order low pass filter in the degrees of freedom of interest. This includes the vertical direction, of which the stability is most important to many customers. The cutt-off frequency of the passive modes is between 15 Hz and 30 Hz and the only job of the active layer is to damp these resonant modes.

The aim for the project this thesis is about, is to design a controller for the active layer, to evaluate its working in ambient conditions — including the impact on the kHz range — and to extrapolate the results to cryogenic conditions.

The performance of a vibration isolator can be characterised with its transmissibility, the transfer function between the movement of its base and the movement of its payload. To calculate it, one needs to measure how these two rigid bodies are moving. All six degrees of freedom for rigid body motion need to be taken into account, because it would be undesirable to damp the movement in one degree of freedom but increase it simultaneously in another that is not measured. Hence, the transmissibility is in this case a six by six transfer function. The optimal transmissibility for a vibration isolator depends on (the spectrum of) the floor vibrations, since lowering the transmissibility in one frequency band, increases it in another. It can be shown that a transmissiblity that decreases the payload motion for one floor vibration spectrum, increases it for another [13]. A noise generating model for the floor vibrations therefore plays an important role in the design of a controller for active vibration isolation.

Chapter 2

System identification

This chapter consists of two sections. The first section is about the identification of a noise generating model that describes the vibrations of a cold plate. This is done using Past Outputs Multivariable Output-Error StatesPace (PO-MOESP) [14, 16] and periodic envelope detection, to model the relevant frequency and time domain behaviour respectively. The second section concerns the identification of the plant, consisting of a Cryo Vibration Isolation Platform 2 (CVIP 2) that is mounted on top of a shaker table. For that purpose the PO-MOESP algorithm was extended for the use of roving sensor data.

2-1 Noise generating model

This section is about the identification of a noise generating model that is needed both for controller synthesis and testing. It is organised in the order of the signal processing steps. Listing B.19 on page 81 shows a MATLAB script that performs the steps described below. Listing B.20 on page 95 shows a script to use the obtained model to generate artificial cold plate vibrations.

2-1-1 Available data

The available cold plate vibration data, was recorded earlier with a single linear accelerometer triad, a Kistler 8688A5, SN2155905. It was rigidly attached to the cold plate of a working Montana cryostation. At a temperature of 280 K, 10 s of data was recorded at a sampling frequency of 20 kHz. Since the measurements include only the 3 linear degrees of freedom (DOF), the rotational acceleration of the cold plate is unknown. The measurement was limited to a few seconds to limit the possible time variant behaviour due to the decreasing temperature. Because the accelerometer was not calibrated for

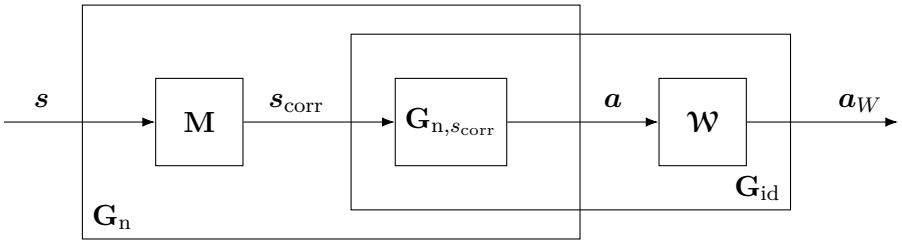


Figure 2-1: Block diagram for noise shaping model identification.

cryogenic temperatures, no reliable measurements could be done at a realistic temperature set point. Nevertheless the measurements in this particular cryostat are assumed to be representative for all cryostats in their ordinary working conditions.

2-1-2 Noise shaping model

The first step is to identify a noise shaping model $\mathbf{G}_\mathrm{n}$. This is a model that, when given independent unit covariance white noise sequences as an input, returns a sequence that approximates the spectrum of the recorded cold plate acceleration in three DOF. This model is meant to be used later on for controller synthesis. Furthermore this model can be used to test the open and closed loop system for a longer duration than the 10 s of recorded data. The identification procedure is illustrated in Figure 2-1 and will be explained below.

Data pre-conditioning

The measurement data is first converted from V to $\mathrm{m/s^2}$ using the calibration supplied by the sensor manufacturer. Next, the data preprocessing steps described in [17, §4.3] are considered. Accordingly, the data is detrended to remove most of the drift. Shaving is omitted, because the data seems not to contain artefacts that need such removal. Initially the data was not decimated: The CVIP 2 is supposed to act as a mechanical low-pass filter with a cut-off frequency below 100 Hz, depending on the payload mass, but there may be unforeseen resonances in the kHz range and hence a noise generating model at 20 kHz is of interest. However, identification for this full bandwidth turned out to be unsatisfactory. Finally the data was resampled at 5 kHz.

Output weighting

A frequency spectrum of the output data (see Figure 2-3 on page 6) reveals that its minimum is roughly between 40 Hz and 200 Hz and that there is much more power at higher frequencies. Matching the frequency spectrum below 100 Hz is important because of the expected transmissibility of the CVIP 2, but the subspace identification method tends to match primarily the frequency bands where the signal power is high. The

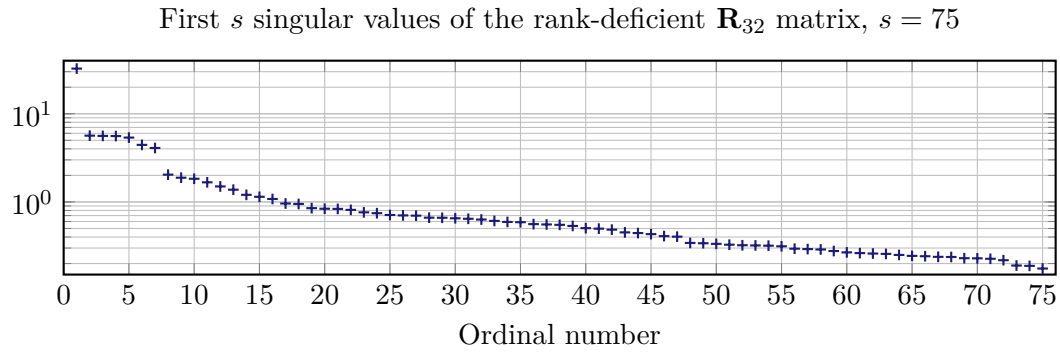


Figure 2-2: First s singular values of the rank-deficient $\mathbf{R}_{32}$ matrix, $s = 75$.

frequency band of interest can be emphasised by prefiltering the data, but because we are dealing with output only identification, the dynamics of this weighting filter will end up in the noise generating model, while these dynamics are generally unrelated to the noise characteristic itself: Referring to Figure 2-1, the sequence $\boldsymbol{a}$ represents the acceleration data and passing it through a diagonal minimum phase weighting filter $\boldsymbol{\mathcal{W}}$ yields the sequence $\boldsymbol{a}_{\mathcal{W}}$. The model that we identify from the latter sequence is $\mathbf{G}_{\mathrm{id}}$. A model to generate the unfiltered sequence $\boldsymbol{a}$ is then found as

$$\mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}} = \boldsymbol{\mathcal{W}}^{-1}\mathbf{G}_{\mathrm{id}}\,, \tag{2-1}$$

from which we see that the filter zeros will appear as poles in the noise generating model, while if the relative degree of the weighting filter is larger than zero, the noise generating model is possibly improper. Hence, it is desirable to keep the order of the weighting filter low. Acceptable results were obtained with a first order filter on each channel with a passband below 100 Hz and a zero just below the Nyquist frequency, yielding a maximum attenuation of 25 dB. Figure 2-3 shows the weighting filter that is applied to each channel.

PO-MOESP identification

Using PO-MOESP [14, 16] a seventh order model $\mathbf{G}_{\mathrm{id}}$ was identified. Figure 2-2 shows the singular values upon which the model order was selected. Removing the effect of the weighting filter $\boldsymbol{\mathcal{W}}$ according to Eq. (2-1) yields a tenth order model $\mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}}$. Figure 2-3 shows the spectrum of the measured acceleration in the x direction, the spectrum generated with this tenth order model and — as an illustration of the effect of the weighting filter — the spectrum generated with an eighteenth order model without using a weighting filter.

Input noise correlation

The model $\mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}}$ generates a vibration spectrum from a white noise sequence $\boldsymbol{s}_{\mathrm{corr}}$ (see Figure 2-1), but its channels are not necessarily independent. If dependence exists, it

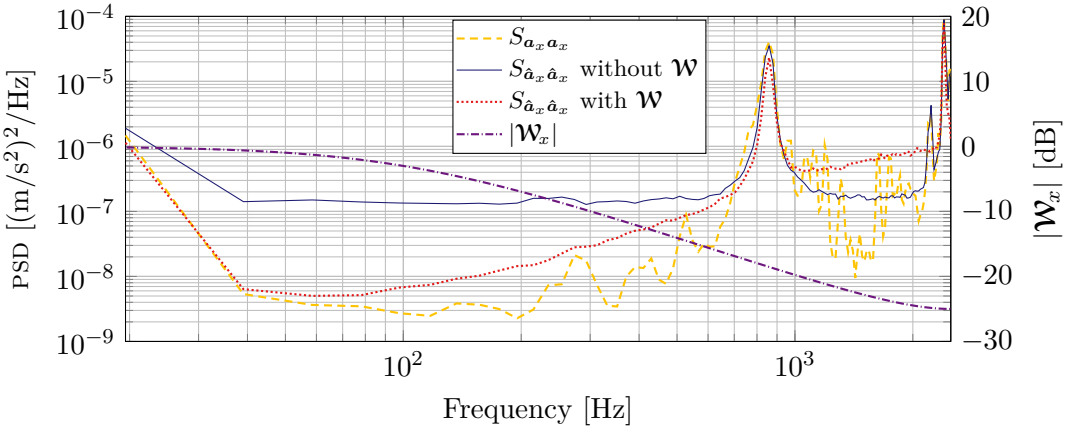


Figure 2-3: Acceleration spectra in x direction: Measured data (----), generated with an 18^{th} order model without output weighting (——), generated with a 10^{th} order model with output weighting (·····); applied weighting filter (-·-·-, right vertical axis).

can be a source of error if left unmodelled. Because $\mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}}$ turned out to be invertible, this was easily verified with

$$\boldsymbol{s}_{\mathrm{corr}} = \mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}}^{-1}\boldsymbol{a} \tag{2-2}$$

and subsequently estimating the cross correlation between the channels of $\boldsymbol{s}_{\mathrm{corr}}$. These channels were decomposed into independent noise sequences with a principle component analysis and subsequently normalised to unit covariance, resulting in the sequence $\boldsymbol{s}$ and the static gain $\mathbf{M}$. Figure 2-4 shows the auto and cross correlation of the input noise sequences before and after this principle component decomposition. From these plots it can be seen that there was a correlation between the input sequences, especially $\hat{R}_{\boldsymbol{s}_1\boldsymbol{s}_3}$, which is reduced by the principle component decomposition. The resulting noise shaping model finally reads as

$$\mathbf{G}_{\mathrm{n}} = \mathbf{G}_{\mathrm{n},s_{\mathrm{corr}}}\mathbf{M}\,. \tag{2-3}$$

2-1-3 Envelope detection

The noise input signal $\boldsymbol{s}$ found in Section 2-1-2 is white and independent, but not stationary Gaussian. It has a clear envelope according to the pumping of the pulse tube, which is shown in Figure 2-6. We will model this periodic envelope with its Fourier coefficients. This envelope is only meant to make the tests on the shaker table more realistic: During the peaks of the envelope, physical limits of the system may be reached, which we would not notice if we were using a signal with the average covariance for testing.

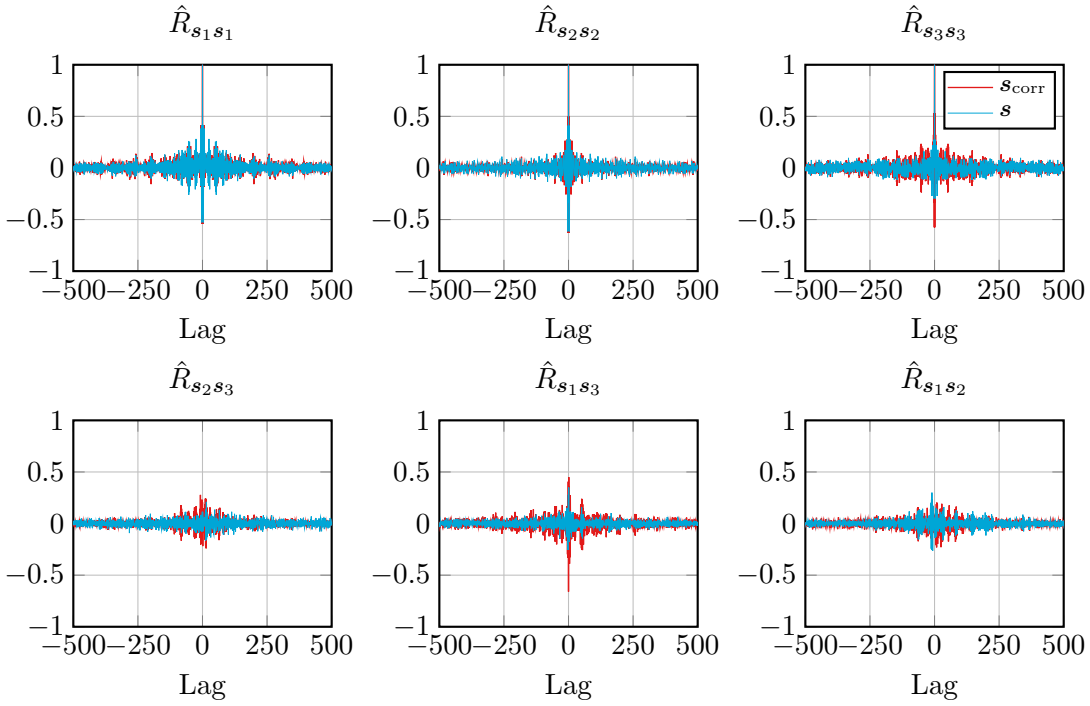


Figure 2-4: Autocorrelation and cross correlation of the reconstructed input noise sequences before (——) and after (——) principle component decomposition, all signals scaled to unit standard deviation.

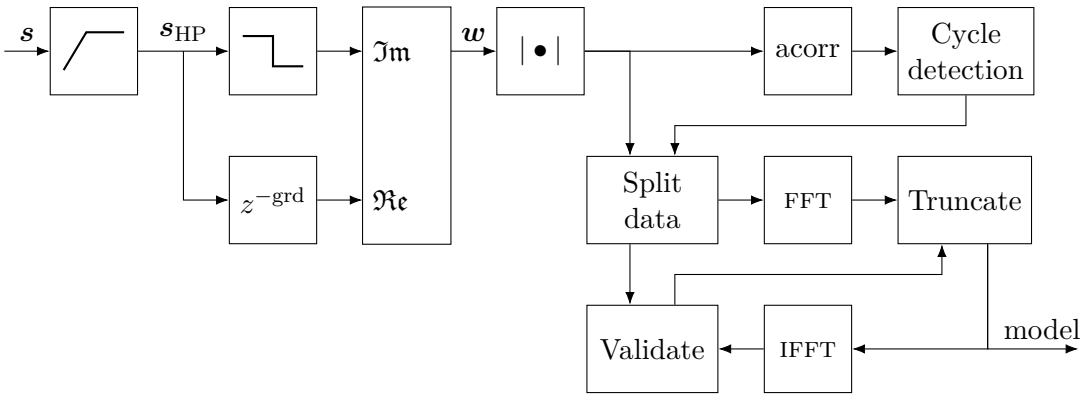


Figure 2-5: Block diagram of the envelope modelling algorithm.

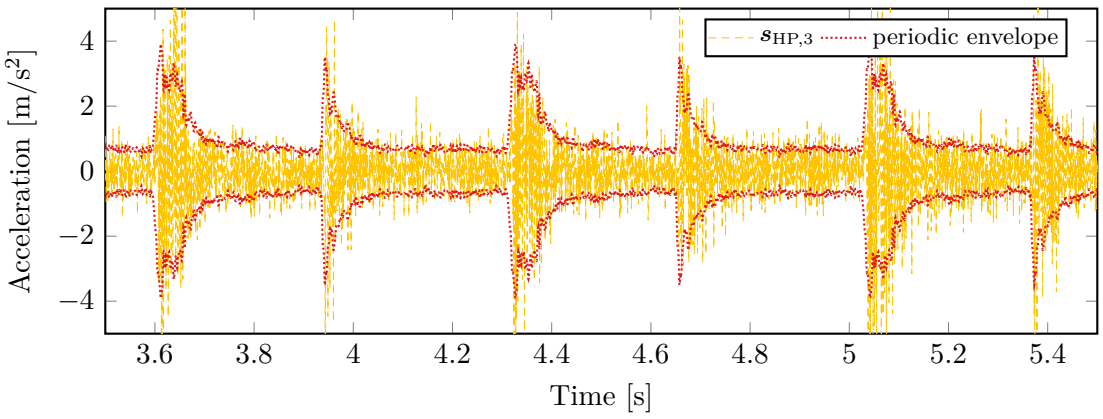


Figure 2-6: Input noise sequence on the third channel for the measured data according to the identified model (- - -) and its identified periodic envelope (·····).

The algorithm illustrated in Figure 2-5 and described below is based on [8, Method 2] and adapted to exploit the periodicity of the envelope.

Analytic signal estimation

The algorithm described in [8, Method 2] is based on the observation that an envelope is the magnitude of an analytic signal. The analytic signal $\boldsymbol{w}$ is estimated by using a Hilbert transform estimation signal to form the imaginary part and delaying the real part according to the group delay of the aforementioned filter. As this algorithm appears to be very sensitive to drift, the signal is high-pass filtered first. The original algorithm ends with a low-pass filter to obtain a smooth envelope. Instead we look for a truncated order discrete Fourier transform (DFT) representation of the periodic envelope. Because the high frequency components do not follow this periodicity, there is no need to remove them.

Truncated order discrete Fourier transform representation

In order to find a DFT representation of the periodic envelope, we first estimate the periodicity, the inverse of the ground frequency of the envelope. By choosing the fast Fourier transform (FFT) window as an integer multiple of the periodicity, all periodic components will exactly fit in a frequency bin. This periodicity is found as the distance between adjacent peaks in the autocorrelation of the envelope.

We take about 2/3 of the data, containing an integer number of cycles, as training data. The remainder is used for validation. We take the FFT of the training data to obtain a DFT representation and set all coefficients that do not correspond to the number of cycles in this data to zero. Next, we truncate this DFT representation at the order that

maximizes the variance accounted for (VAF) with respect to the validation data. The resulting envelope for one of the three channels is shown in Figure 2-6.

2-2 Shaker table and Cryo Vibration Isolation Platform

This section is about the identification of the plant: A CVIP 2 on a pedestal that is mounted on top of a shaker table. These two cannot be modelled separately, because the shaker table has little mass compared to the CVIP 2 and its payload together. Hence, the common assumption that a vibration isolator does not influence the floor vibrations cannot be made. The goal here is to identify a model with both the voice coil actuation of the shaker table and the piezo actuation of the CVIP 2 as inputs and as outputs the acceleration of the shaker table, the acceleration of the payload of the CVIP 2 and the piezo sensor output. The accelerations will be the six DOF rigid body acceleration evaluated in two points: The centre of the pedestal bottom for the shaker table movement, the point that moves rigidly with the cold plate in real operation; and for the payload a point along the centre axis of the CVIP 2 top at a user specified height, a point of interest (POI).

Because there are not enough sensors available to measure both accelerations simultaneously, sensor roving will be used. This is a procedure in which multiple identification experiments are performed on the same plant with the sensors in different positions to mimic a single experiment with enough sensors to populate all the positions at once. The data from these different experiments can only be combined if a part of the sensors is left in the same position. These reference sensors need to be chosen such that the plant is observable from these sensors [13]. A large portion of this section and the companioning MATLAB scripts will deal with the adaptation of the PO-MOESP identification algorithm to this measurement procedure.

2-2-1 Measurement set-up

Figure 2-7 shows a picture of the measurement set-up and schematic drawings of the available sensor positions. The bottom part is a shaker table that is especially designed by Janssen Precision Engineering (JPE) for this purpose. Its top rests on a leave spring and can be moved using three voice coil actuators mounted below the shaker table top. During the experiments it appeared that the movement of the shaker table top was too dominant: There was a significant influence of the piezo actuators on the shaker table and the resonant modes of the shaker table were in the same frequency band as those of the CVIP 2. It would not have been realistic to reduce this by increasing the mass of the shaker table top. Instead the nuts that were only supposed to restrict the maximum movement of the shaker table were tightened. Effectively this makes the shaker table top rest on a much stiffer spring formed by the original leave spring and three metal rods. In Figure 2-7 the nuts are shown in the untightened position.

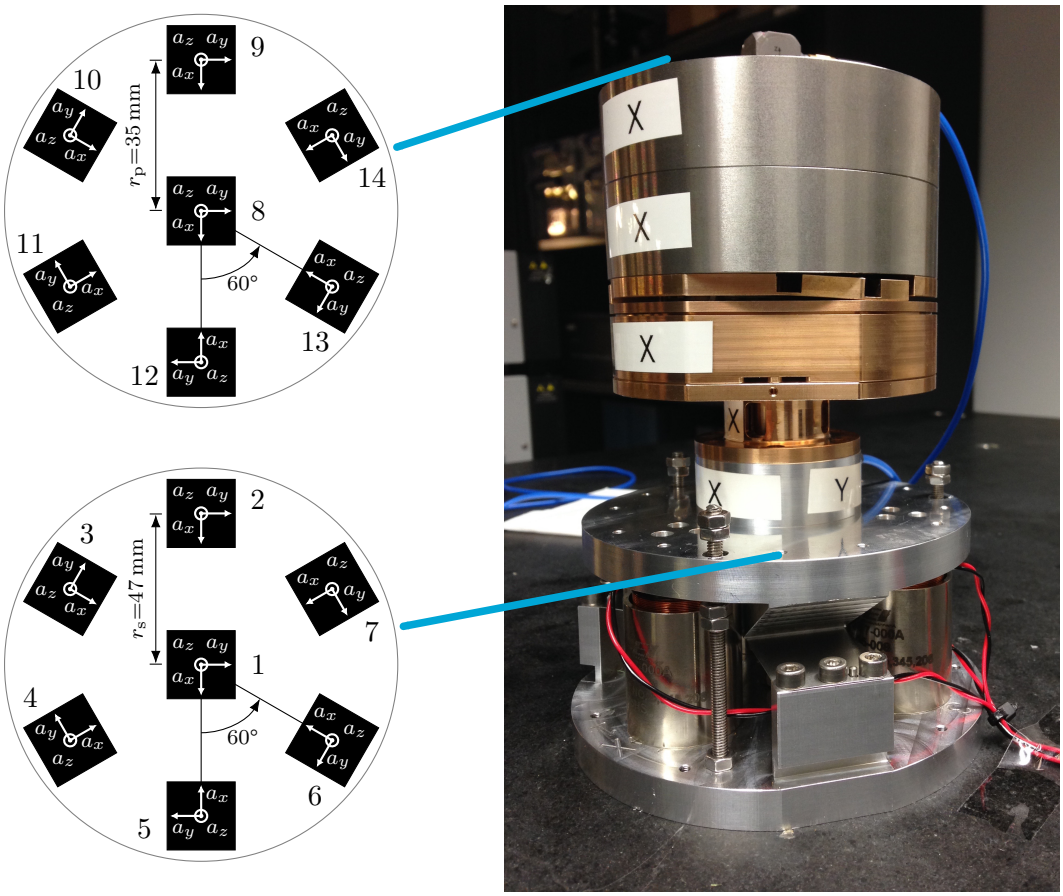


Figure 2-7: Measurement set-up.

The shaker table top has mounts for accelerometer triads at seven positions. These mounts make the sensor placement more accurate and reproducible. On top of it is the CVIP 2 on a pedestal. Two dummy masses stand in for the payload. The topmost dummy mass has another seven sensor mounts. The whole set-up is placed on top of a vibration isolating laboratory table.

Two Kistler 8688A5 accelerometer triads are available (SN2155905 and SN4973067). All sensors and actuators are connected to a single computer and are used with Simulink real time. The sensor SN2155905 is permanently mounted in position 1 and forms together with the piezo sensors in de CVIP 2 the reference sensor pool. The other sensor is consecutively mounted on each of the other sensor positions. Using the data from all these experiments together to mimic the availability of a fully occupied sensor grid is known as sensor roving. Choosing position 1 for the reference sensor position seems suitable from an algorithmic point of view: Judging from a simple one DOF (z only) model of this set-up, a sensor on the shaker table yields better observability than a sensor on the payload. Furthermore it is a pragmatic choice, because everything on top of the shaker table needs to be unscrewed to move this sensor, which is not necessary for any of the other sensor positions.

2-2-2 Model structure

We will consider as a model structure a linear time-invariant (LTI) discrete-time state space model with input and output noise:

$$\begin{cases} \boldsymbol{x}(k+1) = \mathbf{A}\boldsymbol{x}(k) + \mathbf{B}\boldsymbol{u}(k) + \boldsymbol{s}(k)\,, \\ \quad \boldsymbol{y}_0(k) = \mathbf{C}_0\boldsymbol{x}(k) + \mathbf{D}_0\boldsymbol{u}(k) + \boldsymbol{v}_0(k)\,, \\ \qquad\quad \vdots \\ \quad \boldsymbol{y}_J(k) = \mathbf{C}_J\boldsymbol{x}(k) + \mathbf{D}_J\boldsymbol{u}(k) + \boldsymbol{v}_J(k)\,, \end{cases} \tag{2-4}$$

$$\text{where} \quad \mathbf{A} \in \mathbb{R}^{n\times n}\,, \quad \mathbf{B} \in \mathbb{R}^{n\times m}\,, \quad \mathbf{C}_0 \in \mathbb{R}^{\ell_0\times n}\,,$$
$$\mathbf{C}_j \in \mathbb{R}^{\ell_j\times n}\,, \quad \mathbf{D}_0 \in \mathbb{R}^{\ell_0\times m}\,, \quad \mathbf{D}_j \in \mathbb{R}^{\ell_j\times m}\,.$$

The output sequence $\boldsymbol{y}_0$ is the output from the reference sensor pool: the fixed accelerometer triad and the piezo sensors. The remaining sequences $\boldsymbol{y}_j$ with $j \in \{1, 2, \ldots, J\}$ are the roving sensor outputs. The roving sensor output consists of the data of one accelerometer triad. During each experiment only one of these outputs is recorded. For simplicity the experiments are numbered according to the roving sensor output for that experiment. The superscript j can hence both refer to this experiment and to the roving sensor in the position it has during the j-th experiment. The index $j = 0$ refers to the reference sensors. The conversion from sensor outputs to rigid body motion will be postponed till Section 2-2-8 on page 29.

This model structure assumes that all measurements are associated to the same process.[1] Each sensor position is associated with a different output mapping $(\mathbf{C}_j, \mathbf{D}_j)$. One could think of it as using a full sensor grid of which each time only a part of the output data — the sequences $\boldsymbol{y}_0$ and $\boldsymbol{y}_j$ — is recorded.

The model in Eq. (2-4) can be rewritten in innovation form [16, §9.6]:

$$\begin{cases} \boldsymbol{x}(k+1) = \mathbf{A}\boldsymbol{x}(k) + \mathbf{B}\boldsymbol{u}(k) + \mathbf{K}\left[\boldsymbol{e}_0^\top(k) \ \cdots \ \boldsymbol{e}_J^\top(k)\right]^\top, \\ \quad \boldsymbol{y}_0(k) = \mathbf{C}_0\boldsymbol{x}(k) + \mathbf{D}_0\boldsymbol{u}(k) + \boldsymbol{e}_0(k), \\ \qquad \vdots \\ \quad \boldsymbol{y}_J(k) = \mathbf{C}_J\boldsymbol{x}(k) + \mathbf{D}_J\boldsymbol{u}(k) + \boldsymbol{e}_J(k), \end{cases} \tag{2-5}$$

$$\text{where } \mathbf{K} \in \mathbb{R}^{n\times(\ell_0+J\ell_j)}.$$

We will assume that all experiments will be done using the same input sequence. This is not a necessary limitation, but if there is no reason to use different input sequences, it saves some computations and superscripts. Furthermore we will assume that the number of roving sensors is equal for all experiments. Again this is not a necessary limitation, but it greatly simplifies some formulas and it will usually be satisfied anyway. The number of reference sensors does not necessarily equal the number of roving sensors.

2-2-3 Data equation

In the following subsections we will discuss a single sensor roving experiment. For a single experiment, PO-MOESP [14, 16] can be used without adaptation. The only difference here is that we make an explicit separation between the reference and roving outputs, which will allow us to combine the data from different experiments in Section 2-2-5 on page 15.

Based on the model structure in Eq. (2-5), the data equation of a single experiment can be denoted as

$$\begin{bmatrix} \mathbf{Y}_{i,s,N}^{(0,j)} \\ \mathbf{Y}_{i,s,N}^{(j,j)} \end{bmatrix} = \begin{bmatrix} \mathfrak{O}_s^{(0)} \\ \mathfrak{O}_s^{(j)} \end{bmatrix} \mathbf{X}_{i,N}^{(j)} + \begin{bmatrix} \mathfrak{T}_s^{(0)} \\ \mathfrak{T}_s^{(j)} \end{bmatrix} \mathbf{U}_{i,s,N} + \begin{bmatrix} \mathfrak{S}_s^{(0)} \\ \mathfrak{S}_s^{(j)} \end{bmatrix} \mathbf{E}_{i,s,N}^{(j)}, \tag{2-6}$$

where the appearing matrices are defined as follows. The output sequences of the j^{th} experiment are represented as Hankel matrices

$$\mathbf{Y}_{i,s,N}^{(k,j)} := \begin{bmatrix} \boldsymbol{y}_k(i) & \boldsymbol{y}_k(i+1) & \cdots & \boldsymbol{y}_k(i+N-1) \\ \boldsymbol{y}_k(i+1) & \boldsymbol{y}_k(i+2) & \cdots & \boldsymbol{y}_k(i+N) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{y}_k(i+s-1) & \boldsymbol{y}_k(i+s) & \cdots & \boldsymbol{y}_k(i+N+s-2) \end{bmatrix} \in \mathbb{R}^{s\ell_k\times N}. \tag{2-7}$$

[1] This assumption can be easily violated if the sensor, when placed at different positions, significantly interferes with the process.

The input and noise sequences are similarly represented as Hankel matrices, respectively $\mathbf{U}_{i,s,N} \in \mathbb{R}^{sm\times N}$ and $\mathbf{E}^{(j)}_{i,s,N} \in \mathbb{R}^{s(\ell_0+\ell_j)\times N}$. The state sequence is juxtaposed to

$$\mathbf{X}^{(j)}_{i,N} \coloneqq \begin{bmatrix} \boldsymbol{x}(i) & \boldsymbol{x}(i+1) & \cdots & \boldsymbol{x}(i+N-1) \end{bmatrix} \in \mathbb{R}^{n\times N}\,. \tag{2-8}$$

The state space model matrices are represented in the remaining definitions. Two of these symbols double as a function, an abuse of notation that eases the representation of some equations in the sequel:

$$\mathfrak{O}^{(k)}_s = \mathfrak{O}_s(\mathbf{A},\mathbf{C}_k) \coloneqq \begin{bmatrix} \mathbf{C}_k \\ \mathbf{C}_k\mathbf{A} \\ \vdots \\ \mathbf{C}_k\mathbf{A}^{s-1} \end{bmatrix} \in \mathbb{R}^{s\ell_k\times n}\,, \tag{2-9}$$

$$\mathfrak{T}^{(k)}_s = \mathfrak{T}_s(\mathbf{A},\mathbf{B},\mathbf{C}_k,\mathbf{D}_k) \coloneqq \begin{bmatrix} \mathbf{D}_k & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{C}_k\mathbf{B} & \mathbf{D}_k & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{C}_k\mathbf{A}\mathbf{B} & \mathbf{C}_k\mathbf{B} & \mathbf{D}_k & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{C}_k\mathbf{A}^{s-2}\mathbf{B} & \mathbf{C}_k\mathbf{A}^{s-3}\mathbf{B} & \cdots & \mathbf{C}_k\mathbf{B} & \mathbf{D}_k \end{bmatrix} \in \mathbb{R}^{s\ell_k\times sm}\,. \tag{2-10}$$

Finally

$$\mathfrak{S}^{(0)}_s \coloneqq \mathfrak{T}_s\Big(\mathbf{A},\mathbf{K},\mathbf{C}_j,\begin{bmatrix}\mathbf{I}_{\ell_0} & \mathbf{0}_{\ell_0\times\ell_j}\end{bmatrix}\Big)\;; \qquad \mathfrak{S}^{(j)}_s \coloneqq \mathfrak{T}_s\Big(\mathbf{A},\mathbf{K},\mathbf{C}_j,\begin{bmatrix}\mathbf{0}_{\ell_j\times\ell_0} & \mathbf{I}_{\ell_j}\end{bmatrix}\Big)\,. \tag{2-11}$$

Compared to the ordinary data equation, the only difference is that the rows of Eq. (2-6) are permuted, so the whole equation is multiplied from the left by a permutation matrix, which does not change its mathematical meaning.

2-2-4 The PO-MOESP method

The goal of PO-MOESP is to estimate a state-space model (Eq. (2-5)) given output and — if applicable — input data. A silent assumption is that the system of interest fits this model structure (at least approximately), such that Eq. (2-6) holds. Furthermore the noise sequences $\boldsymbol{e}$ are assumed to be white and ergodic and uncorrelated to the ergodic input sequences $\boldsymbol{u}$ and [16, Eqs. 9.58 & 9.59]

$$\operatorname{rank}\left(\lim_{N\to\infty}\frac{1}{N}\begin{bmatrix}\mathbf{X}^{(j)}_{s,N}\\ \mathbf{U}_{s,s,N}\end{bmatrix}\begin{bmatrix}\mathbf{Y}^{(0,j)}_{0,s,N}\\ \mathbf{Y}^{(j,j)}_{0,s,N}\\ \mathbf{U}_{0,2s,N}\end{bmatrix}^\top\right) = n+sm \quad \text{(full row rank)}\,, \tag{2-12}$$

$$\operatorname{rank}\left(\lim_{N\to\infty}\frac{1}{N}\begin{bmatrix}\mathbf{X}^{(j)}_{0,N}\\ \mathbf{U}_{0,2s,N}\end{bmatrix}\begin{bmatrix}\mathbf{X}^{(j)}_{0,N}\\ \mathbf{U}_{0,2s,N}\end{bmatrix}^\top\right) = n+2sm \quad \text{(full rank)}\,. \tag{2-13}$$

In Eq. (2-12) $\mathbf{Y}_{0,s,N}^{(j,j)}$ needs to be left out if only the reference sensor data is used in the PO-MOESP algorithm, which will be the case in Algorithm IIa/b on pages 20 and 21. These assumptions cannot be checked on beforehand. They usually hold in practice if the input sequence is persistently exciting and if there is no feedback from the measured output to the input.

The contribution of the input and the noise can be removed from Eq. (2-6) with an orthogonal projection matrix $\mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp}$ and an instrumental variable matrix $\mathbf{Z}_N^{(j)}$ that are defined as follows.

$$\mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \coloneqq \mathbf{I}_N - \mathbf{U}_{s,s,N}^{\top}\left(\mathbf{U}_{s,s,N}\mathbf{U}_{s,s,N}^{\top}\right)^{-1}\mathbf{U}_{s,s,N}\,, \quad (2\text{-}14) \quad \mathbf{Z}_N^{(j)} \coloneqq \begin{bmatrix} \mathbf{U}_{0,s,N} \\ \mathbf{Y}_{0,s,N}^{(0,j)} \\ \mathbf{Y}_{0,s,N}^{(j,j)} \end{bmatrix}. \quad (2\text{-}15)$$

The useful properties of these matrices are

$$\begin{cases} \mathbf{U}_{s,s,N} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} = \mathbf{0}\,, \\ \lim\limits_{N\to\infty} \dfrac{1}{N}\mathbf{E}_{s,s,N}^{(j)} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top} = \mathbf{0}\,, \\ \operatorname{range}\left(\lim\limits_{N\to\infty} \dfrac{1}{N}\begin{bmatrix} \mathbf{Y}_{s,s,N}^{(0,j)} \\ \mathbf{Y}_{s,s,N}^{(j,j)} \end{bmatrix} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top}\right) = \operatorname{range}\left(\begin{bmatrix} \mathbf{\mathcal{O}}_s^{(0)} \\ \mathbf{\mathcal{O}}_s^{(j)} \end{bmatrix}\right). \end{cases} \quad (2\text{-}16)$$

The difference between Eq. (2-15) and the original PO-MOESP implementation is again the partitioning between the reference sensors and the roving sensors. Notwithstanding this change, $\mathbf{Z}_N^{(j)}$ is still a valid instrumental variable, as shown in the proof of Lemma 13 on page 60. By multiplying Eq. (2-6) from the right, we obtain

$$\lim_{N\to\infty} \frac{1}{N}\begin{bmatrix} \mathbf{Y}_{s,s,N}^{(0,j)} \\ \mathbf{Y}_{s,s,N}^{(j,j)} \end{bmatrix} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top} = \lim_{N\to\infty} \frac{1}{N}\begin{bmatrix} \mathbf{\mathcal{O}}_s^{(0)} \\ \mathbf{\mathcal{O}}_s^{(j)} \end{bmatrix} \cdot \mathbf{X}_{s,N}^{(j)} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top}. \quad (2\text{-}17)$$

This calculation can be done effectively using an RQ-factorisation [14–16]. For the extension of this algorithm to sensor roving (especially the estimation of the pair $(\mathbf{B}, \mathbf{D})$, see Section 2-2-7 on page 23), it is useful to work with a finer partitioning, given by

$$\begin{array}{c} \\ \langle sm\rangle \\ \langle sm\rangle \\ \langle s\ell_0\rangle \\ \langle s\ell_j\rangle \\ \langle s\ell_0\rangle \\ \langle s\ell_j\rangle \end{array} \begin{array}{c} \langle N\rangle \\ \left[\begin{array}{c} \mathbf{U}_{s,s,N} \\ \hdashline \mathbf{U}_{0,s,N} \\ \mathbf{Y}_{0,s,N}^{(0,j)} \\ \mathbf{Y}_{0,s,N}^{(j,j)} \\ \hdashline \mathbf{Y}_{s,s,N}^{(0,j)} \\ \mathbf{Y}_{s,s,N}^{(j,j)} \end{array}\right] \end{array} = \begin{array}{c} \\ \langle sm\rangle \\ \langle sm\rangle \\ \langle s\ell_0\rangle \\ \langle s\ell_j\rangle \\ \langle s\ell_0\rangle \\ \langle s\ell_j\rangle \end{array} \begin{array}{c} \begin{array}{cccccc} \langle sm\rangle & \langle sm\rangle & \langle s\ell_0\rangle & \langle s\ell_j\rangle & \langle s\ell_0\rangle & \langle s\ell_j\rangle \end{array} \\ \left[\begin{array}{c:c:ccc:cc} \mathbf{R}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{R}_{31}^{(0,j)} & \mathbf{R}_{32}^{(0,j)} & \mathbf{R}_{33}^{(0,j)} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{R}_{41}^{(j,j)} & \mathbf{R}_{42}^{(j,j)} & \mathbf{R}_{43}^{(j,j)} & \mathbf{R}_{44}^{(j,j)} & \mathbf{0} & \mathbf{0} \\ \hdashline \mathbf{R}_{51}^{(0,j)} & \mathbf{R}_{52}^{(0,j)} & \mathbf{R}_{53}^{(0,j)} & \mathbf{R}_{54}^{(0,j)} & \mathbf{R}_{55}^{(0,j)} & \mathbf{0} \\ \mathbf{R}_{61}^{(j,j)} & \mathbf{R}_{62}^{(j,j)} & \mathbf{R}_{63}^{(j,j)} & \mathbf{R}_{64}^{(j,j)} & \mathbf{R}_{65}^{(j,j)} & \mathbf{R}_{66}^{(j,j)} \end{array}\right] \end{array} \cdot \begin{array}{c} \langle N\rangle \\ \left[\begin{array}{c} \mathbf{Q}_1 \\ \hdashline \mathbf{Q}_2 \\ \mathbf{Q}_3^{(j)} \\ \mathbf{Q}_4^{(j)} \\ \hdashline \mathbf{Q}_5^{(j)} \\ \mathbf{Q}_6^{(j)} \end{array}\right] \end{array}. \quad (2\text{-}18)$$

A MATLAB function to calculate the compressed data matrices $\mathbf{R}$ for all sensor roving experiments is shown in Listing B.21 on page 96. In this implementation, the computationally cheaper Cholesky decomposition of the Gramian of the left hand side of Eq. (2-18) is used when possible [6, 17]. Furthermore, this Gramian is calculated efficiently — both in terms of memory and computation time — by exploiting the Hankel matrix structure of the left hand side of Eq. (2-18) without needing any Hankel matrix in memory, as explained in Lemma 11 on page 58.

It can be shown that [16, Lemma 9.4]

$$\begin{bmatrix} \mathbf{Y}_{s,s,N}^{(0,j)} \\ \mathbf{Y}_{s,s,N}^{(j,j)} \end{bmatrix} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top} = \begin{bmatrix} \mathbf{R}_{52:54}^{(0,j)} \\ \mathbf{R}_{62:64}^{(j,j)} \end{bmatrix} \cdot \mathbf{R}_{22:44}^{(j)\top}, \tag{2-19}$$

where $\mathbf{R}_{gh:pq}$ denotes the block matrix determined by the upper left submatrix $\mathbf{R}_{gh}$ and the lower right submatrix $\mathbf{R}_{pq}$. Under reasonable assumptions $\mathbf{R}_{22:44}^{(j)}$ is invertible [16, Lemma 9.9] and hence,

$$\lim_{N\to\infty} \frac{1}{N} \begin{bmatrix} \mathbf{R}_{52:54}^{(0,j)} \\ \mathbf{R}_{62:64}^{(j,j)} \end{bmatrix} = \lim_{N\to\infty} \frac{1}{N} \begin{bmatrix} \mathbf{O}_s^{(0)} \\ \mathbf{O}_s^{(j)} \end{bmatrix} \cdot \underbrace{\mathbf{X}_{s,N}^{(j)} \cdot \mathbf{\Pi}_{\mathbf{U}_{s,s,N}}^{\perp} \cdot \mathbf{Z}_N^{(j)\top} \cdot \mathbf{R}_{22:44}^{(j)\,-\top}}_{=:\mathbf{\Xi}_s^{(j)} \in \mathbb{R}^{n\times s(m+\ell_0+\ell_j)}}. \tag{2-20}$$

If we choose N very large — i.e. acquire a lot of samples — we can find an estimate of the extended observability matrices in some basis $\mathbf{T}^{(j)}$ with a truncated order singular value decomposition (SVD):

$$\begin{bmatrix} \mathbf{O}_s^{(0)} \\ \mathbf{O}_s^{(j)} \end{bmatrix} \cdot \mathbf{\Xi}_s^{(j)} \approx \begin{bmatrix} \mathbf{R}_{52:54}^{(0,j)} \\ \mathbf{R}_{62:64}^{(j,j)} \end{bmatrix} \approx \begin{bmatrix} \mathcal{U}_n^{(0,j)} \\ \mathcal{U}_n^{(j,j)} \end{bmatrix} \mathbf{\Sigma}_n^{(j)} \mathcal{V}_n^{(j)\top}; \quad \begin{cases} \hat{\mathbf{O}}_s^{(0)} \mathbf{T}^{(j)} = \mathcal{U}_n^{(0,j)} \\ \hat{\mathbf{O}}_s^{(j)} \mathbf{T}^{(j)} = \mathcal{U}_n^{(j,j)} \\ \mathbf{T}^{(j)-1} \hat{\mathbf{\Xi}}_s^{(j)} = \mathbf{\Sigma}_n^{(j)} \mathcal{V}_n^{(j)\top} \end{cases} \tag{2-21}$$

A MATLAB function to calculate this SVD for the compressed data matrix of a series of sensor roving experiments is shown in Listing B.22 on page 97.

If only the reference sensor data is used in the PO-MOESP algorithm, which will be the case in Algorithm IIa/b on pages 20 and 21, we consider only the top block row in Eq. (2-20), such that the factorisation in Eq. (2-21) can be used with an unpartitioned left singular vector matrix $\mathcal{U}_n$. Hence, we can use $\mathbf{R}_{52:54}^{(0,j)}$ as if it were the compressed data matrix of the ordinary PO-MOESP implementation, even though roving sensor data was used in its calculation.

2-2-5 Sensor roving

The main difficulty in subspace model identification (SMI) from a series of sensor roving experiments is that in general both the extended observability matrix $\mathbf{O}_s^{(j)}$ and the

generalised state sequence $\hat{\boldsymbol{\Xi}}_s^{(j)}$ differ between experiments. With abuse of notation we can denote the compressed data matrices and the corresponding factorisations as follows:

$$\begin{Bmatrix} \mathbf{R}_{52:54}^{(0,1)}, & \mathbf{R}_{52:54}^{(0,2)}, & \cdots & \mathbf{R}_{52:54}^{(0,J)}, \\ \mathbf{R}_{62:64}^{(1,1)}, & & & \\ & \mathbf{R}_{62:64}^{(2,2)}, & & \\ & & \ddots & \\ & & & \mathbf{R}_{62:64}^{(J,J)} \end{Bmatrix} = \begin{Bmatrix} \mathbf{O}_s^{(0)}\boldsymbol{\Xi}_s^{(1)}, & \mathbf{O}_s^{(0)}\boldsymbol{\Xi}_s^{(2)}, & \cdots & \mathbf{O}_s^{(0)}\boldsymbol{\Xi}_s^{(1)}, \\ \mathbf{O}_s^{(1)}\boldsymbol{\Xi}_s^{(1)}, & & & \\ & \mathbf{O}_s^{(2)}\boldsymbol{\Xi}_s^{(2)}, & & \\ & & \ddots & \\ & & & \mathbf{O}_s^{(J)}\boldsymbol{\Xi}_s^{(J)} \end{Bmatrix}, \quad (2\text{-}22)$$

where each column represents an experiment and each row represents a sensor pool position. Because of the missing data it is not possible to obtain all the extended observability matrices from a single SVD. Using multiple SVDs yields equally many different bases $\mathbf{T}$. However, by virtue of the reference sensor measurements, these bases can be related to one another, provided that the plant is observable from the reference sensor pool. For more elaborate discussions see [2, 3, 9, 10, 13].

There are two main approaches to combine the data from sensor roving measurements: Either start with a local SVD for each column in Eq. (2-22), or start with a global SVD for the top row in Eq. (2-22). We will discuss two variants of both approaches. In all cases the result is a global extended observability matrix $\left(\boldsymbol{\mathcal{O}}_{\mathbf{s}}^{\forall}\mathbf{T}\right)$ that is a stack of local observability matrices in the same base, which differs from an ordinary extended observability matrix only by a row permutation:

$$\boldsymbol{\mathcal{O}}_{\mathbf{s}}^{\forall}\mathbf{T} \coloneqq \begin{bmatrix} \boldsymbol{\mathcal{O}}_s(\mathbf{A}, \mathbf{C}_0) \cdot \mathbf{T} \\ \boldsymbol{\mathcal{O}}_s(\mathbf{A}, \mathbf{C}_1) \cdot \mathbf{T} \\ \vdots \\ \boldsymbol{\mathcal{O}}_s(\mathbf{A}, \mathbf{C}_J) \cdot \mathbf{T} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mathcal{O}}_s^{(0)}\mathbf{T} \\ \boldsymbol{\mathcal{O}}_s^{(1)}\mathbf{T} \\ \vdots \\ \boldsymbol{\mathcal{O}}_s^{(J)}\mathbf{T} \end{bmatrix}. \quad (2\text{-}23)$$

Algorithm Ia: Scaling $\mathcal{O}^{(j)}$ with $\mathcal{O}^{(0)}$ after local SVDs

This method was presented by Döhler and Mevel [3, Algorithm 2] for covariance driven SMI, but as the authors commented, it can also be applied to data driven SMI. The main idea is to calculate an SVD for each experiment separately, as in Eq. (2-21), and then use the extended observability matrices of the reference sensors to bring everything to the same base, a procedure known as scaling.

The estimates that are known after the local SVDs are

$$\left\{ \left(\hat{\mathbf{O}}_s^{(0)}\mathbf{T}^{(j)}\right), \quad \left(\hat{\mathbf{O}}_s^{(j)}\mathbf{T}^{(j)}\right), \quad \left(\mathbf{T}^{(j)-1}\hat{\boldsymbol{\Xi}}_s\right) \right\} \qquad \forall j \in \{1, 2, \ldots, J\}. \quad (2\text{-}24)$$

Without loss of generality, assume we want to use the base $\mathbf{T}^{(1)}$ as the common base. If the system is observable from the reference sensors, we can scale the local extended

observability matrices from the other experiments as follows:

$$\begin{aligned}\hat{\mathbf{O}}_s^{(j)}\hat{\mathbf{T}}^{(1)} &= \hat{\mathbf{O}}_s^{(j)} \cdot \overbrace{\mathbf{T}^{(j)} \cdot \mathbf{T}^{(j)-1}}^{\mathbf{I}_n} \cdot \overbrace{\hat{\mathbf{O}}_s^{(0)\dagger} \cdot \hat{\mathbf{O}}_s^{(0)}}^{\mathbf{I}_n} \cdot \mathbf{T}^{(1)} \\ &= \hat{\mathbf{O}}_s^{(j)}\mathbf{T}^{(j)} \cdot \left(\hat{\mathbf{O}}_s^{(0)}\mathbf{T}^{(j)}\right)^{\dagger} \cdot \hat{\mathbf{O}}_s^{(0)}\mathbf{T}^{(1)} . \end{aligned} \tag{2-25}$$

This is allowed by Lemma 2 on page 55 because $\hat{\mathbf{O}}_s^{(0)}$ is full column rank by the virtue of the observability condition and $\mathbf{T}^{(j)}$ is invertible. Now the estimation of the global observability matrix reads as:

$$\hat{\mathfrak{O}}_{\mathbf{s}}^{\forall}\hat{\mathbf{T}}^{(1)} = \begin{bmatrix} \hat{\mathfrak{O}}_s^{(0)}\mathbf{T}^{(1)} \\ \hat{\mathfrak{O}}_s^{(1)}\mathbf{T}^{(1)} \\ \hat{\mathfrak{O}}_s^{(2)}\mathbf{T}^{(2)} \left(\hat{\mathfrak{O}}_s^{(0)}\mathbf{T}^{(2)}\right)^{\dagger} \hat{\mathfrak{O}}_s^{(0)}\mathbf{T}^{(1)} \\ \vdots \\ \hat{\mathfrak{O}}_s^{(J)}\mathbf{T}^{(J)} \left(\hat{\mathfrak{O}}_s^{(0)}\mathbf{T}^{(J)}\right)^{\dagger} \hat{\mathfrak{O}}_s^{(0)}\mathbf{T}^{(1)} \end{bmatrix} . \tag{2-26}$$

An implementation of Eq. (2-26) in MATLAB is shown in Listing B.23 on page 99.

The change of base according to Eq. (2-25) seems mathematically sound, so why is there a hat above the left hand side $\hat{\mathbf{T}}^{(1)}$? — The notation is misleading here. The two occurrences of $\hat{\mathbf{O}}_s^{(0)}$ on the right hand side are two different estimates: one from experiment j and one from experiment 1. In fact we should write Eq. (2-25) like

$$\begin{aligned}\hat{\mathbf{O}}_s^{(j|j)}\hat{\mathbf{T}}^{(1|j)} &= \hat{\mathbf{O}}_s^{(j|j)}\mathbf{T}^{(j)} \cdot \left(\hat{\mathbf{O}}_s^{(0|j)}\mathbf{T}^{(j)}\right)^{\dagger} \cdot \hat{\mathbf{O}}_s^{(0|1)}\mathbf{T}^{(1)} \\ &= \hat{\mathbf{O}}_s^{(j|j)} \cdot \hat{\mathbf{O}}_s^{(0|j)\dagger} \cdot \hat{\mathbf{O}}_s^{(0|1)} \cdot \mathbf{T}^{(1)} \qquad \text{(by Lemma 2)} \end{aligned} \tag{2-27}$$

$$\Rightarrow \quad \hat{\mathbf{T}}^{(1|j)} = \hat{\mathbf{O}}_s^{(0|j)\dagger} \cdot \hat{\mathbf{O}}_s^{(0|1)} \cdot \mathbf{T}^{(1)} . \tag{2-28}$$

Hence, the relative mismatch between the estimated extended observability matrices of the reference sensors, will result in an error in the base of the local extended observability matrices corresponding to sensor positions $2, 3, \ldots, J$.

The original algorithm prescribes that in the global extended observability matrix $\hat{\mathbf{O}}_s^{(0|1)}\mathbf{T}^{(1)}$ should be used for the reference sensor outputs as in Eq. (2-26) — which makes sense because it is already in the right base — but at the same time the estimates for the reference output from the other experiments, according to Eq. (2-27),

$$\hat{\mathbf{O}}_s^{(0|j)}\hat{\mathbf{T}}^{(1|j)} = \underbrace{\hat{\mathbf{O}}_s^{(0|j)} \cdot \hat{\mathbf{O}}_s^{(0|j)\dagger}}_{\neq \mathbf{I}_{s\ell_0}} \cdot \hat{\mathbf{O}}_s^{(0|1)}\mathbf{T}^{(1)} , \tag{2-29}$$

are discarded.

Algorithm Ib: Scaling $\mathbf{R}^{(j)}$ with $\mathbf{R}^{(0)}$ after local SVDs

The other algorithm proposed in Döhler and Mevel [3, Algorithm 1] is, at least for SMI methods without a left weighting matrix, a mathematically equivalent detour to the same result, up to a similarity transform. This algorithm also starts with local SVDs and the known estimates are again given by Eq. (2-24). Instead of scaling with the extended observability matrix, scaling is done with the data matrix that is first truncated at order n.

Let the hat denote this order truncation of the data matrix, then the scaling for this variant in terms of the known estimates reads in our notation as:

$$\begin{aligned}\hat{\mathbf{R}}^{(j,1)} &= \hat{\mathbf{R}}^{(j,j)}_{62:64} \cdot \hat{\mathbf{R}}^{(0,j)\dagger}_{52:54} \cdot \hat{\mathbf{R}}^{(0,1)}_{52:54} \\ &= \hat{\mathfrak{O}}^{(j|j)}_s \mathbf{T}^{(j)} \cdot \mathbf{T}^{(j)-1} \hat{\mathbf{\Xi}}^{(j)}_s \cdot \left(\hat{\mathfrak{O}}^{(0|j)}_s \mathbf{T}^{(j)} \cdot \mathbf{T}^{(j)-1} \hat{\mathbf{\Xi}}^{(j)}_s \right)^{\dagger} \cdot \hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \end{aligned} \tag{2-30}$$

$$\hat{\mathbf{R}}^{(0,1)} = \hat{\mathbf{R}}^{(0,1)}_{52:54} = \hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \tag{2-31}$$

Subsequently the scaled $\hat{\mathbf{R}}^{(j,1)}$ matrices are combined to a global data matrix, which is factored to obtain a global extended observability matrix:

$$\begin{aligned}\begin{bmatrix} \hat{\mathbf{R}}^{(0,1)} \\ \hat{\mathbf{R}}^{(1,1)} \\ \vdots \\ \hat{\mathbf{R}}^{(J,1)} \end{bmatrix} &= \begin{bmatrix} \hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \\ \hat{\mathfrak{O}}^{(1|1)}_s \mathbf{T}^{(1)} \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \left(\hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \right)^{\dagger} \hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \\ \vdots \\ \hat{\mathfrak{O}}^{(J|J)}_s \mathbf{T}^{(J)} \mathbf{T}^{(J)-1} \hat{\mathbf{\Xi}}^{(J)}_s \left(\hat{\mathfrak{O}}^{(0|J)}_s \mathbf{T}^{(J)} \mathbf{T}^{(J)-1} \hat{\mathbf{\Xi}}^{(J)}_s \right)^{\dagger} \hat{\mathfrak{O}}^{(0|1)}_s \mathbf{T}^{(1)} \mathbf{T}^{(1)-1} \hat{\mathbf{\Xi}}^{(1)}_s \end{bmatrix} \\ &= \bar{\mathbf{U}}_n \bar{\mathbf{\Sigma}}_n \bar{\mathbf{V}}_n^{\top}; \qquad \begin{cases} \hat{\bar{\mathfrak{O}}}^{\forall}_s \bar{\mathbf{T}} \coloneqq \bar{\mathbf{U}}_n, \\ \bar{\mathbf{T}}^{-1} \hat{\bar{\mathbf{\Xi}}}_s \coloneqq \bar{\mathbf{\Sigma}}_n \bar{\mathbf{V}}_n^{\top}. \end{cases} \end{aligned} \tag{2-32}$$

The resulting global extended observability matrix $\hat{\bar{\mathfrak{O}}}^{\forall}_s$ is equal to the global extended observability matrix $\hat{\mathfrak{O}}^{\forall}_s$ obtained with Algorithm Ia according to Eq. (2-26). That is, $\hat{\bar{\mathfrak{O}}}^{\forall}_s \bar{\mathbf{T}}$ is equivalent to $\hat{\mathfrak{O}}^{\forall}_{\mathbf{s}} \hat{\mathbf{T}}^{(1)}$ up to a similarity transform.

PROOF We first simplify Eq. (2-30):

$$\begin{aligned}\hat{\mathbf{R}}^{(j,1)} &= \hat{\mathfrak{O}}_s^{(j|j)}\mathbf{T}^{(j)} \cdot \mathbf{T}^{(j)-1}\hat{\mathbf{\Xi}}_s^{(j)} \cdot \left(\hat{\mathfrak{O}}_s^{(0|j)}\mathbf{T}^{(j)} \cdot \mathbf{T}^{(j)-1}\hat{\mathbf{\Xi}}_s^{(j)}\right)^\dagger \cdot \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)} \\ &= \hat{\mathfrak{O}}_s^{(j|j)}\mathbf{T}^{(j)} \left(\mathbf{T}^{(j)-1}\hat{\mathbf{\Xi}}_s^{(j)}\right)\left(\mathbf{T}^{(j)-1}\hat{\mathbf{\Xi}}_s^{(j)}\right)^\dagger \cdot \left(\hat{\mathfrak{O}}_s^{(0|j)}\mathbf{T}^{(j)}\right)^\dagger \cdot \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)} \\ &\qquad\qquad \text{(by Lemma 2)} \\ &= \hat{\mathfrak{O}}_s^{(j|j)}\mathbf{T}^{(j)} \cdot \left(\hat{\mathfrak{O}}_s^{(0|j)}\mathbf{T}^{(j)}\right)^\dagger \cdot \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)} \qquad (2\text{-}33) \\ \hat{\mathbf{R}}^{(1,1)} &= \hat{\mathfrak{O}}_s^{(1|1)}\mathbf{T}^{(1)} \cdot \left(\hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)}\right)^\dagger \cdot \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)} \\ &= \hat{\mathfrak{O}}_s^{(1|1)}\mathbf{T}^{(1)} \cdot \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)}. \qquad (2\text{-}34)\end{aligned}$$

Now Eq. (2-32) reads as

$$\begin{bmatrix}\hat{\mathbf{R}}^{(0,1)} \\ \hat{\mathbf{R}}^{(1,1)} \\ \hat{\mathbf{R}}^{(2,1)} \\ \vdots \\ \hat{\mathbf{R}}^{(J,1)}\end{bmatrix} = \begin{bmatrix}\hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \\ \hat{\mathfrak{O}}_s^{(1|1)}\mathbf{T}^{(1)} \\ \hat{\mathfrak{O}}_s^{(2|2)}\mathbf{T}^{(2)}\left(\hat{\mathfrak{O}}_s^{(0|2)}\mathbf{T}^{(2)}\right)^\dagger \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)} \\ \vdots \\ \hat{\mathfrak{O}}_s^{(J|J)}\mathbf{T}^{(J)}\left(\hat{\mathfrak{O}}_s^{(0|J)}\mathbf{T}^{(J)}\right)^\dagger \hat{\mathfrak{O}}_s^{(0|1)}\mathbf{T}^{(1)}\end{bmatrix} \mathbf{T}^{(1)-1}\hat{\mathbf{\Xi}}_s^{(1)} = \hat{\mathfrak{O}}_s^{\forall} \cdot \hat{\mathbf{\Xi}}_s^{(1)} = \bar{\mathbf{U}}_n\bar{\mathbf{\Sigma}}_n\bar{\mathbf{V}}_n^\top, \qquad (2\text{-}35)$$

where $\hat{\mathfrak{O}}_s^{\forall}$ is the global observability matrix obtained from Algorithm Ia, cf. Eq. (2-26). Multiplying the last two expressions in Eq. (2-35) from left and right yields

$$\bar{\mathbf{U}}_n^\top\hat{\mathfrak{O}}_s^{\forall} \cdot \hat{\mathbf{\Xi}}_s^{(1)}\bar{\mathbf{V}}_n = \bar{\mathbf{U}}_n^\top\bar{\mathbf{U}}_n\bar{\mathbf{\Sigma}}_n\bar{\mathbf{V}}_n^\top\bar{\mathbf{V}}_n = \bar{\mathbf{\Sigma}}_n, \qquad (2\text{-}36)$$

$$\Rightarrow \begin{cases}\operatorname{rank}\left(\bar{\mathbf{U}}_n^\top\hat{\mathfrak{O}}_s^{\forall}\right) = n & \Rightarrow \left(\bar{\mathbf{U}}_n^\top\hat{\mathfrak{O}}_s^{\forall}\right) \in \mathbb{R}^{n\times n} \text{ is invertible,} \\ \operatorname{rank}\left(\hat{\mathbf{\Xi}}_s^{(1)}\bar{\mathbf{V}}_n\right) = n & \Rightarrow \left(\hat{\mathbf{\Xi}}_s^{(1)}\bar{\mathbf{V}}_n\right) \in \mathbb{R}^{n\times n} \text{ is invertible.}\end{cases} \qquad (2\text{-}37)$$

Equation (2-37) follows from applying Lemma 12 to Eq. (2-36).[2] Now we can infer from Eqs. (2-32) and (2-37) that

$$\hat{\bar{\mathfrak{O}}}_s^{\forall}\bar{\mathbf{T}} = \bar{\mathbf{U}}_n \underbrace{\bar{\mathbf{\Sigma}}_n\bar{\mathbf{V}}_n^\top \cdot \bar{\mathbf{V}}_n\bar{\mathbf{\Sigma}}_n^{-1}}_{\mathbf{I}_n} = \hat{\mathfrak{O}}_s^{\forall} \cdot \underbrace{\hat{\mathbf{\Xi}}_s^{(1)}\bar{\mathbf{V}}_n \cdot \bar{\mathbf{\Sigma}}_n^{-1}}_{\text{invertible}}, \text{ and} \qquad (2\text{-}38)$$

$$\bar{\mathbf{T}}^{-1}\hat{\bar{\mathbf{\Xi}}}_s = \underbrace{\bar{\mathbf{U}}_n^\top \cdot \bar{\mathbf{U}}_n}_{\mathbf{I}_n}\bar{\mathbf{\Sigma}}_n\bar{\mathbf{V}}_n^\top = \underbrace{\bar{\mathbf{U}}_n^\top\hat{\mathfrak{O}}_s^{\forall}}_{\text{invertible}} \cdot \hat{\mathbf{\Xi}}_s^{(1)}. \qquad (2\text{-}39)$$

[2] If we would apply Lemma 12 directly to the products in Eq. (2-37), the result would be inconclusive since the common dimensionality of those multiplications are $s(\ell_0 + J\ell_j) > n$ and $s(m + \ell_0 + \ell_j) > n$ respectively.

Equations (2-38) and (2-39) state that

$$\hat{\bar{\mathbf{O}}}_s^{\forall} = \hat{\mathbf{O}}_s^{\forall} \quad \text{and} \quad \hat{\bar{\mathbf{\Xi}}}_s = \hat{\mathbf{\Xi}}_s^{(1)} \quad \text{with} \quad \bar{\mathbf{T}} = \hat{\mathbf{\Xi}}_s^{(1)} \bar{\mathcal{V}}_n \bar{\mathbf{\Sigma}}_n^{-1} \quad \text{and} \quad \bar{\mathbf{T}}^{-1} = \bar{\mathcal{U}}_n^{\top} \hat{\mathbf{O}}_s^{\forall} .$$

As a sanity check, we verify the consistency of the latter two expressions:

$$\bar{\mathbf{T}}^{-1}\bar{\mathbf{T}} = \bar{\mathcal{U}}_n^{\top} \left(\hat{\mathbf{O}}_s^{\forall} \cdot \hat{\mathbf{\Xi}}_s^{(1)} \right) \bar{\mathcal{V}}_n \bar{\mathbf{\Sigma}}_n^{-1} = \bar{\mathcal{U}}_n^{\top} \left(\bar{\mathcal{U}}_n \bar{\mathbf{\Sigma}}_n \bar{\mathcal{V}}_n^{\top} \right) \bar{\mathcal{V}}_n \bar{\mathbf{\Sigma}}_n^{-1} = \mathbf{I}_n , \tag{2-40}$$

$$\begin{aligned} \bar{\mathbf{T}}\bar{\mathbf{T}}^{-1} &= \hat{\mathbf{\Xi}}_s^{(1)} \left(\bar{\mathcal{V}}_n \bar{\mathbf{\Sigma}}_n^{-1} \cdot \bar{\mathcal{U}}_n^{\top} \right) \hat{\mathbf{O}}_s^{\forall} = \hat{\mathbf{\Xi}}_s^{(1)} \left(\hat{\mathbf{O}}_s^{\forall} \cdot \hat{\mathbf{\Xi}}_s^{(1)} \right)^{\dagger} \hat{\mathbf{O}}_s^{\forall} \\ &= \hat{\mathbf{\Xi}}_s^{(1)} \hat{\mathbf{\Xi}}_s^{(1)\dagger} \cdot \hat{\mathbf{O}}_s^{\forall\dagger} \hat{\mathbf{O}}_s^{\forall} = \mathbf{I}_n \qquad \text{(by Lemma 2).} \end{aligned} \tag{2-41}$$

■

Comparing Algorithm Ia and Ib computationally, we see that the latter requires more matrix multiplications, an extra global SVD, an extra pseudo-inverse and the matrices that are pseudo-inverted are bigger: $\hat{\mathbf{R}}_{52:54}^{(0,j)} \in \mathbb{R}^{s\ell_0 \times s(m+\ell_0+\ell_j)}$, whereas $(\hat{\mathbf{O}}_s^{(0)}\mathbf{T}^{(j)}) \in \mathbb{R}^{s\ell_0 \times n}$. Hence, Algorithm Ia is preferable to Algorithm Ib.

Algorithm IIa: Scaling $\mathbf{R}^{(j)}$ with $\mathbf{\Xi}^{(j)}$ after a global SVD

This method was presented in Mevel, Basseville, Benveniste, *et al.* [9] and Mevel, Benveniste, Basseville, *et al.* [10] for covariance driven SMI. In Döhler, Andersen, and Mevel [2] it is applied to the Unweighted Principal Component algorithm (data-driven SMI), for an output-only model structure. The main idea is to start with an SVD for all experiments, but only for the reference outputs:

$$\mathbf{O}_s^{(0)} \cdot \begin{bmatrix} \mathbf{\Xi}_s^{(1)} & \mathbf{\Xi}_s^{(2)} & \cdots & \mathbf{\Xi}_s^{(j)} \end{bmatrix} \approx \begin{bmatrix} \mathbf{R}_{52:54}^{(0,1)} & \mathbf{R}_{52:54}^{(0,2)} & \cdots & \mathbf{R}_{52:54}^{(0,J)} \end{bmatrix} \approx \mathcal{U}_n^{(0)} \mathbf{\Sigma}_n \begin{bmatrix} \mathcal{V}_n^{(1)} \\ \mathcal{V}_n^{(2)} \\ \vdots \\ \mathcal{V}_n^{(J)} \end{bmatrix}^{\top}; \tag{2-42}$$

$$\begin{cases} \hat{\mathbf{O}}_s^{(0)}\mathbf{T} = \mathcal{U}_n^{(0)} , \\ \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)} = \mathbf{\Sigma}_n \mathcal{V}_n^{(j)\top} . \end{cases} \tag{2-43}$$

The obtained estimates and remaining data matrices are

$$\left\{ \left(\hat{\mathbf{O}}_s^{(0)}\mathbf{T} \right), \quad \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)} \right), \quad \mathbf{R}_{62:64}^{(j,j)} \right\} \qquad \forall j \in \{1, 2, \ldots, J\} . \tag{2-44}$$

Subsequently for each experiment the matrix $(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)})$ is used to scale the data matrix that belongs to the roving sensor outputs:

$$\begin{aligned} \hat{\mathbf{R}}^{(j,1)} &= \hat{\mathbf{O}}_s^{(j)} \hat{\mathbf{\Xi}}_s^{(1)} = \hat{\mathbf{O}}_s^{(j)} \underbrace{\hat{\mathbf{\Xi}}_s^{(j)} \cdot \hat{\mathbf{\Xi}}_s^{(j)\dagger}}_{\mathbf{I}_n} \underbrace{\mathbf{T} \cdot \mathbf{T}^{-1}}_{\mathbf{I}_n} \hat{\mathbf{\Xi}}_s^{(1)} \\ &= \mathbf{R}_{62:64}^{(j,j)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)} \right)^{\dagger} \cdot \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)} \qquad \text{(by Lemma 2),} \end{aligned} \tag{2-45}$$

$$\hat{\mathbf{R}}^{(0,1)} = \hat{\mathbf{O}}_s^{(0)}\mathbf{T} \cdot \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)} . \tag{2-46}$$

Finally the scaled data matrices are stacked to a global data matrix, which is decomposed to obtain a global extended observability matrix:

$$\begin{bmatrix} \hat{\mathbf{R}}^{(0,1)} \\ \hat{\mathbf{R}}^{(1,1)} \\ \vdots \\ \hat{\mathbf{R}}^{(J,1)} \end{bmatrix} = \begin{bmatrix} \hat{\mathfrak{O}}_s^{(0)}\mathbf{T} \cdot \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)} \\ \mathbf{R}_{62:64}^{(1,1)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)}\right)^{\dagger} \cdot \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)} \\ \vdots \\ \mathbf{R}_{62:64}^{(J,J)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(J)}\right)^{\dagger} \cdot \mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)} \end{bmatrix} = \bar{\mathcal{U}}_n \bar{\mathbf{\Sigma}}_n \bar{\mathcal{V}}_n^{\top}\,; \tag{2-47}$$

$$\begin{cases} \hat{\bar{\mathfrak{O}}}_s^{\forall}\bar{\mathbf{T}} \coloneqq \bar{\mathcal{U}}_n\,, \\ \bar{\mathbf{T}}^{-1}\hat{\bar{\mathbf{\Xi}}}_s \coloneqq \bar{\mathbf{\Sigma}}_n \bar{\mathcal{V}}_n^{\top}\,. \end{cases} \tag{2-48}$$

An implementation of Algorithm IIa in MATLAB is shown in Listing B.24 on page 100.

Algorithm IIb: Scaling $\mathfrak{O}^{(j)}$ with $\Xi^{(j)}$ after a global SVD

This algorithm is a mathematically equivalent shortcut to the same result as Algorithm IIa, up to a similarity transform.

Looking at Eq. (2-45), we can see that we multiply on the right by $\left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)}\right)$ and as a result every block row in the global extended observability matrix has a right factor $\hat{\mathbf{\Xi}}_s^{(1)}$. Factorisation of the global data matrix removes this right factor again, up to a similarity transform, like in Algorithm Ib. If we do not multiply by this right factor, there is no need for a second SVD. We start again with a global SVD for the reference output data as in Eqs. (2-42) and (2-43), yielding again the estimations and remaining data matrices in Eq. (2-44), and process the roving sensor data as follows:

$$\hat{\mathfrak{O}}_s^{(j)}\mathbf{T} = \hat{\mathfrak{O}}_s^{(j)} \underbrace{\hat{\mathbf{\Xi}}_s^{(j)} \cdot \hat{\mathbf{\Xi}}_s^{(j)\dagger}}_{\mathbf{I}_n} \mathbf{T} = \mathbf{R}_{62:64}^{(j,j)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)}\right)^{\dagger} \qquad \text{(by Lemma 2).} \tag{2-49}$$

This result can be interpreted as

$$\hat{\mathfrak{O}}_s^{(j)}\mathbf{T} = \underset{\left(\mathfrak{O}_s^{(j)}\mathbf{T}\right)}{\arg\min} \left\| \left(\mathfrak{O}_s^{(j)}\mathbf{T}\right)\left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(j)}\right) - \mathbf{R}_{62:64}^{(j,j)} \right\|_{\mathrm{F}} \tag{2-50}$$

$$= \left(\underset{\mathfrak{O}_s^{(j)}}{\arg\min} \left\| \mathfrak{O}_s^{(j)}\hat{\mathbf{\Xi}}_s^{(j)} - \mathbf{R}_{62:64}^{(j,j)} \right\|_{\mathrm{F}} \right) \cdot \mathbf{T}\,. \tag{2-51}$$

Based on Eq. (2-49), we construct the global extended observability matrix like

$$\hat{\mathbf{\Theta}}_s^{\forall}\mathbf{T} = \begin{bmatrix} \hat{\mathbf{\Theta}}_s^{(0)}\mathbf{T} \\ \hat{\mathbf{\Theta}}_s^{(1)}\mathbf{T} \\ \vdots \\ \hat{\mathbf{\Theta}}_s^{(J)}\mathbf{T} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{\Theta}}_s^{(0)}\mathbf{T} \\ \mathbf{R}_{62:64}^{(1,1)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(1)}\right)^{\dagger} \\ \vdots \\ \mathbf{R}_{62:64}^{(J,J)} \cdot \left(\mathbf{T}^{-1}\hat{\mathbf{\Xi}}_s^{(J)}\right)^{\dagger} \end{bmatrix} . \tag{2-52}$$

With less computational effort — about six times as fast in a realistic test — and presumably less numerical errors, this yields the same result as Algorithm IIa up to a similarity transform, according to the same reasoning that proves the equivalence of the results of Algorithm Ia and Ib: Eqs. (2-36) to (2-41). An implementation of Algorithm IIb in MATLAB is shown in Listing B.25 on page 101.

2-2-6 Identification of $(\mathbf{A}, \mathbf{C})$

After obtaining a global extended observability matrix from one of the merging algorithms, we only need to pre-multiply by a suitable permutation matrix $\boldsymbol{\mathcal{P}}$ to obtain an ordinary extended observability matrix:

$$\hat{\mathbf{\Theta}}_s\left(\mathbf{A}, \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_J \end{bmatrix}\right)\mathbf{T} = \boldsymbol{\mathcal{P}} \cdot \hat{\mathbf{\Theta}}_{\mathbf{s}}^{\forall}\mathbf{T} = \boldsymbol{\mathcal{P}} \cdot \begin{bmatrix} \hat{\mathbf{\Theta}}_s(\mathbf{A}, \mathbf{C}_0) \\ \hat{\mathbf{\Theta}}_s(\mathbf{A}, \mathbf{C}_1) \\ \vdots \\ \hat{\mathbf{\Theta}}_s(\mathbf{A}, \mathbf{C}_J) \end{bmatrix}\mathbf{T} . \tag{2-53}$$

From there the pair $(\mathbf{A}, \mathbf{C})$ can be identified using the shift-invariance property of the extended observability matrix, see e.g. [16], or any other method [11, 12].

In case $\ell_0 = \ell_j = 1$, so with one reference sensor output and one roving sensor output per experiment, $\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{P}}^{\star}_{(J+1),s}$, where the latter is the mod-$(J+1)$ perfect shuffle permutation [5, §1.2.11]. Using MATLAB notation:

$$\boldsymbol{\mathcal{P}}^{\star}_{p,s} = \mathbf{I}_{ps}([(1:s:ps), (2:s:ps), \ldots, (s:s:ps)], :) . \tag{2-54}$$

More generally, it can be shown that when $\ell_0 = \ell_j$,

$$\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{P}}^{\star}_{(J+1),s} \otimes \mathbf{I}_{\ell_j} . \tag{2-55}$$

Finally for $\ell_0 \neq \ell_j$ the permutation matrix can be constructed with Eq. (2-56).

$$\boldsymbol{\mathcal{P}} = \left[\left(\mathbf{I}_s \otimes \begin{bmatrix} \mathbf{I}_{\ell_0} \\ \mathbf{0}_{J\ell_j \times \ell_0} \end{bmatrix}\right) \;\vdots\; \left(\mathbf{I}_s \otimes \begin{bmatrix} \mathbf{0}_{\ell_0 \times J\ell_j} \\ \mathbf{I}_{J\ell_j} \end{bmatrix}\right) \cdot \left(\boldsymbol{\mathcal{P}}^{\star}_{J,s} \otimes \mathbf{I}_{\ell_j}\right)\right] . \tag{2-56}$$

Implementations of the equations in this section in MATLAB can be found in Listings B.7 and B.12 on pages 72 and 76.

2-2-7 Identification of (B, D)

Looking again at our model structure in Eq. (2-5) on page 12, we see that all sensor roving experiments correspond to a common input matrix $\mathbf{B}$ and reference output matrix $\mathbf{D}_0$, while there is a separate roving output matrix $\mathbf{D}_j$ for each sensor position. Hence, the data from all experiments must be used together to identify the pair $(\mathbf{B}, \mathbf{D})$.

In Haverkamp [6, Algorithm 3.3] and Verhaegen and Verdult [16, §9.2.4.3] the pair $(\mathbf{B}, \mathbf{D})$ is estimated together with the initial state vector $\boldsymbol{x}_0$, from the uncompressed data. We will refer to this method as Algorithm 2. Using this method in the roving sensor case requires estimating the elements of $\mathbf{B}$, $\mathbf{D}_0$, $\mathbf{D}_1, \ldots, \mathbf{D}_J$ together with the initial state vectors of each experiment from the uncompressed data of all experiments. That boils down to solving a system of the form $\boldsymbol{\Phi\theta} = \boldsymbol{\psi}$ for $\boldsymbol{\theta}$, where $\boldsymbol{\theta} \in \mathbb{R}^{Jn+nm+m\ell_0+Jm\ell_j}$ and $\boldsymbol{\psi} \in \mathbb{R}^{J(\ell_0+\ell_j)(N+s-1)}$, which is demanding in terms of memory usage.

Another method — originally presented by Verhaegen and Dewilde [15], adapted to PO-MOESP by Verhaegen [14] — looks computationally more appealing in two ways: First, the initial state vectors are not estimated, so the amount of parameters to estimate is reduced. Second, it uses the compressed data as input. For a series of roving sensor experiments with equal input sequences, this method boils down to solving a system of the form $\boldsymbol{\Phi\Theta} = \boldsymbol{\Psi}$ for $\boldsymbol{\Theta}$, where $\boldsymbol{\Theta} \in \mathbb{R}^{(J\ell_j+\ell_0+n)\times m}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{((J\ell_j+\ell_0)s^2-(J+1)ns)\times m}$. We will refer to this method as Algorithm 1.

The results obtained using Algorithm 1, in terms of VAF, were not always satisfactory, see Section 2-2-9 on page 33. Therefore Algorithm 2 was implemented as well. Both implementations are included in the MATLAB function shown in Listing B.26 on page 102. Experiments with artificial data suggest that Algorithm 1 is much faster than Algorithm 2 and the results are similar to slightly better when the underlying model structure resembles Eq. (2-5) on page 12. However, Algorithm 1 seems to be less robust to mismatches in the model structure. For example when adding a few percent of total harmonic distortion (THD) to some output channels, formed by the square of that output itself, the results of Algorithm 2 were hardly affected, while those of Algorithm 1 sometimes dropped tens of per cents in VAF for the modified output channels. It is unclear under which circumstances exactly this does or does not happen.

Algorithm 1

We will extend the method from [14, 15] for roving sensor experiments with equal inputs. For self-containment and to introduce the required variables, the original scheme will be rewritten for the adapted RQ-decomposition, Eq. (2-18) on page 14, for a single experiment. From there the combined estimation scheme follows naturally.

Consider Eqs. (2-6) and (2-18) on pages 12 and 14. The pair $(\mathbf{B}, \mathbf{D})$ is contained in the matrices $\mathfrak{T}_s^{(j)}$, so the goal of the following derivation is to free the matrices $\mathfrak{T}_s^{(j)}$ from the data equation and subsequently free the pair $(\mathbf{B}, \mathbf{D})$ from these matrices. According to [14, Theorem 4] the noise term $\mathbf{E}$ can, in short, be neglected during the following

derivation, because it vanishes for $N \to \infty$. We substitute both the $\mathbf{Y}$ and the $\mathbf{U}$ matrices in Eq. (2-6) by their RQ-decomposition, both for the past and the future data and multiply from the right by $\mathbf{Q}_1^\top$ and $\mathbf{Q}_2^\top$ to obtain the following set of equations for each experiment:

$$\begin{cases} \mathbf{Y}^{(0,j)}_{0,s,N}\mathbf{Q}_1^\top = \mathbf{R}^{(0,j)}_{31} & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(0)}\mathbf{U}_{0,s,N}\mathbf{Q}_1^\top & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(0)}\mathbf{R}_{21}\,, \\ \mathbf{Y}^{(0,j)}_{0,s,N}\mathbf{Q}_2^\top = \mathbf{R}^{(0,j)}_{32} & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_2^\top + \mathfrak{T}_s^{(0)}\mathbf{U}_{0,s,N}\mathbf{Q}_2^\top & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_2^\top + \mathfrak{T}_s^{(0)}\mathbf{R}_{22}\,, \\ \mathbf{Y}^{(0,j)}_{s,s,N}\mathbf{Q}_1^\top = \mathbf{R}^{(0,j)}_{51} & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{s,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(0)}\mathbf{U}_{s,s,N}\mathbf{Q}_1^\top & = \mathfrak{O}_s^{(0)}\mathbf{X}^{(j)}_{s,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(0)}\mathbf{R}_{11}\,, \\ \mathbf{Y}^{(j,j)}_{0,s,N}\mathbf{Q}_1^\top = \mathbf{R}^{(j,j)}_{41} & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(j)}\mathbf{U}_{0,s,N}\mathbf{Q}_1^\top & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(j)}\mathbf{R}_{21}\,, \\ \mathbf{Y}^{(j,j)}_{0,s,N}\mathbf{Q}_2^\top = \mathbf{R}^{(j,j)}_{42} & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_2^\top + \mathfrak{T}_s^{(j)}\mathbf{U}_{0,s,N}\mathbf{Q}_2^\top & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{0,N}\mathbf{Q}_2^\top + \mathfrak{T}_s^{(j)}\mathbf{R}_{22}\,, \\ \mathbf{Y}^{(j,j)}_{s,s,N}\mathbf{Q}_1^\top = \mathbf{R}^{(j,j)}_{61} & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{s,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(j)}\mathbf{U}_{s,s,N}\mathbf{Q}_1^\top & = \mathfrak{O}_s^{(j)}\mathbf{X}^{(j)}_{s,N}\mathbf{Q}_1^\top + \mathfrak{T}_s^{(j)}\mathbf{R}_{11}\,. \end{cases} \tag{2-57}$$

To remove the terms containing the extended observability matrices, we multiply from the left by the transpose of their orthogonal complements $\mathfrak{O}_s^{(0)\perp} \in \mathbb{R}^{s\ell_0 \times (s\ell_0 - n)}$ and $\mathfrak{O}_s^{(j)\perp} \in \mathbb{R}^{s\ell_j \times (s\ell_j - n)}$. Furthermore we juxtapose the equations that belong to the same sensor pool:

$$\begin{cases} \mathfrak{O}_s^{(0)\perp\top}\begin{bmatrix}\mathbf{R}^{(0,j)}_{31} & \mathbf{R}^{(0,j)}_{32} & \mathbf{R}^{(0,j)}_{51}\end{bmatrix} = \mathfrak{O}_s^{(0)\perp\top}\mathfrak{T}_s^{(0)}\begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{11}\end{bmatrix}, \\ \mathfrak{O}_s^{(j)\perp\top}\begin{bmatrix}\mathbf{R}^{(j,j)}_{41} & \mathbf{R}^{(j,j)}_{42} & \mathbf{R}^{(j,j)}_{61}\end{bmatrix} = \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_s^{(j)}\begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{11}\end{bmatrix}. \end{cases} \tag{2-58}$$

The rightmost matrices are full row rank as a consequence of sufficiently persistent excitation [14]. Hence

$$\begin{cases} \bar{\mathfrak{T}}^{(0|j)} \coloneqq \mathfrak{O}_s^{(0)\perp\top}\begin{bmatrix}\mathbf{R}^{(0,j)}_{31} & \mathbf{R}^{(0,j)}_{32} & \mathbf{R}^{(0,j)}_{51}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{11}\end{bmatrix}^\dagger = \mathfrak{O}_s^{(0)\perp\top}\mathfrak{T}_s^{(0)}\,, \\ \bar{\mathfrak{T}}^{(j)} \coloneqq \mathfrak{O}_s^{(j)\perp\top}\begin{bmatrix}\mathbf{R}^{(j,j)}_{41} & \mathbf{R}^{(j,j)}_{42} & \mathbf{R}^{(j,j)}_{61}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{11}\end{bmatrix}^\dagger = \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_s^{(j)}\,. \end{cases} \tag{2-59}$$

We partition these equations in blocks of width m (only shown for one of the two):

$$\begin{bmatrix}\bar{\mathfrak{T}}^{(j)}_1 \;\vdots\; \bar{\mathfrak{T}}^{(j)}_2 \;\vdots\; \cdots \;\vdots\; \bar{\mathfrak{T}}^{(j)}_s\end{bmatrix} = \begin{bmatrix}\mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}^{(j)}_{s,1} \;\vdots\; \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}^{(j)}_{s,2} \;\vdots\; \cdots \;\vdots\; \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}^{(j)}_{s,s}\end{bmatrix}. \tag{2-60}$$

The block columns $\mathfrak{T}^{(j)}_{s,i}$ correspond to the block columns in the definition of $\mathfrak{T}^{(j)}_s$ in Eq. (2-10) on page 13. These can be constructed in terms of $(\mathbf{B}, \mathbf{D})$ according to the following recursive description.

$$\mathfrak{T}^{(j)}_{s,1} = \begin{bmatrix}\mathbf{D}_j \\ \mathbf{C}_j\mathbf{B} \\ \mathbf{C}_j\mathbf{A}\mathbf{B} \\ \vdots \\ \mathbf{C}_j\mathbf{A}^{s-2}\mathbf{B}\end{bmatrix} = \begin{bmatrix}\mathbf{0} & \mathbf{I}_{\ell_j} \\ \mathfrak{O}^{(j)}_{s-1} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{B} \\ \mathbf{D}_j\end{bmatrix}; \tag{2-61}$$

$$\mathfrak{T}^{(j)}_{s,i+1} = \mathbf{\Delta}_{\ell_j,s} \cdot \mathfrak{T}^{(j)}_{s,i}\,, \text{ where} \tag{2-62}$$

$$\mathbf{\Delta}_{\ell_j,s} \coloneqq \begin{bmatrix}\mathbf{0} & \mathbf{0}_{\ell_j \times \ell_j} \\ \mathbf{I}_{(s-1)\ell_j} & \mathbf{0}\end{bmatrix}. \tag{2-63}$$

Stacking the blocks of Eq. (2-60) allows us to write

$$\boldsymbol{\Upsilon}^{(j)} := \begin{bmatrix} \bar{\boldsymbol{\Upsilon}}_1^{(j)} \\ \bar{\boldsymbol{\Upsilon}}_2^{(j)} \\ \bar{\boldsymbol{\Upsilon}}_3^{(j)} \\ \vdots \\ \bar{\boldsymbol{\Upsilon}}_s^{(j)} \end{bmatrix} = \begin{bmatrix} \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_{s,1}^{(j)} \\ \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_{s,2}^{(j)} \\ \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_{s,3}^{(j)} \\ \vdots \\ \mathfrak{O}_s^{(j)\perp\top}\mathfrak{T}_{s,s}^{(j)} \end{bmatrix} = \begin{bmatrix} \mathfrak{O}_s^{(j)\perp\top} \\ \mathfrak{O}_s^{(j)\perp\top}\boldsymbol{\Delta}_{\ell_j,s} \\ \mathfrak{O}_s^{(j)\perp\top}\boldsymbol{\Delta}_{\ell_j,s}^2 \\ \vdots \\ \mathfrak{O}_s^{(j)\perp\top}\boldsymbol{\Delta}_{\ell_j,s}^{s-1} \end{bmatrix} \mathfrak{T}_{s,1}^{(j)}$$

$$= \mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_j,s}\,, \mathfrak{O}_s^{(j)\perp\top}\Big) \begin{bmatrix} \mathbf{0} & \mathbf{I}_{\ell_j} \\ \mathfrak{O}_{s-1}^{(j)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{B} \\ \mathbf{D}_j \end{bmatrix}, \tag{2-64}$$

$$\text{and likewise} \qquad \boldsymbol{\Upsilon}^{(0|j)} = \mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_0,s}\,, \mathfrak{O}_s^{(0)\perp\top}\Big) \begin{bmatrix} \mathbf{0} & \mathbf{I}_{\ell_0} \\ \mathfrak{O}_{s-1}^{(0)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{B} \\ \mathbf{D}_0 \end{bmatrix}. \tag{2-65}$$

The upper left block diagonal matrix $\mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_j,s}\,, \mathfrak{O}_s^{(j)\perp\top}\Big) \in \mathbb{R}^{s(s\ell_j-n)\times s\ell_j}$ has nothing to do with observability; this is both a convenient notation and a monkey-proof way to construct it.

If we combine these equations for all experiments, we obtain an estimate of $(\mathbf{B}, \mathbf{D})$ with the following formula, the derivation of which is shown in Lemma 14 on page 61.

$$\begin{bmatrix} \hat{\mathbf{B}} \\ \hat{\mathbf{D}} \end{bmatrix} = \left(\operatorname{diag}\left(\begin{matrix} \sqrt{J}\cdot\mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_0,s}\,, \mathfrak{O}_s^{(0)\perp\top}\Big)\,, \\ \mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_j,s}\,, \mathfrak{O}_s^{(1)\perp\top}\Big)\,, \\ \cdots, \\ \mathfrak{O}_s\Big(\boldsymbol{\Delta}_{\ell_j,s}\,, \mathfrak{O}_s^{(J)\perp\top}\Big) \end{matrix} \right) \cdot \mathfrak{M} \right)^{\dagger} \cdot \underbrace{\begin{bmatrix} \frac{1}{\sqrt{J}} \sum_{j=1}^{J} \boldsymbol{\Upsilon}^{(0|j)} \\ \boldsymbol{\Upsilon}^{(1)} \\ \vdots \\ \boldsymbol{\Upsilon}^{(J)} \end{bmatrix}}_{=:\boldsymbol{\Upsilon}^{\forall} \in \mathbb{R}^{\left(s^2(\ell_0+J\ell_j)-(J+1)sn\right)\times m}}, \text{where} \tag{2-66}$$

$$\mathfrak{M} := \begin{bmatrix} \begin{bmatrix} \mathbf{0}_{\ell_0\times(s-1)\ell_0} \\ \mathbf{I}_{(s-1)\ell_0} \end{bmatrix} & \mathbf{0}_{s\ell_0\times J(s-1)\ell_j} \\ \mathbf{0}_{J(s-1)\ell_j\times s\ell_0} & \mathbf{I}_J \otimes \begin{bmatrix} \mathbf{0}_{\ell_j\times(s-1)\ell_j} \\ \mathbf{I}_{(s-1)\ell_j} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} \mathfrak{O}_{s-1}^{(0)} \\ \mathfrak{O}_{s-1}^{(1)} \\ \vdots \\ \mathfrak{O}_{s-1}^{(J)} \end{bmatrix} \cdot \begin{bmatrix} \begin{bmatrix} \mathbf{I}_{\ell_0} \\ \mathbf{0}_{(s-1)\ell_0\times\ell_0} \end{bmatrix} & \mathbf{0}_{s\ell_0\times J\ell_j} \\ \mathbf{0}_{J\ell_j\times s\ell_0} & \mathbf{I}_J \otimes \begin{bmatrix} \mathbf{I}_{\ell_j} \\ \mathbf{0}_{(s-1)\ell_j\times\ell_j} \end{bmatrix} \end{bmatrix}, \tag{2-67}$$

and all ordinary observability matrices and their orthogonal complements are calculated from $(\hat{\mathbf{A}}, \hat{\mathbf{C}})$. A necessary condition for Eq. (2-66) to hold is

$$\operatorname{rank}\left(\begin{bmatrix} \mathbf{0}_{\ell_j\times(s-1)\ell_j} & \mathbf{I}_{\ell_j} \end{bmatrix} \cdot \mathfrak{O}_s^{(j)\perp} \right) = \ell_j \qquad \forall j \in \{0, 1, \ldots, J\}, \tag{2-68}$$

c.f. [15, Theorem 4 (5)]. In case Eq. (2-68) is not satisfied, this may be remedied by choosing the parameter s larger [15].

Since only the scaled sum of the matrices $\boldsymbol{\Upsilon}^{(0|j)}$ is required, instead of using Eq. (2-59) (upper) for each experiment separately, the top block row of $\boldsymbol{\Upsilon}^{\forall}$ can be efficiently calculated from

$$\frac{1}{\sqrt{J}}\sum_{j=1}^{J}\bar{\boldsymbol{\Upsilon}}^{(0|j)} = \frac{1}{\sqrt{J}}\cdot\mathfrak{O}_s^{(0)\perp\top}\cdot\sum_{j=1}^{J}\begin{bmatrix}\mathbf{R}_{31}^{(0,j)} & \mathbf{R}_{32}^{(0,j)} & \mathbf{R}_{51}^{(0,j)}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{11}\end{bmatrix}^{\dagger}, \quad (2\text{-}69)$$

which after block reshaping like Eq. (2-64) yields the required block row.

Algorithm 2

This algorithm is a linear regression on the uncompressed input and output data given the estimates of $(\mathbf{A}, \mathbf{C})$. It is based on the observation that the deterministic part of an output sample of a discrete-time state space model like Eq. (2-5) on page 12 can be calculated as [6, 16, 17]

$$\begin{aligned}\boldsymbol{y}(k) &= \mathbf{C}\mathbf{A}^k\boldsymbol{x}(0) + \sum_{\tau=0}^{k-1}\mathbf{C}\mathbf{A}^{k-1-\tau}\mathbf{B}\boldsymbol{u}(\tau) + \mathbf{D}\boldsymbol{u}(k)\\ &= \left[\mathbf{C}\mathbf{A}^k \;\vdots\; \sum_{\tau=0}^{k-1}\boldsymbol{u}^{\top}(\tau)\otimes\mathbf{C}\mathbf{A}^{k-1-\tau} \;\vdots\; \boldsymbol{u}^{\top}(k)\otimes\mathbf{I}_{\ell}\right]\cdot\begin{bmatrix}\boldsymbol{x}(0)\\ \mathrm{vec}(\mathbf{B})\\ \mathrm{vec}(\mathbf{D})\end{bmatrix} \quad \text{by Lemma 6, (2-70)}\end{aligned}$$

where $\boldsymbol{u}^{\top}(k)$ is a row vector. Using the previously calculated estimations for $\mathbf{A}$ and $\mathbf{C}$, all the unknowns are in the right hand side vector. Stacking this equation for all recorded samples $k \in \left\{1, 2, \ldots \bar{N}\right\}$ yields an overdetermined equation $\mathbf{Y}_{0,\bar{N},1} = \boldsymbol{\Phi}\boldsymbol{\theta}$, from which the vector of unknowns can be estimated as a least squares solution. Haverkamp [6] shows how to fill the matrix $\boldsymbol{\Phi}$ efficiently and we will extend this approach to process sensor roving data. One of his ideas is to sort the equations first on scalar outputs to get rid of one of the Kronecker products, i.e.

$$y_i(k) = \left[\mathbf{C}(i,:)\mathbf{A}^k \;\vdots\; \sum_{\tau=0}^{k-1}\boldsymbol{u}^{\top}(\tau)\otimes\mathbf{C}(i,:)\mathbf{A}^{k-1-\tau} \;\vdots\; \boldsymbol{u}^{\top}(k)\right]\cdot\begin{bmatrix}\boldsymbol{x}(0)\\ \mathrm{vec}(\mathbf{B})\\ (\mathbf{D}(i,:))^{\top}\end{bmatrix}. \quad (2\text{-}71)$$

This yields for each scalar output signal a set of $\bar{N}$ equations (the number of recorded samples) with $(n + mn + m)$ unknowns. When combining the sets of equations for all output channels that depend on different rows of $\mathbf{D}$, the number of unknowns becomes

$(n + mn + \ell m)$:

$$\boldsymbol{\psi} = \mathrm{vec}\left(\mathbf{Y}_{0,1,\bar{N}}^{\top}\right) = \begin{bmatrix} y_1(1) \\ \vdots \\ y_1(\bar{N}) \\ y_2(1) \\ \vdots \\ \vdots \\ y_\ell(\bar{N}) \end{bmatrix} \quad ; \quad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{x}(0) \\ \mathrm{vec}(\mathbf{B}) \\ (\mathbf{D}(1,:))^{\top} \\ \vdots \\ (\mathbf{D}(\ell,:))^{\top} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}(0) \\ \mathrm{vec}(\mathbf{B}) \\ \mathrm{vec}\left(\mathbf{D}^{\top}\right) \end{bmatrix} \tag{2-72}$$

and columns filled with zeros have to be added to $\boldsymbol{\Phi}$ for the unrelated rows of $\mathbf{D}$.

For sensor roving only a small modification is needed. Instead of one initial state vector $\boldsymbol{x}(0)$, there is one for each experiment, so we have

$$\boldsymbol{\psi} = \mathrm{vec}\left(\begin{bmatrix} \mathbf{Y}_{0,1,\bar{N}}^{(0,1)} \\ \mathbf{Y}_{0,1,\bar{N}}^{(1,1)} \\ \mathbf{Y}_{0,1,\bar{N}}^{(0,2)} \\ \vdots \\ \vdots \\ \mathbf{Y}_{0,1,\bar{N}}^{(J,J)} \end{bmatrix}^{\top}\right) \quad ; \quad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{x}^{(1)}(0) \\ \vdots \\ \boldsymbol{x}^{(J)}(0) \\ \mathrm{vec}(\mathbf{B}) \\ (\mathbf{D}(1,:))^{\top} \\ \vdots \\ (\mathbf{D}(\ell,:))^{\top} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}^{(1)}(0) \\ \vdots \\ \boldsymbol{x}^{(J)}(0) \\ \mathrm{vec}(\mathbf{B}) \\ \mathrm{vec}\left(\mathbf{D}^{\top}\right) \end{bmatrix} \tag{2-73}$$

All we have to do is fill the matrix in Eq. (2-71) for each scalar output signal for each experiment and "wire" each of them to the right initial state vector $\boldsymbol{x}^{(j)}(0)$ and right row of $\mathbf{D}$.

However, this results in a large matrix $\boldsymbol{\Phi} \in \mathbb{R}^{J\bar{N}(\ell_0+\ell_j)\times(Jn+nm+m(\ell_0+J\ell_j))}$. The number of elements grows quadratically in the number of sensor roving experiments J. If we throw in some realistic numbers, the size of the matrix $\boldsymbol{\Phi}$ easily reaches the order of magnitude of 10^9 to 10^{10} elements, which for doubles occupies in the order of magnitude of 10 GB to 100 GB of memory.

To reduce the required array size, we make use of QR compression. This is the same method as used by the *LTI System identification toolbox*. [17] It works as follows: For any unitary matrix $\mathbf{Q}$,

$$\arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\psi} - \boldsymbol{\Phi}\boldsymbol{\theta}\|_2 = \boldsymbol{\Phi}^{\dagger}\boldsymbol{\psi} = \left(\mathbf{Q}^{\top}\boldsymbol{\Phi}\right)^{\dagger}\mathbf{Q}^{\top}\boldsymbol{b} = \arg\min_{\boldsymbol{\theta}} \left\|\mathbf{Q}^{\top}\boldsymbol{\psi} - \mathbf{Q}^{\top}\boldsymbol{\Phi}\boldsymbol{\theta}\right\|_2 , \tag{2-74}$$

by lemma Lemma 1 on page 55. If we choose $\mathbf{Q}$ according to the following QR factorisation,

$$\left[\boldsymbol{\Phi} \;\vdots\; \boldsymbol{\psi}\right] = \mathbf{Q}\left[\mathbf{R}_{\boldsymbol{\Phi}} \;\vdots\; \mathbf{R}_{\boldsymbol{\psi}}\right] , \tag{2-75}$$

the minimisation reads as

$$\arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\psi} - \mathbf{A}\boldsymbol{x}\|_2 = \arg\min_{\boldsymbol{\theta}} \left\|\mathbf{R}_{\boldsymbol{\psi}} - \mathbf{R}_{\boldsymbol{\Phi}}\boldsymbol{\theta}\right\|_2 . \tag{2-76}$$

For a very thin matrix $\boldsymbol{\Phi}$ the matrix $\mathbf{R}_{\boldsymbol{\Phi}}$ has a lot of zero rows that provide no information and can be omitted. Hence, this procedure reduces the amount of data required to solve the minimization. This can be done without calculating $\mathbf{Q}$ explicitly.

Since a block diagonal matrix of unitary matrices is again unitary, we can apply this procedure to any selection of rows of the original problem and stack the resulting matrices $\mathbf{R}_{\boldsymbol{\Phi}}$. Furthermore since any product of unitary matrices is again unitary, we can repeat the procedure with the resulting stack of $\mathbf{R}_{\boldsymbol{\Phi}}$ matrices. Finally since Eq. (2-76) remains valid if we add zero columns between columns of $\boldsymbol{\Phi}$ and between the same columns of $\mathbf{R}_{\boldsymbol{\Phi}}$, we can leave out zero columns in $\boldsymbol{\Phi}$ while calculating $\mathbf{R}_{\boldsymbol{\Phi}}$ and add them when needed, i.e. when stacking the $\mathbf{R}_{\boldsymbol{\Phi}}$ matrices.

How can we use this effectively? — Consider a single (scalar) output signal from a single experiment. This signal depends on the matrix $\mathbf{B}$, one row of the matrix $\mathbf{D}$, and one initial state vector $\boldsymbol{x}^{(j)}(0)$. The size of the matrix $\boldsymbol{\Phi}$ for this signal only is $\bar{N} \times (n + mn + m)$ and can be compressed to $\mathbf{R}_{\boldsymbol{\Phi}} \in \mathbb{R}^{(n+mn+m)\times(n+mn+m)}$. We can group $\mathbf{R}_{\boldsymbol{\Phi}}$ matrices that correspond to the same experiment (initial state vector) or to the same row of $\mathbf{D}$, thereby adding proper zero columns, and perform another compression step. Finally the solution can be found from a matrix $\mathbf{R}_{\boldsymbol{\Phi}}$ with $(Jn + mn + m(\ell_0 + J\ell_j))$ rows and columns, and the corresponding matrix $\mathbf{R}_{\boldsymbol{\psi}}$.

This procedure seems to limit the maximum array size that needs to be in memory to $\bar{N} \times (n + mn + m)$. That is certainly possible, but not very efficient. Haverkamp [6] fills the coefficients in $\boldsymbol{\Phi}$ corresponding to $\mathrm{vec}(\mathbf{B})$ as the output sequences of a discrete time state-space system for each element in $\mathrm{vec}(\mathbf{B})$. If we strictly process one scalar output signal at a time, we need to calculate these $n \cdot m$ responses again for each scalar output sequence and each time save only one of the output channels. That is at least $(\ell_0 + J\ell_j)$ times.[3] It is more efficient to save either all outputs or all state sequences for all these state-space systems in one go, but storing that data requires a matrix with $nm\bar{N}(\ell_0 + J\ell_j)$ or $n^2m\bar{N}$ elements respectively. The required memory size to store either of them can be realistically in the order of magnitude of 1 GB to 10 GB, so that is still not always a practical solution.

A further improvement in terms of memory usage can be achieved by splitting the scalar signals in multiple time frames and QR compress them separately. To avoid recalculating the earlier parts of the $n \cdot m$ state trajectories for the coefficients in $\boldsymbol{\Phi}$ corresponding to $\mathrm{vec}(\mathbf{B})$, we store both the $n \cdot m$ output sequences and the $n \cdot m$ final states. By virtue of these final states, we can resume for the next time frame where we stopped. By choosing the time frames sufficiently short, we can split the original problem in bite size chunks for

[3] The coefficients in $\boldsymbol{\Phi}$ corresponding to $\mathrm{vec}(\mathbf{B})$ for each of the reference output channels are equal for all experiments.

the memory of the computer we are working on, independent of the number of samples $\bar{N}$.[4]

The above algorithm minimizes the average error for all output samples rather than for all output channels. Since there is more data from the reference sensor outputs than from the roving sensor outputs, it tends to result in a model that has a poorer fit for the reference sensor outputs than for the roving sensor outputs. That may be remedied by adding extra weight to the roving sensor outputs.

2-2-8 From sensor readings to rigid body motion

We now have an algorithm to model the sensor readings at each position, but to interpret these readings, it needs conversion to two rigid body accelerations in six DOF each. This conversion can be seen as a static gain from $3(J+1)$ accelerometer outputs to 3 linear and 3 angular accelerations for two rigid bodies. To avoid cumbersome formulations, the explanation in this subsection will be as if there were a full sensor grid. Furthermore it is aimed at the movement of the shaker table top, but everything holds as well for the payload top with sensor eight to fourteen instead of sensor one to seven.

The shaker table top frame Ψ_{s} will be the frame of sensor one, which is the one in the centre, see Figure 2-7 on page 10. The homogeneous transformation between these to frames is $\mathbf{I}_4$. The homogeneous matrices that describe the coordinate transformation from the rim sensor frames $\Psi_{\mathrm{s}j}$ to frame Ψ_{s}, read as

$$\mathcal{H}^{\mathrm{s}}_{\mathrm{s}j} = \begin{matrix} \\ \langle 3\rangle \\ \langle 1\rangle \end{matrix} \begin{matrix} \begin{matrix} \langle 3\rangle & \langle 1\rangle \end{matrix} \\ \begin{bmatrix} \mathcal{R}^{\mathrm{s}}_{\mathrm{s}j} & \boldsymbol{p}^{\mathrm{s}}_{\mathrm{s}j} \\ \mathbf{0}^{\top} & 1 \end{bmatrix} \end{matrix} = \left[\begin{array}{ccc:c} c_j & -s_j & 0 & -r_{\mathrm{s}}c_j \\ s_j & c_j & 0 & -r_{\mathrm{s}}s_j \\ 0 & 0 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right], \text{ where } \begin{array}{l} c_j \coloneqq \cos\big((j-2)\cdot\pi/3\big)\ ; \\ s_j \coloneqq \sin\big((j-2)\cdot\pi/3\big)\ ; \\ j \in \{2,3,\ldots,7\}\,. \end{array}$$

In [13, p. 19] the relation between the rigid body acceleration matrix and the measurements was derived as

$$\begin{bmatrix} \ddot{\boldsymbol{P}}^{\mathrm{s},0}_{\mathrm{s}1} & \ddot{\boldsymbol{P}}^{\mathrm{s},0}_{\mathrm{s}2} & \cdots & \ddot{\boldsymbol{P}}^{\mathrm{s},0}_{\mathrm{s}7} \end{bmatrix}(t) = \tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_{\mathrm{s}}(t) \cdot \begin{bmatrix} \boldsymbol{P}^{\mathrm{s}}_{\mathrm{s}1} & \boldsymbol{P}^{\mathrm{s}}_{\mathrm{s}2} & \cdots & \boldsymbol{P}^{\mathrm{s}}_{\mathrm{s}7} \end{bmatrix}. \tag{2-77}$$

On the left hand side is a juxtaposition of the instantaneous acceleration vector of each of the sensors in projective coordinates, expressed in the centre frame Ψ_{s}. This means that every three DOF sensor reading $\ddot{\boldsymbol{p}}^{\mathrm{s}j,0}_{\mathrm{s}j}(t)$ needs to be rotated first:

$$\ddot{\boldsymbol{P}}^{\mathrm{s},0}_{\mathrm{s}j}(t) = \mathcal{H}^{\mathrm{s}}_{\mathrm{s}j}\ddot{\boldsymbol{P}}^{\mathrm{s}j,0}_{\mathrm{s}j}(t) = \begin{bmatrix} \mathcal{R}^{\mathrm{s}}_{\mathrm{s}j}\ddot{\boldsymbol{p}}^{\mathrm{s}j,0}_{\mathrm{s}j}(t) \\ 0 \end{bmatrix}. \tag{2-78}$$

[4] For the sake of completeness: For an overdetermined equation, the size of the final matrix $\mathbf{R}_{\boldsymbol{\Phi}}$ is the square of the length of the parameter vector $\boldsymbol{\theta}$. Hence, one could still run in to memory problems if $(Jn + mn + Jm(\ell_0 + J\ell_j))$ is very high.

The rightmost matrix in Eq. (2-77) is a juxtaposition of each of the sensor locations with respect to the frame Ψ_s, i.e.

$$\boldsymbol{P}^\mathrm{s}_{\mathrm{s}j} = \mathcal{H}^\mathrm{s}_{\mathrm{s}j}\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^\top = \begin{cases}\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^\top & j = 1\,,\\ \begin{bmatrix}-r_\mathrm{s}c_j & -r_\mathrm{s}s_j & 0 & 1\end{bmatrix}^\top & j \in \{2,3,\ldots,7\}\,.\end{cases} \tag{2-79}$$

The acceleration matrix $\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}(t)$ is defined from the time dependent homogeneous transformation matrix between the moving body frame Ψ_s and an inertial reference frame Ψ_0. It can be shown that its entries depend linearly on the linear and angular acceleration of the moving body and quadratically on its angular velocity:

$$\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}(t) \coloneqq \mathcal{H}^\mathrm{s}_0(t)\cdot\ddot{\mathcal{H}}^0_\mathrm{s}(t) = \begin{bmatrix}-\omega_y^2-\omega_z^2 & \omega_x\omega_y-\alpha_z & \omega_x\omega_z+\alpha_y & a_x\\ \omega_x\omega_y+\alpha_z & -\omega_x^2-\omega_z^2 & \omega_y\omega_z-\alpha_x & a_y\\ \omega_x\omega_z-\alpha_y & \omega_y\omega_z+\alpha_x & -\omega_x^2-\omega_y^2 & a_z\\ 0 & 0 & 0 & 0\end{bmatrix}^{\mathrm{s},0}_\mathrm{s}(t)\,. \tag{2-80}$$

From now on we will omit the notation of the time dependence.

Since in our case all sensors are in the plane $z = 0$, the third row of the sensor position matrix is zero. Equation (2-77) tells that in that case the third column of the acceleration matrix $\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}$ cannot be measured with this sensor grid. Excluding the bottom zero row as well, we measure only nine elements of the acceleration matrix $\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}$. We remove the other elements from Eq. (2-77) and then using MATLAB notation it reads as

$$\begin{bmatrix}\mathcal{R}^\mathrm{s}_{\mathrm{s}1}\ddot{\boldsymbol{p}}^{\mathrm{s}1,0}_{\mathrm{s}1} & \mathcal{R}^\mathrm{s}_{\mathrm{s}2}\ddot{\boldsymbol{p}}^{\mathrm{s}2,0}_{\mathrm{s}2} & \cdots & \mathcal{R}^\mathrm{s}_{\mathrm{s}7}\ddot{\boldsymbol{p}}^{\mathrm{s}7,0}_{\mathrm{s}7}\end{bmatrix} = \tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}\Big(1{:}3,[1,2,4]\Big)\cdot\boldsymbol{\Lambda}\,, \tag{2-81}$$

$$\text{where}\quad \boldsymbol{\Lambda} = \begin{bmatrix}-r_\mathrm{s} & 0 & 0\\ 0 & -r_\mathrm{s} & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\underbrace{\begin{bmatrix}0 & c_2 & c_3 & \cdots & c_7\\ 0 & s_2 & s_3 & \cdots & s_7\\ 1 & 1 & 1 & \cdots & 1\end{bmatrix}}_{=:\mathfrak{C}}\,. \tag{2-82}$$

To find an explicit mapping from the modelled sensor readings $\boldsymbol{y}_\mathrm{local}$ to the rigid body acceleration, we need to reshape the remaining elements of $\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}$ to a vector by applying Lemma 6 on page 56 to Eq. (2-81), which results in

$$\underbrace{\begin{bmatrix}\mathcal{R}^\mathrm{s}_{\mathrm{s}1} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathcal{R}^\mathrm{s}_{\mathrm{s}2} & \ddots & \vdots\\ \vdots & \ddots & \ddots & \mathbf{0}\\ \mathbf{0} & \cdots & \mathbf{0} & \mathcal{R}^\mathrm{s}_{\mathrm{s}7}\end{bmatrix}}_{=:\mathcal{R}^\mathrm{s}_{\mathrm{s}\forall}}\cdot\underbrace{\begin{bmatrix}\ddot{\boldsymbol{p}}^{\mathrm{s}1,0}_{\mathrm{s}1}\\ \ddot{\boldsymbol{p}}^{\mathrm{s}2,0}_{\mathrm{s}2}\\ \vdots\\ \ddot{\boldsymbol{p}}^{\mathrm{s}7,0}_{\mathrm{s}7}\end{bmatrix}}_{=:\boldsymbol{y}_\mathrm{sensors}} = \Big(\boldsymbol{\Lambda}^\top\otimes\mathbf{I}_3\Big)\cdot\underbrace{\mathrm{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}^{\mathrm{s},0}_\mathrm{s}\Big(1{:}3,[1,2,4]\Big)\Big)}_{=:\boldsymbol{y}_\mathrm{s}}\,. \tag{2-83}$$

Now the least squares estimate of $\boldsymbol{y}_\mathrm{s}$ given the sensor readings is found to be

$$\hat{\boldsymbol{y}}_\mathrm{s} = \Big(\boldsymbol{\Lambda}^\top\otimes\mathbf{I}_3\Big)^\dagger\,\mathcal{R}^\mathrm{s}_{\mathrm{s}\forall}\boldsymbol{y}_\mathrm{sensors} = \left(\left(\mathrm{diag}\Big(\frac{-2}{6r_\mathrm{s}},\frac{-2}{6r_\mathrm{s}},\frac{1}{7}\Big)\cdot\mathfrak{C}\right)\otimes\mathbf{I}_3\right)\mathcal{R}^\mathrm{s}_{\mathrm{s}\forall}\boldsymbol{y}_\mathrm{sensors}\,. \tag{2-84}$$

The derivation of Eq. (2-84) is shown in Lemma 16 on page 64 and its implementation in MATLAB is shown in Listing B.9 on page 74. The accelerations can be calculated from this vector as follows.

$$\alpha^{\mathrm{s},0}_{\mathrm{s}\,x} \approx y_{\mathrm{s}\,6}\,,\ \text{if}\ \mathrm{E}\Big[y^2_{\mathrm{s}\,6}\Big] \gg \tfrac{1}{2}\,\mathrm{E}\Big[y^2_{\mathrm{s}\,1}\Big]\,, \tag{2-85}$$

$$\alpha^{\mathrm{s},0}_{\mathrm{s}\,y} \approx -y_{\mathrm{s}\,3}\,,\ \text{if}\ \mathrm{E}\Big[y^2_{\mathrm{s}\,3}\Big] \gg \tfrac{1}{2}\,\mathrm{E}\Big[y^2_{\mathrm{s}\,5}\Big]\,, \tag{2-86}$$

$$\alpha^{\mathrm{s},0}_{\mathrm{s}\,z} = \tfrac{1}{2}y_{\mathrm{s}\,2} - \tfrac{1}{2}y_{\mathrm{s}\,4}\,, \tag{2-87}$$

$$a^{\mathrm{s},0}_{\mathrm{s}\,x} = y_{\mathrm{s}\,7}\,, \tag{2-88}$$

$$a^{\mathrm{s},0}_{\mathrm{s}\,y} = y_{\mathrm{s}\,8}\,, \tag{2-89}$$

$$a^{\mathrm{s},0}_{\mathrm{s}\,z} = y_{\mathrm{s}\,9}\,. \tag{2-90}$$

If the sufficient conditions in Eqs. (2-85) and (2-86) hold — i.e. we are allowed to neglect the angular velocity ω_z (see [13, p. 22])— expansion of Eq. (2-84) for the terms of interest yields

$$\begin{bmatrix}\hat{\alpha}_x\\ \hat{\alpha}_y\\ \hat{\alpha}_z\\ \hat{a}_x\\ \hat{a}_y\\ \hat{a}_z\end{bmatrix}_{\mathrm{s}}^{\mathrm{s},0} = \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\\1/7&0&0\\0&1/7&0\\0&0&1/7\end{bmatrix}\begin{bmatrix}\ddot{\boldsymbol{p}}_x\\ \ddot{\boldsymbol{p}}_y\\ \ddot{\boldsymbol{p}}_z\end{bmatrix}_{\mathrm{s}1}^{\mathrm{s}1,0} + \sum_{j=2}^{7}\begin{bmatrix}0&0&-2s_j/(6r_{\mathrm{s}})\\0&0&2c_j/(6r_{\mathrm{s}})\\0&-1/(6r_{\mathrm{s}})&0\\c_j/7&-s_j/7&0\\s_j/7&c_j/7&0\\0&0&1/7\end{bmatrix}\begin{bmatrix}\ddot{\boldsymbol{p}}_x\\ \ddot{\boldsymbol{p}}_y\\ \ddot{\boldsymbol{p}}_z\end{bmatrix}_{\mathrm{s}j}^{\mathrm{s}j,0}\,. \tag{2-91}$$

The vectors $\ddot{\boldsymbol{p}}$ are the accelerations as measured by the sensors: one in the centre and six on the rim. Equation (2-91) is what one may intuitively expect. Starting from the bottom half, we can recognise a rotation in z to align the sensor frames and the mean gives the linear acceleration. The centre sensor does not provide any information on the angular acceleration, because its origin is chosen as the body frame. The angular acceleration in z is the negative mean of the y-measurements of the rim sensors; their y-axes all point in the negative azimuthal direction. The angular accelerations in x and y are provided by measurements in z of the rim sensors. The leverage indeed has this sine/cosine dependency on the position angle. This mean is corrected by a factor 2, because on average these sensors have half of the maximum leverage.

Because the sensor plane has no meaning in a practical situation, we want to change the frame in which the accelerations are expressed from Ψ_{s} to a vertically shifted frame Ψ_{st} in the bottom plane of the pedestal, the plane that is to be mounted to the cold plate. We can describe (the inverse of) this change of frame with

$$\mathfrak{H}^{\mathrm{s}}_{\mathrm{st}} = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&h^{\mathrm{s}}_{\mathrm{st}}\\0&0&0&1\end{bmatrix}\,, \tag{2-92}$$

where $h^{\mathrm{s}}_{\mathrm{st}}$ is the height of the pedestal bottom above the shaker table sensor plane. The

required coordinate transformation is

$$\begin{aligned}\operatorname{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{st},0}(1:3,:)\Big) &= (\boldsymbol{\mathcal{H}}_{\mathrm{st}}^{\mathrm{s}} \otimes \boldsymbol{\mathcal{R}}_{\mathrm{st}}^{\mathrm{s}})^{\top} \cdot \operatorname{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{s},0}(1:3,:)\Big) && \text{(by Lemma 17)} \\ &= \begin{bmatrix} \mathbf{I}_6 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & h_{\mathrm{st}}^{\mathrm{s}} \cdot \mathbf{I}_3 & \mathbf{I}_3 \end{bmatrix} \cdot \operatorname{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{s},0}(1:3,:)\Big) \,. && \text{(2-93)}\end{aligned}$$

To make this calculation, we need explicitly the third column of the acceleration matrix, which is unmeasurable with a coplanar sensor grid. If the conditions in Eq. (2-85) and Eq. (2-86) hold, we assume $\omega_z \approx 0$ in Eq. (2-80) to find the following approximation:

$$\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{s},0} \approx \begin{bmatrix} y_{\mathrm{s}\,1} & y_{\mathrm{s}\,4} & -y_{\mathrm{s}\,3} & y_{\mathrm{s}\,7} \\ y_{\mathrm{s}\,2} & y_{\mathrm{s}\,5} & -y_{\mathrm{s}\,6} & y_{\mathrm{s}\,8} \\ y_{\mathrm{s}\,3} & y_{\mathrm{s}\,6} & y_{\mathrm{s}\,1} + y_{\mathrm{s}\,5} & y_{\mathrm{s}\,9} \\ 0 & 0 & 0 & 0 \end{bmatrix} , \tag{2-94}$$

or equivalently

$$\operatorname{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{s},0}(1:3,:)\Big) \approx \begin{bmatrix} \mathbf{I}_6 & \mathbf{0} \\ \left[\begin{smallmatrix} 0&0&-1&0&0&0 \\ 0&0&0&0&0&-1 \\ 1&0&0&0&1&0 \end{smallmatrix}\right] & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix} \cdot \operatorname{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{s}}^{\mathrm{s},0}(1:3,[1,2,4])\Big) \,. \tag{2-95}$$

Using Eqs. (2-80), (2-93) and (2-95) we find for the accelerations

$$\alpha_{\mathrm{s}\,x}^{\mathrm{st},0} \approx y_{\mathrm{s}\,6} \,, \tag{2-96}$$
$$\alpha_{\mathrm{s}\,y}^{\mathrm{st},0} \approx -y_{\mathrm{s}\,3} \,, \tag{2-97}$$
$$\alpha_{\mathrm{s}\,z}^{\mathrm{st},0} = \tfrac{1}{2} y_{\mathrm{s}\,2} - \tfrac{1}{2} y_{\mathrm{s}\,4} \,, \tag{2-98}$$
$$a_{\mathrm{s}\,x}^{\mathrm{st},0} \approx y_{\mathrm{s}\,7} - h_{\mathrm{st}}^{\mathrm{s}} \cdot y_{\mathrm{s}\,3} \,, \tag{2-99}$$
$$a_{\mathrm{s}\,y}^{\mathrm{st},0} \approx y_{\mathrm{s}\,8} - h_{\mathrm{st}}^{\mathrm{s}} \cdot y_{\mathrm{s}\,6} \,, \tag{2-100}$$
$$a_{\mathrm{s}\,z}^{\mathrm{st},0} \approx y_{\mathrm{s}\,9} + h_{\mathrm{st}}^{\mathrm{s}} \cdot (y_{\mathrm{s}\,1} + y_{\mathrm{s}\,5}) \,. \tag{2-101}$$

In summary of this subsection, we can approximate the six DOF acceleration with respect to the bottom of the pedestal from the shaker table sensors according to the following formula.

$$\begin{aligned}\boldsymbol{y}_{\mathrm{st}} \coloneqq \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ a_x \\ a_y \\ a_z \end{bmatrix}_{\mathrm{s}}^{\mathrm{st},0} \approx \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -h_{\mathrm{st}}^{\mathrm{s}} & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -h_{\mathrm{st}}^{\mathrm{s}} & 0 & 1 & 0 \\ h_{\mathrm{st}}^{\mathrm{s}} & 0 & 0 & 0 & h_{\mathrm{st}}^{\mathrm{s}} & 0 & 0 & 0 & 1 \end{bmatrix} \times \\ \left(\left(\operatorname{diag}\Big(\frac{-2}{6r_{\mathrm{s}}}, \frac{-2}{6r_{\mathrm{s}}}, \frac{1}{7} \Big) \cdot \boldsymbol{\mathcal{C}} \right) \otimes \mathbf{I}_3 \right) \boldsymbol{\mathcal{R}}_{\mathrm{s}\forall}^{\mathrm{s}} \cdot \boldsymbol{y}_{\mathrm{sensors}} \end{aligned} \tag{2-102}$$

Exactly the same machinery can be used to convert the readings from the sensors on the dummy masses to the six DOF acceleration with respect to a point of interest on the centre axis of the CVIP 2 top at some specified height.

2-2-9 Identification results

Measurements were done using the measurement set-up described in Section 2-2-1. Initial experiments using the highest possible sample rate of the equipment, 20 kHz, showed no sign of the assumed passive low pass behaviour of the plant, with a bandwidth below 100 Hz. It shows several resonant modes until about 5 kHz, with a similar or larger magnitude than the foreseen passive resonant modes of the CVIP 2 between 15 Hz and 30 Hz. It is likely that the shaker table introduces modes in the combined plant that would not be present if the CVIP 2 were mounted on a surface with a much higher mass. The sample frequency was subsequently reduced to focus on the baseband resonances of the CVIP 2. Because anti alias and reconstruction filters needed to suppress all resonant modes above the new Nyquist frequency, while not introducing too much phase shift for the remaining modes, a sample frequency of 1 kHz was chosen. This leaves a couple of severe shaker table modes near 225 Hz to be included in the model. Power spectral density (PSD) estimates of the response data, showed that the effect of modes in the horizontal plane of the shaker table top on the vibration power in the vertical movement of the payload was even higher than the effect of the vertical resonant mode of the CVIP 2 itself.

For the PO-MOESP with sensor roving identification experiments, white noise sequences at a sample rate of 1 kHz were used as excitation.[5] This noise was chosen to be partly correlated between the actuators of the same kind, because independent noise sequences excite too little movement in the z direction. Each time, the roving sensor was mounted in a certain position. After waiting a few minutes for the accelerometer readings to settle, for one minute all actuators were excited. A few seconds later only the piezo actuators were excited for one minute and again a few seconds later only the voice coil actuators were excited for one minute. The process was repeated for each roving sensor position. The separation between the actuators appeared necessary, because the influence of the piezo actuators is so small compared to the influence of the voice coil actuators that a single collective identification experiment resulted in a model that describes the influence of the voice coils rather well, while the influence of the piezo actuators was just nearly zero.

The response data was shaved, detrended and highpass filtered at 3 Hz to remove spikes and drift and the accelerometer data was converted to m/s^2. The data was split between only piezo actuation, only voice coil actuation and both piezo and voice coil actuation and processed separately. Hence for each sensor position three sets of data were obtained of one minute each. Preliminary identification results indicated an input-output delay of 2 samples between every actuator-sensor pair. This delay is removed from the data in all results described below.

The algorithms for PO-MOESP with sensor roving described in Sections 2-2-5 to 2-2-7 were applied to the first 40 s of each piece of experimental data. The remaining 20 s of

[5] The AD and DA converters were running at 10 kHz. This allowed the use of digital anti alias and reconstruction filters with a passband frequency of 500 Hz to help the analog anti alias filters with a passband frequency of about 2 kHz

Table 2-1: Variance accounted for on validation data when the model is identified using ordinary PO-MOESP with Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$) on only the reference data of the experiment for roving sensor position 8, without voice coil actuation.

Sensor output	Roving sensor position
	2
Piezo sensor 1	100%
Piezo sensor 2	100%
Piezo sensor 3	99%
Reference accelerometer x	99%
Reference accelerometer y	99%
Reference accelerometer z	91%

each piece of experimental data was used for model validation in terms of VAF, thereby reducing the risk of overfitting. All identification results shown below apply to this validation data. The script used to produce the results below is shown in Listing B.27 on page 109.

Sadly, it appeared impossible to identify a good model for this experimental set-up in the described manner. This is caused by the violation of two assumptions on which the sensor roving algorithms are based. (In Section 2-2-10 on page 39 it will be shown that the algorithms do apply when these assumptions are valid.) To show which assumptions are violated and how this affects the identified model, we will build up the identification step by step, starting with ordinary PO-MOESP based on a single experiment in which only the piezo actuators are excited.

Step 1: Single experiment, only reference sensor data

Table 2-1 shows the result of ordinary PO-MOESP identification using only the reference data of the experiment in which the roving sensor was mounted in position 8, which is in the centre of the payload. Although this is ordinary PO-MOESP, the scripts for sensor roving PO-MOESP were used. The model order was increased until the VAF on the validation data stopped improving, which resulted in $n = 60$. This high order is mostly required for the output of the z direction of the reference accelerometer. Lower order identified models tend to capture only the dynamics of the shaker table and miss the sharp resonant modes of the CVIP 2.

Step 2: All experiments, only reference sensor data

Since we only consider reference sensor data, identification based on all experiments can be done with ordinary PO-MOESP for multiple batches. The result is shown in Table 2-2. Apart from including the reference data from experiment 3 till 14, nothing

Table 2-2: Variance accounted for on validation data when the model is identified using ordinary PO-MOESP with Algorithm 1 for (**B**,**D**) on only the reference data of all sensor roving experiments, without voice coil actuation.

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	98%	98%	98%	98%	98%	96%
Piezo sensor 2	97%	97%	98%	97%	97%	90%
Piezo sensor 3	93%	95%	96%	94%	94%	82%
Reference accelerometer x	88%	88%	86%	86%	87%	93%
Reference accelerometer y	92%	88%	86%	84%	86%	91%
Reference accelerometer z	42%	65%	55%	65%	59%	59%

Sensor output	Roving sensor position						
	8	9	10	11	12	13	14
Piezo sensor 1	98%	99%	99%	97%	98%	99%	99%
Piezo sensor 2	98%	99%	97%	98%	97%	98%	99%
Piezo sensor 3	98%	96%	95%	97%	91%	97%	98%
Reference accelerometer x	67%	85%	76%	94%	83%	82%	93%
Reference accelerometer y	65%	78%	81%	94%	78%	84%	92%
Reference accelerometer z	79%	70%	67%	66%	24%	69%	77%

has changed compared to step 1. One might expect the identified model to improve, because the identification is based on thirteen times as much data. Instead, the results are significantly worse. Visual inspection of the response data of the the batches by their PSD (not printed in this thesis), reveals that the resonant modes change between the experiments. This implies that the dynamics of the set-up change significantly when the roving sensor changes position, even though its mass is in the order of magnitude of only one percent of the total moving mass. This violates the assumption that the matrix **A** is the same for all sensor roving experiments. The same effect is observed for voice coil actuation, but these results are not included in this thesis.

Step 3: single experiment, all data, ordinary PO-MOESP

The second and third column of Table 2-3 show the result of ordinary PO-MOESP identification for all data of experiment 2 without voice coil actuation. Apart from including the response on the roving sensor, everything is the same compared to Step 1 and the results are similar to those in Table 2-1. In this case the results from both algorithms for (**B**,**D**) estimation are shown.

Table 2-3: Variance accounted for on validation data when the model is identified with both the reference and roving data of the experiment for roving sensor position 8, without voice coil actuation.

	Step 3		Step 4	
(**B**,**D**) estimation algorithm	1	2	1	2
Piezo sensor 1	100%	100%	99%	100%
Piezo sensor 2	100%	100%	99%	99%
Piezo sensor 3	99%	99%	98%	99%
Reference accelerometer x	99%	99%	96%	98%
Reference accelerometer y	99%	99%	98%	99%
Reference accelerometer z	90%	90%	28%	89%
Roving accelerometer x	97%	97%	89%	91%
Roving accelerometer y	96%	96%	92%	92%
Roving accelerometer z	90%	89%	63%	82%

Step 4: single experiment, all data, sensor roving PO-MOESP

The last two columns of Table 2-3 are obtained by applying roving sensor algorithm IIb to the data from a single experiment: First, the extended observability matrix of the reference data is estimated, the same as with Step 1. Next, the extended observability matrix of the roving outputs is estimated according to Eq. (2-50) on page 21. The estimated model becomes significantly worse compared to Step 3 and also the (**B**,**D**) estimation algorithms perform differently. What happens here is that some modes that are observable from all nine outputs together, are unobservable from the reference sensors only. More precisely, when looking at the SVD of the reference data, some singular values that are needed to model the roving outputs drop below spurious singular values due to noise. This can be verified by increasing the model order. The added modes have by construction little effect on the reference outputs, while they may include the modes that are needed for the roving outputs. With a model order of about $n = 90$ and (**B**,**D**) estimation algorithm 2, the results are again almost equal to those obtained in Step 3 with $n = 60$. It seems that (**B**,**D**) estimation algorithm 1 is less robust with respect to spurious modes: With this algorithm the results stay behind for higher model orders.

Step 5: all experiments, all data, sensor roving PO-MOESP

Finally for the complete sensor roving identification procedure for the set-up with only piezo actuation, we find the results listed in Tables 2-4 and 2-5. Because of the two violated assumptions: the changing dynamics with respect to the roving sensor position and the unobservability from the reference sensors, bad models result. In this case it was found that increasing the model order did not lead to a significant improvement.

Table 2-4: Variance accounted for on validation data when the model is identified using sensor roving PO-MOESP Algorithm IIb with Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$) on all sensor roving experiments, without voice coil actuation.

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	93%	93%	93%	93%	92%	90%
Piezo sensor 2	92%	93%	93%	92%	92%	85%
Piezo sensor 3	90%	91%	92%	91%	90%	78%
Reference accelerometer x	49%	52%	54%	52%	52%	53%
Reference accelerometer y	62%	60%	63%	61%	61%	61%
Reference accelerometer z	0%	6%	6%	40%	0%	31%
Roving accelerometer x	30%	40%	43%	59%	51%	5%
Roving accelerometer y	0%	36%	37%	59%	10%	0%
Roving accelerometer z	0%	28%	13%	61%	28%	25%

Sensor output	Roving sensor position						
	8	9	10	11	12	13	14
Piezo sensor 1	94%	94%	94%	94%	92%	94%	93%
Piezo sensor 2	94%	94%	93%	94%	92%	93%	94%
Piezo sensor 3	94%	92%	94%	94%	88%	93%	94%
Reference accelerometer x	54%	58%	53%	54%	56%	54%	58%
Reference accelerometer y	62%	64%	62%	68%	63%	62%	69%
Reference accelerometer z	10%	29%	25%	12%	0%	39%	39%
Roving accelerometer x	0%	27%	16%	42%	29%	42%	55%
Roving accelerometer y	19%	65%	54%	58%	15%	2%	63%
Roving accelerometer z	15%	58%	69%	66%	50%	49%	67%

Table 2-5: Variance accounted for on validation data when the model is identified using sensor roving PO-MOESP Algorithm IIb with Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$) on all sensor roving experiments, without voice coil actuation.

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	95%	95%	95%	95%	95%	92%
Piezo sensor 2	94%	94%	95%	94%	94%	87%
Piezo sensor 3	92%	93%	94%	92%	91%	79%
Reference accelerometer x	30%	29%	31%	30%	29%	30%
Reference accelerometer y	25%	22%	22%	21%	21%	24%
Reference accelerometer z	41%	51%	47%	60%	50%	45%
Roving accelerometer x	20%	12%	7%	0%	34%	30%
Roving accelerometer y	26%	18%	0%	0%	0%	0%
Roving accelerometer z	37%	0%	0%	0%	0%	0%

Sensor output	Roving sensor position						
	8	9	10	11	12	13	14
Piezo sensor 1	96%	96%	96%	95%	95%	96%	96%
Piezo sensor 2	96%	96%	95%	95%	94%	96%	96%
Piezo sensor 3	96%	94%	94%	96%	90%	94%	96%
Reference accelerometer x	34%	35%	30%	33%	35%	30%	35%
Reference accelerometer y	28%	28%	24%	28%	27%	23%	30%
Reference accelerometer z	63%	63%	66%	61%	26%	64%	71%
Roving accelerometer x	0%	0%	0%	0%	0%	5%	6%
Roving accelerometer y	0%	0%	4%	0%	0%	0%	42%
Roving accelerometer z	0%	13%	32%	14%	0%	3%	46%

2-2-10 Proof of concept for PO-MOESP with sensor roving

To demonstrate PO-MOESP with sensor roving and to compare the different algorithms presented in this thesis, a simplified experiment is used. The mechanical set-up is unchanged with the nuts on the rods tightened, but the piezo actuators are not used and only the piezo sensors and the accelerometers on the shaker table top are considered as an output: positions 1 to 7, position 1 being the reference sensor position, see Figure 2-7. The voice coils are actuated for 60 seconds with white noise sequences, the same sequences for each experiment. The sample frequency is set to 1 kHz. Two thirds of the collected data, 40 seconds for each roving sensor position, is processed with the algorithms for PO-MOESP with sensor roving described in Sections 2-2-6 and 2-2-7 to identify a model. The remainder of the data is used to validate the model in terms of VAF. This model has 24 outputs: 6 corresponding to the reference sensors and 6 times 3 corresponding to each of the roving sensor outputs.

The script shown in Listing B.28 on page 111 is used to process the data. To be able to compare the results as well as possible, the parameter s is fixed to 75 and in every case the model order $n = 11$ is selected, which appears to be a reasonable choice. By plotting a PSD of the measurement data and the corresponding prediction from the identified models (not printed in this thesis) it is clear that the resonances of the CVIP 2 (near 16 Hz and 30 Hz) are recognizable in the measurement data (both in the piezo sensor readings and in the z-direction of the roving accelerometer), but not in the prediction. The dominant frequency band in the measurement data is above 200 Hz where the resonant modes of the shaker table are found. These dynamics are contained in the models.

Table 2-6 shows the result of using Algorithm Ia for ($\mathbf{A}$,$\mathbf{C}$), where the experiment on roving sensor position 2 is chosen to provide the basis $\mathbf{T}^{(1)}$ and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$). Validation data for the reference sensors is obtained during each of the six experiments. This data is compared to the same reconstruction, because they belong to the same six model outputs. For the roving sensors only one set of validation data is obtained for each position, so for the last three lines of the table every column corresponds to a different model output. The table shows that the model predicts most validation data well, but the acceleration on all roving sensor positions in z direction worse than the other outputs, while the prediction on the reference sensor position in z direction is almost perfect. This is a pattern we will see for all algorithms.

As discussed in Section 2-2-6, Algorithm Ia is asymmetric with respect to the roving sensor positions. The only difference between Table 2-7 and Table 2-6 is that the experiment with the roving sensor on position 3 is selected as the base instead of position 2. We observe that the results on position 2 are slightly worse, while those on position 3 improve. On the other sensor positions we see differences in both directions.

Table 2-6: Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 2 as a basis and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$).

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	97%	97%	97%	97%	97%	97%
Piezo sensor 2	97%	96%	96%	96%	97%	97%
Piezo sensor 3	98%	97%	97%	97%	98%	98%
Reference accelerometer x	97%	97%	97%	97%	97%	97%
Reference accelerometer y	97%	97%	97%	97%	97%	97%
Reference accelerometer z	100%	100%	100%	100%	100%	100%
Roving accelerometer x	97%	97%	96%	97%	97%	97%
Roving accelerometer y	97%	97%	97%	97%	97%	97%
Roving accelerometer z	92%	86%	88%	84%	90%	86%

Table 2-7: Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 3 as a basis and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$).

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	97%	97%	97%	97%	97%	97%
Piezo sensor 2	96%	97%	97%	97%	97%	96%
Piezo sensor 3	97%	98%	98%	98%	98%	97%
Reference accelerometer x	96%	97%	97%	97%	97%	97%
Reference accelerometer y	96%	97%	97%	97%	97%	97%
Reference accelerometer z	100%	100%	100%	100%	100%	100%
Roving accelerometer x	96%	98%	97%	97%	97%	96%
Roving accelerometer y	96%	97%	97%	97%	97%	97%
Roving accelerometer z	91%	86%	89%	85%	91%	86%

Table 2-8: Variance accounted for on validation data when the model is identified using Algorithm Ia (or Algorithm Ib) for ($\mathbf{A}$,$\mathbf{C}$) with the experiment for roving sensor position 2 as a basis and Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$).

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	92%	92%	92%	92%	93%	92%
Piezo sensor 2	91%	91%	91%	91%	91%	91%
Piezo sensor 3	92%	92%	93%	92%	93%	92%
Reference accelerometer x	92%	91%	92%	92%	92%	92%
Reference accelerometer y	92%	92%	92%	92%	93%	92%
Reference accelerometer z	99%	99%	99%	99%	99%	99%
Roving accelerometer x	91%	92%	92%	92%	93%	92%
Roving accelerometer y	92%	92%	92%	92%	92%	92%
Roving accelerometer z	87%	83%	85%	81%	88%	82%

Table 2-8 differs from Table 2-6 only by the estimation of $(\mathbf{B}, \mathbf{D})$ with Algorithm 2 instead of Algorithm 1. The results appear to be significantly worse for this dataset.

In Table 2-9 the results are listed when using Algorithm IIb and Algorithm 1. Compared to Table 2-6 the model seems to improve for all experiments except for the selected base experiment 2, but a comparison to Table 2-7 does not support this observation.

Table 2-10 differs from Table 2-9 only by the estimation of $(\mathbf{B}, \mathbf{D})$ with Algorithm 2 instead of Algorithm 1. Again we observe that Algorithm 1 outperforms Algorithm 2.

Table 2-9: Variance accounted for on validation data when the model is identified using Algorithm IIb (or Algorithm IIa) for ($\mathbf{A}$,$\mathbf{C}$) and Algorithm 1 for ($\mathbf{B}$,$\mathbf{D}$).

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	97%	97%	97%	97%	97%	97%
Piezo sensor 2	96%	97%	97%	97%	97%	97%
Piezo sensor 3	97%	98%	98%	98%	98%	98%
Reference accelerometer x	97%	97%	97%	97%	97%	97%
Reference accelerometer y	97%	97%	97%	97%	97%	97%
Reference accelerometer z	100%	100%	100%	100%	100%	100%
Roving accelerometer x	96%	98%	97%	97%	97%	97%
Roving accelerometer y	96%	97%	97%	97%	97%	97%
Roving accelerometer z	91%	86%	88%	85%	91%	86%

Table 2-10: Variance accounted for on validation data when the model is identified using Algorithm IIb (or Algorithm IIa) for ($\mathbf{A}$,$\mathbf{C}$) and Algorithm 2 for ($\mathbf{B}$,$\mathbf{D}$).

Sensor output	Roving sensor position					
	2	3	4	5	6	7
Piezo sensor 1	92%	92%	93%	93%	93%	92%
Piezo sensor 2	91%	91%	92%	91%	92%	92%
Piezo sensor 3	92%	92%	93%	92%	93%	92%
Reference accelerometer x	92%	92%	93%	92%	93%	92%
Reference accelerometer y	92%	92%	92%	92%	93%	92%
Reference accelerometer z	99%	99%	99%	99%	99%	99%
Roving accelerometer x	91%	92%	92%	92%	93%	91%
Roving accelerometer y	91%	92%	93%	92%	92%	92%
Roving accelerometer z	87%	83%	85%	81%	88%	82%

Chapter 3

Controller design

This chapter is about controller synthesis. The second section concerns a controller for the Cryo Vibration Isolation Platform 2 (CVIP 2) that uses only the piezo sensors and actuators. The proposed controller is a cautious Wiener filter. The first section deals with a controller for the shaker table. The goal is to let the shaker table reproduce floor vibrations as measured on a real cold plate, according to the model obtained in Section 2-1. Janssen Precision Engineering (JPE) asked for a feedforward controller to avoid investing time in a feedback controller for the shaker table. Furthermore, the original plan was to identify both the plant with a passive CVIP 2 and the plant with a controlled CVIP 2 to evaluate the performance of the CVIP 2 controller. This would have been more difficult with a feedback controller on the shaker table, because then the identification would have been based on closed-loop data.[1]

Because no valid plant model was obtained, no meaningful simulations can be made to support the theory in this chapter.

3-1 Shaker table controller

To replicate cold plate vibrations on the shaker table, a causal Wiener filter can be used as a feedforward controller. The goal is to approach the noise generating model with the series connection of the feedforward controller and the transfer function from voice coil actuation to linear shaker table acceleration. This is illustrated in Figure 3-1, where $\boldsymbol{s}$ is a unit covariance independent white noise sequence and $\boldsymbol{\epsilon}$ is the error. The controller synthesis thus reads as:

$$\mathbf{C}_{\mathrm{vc}} = \arg\min_{\mathbf{C}_{\mathrm{vc}}} \|\boldsymbol{\epsilon}\|_2 = \arg\min_{\mathbf{C}_{\mathrm{vc}}} \left\|\mathbf{P}_{\mathrm{st,vc}}\mathbf{C}_{\mathrm{vc}} - \mathbf{G}_{\mathrm{n}}\right\|_2 . \tag{3-1}$$

[1] This issue is discussed in [16, §9.7].

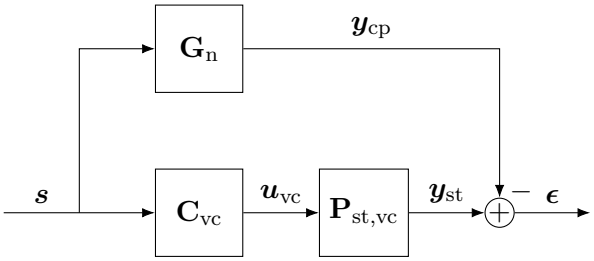


Figure 3-1: Block diagram for voice coil controller synthesis; vc=voice coil, st=shaker table.

The solution in case $\mathbf{C}_\mathrm{vc}$ is restricted to be stable and causal is given by Fraanje [4, Theorem 2.1]:

$$\mathbf{C}_\mathrm{vc} = \mathbf{P}^{\dagger}_{\mathrm{st,vc,o}} \left[\mathbf{P}^{\mathrm{H}}_{\mathrm{st,vc,i}} \mathbf{G}_\mathrm{n} \right]_+ \text{, where} \quad \mathbf{P}_\mathrm{st,vc} = \mathbf{P}_{\mathrm{st,vc,i}} \mathbf{P}_{\mathrm{st,vc,o}} \tag{3-2}$$

is the inner-outer factorisation of $\mathbf{P}_\mathrm{st,vc}$ and $[\square]_+$ is the operator that removes the anti-causal part of the system between the brackets. MATLAB implementations of these and similar operations on state space models are shown in Appendix B-1-2 on page 77.

The following remark is in place: We are considering only the shaker table movement due to the voice coil actuation. Hence, the signal $\boldsymbol{u}_\mathrm{vc}$ can be calculated offline before the start of the experiment and therefore, the error may be further reduced with a non-causal controller $\mathbf{C}_\mathrm{vc}$.

In practice, the movement of the shaker table depends on the piezo actuation as well: Because the mass of the shaker table is low compared to the CVIP 2 and the payload, the usual assumption in active vibration control that the vibration isolator does not influence the floor vibrations, does not hold. As a consequence, the notion of transmissiblity as a property of the vibration isolator, not depending on the dynamics of the floor, or shaker table in this case, is invalid as well.

3-2 Cryo Vibration Isolation Platform controller

3-2-1 Nominal controller

For the CVIP 2 a controller is proposed that receives input from the piezo sensors (ps) and controls the piezo actuators (pa). The top level block diagram of the set-up is shown in Figure 3-2. The voice coil input (vc) is controlled by a feedforward controller $\mathbf{C}_\mathrm{vc}$ that aims to replicate the cold plate vibrations at the shaker table output (st), see Section 3-1, but this output is not used directly for synthesis of the controller $\mathbf{C}_\mathrm{pa}$. The payload acceleration (pl) is passed to a weighting filter $\mathbf{J}$, which results in the error signal $\boldsymbol{e}$ that we try to minimise.

Since no measurement of the payload acceleration is available as a feedback signal, the performance of any controller for this system will be limited. Poles that are not observable

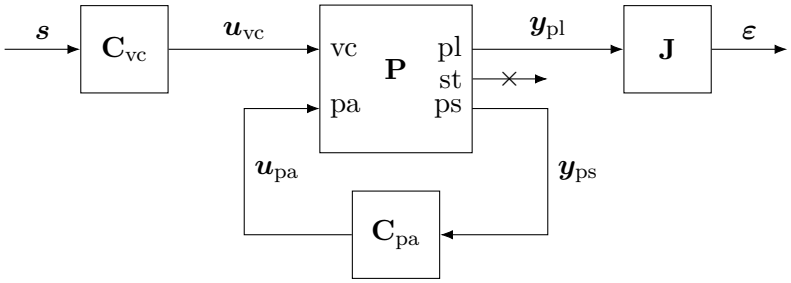


Figure 3-2: Block diagram of the piezo feedback interconnection; vc=voice coil, pa=piezo actuator, pl=payload, st=shaker table, ps=piezo sensor.

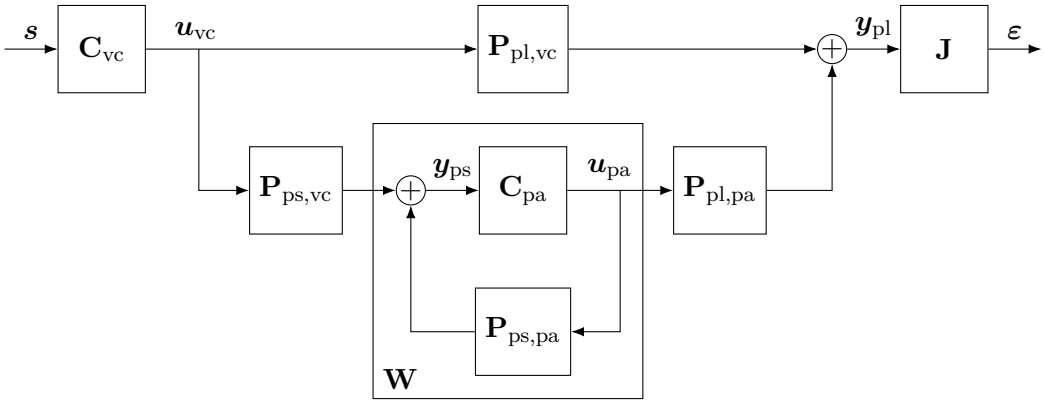


Figure 3-3: Block diagram for nominal piezo actuator controller synthesis; vc=voice coil, pa=piezo actuator, pl=payload, st=shaker table, ps=piezo sensor.

from the piezo sensors cannot be damped or otherwise replaced. The controlled system will also be more prone to model errors than a similar system with a feedback signal of its objective quantity.

Figure 3-3 shows the same block diagram, redrawn in the form used by Fraanje [4, Part 1] to derive the cautious Wiener filter. Unfortunately this problem appears not to fit the feedback framework presented in that source, because there it is explicitly assumed that the input of the feedback controller is equal to the error signal. Instead, it almost fits its feedforward framework [4, Section 2.2]. Following the internal model control (IMC) approach on this framework, we can find the desired controller by first calculating the causal Wiener filter $\mathbf{W}$ that minimises the error signal $\boldsymbol{\varepsilon}$ and subsequently calculating the controller $\mathbf{C}_{\mathrm{pa}}$ such that the feedback connection with the transfer function $\mathbf{P}_{\mathrm{ps,pa}}$ equals $\mathbf{W}$.

If we define for convenience of notation

$$\mathbf{H}_{\boldsymbol{\varepsilon},\boldsymbol{s}} \coloneqq \mathbf{J}\mathbf{P}_{\mathrm{pl,vc}}\mathbf{C}_{\mathrm{vc}}\,, \tag{3-3}$$

$$\mathbf{H}_{\mathrm{ps},\boldsymbol{s}} \coloneqq \mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\,, \tag{3-4}$$

$$\mathbf{H}_{\boldsymbol{\varepsilon},\mathrm{pa}} \coloneqq \mathbf{J}\mathbf{P}_{\mathrm{pl,pa}}\,, \tag{3-5}$$

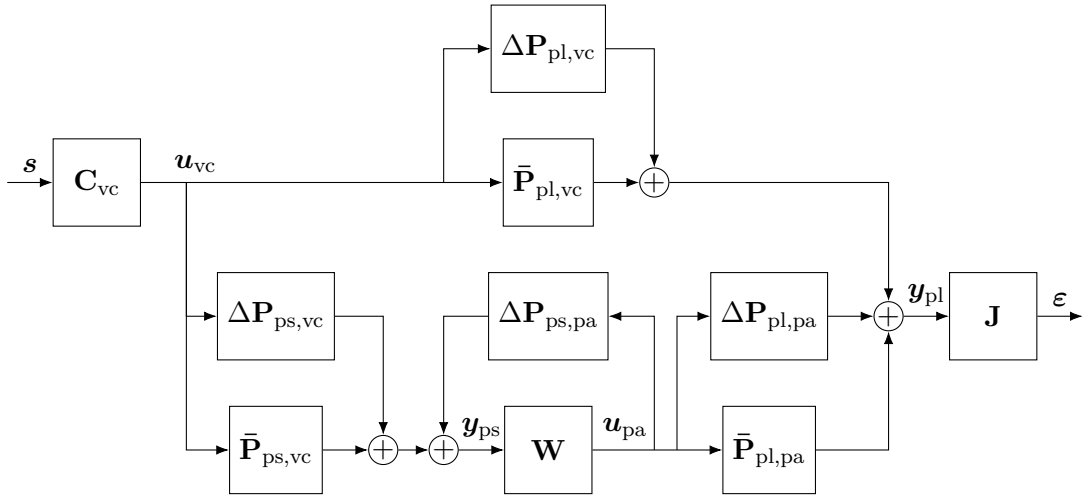


Figure 3-4: Block diagram for robust piezo actuator controller synthesis; vc=voice coil, pa=piezo actuator, pl=payload, st=shaker table, ps=piezo sensor.

then the Wiener filter solution reads as

$$\begin{aligned}\mathbf{W} &= \arg\min_{\mathbf{W}} \|\boldsymbol{\varepsilon}\|_2^2 = \arg\min_{\mathbf{W}} \left\|\mathbf{J}(\mathbf{P}_{\text{pl,vc}} + \mathbf{P}_{\text{pl,pa}}\mathbf{W}\mathbf{P}_{\text{ps,vc}})\mathbf{C}_{\text{vc}}\right\|_2^2 \\ &= \arg\min_{\mathbf{W}} \|\mathbf{H}_{\boldsymbol{\varepsilon},\boldsymbol{s}} + \mathbf{H}_{\boldsymbol{\varepsilon},\text{pa}}\mathbf{W}\mathbf{H}_{\text{ps},\boldsymbol{s}}\|_2^2 = -\mathbf{H}^{\dagger}_{\boldsymbol{\varepsilon},\text{pa,o}} \left[\mathbf{H}^{\text{H}}_{\boldsymbol{\varepsilon},\text{pa,i}}\mathbf{H}_{\boldsymbol{\varepsilon},\boldsymbol{s}}\mathbf{H}^{\text{H}}_{\text{ps},\boldsymbol{s},\text{ci}}\right]_+ \mathbf{H}^{\dagger}_{\text{ps},\boldsymbol{s},\text{co}}\,, \end{aligned} \tag{3-6}$$

where

$$\mathbf{H}_{\boldsymbol{\varepsilon},\text{pa}} = \mathbf{H}_{\boldsymbol{\varepsilon},\text{pa,i}} \cdot \mathbf{H}_{\boldsymbol{\varepsilon},\text{pa,o}} \tag{3-7}$$

is an inner-outer factorisation, and

$$\mathbf{H}_{\text{ps},\boldsymbol{s}} = \mathbf{H}_{\text{ps},\boldsymbol{s},\text{co}} \cdot \mathbf{H}_{\text{ps},\boldsymbol{s},\text{ci}} \tag{3-8}$$

is an outer-inner factorisation. Then

$$\mathbf{C}_{\text{pa}} = \left(\mathbf{I} + \mathbf{W}\mathbf{P}_{\text{ps,pa}}\right)^{-1}\mathbf{W} \tag{3-9}$$

yields the required controller.

3-2-2 Robust controller without feedback uncertainty

Again following the approach of Fraanje [4, Chapter 4], we solve the robust controller synthesis problem by considering independent additive uncertainty on each of the plant model components, as shown in Figure 3-4. The cost function to minimise becomes

$$\mathbf{W} = \arg\min_{\mathbf{W}} \bar{\text{E}}\left[\|\boldsymbol{\varepsilon}\|_2^2\right]\,, \tag{3-10}$$

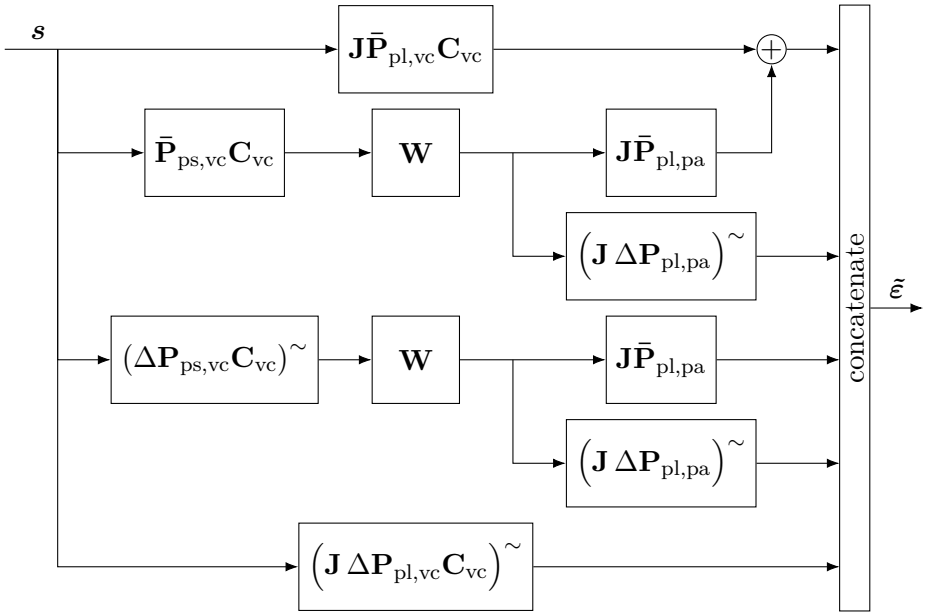


Figure 3-5: Equivalent block diagram for robust piezo actuator controller synthesis without feedback uncertainty; vc=voice coil, pa=piezo actuator, pl=payload, st=shaker table, ps=piezo sensor.

where $\bar{\mathrm{E}}$ denotes the expectation over the model uncertainty.

The solution in case $\Delta\mathbf{P}_{\mathrm{ps,pa}} = 0$ is given by [4, Theorem 4.3]. The first key idea is to assume that the uncertainties are statistically independent from each other and from the nominal path, such that the cross terms in the expectation in Eq. (3-10) cancel, while the remaining terms can be replaced by their minimum phase spectral factors. A useful interpretation of this result is that it is the solution to an equivalent problem in which the error terms corresponding to each statistically independent path are concatenated instead of added. This interpretation is illustrated in Figure 3-5. Using the concatenated error $\tilde{\boldsymbol{\varepsilon}}$, Eq. (3-10) can be rewritten as

$$\begin{aligned}\mathbf{W} &= \arg\min_{\mathbf{W}} \|\tilde{\boldsymbol{\varepsilon}}\|_2^2 \\ &= \arg\min_{\mathbf{W}} \left\| \underbrace{\begin{bmatrix} \mathbf{J}\bar{\mathbf{P}}_{\mathrm{pl,vc}}\mathbf{C}_{\mathrm{vc}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_{m_{\mathrm{pa}}\times\ell_{\mathrm{ps}}} \end{bmatrix}}_{=:\mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\boldsymbol{s}}} + \underbrace{\begin{bmatrix} \mathbf{J}\bar{\mathbf{P}}_{\mathrm{pl,pa}} \\ \left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\sim} \end{bmatrix}}_{=:\mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\mathrm{pa}}} \mathbf{W} \underbrace{\begin{bmatrix} \bar{\mathbf{P}}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}} & \left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\sim} \end{bmatrix}}_{=:\mathbf{H}^{\mathrm{aug}}_{\mathrm{ps},\boldsymbol{s}}} \right\|_2^2,\end{aligned} \tag{3-11}$$

where the uncertainties are zero mean and their magnitude is represented by their

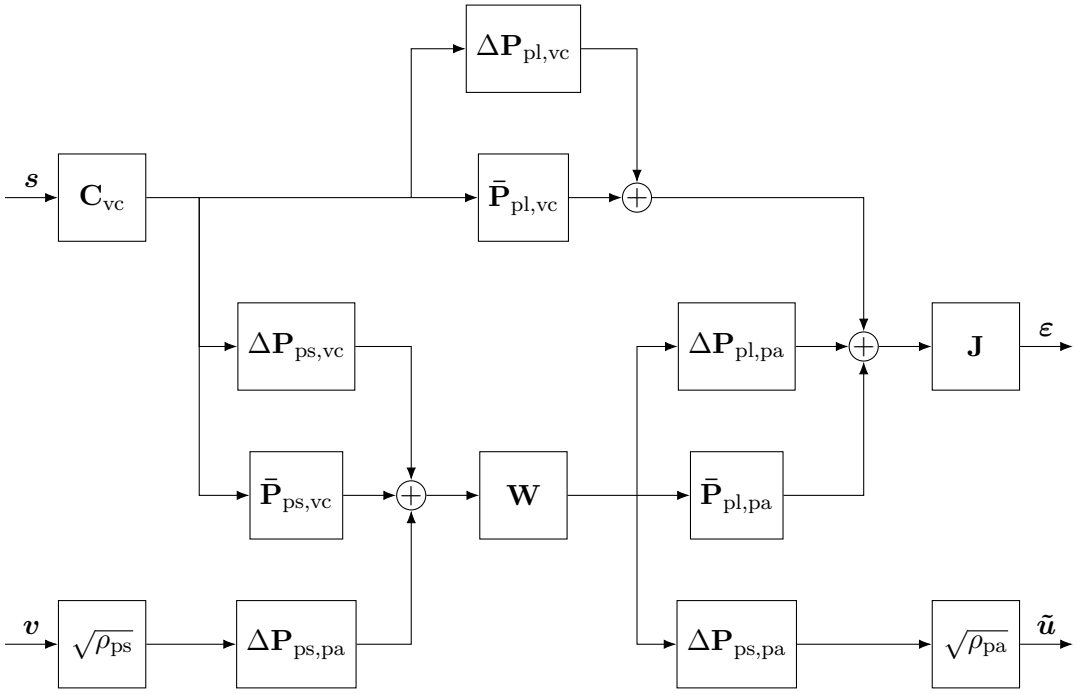


Figure 3-6: Block diagram for robust piezo actuator controller synthesis with artificial feedback uncertainty paths; vc=voice coil, pa=piezo actuator, pl=payload, st=shaker table, ps=piezo sensor.

minimum phase spectral factor denoted with the superscript tilde:

$$\bar{\mathrm{E}}\left[\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right] = \mathbf{0}\,, \tag{3-12}$$

$$\bar{\mathrm{E}}\left[\left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)\left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\mathrm{H}}\right] = \left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\sim}\left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\sim\mathrm{H}}\,, \tag{3-13}$$

$$\bar{\mathrm{E}}\left[\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right] = \mathbf{0}\,, \tag{3-14}$$

$$\bar{\mathrm{E}}\left[\left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\mathrm{H}}\left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)\right] = \left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\sim\mathrm{H}}\left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\sim}\,. \tag{3-15}$$

The second key idea in [4, Theorem 4.3] is the representation of the minimisation as in Eq. (3-11). Because it is in the form of Eq. (3-6), the solution is like a causal Wiener filter:

$$\mathbf{W} = -\mathbf{H}^{\mathrm{aug}\,\dagger}_{\boldsymbol{\varepsilon},\mathrm{pa,o}}\left[\mathbf{H}^{\mathrm{aug}\,\mathrm{H}}_{\boldsymbol{\varepsilon},\mathrm{pa,i}}\mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\boldsymbol{s}}\mathbf{H}^{\mathrm{aug}\,\mathrm{H}}_{\mathrm{ps},\boldsymbol{s},\mathrm{ci}}\right]_{+}\mathbf{H}^{\mathrm{aug}\,\dagger}_{\mathrm{ps},\boldsymbol{s},\mathrm{co}}\,. \tag{3-16}$$

3-2-3 Robust feedback controller with feedback uncertainty

Now we return to the uncertainty model in Figure 3-4 with $\Delta\mathbf{P}_{\mathrm{ps,pa}} \neq \mathbf{0}$.

Fraanje [4, Section 4.5] invokes the small gain theorem, which yields that robust stability

is ensured in case on the unit circle

$$\left\|\Delta\mathbf{P}_{\mathrm{ps,pa}}\right\|_\infty \cdot \|\mathbf{W}\|_\infty < 1\,. \tag{3-17}$$

Since Fraanje assumes that the error signal is the controller input, $\Delta\mathbf{P}_{\mathrm{ps,pa}} = \mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}$ and $\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}} = \mathbf{0}$. The argument continues with the observation that the loop gain $\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\mathbf{W}$ is already contained in one of the uncertain paths, namely (see Figure 3-5)

$$\left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\sim} \cdot \mathbf{W} \cdot \bar{\mathbf{P}}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\,. \tag{3-18}$$

Fraanje argues that by replacing $\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}} \to \sqrt{\rho}\,\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}$ in Eq. (3-11), the loop gain is "pushed down" where $\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}$ is large, such that for some positive ρ Eq. (3-17) is satisfied. The remainder of this uncertain path, $\bar{\mathbf{P}}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}$, is included in this frequency dependent weighting as well. That is undesirable since this factor has nothing to do with the loop gain, but unavoidable if the problem needs to remain solvable as a Wiener filter.

We cannot use this method directly, because in our case the result in Eq. (3-16) does not contain the uncertain term in the feedback path $\Delta\mathbf{P}_{\mathrm{ps,pa}}$. Instead we can add an additional uncertain path that contains the loop gain. Preferably this uncertain path should contain only the loop gain itself and a weighting factor like $\sqrt{\rho}$, but then we cannot rewrite the minimisation like Eqs. (3-6) and (3-11) and hence not solve it as a causal Wiener filter problem. Within this restriction there are three possibilities for adding an uncertain path that contains the loop gain: with $\Delta\mathbf{P}_{\mathrm{ps,pa}}$ as a control effort weighting, cf. [4, §2.2.4]; with $\Delta\mathbf{P}_{\mathrm{ps,pa}}$ as a measurement noise shaping filter, cf. [4, §2.2.3]; or both. This is shown in Figure 3-6. Again we make the simplifying assumption that the model uncertainties are independent, including the two instances of $\Delta\mathbf{P}_{\mathrm{ps,pa}}$ on the artificial uncertainty paths. The cautious Wiener filter becomes

$$\begin{aligned}
\mathbf{W} &= \arg\min_{\mathbf{W}} \bar{\mathrm{E}} \left\| \begin{bmatrix} \tilde{\boldsymbol{\varepsilon}} \\ \tilde{\boldsymbol{u}} \end{bmatrix} \right\|_2^2 \\
&= \arg\min_{\mathbf{W}} \left\| \underbrace{\begin{bmatrix} \mathbf{J}\bar{\mathbf{P}}_{\mathrm{pl,vc}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_{m_{\mathrm{pa}}\times\ell_{\mathrm{ps}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0}_{m_{\mathrm{pa}}\times\ell_{\mathrm{ps}}} \end{bmatrix}}_{=:\mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\boldsymbol{s}}} + \underbrace{\begin{bmatrix} \mathbf{J}\bar{\mathbf{P}}_{\mathrm{pl,pa}} \\ \left(\mathbf{J}\,\Delta\mathbf{P}_{\mathrm{pl,pa}}\right)^{\sim} \\ \sqrt{\rho_{\mathrm{pa}}}\,\Delta\mathbf{P}^{\sim}_{\mathrm{ps,\underline{pa}}} \end{bmatrix}}_{=:\mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\mathrm{pa}}} \mathbf{W} \underbrace{\begin{bmatrix} \left(\bar{\mathbf{P}}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\top} \\ \left(\Delta\mathbf{P}_{\mathrm{ps,vc}}\mathbf{C}_{\mathrm{vc}}\right)^{\sim\top} \\ \sqrt{\rho_{\mathrm{ps}}}\,\Delta\mathbf{P}^{\sim\top}_{\underline{\mathrm{ps}},\mathrm{pa}} \end{bmatrix}^{\top}}_{=:\mathbf{H}^{\mathrm{aug}}_{\mathrm{ps},\boldsymbol{s}}} \right\|_2^2 \\
&= -\mathbf{H}^{\mathrm{aug}\,\dagger}_{\boldsymbol{\varepsilon},\mathrm{pa,o}} \left[\mathbf{H}^{\mathrm{aug}\,\mathrm{H}}_{\boldsymbol{\varepsilon},\mathrm{pa,i}} \mathbf{H}^{\mathrm{aug}}_{\boldsymbol{\varepsilon},\boldsymbol{s}} \mathbf{H}^{\mathrm{aug}\,\mathrm{H}}_{\mathrm{ps},\boldsymbol{s},\mathrm{ci}}\right]_+ \mathbf{H}^{\mathrm{aug}\,\dagger}_{\mathrm{ps},\boldsymbol{s},\mathrm{co}}\,,
\end{aligned} \tag{3-19}$$

where the minimum phase spectral factors of the feedback uncertainty satisfy

$$\bar{\mathrm{E}}\left[\Delta\mathbf{P}_{\underline{\mathrm{ps}},\mathrm{pa}}\right] = \bar{\mathrm{E}}\left[\Delta\mathbf{P}_{\mathrm{ps},\underline{\mathrm{pa}}}\right] = 0\,, \tag{3-20}$$

$$\bar{\mathrm{E}}\left[\Delta\mathbf{P}_{\underline{\mathrm{ps}},\mathrm{pa}} \cdot \Delta\mathbf{P}^{\mathrm{H}}_{\underline{\mathrm{ps}},\mathrm{pa}}\right] = \Delta\mathbf{P}^{\sim}_{\underline{\mathrm{ps}},\mathrm{pa}} \cdot \Delta\mathbf{P}^{\sim\mathrm{H}}_{\underline{\mathrm{ps}},\mathrm{pa}}\,, \tag{3-21}$$

$$\bar{\mathrm{E}}\left[\Delta\mathbf{P}^{\mathrm{H}}_{\mathrm{ps},\underline{\mathrm{pa}}} \cdot \Delta\mathbf{P}_{\mathrm{ps},\underline{\mathrm{pa}}}\right] = \Delta\mathbf{P}^{\sim\mathrm{H}}_{\mathrm{ps},\underline{\mathrm{pa}}} \cdot \Delta\mathbf{P}^{\sim}_{\mathrm{ps},\underline{\mathrm{pa}}}\,. \tag{3-22}$$

To include $\Delta\mathbf{P}^{\sim}_{\mathrm{ps,pa}}$ only as a control effort weighting, ρ_{ps} can be set to 0. Likewise ρ_{pa} can be set to zero to include $\Delta\mathbf{P}^{\sim}_{\mathrm{ps,pa}}$ only as a measurement noise shaping filter.

Chapter 4

Conclusion and recommendations

4-1 Conclusion

A noise generating model in three degrees of freedom was successfully made, based on the available cold plate vibration data. This model contains both the noise spectrum and the pulse tube modulation.

Three existing methods for output only, covariance driven subspace identification for sensor roving experiments, have been applied to Past Outputs Multivariable Output-Error StatesPace (PO-MOESP), which allows the identification of models with a known input from such experiments, i.e. state space models with a non-empty **B** and **D** matrix. It has been mathematically proven that two of these methods (Algorithms Ia and Ib) produce equivalent results, while differing in computation time. A fourth method (Algorithm IIb) was added that is provably equivalent to the third one (Algorithm IIa), but takes less computation time, because it spares one singular value decomposition (SVD). The algorithms have been tested successfully on experimental data.

Due to a violation of two assumptions, sensor roving PO-MOESP does not work on the test set-up that was used to model the Cryo Vibration Isolation Platform 2 (CVIP 2): Moving the roving sensor from one place to another appears to change the dynamics of the plant under test too much and some modes that are relevant to model the payload motion, appear to be unobservable from the reference sensors. It should be noted that the latter is not only problematic for identification, but also for controlling the CVIP 2: Unobservability from the reference sensors implies unobservability from the piezo sensors and modes that are unobservable from the feedback signal cannot be altered by the controller.

An existing method for controller synthesis, a cautious Wiener filter, has been elaborated for the required controller structure, in which the feedback is another signal than the

objective quantity. Because no valid plant model was obtained, it was not possible to test this method.

The initial idea that the passive layer of the CVIP 2 filters out vibrations above approximately 100 Hz in the directions of interest, appeared false. Passive filtering does occur if there is only a purely vertical floor motion. The same probably holds for a purely rotational motion around an axis in a horizontal plane through the passive layer, but this motion cannot be generated on the available shaker table. Floor vibrations in the other three degrees of freedom are not passively isolated and hence, these will become dominant in the payload vibrations in the high frequency region, in all directions. For example, vibrations in the horizontal (x, y) plane of the base of the CVIP 2 easily pass the passive layer, towards the payload. This makes the payload tilt, such that these vibrations become measurable in the vertical (z) direction of the payload.

The shaker table that was used in this project, is unsuited for its task. The shaker table top has little mass compared to the CVIP 2 and payload together. Therefore, the movement of the CVIP 2 influences the motion of the shaker table top. A vibration isolator behaves differently on a wobbly surface than on a fairly solid cold plate. Therefore, experiments on this shaker table have little meaning for the intended use of the CVIP 2. Furthermore, the resonant modes of the shaker table cause errors in the identification procedure, because they tend to be dominant compared to the CVIP 2 modes. This problem is similar to measuring a small quantity as the difference between two much larger quantities.

4-2 Recommendations

4-2-1 Vibration isolator

Realistic transmissibility measurements require mounting the vibration isolator on a solid surface. This can be done by using a real cold plate, by using a heavy table with hammer excitation, or by using a well damped, heavy shaker table.

When the aim is to prevent the payload from vibrating in the vertical direction (z), one needs to consider not only the floor vibrations in the vertical direction, but those in all directions. A next generation of the mechanical design should take into account the transmissibility components from any direction of the base movement to the z movement of the payload. It may, for example, be possible to keep the payload upright with respect to the floor to reduce the transmissibility from horizontal and rotational components of the floor vibrations to the vertical component of the payload vibration.

A controller structure in which the feedback signal is not a measurement of the objective quantity, in this case the acceleration of the payload, is intrinsically prone to model errors, including for example changes in the dynamics due to a different mass (distribution) of the payload, and suffers from difficulties with unobservable modes. In the long term, a fundamental improvement would be to have accelerometers that can be used in a cryostat.

It is good practice to identify a system under conditions that approach the working conditions as good as possible. Extrapolation of a model from ambient conditions to cryogenic conditions is unlikely to provide good results, especially if that requires changing the electronic amplifiers as well. Cryogenic identification will be a challenge when it comes to the acceleration, but measurements from the piezo sensors, with excitation from the piezo actuators and the real pulse tube, may already provide useful insights.

4-2-2 PO-MOESP with sensor roving

Future developments of PO-MOESP with sensor roving could include the following aspects.

- Calculation of the Kalman gain from data obtained from a series of sensor roving experiments. This allows model validation based on the one-step-ahead prediction error, such as the auto-correlation test and the cross correlation test [16, 17].
- Implementation of a parameter optimization method, cf. the LTI toolbox function `doptlti`, that handles data obtained from sensor roving experiments [16, 17].
- An improved method for model order selection. With ordinary PO-MOESP, the singular values of a rank deficient matrix can be used to discriminate between system modes and spurious modes. This method partly looses its value for sensor roving: For Algorithm Ia/b, it is unclear whether selecting the same order for every experiment results in the same modes for every experiment, because the observability may vary between experiments. For Algorithm IIa/b, the singular values are based on the reference data only, so modes that seem spurious may have a high impact on the roving outputs. Currently the only way to find out is to complete the identification procedure and validate the model, which is time consuming. A normalized residue of the step in which the roving sensor data is combined, like Eq. (2-50) on page 21, may provide insight at an early stage to make a suitable model order selection.
- An algorithm to improve the inclusion of modes that are hard to observe from the basis experiment (Algorithm Ia/b) or reference sensor data (Algorithm IIa/b) but important for the remainder of the output data. Initially choosing a high model order during identification and model order reduction afterwards is computationally demanding and seems to leave spurious modes in the model. An alternative idea is to pre-filter the basis experiment or reference sensor input and output data to emphasise modes that are important for the remainder of the data.

Appendix A

Lemmas

A-1 General lemmas

A-1-1 General lemmas involving the Moore-Penrose pseudo-inverse

Definition 1 (Moore-Penrose pseudo-inverse) The Moore-Penrose pseudo-inverse of a matrix $\mathbf{A}$, denoted $\mathbf{A}^\dagger$, is a unique matrix that satisfies the following four criteria [5, §5.5.2]:

$$\mathbf{A}^\dagger = \mathbf{B} \iff \begin{cases} 1. & \mathbf{ABA} = \mathbf{A} \\ 2. & \mathbf{BAB} = \mathbf{B} \\ 3. & (\mathbf{AB})^{\mathrm{H}} = \mathbf{AB} \\ 4. & (\mathbf{BA})^{\mathrm{H}} = \mathbf{BA} \end{cases}$$

□

Lemma 1 (Pseudo-inverse of a product (1)) [18]
If $\mathbf{A} = (\mathbf{BC})^\dagger$ *and* $\mathbf{B}$ *has orthonormal columns, i.e.* $\mathbf{B}^{\mathrm{H}}\mathbf{B} = \mathbf{I}$*, then* $\mathbf{A} = \mathbf{C}^\dagger\mathbf{B}^{\mathrm{H}}$. □

Lemma 2 (Pseudo-inverse of a product (2)) [18]
If $\mathbf{A} = (\mathbf{BC})^\dagger$, $\mathbf{B}$ *is full column rank and* $\mathbf{C}$ *is full row rank, then* $\mathbf{A} = \mathbf{C}^\dagger\mathbf{B}^\dagger$. □

A-1-2 General lemmas involving the Kronecker product

Lemma 3 (Mixed-product property) [5, §1.3.6]
$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \otimes (\mathbf{B} \cdot \mathbf{D})$, *provided that the matrix dimensions are such that all regular matrix products in this equation are valid.* □

Lemma 4 (Transpose of a Kronecker product) [5, §1.3.6]
$(\mathbf{A} \otimes \mathbf{B})^\top = \mathbf{A}^\top \otimes \mathbf{B}^\top$. □

Lemma 5 (Inverse of a Kronecker product) [5, §1.3.6]
If $\mathbf{A}$ *and* $\mathbf{B}$ *are invertible matrices, then* $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$. □

Lemma 6 (Reshaping a Kronecker product) [5, §1.3.7]
$\mathbf{Y} = \mathbf{C}\mathbf{X}\mathbf{B}^\top \quad \Longleftrightarrow \quad \operatorname{vec}(\mathbf{Y}) = (\mathbf{B} \otimes \mathbf{C}) \operatorname{vec}(\mathbf{X})$ □

Lemma 7 (Rank of a Kronecker product) [1]
$\mathbf{A} = \mathbf{B} \otimes \mathbf{C} \quad \Longrightarrow \quad \operatorname{rank}(\mathbf{A}) = \operatorname{rank}(\mathbf{B}) \cdot \operatorname{rank}(\mathbf{C})$ □

Lemma 8 (Pseudo-inverse of a Kronecker product) [7]
$(\mathbf{A}_1 \otimes \mathbf{A}_2 \otimes \cdots \otimes \mathbf{A}_n)^\dagger = \mathbf{A}_1^\dagger \otimes \mathbf{A}_2^\dagger \otimes \cdots \otimes \mathbf{A}_n^\dagger$. □

A-1-3 General lemmas involving Hankel matrices

Data driven subspace model identification (SMI) algorithms require the RQ (actually LQ) factorisation of a row-permuted block Hankel matrix $\mathbf{H}$, which is usually a very fat matrix. Since the large unitary $\mathbf{Q}$ matrix is not required, one can also calculate the Choleski factorisation of $\mathbf{H}\mathbf{H}^\top = \mathbf{R}\mathbf{Q}\mathbf{Q}^\top\mathbf{R}^\top = \mathbf{R}\mathbf{R}^\top$ to find the matrix $\mathbf{R}$ if this factorisation exists (i.e. if $\mathbf{H}\mathbf{H}^\top$ is strictly positive definite), which is computationally cheaper [6, 17]. Still, calculating the relatively small matrix $\mathbf{H}\mathbf{H}^\top$ from the very fat matrix $\mathbf{H}$ is unattractive, both in terms of memory and CPU usage. Haverkamp [6, p. 180] shows a cheaper way to calculate $\mathbf{H}\mathbf{H}^\top$ directly from the sequences that determine $\mathbf{H}$ without storing $\mathbf{H}$ itself. However, because this is a recursive algorithm that is based on the differences along the (sub-)diagonals of $\mathbf{H}\mathbf{H}^\top$, this method is prone to numerical errors when implemented in finite precision.

Lemma 11 shows a non-recursive method to calculate the product of a Hankel matrix and a transposed Hankel matrix of the same size directly from their defining sequences. The result easily extends to block Hankel matrices, by rewriting them as as a row permutation matrix times a vertical concatenation of (scalar) Hankel matrices.

Lemmas 9 and 10 are intermediate results, instrumental to Lemma 11.

Lemma 9 (Product of triangular Hankel matrices) *Let* $\mathbf{H}_U$ *be an upper left triangular Hankel matrix and* $\mathbf{H}_L$ *be a lower right triangular Hankel matrix, then* $\mathbf{H}_U \cdot \mathbf{H}_L$ *is an upper triangular Toeplitz matrix and* $\mathbf{H}_L \cdot \mathbf{H}_U$ *is a lower triangular Toeplitz matrix. Moreover, the triangularity of the Toeplitz matrices is strict if the triangularity of either or both of the Hankel matrices is strict.*

PROOF Using $\boldsymbol{\Delta}_{1,s} = \left[\begin{smallmatrix} \mathbf{0}^\top & 0 \\ \mathbf{I}_{s-1} & \mathbf{0} \end{smallmatrix}\right]$, we can write triangular Hankel matrices as

$$\mathbf{H}_U = \begin{bmatrix} \boldsymbol{h}_U^\top \\ \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s} \\ \vdots \\ \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{s-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_U & \boldsymbol{\Delta}_{1,s}^\top \boldsymbol{h}_U & \cdots & \boldsymbol{\Delta}_{1,s}^{(s-1)\top} \boldsymbol{h}_U \end{bmatrix},$$

$$\mathbf{H}_L = \begin{bmatrix} \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{(s-1)\top} \\ \vdots \\ \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^\top \\ \boldsymbol{h}_L^\top \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L & \cdots & \boldsymbol{\Delta}_{1,s} \boldsymbol{h}_L & \boldsymbol{h}_L \end{bmatrix}.$$

Then since $\boldsymbol{\Delta}_{1,s}^s = \mathbf{0}$,

$$\mathbf{H}_U \cdot \mathbf{H}_L = \begin{bmatrix} \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L & \cdots & \boldsymbol{h}_U^\top \boldsymbol{h}_L \\ \vdots & \ddots & \vdots \\ \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{2s-2} \boldsymbol{h}_L & \cdots & \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L & \cdots & \cdots & \boldsymbol{h}_U^\top \boldsymbol{h}_L \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \boldsymbol{h}_U^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L \end{bmatrix},$$

$$\mathbf{H}_L \cdot \mathbf{H}_U = \begin{bmatrix} \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{(s-1)\top} \boldsymbol{h}_U & \cdots & \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{(2s-2)\top} \boldsymbol{h}_U \\ \vdots & \ddots & \vdots \\ \boldsymbol{h}_L^\top \boldsymbol{h}_U & \cdots & \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_U \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{(s-1)\top} \boldsymbol{h}_U & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \boldsymbol{h}_L^\top \boldsymbol{h}_U & \cdots & \cdots & \boldsymbol{h}_L^\top \boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_U \end{bmatrix}.$$

These are triangular Toeplitz matrices. If $\mathbf{H}_U$ is strictly upper left triangular, the last element of $\boldsymbol{h}_U$ equals zero, so $\boldsymbol{\Delta}_{1,s}^{(s-1)\top} \boldsymbol{h}_U = \mathbf{0}$. Likewise, if $\mathbf{H}_L$ is strictly lower right triangular, $\boldsymbol{\Delta}_{1,s}^{s-1} \boldsymbol{h}_L = \mathbf{0}$, because in that case the first element of $\boldsymbol{h}_L$ equals zero. In both cases the main diagonals of the Toeplitz matrices are zero, so the triangularities are strict. ∎

Lemma 10 (Product of circulant matrices) *Given two equal sized zero ending circulant Hankel matrices,*[1]

$$\mathring{\mathbf{H}}_y = \begin{bmatrix} 0 & \cdots & 0 & y_0 & y_1 & \cdots & \cdots & \cdots & y_M \\ \vdots & \iddots & \iddots & \iddots & \cdots & \cdots & \cdots & \iddots & 0 \\ 0 & y_0 & y_1 & \cdots & \cdots & \cdots & y_M & \iddots & \vdots \\ y_0 & y_1 & \cdots & \cdots & \cdots & y_M & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{s \times N}, \text{ where } N = M + s,$$

[1]The same holds for the product of two circulant Toeplitz matrices, formed by inverting the order of the columns of both $\mathring{\mathbf{H}}_y$ and $\mathring{\mathbf{H}}_u$, since $\mathring{\mathbf{H}}_y \cdot \acute{\mathbf{I}}_N \cdot \acute{\mathbf{I}}_N^\top \cdot \mathring{\mathbf{H}}_u^\top = \mathring{\mathbf{H}}_y \cdot \mathring{\mathbf{H}}_u^\top$, where $\acute{\mathbf{I}}_N$ denotes the $N \times N$ permutation matrix with ones on the anti-diagonal. Beware that reversing the columns implies that the sequence in the top row is also reversed.

and $\mathring{\mathbf{H}}_u$ *defined likewise, the product* $\mathring{\mathbf{H}}_y \cdot \mathring{\mathbf{H}}_u^\top$ *is a Toeplitz matrix of which the first column and first row are given by the sample cross correlation between the sequences* u *and* y*.*

PROOF Let $\mathring{\mathbf{\Delta}}_{1,N} \coloneqq \begin{bmatrix} \mathbf{0}^\top & 1 \\ \mathbf{I}_{N-1} & \mathbf{0} \end{bmatrix}$ denote the circular shift permutation matrix, $\mathring{\boldsymbol{y}}^\top$ the first row of $\mathring{\mathbf{H}}_y$, and $\mathring{\boldsymbol{u}}^\top$ the first row of $\mathring{\mathbf{H}}_u$, then

$$\mathring{\mathbf{H}}_y = \begin{bmatrix} \mathring{\boldsymbol{y}}^\top \\ \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N} \\ \vdots \\ \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^{s-1} \end{bmatrix}; \qquad \mathring{\mathbf{H}}_u^\top = \begin{bmatrix} \mathring{\boldsymbol{u}} & \mathring{\mathbf{\Delta}}_{1,N}^\top \mathring{\boldsymbol{u}} & \cdots & \mathring{\mathbf{\Delta}}_{1,N}^{(s-1)\top} \mathring{\boldsymbol{u}} \end{bmatrix};$$

$$\mathring{\mathbf{H}}_y \cdot \mathring{\mathbf{H}}_u^\top = \begin{bmatrix} \mathring{\boldsymbol{y}}^\top \mathring{\boldsymbol{u}} & \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^{-1} \mathring{\boldsymbol{u}} & \cdots & \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^{1-s} \mathring{\boldsymbol{u}} \\ \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N} \mathring{\boldsymbol{u}} & \mathring{\boldsymbol{y}}^\top \mathring{\boldsymbol{u}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^{-1} \mathring{\boldsymbol{u}} \\ \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^{s-1} \mathring{\boldsymbol{u}} & \cdots & \mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N} \mathring{\boldsymbol{u}} & \mathring{\boldsymbol{y}}^\top \mathring{\boldsymbol{u}} \end{bmatrix},$$

since $\mathring{\mathbf{\Delta}}_{1,N}^\top = \mathring{\mathbf{\Delta}}_{1,N}^{-1}$. By virtue of the padded zeros,

$$\mathring{\boldsymbol{y}}^\top \mathring{\mathbf{\Delta}}_{1,N}^k \mathring{\boldsymbol{u}} = \begin{cases} \begin{bmatrix} y_k & \cdots & y_M \end{bmatrix} \cdot \begin{bmatrix} u_0 & \cdots & u_{M-k} \end{bmatrix}^\top & k \geqslant 0\,, \\ \begin{bmatrix} y_0 & \cdots & y_{M+k} \end{bmatrix} \cdot \begin{bmatrix} u_{-k} & \cdots & u_M \end{bmatrix}^\top & k < 0\,, \end{cases}$$

which is the sample cross correlation between the sequences u and y. ∎

A MATLAB implementation of Lemma 10 is shown in Listing B.4 on page 70.

Lemma 11 (Product of Hankel matrices) *The result of a Hankel matrix times the transpose of another Hankel matrix of equal size can be calculated from their unique elements directly, without constructing the Hankel matrices themselves as shown below. (This saves memory and computation time for very fat Hankel matrices.)*

PROOF Consider the Hankel matrices corresponding to a scalar output sequence y and a scalar input sequence u. For large N the product of interest is almost the product of two circulant Hankel matrices. By writing both operands as a sum of a strictly upper left triangular Hankel matrix, a circulant matrix and a strictly lower right triangular Hankel matrix, we obtain the product of two circulant Hankel matrices plus the sum of six pairs of triangular Hankel matrices. The first can be calculated according to Lemma 10 and four of the latter can be calculated according to Lemma 9.

$$\mathbf{Y}_{0,s,N} \cdot \mathbf{U}_{0,s,N}^\top =$$

$$
\begin{bmatrix} y_0 & \cdots & y_{s-2} & y_{s-1} & \cdots & y_{N-s+1} & \cdots & y_{N-1} \\ \vdots & \iddots & y_{s-1} & \iddots & \iddots & \vdots & \iddots & y_N \\ y_{s-2} & \iddots & \vdots & \iddots & \iddots & y_{N-1} & \iddots & \vdots \\ y_{s-1} & \cdots & y_{2s-3} & \cdots & y_{N-1} & y_N & \cdots & y_{N+s-2} \end{bmatrix} \cdot \left[\begin{array}{ccc:c} u_0 & \cdots & u_{s-2} & u_{s-1} \\ \vdots & \iddots & \iddots & \vdots \\ u_{s-2} & u_{s-1} & \cdots & u_{2s-3} \\ \hdashline u_{s-1} & \iddots & \iddots & \vdots \\ \vdots & \iddots & \iddots & u_{N-1} \\ \hdashline u_{N-s+1} & \cdots & u_{N-1} & u_N \\ \vdots & \iddots & \iddots & \vdots \\ u_{N-1} & u_N & \cdots & u_{N+s-2} \end{array}\right]
$$

$$
= \begin{bmatrix} 0 & \cdots & 0 & y_{s-1} & \cdots & \cdots & \cdots & y_{N-1} \\ \vdots & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots & 0 \\ 0 & \iddots & \iddots & \iddots & \iddots & \iddots & \iddots & \vdots \\ y_{s-1} & \cdots & \cdots & \cdots & y_{N-1} & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & u_{s-1} \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & \iddots & \vdots \\ u_{s-1} & \iddots & \iddots & \vdots \\ \vdots & \iddots & \iddots & u_{N-1} \\ \vdots & \iddots & \iddots & 0 \\ \vdots & \iddots & \iddots & \vdots \\ u_{N-1} & 0 & \cdots & 0 \end{bmatrix} +
$$

$$
\begin{bmatrix} y_0 & \cdots & y_{s-2} \\ \vdots & \iddots & 0 \\ y_{s-2} & \iddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & u_{s-1} \\ \vdots & \iddots & \iddots & \vdots \\ 0 & u_{s-1} & \cdots & u_{2s-3} \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \iddots & y_{s-1} \\ 0 & \iddots & \vdots \\ y_{s-1} & \cdots & y_{2s-3} \end{bmatrix} \cdot \begin{bmatrix} u_0 & \cdots & u_{s-2} & 0 \\ \vdots & \iddots & \iddots & \vdots \\ u_{s-2} & 0 & \cdots & 0 \end{bmatrix} +
$$

$$
\begin{bmatrix} y_{N-s+1} & \cdots & y_{N-1} \\ \vdots & \iddots & 0 \\ y_{N-1} & \iddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & u_N \\ \vdots & \iddots & \iddots & \vdots \\ 0 & u_N & \cdots & u_{N+s-2} \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \iddots & y_N \\ 0 & \iddots & \vdots \\ y_N & \cdots & y_{N+s-2} \end{bmatrix} \cdot \begin{bmatrix} u_{N-s+1} & \cdots & u_{N-1} & 0 \\ \vdots & \iddots & \iddots & \vdots \\ u_{N-1} & 0 & \cdots & 0 \end{bmatrix} +
$$

$$
\begin{bmatrix} y_0 & \cdots & y_{s-2} \\ \vdots & \iddots & 0 \\ y_{s-2} & \iddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} u_0 & \cdots & u_{s-2} & 0 \\ \vdots & \iddots & \iddots & \vdots \\ u_{s-2} & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \iddots & y_N \\ 0 & \iddots & \vdots \\ y_N & \cdots & y_{N+s-2} \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & u_N \\ \vdots & \iddots & \iddots & \vdots \\ 0 & u_N & \cdots & u_{N+s-2} \end{bmatrix} =
$$

$$
\begin{bmatrix} t_0 & \boldsymbol{t}_{\mathrm{r}}^\top \\ \boldsymbol{t}_{\mathrm{c}} & \ddots \end{bmatrix} + \begin{bmatrix} y_0 & \cdots & y_{s-2} \\ \vdots & \iddots & 0 \\ y_{s-2} & \iddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} u_0 & \cdots & u_{s-2} & 0 \\ \vdots & \iddots & \iddots & \vdots \\ u_{s-2} & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \iddots & y_N \\ 0 & \iddots & \vdots \\ y_N & \cdots & y_{N+s-2} \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & 0 & u_N \\ \vdots & \iddots & \iddots & \vdots \\ 0 & u_N & \cdots & u_{N+s-2} \end{bmatrix}
$$

The Toeplitz matrix $\begin{bmatrix} t_0 & \boldsymbol{t}_{\mathrm{r}}^\top \\ \boldsymbol{t}_{\mathrm{c}} & \ddots \end{bmatrix}$ is defined by the top left scalar t_0, the remainder of the top row $\boldsymbol{t}_{\mathrm{r}}^\top$ and the remainder of the first column $\boldsymbol{t}_{\mathrm{c}}$ as follows:

$$
t_0 \coloneqq R_{yu}(0)\,,
$$

$$
\boldsymbol{t}_{\mathrm{r}}^\top \coloneqq \begin{bmatrix} R_{yu}(-1) \\ \vdots \\ R_{yu}(1-s) \end{bmatrix}^\top + \begin{bmatrix} y_0 \\ \vdots \\ y_{s-2} \end{bmatrix}^\top \cdot \begin{bmatrix} 0 & \cdots & 0 & u_{s-1} \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & \iddots & \vdots \\ u_{s-1} & \cdots & \cdots & u_{2s-3} \end{bmatrix} + \begin{bmatrix} y_{N-s+1} \\ \vdots \\ y_{N-1} \end{bmatrix}^\top \cdot \begin{bmatrix} 0 & \cdots & 0 & u_N \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & \iddots & \vdots \\ u_N & \cdots & \cdots & u_{N+s-2} \end{bmatrix},
$$

$$\boldsymbol{t}_{\mathrm{c}} \coloneqq \begin{bmatrix} R_{yu}(1) \\ \vdots \\ R_{yu}(s-1) \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 & y_{s-1} \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & \iddots & \vdots \\ y_{s-1} & \cdots & \cdots & y_{2s-3} \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ \vdots \\ u_{s-2} \end{bmatrix} + \begin{bmatrix} 0 & \cdots & 0 & y_N \\ \vdots & \iddots & \iddots & \vdots \\ 0 & \iddots & \iddots & \vdots \\ y_N & \cdots & \cdots & y_{N+s-2} \end{bmatrix} \cdot \begin{bmatrix} u_{N-s+1} \\ \vdots \\ u_{N-1} \end{bmatrix},$$

$$\text{where} \qquad R_{yu}(k) \coloneqq \begin{cases} \begin{bmatrix} y_{s-1+k} & \cdots & y_{N-1} \end{bmatrix} \cdot \begin{bmatrix} u_{s-1} & \cdots & u_{N-1-k} \end{bmatrix}^{\top} & k \geqslant 0\,, \\ R_{uy}(-k) & k < 0\,.^{2} \end{cases} \qquad \blacksquare$$

Hence, for Hankel matrices of size $s \times N$ the calculation consists of a sum of a Toeplitz matrix and two products of triangular matrices of size $(s-1)$. This Toeplitz matrix is determined by a sample cross covariance between two sequences of length $(N-s+1)$ (which has an FFT *based implementation that is efficient in case of a large number of samples) and four products of a triangular matrix and a vector of length $(s-1)$.* □

A MATLAB implementation of Lemma 11 is shown in Listing B.8 on page 73. This implementation is generalised for block Hankel matrices and applied in the implementation of the Past Outputs Multivariable Output-Error StatesPace (PO-MOESP) compression, shown in Listing B.21 on page 96.

A-1-4 Other general lemmas

Lemma 12 (Sylvester's inequality) [16, §2.3] *Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$,* $\operatorname{rank}(\mathbf{A}) + \operatorname{rank}(\mathbf{B}) - n \leqslant \operatorname{rank}(\mathbf{AB}) \leqslant \min\bigl(\operatorname{rank}(\mathbf{A})\,, \operatorname{rank}(\mathbf{B})\bigr)\,.$ □

A-2 Derivations for PO-MOESP with sensor roving

Lemma 13 (Instrumental variable matrix for sensor roving PO-MOESP) *The matrix $\mathbf{Z}_N^{(j)}$ defined in Eq. (2-15) on page 14 is a valid instrumental variable.*

PROOF Since we are looking at a single experiment, we can drop the superscript (j). An instrumental variable $\mathbf{Z}_N$ is valid if and only if [16, §9.4]

$$\begin{cases} \lim\limits_{N\to\infty} \frac{1}{N} \mathbf{E}_{s,s,N} \mathbf{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}_N^{\top} = 0\,, \\ \operatorname{rank}\Bigl(\lim\limits_{N\to\infty} \frac{1}{N} \mathbf{X}_{s,N} \mathbf{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}_N^{\top} \Bigr) = n\,. \end{cases} \tag{A-1}$$

[2] $R_{yu}(k)$ can be calculated with the MATLAB command `[Ryu,k]=xcorr(y(s:N),u(s:N),s-1);` where `y(s)` $= y_{s-1}$, because MATLAB indexing starts at 1. This function automatically chooses an fast Fourier transform (FFT) based algorithm if that is computationally advantageous.

Consider the permutation matrix $\boldsymbol{\mathcal{P}} \in \mathbb{R}^{s(\ell_0+\ell_j)\times s(\ell_0+\ell_j)}$ that achieves the following row reordering:

$$\boldsymbol{\mathcal{P}} \cdot \begin{bmatrix} \mathcal{O}_s(\mathbf{A}, \mathbf{C}_0) \\ \mathcal{O}_s(\mathbf{A}, \mathbf{C}_j) \end{bmatrix} = \mathcal{O}_s\left(\mathbf{A}, \begin{bmatrix} \mathbf{C}_0 \\ \mathbf{C}_j \end{bmatrix}\right) ,$$

then

$$\boldsymbol{\mathcal{P}} \cdot \begin{bmatrix} \mathbf{Y}_{0,s,N}^{(0,j)} \\ \mathbf{Y}_{0,s,N}^{(j,j)} \end{bmatrix} = \mathbf{Y}'_{0,s,N} \qquad \Rightarrow \qquad \mathbf{Z}'_N = \begin{bmatrix} \mathbf{U}_{0,s,N} \\ \mathbf{Y}'_{0,s,N} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{sm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{P}} \end{bmatrix} \cdot \mathbf{Z}_N ,$$

where $\mathbf{Y}'_{0,s,N}$ and $\mathbf{Z}'_N$ denote the respective matrices in the original PO-MOESP implementation. For $\mathbf{Z}'_N$ it is proven in [16, §9.6] that

$$\begin{cases} \lim\limits_{N\to\infty} \frac{1}{N} \mathbf{E}_{s,s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}'^{\top}_N = \mathbf{0} , \\ \operatorname{rank}\left(\lim\limits_{N\to\infty} \frac{1}{N} \mathbf{X}_{s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}'^{\top}_N \right) = n . \end{cases}$$

It follows that

$$\lim_{N\to\infty} \tfrac{1}{N} \mathbf{E}_{s,s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}^{\top}_N = \lim_{N\to\infty} \tfrac{1}{N} \mathbf{E}_{s,s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}'^{\top}_N \begin{bmatrix} \mathbf{I}_{sm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{P}} \end{bmatrix} = \mathbf{0} \begin{bmatrix} \mathbf{I}_{sm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{P}} \end{bmatrix} = \mathbf{0} ,$$

and using Lemma 12:

$$\begin{aligned} \operatorname{rank}\left(\lim_{N\to\infty} \tfrac{1}{N} \mathbf{X}_{s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}'^{\top}_N \right) + \operatorname{rank}\left(\begin{bmatrix} \mathbf{I}_{sm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{P}} \end{bmatrix} \right) - s(m+\ell_0+\ell_j) &= n \\ \leqslant \operatorname{rank}\left(\lim_{N\to\infty} \tfrac{1}{N} \mathbf{X}_{s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}^{\top}_N \right) & \\ \leqslant \min\left(\operatorname{rank}\left(\lim_{N\to\infty} \tfrac{1}{N} \mathbf{X}_{s,N} \boldsymbol{\Pi}^{\perp}_{\mathbf{U}_{s,s,N}} \mathbf{Z}'^{\top}_N \right) , \operatorname{rank}\left(\begin{bmatrix} \mathbf{I}_{sm} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\mathcal{P}} \end{bmatrix} \right) \right) &= n . \end{aligned}$$

This shows that Eq. (A-1) is satisfied and completes the proof. ∎

Lemma 14 (Derivation of Eq. (2-66) on page 25) *Given Eqs.* (2-64) *and* (2-65) *on page 25 for all experiments* $j \in \{1, \ldots, J\}$*, the least squares estimate of* $(\mathbf{B}, \mathbf{D})$ *is given by Eq.* (2-66) *on page 25.*

PROOF Combining Eqs. (2-64) and (2-65) for all J experiments yields

$$
\underbrace{\begin{bmatrix} \boldsymbol{\Upsilon}^{(0|1)} \\ \vdots \\ \boldsymbol{\Upsilon}^{(0|J)} \\ \hdashline \boldsymbol{\Upsilon}^{(1)} \\ \vdots \\ \boldsymbol{\Upsilon}^{(J)} \end{bmatrix}}_{=:\check{\boldsymbol{\Upsilon}}} =
\left[\begin{array}{c:ccc}
\mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_0,s}, \mathfrak{O}_s^{(0)\perp\top}\right) & \mathbf{0} & \cdots & \mathbf{0} \\
\vdots & \vdots & & \vdots \\
\mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_0,s}, \mathfrak{O}_s^{(0)\perp\top}\right) & \mathbf{0} & \cdots & \mathbf{0} \\
\hdashline
\mathbf{0} & \mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_j,s}, \mathfrak{O}_s^{(1)\perp\top}\right) & \ddots & \vdots \\
\vdots & \ddots & \ddots & \mathbf{0} \\
\mathbf{0} & \cdots & \mathbf{0} & \mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_j,s}, \mathfrak{O}_s^{(J)\perp\top}\right)
\end{array}\right] \times
$$

$$
\underbrace{\left[\begin{array}{c:c:ccc}
\begin{bmatrix} \mathbf{0} \\ \mathfrak{O}_{s-1}^{(0)} \end{bmatrix} & \begin{bmatrix} \mathbf{I}_{\ell_0} \\ \mathbf{0} \end{bmatrix} & \mathbf{0} & \cdots & \mathbf{0} \\
\hdashline
\begin{bmatrix} \mathbf{0} \\ \mathfrak{O}_{s-1}^{(1)} \end{bmatrix} & \mathbf{0} & \begin{bmatrix} \mathbf{I}_{\ell_j} \\ \mathbf{0} \end{bmatrix} & \ddots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \mathbf{0} \\
\begin{bmatrix} \mathbf{0} \\ \mathfrak{O}_{s-1}^{(J)} \end{bmatrix} & \mathbf{0} & \cdots & \mathbf{0} & \begin{bmatrix} \mathbf{I}_{\ell_j} \\ \mathbf{0} \end{bmatrix}
\end{array}\right]}_{=:\boldsymbol{\mathcal{M}}} \cdot
\left[\begin{array}{c} \mathbf{B} \\ \hdashline \mathbf{D}_0 \\ \hdashline \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_J \end{array}\right]
$$

$$
= \underbrace{\left[\begin{array}{c:c}
\frac{\mathbf{I}_{s(s\ell_0-n)}}{\sqrt{J}} & \mathbf{0} \\
\vdots & \vdots \\
\frac{\mathbf{I}_{s(s\ell_0-n)}}{\sqrt{J}} & \mathbf{0} \\
\hdashline
\mathbf{0} & \mathbf{I}_{Js(s\ell_j-n)}
\end{array}\right]}_{=:\boldsymbol{\Phi}} \cdot \underbrace{\operatorname{diag}\left(\begin{array}{c}
\sqrt{J} \cdot \mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_0,s}, \mathfrak{O}_s^{(0)\perp\top}\right), \\
\mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_j,s}, \mathfrak{O}_s^{(1)\perp\top}\right), \\
\ldots, \\
\mathfrak{O}_s\left(\boldsymbol{\Delta}_{\ell_j,s}, \mathfrak{O}_s^{(J)\perp\top}\right)
\end{array}\right)}_{=:\boldsymbol{\Omega}} \cdot \boldsymbol{\mathcal{M}} \cdot \begin{bmatrix} \mathbf{B} \\ \mathbf{D} \end{bmatrix} . \quad \text{(A-2)}
$$

The matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\mathcal{M}}$ are clearly full column rank. The matrix $\boldsymbol{\Omega}$ is full column rank if and only if the anti-diagonal blocks of each of the upper left triangular blocks in $\boldsymbol{\Omega}$ are full column rank, which is stated in the condition in Eq. (2-68) on page 25. Furthermore the product $(\boldsymbol{\Phi\Omega\mathcal{M}})$ is a thin matrix ($\in \mathbb{R}^{Js(s(\ell_0+\ell_j)-2n)\times(n+\ell_0+J\ell_j)}$), so Eq. (A-2) is overdetermined. Hence, using Lemma 1, the least squares solution of this equation can

be written as

$$\begin{bmatrix}\hat{\mathbf{B}}\\ \hat{\mathbf{D}}\end{bmatrix} = (\boldsymbol{\Phi}\boldsymbol{\Omega}\mathfrak{M})^{\dagger}\,\check{\boldsymbol{\Upsilon}} = (\boldsymbol{\Omega}\mathfrak{M})^{\dagger}\,\boldsymbol{\Phi}^{\top}\check{\boldsymbol{\Upsilon}} = (\boldsymbol{\Omega}\mathfrak{M})^{\dagger}\begin{bmatrix}\frac{1}{\sqrt{J}}\sum_{j=1}^{J}\boldsymbol{\Upsilon}^{(0|j)}\\ \boldsymbol{\Upsilon}^{(1)}\\ \vdots\\ \boldsymbol{\Upsilon}^{(J)}\end{bmatrix}. \tag{A-3}$$

The latter expression resembles Eq. (2-66) on page 25. ■

A-3 Derivations for Section 2-2-8

Lemma 15 (Derivation of a pseudo-inverse required in Lemma 16) *The Moore-Penrose pseudo-inverse of the matrix*

$$\mathfrak{C} = \begin{bmatrix}0 & \cos\!\left(0\cdot\frac{2\pi}{N}\right) & \cos\!\left(1\cdot\frac{2\pi}{N}\right) & \cdots & \cos\!\left((N-1)\cdot\frac{2\pi}{N}\right)\\ 0 & \sin\!\left(0\cdot\frac{2\pi}{N}\right) & \sin\!\left(1\cdot\frac{2\pi}{N}\right) & \cdots & \sin\!\left((N-1)\cdot\frac{2\pi}{N}\right)\\ 1 & 1 & 1 & \cdots & 1\end{bmatrix} \in \mathbb{R}^{3\times(N+1)},$$

with $2 < N \in \mathbb{N}$*, is given by* $\mathfrak{C}^{\dagger} = \mathfrak{C}^{\top}\mathfrak{D}$*, where* $\mathfrak{D} \coloneqq \operatorname{diag}\!\left(\frac{2}{N}, \frac{2}{N}, \frac{1}{N+1}\right)$.

Proof Consider Lagrange's trigononmetric identities:

$$\sum_{n=1}^{N}\cos(n\vartheta) = -\frac{1}{2} + \frac{\sin\!\left(N\vartheta+\vartheta/2\right)}{2\sin\!\left(\vartheta/2\right)},$$
$$\sum_{n=1}^{N}\sin(n\vartheta) = \frac{1}{2}\cot\!\left(\frac{\vartheta}{2}\right) - \frac{\cos\!\left(N\vartheta+\vartheta/2\right)}{2\sin\!\left(\vartheta/2\right)}.$$

Substitution of $\vartheta = 2\pi k/N$, with $k \in \mathbb{Z}$ and $k/N \notin \mathbb{Z}$ yields

$$\sum_{n=0}^{N-1}\cos\!\left(kn\cdot\frac{2\pi}{N}\right) = -\frac{1}{2} + \frac{\sin\!\left(2\pi k+\pi k/N\right)}{2\sin\!\left(\pi k/N\right)} = 0,$$
$$\sum_{n=0}^{N-1}\sin\!\left(kn\cdot\frac{2\pi}{N}\right) = \frac{1}{2}\cot\!\left(\frac{\pi k}{N}\right) - \frac{\cos\!\left(2\pi k+\pi k/N\right)}{2\sin\!\left(\pi k/N\right)} = 0,$$

and furthermore

$$\sum_{n=0}^{N-1}\cos^2\left(n\cdot\frac{2\pi}{N}\right) = \sum_{n=0}^{N-1}\frac{1}{2} + \frac{1}{2}\cos\!\left(2n\cdot\frac{2\pi}{N}\right) = \frac{N}{2},$$
$$\sum_{n=0}^{N-1}\sin^2\left(n\cdot\frac{2\pi}{N}\right) = \sum_{n=0}^{N-1}\frac{1}{2} - \frac{1}{2}\cos\!\left(2n\cdot\frac{2\pi}{N}\right) = \frac{N}{2},$$
$$\sum_{n=0}^{N-1}\cos\!\left(n\cdot\frac{2\pi}{N}\right)\sin\!\left(n\cdot\frac{2\pi}{N}\right) = \sum_{n=0}^{N-1}\frac{1}{2}\sin\!\left(2n\cdot\frac{2\pi}{N}\right) = 0.$$

Consequently $\mathcal{C}\mathcal{C}^\top\mathcal{D} = \mathbf{I}_3$. The proposed pseudo-inverse satisfies the criteria of Definition 1:

1. $\mathcal{C}(\mathcal{C}^\top\mathcal{D})\mathcal{C} = \mathbf{I}\mathcal{C} = \mathcal{C}$,
2. $(\mathcal{C}^\top\mathcal{D})\mathcal{C}(\mathcal{C}^\top\mathcal{D}) = \mathcal{C}^\top\mathcal{D}\mathbf{I} = \mathcal{C}^\top\mathcal{D}$,
3. $(\mathcal{C}(\mathcal{C}^\top\mathcal{D}))^{\mathrm{H}} = \mathbf{I}^{\mathrm{H}} = \mathbf{I} = \mathcal{C}(\mathcal{C}^\top\mathcal{D})$,
4. $((\mathcal{C}^\top\mathcal{D})\mathcal{C})^{\mathrm{H}} = \mathcal{C}^{\mathrm{H}}\mathcal{D}^{\mathrm{H}}\mathcal{C} = (\mathcal{C}^\top\mathcal{D})\mathcal{C}$.

$$\therefore \mathcal{C}^\dagger = \mathcal{C}^\top\mathcal{D} \qquad \blacksquare$$

Lemma 16 (Derivation of Eq. (2-84) on page 30)
Equation (2-84) *on page 30 is the least squares estimate of Eq.* (2-83).

Proof Equation (2-84) reads as

$$\begin{aligned}
\boldsymbol{y}'_{\text{sensors}} \coloneqq \mathcal{R}^{\text{s}}_{\text{s}\forall}\boldsymbol{y}_{\text{sensors}} &= \left(\boldsymbol{\Lambda}^\top \otimes \mathbf{I}_3\right)\boldsymbol{y}_{\text{s}} \\
&= \left(\mathcal{C}^\top \otimes \mathbf{I}_3\right)\underbrace{\left(\operatorname{diag}(-r_{\text{s}}, -r_{\text{s}}, 1) \otimes \mathbf{I}_3\right)\boldsymbol{y}_{\text{s}}}_{=:\boldsymbol{y}'_{\text{s}}} \quad \text{(by Lemma 3), (A-4)}
\end{aligned}$$

$$\text{where} \quad \mathcal{C} = \begin{bmatrix} 0 & c_2 & c_3 & \cdots & c_7 \\ 0 & s_2 & s_3 & \cdots & s_7 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}.$$

$\left(\mathcal{C}^\top \otimes \mathbf{I}_3\right) \in \mathbb{R}^{21\times 9}$ and $\operatorname{rank}\left(\mathcal{C}^\top \otimes \mathbf{I}_3\right) = \operatorname{rank}(\mathcal{C}) \cdot \operatorname{rank}(\mathbf{I}_3) = 9$ by Lemma 7, so $\left(\mathcal{C}^\top \otimes \mathbf{I}_3\right)$ is a thin full column rank matrix and hence Eq. (A-4) is overdetermined.
Consider the following least squares problem

$$\min_{\boldsymbol{y}'_{\text{s}}} \boldsymbol{\varepsilon}^\top\boldsymbol{\varepsilon} \qquad \text{s.t.} \qquad \boldsymbol{y}'_{\text{sensors}} = \left(\mathcal{C}^\top \otimes \mathbf{I}_3\right)\boldsymbol{y}'_{\text{s}} + \boldsymbol{\varepsilon}$$

and its well-known solution

$$\hat{\boldsymbol{y}}'_{\text{s}} = \left(\mathcal{C}^\top \otimes \mathbf{I}_3\right)^\dagger \boldsymbol{y}'_{\text{sensors}}.$$

This pseudo-inverse can be rewritten as

$$\begin{aligned}
\left(\mathcal{C}^\top \otimes \mathbf{I}_3\right)^\dagger &= \left(\mathcal{C}^{\dagger\top} \otimes \mathbf{I}_3\right) && \text{(by Lemma 8)} \\
&= \left(\left(\mathcal{C}^\top \cdot \operatorname{diag}\left(\tfrac{2}{6}, \tfrac{2}{6}, \tfrac{1}{7}\right)\right)^\top \otimes \mathbf{I}_3\right) && \text{(by Lemma 15)} \\
&= \left(\left(\operatorname{diag}\left(\tfrac{2}{6}, \tfrac{2}{6}, \tfrac{1}{7}\right) \cdot \mathcal{C}\right) \otimes \mathbf{I}_3\right).
\end{aligned}$$

Because $\mathrm{diag}(-r_\mathrm{s}, -r_\mathrm{s}, 1)$ is invertible for $r_\mathrm{s} \neq 0$, by Lemma 5

$$\begin{aligned}
\hat{\boldsymbol{y}}_\mathrm{s} &= \Big(\mathrm{diag}(-r_\mathrm{s}, -r_\mathrm{s}, 1) \otimes \mathbf{I}_3 \Big)^{-1} \hat{\boldsymbol{y}}'_\mathrm{s} \\
&= \Big(\mathrm{diag}(-r_\mathrm{s}, -r_\mathrm{s}, 1)^{-1} \otimes \mathbf{I}_3 \Big) \left(\left(\mathrm{diag}\Big(\tfrac{2}{6}, \tfrac{2}{6}, \tfrac{1}{7}\Big) \cdot \boldsymbol{\mathcal{C}} \right) \otimes \mathbf{I}_3 \right) \boldsymbol{y}'_\mathrm{sensors} \\
&= \left(\left(\mathrm{diag}\Big(\frac{-2}{6r_\mathrm{s}}, \frac{-2}{6r_\mathrm{s}}, \frac{1}{7}\Big) \cdot \boldsymbol{\mathcal{C}} \right) \otimes \mathbf{I}_3 \right) \boldsymbol{\mathcal{R}}^\mathrm{s}_{\mathrm{s}\forall} \boldsymbol{y}_\mathrm{sensors} \qquad \text{(by Lemma 3).}
\end{aligned}$$

■

Lemma 17 (Coordinate transformation of an acceleration matrix)
Consider three frames: 1, 2 and 3, with a fixed position with respect to each other—i.e. they are attached to the same rigid body—and an acceleration matrix $\tilde{\boldsymbol{\mathcal{A}}}_1^{2,0}$ that expresses the acceleration of frame 1 with respect to (inertial) frame 0 in frame 2. Its coordinate transformation to frame 3 is given in MATLAB *notation by*

$$\mathrm{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_1^{3,0}(1:3,:)\Big) = \Big(\boldsymbol{\mathcal{H}}_3^2 \otimes \boldsymbol{\mathcal{R}}_3^2\Big)^\top \cdot \mathrm{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_1^{2,0}(1:3,:)\Big) \,,$$

where the matrix $\boldsymbol{\mathcal{H}}_3^2 = \begin{bmatrix} \boldsymbol{\mathcal{R}}_3^2 & \boldsymbol{p}_3^2 \\ \mathbf{0} & 1 \end{bmatrix}$ *is the homogeneous transformation matrix from frame 3 to frame 2.*

PROOF Frames 1, 2 and 3 are fixed with respect to each other, so $\ddot{\boldsymbol{\mathcal{H}}}_2^1 = 0$, $\ddot{\boldsymbol{\mathcal{H}}}_3^2 = 0$. Then using the definition of the acceleration matrix

$$\begin{aligned}
\tilde{\boldsymbol{\mathcal{A}}}_1^{1,0} &\coloneqq \boldsymbol{\mathcal{H}}_0^1 \ddot{\boldsymbol{\mathcal{H}}}_1^0 \,, \\
\tilde{\boldsymbol{\mathcal{A}}}_1^{2,0} \coloneqq \boldsymbol{\mathcal{H}}_1^2 \cdot \tilde{\boldsymbol{\mathcal{A}}}_1^{1,0} \cdot \boldsymbol{\mathcal{H}}_2^1 &= \boldsymbol{\mathcal{H}}_1^2 \cdot \boldsymbol{\mathcal{H}}_0^1 \ddot{\boldsymbol{\mathcal{H}}}_1^0 \cdot \boldsymbol{\mathcal{H}}_2^1 = \boldsymbol{\mathcal{H}}_1^2 \boldsymbol{\mathcal{H}}_0^1 \cdot \tfrac{\mathrm{d}^2}{\mathrm{d}t^2} \Big(\boldsymbol{\mathcal{H}}_1^0 \boldsymbol{\mathcal{H}}_2^1 \Big) = \boldsymbol{\mathcal{H}}_0^2 \ddot{\boldsymbol{\mathcal{H}}}_2^0 \,, \\
\tilde{\boldsymbol{\mathcal{A}}}_1^{3,0} = \boldsymbol{\mathcal{H}}_0^3 \ddot{\boldsymbol{\mathcal{H}}}_3^0 &= \boldsymbol{\mathcal{H}}_2^3 \boldsymbol{\mathcal{H}}_0^2 \cdot \tfrac{\mathrm{d}^2}{\mathrm{d}t^2} \Big(\boldsymbol{\mathcal{H}}_2^0 \boldsymbol{\mathcal{H}}_3^2 \Big) = \boldsymbol{\mathcal{H}}_2^3 \cdot \boldsymbol{\mathcal{H}}_0^2 \ddot{\boldsymbol{\mathcal{H}}}_2^0 \cdot \boldsymbol{\mathcal{H}}_3^2 = \boldsymbol{\mathcal{H}}_2^3 \cdot \tilde{\boldsymbol{\mathcal{A}}}_1^{2,0} \cdot \boldsymbol{\mathcal{H}}_3^2 \,, \\
&\Rightarrow \tilde{\boldsymbol{\mathcal{A}}}_1^{3,0}(1:3,:) = \boldsymbol{\mathcal{R}}_2^3 \cdot \tilde{\boldsymbol{\mathcal{A}}}_1^{2,0}(1:3,:) \cdot \boldsymbol{\mathcal{H}}_3^2 \,, \\
\Rightarrow \mathrm{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_1^{3,0}(1:3,:)\Big) &= \Big(\boldsymbol{\mathcal{H}}_3^{2\top} \otimes \boldsymbol{\mathcal{R}}_2^3\Big) \cdot \mathrm{vec}\Big(\tilde{\boldsymbol{\mathcal{A}}}_1^{2,0}(1:3,:)\Big) \qquad \text{(by Lemma 6).}
\end{aligned}$$

Since $\boldsymbol{\mathcal{R}}_3^2 = \boldsymbol{\mathcal{R}}_2^{3\top}$, applying Lemma 4 completes the proof. ■

Appendix B

Matlab files

The scripts and functions in this appendix are designed for Matlab R2016a. Besides toolboxes distributed with Matlab, some depend on the *LTI System Identification Toolbox* (version 2.4), which is freely available from the Delft University of Technology (TU Delft). The m-files are included as embedded files in the digital version of this thesis. Even though this feature is contained in the PDF standard since version 1.3, at the moment of writing only Adobe viewers seem to have implemented it.

B-1 Supporting Matlab functions

B-1-1 Identification

Listing B.1: blockhankel.m

```
function [Y,N] = blockhankel( y, s )

% Construct a block hankel matrix
%
% [Y,N] = blockhankel( y, s )
%
% Inputs:   y   Data with samples in rows, channels in columns:
%                   |" y_x[1]   , y_y[1]   , Y_z[1]   , ... "|
%               y = |    :          :          :             |
%                   |_ y_x[end] , y_y[end] , Y_z[end] , ... _|
%           s   Number of block rows in the block hankel matrix
%
% Outputs:  Y   Block Hankel matrix:
%                   |" y_x[1] , y_x[2]   , ...  , y_x[N]   "|
```

```
15 %                   |  y_y[1] , y_y[2]   , ...  , y_y[N]    |
16 %                   |  y_z[1] , y_z[2]   , ...  , y_z[N]    |
17 %                   |    :    ,   :                 :       |
18 %                   |  y_x[2] ,   :                 :       |
19 %                   |  y_y[2] ,   :                 :       |
20 %               Y = |  y_z[2] ,   :                 :       |
21 %                   |    :        :                 :       |
22 %                   |    :        :                 :       |
23 %                   |  y_x[s] , y_x[s+1] , ...  , y_x[end]  |
24 %                   |  y_y[s] , y_x[s+1] , ...  , y_y[end]  |
25 %                   |_ y_z[s] , y_x[s+1] , ...  , y_z[end] _|
26 %           N   Number of columns in Y
27 %
28 % See Eq. (2-7) on page 12.
29 %
30 % For PO-MOESP, use
31 %   [Y,N] = blockhankel( y, 2*s )
32 %   Y_0sN = Y( 1:s*size(y,2) , : );
33 %   Y_ssN = Y( s*size(y,2)+1:end , : );
34
35 if size(y,1)==1
36     y = y.';
37 end;
38
39 l = size(y,2);      % Number of output channels
40 N = size(y,1)+1-s;  % Number of columns of block hankel matrix
41 Y = nan(s*l,N);
42 if ~isnumeric(y)
43     Y = sym(Y);
44 end
45 if N<1;
46     error('s is larger than the number of samples.');
47 end;
48
49 if N<s;
50     warning('N seems a bit small');
51 end;
52
53 for j = 1:l
54     Y(j:l:s*l,:) = hankel( y(1:s,j), y(s:end,j) );
55 end;
```

Listing B.2: block_hankel_permutation.m

```
function P = block_hankel_permutation( s, m, l0, lj )

%   P = block_hankel_permutation( s, m, l0, lj )
%
% Calculates a permutation matrix P such that
% P * blockhankel( [u,y(0),y(j)], s ) = [ Uf; Up; Yp(0); Yp(j); Yf(0); Yp(j) ];
%
% Inputs:  s    Parameter of the subspace identification algorithm
%          m    Number of system inputs
%          l0   Number of reference outputs
%          lj   Number of roving outputs
%
% Outputs: P    The permutation matrix described above

l = l0 + lj;
P = [ [ sparse(s*m,s*m), speye(s*m); speye(s*m), sparse(s*m,s*m) ] *...
                           kron( speye(2*s), [ speye(m), sparse(m,l) ] )    ;
      kron( speye(2), [ kron( speye(s), [ speye(l0), sparse(l0,lj) ] ) ;
                        kron( speye(s), [ sparse(lj,l0), speye(lj) ] ) ] ) *...
                           kron( speye(2*s), [ sparse(l,m), speye(l) ] )    ];
```

Listing B.3: block_vectorise.m

```
function W = block_vectorise( W, nblock, mblock )

% Reshape a block partitioned matrix into a block vector
%
% Wout = block_vectorise( Win, nblock, mblock )
%
% Win = [ W_11, ..., W_1M ;
%           :          :
%         W_N1, ..., W_NM ]
%
% [nblock, mblock] = size(W_ij) for all i,j
%
% Wout = [ W_11 ;
%            :
%          W_1M ;
%            :
%          W_N1 ;
%            :
%          W_NM ];
%
% size(Wout) = [numel(Win)/mblock , mblock]
%
% See Eqs. (2-60) and (2-64) on pages 24 and 25.

```

```
% number of elements: M*N*mblock*nblock
[nmx,mmx] = size(W);
N = nmx/nblock;
M = mmx/mblock;

if mod(N,1) % true if N is a non-integer value
    error('Full number of vertical blocks required.')
end;
if mod(M,1) % true if N is a non-integer value
    error('Full number of horizontal blocks required.')
end;

W = reshape( W, nblock*N, mblock, M );
W = permute( W, [2,1,3] );
W = reshape( W, nblock*mblock, N, M );
W = permute( W, [1,3,2] );
W = reshape( W, mblock, M*N*nblock );
W = permute( W, [2,1,3] );
```

Listing B.4: circulant_multiply.m

```
function [c,r] = circulant_multiply( ya, yb, s )

%   [c,r] = circulant_multiply( ya, yb, s )
%
% Calculates the product of a circulant Toeplitz matrix with zero sub-diagonal
% entries and the transpose of an equal sized circulant matrix with zero
% sub-diagonal entries.
%
% Inputs: ya, yb, s  define the operand matrices as follows:
%                    A=toeplitz( [ya(1);zeros(s-1,1)] , [ya(:);zeros(s-1,1)] )
%                    B=toeplitz( [yb(1);zeros(s-1,1)] , [yb(:);zeros(s-1,1)] )
% Outputs c, r       define the result matrix M = A * B.' as follows:
%                    M=toeplitz(c,r)
%
% It can be shown that this also gives the result for A * B.' where
%                    A=hankel( [zeros(s-1,1);ya(end)] , ya(end:-1:1) ), and
%                    B=hankel( [zeros(s-1,1);yb(end)] , yb(end:-1:1) )
%
% See Lemma 10 on page 57.

[xc,lag] = xcorr(ya,yb,s-1);
c        = xc(lag<=0);
c        = c(end:-1:1);
r        = xc(lag>=0);
```

Listing B.5: dinit_batch.m

```
function x0 = dinit_batch(sys, u, y, l0)

%   x0 = dinit_batch(sys, u, y, l0)
%
% Estimate initial state for a batch of roving sensor experiments.
%
% Inputs    sys   (Identified) LTI system. First l0 outputs represent the
%                 reference outputs. The remaining outputs represent the
%                 roving outputs, grouped by experiment.
%           u     Input sequences, one row per sample, one column per channel.
%           y     Output sequences, one row per sample, one column per channel.
%           l0    Number of reference outputs
% Outputs   x0    Cell array of initial states, one cell per experiment.
%
% Depends on LTI toolbox function "dinit"

J  = length(y);
lv = size(sys,1);
lj = (lv-l0)/J;
x0 = cell(J,1);

for j = 1:J
    outrangej = [ 1 : l0 , l0+(j-1)*lj+1 : l0+j*lj ];
    x0{j}  = dinit( sys(outrangej,:).a,...
                    sys(outrangej,:).b,...
                    sys(outrangej,:).c,...
                    sys(outrangej,:).d, u, y{j} );
end;
```

Listing B.6: filter_batch.m

```
function [uw,yw] = filter_batch(u,y,filt)

% function [uw,yw] = filter_batch(u,y,filt)
%
% Filters data from a batch of roving sensor experiments.
%
% Inputs:   u      Input sequence, one row per sample, one column per channel.
%           y      Cell array of output sequences, one cell per experiment,
%                  within each cell an array with one row per sample, one
%                  column per channel.
%           filt   Discrete time weighting filter to apply.
% Outputs:  uw     Filtered input sequence, same format as u.
%           yw     Filtered output sequences, same format as y.

filt = ss(filt);
m    = size(u,2);
```

```
J    = length(y);
uw   = lsim(ss_kron(filt,m),u);
yw   = cell(size(y));
for j = 1 : J
    l     = size(y{j},2);
    yw{j} = lsim(ss_kron(filt,l),y{j});
end;
end;

function sys_kron = ss_kron(sys,k)
% Block diagonal repetition of a state space system:
% In pseudo code: sys_kron = kron ( eye(k), sys )
    sys_kron = ss( kron(eye(k),sys.a),...
                   kron(eye(k),sys.b),...
                   kron(eye(k),sys.c),...
                   kron(eye(k),sys.d), sys.ts);
end;
```

Listing B.7: global_obsv_permutation.m

```
function [P_idx,P] = global_obsv_permutation( s, l0, lj, J)

% [P_idx,P] = global_obsv_permutation( s, l0, lj, J)
%
% Form the permutation matrix that permutes the global extended
% observability matrix to an ordinary extended observability matrix.
% See Eq. (2-56) on page 22.
%
% Inputs:  s      Number of blockrows in the extended observability
%                 matrices
%          l0     Number of rows in C0: number of reference outputs
%          lj     Number of rows in Cj for j in {1,2,...,J}: number of
%                 roving sensor outputs
%          J      Number of experiments
%
% Outputs: P      Permutation matrix
%          P_idx  Permutation indices
%
%   Given:
% O_s_global = [C0; C0*A; ... C0*A^(s-1); C1; C1*A; ...; ... CJ*A^(s-1)];
%
%   Form:
% O_s_ord = [ C0; C1; ... CJ; C0*A; C1*A; ...; ... CJ*A^(s-1) ];
%
%   Do:
% O_s_ord = P * O_s_global;   -or-   O_s_ord = O_s_global( P_idx, :);

[~,Pstar_Js] = perfect_shuffle(J,s);

```

```
30 P = [ kron( speye(s), [ speye(l0) ; spalloc( J*lj, l0, 0) ])     ,...
31       kron( speye(s), [ spalloc( l0, J*lj, 0) ; speye(J*lj) ]) *...
32         kron( Pstar_Js, speye(lj) )                              ];
33
34 [P_idx,~] = find(P.');
```

Listing B.8: hankel_gram.m

```
1  function G = hankel_gram( y, s, varargin )
2
3  %   G = hankel_gram( y, s (, m) )
4  %
5  % Calculates Y_1,s,N * Y_1,s,N.' without filling the Hankel matrix
6  % itself, to save memory and computation time for large N.
7  %
8  % Inputs:  y    Data sequence, one row per sample, one column per channel.
9  %               Include input signals as [u,y]
10 %          s    Parameter of the subspace identification algorithm
11 %          m    Optional: number of inputs. If provided, the output values
12 %               corresponding to multiplications between Hankel matrices
13 %               that are both constructed from one of the first m signals,
14 %               are not calculated, NaNs are returned instead.
15 %               Default: m=0 (all values are calculated);
16 %
17 % Outputs: G    The Hankel Gram matrix Y_1,s,N * Y_1,s,N.'. To interpret
18 %               the result for a matrix with Hankel blocks, like
19 %               [Uf, Up, Yp, Yf], use P * G * P.', with P a permutation matrix,
20 %               calculated using block_hankel_permutation.m
21 %
22 % See Lemma 11 on page 58.
23
24 if nargin>=3
25     m = varargin{1};
26 else
27     m = 0;
28 end
29
30 l = size(y,2);
31 G = nan(s*l);
32 N = size(y,1)-s+1;
33
34 % Calculate (sub-)diagonal elements. See Lemma 11 on page 58.
35 for na = m+1:l
36     for nb = 1:na
37         [c,r]    = circulant_multiply(y(N:-1:s,na),y(N:-1:s,nb),s);
38         c(2:end) = c(2:end) + ...
39                         toeplitz(y(s:2*s-2,na),[y(s,na);zeros(s-2,1)]) * ...
40                             y(s-1:-1:1,nb);
41         r(2:end) = r(2:end) + ...
```

```
42                          toeplitz(y(N+1:end,nb),[y(N+1,nb);zeros(s-2,1)]) * ...
43                              y(N:-1:N-s+2,na);
44          G(na:l:end,nb:l:end)   = toeplitz(c,r);
45          G(na:l:end-l,nb:l:end) = G(na:l:end-l,nb:l:end) + ...
46                                   toeplitz( [ y(s-1,na); zeros(s-2,1) ],...
47                                             y(s-1:-1:1,na)             ) *...
48                                   toeplitz( y(s-1:-1:1,nb), y(s-1:2*s-2,nb) );
49          G(na+l:l:end,nb:l:end) = G(na+l:l:end,nb:l:end) + ...
50                                   toeplitz( y(N+1:end,na),...
51                                             [ y(N+1,na); zeros(s-2,1) ]) *...
52                                   toeplitz( y(N:-1:N-s+2,nb), y(N:end,nb) );
53      end
54  end
55
56  % Fill required superdiagonal elements.
57  for nb = m+2:l
58      for na = m+1:nb-1
59          G(na:l:end,nb:l:end)   = G(nb:l:end,na:l:end).';
60      end
61  end
```

Listing B.9: local2global.m

```
1   function W = local2global(r,k)
2
3   %   W = local2global(r,k)
4   %
5   % Calculate the matrix for the least squares estimate given acceleration
6   % data from a certain strapdown accelerometer grid to a single rigid body.
7   % The layout of the sensor grid is as follows: Sensor triad 1 is in the centre,
8   % its coordinate frame will be the rigid body coordinate frame. All sensors
9   % are in the plane z=0 and sensor 2:k (the rim sensors) are at a distance r
10  % from the centre sensor at equal angles from eachother. All z-axes are
11  % parallel and the rim sensor x-axes point towards the centre sensor.
12  %
13  % Inputs:  r  Distance between the rim sensors and the centre sensor
14  %          k  Number of sensors
15  %
16  % Outputs: W  Matrix such that W*y_sensors is the least squares estimate of
17  %             the vector of measurable quantities of the rigid body
18  %             acceleration y_s
19  %
20  % See Eq. (2-84) on page 30.
21
22  if isempty(ver('symbolic'))       % Check whether symbolic toolbox is available
23      angle_j = [ 0, 0, (1:k-2)*2*pi/(k-1) ];
24  else
25      angle_j = [ 0, 0, (1:k-2)*2*sym(pi)/(k-1) ]; % sym prevents roundoff errors
26  end
```

```
cj = cos(angle_j);
sj = sin(angle_j);

Rk = cell(1,k);
for j = 1:k
    Rk{j} = [ cj(j), -sj(j), 0;
              sj(j),  cj(j), 0;
              0    ,  0    , 1];
end
R_forall = blkdiag(Rk{:});

C = [ [0 cj(2:k)]; [0 sj(2:k)]; ones(1,k) ];

W = kron( diag( [-2/((k-1)*r), -2/((k-1)*r), 1/k] ) * C , eye(3) ) * R_forall;
W = double(W);
```

Listing B.10: lsim_batch.m

```
function y_est = lsim_batch(sys,u,x0,l0)

% y_est = lsim_batch( sys, u, l0, x0)
%
% Simulate output data for a batch of roving sensor experiments.
%
% Inputs    sys    (Identified) LTI system. First l0 outputs represent the
%                  reference outputs. The remaining outputs represent the
%                  roving outputs, grouped by experiment.
%           u      Input sequences, one row per sample, one column per channel.
%           x0     Cell array of initial states, one cell per experiment.
%           l0     Number of reference outputs
% Outputs   y_est  Cell array of simulated output sequences, one cell per
%                  experiment, therein an array with one row per sample,
%                  one column for each channel for that experiment: the
%                  reference outputs and the roving output for that experiment.

if iscell(x0)
    J  = length(x0);
else
    J  = 1;
    x0 = {x0};
end;
lv    = size(sys,1);
lj    = (lv-l0)/J;
y_est = cell(J,1);

for j = 1:J
    outrangej = [ 1 : l0 , l0+(j-1)*lj+1 : l0+j*lj ];
    y_est{j}  = lsim( sys(outrangej,:), u, [], x0{j} );
end;
```

Listing B.11: obsv_ext.m

```
function Os = obsv_ext(A,C,s)

% Calculate the extended observability matrix
%
% Os = obsv_ext(A,C,s)
%
% Os = [C; CA; ...; CA^(s-1)];
%
% Also works for s<n, with a warning
%
% Algorithm by LRJ Haverkamp, 'State space identification' PhD-thesis (2001)

n  = size(A,1);
l  = size(C,1);
if s<n
    warning('obsv_ext:s_smaller_than_n','s<n');
end;

Os = nan(s*l,n);
Os(1:l,:) = C;
for i = 1:floor(log(s)/log(2)),
    Os(l*2^(i-1)+1:l*2^i,:) = Os(1:2^(i-1)*l,:)*A;
    A = A^2;                                          % Be careful: redefining A
end;
Os(l*2^i+1:end,:) = Os(1:end-l*2^i,:)*A;
```

Listing B.12: perfect_shuffle.m

```
function [P_idx,P] = perfect_shuffle(p,r)

% [P_idx,P] = perfect_shuffle(p,r)
%
% Form perfect shuffle matrix (Golub & van Loan "Matrix Computations" [5]
% and Eq. (2-54) on page 22 ) and corresponding indexing vector, such that
%
% P * A = A( P_idx(:), : )
%
% example: p=3, r=4, => P_idx    = [ 1, 2,  3,  4  ;
%                                    5, 6,  7,  8  ;
%                                    9, 10, 11, 12 ]
%                    => P_idx(:) = [ 1, 5, 9, 2, 6, 10, ... 8, 12]
%
% Useful property:
%
% inv(P) = P.';
%
% Selecting interleaved subblocks:
```

```
%
%   Given:
% O = [C1; C1*A; ...; C1*A^(s-1); C2; C2*A; ...; ...; CJ A^(s-1)]
% l = size( Cj, 1 ) for all j
%
%   Form:
% [C1; C2; ...; CJ]
%
%   Do:
% P_idx = perfect_shuffle( J , s*l );
% O( P_idx(:,1:l).', : ) = [C1; C2; ...; CJ]

P_idx = reshape(1:p*r,r,p).';

if nargout>1
    P = speye(p*r);
    P = P(P_idx(:),:);
end;
```

Listing B.13: vaf_batch.m

```
function p = vaf_batch(y,y_est)

%   p = vaf_batch( y, y_est );
%
% Evaluate variance accounted for a batch of sensor roving experiments
%
% Inputs:  y           Cell array with measured output sequences, one cell per
%                      experiment, and therein a matrix with one column per
%                      input channel, one row per sample
%          y_est       Cell array with estimated output sequences, one cell per
%                      experiment, and therein a matrix with one column per
%                      input channel, one row per sample
% Outputs: p           Variance accounted for, one column per experiment,
%                      one row per output.
%
% Depends on LTI System Identification Toolbox function "vaf".

J = length(y);
l = size(y{1},2);
p = nan(l,J);
for j = 1:J
    p(:,j) = vaf( y{j}, y_est{j} );
end;
```

B-1-2 State-space algebra

Listing B.14: sscausal_adleft.m

```
function syso = sscausal_adleft(sys2,sys1)

% syso = sscausal_adleft(sys2,sys1)
%
% Calculates the causal part of a the series connection of a discrete time
% statespace system and the adjoint of another discrete time statespace
% system, according to RF Fraanje "Robust and fast schemes in broadband active
% noise and vibration control", 2004 [4] .
%
% Inputs:   sys1       Discrete-time state-space system.
%           sys2       Discrete-time state-space system.
%
% Outputs:  syso       Discrete-time state-space system:
%                      syso = [ sys2' * sys1 ]_+

X21  = dlyap(sys2.a.', sys1.a, sys2.c.'*sys1.c);
syso = ss( sys1.a, ...
           sys1.b, ...
           sys2.d.'*sys1.c + sys2.b.'*X21*sys1.a, ...
           sys2.d.'*sys1.d + sys2.b.'*X21*sys1.b, ...
           sys1.ts                                    );

syso.inputname   = sys1.inputname;
syso.inputunit   = sys1.inputunit;
syso.inputgroup  = sys1.inputgroup;
end
```

Listing B.15: sscausal_adright.m

```
function syso = sscausal_adright(sys1,sys2)

% syso = sscausal_adleft(sys2,sys1)
%
% Calculates the causal part of a the series connection of the adjoint of a
% discrete time statespace system and another discrete time statespace
% system, according to RF Fraanje "Robust and fast schemes in broadband active
% noise and vibration control", 2004 [4] .
%
% Inputs:   sys1       Discrete-time state-space system.
%           sys2       Discrete-time state-space system.
%
% Outputs:  syso       Discrete-time state-space system:
%                      syso = [ sys1 * sys2' ]_+

X12  = dlyap(sys1.a, sys2.a.', sys1.b*sys2.b.');
syso = ss( sys1.a, ...
           sys1.b*sys2.d.' + sys1.a*X12*sys2.c.', ...
```

```
19            sys1.c, ...
20            sys1.d*sys2.d.' + sys1.c*X12*sys2.c.', ...
21            sys1.ts                                    );
22
23  syso.outputname   = sys1.outputname;
24  syso.outputunit   = sys1.outputunit;
25  syso.outputgroup  = sys1.outputgroup;
26  end
```

Listing B.16: ssIO.m

```
1   function [sys_inner,sys_outer] = ssIO(sys)
2
3   % [sys_inner,sys_outer] = ssIO(sys)
4   %
5   % Calculates the inner-outer factorisation of a discrete time statespace
6   % system.
7   %
8   % Inputs:   sys        Discrete-time state-space system.
9   %
10  % Outputs:  sys_inner  Discrete-time state-space system.
11  %                      sys_inner' * sys_inner = I
12  %           sys_outer  Square discrete-time state-space system.
13  %                      sys_outer has a stable left inverse, i.e. all its
14  %                      poles and zeros are inside the unit disk.
15  %                      sys_inner * sys_outer = sys
16
17  sys_outer = spectralfact(sys,[]);
18  sys_inner = minreal(sys/sys_outer);
19
20  sys_outer.InputName  = sys.InputName;
21  sys_outer.InputUnit  = sys.InputUnit;
22  sys_outer.InputGroup = sys.InputGroup;
```

Listing B.17: ssOI.m

```
1   function [sys_outer,sys_inner] = ssOI(sys)
2
3   % [sys_outer,sys_inner] = ssOI(sys)
4   %
5   % Calculates the coouter-inner factorisation of a discrete time statespace
6   % system.
7   %
8   % Inputs:   sys        Discrete-time state-space system.
9   %
10  % Outputs:  sys_outer  Square discrete-time state-space system.
11  %                      sys_outer has a stable left inverse, i.e. all its
12  %                      poles and zeros are inside the unit disk.
```

```
%           sys_inner  Discrete-time state-space system.
%                      sys_inner * sys_inner' = I
%                      sys_outer * sys_inner  = sys

sys_outer = spectralfact(sys.',[]).';
sys_inner = minreal(sys_outer\sys);

sys_outer.OutputName  = sys.OutputName;
sys_outer.OutputUnit  = sys.OutputUnit;
sys_outer.OutputGroup = sys.OutputGroup;
```

Listing B.18: sspinv.m

```
function pinv_sys = sspinv(sys)

% pinv_sys = sspinv(sys)
%
% Calculates the pseudo inverse of a discrete time statespace system
% according to RF Fraanje "Robust and fast schemes in broadband active noise
% and vibration control", 2004 [4] .
%
% Inputs:  sys       Discrete time statespace system, must have a non-singular
%                    D-matrix.
%
% Outputs: pinv_sys  Discrete time pseudo-inverse statespace system.
%                    If D is full column rank pinv_sys is a left-inverse of
%                    sys, i.e. pinv_sys * sys = I;
%                    If D is full row rank pinv_sys is a right-inverse of
%                    sys, i.e. sys * pinv_sys = I;

[UD,SD,VD] = svd(sys.d);
sD = diag(SD);
if sD(1)/sD(end)>1e4;
    num2str(sD)
    warning(['D is badly conditioned, singular values: ' num2str(sD.')]);
end;
Do = VD*pinv(SD)*UD.';              % = pinv(sys.d)
Ao = sys.a-sys.b*Do*sys.c;
Bo = sys.b*Do;
Co = -Do*sys.c;
pinv_sys = ss(Ao,Bo,Co,Do,sys.ts);

pinv_sys.Name        = ['pinv (' sys.Name ')'];
pinv_sys.InputName   = sys.OutputName;
pinv_sys.InputUnit   = sys.OutputUnit;
pinv_sys.InputGroup  = sys.OutputGroup;
pinv_sys.OutputName  = sys.InputName;
pinv_sys.OutputUnit  = sys.InputUnit;
pinv_sys.OutputGroup = sys.InputGroup;
```

B-2 System identification

B-2-1 Noise generating model

Listing B.19: cold_plate_data_identification.m

```
% Cold plate identification
% See Section 2-1 on page 3.

clear variables;
close all;

%% Settings
b.verbose              = true;              % Output plots
b.weighting            = true;              % Apply weighting filter
b.reduce_sampling_rate = true;              % Resample
b.use_saved_filters    = true;              % Don't recalculate filters
b.overwrite_files      = true;              % Overwrite previous files


%% Load data
sig     = load('Delft/Accelerodata_14_00_53_280.mat');
                                    % contains *.t, *.rawdata, *.Temperature
sig.ts  = round(median( diff(sig.t) ),6);   % Sample time
sig.fs  = 1/sig.ts;                         % Sample frequency
sig.num = size(sig.rawdata,2);              % NUmber of samples

%%      *** NOISE SHAPING MODEL IDENTIFICATION ***
%
% Structure:
%
%                 Gn
%   |"""""""""""""""""""""""""""""""|
%    s      s_corr               a         a_W
%      +---+      +------------+      +---+
% ---->| M |----->| Gn_corr_s |----->| W |----->
%      +---+      +------------+      +---+
%             |_____________________________|
%                           G_id
% Signals: (all have dimension 3)
%   a               Measured acceleration data
%   a_weighted      Weighted acceleration data
%   s_corr          White noise sequence
%   s               Independent white noise sequence
%
% Subsystems: (all have size 3x3)
%   W               Manually tuned output weighting filter
%   G_id            PO-MOESP identified model
```

```
43 %   Gn_corr_s       = W \ G_id
44 %   M               Obtained from PCA on e_corr
45 %   Gn              Desired noise generating model
46 %
47
48 %% Pre-process measurement data
49 % -Convert sensor data to m/s^2
50 g_vms2 = 9.80665 ./ [0.985, 0.935, 0.973]; %[ (m/s^2) /V ]
51                         % Sensor SN2155905 @159Hz  % (1 V -> 0.985*g, etc.)
52 %g_vms2 = 9.807  ./ [1.015, 1.059, 1.005]; %[ (m/s^2) /V ]
53                         % Sensor SN4973067 @159Hz  % (1 V -> 1.015*g, etc.)
54 %g_vms2 = 9.807  ./ [1.020, 1.065, 1.009]; %[ (m/s^2) /V ]
55                         % Sensor SN4973067 @100Hz  % (1 V -> 1.020*g, etc.)
56 sig.a_raw = sig.rawdata .* ( ones(size(sig.rawdata,1),1) * g_vms2 );
57 sig.t_raw = sig.t;
58
59 clear g_vms2
60
61 % -Detrend
62 sig.a = detrend(sig.a_raw);
63
64 % -Reduce sampling rate
65 if b.reduce_sampling_rate
66     fs_new = 5e3;                          % Desired sampling frequency [Hz]
67     sig.a  = resample(sig.a,fs_new,round(sig.fs));
68     sig.t  = resample(sig.t,fs_new,round(sig.fs));
69     sig.ts = sig.ts * sig.fs/fs_new;
70     sig.fs = fs_new;
71     clear fs_new;
72 end;
73
74 % -- Frequency weighting
75 if b.weighting
76     wc_pole    = 2*pi*100;                 % pole cut-off frequency [rad/s]
77     Ast        = 25;                       % stopband attenuation [dB]
78     wc_zero    = wc_pole*10^(Ast/20);
79     Hweight.ct = tf( [1, wc_zero] * wc_pole ,...
80                      [1, wc_pole] * wc_zero );
81     Hweight.dt = ss(c2d( Hweight.ct, sig.ts, 'TimeUnits','seconds'));
82     W          = [ Hweight.dt, 0,          0          ;
83                    0,          Hweight.dt, 0          ;
84                    0,          0,          Hweight.dt ];
85     clear wc_pole Ast wc_zero Hweight
86 else
87     W = ss([],[],[],eye(sig.num),sig.ts);
88 end;
89
90 W.InputName  = {'a_x';'a_y';'a_z'};
91 W.InputUnit  = {'m/s^2';'m/s^2';'m/s^2'};
```

```
92  W.OutputName = {'a_x,weighted';'a_y,weighted';'a_z,weighted'};
93  W.OutputUnit = {'m/s^2';'m/s^2';'m/s^2'};
94  W.Name       = 'Output weighting filter';
95  sig.a_W      = lsim(W,sig.a);
96
97  % Empty input signal
98  sig.uu = zeros(size(sig.a,1),0);
99
100 % Plot
101 if b.verbose
102     % plot time domain data
103     h.time = figure();
104     for j = 1:sig.num
105         subplot(sig.num,1,j);
106         hold on;
107         plot(sig.t_raw,sig.a_raw(:,j));
108         plot(sig.t,sig.a(:,j));
109         plot(sig.t,sig.a_W(:,j));
110         legend('raw','unweighted','weighted');
111         xlabel('time [s]');
112         ylabel('acceleration [m/s^2]');
113         title('Acceleration (xyz)');
114         grid on;
115         hold off;
116     end;
117     clear j;
118
119     % plot frequency domain data
120     h.freq = figure();
121     n_avg  = 2^7;
122     win    = 2^(floor((log(size(sig.a,1))-log(n_avg))/log(2)));
123     for j = 1:sig.num
124         subplot(sig.num,1,j);
125         hold on;
126         pwelch(sig.a(:,j),win, win/2, win, sig.fs);
127         set(gca,'xscale','log');
128         pwelch(sig.a_W(:,j),win, win/2, win, sig.fs);
129         grid on;
130         title('Acceleration spectrum (xyz)');
131         hold off;
132     end;
133     clear j n_avg win;
134
135     % plot bodeplot of weighting filter
136     if b.weighting
137         h.bode = figure();
138         h.bodeplot = bodeplot(W(1,1));
139         setoptions(h.bodeplot,'FreqUnits','Hz');
140         grid on;
```

```
141         title('Weighting filter');
142     end;
143 end;
144 
145 %% PO-MOESP
146 s_range = 75;
147 leg = cell(1,length(s_range));
148 
149 if b.verbose
150     h.singular_values = figure();
151 end;
152 
153 for j = 1:length(s_range)
154     [S,R] = dordpo(sig.uu,sig.a_W,s_range(j));
155     if b.verbose
156         figure(h.singular_values);
157         semilogy(1:s_range(j),S,'+');hold on;
158         leg{j}=['s=' num2str(s_range(j))];
159     end;
160 end;
161 
162 if b.verbose
163     figure(h.singular_values);
164     hold off;
165     title('First s singular values of the rank-deficient R32 matrix')
166     xlabel('Ordinal number of the singular value')
167     ylabel('Magnitude of the singular value')
168     legend(leg{:});
169 end;
170 
171 clear j S leg s_range;
172 %return;
173 n = input('Enter order: '); % Used: n = 7;
174 
175 [Ae,Ce,Ke]      = dmodpo(R,n);
176 G_id            = ss(Ae,Ke,Ce,eye(size(Ce,1)),sig.ts,'TimeUnit','seconds');
177 G_id.InputName  = {'e_corr'};
178 G_id.InputUnit  = {'m/s^2';'m/s^2';'m/s^2'};
179 G_id.OutputName = {'a_xW';'a_yW';'a_zW'};
180 G_id.OutputUnit = {'m/s^2';'m/s^2';'m/s^2'};
181 G_id.Name       = 'Identified system including weighting dynamics';
182 Gn_corr_s       = W \ G_id;
183 Gn_corr_s.Name  = ...
184             'Noise generating model, no input error correlation correction';
185 
186 clear Ae Ce Ke n R
187 
188 %% Reconstruct error signal
189 sig.s_corr = lsim(inv(Gn_corr_s),sig.a);
```

```
190
191 % Plot reconstructed error signal
192 if b.verbose
193     % plot time domain data
194     h.time_error = figure();
195     for j = 1:sig.num
196         subplot(sig.num,1,j);
197         hold on;
198         plot(sig.t,sig.s_corr(:,j));
199         xlabel('time [s]');
200         ylabel('acceleration [m/s^2]');
201         title('Error');
202         grid on;
203         hold off;
204     end;
205     clear j;
206
207     % plot frequency domain data
208     h.freq = figure();
209     n_avg  = 2^7;
210     win    = 2^(floor((log(size(sig.a,1))-log(n_avg))/log(2)));
211     for j = 1:sig.num
212         subplot(sig.num,1,j);
213         pwelch(sig.s_corr(:,j),win, win/2, win, sig.fs);
214         set(gca,'xscale','log');
215         grid on;
216         title('Error spectrum (xyz)');
217         hold off;
218     end;
219     clear j n_avg win;
220
221     % plot correlation between error channels
222     h.corr=figure();
223     d = 500;
224     for i = 1:sig.num
225         for j = 1:sig.num
226             correl = xcorr( sig.s_corr(:,i)/std(sig.s_corr(:,i)),...
227                             sig.s_corr(:,j)/std(sig.s_corr(:,j)),...
228                             d, 'unbiased');
229             subplot( sig.num, sig.num, j+sig.num*(i-1) );
230             plot(-d:d,correl);
231             axis([-d,d,-1,1]);
232         end;
233     end;
234     clear i j d;
235 end;
236
237 %% Decompose error signal in independent noise sequences
238 [M_unsc, s_unsc] = pca(sig.s_corr);
```

```
239 sig.s = s_unsc / diag( std(s_unsc) );
240 M     = M_unsc * diag( std(s_unsc) );
241                     % => sig.s_corr = sig.s * M.' = sig.s_unsc * M_unsc.'
242
243 Gn = Gn_corr_s * M;
244 Gn.InputName = {'e'};
245 Gn.InputUnit = {'m/s^2';'m/s^2';'m/s^2'};
246 Gn.Name = 'Noise generating model';
247
248 clear M_unsc e_unsc
249 % Plot reconstructed error signal
250 if b.verbose
251     % plot time domain data
252     h.time_error_orth = figure();
253     for j = 1:sig.num
254         subplot(sig.num,1,j);
255         hold on;
256         plot(sig.t,sig.s(:,j));
257         xlabel('time [s]');
258         ylabel('acceleration [m/s^2]');
259         title('Error (principle components)');
260         grid on;
261         hold off;
262     end;
263     clear j;
264
265     % plot frequency domain data
266     h.freq = figure();
267     n_avg  = 2^7;
268     win    = 2^(floor((log(size(sig.a,1))-log(n_avg))/log(2)));
269     for j = 1:sig.num
270         subplot(sig.num,1,j);
271         pwelch(sig.s(:,j),win, win/2, win, sig.fs);
272         set(gca,'xscale','log');
273         grid on;
274         title('Error spectrum (principle components)');
275         hold off;
276     end;
277     clear j n_avg win;
278
279     % plot correlation between error channels
280     h.corr=figure();
281     d = 500;
282     for i = 1:sig.num
283         for j = 1:sig.num
284             correl = xcorr( sig.s(:,i)/std(sig.s(:,i)), ...
285                             sig.s(:,j)/std(sig.s(:,j)), ...
286                             d, 'unbiased');
287             subplot( sig.num, sig.num, j+sig.num*(i-1) );
```

```
            plot(-d:d,correl);
            axis([-d,d,-1,1]);
        end;
    end;
    clear i j d;
end;

%% Simulate with other noise realisation
sim.s = wgn(size(sig.a,1),sig.num,0);
sim.a = lsim(Gn,sim.s);

if b.verbose
    % plot frequency domain data
    figure(h.freq);
    clf;
    n_avg  = 2^7;
    win    = 2^(floor((log(size(sig.a,1))-log(n_avg))/log(2)));
    for j = 1:sig.num
        subplot(sig.num,1,j);
        hold on;
        pwelch(sig.a(:,j),win, win/2, win, sig.fs);
        set(gca,'xscale','log');
        pwelch(sim.a(:,j),win, win/2, win, sig.fs);
        grid on;
        title('Acceleration spectrum (xyz)');
        legend('measured','simulated','Location','EastOutside');
        hold off;
    end;
    clear j n_avg win;
end;

%%      *** PERIODIC ENVELOPE DETECTION ***
%
% Detection algorithm:
%
%         |  s
%      +--V--+
%      | /"" |      High-pass filter
%      +--|--+
%         |  s_HP
%    +----+----+
%    | Delay   |
% +--V--+   +--V--+
% | -grd|   | _   | Hilbert transform estimator
% |z    |<--|  |_ |
% +--|--+   +--|--+
%    |         |
%   +V---------V+
%   |Re       Im|
```

```
337 %   +-----|-----+
338 %         |  w (analytic signal)
339 %     +---V---+
340 %     | abs() |
341 %     +---|---+     Low-pass filter
342 %         |            +-----+
343 %         +----------->| ""\ |--> env_LP (reference envelope, non-periodic)
344 %         |            +-----+
345 %         |           +-------+ validation data
346 %         +---------->| Split |---------------+
347 %         |       +-->| data  |               |
348 %         |       |   +---|---+               |
349 %         |       |       | train data        |
350 %    +----V----+  |   +---V---+               |
351 %    |Autocorr.|  |   |  fft  |               |
352 %    +----|----+  |   +---|---+               |
353 %         |       |       |                   |
354 %    +----V----+  |   +---V---+               |
355 %    |  Cycle  |--+   | trun- |<------------+ |
356 %    |detection|      |  cate |             | | |
357 %    +---------+      +---|---+             | | |
358 %                         |                 | | |
359 %           Result  <-----+                 | | |
360 %                         |                 | | |
361 %                     +---V---+         +---|--V--+
362 %                     | ifft  |-------->|Validate|
363 %                     +-------+ env_rec +--------+
364 %                          (periodic envelope)
365 %
366
367 %% Highpass filter to remove drift
368 if b.use_saved_filters
369     load('detect_envelope_filters.mat');
370 else
371     if b.overwrite_files
372         load('detect_envelope_filters.mat');
373     end;
374     N              = 8; % Filter order
375     F3dB           = 30; % -3dB frequency [Hz]
376     filt.HP_design = fdesign.highpass('N,F3dB', N, F3dB, sig.fs);
377     filt.HP        = design(filt.HP_design,'butter');
378     if b.overwrite_files
379         save('detect_envelope_filters.mat','filt');
380     end;
381     clear N Fc;
382 end;
383
384 sig.s_HP = filter(filt.HP,sig.s);
385
```

```
386 if b.verbose
387     h.fvtool_filt_HP = fvtool(filt.HP);
388     h.sig_s          = figure();
389     for j = 1:sig.num
390         subplot(sig.num,1,j);
391         hold on;
392         plot(sig.t,sig.s(:,j),TUDline(1));
393         plot(sig.t,sig.s_HP(:,j),TUDline(2));
394         plot(sig.t([1,end]),[0,0],'k-');
395         legend('e','e_{HP}','Location','EastOutside')
396         xlabel('time [s]');
397         ylabel('e,e_{HP} [AU]');
398         title(['Highpass filtering, signal ' num2str(j)]);
399         grid on;
400         hold off;
401     end;
402     clear j;
403 end;
404
405 %% Estimate analytic signal
406 % Hilbert transform estimation
407
408 if ~b.use_saved_filters
409     tw               = 6; % Double-sided BW of the transition region [Hz]
410     Ap               = 1; % Passband ripple [dB]
411     filt.Hilb_design = fdesign.hilbert('TW,Ap',tw,Ap,sig.fs);
412     filt.Hilb        = design(filt.Hilb_design,'equiripple');
413     if b.overwrite_files
414         save('detect_envelope_filters.mat','filt');
415     end;
416     clear tw Ap;
417 end;
418
419 sig.w_Im = filter(filt.Hilb,sig.s_HP);
420
421 % Correct for group delay of Hilbert transform estimation
422
423 grd      = round( mean( grpdelay( filt.Hilb ) ) );
424 sig.t_w  = sig.t(1 : end-grd, : );
425 sig.w_Im = sig.w_Im( grd+1 : end, : );
426 sig.w_Re = sig.s_HP( 1 : end-grd, : );
427
428 % Compose analytic signal
429 sig.w = complex( sig.w_Re, sig.w_Im );
430
431 if b.verbose
432     h.fvtool_filt_Hilb = fvtool(filt.Hilb,...
433                                 'frequencyrange','[-fs/2, fs/2)',...
434                                 'magnitudedisplay','zero-phase');
```

```
435     h.sig_w            = figure();
436     for j = 1:sig.num
437         subplot(sig.num,1,j);
438         hold on;
439         plot3(sig.t_w,sig.w_Re(:,j),sig.w_Im(:,j),TUDline(1));
440         plot3(sig.t_w([1,end]),[0,0],[0,0],'k-');
441         legend('w','Location','EastOutside')
442         xlabel('time [s]');
443         ylabel('Re(w) [AU]');
444         zlabel('Im(w) [AU]');
445         title(['Estimated analytic signal ' num2str(j)]);
446         grid on;
447         hold off;
448     end;
449     clear j;
450 end;
451
452 clear grd;
453
454 %% Lowpass filter to obtain a smooth envelope
455
456 if b.use_saved_filters
457     Fp             = filt.LP_design.Fpass;
458 else
459     Fp             = 80;  % Passband frequency [Hz]
460     Fst            = 400; % Stopband frequency [Hz]
461     Ap             = 0.1; % Passband ripple [dB]
462     Ast            = 60;  % Stopband attenuation [dB]
463     filt.LP_design = fdesign.lowpass('Fp,Fst,Ap,Ast', Fp, Fst, Ap, Ast, ...
464                                                                 sig.fs);
465     filt.LP        = design(filt.LP_design,'equiripple','minphase',true);
466     if b.overwrite_files
467         save('detect_envelope_filters.mat','filt');
468     end;
469     clear Fst Ap Ast;
470 end;
471
472 sig.env_LP   = filter(filt.LP,abs(sig.w));
473 [grds,wgrds] = grpdelay(filt.LP);
474 grd          = round(mean(grds(wgrds<2*pi*Fp)));
475 sig.env_LP   = sig.env_LP(grd+1:end,:);
476 sig.t_env_LP = sig.t_w(1:end-grd,:);
477
478 if b.verbose
479     fvtool(filt.LP);
480     figure(h.sig_s);
481     clf;
482     for j = 1:sig.num
483         subplot(sig.num,1,j);
```

```
484         hold on;
485         plot(sig.t,sig.s_HP(:,j),TUDline(2));
486         plot(sig.t_env_LP,sig.env_LP(:,j),TUDline(1));
487         plot(sig.t([1,end]),[0,0],'k-');
488         legend('e_{HP}','env','Location','EastOutside')
489         xlabel('time [s]');
490         ylabel('e_{HP},env [AU]');
491         title(['Envelope, signal ' num2str(j)]);
492         grid on;
493         hold off;
494     end;
495     clear j;
496 end;
497
498 clear grd grds wgrds Fp;
499
500 %% Use the autocorrelation of the envelope to find its cycle duration
501
502 cycle_estimate   = 0.7;             % Estimated cycle duration [s]
503 minwidth         = 0.05;            % Minimum width for peak detection [s]
504 maxshift = 1.5 * cycle_estimate;    % Maximum shift for autocorr [s]
505 d = uint32(round(maxshift*sig.fs)); % Maximum shift for autocorr [# samples]
506
507 % Autocorrelation
508 [ sig.w_ac, sig.w_ac_k ] = xcorr( rssq(abs(sig.w),2), d,'unbiased');
509
510 % Smoothen autocorrelation
511 gauss_win = gausswin( 2*d+1, 2*d/(minwidth*sig.fs) );
512 sig.w_ac_smooth = conv( sig.w_ac,gauss_win, 'same' );
513
514 % Peak detection
515 [ sig.w_ac_pks, sig.w_ac_pks_k ] = findpeaks(sig.w_ac_smooth,...
516                             sig.w_ac_k,'MinPeakWidth', minwidth*sig.fs);
517 peakdist  = abs( sig.w_ac_pks_k*sig.ts - cycle_estimate );       % [s]
518 sig.cycle = uint32(sig.w_ac_pks_k( peakdist == min(peakdist) )); % [# sam.]
519
520 if b.verbose
521     h.sig_xc = figure();
522     for j = 1:sig.num
523         hold on;
524         plot( sig.w_ac_k*sig.ts, sig.w_ac_smooth, TUDline(1));
525         plot( sig.w_ac_pks_k*sig.ts, sig.w_ac_pks, '+');
526         plot( sig.w_ac_pks_k( peakdist == min(peakdist) )*sig.ts,...
527                             sig.w_ac_pks( peakdist == min(peakdist) ), 'O');
528         xlabel('lag [s]');
529         ylabel('autocorrelation [AU]');
530         title('Autocorrelation of the envelope magnitude');
531         grid on;
532         hold off;
```

```
533     end;
534     clear j;
535 end;
536 clear maxshift minwidth peakdist cycle_estimate d gauss_win;
537 
538 %% Find truncated order Fourier series for envelopes
539 
540 trainfrac         = 2/3;   % Approximate portion o.t. data used for fitting
541 numfullcycles     = uint32(floor( size(sig.w,1) / sig.cycle ));
542 numtraincycles    = uint32(ceil(trainfrac * numfullcycles));
543 % Indices of validation and train range:
544 valrange          = 1:size(sig.w,1)-numtraincycles*sig.cycle;
545 trainrange        = valrange(end)+1:size(sig.w,1);
546 trainrange_env_LP = trainrange( trainrange<=size(sig.env_LP,1) );
547 % Take fft of un-lowpassfiltered signal:
548 ffttrain          = fft( abs(sig.w( trainrange , : )) );
549 harmonicrange     = 1:200; % Evaluate VAF for these numbers of harmonics
550 % Initialize
551 vafscore.train    = nan(length(harmonicrange),3);
552 vafscore.val      = nan(length(harmonicrange),3);
553 
554 for j = 1:length(harmonicrange)
555     % DC and positive harmonics:
556     n           = uint32(harmonicrange(j));
557     harm_idxs   = 1+(0:n-1)*numtraincycles;
558     fftcoefs    = ffttrain(harm_idxs,:) / double(numtraincycles);
559                               % (shorter time => scale power accordingly)
560     % Truncated order fft reconstruction:
561     fftrec      = [ fftcoefs                                      ;
562                     zeros(sig.cycle+1-2*length(fftcoefs),sig.num) ;
563                     conj(fftcoefs(end:-1:2,:))                    ];
564     sig.env_rec = repmat(ifft(fftrec),numfullcycles+1,1);
565     sig.env_rec = sig.env_rec(end-size(sig.w,1)+1:end,:);
566     % Calculate variance accounted for:
567     vafscore.w_train(j,:)      = vaf( abs(sig.w(trainrange,:))  ,...
568                                       sig.env_rec(trainrange,:) );
569     vafscore.w_val(j,:)        = vaf( abs(sig.w(valrange,:))  ,...
570                                       sig.env_rec(valrange,:) );
571     vafscore.env_LP_train(j,:) = vaf( sig.env_LP(trainrange_env_LP,:)  ,...
572                                       sig.env_rec(trainrange_env_LP,:) );
573     vafscore.env_LP_val(j,:)   = vaf( sig.env_LP(valrange,:)  ,...
574                                       sig.env_rec(valrange,:) );
575 end;
576 
577 [maxvaf,maxvafidx] = max(vafscore.w_val,[],1);
578 
579 clear fftcoefs fftrec harm_idxs j n trainfrac trainrange_env_LP valrange;
580 
581 % reconstruct with optimal harmonics
```

```
582 fftcoefs = cell(1,3);
583 fftlen   = sig.cycle;
584 sim.env  = nan(size(sig.a,1),sig.num);
585 for j = 1:sig.num
586     % DC and positive harmonics:
587     n                = uint32(maxvafidx(j));
588     harm_idxs        = 1+(0:n-1)*numtraincycles;
589     fftcoefs{j}      = ffttrain(harm_idxs,j)/double(numtraincycles);
590     % Truncated order fft reconstruction:
591     fftrec           = [ fftcoefs{j};
592                          zeros(fftlen+1-2*length(fftcoefs{j}),1);
593                          conj(fftcoefs{j}(end:-1:2))];
594     env_rec_temp     = repmat(ifft(fftrec),numfullcycles+1,1);
595     sim.env(:,j)     = env_rec_temp(end-size(sig.a,1)+1:end);
596 end;
597
598 if b.overwrite_files
599     save('fft_coefs_envelopes.mat','fftcoefs','fftlen');
600 end;
601
602 clear env_rec_temp j n harm_idxs fftrec ffttrain ...
603       numfullcycles numtraincycles;
604
605 if b.verbose
606     % Plot VAF-graph with maxima indicated
607     h.vaf=figure();
608         subplot(2,2,1)
609             plot(harmonicrange,vafscore.w_val);
610             hold on;grid on;
611             plot(harmonicrange(maxvafidx),maxvaf,'+')
612             legend('signal 1', 'signal 2', 'signal 3','Location','SE')
613             xlabel('Number of harmonics')
614             ylabel('VAF [%]')
615             title('Variance Accounted For w.r.t validation data w');
616         subplot(2,2,2)
617             plot(harmonicrange,vafscore.w_train);
618             grid on;
619             legend('signal 1', 'signal 2', 'signal 3','Location','SE')
620             xlabel('Number of harmonics')
621             ylabel('VAF [%]')
622             title('Variance Accounted For w.r.t train data w');
623         subplot(2,2,3)
624             plot(harmonicrange,vafscore.env_LP_val);
625             grid on;
626             legend('signal 1', 'signal 2', 'signal 3','Location','SE')
627             xlabel('Number of harmonics')
628             ylabel('VAF [%]')
629             title('Variance Accounted For w.r.t validation data env_{LP}');
630         subplot(2,2,4)
```

```
631             plot(harmonicrange,vafscore.env_LP_train);
632             grid on;
633             legend('signal 1', 'signal 2', 'signal 3','Location','SE')
634             xlabel('Number of harmonics')
635             ylabel('VAF [%]')
636             title('Variance Accounted For w.r.t train data env_{LP}');
637     % Plot e_HP, filtered envelope, fourier envelope
638     figure(h.sig_s);
639     clf;
640     for j = 1:sig.num
641         subplot(sig.num,1,j);
642         hold on;
643         plot(sig.t,sig.s_HP(:,j),TUDline(2));
644         plot(sig.t_env_LP,sig.env_LP(:,j),TUDline(3));
645         plot(sig.t_w,sig.env_rec(:,j),TUDline(1));
646         legend('e_{HP}','env_{LP}','env_{rec}','Location','EastOutside')
647         xlabel('time [s]');
648         ylabel('e_{HP},env,env_{rec} [AU]');
649         title(['Envelope, signal ' num2str(j)]);
650         grid on;
651         hold off;
652     end;
653     clear j;
654 end;
655
656 clear harmonicrange trainrange;
657
658 %% Simulate with envelope
659
660 sim.s_env = sim.s .* sim.env;
661 sim.a_env = lsim(Gn,sim.s_env);
662
663 if b.verbose
664     figure()
665     for j = 1:sig.num
666         subplot(sig.num,2,2*j-1);
667         plot(sig.t,sig.a(:,j));
668         xlabel('time [s]');
669         ylabel('acceleration [m/s^2]');
670         title('Measured acceleration (xyz)');
671         grid on;
672         axis([sig.t(1),sig.t(end),-2,2]);
673
674         subplot(sig.num,2,2*j);
675         plot(sig.t,sim.a_env(:,j));
676         xlabel('time [s]');
677         ylabel('acceleration [m/s^2]');
678         title('Simulated acceleration (xyz)');
679         grid on;
```

```
680         axis([sig.t(1),sig.t(end),-2,2]);
681     end;
682
683     figure()
684     n_avg  = 2^7;
685     win    = 2^(floor((log(size(sig.a,1))-log(n_avg))/log(2)));
686     for j = 1:sig.num
687         subplot(sig.num,1,j);
688         pwelch(sig.a(:,j),win, win/2, win, sig.fs);hold on;
689         pwelch(sim.a_env(:,j),win, win/2, win, sig.fs);
690         title('Acceleration spectrum (xyz)');
691         legend('measured','simulated','Location','EastOutside');
692         set(gca,'xscale','log');
693     end;
694 end;
695
696 if b.overwrite_files
697     save('cold_plate_data_identification_result.mat','Gn','fftcoefs','fftlen');
698 end;
```

Listing B.20: artificial_cold_plate_noise.m

```
1  function a_env = artificial_cold_plate_noise(len)
2
3  %   a_env = artificial_cold_plate_noise( len )
4  %
5  % Generate artificial cold plate vibrations, based on a model of both the
6  % spectrum and the pulse tube modulation
7  %
8  % Inputs:   len      Required number of samples
9  %
10 % Outputs:  a_env    Artificial cold plate data, one row per sample, three
11 %                    columns: (x,y,z)
12
13 load('cold_plate_data_identification_result.mat')
14 % contains:'Gn','fftcoefs','fftlen';
15
16 %% Reconstruct envelope
17 env = nan( (len/fftlen+2)*fftlen , length(fftcoefs) );
18 for j = 1:length(fftcoefs)
19     % DC and positive harmonics:
20     fftrec   = [ fftcoefs{j};
21                  zeros(fftlen+1-2*length(fftcoefs{j}),1);
22                  conj(fftcoefs{j}(end:-1:2))];
23     env(:,j) = repmat(ifft(fftrec),len/fftlen+2,1);
24 end;
25
26 %% Reconstruct noise spectrum
27 e = wgn(size(env,1),length(fftcoefs),0);
```

```
28 e_env = e .* env;
29 a_env_temp = lsim(Gn,e_env);
30 a_env = a_env_temp(end-len+1:end,:);
```

B-2-2 Shaker table and Cryo Vibration Isolation Platform

Listing B.21: compress_PO_MOESP.m

```
1  function [R_1122, R_3166] = compress_PO_MOESP( u, y, s, l0 )
2
3  %   [R_1122, R_3166] = compress_PO_MOESP( u, y, s, l0 )
4  %
5  % Separate PO-MOESP RQ compression for a batch of roving sensor experiments
6  % with equal input sequence.
7  % See Eq. (2-18) on page 14.
8  %
9  % Inputs:  u       Input sequence, one column per input channel, one row per
10 %                  sample
11 %          y       Cell array with output sequences, one cell per experiment,
12 %                  and therein a matrix with one column per input channel,
13 %                  one row per sample
14 %          s       Parameter in the subspace identification algorithm
15 %          l0      Number of reference output channels
16 %
17 % Outputs: R_1122  Upper left block row of the compressed data matrix, equal
18 %                  for all experiments: R_1122 = [ R11, 0   ;
19 %                                                  R21, R22 ];
20 %          R_3166  3D matrix with lower block row of compressed data
21 %                  matrices, one page per experiment:
22 %     R_3166(:,:,j) = [ R31(0,j) R32(0,j) R33(0,j) 0        0        0        ;
23 %                       R41(j,j) R42(j,j) R43(j,j) R44(j,j) 0        0        ;
24 %                       R51(0,j) R52(0,j) R53(0,j) R54(0,j) R55(0,j) 0        ;
25 %                       R61(j,j) R62(j,j) R63(j,j) R64(j,j) R65(j,j) R66(j,j) ]
26
27 %% Choleski factorisation of the inputs and outputs
28 if isempty(y{1})
29     y(1) = [];         % This odd syntax removes the first cell
30 end;
31
32 % Exclude unactuated inputs
33 m_input  = size( u, 2 );
34 nonzero_inputs = ~all(u==0);
35 u        = u(:,nonzero_inputs);
36
37 m        = size( u, 2 );
38 J        = length(y);
39 [N,l]    = size(y{1});
40 if N<2*s*(m+l)
41     error('The data sequence is too short for this value of s')
```

```
42 end;
43
44 R_1122   = zeros( 2*m*s, 2*m*s );
45 R_3166   = nan( 2*l*s, 2*s*(m+l), J );
46 P        = block_hankel_permutation( s, m, l0, l-l0 );
47
48 try
49     for j = 1:J
50         PGPt   = P * hankel_gram( [u,y{j}], 2*s ) * P.';
51         R      = chol( PGPt , 'lower' );
52         R_1122 = R_1122 + 1/J * R(1:2*m*s, 1:2*m*s);
53         R_3166(:,:,j) = R (2*m*s+1:end,:);
54     end
55 catch ME
56     if strcmp (ME.identifier,'MATLAB:posdef')
57         warning('Choleski failed, using RQ instead')
58         R_1122 = zeros( 2*m*s, 2*m*s );
59         for j = 1:J
60             Rt = triu(qr((P*blockhankel( [u,y{1}], 2*s )).',0));
61             R_1122 = R_1122 + 1/J * Rt(1:2*m*s, 1:2*m*s).';
62             R_3166(:,:,j) = Rt(1:2*s*(m+l),2*s*m+1:2*s*(m+l)).';
63         end
64     else
65         rethrow(ME);
66     end;
67 end
68
69 %% Augment result with zeros for unactuated inputs if necessary
70 if ~all(nonzero_inputs)
71     R_1122temp = R_1122;
72     R_1122 = zeros(2*s*m_input,2*s*m_input);
73     nonzero_inputs = repmat(nonzero_inputs,1,2*s);
74     R_1122(nonzero_inputs,nonzero_inputs) = R_1122temp;
75     R_3166temp = R_3166;
76     R_3166 = zeros( 2*l*s, 2*s*(m+l), J );
77     R_3166(:,[nonzero_inputs,true(1,2*l*s)],:) = R_3166temp;
78 end
```

Listing B.22: svd_PO_MOESP.m

```
1 function varargout = svd_PO_MOESP( R_3166, s, l0, varargin )
2
3 %                  S = svd_PO_MOESP( R_3166, s, l0 )
4 %   [ OsOT, TinvXi ] = svd_PO_MOESP( R_3166, s, l0, n )
5 %     [ Un, Sn, Vn ] = svd_PO_MOESP( R_3166, s, l0, n )
6 %
7 % Singular value decomposition for a batch of R_{52:54}^(0,j) matrices for
8 % all experiments.
9 % See Eq. (2-21) on page 15.
```

```
10 %
11 % Inputs:  R_3166  R_3166  3D matrix with lower block row of compressed data
12 %                  matrices, one page per experiment:
13 %     R_3166(:,:,j) = [ R31(0,j) R32(0,j) R33(0,j) 0        0        0        ;
14 %                       R41(j,j) R42(j,j) R43(j,j) R44(j,j) 0        0        ;
15 %                       R51(0,j) R52(0,j) R53(0,j) R54(0,j) R55(0,j) 0        ;
16 %                       R61(j,j) R62(j,j) R63(j,j) R64(j,j) R65(j,j) R66(j,j) ]
17 %          s       Parameter in the subspace identification algorithm
18 %          l0      Number of reference output channels
19 %          n       Desired model order
20 % Outputs: S       Vector with the first s singular values of the juxtaposed
21 %                  [ R_{52:54}^(0,1) , ... , R_{52:54}^(0,J) ] matrix
22 %          Os0T    Extended local observability matrix for the reference
23 %                  outputs in some base T
24 %          TinvXi  3D matrix with the generalised state sequence in some
25 %                  base T, one page per experiment
26 %          Un      left singular vector matrix of the order-n trucated svd.
27 %          Sn      diagonal singular value matrix of the order-n trucated svd.
28 %          Vn      right singular vector matrix of the order-n trucated svd.
29 %
30 % Alternative use for local observability matrix including roving sensor
31 % outputs: enter an R_3166 matrix with only one page (=one experiment) and
32 % enter (l0+lref) for l0. Then Os^(j)*T^(j) = OsT( 1:s*l0 , : ) and
33 % Os^(j)*T^(j) = OsT( s*l0+1 : end , : )
34
35 sl = size(R_3166,1) / 2;
36 sm = size(R_3166,2) / 2 - sl;
37 J  = size(R_3166,3);
38
39 if sl == s*l0 && J>1
40     warning([ 'You may be doing something wrong: Single Factorisa'...
41               'tion for roving outputs of multiple experiments is illegal.'...
42               ' Include only reference sensor data.']);
43 end
44
45 % R_{52:54}^(0,j) = R_3166( sl+1:sl+s*l0 , sm+1:2*sm+sl , j )
46 if nargout == 1 && nargin == 3                      % Output: S
47     % Singular value decomposition
48     varargout{1} = svd( reshape( ...
49           R_3166( sl+1:sl+s*l0 , sm+1:2*sm+sl , :), s*l0, J*(sm+sl) ) );
50     varargout{1} = varargout{1}( 1:s );
51 elseif nargout == 2 && nargin == 4                  % Output: [ Os0T, TinvXi ]
52     % Singular value decomposition
53     [ varargout{1}, S, V ] = svd( reshape( ...
54           R_3166( sl+1:sl+s*l0 , sm+1:2*sm+sl , :), s*l0, J*(sm+sl) ) ,'econ');
55     % Truncate order
56     n = varargin{1};
57     varargout{1} = varargout{1}( : , 1:n );
58     varargout{2} = reshape( S( 1:n , 1:n ) * V( : , 1:n)', n, sm+sl, J );
```

```
59 elseif nargout == 3 && nargin == 4                    % Output: [ Un, Sn, Vn ]
60     % Singular value decomposition
61     [ varargout{1}, varargout{2}, varargout{3} ] = svd( reshape( ...
62           R_3166( sl+1:sl+s*l0 , sm+1:2*sm+sl , :), s*l0, J*(sm+sl) ) ,'econ');
63     % Truncate order
64     n = varargin{1};
65     varargout{1} = varargout{1}( : , 1:n );
66     varargout{2} = varargout{2}( 1:n , 1:n );
67     varargout{3} = ...
68         permute( reshape( varargout{3}( : , 1:n).' , n, sm+sl, J ), [2 1 3] );
69 else
70     error('Illegal number of input/output parameters');
71 end
```

Listing B.23: algorithm_Ia.m

```
1  function Os_ord = algorithm_Ia( Os0jT, s, l0, jbase)
2
3  %   Os_ord = algorithm_Ia( Os0jT, s, l0, jbase)
4  %
5  % Calculates an estimation of the ordinary extended observability matrix for
6  % the full virtual sensor grid, using Algorithm Ia
7  %
8  % Inputs:  Os0jT   Cell array with one cell per experiment with each cell
9  %                  containing the estimated stacked extended observability
10 %                  matrix [ Os^(0); Os^(j) ] * T^(j) resulting from a separate
11 %                  singular value decomposition for each experiment using
12 %                  both reference and roving sensor data. The size of these
13 %                  matrices must be (s*l0 + s*lj) x n
14 %          s       Parameter in the subspace identification algorithm
15 %          l0      Number of reference sensor channels.
16 %          jbase   The number of the experiment of which the base is used.
17 %
18 % Outputs: Os_ord  Estimation of the ordinary extended observability matrix
19 %                  for the full virtual sensor grid
20 %
21 % See Section 2-2-5 on page 15.
22
23 sl      = size(Os0jT{jbase},1);
24 n       = size(Os0jT{jbase},2);
25 J       = length(Os0jT);
26 l       = sl/s;
27 lj      = l-l0;
28
29 %% Fill a global observability matrix
30 Os_ord  = nan( s*l0+J*s*lj, n );
31 Os_ord( 1:s*l0 , : ) = Os0jT{jbase}( 1:s*l0 , : );   % Os^(0) * T^(jbase)
32 for j = 1:J
33     if j==jbase                                      % Os^(jbase) * T^(jbase)
```

```
34          Os_ord( s*l0+(j-1)*s*lj+1:s*l0+j*s*lj , : ) = ...
35                                                    Os0jT{j}( s*l0+1:end , : );
36      else                                             % Os^(j) * T^(jbase)
37          Os_ord( s*l0+(j-1)*s*lj+1:s*l0+j*s*lj , : ) = ...
38              Os0jT{j}( s*l0+1:end , : ) / Os0jT{j}( 1:s*l0 , : ) ...
39                                                  * Os0jT{jbase}( 1:s*l0 , : );
40      end;
41  end;
42
43  %% Permute rows to obtain an ordinary extended observability matrix
44  [~,P]  = global_obsv_permutation( s, l0, lj, J);
45  Os_ord = P * Os_ord;
```

Listing B.24: algorithm_IIa.m

```
1   function Os_ord = algorithm_IIa( R_3166, s, Os0T, TinvXi, varargin)
2
3   %   Os_ord = algorithm_IIa( R_3166, s, Os0T, TinvXi(, jbase))
4   %
5   % Calculates an estimation of the ordinary extended observability matrix for
6   % the full virtual sensor grid, using Algorithm IIa
7   %
8   % Inputs:  R_3166  3D matrix with lower block row of compressed data
9   %                  matrices, one page per experiment:
10  %     R_3166(:,:,j) = [ R31(0,j) R32(0,j) R33(0,j) 0        0        0        ;
11  %                       R41(j,j) R42(j,j) R43(j,j) R44(j,j) 0        0        ;
12  %                       R51(0,j) R52(0,j) R53(0,j) R54(0,j) R55(0,j) 0        ;
13  %                       R61(j,j) R62(j,j) R63(j,j) R64(j,j) R65(j,j) R66(j,j) ]
14  %          s       Parameter in the subspace identification algorithm
15  %          Os0T    Extended local observability matrix for the reference
16  %                  outputs in some base T
17  %          TinvXi  3D matrix with the generalised state sequence in some
18  %                  base T, one page per experiment
19  %          jbase   Optional: the number of the experiment of which the
20  %                  generalised state sequence is used. Default: 1
21  %
22  % Outputs: Os_ord  Estimation of the ordinary extended observability matrix
23  %                  for the full virtual sensor grid
24  %
25  % See Section 2-2-5 on page 15.
26
27  warning(['This function is equivalent to algorithm_IIb, but takes more '...
28           'computation time. Preferrably use algorithm_IIb.']);
29  sl      = size(R_3166,1) / 2;
30  sm      = size(R_3166,2) / 2 - sl;
31  J       = size(R_3166,3);
32  [sl0,n] = size(Os0T);
33  slj     = sl - sl0;
34  if nargin == 5
```

```
35      jbase = varargin{1};
36  else
37      jbase = 1;
38  end;
39
40  %% Fill a global data matrix
41  Os_ordT  = nan( sl0+J*slj, n);
42  Os_ordT( 1:sl0 , : ) = Os0T;
43  for j = 1:J
44      % Estimated local data matrix for roving output j for generalised state
45      % sequence jbase
46      % R_{62:64}^(j,j) = R_3166( sl+sl0+1:end , sm+1:2*sm+sl , j )
47      Os_ordT ( sl0+(j-1)*slj+1:sl0+j*slj , : ) = ...
48          R_3166( sl+sl0+1:end , sm+1:2*sm+sl , j ) / TinvXi(:,:,j);
49  end;
50  R_glob = Os_ordT * TinvXi(:,:,jbase);
51
52  %% Global SVD
53  [ Os_ord, ~, ~ ] = svd( R_glob, 'econ');
54  Os_ord = Os_ord(:,1:n);
55
56  %% Permute rows to obtain an ordinary extended observability matrix
57  [~,P]  = global_obsv_permutation( s, sl0/s, slj/s, J);
58  Os_ord = P * Os_ord;
```

Listing B.25: algorithm_IIb.m

```
1   function Os_ord = algorithm_IIb( R_3166, s, Os0T, TinvXi)
2
3   %   Os_ord = algorithm_IIb( R_3166, s, Os0T, TinvXi)
4   %
5   % Calculates an estimation of the ordinary extended observability matrix for
6   % the full virtual sensor grid, using Algorithm IIb
7   %
8   % Inputs:  R_3166  3D matrix with lower block row of compressed data
9   %                  matrices, one page per experiment:
10  %     R_3166(:,:,j) = [ R31(0,j) R32(0,j) R33(0,j) 0        0        0        ;
11  %                       R41(j,j) R42(j,j) R43(j,j) R44(j,j) 0        0        ;
12  %                       R51(0,j) R52(0,j) R53(0,j) R54(0,j) R55(0,j) 0        ;
13  %                       R61(j,j) R62(j,j) R63(j,j) R64(j,j) R65(j,j) R66(j,j) ]
14  %          s       Parameter in the subspace identification algorithm
15  %          Os0T    Extended local observability matrix for the reference
16  %                  outputs in some base T
17  %          TinvXi  3D matrix with the generalised state sequence in some
18  %                  base T, one page per experiment
19  %
20  % Outputs: Os_ord  Estimation of the ordinary extended observability matrix
21  %                  for the full virtual sensor grid
22  %
```

```
% See Section 2-2-5 on page 15.

sl      = size(R_3166,1) / 2;
sm      = size(R_3166,2) / 2 - sl;
J       = size(R_3166,3);
[sl0,n] = size(Os0T);
slj     = sl - sl0;

%% Fill a global observability matrix
Os_ord  = nan( sl0+J*slj, n );
Os_ord( 1:sl0 , : ) = Os0T;       % Local extended obsv mx for reference output

% R_{62:64}^(j,j) = R_3166( sl+sl0+1:end , sm+1:2*sm+sl , j )
for j = 1:J
    % Local extended obsv mx for roving output j
    Os_ord( sl0+(j-1)*slj+1:sl0+j*slj , : ) = ...
                      R_3166( sl+sl0+1:end , sm+1:2*sm+sl , j ) / TinvXi(:,:,j);
end

%% Permute rows to obtain an ordinary extended observability matrix
[~,P]  = global_obsv_permutation( s, sl0/s, slj/s, J);
Os_ord = P * Os_ord;
```

Listing B.26: mdac2bd.m

```
function varargout = mdac2bd( A, C, u, y, l0, varargin)

%   [B,D,x0] = mdac2bd( A, C, u, y ,l0 (,Rin))
%   Rout = mdac2bd( A, C, u, y ,l0 (,Rin))
%   [B,D] = mdac2bd( A, C, R_1122, R_3166, l0 )
%
% Estimates the pair (B,D) from the input and output signals or the compressed
% data matrices and the estimated pair (A,C) for sensor roving identification.
% If u and y are provided an algorithm based on LRJ Haverkamp, 'State space
% identification - Theory and practice' PhD-thesis (2001) is used and the
% initial state can be calculated on the fly.
% If the compressed data matrices R_1122 and R_3166 are provided, an
% algorithm based on Verhaegen M, Dewilde Pă'Subspace model identification
% part 1. The output-error state-space model identification class of
% algorithms.' (1992)ăInternational journal of Control 56(5):1187-1210 is used.
%
% Inputs:  A, C    System matrices, for virtual sensor grid
%          u       Input sequence, one column per input channel, one row per
%                  sample
%          y       Cell array with output sequences, one cell per experiment,
%                  and therein a matrix with one column per input channel,
%                  one row per sample
%          R_1122  Upper left block row of the compressed data matrix, equal
%                  for all experiments: R_1122 = [ R11, 0   ;
```

```
25 %                                                R21, R22 ];
26 %          R_3166  3D matrix with lower block row of compressed data
27 %                  matrices, one page per experiment:
28 %     R_3166(:,:,j) = [ R31(0,j) R32(0,j) R33(0,j) 0        0        0        ;
29 %                       R41(j,j) R42(j,j) R43(j,j) R44(j,j) 0        0        ;
30 %                       R51(0,j) R52(0,j) R53(0,j) R54(0,j) R55(0,j) 0        ;
31 %                       R61(j,j) R62(j,j) R63(j,j) R64(j,j) R65(j,j) R66(j,j) ]
32 %          l0      Number of reference outputs
33 %                  (Other dimension parameters are inferred from the
34 %                  sizes of the other inputs.)
35 %
36 % Outputs: B, D    System matrices, for virtual sensor grid
37 %
38 % See Section 2-2-7 on page 23.
39
40 if size(u,1)==size(u,2) && all(all(triu(u,1)==0));
41 %% If compressed data is provided
42
43 if nargout == 3
44     error('When compressed data is provided, mdac2bd does not return x0.')
45 end;
46 R_1122 = u;
47 R_3166 = y;
48
49 J    = size(R_3166,3);      % Number of experiments & roving sensor positions
50 lv   = size(C,1);           % Number of virtual sensor outputs
51 lj   = (lv-l0)/J;           % Number of roving sensor outputs
52 l    = l0+lj;               % Total number of outputs per experiment
53 sl   = size(R_3166,1)/2;
54 s    = sl/l;                % Number of block rows of extended obsv mx
55 sm   = size(R_1122,1)/2;
56 m    = sm/s;                % Number of inputs
57 n    = size(A,1);           % System order
58 l0lj = [l0, lj*ones(1,(J*(lj>=1)))];
59
60 %% Observability matrices and friends
61
62 % Global observability matrix
63 Os     = obsv_ext( A, C, s);                        % Ordinary obsv mx
64 [~,P]  = global_obsv_permutation( s-1, l0, lj, J); % Permutation matrix
65 Osmin1 = P.' * Os ( 1:(s-1)*lv , : );               % Global obsv mx, order s-1
66 [~,P]  = global_obsv_permutation( s, l0, lj, J);   % Permutation matrix
67 Os     = P.' * Os;                                  % Global obsv mx, order s
68
69 % Local observability matrices
70 Osj    = mat2cell( Os, s*l0lj ,n);  % index 1 -> reference
71 clear Os
72
73 % Orhogonal complement of local observability matrices
```

```
Os_orthT = cell(J*(lj>=1)+1,1);

for j = 1:J*(lj>=1)+1
    % Transpose of orthornormal base for left null space
    [Q,~] = qr(Osj{j});
    Os_orthT{j} = Q(:,n+1:end).';

    % Check rank condition
    if cond( Os_orthT{j}( : , end-l0lj(j)+1:end )) > 1e3
        error( ['Rank condition for (B,D) estimation not satisfied. \n'...
                'Repeat with larger s'                                ]);
    end;
end;

% M
M = [ blkdiag( [ spalloc( l0,(s-1)*l0,0 ) ;
                  speye( (s-1)*l0 )        ]                    ,...
               kron( speye(J), [ spalloc( lj,(s-1)*lj,0 ) ;
                                 speye( (s-1)*lj )        ] ) ) * Osmin1 , ...
      full( blkdiag( [ speye( l0 )              ;
                       spalloc( (s-1)*l0,l0,0 ) ]                  ,...
                     kron( speye(J), [ speye( lj )               ;
                                       spalloc( (s-1)*lj,lj,0 ) ] ) ) ) ];

clear Osmin1

% Omega
Os_orthTtriu = cell(J*(lj>=1)+1,1);
warning('off','obsv_ext:s_smaller_than_n');
for j = 1:J*(lj>=1)+1
    Os_orthTtriu{j} = obsv_ext( Delta( l0lj(j), s), Os_orthT{j}, s);
end;
warning('on','obsv_ext:s_smaller_than_n');

OmegaM = blkdiag( sqrt(J) * sparse(Os_orthTtriu{1}), Os_orthTtriu{2:end} ) * M;

clear Os_orthTtriu

%% Data matrices
R_inputs            = [ R_1122( sm+1:2*sm , :) , R_1122( 1:sm , 1:sm ) ];
R_outputs           = nan( s*lv, 3*sm);
R_outputs(1:s*l0,:) = [ sum( R_3166( 1:s*l0 , 1:2*sm , : ), 3) ,...
                        sum( R_3166( s*l+1:s*(l+l0) , 1:sm , : ), 3) ]/sqrt(J);
for j = 1:J
    R_outputs( s*(l0+(j-1)*lj)+1:s*(l0+j*lj) , : ) = ...
        [ R_3166( s*l0+1:s*l , 1:2*sm , j) , ...
          R_3166( s*(l+l0)+1:end , 1:sm , j) ];
end;
Upsilon = blkdiag( Os_orthT{:} ) * R_outputs / R_inputs;      % Upsilon_bar
```

```
123
124 if lj>=1
125     Upsilon = [ block_vectorise( Upsilon( 1:s*l0-n     , : ), s*l0-n, m) ;
126                 block_vectorise( Upsilon( s*l0-n+1:end , : ), s*lj-n, m) ];
127 else % lj=0
128     Upsilon = block_vectorise( Upsilon( 1:s*l0-n     , : ), s*l0-n, m) ;
129 end;
130
131 clear Os_orthT R_outputs R_inputs
132
133 %% Solve overdetermined equation
134 BD = ( OmegaM ) \ Upsilon;
135 varargout{1} = BD( 1:n , : );        %B
136 varargout{2} = BD( n+1:end , : );    %D
137
138 else
139 %% Uncompressed data is provided
140
141 [N,m]  = size(u);
142 [lv,n] = size(C);
143 J      = length(y);
144 lj     = (lv-l0)/J;
145
146 %% Sanity checks
147 if size(C,2)~=n
148     error('Size of C not compatible to size of A.');
149 end;
150 if l0>lv
151     error(['Number of reference outputs is '...
152            'larger than the total number of outputs'])
153 end;
154 for j=1:J
155     if size(y{j},1)~=N
156         error(['Lengths of output ' num2str(j) ...
157                ' and maybe others differs from the length of the input']);
158     end;
159     if size(y{j},2)~=l0+lj
160         error(['Number of output signals for experiment ' num2str(j) ...
161             ' and maybe others not compatible to size of C, or l0 incorrect']);
162     end;
163 end;
164
165 %% Change base to Schur form to improve computational efficiency
166 [T,A1] = schur(A);
167 C1     = C*T;
168
169 %% Split data in time batches to avoid excessive memory usage
170 maxnumel                = 1e8;                    % Rough maximum numel(PhiTh)
171 numbatches              = ceil(N*lv*(n+n*m+m)/maxnumel);
```

```
maxbatchlength           = ceil(N/numbatches);
bl                       = maxbatchlength*ones(numbatches,1);   % batch lengths
numsmaller               = numbatches*maxbatchlength-N;
bl(end-numsmaller+1:end) = maxbatchlength-1; % minimize changes in matrix sizes
batchind                 = [cumsum([1;bl(1:end-1)]), cumsum(bl)];
                                                    % first and last indices

%% Init
sizesingle   = n + m*n + m;               % single exp., scalar signal
sizerefil    = J*n + m*n + m;             % all exp., scalar signal
sizeref      = J*n + m*n + l0*m;          % all exp., vector ref. signal
sizerovj     = n + m*n + m*lj;            % single exp., vector rov. signal
sizerov      = J*n + m*n + J*m*lj;        % all exp., vector rov. signal
sizeall      = J*n + m*n + m*l0 + J*m*lj; % all exp., all output signals
% Init with zeros is necessary for the following matrices. Because the zero
% columns remain in the same positon in each loop iteration, resetting to zero
% is not necessary.
Mrefil       = zeros( J * sizesingle,  sizerefil+ 1 );
Mref         = zeros( l0* sizerefil,   sizeref  + 1 );
Mrovj        = zeros( lj* sizesingle,  sizerovj + 1 );
Mrov         = zeros( J * sizerovj,    sizerov  + 1 );
Mrefrov      = zeros( sizeref+sizerov, sizeall  + 1 );
Mall         = zeros( numbatches * sizeall, sizeall + 1 );
% Start without initial condition for each of the m*n generating lti systems
% for Yij
X0new        = [];
% A1batchstart is used for calculating the extended observability matrix
% with a batch offset.
A1batchstart = eye(n);

%% Iterate
for ib = 1:numbatches % for all batches
    disp(['Processing batch ' num2str(ib) ' of ' num2str(numbatches) '.']);

    [Y_ij,X0new] = Yij(A1,C1,u(batchind(ib,1):batchind(ib,2),:),X0new);

    %% Reference sensor data
    for il = 1:l0 % for all reference output signals
        Ossingle = obsv_ext(A1,C1(il,:)*A1batchstart,bl(ib));
        for j = 1:J % for all experiments
            % Compose
            PhiTh = [Ossingle, ...
                     Y_ij((il-1)*bl(ib)+1:il*bl(ib),:), ...
                     u(batchind(ib,1):batchind(ib,2),:), ...
                     y{j}(batchind(ib,1):batchind(ib,2),il)];
            % Compress
            Rsingle = triu(qr(PhiTh,0));
            % Accumulate: single reference output channel for all experiments
            Mrefil((j-1)*sizesingle+1 : j*sizesingle ,...
```

```
221                     [(j-1)*n+(1:n), end-(n*m+m):end]) = Rsingle(1:sizesingle,:);
222         end
223         % Compress
224         Rrefil = triu(qr(Mrefil,0));
225         % Accumulate: all reference output channels for all experiments
226         Mref((il-1)*sizerefil+1:il*sizerefil,...
227             [1:J*n+m*n,J*n+m*n+((il-1)*m+(1:m)), end]) = Rrefil(1:sizerefil,:);
228     end
229     % Compress
230     Rref = triu(qr(Mref,0));
231
232     %% Roving sensor data
233     if lj>0                         % If roving sensors used
234         for j = 1:J                 % for all experiments
235             for il = l0+1:l0+lj     % for all roving output signals
236                 Ossingle = obsv_ext(A1,C1(il+(j-1)*lj,:)*A1batchstart,bl(ib));
237                 % Compose
238                 PhiTh = [Ossingle, ...
239                          Y_ij((il-1)*bl(ib)+1:il*bl(ib),:), ...
240                          u(batchind(ib,1):batchind(ib,2),:), ...
241                          y{j}(batchind(ib,1):batchind(ib,2),il)];
242                 % Compress
243                 Rsingle = triu(qr(PhiTh,0));
244                 % Accumulate: all roving outputs for a single experiment
245                 Mrovj((il-l0-1)*sizesingle+1 : (il-l0)*sizesingle , ...
246                     [1:n+m*n, n+m*n+(il-l0-1)*m+1 : n+m*n+(il-l0)*m, end]) =...
247                                                     Rsingle(1:sizesingle,:);
248             end;
249             % Compress
250             Rrovj = triu(qr(Mrovj,0));
251             % Accumulate: all roving output channels for all experiments
252             Mrov((j-1)*sizerovj+1:j*sizerovj,...
253                     [(j-1)*n+1:j*n, J*n+1:J*n+m*n,...
254                     J*n+m*n+ (j-1)*m*lj+1:J*n+m*n+j*m*lj, end]) = ...
255                                                       Rrovj(1:sizerovj,:);
256         end
257         % Compress
258         Rrov = triu(qr(Mrov,0));
259
260         %% Accumulate and compress all sensor data for this batch
261         Mrefrov(1:sizeref,[1:sizeref, end]) = Rref(1:sizeref,:);
262         Mrefrov(sizeref+1:end, [1:J*n+m*n, sizeref+1:sizeref+J*m*lj, end]) =...
263                                                         Rrov(1:sizerov,:);
264         Rrefrov = triu(qr(Mrefrov),0);
265         Mall((ib-1)*sizeall+1:ib*sizeall,:) = Rrefrov(1:sizeall,1:sizeall+1);
266     else % No roving sensors used
267         Mall((ib-1)*sizeall+1:ib*sizeall,:) = Rref(1:sizeref,:);
268     end
269
```

```
270     if ib < numbatches
271         A1batchstart = A1batchstart*A1^(bl(ib));
272     end;
273 end
274 
275 if nargin==5
276     Rin=zeros(0,sizeall + 1);
277 else
278     Rin=varargin{1};
279 end;
280 
281 if nargout==1 % Output data matrix to be used in next batch
282     varargout{1}=triu(qr([Mall;Rin]),0);
283 else
284     x0BD = [Mall(:,1:end-1);Rin(:,1:end-1)]\[Mall(:,end);Rin(:,end)];
285     if nargout>2
286         x0 = nan(n,J);
287         x0(:) = x0BD(1:J*n);
288         x0 = mat2cell(T*x0,n,ones(1,J));  % Change base back from schur to
        original
289         varargout{3} = x0;
290     end;
291     varargout{1} = nan(n,m); varargout{1}(:) = x0BD( J*n+1 : J*n+n*m);    % B
292     varargout{1} = T*varargout{1};    % Change base back from schur to original
293     DT = nan(m,lv);DT(:) = x0BD(J*n+n*m+1:end);
294     varargout{2} = DT.';                                                  % D
295 end
296 end
297 end
298 
299 function Del = Delta(lj,s)
300 Del = sparse( lj+1:lj*s, 1:(s-1)*lj, 1, s*lj, s*lj);
301 end
302 
303 function [Y_ij,X0new] = Yij(A1,C1,u,varargin)
304     m = size(u,2);
305     n = size(A1,1);
306     N = size(u,1);
307     l = size(C1,1);
308     Y_ij = nan(N*l,n*m);
309     X0new = cell(m,n);
310     e = eye(n);
311     s = (1:n) + (diag(blkdiag(A1,0),-1)~=0).'; % blkdiag is needed for n=1
312     if nargin>3 && ~isempty(varargin{1})    % If initial state vectors provided
313         for i = 1:n
314             sysout = ss(A1(1:s(i),1:s(i)),e(1:s(i),i),C1(:,1:s(i)),[],-1);
315             for j = 1:m
316                 [yij,~,xij] = lsim(sysout,u(:,j),[],varargin{1}{i,j});
317                 Y_ij(:,(j-1)*n+i) = yij(:);
```

```
318                 X0new{i,j} = xij(end,:)*A1(1:s(i),1:s(i)).' +...
319                                 u(end,j)* e(1:s(i),i).'; % Needs one step ahead
320             end
321         end
322     else
323         for i = 1:n
324             sysout = ss(A1(1:s(i),1:s(i)),e(1:s(i),i),C1(:,1:s(i)),[],-1);
325             for j = 1:m
326                 [yij,~,xij] = lsim(sysout,u(:,j));
327                 Y_ij(:,(j-1)*n+i) = yij(:);
328                 X0new{i,j} = xij(end,:)*A1(1:s(i),1:s(i)).' +...
329                                 u(end,j)* e(1:s(i),i).'; % Needs one step ahead
330             end
331         end
332     end
333 end
```

Listing B.27: identification_step_by_step_pa.m

```
1  % This script performs PO-MOESP with sensor roving step by step, using only
2  % response data from only piezo actuation.
3  % See Section 2-2-9 on page 33.
4
5  clear variables;
6  load('data_1130_clean');
7
8  %% Select step and (B,D) estimation algorithm
9  step   = 5
10 BD_alg = 1
11
12 %% Describing parameters of the sensor roving experiment
13 if step==1||step==2
14     l0 = 6;                  % Number reference outputs (including piezo)
15     lj = 0;                  % Number of roving outputs per experiment
16 elseif step==3
17     l0 = 9;
18     lj = 0;
19 elseif step==4||step==5
20     l0 = 6;
21     lj = 3;
22 end;
23 if step==1||step==3||step==4
24     experiment = 8;          % Centre of payload
25 elseif step==2||step==5
26     experiment = 2:14;       % Position 1 is the reference sensor
27 end;
28 J  = length(experiment);    % Number of experiments
29 m  = 6;                     % Number of inputs
30 l  = l0+lj;                 % Total number of outputs per experiment
```

```
lv = l0+J*lj;                   % Number of virtual outputs

%% Split train data and validation data
splitind     = round( size(meas.pavc.u,1) * 2/3 );
delay        = 2;
train.pa.u   = meas.pa.u(1:splitind-delay,:);
val.pa.u     = meas.pa.u(splitind+1-delay:end-delay,:);
train.pa.y   = cell(J,1);
val.pa.y     = cell(J,1);

for j = 1:J
    train.pa.y{j}   = meas.pa.y{experiment(j)}(delay+1:splitind,1:l);
    val.pa.y{j}     = meas.pa.y{experiment(j)}(splitind+1:end,1:l);
end;

meas = rmfield(meas,{'pa','vc','pavc'}); % Not needed anymore, free up memory

%% Weighting of the train data: Emphasise parts of the reference sensor
% data, that show the relevant modes for the roving sensor data
% Static weighting of the output data only, needs compensation in (C,D)
outputweights = [ [1 1 1 1 1 50] , repmat([2 1 10],1,J*(step>2)) ];
disp('outputweights:');
disp(outputweights);
train.pa.y_w = cell(J,1);
for j = 1:J
    train.pa.y_w{j}   = train.pa.y{j} * ...
                            diag( outputweights([1:l0,l0+(j-1)*lj+1:l0+j*lj]) );
end;

%% PO-MOESP compression of the data per experiment
disp('Compressing the data')
s = 150;
disp(['s = ' num2str(s)]);
[R_1122, R_3166] = compress_PO_MOESP(train.pa.u,train.pa.y_w,s,l0);

%% PO-MOESP identification of the observability matrix
S = svd_PO_MOESP( R_3166, s, l0 );

figure('WindowStyle','normal','units','normalized','outerposition',[0 0 1 1]);
semilogy( 1:s, S(1:s), '+' ); grid on; hold on;
semilogy( 5:5:s, S(5:5:s), '+'); hold off;
labels = cellstr(num2str((5:5:s).'));
text( 5:5:s, S(5:5:s), labels, 'VerticalAlignment','bottom', ...
                                             'HorizontalAlignment','center')
% Model order
n = 60;
disp(['n = ' num2str(n)]);

% SVD for reference data
```

```
80  [ OsOT, TinvXi ] = svd_PO_MOESP(R_3166,s,l0,n);
81
82  % Scaling of the moving sensor data
83  Os = algorithm_IIb( R_3166, s, OsOT, TinvXi);
84
85  %% Estimation of the system matrices
86  % Estimate (A,C) using the shift-invariance property
87  disp('(A,C) estimation')
88  Ci = Os( 1 : lv , : ); Ci = diag(1./outputweights)*Ci;
89  Ai = Os( 1 : (s-1)*lv , : ) \ Os( lv+1 : end , : );
90
91  if max(abs(eig(Ai)))>1;
92      % Slightly move marginally stable poles that are estimated as unstable
93      warning('Identified A matrix is unstable, trying to recover.');
94      [V,D]       = eig(Ai);
95      D(abs(D)>1) = (1-1e-8) * D(abs(D)>1) ./ abs( D(abs(D)>1) );
96      Ai          = real(V*D/V);
97  end;
98
99  % Estimate (B,D)
100 disp('(B,D) estimation')
101
102 if BD_alg==1
103     [Bi,Di]= mdac2bd( Ai, diag(outputweights)*Ci, R_1122, R_3166, l0 );
104 else
105     [Bi,Di]= mdac2bd(Ai, diag(outputweights)*Ci, train.pa.u, train.pa.y_w, l0);
106 end;
107 Di   = diag(1./outputweights)*Di;
108
109 sysi = ss(Ai, Bi, Ci, Di, meas.ts); % Identified model, excluding delay!
110
111 %% Validation
112 disp('VAF evaluation, validation data, only piezo actuators')
113 val.pa.x0    = dinit_batch(sysi,val.pa.u,val.pa.y,l0);
114 val.pa.y_est = lsim_batch(sysi,val.pa.u,val.pa.x0,l0);
115 val.pa.vaf   = vaf_batch(val.pa.y,val.pa.y_est);
116 disp(val.pa.vaf);
```

Listing B.28: identification_proof_of_concept.m

```
1  % This script demonstrates PO-MOESP with sensor roving.
2  % See Section 2-2-10 on page 39.
3
4  clear variables;
5  load('data_1130_clean');
6
7  %% Describing parameters of the sensor roving experiment
8  experiment = 2:7;             % Position 1 is the reference sensor
9  J  = length(experiment);      % Number of experiments
```

```
m  = 3;                         % Number of inputs
l0 = 6;                         % Number of reference outputs
lj = 3;                         % Number of roving outputs per experiment
lp = 3;                         % Number of piezo outputs
l  = l0+lj;                     % Total number of outputs per experiment
lv = l0+J*lj;                   % Number of virtual outputs

%% Select algorithm
AC_alg         = 1 % 1 for Algorithm Ia, 2 for Algorithm IIb
BD_alg         = 1 % 1 for Algorithm 1, 2 for Algorithm 2
baseexperiment = 2 % Base experiment for Algorithm Ia

%% Split train data and validation data
splitind     = round( size(meas.pavc.u,1) * 2/3 );
train.vc.u   = meas.vc.u(1:splitind,4:6);
val.vc.u     = meas.vc.u(splitind+1:end,4:6);
train.vc.y   = cell(J,1);
val.vc.y     = cell(J,1);

for j = 1:J
    train.vc.y{j}   = meas.vc.y{ experiment(j) }( 1:splitind, 1:l );
    val.vc.y{j}     = meas.vc.y{ experiment(j) }( splitind+1:end, 1:l );
end;

meas = rmfield(meas,{'pa','vc','pavc'}); % Not needed anymore, free up memory

%% Weighting of the train data: Emphasise parts of the reference sensor
% data, that show the relevant modes for the roving sensor data
% Static weighting of the output data only, needs compensation in (C,D)
outputweights         = ones(1,lv);
outputweights(6)      = 50;           % Reference az
train.vc.y_w          = cell(J,1);
for j = 1:J
    train.vc.y_w{j}   = train.vc.y{j} * ...
                            diag( outputweights([1:l0,l0+(j-1)*lj+1:l0+j*lj]) );
end;

%% PO-MOESP compression of the data per experiment
disp('Compressing the data')
s = 75;
disp(['s = ' num2str(s)]);
[ R_1122, R_3166 ] = compress_PO_MOESP(train.vc.u,train.vc.y_w,s,l0);

if AC_alg==1 % Algorithm Ia
    jbase = find(experiment==baseexperiment);
    figure('WindowStyle','normal','units','normalized','outerposition',...
                                                                    [0 0 1 1]);
    for j = 1:J
        % Singular values per experiment
```

```
59          S = svd_PO_MOESP( R_3166(:,:,j), s, l );
60          semilogy( 1:s, S(1:s), '+' );    grid on;  hold on;
61          semilogy( 5:5:s, S(5:5:s), '+');
62          labels = cellstr(num2str((5:5:s).'));
63      end;
64      hold off;
65
66      % Model order
67      n = 11;
68      disp(['n = ' num2str(n)]);
69
70      Os0jT = cell(J,1);
71      for j = 1:J
72          % SVD per experiment
73          [ Os0jT{j}, ~ ] = svd_PO_MOESP(R_3166(:,:,j),s,l,n);
74      end;
75
76      % Scaling to combine the experiments
77      Os = algorithm_Ia( Os0jT, s, l0, jbase);
78
79  elseif AC_alg==2 % Algorithm IIb
80      % Singular values for the reference sensor data
81      S = svd_PO_MOESP( R_3166, s, l0 );
82      figure('WindowStyle','normal','units','normalized','outerposition',...
83                                                                  [0 0 1 1]);
84      semilogy( 1:s, S(1:s), '+' );    grid on;  hold on;
85      semilogy( 5:5:s, S(5:5:s), '+'); hold off;
86      labels = cellstr(num2str((5:5:s).'));
87      text( 5:5:s, S(5:5:s), labels, 'VerticalAlignment','bottom', ...
88                                          'HorizontalAlignment','center');
89      % Model order
90      n = 11;
91      disp(['n = ' num2str(n)]);
92
93      % SVD for the reference sensor data
94      [ Os0T, TinvXi ] = svd_PO_MOESP(R_3166,s,l0,n);
95
96      % Scaling of the moving sensor data
97      Os = algorithm_IIb( R_3166, s, Os0T, TinvXi);
98  end;
99
100 %% Estimation of the system matrices
101 % Estimate (A,C) using the shift-invariance property
102 disp('(A,C) estimation')
103 Ci = Os( 1 : lv , : ); Ci = diag(1./outputweights)*Ci;
104 Ai = Os( 1 : (s-1)*lv , : ) \ Os( lv+1 : end , : );
105
106 if max(abs(eig(Ai)))>=1;
107     % Slightly move marginally stable poles that are estimated as unstable
```

```
108     warning('Identified A matrix is unstable, trying to recover.');
109     [V,D]         = eig(Ai);
110     D(abs(D)>=1) = (1-1e-8) * D(abs(D)>=1) ./ abs( D(abs(D)>=1) );
111     Ai            = real(V*D/V);
112 end;
113 
114 % Estimate (B,D)
115 disp('(B,D) estimation')
116 if BD_alg==1
117     [Bi,Di]= mdac2bd(Ai, diag(outputweights)*Ci, R_1122, R_3166, 10);
118 elseif BD_alg==2
119     [Bi,Di]= mdac2bd(Ai, diag(outputweights)*Ci, train.vc.u, train.vc.y_w, 10);
120 end;
121 Di   = diag(1./outputweights)*Di;
122 
123 sysi = ss(Ai, Bi, Ci, Di, meas.ts); % Identified model, excluding delay!
124 
125 
126 %% Validation
127 % Train data
128 disp('VAF evaluation, train data, only voice coil actuators');
129 train.vc.x0    = dinit_batch(sysi, train.vc.u, train.vc.y, 10);
130 train.vc.y_est = lsim_batch(sysi, train.vc.u, train.vc.x0, 10);
131 train.vc.vaf   = vaf_batch(train.vc.y,train.vc.y_est);
132 disp(train.vc.vaf);
133 
134 % Validation data
135 disp('VAF evaluation, validation data, only voice coil actuators');
136 val.vc.x0    = dinit_batch(sysi, val.vc.u, val.vc.y, 10);
137 val.vc.y_est = lsim_batch(sysi, val.vc.u, val.vc.x0, 10);
138 val.vc.vaf   = vaf_batch(val.vc.y,val.vc.y_est);
139 disp(val.vc.vaf);
```

Bibliography

[1] J. Chuai and Y. Tian, "Rank equalities and inequalities for kronecker products of matrices with applications," *Applied Mathematics and Computation*, vol. 150, no. 1, pp. 129–137, 2004, ISSN: 0096-3003. DOI: 10.1016/S0096-3003(03)00203-0.

[2] M. Döhler, P. Andersen, and L. Mevel, "Data merging for multi-setup operational modal analysis with data-driven SSI," in *Structural Dynamics, Volume 3*, Springer, 2011, pp. 443–452. DOI: 10.1007/978-1-4419-9834-7_42.

[3] M. Döhler and L. Mevel, "Modular subspace-based system identification from multi-setup measurements," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2951–2956, 2012. DOI: 10.1109/TAC.2012.2193711.

[4] P. R. Fraanje, "Robust and fast schemes in broadband active noise and vibration control," PhD thesis, University of Twente, Enschede, May 2004, ISBN: 90-9018002-8. [Online]. Available: http://doc.utwente.nl/41476/.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. JHU Press, 2013.

[6] L. R. J. Haverkamp, "State space identification — theory and practice," PhD thesis, Delft University of Technology, 2001.

[7] A. N. Langville and W. J. Stewart, "The kronecker product and stochastic automata networks," *Journal of Computational and Applied Mathematics*, vol. 167, no. 2, pp. 429–447, Jun. 2004, ISSN: 0377-0427. DOI: 10.1016/j.cam.2003.10.010.

[8] The MathWorks, Inc., "Envelope detection," [Online]. Available: https://nl.mathworks.com/help/dsp/examples/envelope-detection.html (visited on 06/08/2016).

[9] L. Mevel, M. Basseville, A. Benveniste, and M. Goursat, "Merging sensor data from multiple measurement set-ups for non-stationary subspace-based modal analysis," *Journal of Sound and Vibration*, vol. 249, no. 4, pp. 719–741, 2002. DOI: 10.1006/jsvi.2001.3880.

[10] L. Mevel, A. Benveniste, M. Basseville, and M. Goursat, "Blind subspace-based eigenstructure identification under nonstationary excitation using moving sensors," *IEEE Transactions on Signal Processing*, vol. 50, no. 1, pp. 41–48, 2002. DOI: 10.1109/78.972480.

[11] B. Ottersten and M. Viberg, "A subspace based instrumental variable method for state-space system identification," in *In proceedings of 10-th IFAC Symposium on System Identification*, Royal institute of technology Stockholm, Mar. 1994. DOI: 10.1.1.50.5669.

[12] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems*. Springer Science & Business Media, 1996. DOI: 10.1007/978-1-4613-0465-4.

[13] P. J. Prins, "Literature review: Control for a cryo vibration isolation platform, active vibration isolation in 3 degrees of freedom," TU Delft, Tech. Rep., Jun. 2016.

[14] M. Verhaegen, "Identification of the deterministic part of MIMO state space models given in innovations form from input-output data," *Automatica*, vol. 30, no. 1, pp. 61–74, 1994. DOI: 10.1016/0005-1098(94)90229-1.

[15] M. Verhaegen and P. Dewilde, "Subspace model identification part 1. the output-error state-space model identification class of algorithms," *International journal of Control*, vol. 56, no. 5, pp. 1187–1210, 1992. DOI: 10.1080/00207179208934363.

[16] M. Verhaegen and V. Verdult, *Filtering and system identification: A least squares approach*. Cambridge university press, 2007. DOI: 10.1017/cbo9780511618888.

[17] M. Verhaegen, V. Verdult, and N. Bergboer, *Filtering and system identification: An introduction to using Matlab software*, TU Delft, Mekelweg 2, 2628 CD, Delft, The Netherlands, Aug. 2007.

[18] Wikipedia contributors, "Proofs involving the Moore–Penrose pseudoinverse — Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proofs_involving_the_Moore%E2%80%93Penrose_pseudoinverse&oldid=720258262#Products (visited on 07/13/2016).

Glossary

List of Acronyms

CVIP 2	Cryo Vibration Isolation Platform 2
DFT	Discrete Fourier transform
DOF	Degrees of freedom
FFT	Fast Fourier transform
IMC	Internal model control
JPE	Janssen Precision Engineering
LTI	Linear time-invariant
POI	Point of interest
PO-MOESP	Past Outputs Multivariable Output-Error StatesPace
PSD	Power spectral density
SMI	Subspace model identification
SVD	Singular value decomposition
THD	Total harmonic distortion
TU Delft	Delft University of Technology
VAF	Variance accounted for

List of Symbols

$[\square]_+$	Causality operator.
$\hat{\square}$	Estimate of $\square$.
$\dot{\square}$	First order time derivative: $\dot{\square} = \frac{\mathrm{d}}{\mathrm{d}t}\square$.
$\|\square\|_\infty$	H_∞ norm operator.
$\otimes$	Kronecker product.

$\lvert\square\rvert$	Magnitude of $\square$.	
$\square^\top$	Matrix transpose.	
$\square^\dagger$	Moore-Penrose pseudo-inverse, see Definition 1 on page 55.	
$\bar{\square}$	Nominal model of the uncertain system $\square$.	
$\square^\perp$	Orthogonal complement of $\square$.	
$\square_{gh:pq}$	Range of matrix partitions defined by the upper left block $\square_{gh}$ and the lower right block $\square_{pq}$.	
$\ddot{\square}$	Second order time derivative: $\ddot{\square} = \frac{\mathrm{d}^2}{\mathrm{d}t^2}\square$.	
$\square^\sim$	Spectral factor of the uncertain system $\square$.	
$\square^{-1}$	Matrix inverse.	
$\square_0 \,/\, \square^0$	Inertial reference frame Ψ_0.	
$\mathbf{0}$	Zero matrix or vector, a subscript indicates its size if it is not clear from the context.	
$\lVert\square\rVert_2$	H_2 norm operator.	
α	Angular acceleration.	rad/s^2
$\Delta\square$	Additive uncertainty of the uncertain system $\square$.	
$\boldsymbol{\Delta}_{\ell_j,s}$	Non-circular shift matrix, see Eq. (2-63) on page 24.	
$\boldsymbol{\epsilon}$	Error sequence in the voice coil controller synthesis, see Figure 3-1 on page 44.	
$\boldsymbol{\varepsilon}$	Error sequence in the piezo actuator controller synthesis, see Figure 3-2 on page 45.	
$\boldsymbol{\Lambda}$	See Eq. (2-82) on page 30.	
$\boldsymbol{\Xi}_s^{(j)}$	Generalised state sequence of the j^{th} experiment, see Eq. (2-20) on page 15. unit	
$\boldsymbol{\Pi}^\perp_{\mathbf{U}_{s,s,N}}$	Orthogonal projection matrix of $\mathbf{U}_{s,s,N}$, see Eq. (2-14) on page 14.	
ρ	Positive real tuning parameter of the Cautious Wiener filter.	
$\boldsymbol{\Sigma}_n$	Singular value matrix a rank n truncated order singular value decomposition.	
$\boldsymbol{\Upsilon}$	See Eq. (2-64) on page 25.	
$\bar{\boldsymbol{\Upsilon}}$	See Eq. (2-59) on page 24.	
$\boldsymbol{\Upsilon}^\forall$	See Eq. (2-66) on page 25.	
Ψ	Coordinate frame.	
ω	Angular velocity.	rad/s
$\mathbf{A}$	Dynamics matrix of a state space model.	
$\tilde{\boldsymbol{\mathcal{A}}}_{\mathrm{a}}^{\mathrm{a,b}}$	Acceleration matrix describing the acceleration of frame Ψ_{a} with respect to frame Ψ_{b} expressed in frame Ψ_{a} (strapdown), see Eq. (2-80) on page 30.	
a	Linear acceleration.	m/s^2
$\boldsymbol{a}$	Floor acceleration in three degrees of freedom, see Section 2-1 on page 3.	m/s^2
$\mathbf{B}$	Input matrix of a state space model.	
$\mathbf{C}$	Output matrix of a state space model.	

$\mathbf{C}_{\text{pa}}$	Piezo actuator feedback controller.	
$\mathbf{C}_{\text{vc}}$	Voice coil feedforward controller.	
$\mathcal{C}$	See Eq. (2-82) on page 30.	
$\square_{\text{ci}}$	Co-inner factor of $\square$: $\square_{\text{co}}\square_{\text{ci}} = \square$; $\square_{\text{ci}}\square_{\text{ci}}^{\text{H}} = \mathbf{I}$.	
$\square_{\text{co}}$	Co-outer factor of $\square$: $\square_{\text{co}}\square_{\text{ci}} = \square$; $\square_{\text{co}}\square_{\text{co}}^{\text{H}} = \square\square^{\text{H}}$.	
c_j	Cosine of angle of sensor j, see page 29.	
$\mathbf{D}$	Feedthrough matrix of a state space model.	
$\text{diag}(\square, \square, \ldots)$	Block diagonal matrix.	
$\text{E}[\square]$	Expectation of $\square$ with respect to stochastic signals.	
$\bar{\text{E}}[\square]$	Expectation of $\square$ with respect to uncertain systems.	
$\mathbf{E}_{i,s,N}^{(j)}$	Hankel matrix of the noise sequence during the j^{th} experiment, see Eq. (2-7) on page 12.	
$\boldsymbol{e}$	Innovation sequence.	
$\mathbf{G}_{\text{n}}$	Noise generating model, see Section 2-1 on page 3.	
$\mathbf{G}_{\text{n},s_{\text{corr}}}$	Noise generating model not accounting for input noise correlation, see Section 2-1 on page 3.	
$\square^{\text{H}}$	Matrix Hermitian transpose.	
$\mathbf{H}$	Transfer function for Wiener filter synthesis, see Eqs. (3-3) to (3-5), (3-11) and (3-19) on pages 45, 47 and 49.	
$\mathcal{H}_{\text{a}}^{\text{b}}$	Homogeneous transformation matrix from frame Ψ_a to frame Ψ_b.	
h_{st}^{s}	Height of the pedestal bottom above the shaker table sensor plane.	m
$\mathbf{I}_n$	Identity matrix $\in \mathbb{R}^{n\times n}$.	
$\square_{\text{i}}$	Inner factor of $\square$: $\square_{\text{i}}\square_{\text{o}} = \square$; $\square_{\text{i}}^{\text{H}}\square_{\text{i}} = \mathbf{I}$.	
J	Number of experiments.	
$\mathbf{J}$	Cost gain, see Figure 3-2 on page 45.	
$\mathbf{K}$	Kalman gain, see Eq. (2-5) on page 12.	
ℓ_0	Number of reference sensor outputs.	
ℓ_j	Number of roving sensor outputs (for the j^{th} experiment).	
$\mathbf{M}$	Gain matrix to model the correlation of the input noise, see Figure 2-1 on page 4.	
$\mathcal{M}$	See Eq. (2-67) on page 25.	
m	Number of inputs.	
N	Number of columns in the Hankel matrices used for subspace identification, see e.g. Eq. (2-7) on page 12.	
$\bar{N}$	Number of of samples.	
$\mathbb{N}$	Set of natural numbers.	
n	Model order.	
$\mathcal{O}_s$	Extended observability matrix, see Eq. (2-9) on page 13.	
$\mathcal{O}_s^{\forall}$	Global extended observability matrix, see Eq. (2-23) on page 16.	
$\square_{\text{o}}$	Outer factor of $\square$: $\square_{\text{i}}\square_{\text{o}} = \square$; $\square_{\text{o}}^{\text{H}}\square_{\text{o}} = \square^{\text{H}}\square$.	

$\boldsymbol{P}_{\mathrm{a}}^{\mathrm{b}}$	Projective coordinates of the origin of frame Ψ_a expressed in frame Ψ_b: $\boldsymbol{P}_{\mathrm{a}}^{\mathrm{b}} = \begin{bmatrix} \boldsymbol{p}_{\mathrm{a}}^{\mathrm{b}\top} & 1 \end{bmatrix}^{\top}$.	
$\mathbf{P}$	Plant, see Figure 3-2 on page 45.	
$\boldsymbol{\mathcal{P}}$	Permutation matrix.	
$\boldsymbol{\mathcal{P}}_{p,s}^{\star}$	Mod-p perfect shuffle permutation, see Eq. (2-54) on page 22.	
$\boldsymbol{p}_{\mathrm{a}}^{\mathrm{b}}$	Cartesian coordinates of the origin of frame Ψ_a expressed in frame Ψ_b.	
$\mathbf{Q}$	See Eq. (2-18) on page 14..	
$\mathbf{R}$	Compressed data matrix, see Eq. (2-18) on page 14.	
$\boldsymbol{\mathcal{R}}_{\mathrm{a}}^{\mathrm{b}}$	Rotation matrix from frame Ψ_a to frame Ψ_b.	
$\boldsymbol{\mathcal{R}}_{\mathrm{s}\forall}^{\mathrm{s}}$	See Eq. (2-83) on page 30.	
$\mathbb{R}$	Set of real numbers.	
r_{s}	Centre to shaker table rim sensor distance, see Figure 2-7 on page 10.	m
r_{p}	Centre to payload rim sensor distance, see Figure 2-7 on page 10.	m
$S_{\square\square}$	(Cross) power spectral density.	
$\boldsymbol{\mathcal{S}}_s^{(j)}$	See Eq. (2-11) on page 13.	
$\square_{\mathrm{s}}$ / $\square^{\mathrm{s}}$	Shaker table sensor frame $\Psi_{\mathrm{s}} = \Psi_{\mathrm{s}1}$.	
$\square_{\mathrm{s}j}$ / $\square^{\mathrm{s}j}$	Shaker table fixed sensor frame number $\Psi_{\mathrm{s}j}$.	
$\square_{\mathrm{st}}$ / $\square^{\mathrm{st}}$	Pedestal bottom frame Ψ_{st}.	
s	Positive integer parameter, $n < s \ll N$.	
s_j	Sine of angle of sensor j, see page 29.	
$\boldsymbol{s}$	Input noise sequence, see Section 2-1 on page 3.	m/s^2
$\mathbf{T}$	Basis of a state space model.	
$\boldsymbol{\mathcal{T}}_s$	See Eq. (2-10) on page 13.	
t	Time.	s
$\mathbf{U}_{i,s,N}$	Hankel matrix of the input sequence, see Eq. (2-7) on page 12.	
$\boldsymbol{\mathcal{U}}_n$	Left singular vector matrix of a rank n truncated order singular value decomposition.	
$\boldsymbol{u}$	Input sequence.	
$\tilde{\boldsymbol{u}}$	Artificial error sequence, see Figure 3-6 on page 48.	
$\boldsymbol{\mathcal{V}}_n$	Right singular vector matrix of a rank n truncated order singular value decomposition.	
$\mathrm{vec}(\square)$	A vector constructed by stacking the columns of the matrix $\square$ on top of each other, in MATLAB notation: $\square(:)$.	
$\boldsymbol{v}$	Output noise sequence.	
$\mathbf{W}$	Wiener filter.	
$\boldsymbol{\mathcal{W}}$	Weighting matrix during the identification of the noise generating model, see Section 2-1 on page 3.	
$\boldsymbol{w}$	Analytic signal, see Section 2-1-3 on page 6.	m/s^2
$\mathbf{X}_{i,N}^{(j)}$	State sequence during the j^{th} experiment, see Eq. (2-8) on page 13.	
$\boldsymbol{x}$	State sequence.	

$\mathbf{Y}_{i,s,N}^{(k,j)}$	Hankel matrix of the output sequence at the k^{th} sensor position during the j^{th} experiment, see Eq. (2-7) on page 12.	
$\boldsymbol{y}$	Output sequence.	
$\boldsymbol{y}_{\text{s}}$	See Eq. (2-83) on page 30.	m/s^2
$\boldsymbol{y}_{\text{sensors}}$	Vector of sensor readings, see Eq. (2-83) on page 30.	m/s^2
$\mathbf{Z}_N^{(j)}$	Instrumental variable matrix of the j^{th} experiment, see Eq. (2-15) on page 14.	
$\mathbb{Z}$	Set of integers.	