Mechatronics 52 (2018) 78–89

Contents lists available at ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

Linear Control Toolbox - supporting B-splines in LPV control☆,☆☆

Maarten Verbandt*, Laurens Jacobs, Dora Turk, Taranjitsingh Singh, Jan Swevers, Goele Pipeleers

MECO Research Team, Department of Mechanical Engineering, KU Leuven; DMMS lab, Flanders Make, Leuven, Belgium

ARTICLE INFO

Keywords:
Linear Control Toolbox
Matlab
LPV
B-spline
Modeling
Identification
Control

ABSTRACT

This paper presents the Linear Control Toolbox, a freely available Matlab-based software package primarily focused on optimal feedback controller design for LTI and LPV systems. Its main goal is to make more involved controller design methodologies accessible to the broader public by providing tailored synthesis and analysis tools. The paper gives an overview of the different modules within the toolbox, the underlying data structures and the supported functionalities and algorithms. Moreover two case studies are elaborated. The autopilot of a missile serves as a comparison of the Linear Control Toolbox with its competitors and an overhead crane setup was used to demonstrate the use and experimentally validate the presented toolbox.

1. Introduction

Since the early days of control the $\mathscr{H}_\infty/\mathscr{H}_2$ framework has received a lot of attention. It reformulates the LTI controller design as an optimization problem where the objectives and constraints are based on $\mathscr{H}_\infty$ and/or $\mathscr{H}_2$ norms of closed-loop transfer functions. As a result, one is guaranteed to obtain the optimal stabilizing controller given a particular set of constraints. Also robustness requirements fit naturally in this framework. All these aspects combined make the $\mathscr{H}_\infty/\mathscr{H}_2$ framework very appealing.

An additional advantage of the $\mathscr{H}_\infty/\mathscr{H}_2$ framework is that it naturally extends to Linear Parameter Varying control. The LPV paradigm holds the middle ground between the restricted class of LTI systems on the one hand and the general class of nonlinear systems on the other: while being able to capture particular nonlinear phenomena, LPV control is supported by a similar wealth of analysis and design tools as LTI control. Moreover, LPV control has been widely validated in academia, supporting its practical value [1]. However, the absence of facilitating software tools seems to be the impeding factor to get LPV control adopted by industry. For LPV $\mathscr{H}_\infty/\mathscr{H}_2$ control, the need for dedicated software is even more emergent than for LTI. In comparison to the LTI case, optimization based LPV control faces the additional challenge of satisfying the constraints over the entire parameter domain. Various methods have been designed to handle this problem, but still a substantial effort is needed to transform the controller design into a tractable optimization problem. Some steps in the direction of user-friendly software have been made though. Matlab's Robust Control Toolbox supports LPV controller design in a very general fashion that is mainly based on the work of Apkarian and co-workers [2,3]. So-called `tunableSurface` objects capture a general parameter dependency and are used to parameterize and design an LPV controller. In case the parameter dependency is affine, the user can resort to the more convenient `hinfgs`. LPVtools, as described by Hjartarson et al. [4], is another Matlab-based package which is intended to ease the manipulation with, analysis of and design for both gridded and parameterized LPV systems. Within LPVtools a polynomial parameter dependency is employed.

This paper focuses on a new Linear Control Toolbox,[1] which is an extension to Verbandt [5]. The motivation is twofold. The main driver is to support recent developments in LPV controller design, based on B-splines as described by Hilhorst et al. [6]. This method allows parameter dependencies to be described by B-splines which are more flexible than the traditionally employed polynomials. Since B-splines encompass polynomials, traditional analysis and design methods can still be interfaced with this new toolbox. Second existing tools are primarily focused on control, tailored to a specific LPV design problem. This forces the user to switch toolboxes to for instance identify an LPV system or to design an LTI controller. On the contrary, the Linear

☆ This work benefits from KU Leuven-BOF PFV/10/002 Centre of Excellence: Optimization in Engineering (OPTEC); the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office (DYSCO); Flanders Make project ROCSIS: Robust and Optimal Control of Systems of Interacting Subsystems; and project G091514N of the Research Foundation-Flanders (FWO-Flanders). Flanders Make is the Flemish strategic research center for the manufactoring industry.

☆☆ This paper was recommended for publication by Associate Editor Dr Oomen Tom.

* Corresponding author.
E-mail address: maarten.verbandt@kuleuven.be (M. Verbandt).

[1] The Linear Control Toolbox can be found at https://github.com/meco-group/lc_toolbox.

https://doi.org/10.1016/j.mechatronics.2018.04.007
Received 23 November 2017; Received in revised form 13 April 2018; Accepted 25 April 2018

Control Toolbox supports the control engineer throughout the different steps involved in controller design both in an LTI and an LPV setting. It provides the functionality to identify a system, facilitates general manipulations with systems, enables the user to declare a control configuration and a list of specifications, automatically solves the optimal controller design problem and provides frequency and time domain validation tools.

The remainder of the paper is structured as follows. The first part focuses on the internal system modeling module, which handles general system manipulations. Next, the identification module is touched upon, followed by a detailed explanation of the controller design module. In addition, two case studies are provided. The first is concerned with the design of a missile's autopilot and compares the Linear Control Toolbox to LPVtools and the Robust Control Toolbox. The second example considers an overhead crane with varying cable length for which an LPV controller is designed and experimentally validated. The paper ends with some concluding remarks.

2. System modeling module

At the core of the presented toolbox resides a general system modeling module. This module adopts a new design paradigm which makes a clear distinction between systems and models. A system reflects a physical entity, e.g. a pump, an electrical circuit or a chemical process. All of these have inputs and outputs which are intrinsically related. This relationship is never truly known but one or more models of the same system might be available. Therefore a system can be regarded as a collection of models which all refer to the same physical entity. The advantage of this approach is that different models can be consulted when most relevant. For instance, the controller design might be based on a simplified linear model whereas a time-domain simulation of a high-fidelity nonlinear model is preferable for a validation. Using a custom modeling module also gives the opportunity to introduce new model classes such as a general nonlinear model, a gridded (LPV) model or a parameter dependent state-space model.

The following paragraphs shed a brighter light on the toolbox's understanding of systems and signals, models and parameters, and their mutual relation.

2.1. Systems and signals

Within the Linear Control Toolbox, systems and signals serve the purpose of describing a configuration, i.e. how different components interact with each other. Systems are created via the function call

```
> G = IOSystem(ni,no)
```

where `ni` and `no` denote the number of inputs and outputs. Inputs and outputs are internally stored as a `Signal` and can be accessed via:

```
> G.in, G.out
```

The user is also encouraged to use signal aliases which make the code easier to read:

```
> u = G.in, y = G.out
```

Moreover, new signals can be constructed either via the `Signal()` call or as a linear combination of other signals, for example:

```
> r = Signal(n)
> e = r - y
```

where the optional argument `n` indicates the dimension of the signal. Signals serve the purpose of connecting systems together. A connection is established by simply equating the signals with the `==` operator and creating a new system from the subsystems and the connection list:

```
> c1 = (K.in == e)
> c2 = (K.out == u)
> P = IOSystem(G,K,[c1;c2])
```

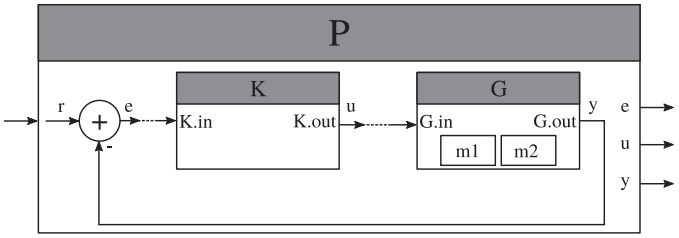

Fig. 1. Schematic of the plant constructed throughout Section 2. Systems K and G are connected to form system P with input r and outputs e, u, and y. Two models for G are available and added to the system while K remains empty.

In this case, the new system `P` is constructed from `G` and `K`, arranged in a standard error feedback configuration. A schematic representation is found in Fig. 1.

2.2. The scheduling parameter

Models provide a relation between the inputs and outputs of a system. This relation can be either time independent or depending on a parameter. To this end, the `SchedulingParameter` is introduced:

```
> p = SchedulingParameter(name,range,rate)
```

This object has three attributes: a name, a range indicating the bounds of parameter values and a range for the rate of variation, keeping track of the minimum and maximum change of the parameter value over time. Special cases include fixed parameter values (rate of variation [0, 0]) and unbounded rate of variation, $[-\infty, \infty]$. Under the hood, a `SchedulingParameter` is a B-spline based identity function. Therefore the `SchedulingParameter` supports all operations that yield piecewise polynomial functions i.e. the addition and multiplication. This allows the user to easily declare complex parameter dependencies, for example the A-matrix of a state-space model:

```
> A = [p 2, 1;p 3 - 2*p, 0]
```

2.3. Design models

Because the controller design is based on linear parametric models, the most important model class is the general `LFTmod`. It incorporates a pair of Linear Fractional Transforms applied to an underlying standard state-space model, as depicted in Fig. 2. Note that the matrices involved

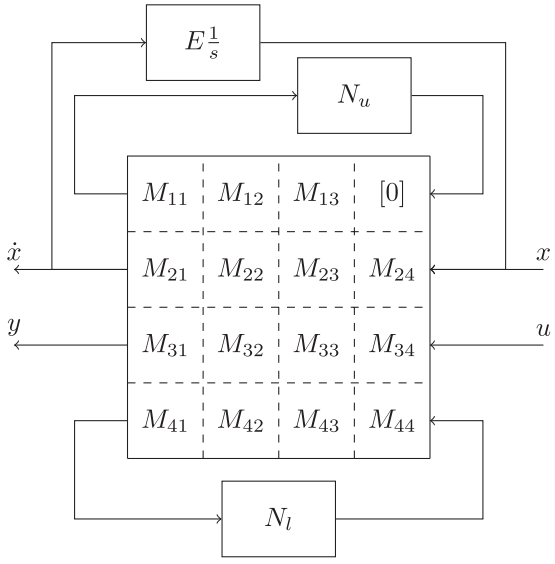

Fig. 2. Graphical representation of the `LFTmod` in the Linear Control Toolbox. Two LFTs are included to describe a more general class of models. The dynamics are closed by $E\frac{1}{s}$ so that improper models are allowed.

need not be constant, allowing linear parameter varying (LPV) models. The resulting state and output dynamics are given by:

$$F_u = N_u(I - M_{11}N_u)^{-1},\ F_l = N_l(I - M_{44}N_l)^{-1}$$

$$\begin{bmatrix} E\vec{x} \\ \vec{y} \end{bmatrix} = \left[\begin{pmatrix} M_{22} & M_{23} \\ M_{32} & M_{33} \end{pmatrix} + \begin{pmatrix} M_{24} \\ M_{34} \end{pmatrix} F_l M_{41} F_u (M_{12} \quad M_{13}) + \begin{pmatrix} M_{21} \\ M_{31} \end{pmatrix} F_u (M_{12} \quad M_{13}) + \begin{pmatrix} M_{24} \\ M_{34} \end{pmatrix} F_l (M_{42} \quad M_{43}) \right] \begin{bmatrix} \vec{x} \\ \vec{u} \end{bmatrix}$$

A corresponding constructor is provided as:

```
> mlft = LFTmod(M,Nu,Nl,E,p,Ts)
```

where `M` is a 4-by-4 cell-array and `Nu`, `Nl` and `E` are matrices of corresponding sizes. It is readily seen that providing empty feedback matrices `Nu` and `Nl` reduces the `LFTmod` to a standard descriptor state-space model. The latter is made directly available via the call:

```
> mdss = DSSmod(A,B,C,D,E,p,Ts)
```

where `A`, `B`, `C`, `D` and `E` are standard state-space matrices, `Ts` denotes the sample time in seconds and `p` is the scheduling parameter which is only required if one or more state-space matrices depend on `p`. In case `E` happens to be the unit matrix, the `DSSmod` further reduces to a standard state-space model which is constructed straightaway as:

```
> mss = SSmod(A,B,C,D,p,Ts)
```

Although the implementation relying heavily on the `LFTmod` might look over-complicated, there are two advantages in doing so. First, more complex operations such as closing a loop become very simple on models in the LFT form [7]. Second, the `LFTmod` is able to represent the complex model structures which arise naturally from an LPV controller design [8].

2.4. Validation models

For validation purposes, the toolbox provides more dedicated model classes such as nonlinear models or nonparametric models like a sampled frequency response.

To capture arbitrary nonlinear dynamics, e.g. to perform a time domain validation, the `ODEmod` is introduced. This model incorporates a general ordinary differential equation (ODE) which is driven by u and p, i.e. the input to the system and the parameter value. The state x is updated via Eq. (1). Note that x might be empty, e.g. for a saturation block. Eq. (2) returns the output, y, of the model.

$$E(x, u, p)\dot{x} = f(x, u, p) \tag{1}$$

$$y = g(x, u, p) \tag{2}$$

An `ODEmod` is constructed by:

```
> mode = ODEmod(f,g,E,p)
```

where `f`, `g` and `E` are function handles with two or three inputs, depending on whether a scheduling parameter is involved. In case these function handles do depend on a scheduling parameter, the latter is also required as an input argument of the model.

The presented toolbox also offers the possibility to work with non-parametric FRFs. The `FRDmod` class is built on top of Matlab's `frd` class, transferring all original functionality:

```
> mfrd = FRDmod(resp,freq)
```

The `Gridmod` represents a gridded implementation of an LPV model. Take for instance a series of local identification experiments which each yield an `FRDmod` linked to a particular parameter value. These are readily combined in a `Gridmod` via the function call:

```
> mgrd = Gridmod({frd1,frd2,frd3},... [pval1,pval2,pval3])
```

The main advantage is that all operations performed on the `Gridmod` are carried out on all underlying elements at once, e.g. computing an element-wise norm or plotting a series of Bode diagrams.

2.5. Adding models to systems

Once one or more models have been created, they should be linked to the corresponding system. To do so, the function `add` is used:

```
> G.add(mss,mode,mfrd)
```

Thereafter, `G.content()` is no longer empty. Though standard plotting functionality like `bode` or `step` is supported for individual models, calling these functions on systems is more powerful. This shows all responses on a single graph, facilitating the comparison between different (closed-loop) models.

3. System identification

The system identification module provides the necessary functionality to convert measured time or frequency domain data into parametric models. It interfaces the Matlab implementations [9] of the frequency domain system identification methods in detail elaborated in [10].

Although the focus currently lies on LTI identification, LPV identification is planned to be incorporated in the near future. In particular the combined global and local LPV system identification approach [11], as well as the reweighted $\ell_{2,\,1}$-norm regularization approach [12] will be supported. In the meantime, LPV models are identified via the State-space Model Interpolation of Local Estimates (SMILE) [13], a technique that interpolates LTI models with a user-specified set of basis functions of the scheduling parameter. B-spline basis functions are used here in order to comply with the B-spline based LPV controller design within the toolbox.

3.1. Excitation signals and measurement data

The first step of the system identification procedure is the generation of excitation signals. At present, the toolbox only supports the generation of multisine signals [10]. The command

```
> u = Multisine('label', 'experiment1', 'fs', 100, 'fwindow', [0.01 2], 'freqres', 0.01)
```

creates a multisine, sampled at 100 Hz, exciting a linearly spaced frequency grid in the interval [0.01, 2] Hz, with a frequency resolution of 0.01 Hz. This choice automatically determines the period of the multisine, in this case 100 s. The multisine `u` is a so-called `TimeSignal` object that contains all information that is characteristic for this specific signal. Additional name-value pairs allow the user to specify more options, e.g. the phase distribution of the signal, its amplitude spectrum, etc. `TimeSignal` objects feature useful methods; the user can for example plot the signal spectrum by

```
> plotSpectrum(u, 'dft')
```

The actual signal content (one period of the multisine in this case) is obtained by

```
> [s,t] = signal(u)
```

Once the excitation has been applied, the experimental data is imported in the toolbox. This is done by wrapping the measurements into `TDMeasurementData` objects. In case of a system with two inputs and two outputs, a `TDMeasurementData` object is constructed as follows:

```
> meas1 = TDMeasurementData('label', 'meas1',
```

```
'excitation', [u1 u2], 'data', [x1 x2 y1 y2], 'data-
labels', {'x1', 'x2', 'y1', 'y2'}, 'periodic', true)
```

where `u1` and `u2` are objects of the aforementioned class `TimeSignal` representing the first and the second input. Similar to `TimeSignal`, the toolbox supports several relevant operations on `TDMeasurementData` objects. For example, to avoid the effect of transients in the FRF estimation (Section 3.2), one may decide to keep the last 7 periods of the measurements:

```
> meas1 = clip(meas1, 'lastnper', 7)
```

Complementary to `TDMeasurementData`, the `FDMeasurementData` class handles nonparametric frequency response function (FRF) measurements. FRF measurements can either be loaded directly into Matlab or can be calculated based on time domain data from a `TDMeasurementData` object. The latter is discussed in the following paragraph.

3.2. Nonparametric frequency response function estimation

The toolbox supports two methods to estimate the system's FRF based on multiple measurements obtained with different random multisine realizations: the robust detection algorithm for nonlinearities [10, Section 4.3.1] (`Robust_NL_Anal` [9]) intended for SISO systems and the robust local polynomial method [10, Section 7.2.2] (`RobustLocalPolyAnal` [9]) for MIMO systems. These methods combine measurements from different random multisine excitations to average out both the effects of nonlinear distortions and measurement noise. They yield an FRF estimate, also referred to as the best linear approximation, the sample total variance (contributions of measurement noise and nonlinear distortions) and the sample noise variance.

The nonparametric FRF estimate based on e.g. four different measurements is obtained by calling the routine `nonpar_ident` with the arguments as shown below:

```
> IOlabels.input = {'x1', 'x2'};
> IOlabels.output = {'y1', 'y2'};
> FRF = nonpar_ident(meas1, meas2, meas3, meas4,
  IOlabels, 'RobustLocalPolyAnal');
```

where `'x1'`, `'x2'`, `'y1'` and `'y2'` refer to the datalabels provided to the `TDMeasurementData` objects. The result, `FRF`, is a `FDMeasurementData` object.

3.3. Parametric LTI model identification

The nonparametric FRF of the system is further used to estimate a parametric model. For the time being, the toolbox comes with the routine `MIMO_ML`, a maximum likelihood identification algorithm [10], `MIMO_NLS`, a nonlinear least-squares fitting algorithm allowing an arbitrary frequency weighting `weight`, and also interfaces Matlab's routine `tfest`. These methods estimate common denominator transfer function models. The function `param_ident` parses the data for each of these routines and is called as shown below:

```
> settings = struct('denh', 2, 'denl', 0, 'numh', 2,
  'numl', 0, 'W', weight, 'Ts', 0.01)
> model = param_ident('data', FRF, 'method',
  'MIMO_NLS', "settings', settings)
```

where `denh` (`numh`) and `denl` (`numl`) in `settings` represent the highest and lowest degree of the denominator (numerator) of the underlying transfer functions, respectively. `numh` and `numl` are $n_o \times n_i$ matrices, with n_o and n_i the number of system outputs and inputs, respectively.

3.4. LTI state-space model interpolation

To obtain an LPV model the aforementioned procedure is repeated for different fixed values of the scheduling parameter yielding a set of parametric LTI models. Subsequently, an LPV state-space model is obtained in two steps. First, the identified LTI models are packed in a `Gridmod`:

```
> mgrd = Gridmod(models,schGrid)
```

where `models` is a cell array of local LTI models and `schGrid` is the set of corresponding values of the scheduling parameter.

Second, an LPV state-space model is obtained from the gridded model through an interpolation carried out by the SMILE technique [13] and a user defined basis:

```
> basis = BSplineBasis(range,degree,knots)
> mlpv_interp = SSmod(mgrd,basis,schParam)
```

4. Controller design

Since one of the goals of the presented toolbox is to facilitate the design of optimal controllers, another key module is the controller design module. The $\mathscr{H}_\infty/\mathscr{H}_2$ design paradigm is introduced first, followed by the formulation and the solution of a standard control problem using the toolbox.

4.1. The $\mathscr{H}_\infty/\mathscr{H}_2$ paradigm

The idea behind the $\mathscr{H}_\infty/\mathscr{H}_2$ paradigm [14] is to formulate the controller design procedure as an optimization problem. Objectives and constraints are imposed directly on the closed-loop transfer functions by means of $\mathscr{H}_\infty$ and/or $\mathscr{H}_2$ norms. By doing so, one is guaranteed to obtain the optimal stabilizing controller given a particular set of constraints. Moreover, it is well known that standard objectives and constraints such as bandwidth, robustness or noise suppression can be expressed readily within this framework.

Eq. (3)displays the standard $\mathscr{H}_\infty/\mathscr{H}_2$ optimization problem. $\mathscr{C}_i$ and $\mathscr{C}_j$ denote the closed-loop transfer functions of interest in the objective respectively the constraints. Frequency dependent weights [15] $\mathscr{W}_i$, $\mathscr{V}_i$, $\mathscr{W}_j$ and $\mathscr{V}_j$ are usually added to balance the importance of different frequencies. p_i and p_j indicate which system norm is used for each channel and can either be 2 of ∞. α_i allows a trade-off between multiple objectives.

$$\begin{aligned} &\underset{K}{\text{minimize}} && \sum_i \alpha_i \left\| \mathscr{W}_i \mathscr{C}_i \mathscr{V}_i \right\|_{p_i} \\ &\text{subject to} && \left\| \mathscr{W}_j \mathscr{C}_j \mathscr{V}_j \right\|_{p_j} \leqslant 1 \end{aligned} \tag{3}$$

Most solvers reformulate optimization problem (3) in terms of linear matrix inequalities (LMIs) [16]. By choosing a single Lyapunov matrix which applies to all of the considered channels, it is possible to render the design problem convex. This yields an efficient solution strategy which naturally extends to the design of LPV controllers. In that case, the LMIs become parameter dependent and have to be relaxed to obtain a tractable formulation. Traditionally Pólya's method [17] or Sum-of-Squares [18] are employed, but more recently it has been shown that B-spline relaxations outperform these methods in many cases [6]. It is shown that the parameter dependent LMIs are guaranteed to hold over the whole parameter domain by putting constraints on the coefficients F_i, of the basis functions $\mathscr{B}_i$, as described by Eq. (4).

$$F(p) = \sum_{i=1}^{N} \mathscr{B}_i(p) F_i \prec 0, \;\; \forall\, p \in P \;\Leftarrow\; F_i \prec 0 \tag{4}$$

Moreover, B-splines are capable of describing non-hyperrectangular parameter domains. The aforementioned reasons encouraged the use of

B-splines to capture parameter dependencies and to solve the LPV optimal controller design problem.

4.2. Channels, norms and weights

Though optimization problem (3) might still be comprehensible to a control practitioner, the actual implementation is not. Two main difficulties arise. First, the control engineer has to choose a solver implementation which fits the problem at hand. This task is not obvious as it depends on several criteria, for instance: Does the problem involve hard constraints? Is the problem formulated with $\mathscr{H}_\infty$ norms only? Is the problem single- or multi-objective? The second hurdle is providing the right arguments to the routine. Not only do these arguments vary from routine to routine, but also the construction of the usually required augmented plant itself might cause problems.

In order to simplify the problem formulation, the Linear Control Toolbox offers the `Channel` and `Norm` classes. A `Channel` is an abstract representation of a transfer function with input `in` and output `out` and is constructed as:

```
> C = Channel(in,out,name)
```

Multidimensional `in` and `out` are allowed, yielding a MIMO channel.

The second ingredient is the so-called weighting function. Although weighting functions can in principle be any transfer function, they often boil down to standard filters. Therefore, the presented toolbox offers the `Weight` class which implements these standards. Moreover, in the SISO case, the inverse of the weights can be regarded as upper bounds to the closed-loop transfer functions. Therefore `Weight` employs an inverse definition allowing the user to interpret the specified transfer functions as constraints for the optimization rather than frequency dependent weights for the closed-loop transfer functions. The simplest possible weight is a constant, constraining the peak of a transfer function:

```
> W = Weight.DC(value)
```

Since control engineers typically prefer a logarithmic scale, `value` is by default interpreted in [dB]. In case `value` is to be interpreted as is, the argument `'linear'` should be supplied. Due to the inverse definition, the actual magnitude of `W` will be `-value` [dB].

```
> W = Weight.LF(fco,ord,lfgain)
```

puts emphasis on lower frequencies and is therefore suited to shape a sensitivity transfer function. At low frequencies, its magnitude approaches `-lfgain` [dB]. `ord` poles are inserted so that the cross-over frequency becomes `fco`. Again, natural parameters are employed to shape the weight. Its counterpart `Weight.HF` also exists, amplifying high frequencies which is for instance applicable to ensure noise suppression.

In combination with a weighting function, the `Channel` yields a `Norm` object. The latter is then used to formulate the optimization problem in a natural way (cfr. Eq. (3)). A `Norm` is readily constructed via:

```
> N = Norm(W,C,p)
```

where `W` and `C` are the weight and channel respectively. In this case the output of `C` is weighted by `W`. To have weighted inputs which is especially relevant in a MIMO setting, the arguments `W` and `C` are swapped. The third argument `p` specifies which norm is being employed and can take two values: `2` or `inf`. In case `p` is omitted, the default `inf` is employed. In that case, the shorthand notation

```
> N = W*C
```

is also available.

4.3. Obtaining a solution

Once the control loop and design objectives have been formulated, they are dispatched to the appropriate solver. This is done by calling the `solve` function on the constructed closed-loop system. A schematic representation of the controller design is shown in Fig. 3.

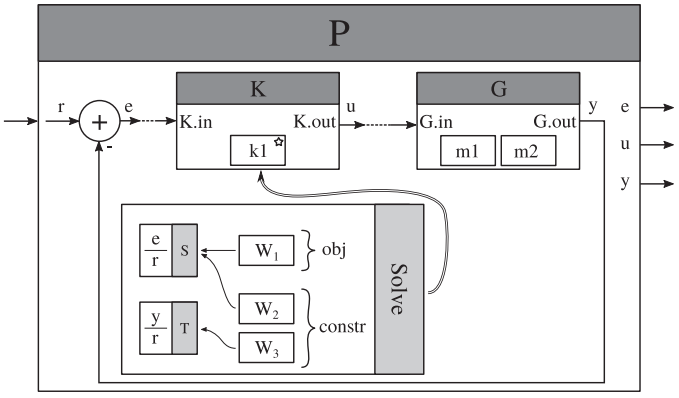

Fig. 3. Schematic of the controller design. Channels S and T are constructed first. Together with the appropriate weights, they yield several norm objects which form the objective and constraints for the optimization procedure. The resulting controller k1 is automatically added to the optimized system, K.

```
> S = Channel(e,r,'S'), T = Channel(y,r,'T')
2. obj = W1*S
> constr = [W2*S<=1,W3*T<=1]
> [P,k1,info] = P.solve(obj,constr,K,opts)
```

`obj` and `constr` are the objective and constraints which are readily constructed from the appropriate `Norm`s. The argument `K` reflects the optimization variable. In case the optimization succeeds, the solution, `k1` is automatically added to the optimization variable, `K`. Along with the actual solution, `solve` produces a report on the internal optimization process, `info`. This allows the user to assess the quality of the solution. The easiest way (only for SISO controllers) is to call:

```
> bodemag(info)
```

showing the closed-loop frequency responses along with the constraints. For MIMO designs, `sigma` is provided, which plots the weighted closed-loop singular values.

Since controller design is mostly an iterative process, the Linear Control Toolbox supports multiple designs in a straightforward manner. Here too the system-model approach proves itself useful. Subsequent `solve` calls with different objectives and constraints will come up with different control models, which are all added to the controller system `K`. Therefore, the closed-loop system `P` will carry multiple models which are readily compared.

Another more elaborate case would be the design of multiple decentralized controllers. Contrary to Matlab's `hinfstruct`, the Linear Control Toolbox does not support a single-step design procedure for this case. However, the intuitive design procedure of sequential loop-closure is easily carried out. First, the inner controller is designed:

```
> [CL,~,info1] = CL.solve(obj_in, constr_in, K_in)
```

Once `K_in`

has an implementation, the inner control loop is fully known. Thereafter, the outer loop can be shaped, which is done in a very similar way:

```
> [CL,~, info2] = CL.solve(obj_out, constr_out, K_out)
```

This yields a control law for `K_out` which is the second part of the decentralized controller. This shows that sequentially calling `solve`, each time indicating another system in the decentralized control structure, gradually constructs the overall controller.

5. Example 1: a comparative case study

This section investigates the advantages and drawbacks of the Linear Control Toolbox compared to Matlab's Robust Control Toolbox [19] and LPVtools [4]. The comparison is based on a case study described in [20] which concerns the control of a missile. The details on

the model and the design objectives are followed by a discussion on the problem formulation in each of the toolboxes. Finally an assessment of the underlying methodologies for the controller synthesis is done, based on the quality of the solution as well as the speed at which it is obtained.

5.1. Model and design objectives

The controlled system is chosen to be the missile model described in [20] which is described by Eq. (5). The system consists of one control input, δ_m, which represents the fin deflection. The measured outputs are the normalized vertical acceleration, α_{zv}, and the missile's pitch rate, q. Moreover, the missile's dynamics change under influence of two aerodynamical coefficients, $Z_\alpha \in [0.5\ 4]$ and $M_\alpha \in [0\ 105]$, which are the scheduling parameters for the LPV controller design.

$$\begin{bmatrix}\dot{\alpha}\\ \dot{q}\end{bmatrix} = \begin{bmatrix}-Z_\alpha & 1\\ -M_\alpha & 0\end{bmatrix}\begin{bmatrix}\alpha\\ q\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}\delta_m$$
$$\begin{bmatrix}\alpha_{zv}\\ q\end{bmatrix} = \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\alpha\\ q\end{bmatrix} \tag{5}$$

The control configuration is depicted in Fig. 4. The normalized vertical acceleration's tracking error e is fed back together with the missile's pitch rate q. Based on these measurements, the control law computes the fin deflection. The goal is to achieve accurate tracking for the normalized vertical acceleration while taking into account robustness requirements. Therefore the design objective, given by (6), consists of two parts. Adequate reference tracking is ensured by taking S: $r \to e = \alpha_{zv} - r$ into account. The second term in the objective encourages robustness since the latter is guaranteed if $\|W_2 KS\| \leq 1$.

$$\text{minimize} \left\| \begin{matrix} W_1 S \\ W_2 KS \end{matrix} \right\|_\infty \tag{6}$$

with

$$W_1 = \frac{2.01}{s + 0.201}$$
$$W_2 = \frac{9.678s^3 + 0.029s^2}{s^3 + 1.203e4s^2 + 1.136e7s + 1.066e10} \tag{7}$$

5.2. Constructing an LPV model

The first step in the controller design is to enter the missile's LPV model in the different toolboxes (see code Example 1). LPVtools and the Linear Control Toolbox employ a very similar approach when it comes to declaring an LPV model. Each of the toolboxes provides a parameter object: LPVtools comes with `tvreal` and `pgrid` whereas the Linear Control Toolbox offers `SchedulingParameter`. This approach makes it very intuitive to declare for instance a parameter varying state-space matrix. Consequently, the resulting code resembles the model description in (5) which is helpful to the user. Matlab's Robust Control Toolbox employs a very different strategy inspired by the polytopic nature of the supported models. The vertices of the polytope are defined as a series of LTI models which combined with a description of the parameter domain result in a polytopic system. This approach requires more care as the order of vertices has to correspond to the list of LTI models. Also, the resulting code shows less resemblance to the model description (5) which might be perceived by the user as more challenging.

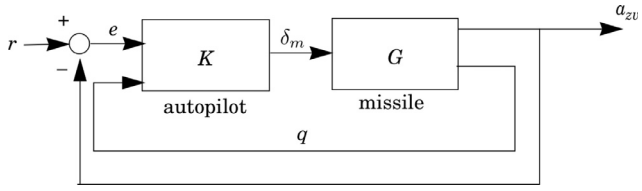

Fig. 4. Control configuration for the vertical acceleration control of a missile. The control output δ_m is based on the tracking error e and the missile's pitch rate q as an additional measurement. (adapted from [20]).

5.3. Declaring the control configuration

To proclaim the control configuration depicted in Fig. 4, each of the discussed toolboxes provides its own signal-based method. The Linear Control Toolbox uses a syntax as discussed in Section 2.1 and distinguishes itself from the other toolboxes by using explicit `Signal` objects and `IOSystem` objects which carry input and output signals. LPVtools uses `sysic` as a connection front-end which is a string-based approach. The Robust Control Toolbox offers similar functionality through `sconnect`, also parsing a series of strings to construct the interconnected system. Code Example 2 shows the construction of the control configuration within the different toolboxes.

Although a comparison of these different implementations depends heavily on the user's preferences, it is tried to make it as objective as possible. The first criterion is the readability of the produced code. In that respect the Linear Control Toolbox and LPVtools would surpass the Robust Control Toolbox since the former rely on additional variables to declare the plant. The ability to supply convenient names for the signals also adds to the readability of the code. This feature is present in both the Robust Control Toolbox and the Linear Control Toolbox which in that regard outperform LPVtools. Moreover, the Linear Control Toolbox and the Robust Control Toolbox also include the controller in the formulation of the control configuration which feels more natural than constructing an open loop to which the controller still needs to be connected, as is the case with LPVtools. On the other hand, LPVtools and the Robust Control Toolbox impose a more structured declaration than the Linear Control Toolbox which helps preventing unconnected models or unresolved connections.

When it comes to expressing the control objectives, it is usually required to extend the control configuration with appropriate weights to obtain the augmented plant. Therefore, these weights are already included in the control configuration `Paug` when using LPVtools or the Robust Control Toolbox. Contrary to its alternatives, the Linear Control Toolbox provides a higher level of abstraction to formulate the optimization problem as described by Section 4.2. Based on this optimization problem, the augmented plant is constructed behind the scenes. This allows the user to declare simply the actual control configuration rather than the augmented plant which is merely a mathematical construct.

Generally speaking, the Linear Control Toolbox is very much oriented to the user's comfort, mainly improving the readability of the code. The Robust Control Toolbox sits at the other end of the spectrum, usually resulting in compact code intended for expert users. LPVtools holds the middle ground providing tools to simplify the controller design to some extent.

5.4. Solving the control problem

The missile control problem is now passed to the solvers in the different toolboxes in order to compare the quality of the solution as well as the speed at which it is obtained. However, when passing the problem to LPVtools, an error is thrown related to a rank deficient D21 matrix in the augmented plant. Following the instructions provided by Megretski [21], a regularization input was added to the control configuration leading to a successful solution. The Linear Control Toolbox's solver was invoked with the `var_deg = 0` resulting in a constant Lyapunov matrix. LPVtools' `lpvsyn` was executed with the `'MinGamma'` flag to avoid a relaxation of the optimal value which would lead to an unfair comparison. Because LPVtools offers two parameter representations, `tvreal` and `pgrid`, both solutions are presented.

The results are gathered in Table 1. In terms of optimality, the

```
% Linear Control Toolbox
Z = SchedulingParameter('Z',[0.5 4]);
M = SchedulingParameter('M',[0 105]);
G = SSmod([-Z 1;-M 0],[0;1],...
    [-1 0;0 1],[0;0]{Z,M});

% LPVtools
Z = tvreal('Z',[0.5 4]); % or pgrid
M = tvreal('M',[0 105]); % or pgrid
G = ss([-Z 1;-M 0],[0;1],...
    [-1 0;0 1],[0;0]);

% Robust Control Toolbox
pv = pvec('box',[0.5 4; 0 105]);
s0 = ltisys([0 1;0 0],[0;1],[-1 0;0 1],[0;0]);
sZ = ltisys([-1 0;0 0],[0;0],zeros(2),[0;0],0);
sM = ltisys([0 0;-1 0],[0;0],zeros(2),[0;0],0);
G = psys(pv,[s0 sZ sM]);
```

Code example 1: Comparison of the Linear Control Toolbox, LPVtools and the Robust Control Toolbox when constructing the missile's LPV model.

Code example 1. Comparison of the Linear Control Toolbox, LPVtools and the Robust Control Toolbox when constructing the missile's LPV model.

```
% Linear Control Toolbox
Gsys = IOSystem(G);
Ksys = IOSystem(2,1);
r = Signal; u = K.out;
a = G.out(1); q = G.out(2);
e = r - a;
conn = [G.in == u; K.in == [e;q]];
P = IOSystem(Gsys,Ksys,conn);

obj = [WS*(e/r);WKS*(u/r)];
constr = [];
[P,K] = P.solve(obj,constr);

% LPVtools
systemnames = 'G WS WKS';
inputvar = '[r{1}; u{1}]';
outputvar = '[WS ; WKS ; r-G(1) ; G(2)]';
input_to_G = '[u]';
input_to_WS = '[r-G(1)]';
input_to_WKS = '[u]';
cleaupsysic = 'yes';
Paug = sysic;
[K,gam] = lpvsyn(Paug,2,1);

% Robust Control Toolbox
Paug = sconnect('r','WS;WKS',...
        'K:e=r-G(1);G(2)','G:K',G,...
        'WS:e',WS,'WKS:K',WKS);
[gam,K] = hinfgs(Paug,[2,1]);
```

Code example 2: Comparison of the Linear Control Toolbox, LPVtools and the Robust Control Toolbox when constructing the control configuration.

Code example 2. Comparison of the Linear Control Toolbox, LPVtools and the Robust Control Toolbox when constructing the control configuration.

Table 1
Comparison of the optimality and solver time of the Robust Control Toolbox, LPVtools and the Linear Control Toolbox.

	RCtoolbox	LPVtools -tvreal	LPVtools -pgrid	LCtoolbox
$\|\cdot\|_\infty$	0.2054	0.2778	0.6490	0.2021
t [s]	0.73	0.18	1.8612	17.23 (0.96)

Linear Control Toolbox and the Robust Control Toolbox show comparable performance whereas LPVtools lags behind with both solution strategies. This is also clearly visible from Fig. 5, particularly from the sensitivity: the green and yellow curves show a lower bandwidth than the red and blue curves, which implies that LPVtools reaches a lower degree of optimality. This also shows in the step response subject to a spiral parameter trajectory (8) depicted in Fig. 6. The two controllers designed with LPVtools are slower than the other two responses.

$$\begin{cases} Z(t) = 2.25 + 1.70\exp(-4t)\cos(100t) \\ M(t) = 50 + 49\exp(-4t)\sin(100t) \end{cases} \tag{8}$$

The opposite holds when comparing the solver times. Here LPVtools as well as the Robust Control Toolbox outperform the Linear Control Toolbox. Moreover the Linear Control Toolbox's solver is an order of

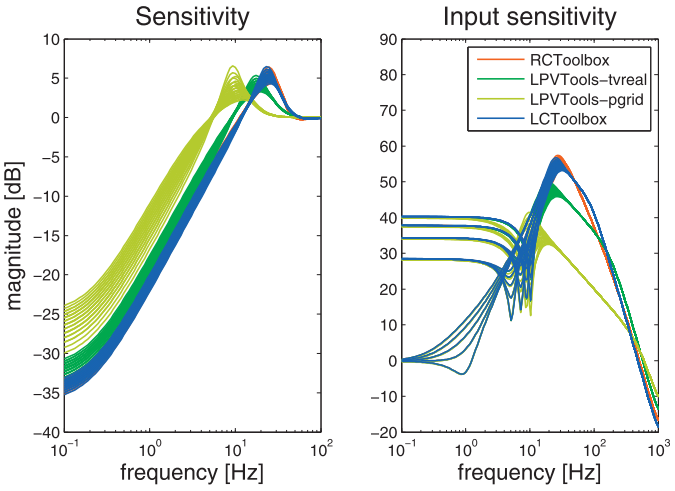

Fig. 5. Comparison of the closed-loop transfer functions within the three controller design toolboxes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

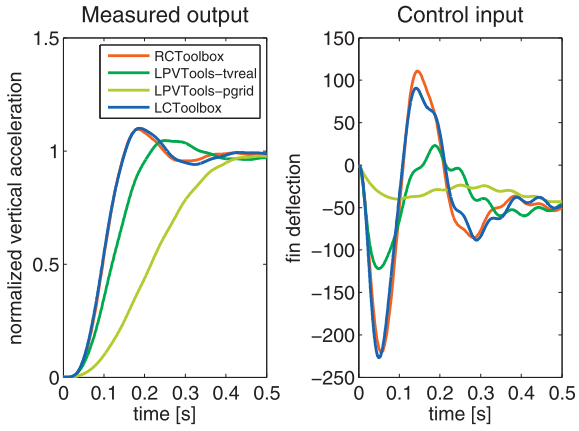

Fig. 6. Step response of the missile's autopilot on a spiral parameter trajectory (8).

magnitude slower than its competitors. However the time it takes to solve the underlying SDP is only 0.96 s which is more in line with LPVtools and the Robust Control Toolbox. The 16 s overhead is caused by the controller synthesis tools being based on the 'OptiSpline' toolbox [22]. This toolbox eases the manipulation of splines at the expense of an extra cost to export the underlying SPD. A direct implementation of the LMIs would put the B-spline based method amongst its competitors.

Although this case study would suggest there are only minor differences between the different toolboxes regarding the quality of the solution, it should be noted that the case study was chosen so that all toolboxes were able to solve the same problem. Both LPVtools and the Linear Control Toolbox are capable of solving a broader variety of problems than the Robust Control Toolbox. They can handle polynomial and B-spline parameterized models respectively and deal with constrained parameter variations explicitly.

6. Example 2: experimental validation on an overhead crane

This section demonstrates the use of the Linear Control Toolbox when designing an LPV controller for an overhead crane system. The first part covers the physical modeling of the system resulting in a nonlinear model. This model is well suited for validation but not for controller design. This is why a second model is derived, based on a series of identification experiments. These result in local LTI models which are interpolated to obtain an LPV model. Based on the latter, a parameter varying controller is designed and validated both in simulation and experimentally.

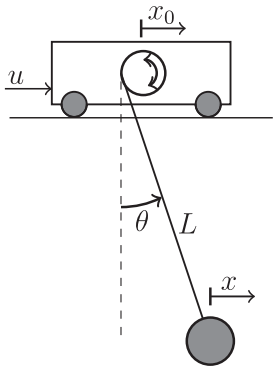

Fig. 7. Schematic of the considered overhead crane. The cart is driven by the velocity reference u. x_0, θ and L are measured.

6.1. Modeling

A schematic of the considered mechatronic system is depicted in Fig. 7. The base of the system consists of a velocity controlled trolley with time constant τ and reference u. The encoder on the trolley's track provides a measurement of x_0. The trolley is equipped with a hoisting mechanism which allows the system to vary the cable length L, which is also being measured. Moreover, L is constrained between 0.3 m and 0.8 m with a maximum rate of variation of 10 cm/s. Another sensor on the hoisting mechanism measures the load angle θ. These measurements are combined to obtain a measurement for the output of interest, x. Eq. (9) shows the governing dynamics, adapted from Debrouwere et al. [23]. The model parameter τ is approximately 0.01 s and the damping coefficient c is set to 0 m/s.

$$\begin{cases} \ddot{x}_0 &= -\tau^{-1}(\dot{x}_0 - u) \\ \ddot{\theta} &= L^{-1}(\tau^{-1}(\dot{x}_0 - u)\cos\theta - g\sin\theta - c\dot{\theta}) \\ x &= x_0 + L\sin\theta \end{cases} \tag{9}$$

Code example 3describes the declaration of the crane. First a SISO system, `crane`, is constructed. Next, the scheduling parameter `L` is introduced followed by a declaration of the nonlinear equations of motion. As a last step, the nonlinear model `nl` is added to the system `crane`.

6.2. Identification of an LPV model

Because the previously derived nonlinear model is not suited for controller design, an identification experiment is set up to derive several LTI models for a set of fixed cable lengths. Code example 4 describes how the toolbox creates an excitation signal and how it is exported to drive the actual setup. First the experimental settings are defined: the sample frequency `fs`, the number of samples of the periodic excitation signal `N`, the frequency resolution `df` and the range of excited frequencies `frange` (all frequencies specified in [Hz]). Based on these settings, an excitation signal is generated. A full random phase multisine with amplitude 0.05 m/s is opted for. This signal is then written to a text file using Matlab's `dlmwrite`, which in turn is fed to the overhead crane. Once an experiment has been carried out, the data is imported in the Linear Control Toolbox. `TDMeasurementData` is provided as the standard time domain data container and is constructed as described by Code example 5. First, the control value `u` and the measured output `y` are read from the data file 'exp_50.nc', created during the experiment. Next they are packed together with the originally constructed excitation signal in the `TDMeasurementData` object. Moreover, the signals are labeled 'input' and 'loadpos' and the data is

```
% constructing the crane
Crane = IOSystem(1,1);

% scheduling parameter L
range = [0.3,0.8];
rate = [-0.1,0.1];
L = SchedulingParameter('L',range,rate);

% nonlinear model
f = @(x,u,p) [x(2);(-x(2)+u)/tau;x(4);...
((x(2)-u)*cos(x(3))/tau-g*sin(x(3))-c*x(4))/p];
g = @(x,u,p) [x(1) + sin(x(3))*p];
nl = ODEmod(f,g,L,[1,1,4]);
nl.name = 'nonlin';
Crane.add(nl);
```

Code example 3: Declaration of the crane system and construction of the nonlinear model.

Code example 3. Declaration of the crane system and construction of the nonlinear model.

```
% identification settings
fs = 50; N = 4096; df = fs/N; frange = [df,5];
L = 50; %[cm] adapted for each experiment

% exication signals
exc = 0.05*Multisine('label','musin',...
  'fs',fs,'fwindow',frange,'freqres',df);
dlmwrite(sprintf('excitation_%d',L),...
  exc.signal_.Value);
```

Code example 4: Code necessary to generate and export a multisine excitation signal for the overhead crane setup.

Code example 4. Code necessary to generate and export a multisine excitation signal for the overhead crane setup.

marked as being periodic. In order to estimate the LTI models for different cable lengths, a two-step approach is used. This is described by Code example 6. In the first step, the measured data is fed to `nonpar_ident` which estimates a nonparametric FRF. The input and output label indicate which measurements are to be used and `'Robust_NL_anal'` indicates the employed method. Note that `clip` is used to only retain the last three periods of the measured response thereby suppressing the influence of the transient on the resulting frequency response. The nonparametric estimates for various cable lengths are shown on Fig. 8. It is observed that the identification results degrade towards higher frequencies. However, from a control point of view, it is mainly the resonance frequency that needs a decent estimate which justifies continuing with these models. The second step involves the estimation of a parametric model. To this end, `param_ident` is invoked on `npmod_50` using the method `'MIMO_NLS'`.

The previously described sequence of creating an excitation signal, loading the measured data and estimating a model is repeated for cable lengths from 0.3 m to 0.8 m in steps of 0.05 m. The different LTI models are combined in one `Gridmod`, as it serves naturally as a sampled version of an underlying LPV model. Given a B-spline basis and a scheduling parameter, the gridded model is readily transformed into an LPV state-space model through interpolation. This LPV model is then added to the system `crane`. This is described in Code example 7. Fig. 8 shows the bode diagrams of the different models involved: the fixed-length FRFs, the fixed-length LTI models and the varying length LPV model.

6.3. Controller design

Designing a controller involves two steps: proclaiming the control configuration and stating the design specifications. The presented toolbox facilitates both by offering a suitable syntax.

First the configuration, depicted in Fig. 9, is built. The corresponding code is found in Code example 8. Since the system `crane`, has already been declared, the only system still needed is the controller, `K`. After introducing the reference signal `r`, aliases are provided for the measured output of the crane, `y`, the control signal, `u` and the error signal, `e`, which improves the readability. Finally, the set of connections is passed on to a new system together with the subsystems involved, resulting in the desired configuration. Note that at this point, the closed-loop transfer functions are not accessible because the controller `K` does not yet contain any models.

Second, the list of specifications is provided, as shown in Code example 9. Two requirements are imposed: the controller's high frequency roll-off should be -20 dB per decade and the bandwidth should be at least 0.15 Hz. The transfer functions of interest are r → e and r → u, also referred to as the sensitivity (`S`) and the input sensitivity (`U`). Roll-off is guaranteed by constraining the weighted input sensitivity. The weight is designed so that high frequencies become dominant in the $\mathscr{H}_\infty$ criterion, effectively suppressing them. By choosing a first order weight, the requirement of high frequency controller roll-off is met. Similarly, the sensitivity's low frequencies are amplified by `Ws` in order to obtain better tracking behavior. The cross-over frequency of `Ws` is chosen 0.15 Hz to satisfy the design requirement. The remaining

```
u = ncread('exp_50.nc',...
      'controller.ControlValues.0');
y = ncread('exp_50.nc',...
      'plant.Measurements.0');

% pack data in a TDMeasurementData object
data = TDMeasurementData('label','exp',...
  'excitation',exc,'data',[u,y],...
  'datalabels',{'input','loadpos'},...
  'periodic',true);
```

Code example 5: Code necessary to generate and export a multisine excitation signal for the overhead crane setup.

Code example 5. Code necessary to generate and export a multisine excitation signal for the overhead crane setup.

```
% nonparametric estimation
labels.input = 'input';
labels.output = 'loadpos';
npmod_50 = nonpar_ident(data{:},labels,
                  'Robust_NL_anal');

% parametric estimation
set = struct('Ah',3,'Al',1,...
             'Bh',1,'Bl',0,'Ts',0);
lmod_50 = param_ident('data',npmod_50,...
    'method','MIMO_NLS','settings',set);
```

Code example 6: Estimation of a fixed-length LTI models for the overhead crane.

Code example 6. Estimation of a fixed-length LTI models for the overhead crane.

```
% generation of local LTI models
Lgrid = 0.3:0.05:0.8;
lmod = {lmod_30,lmod_35,...,lmod_75,lmod_80};

% LTI models to Gridmod
gmod = Gridmod(lmod,{'L',Lgrid});

% Gridmod to LPV state-space
basis = BSplineBasis([0.3,0.8],2,3);
lpv = SSmod(gmod,basis,L);
lpv.name = 'lpv';
crane.add(lpv);
```

Code example 7: Interpolation of local LTI models to obtain an LPV model.

Code example 7. Interpolation of local LTI models to obtain an LPV model.

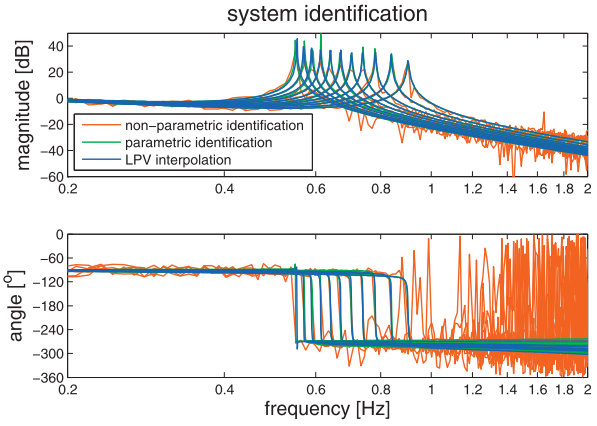

Fig. 8. Bode diagram of the different models involved in the identification process. In blue the estimated nonparametric FRF for fixed cable lengths, in green the corresponding LTI models and in red the evaluation of the interpolated LPV model at the corresponding cable lengths. The B-spline's four degrees of freedom are used to optimally interpolate the 11 local LTI models. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

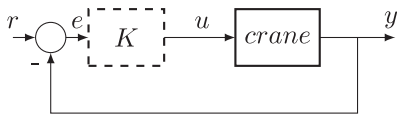

Fig. 9. Schematic of the control configuration of the overhead crane.

freedom is used to maximize the damping of the closed loop, i.e. penalizing `Ms*S`. By calling `solve` on the closed loop `P`, along with the design specifications, the optimal controller is computed and automatically added to system `K`.

```
K = IOSystem(1,1);

r = Signal();
y = crane.out; u = K.out;
e = r - y;

conn = [K.in == e; crane.in == u];
P = IOSystem(crane,K,conn);
```

Code example 8: Declaration of the control configuration in the Linear Control Toolbox.

Code example 8. Declaration of the control configuration in the Linear Control Toolbox.

```
S = Channel(e/r,'Sensitivity');
U = Channel(u/r,'Input Sens');
T = Channel(y/r,'Compl. Sens');

Ws = Weight.LF(0.15,2,-40);
Ms = Weight.DC(10);
Wu = Weight.HF(300,1,-20);

[P,C,sol] = P.solve(Ms*S,[Ws*S<=1,Wu*U<=1]);
```

Code example 9: Stating the design specifications and computing the optimal controller.

Code example 9. Stating the design specifications and computing the optimal controller.

6.4. Validation

The Linear Control Toolbox also offers some validation tools, allowing the user to critically assess the design.

Because the controller design was based on an $\mathscr{H}_\infty$ criterion, one can check how the closed-loop transfer functions relate to the imposed constraints. By simply calling `bodemag(sol,S,U,T)`, the closed-loop transfer functions are shown along the with weights, as depicted in Fig. 10. It is clearly visible that all constraints are met: the closed-loop

```
% time domain validation
ref = @(t) pfun(t);
l = @(t) lfun(t);
[out,t] = sim(P(y,r),ref,l,30);
```

Code example 10: Validation of the LPV controller design in time domain.

Code example 10. Validation of the LPV controller design in time domain.

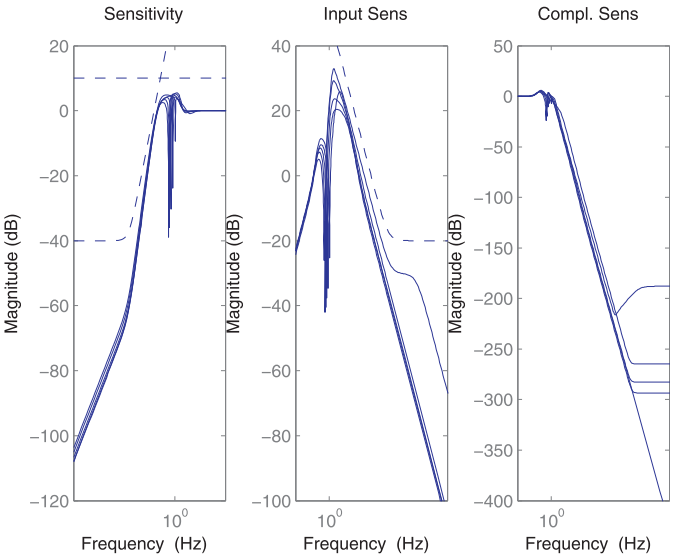

Fig. 10. Frequency domain validation of the LPV controller design for the overhead crane example. The closed-loop frequency responses are solid, the constraints are dashed.

transfer functions (solid lines) have a smaller magnitude than their respective weights (dashed lines).

After a quick frequency domain assessment, time domain simulations may yield additional insight in the behavior of the controller. Although standard commands such as `step` or `impulse` are available, the most powerful command is `sim`. This function carries out a time domain simulation for a system, provided a time dependent input and parameter value. Because a system may contain several models, responses are readily compared. Fig. 11 shows the simulation result when the cart moves 0.3 m while the load is lowered from 0.4 m to 0.7 m at maximum velocity and executes the opposite manoeuvre after a few seconds (Code example 10). In this case, two simulations are shown: one for the LPV model and one for the nonlinear model. As expected, only small differences occur between the two responses. This observation indicates that the interpolated LPV model describes the nonlinear dynamics sufficiently accurate. Second, the tracking behavior looks adequate and the resonance frequency is sufficiently dampened, even for a varying cable length. To experimentally validate the designed B-spline LPV controller, it is exported to a C++ implementation with Eigen [24] as a linear algebra back-end and a custom implementation of De Boor's algorithm [25] to evaluate the B-splines. Since the controller was designed in continuous time, a backward Euler scheme is employed to update the control law at a rate of 100 Hz. Fig. 11 shows the measured load position as well as the control signal along the simulations. It is readily seen that the experimental results match the simulations very well both qualitatively and quantitatively.

7. Conclusion

This paper has highlighted the main features of the novel Linear Control Toolbox. Three complementary modules, i.e. the system modeling module, identification module and controller design module, as well as the different validation tools assist the practicing control

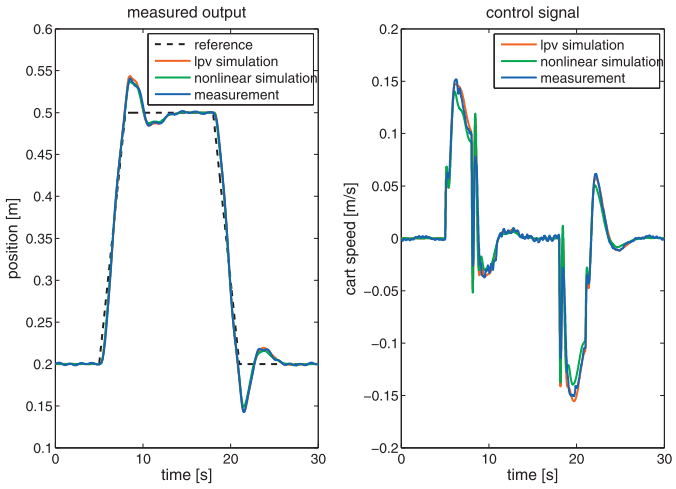

Fig. 11. Response of the closed loop when hoisting the load from 0.4 m to 0.7 m and moving the load 0.3 m simultaneously for a time span of 3 s. After 10 s, the motion is repeated in the opposite direction so that the load returns to its initial position.

engineer along the journey towards a high performance model-based $\mathscr{H}_\infty/\mathscr{H}_2$ controller. The most important data structures have been presented and important design choices were justified. Based on a two-parameter missile control case study, the Linear Control Toolbox was compared to its competitors: LPVtools and Matlab's Robust Control Toolbox. It was shown that the Linear Control Toolbox stands out when it comes to the variety of problems it is able to solve as well as formulating the control problem in a readable manner. An overhead crane setup served as a second case study to validate the Linear Control Toolbox both in simulation and experimentally. Moreover, developments are still ongoing and include the use of uncertain models to design robust controllers, the implementation of direct LPV identification methods and the optimal selection of sensors and actuators.

References

[1] Hoffmann C, Werner H. A survey of linear parameter-varying control applications validated by experiments or high-fidelity simulations. IEEE Trans Control Syst Technol 2015;23(2):416–33.
[2] Apkarian P, Dao MN, Noll D. Parametric robust structured control design. Autom Control IEEE Trans 2015;60(7):1857–69.
[3] Apkarian P, Gahinet P, Becker G. Self-scheduled H∞ control of linear parameter-varying systems. 1. American Automatic Control Council; 1994. p. 856–60.
[4] Hjartarson A, Seiler P, Packard A. Lpvtools: a toolbox for modeling, analysis, and synthesis of parameter varying control systems. IFAC-Papers OnLine 2015;48(26):139–45.
[5] Verbandt M. An LTI control toolbox - simplifying optimal feedback controller design. Conference proceedings European control conference (2016). 2016.
[6] Hilhorst G, Lambrechts E, Pipeleers G. Control of linear parameter-varying systems using B-Splines. 55th IEEE conference on decision and control, Las Vegas, USA, December. 2016. p. 12–4.
[7] Doyle J, Packard A, Zhou K. Review of LFTs, LMIs, and mu. IEEE Publishing; 1991. p. 1227–32. ISBN 0-7803-0450-0
[8] Apkarian P, Adams RJ. Advanced gain-scheduling techniques for uncertain systems. IEEE Trans Control Syst Technol 1998;6(1):21–32.
[9] Pintelon R., Schoukens J. http://wiley.mpstechnologies.com/wiley/BOBContent/downloadBobProjectFile.do?bpfId=1029&bpfFileType=.zip&bpfFileName=FreqDomBox.zip; 2012a. Accessed October 20, 2017.
[10] Pintelon R, Schoukens J. System identification: a frequency domain approach. Second Edition Wiley; 2012. ISBN 9781118287392
[11] Turk D, Pipeleers G, Swevers J. A combined global and local identification approach for LPV systems. IFAC PapersOnLine 2015;48(28):184–9.
[12] Turk D, Gillis J, Pipeleers G, Swevers J. Identification of linear parameter-varying systems: a reweighted L2,1-norm regularization approach. Mech Syst Signal Process 2018;100:729–42.
[13] Caigny JD, Camino JF, Swevers J. Interpolation-based modeling of MIMO LPV systems. IEEE Trans Control Syst Tech 2011;19(1):46–63. http://dx.doi.org/10.1109/TCST.2010.2078509.
[14] Apkarian P, Noll D. The $\mathscr{H}_\infty$ control problem is solved. AerospaceLab J 2017(13).
[15] Kwakernaak H. Robust control and $\mathscr{H}_\infty$ optimization – tutorial paper. Automatica 1993;29(2):255–73.

[16] Gahinet P, Apkarian P. A linear matrix inequality approach to $\mathscr{H}_\infty$ control. Int J Robust Nonlinear Control 1994;4:421–48.
[17] Pólya G. Über positive darstellung von polynomen. Vierteljschr Naturforsch Ges Zürich 1928;73:141–5.
[18] Marshall M. Positive polynomials and sum of squares. Mathematical surveys and monographs. American Mathematical Society; 2008.
[19] Mathworks. Robust control toolbox. https://nl.mathworks.com/products/robust.html, Accessed: 2018-03-05.
[20] Gahinet P., Nemirovski A., Laub A.J., Chilali M. LMI control toolbox; chap. 7. 1995, p. 10–14.
[21] Megretski A. Dynamic systems and control. Lecture notes (Fall 2006) Well Posedness of LTI Feedback Design; 2006. p. 9–10. Chap. 21
[22] Lambrechts E., Gillis J. Optispline. https://github.com/meco-group/optispline; 2018.
[23] Debrouwere F, Vukov M, Quirynen R, Diehl M, Swevers J. Experimental validation of combined nonlinear optimal control and estimation of an overhead crane. Proceedings of the 19th IFAC world congress. 2014. p. 9617–22.
[24] Guennebaud G., Jacob B., et al. Eigen v3. http://eigen.tuxfamily.org; 2010.
[25] De Boor C. A practical guide to splines. 27. Applied Mathematical Sciences; 2001. revised edition