Hindawi
Journal of Computational Engineering
Volume 2017, Article ID 3169785, 14 pages
https://doi.org/10.1155/2017/3169785

Research Article

New Coordination Software for Parameter Identification Applied to Thermal Models of an Actuator Strut

Burkhard Hensel,[1] Steffen Schroeder,[2] and Klaus Kabitzsch[1]

[1]*Chair for Technical Information Systems, Dresden University of Technology, 01062 Dresden, Germany*
[2]*Institute of Machine Tools and Control Engineering, Dresden University of Technology, 01062 Dresden, Germany*

Correspondence should be addressed to Burkhard Hensel; burkhard.hensel@tu-dresden.de

Received 24 May 2017; Accepted 28 August 2017; Published 10 October 2017

Academic Editor: Damodar Maity

The practical worth of models of technical processes depends on their accuracy, that is, the difference between model outputs and real measurements. For minimizing these differences, process identification methods are used. In this article, coordination software for process identification is presented which has the unique feature that it allows the integration of models that have been created with external tools, for example, Matlab or Python scripts. There is no need to transform the models into another type of software format to use the common identification coordinator. The concept of the software is described and two examples for the coupling with external simulation software are given. Additionally, this article contains a detailed case study of the parameter identification of two models using that identification coordination software. This highlights the benefit of the new coordination software regarding similar work flow for different model types. The modeled physical subject is the thermal behavior of an actuator strut.

1. Introduction

Models of technical systems are used for solving many kinds of problems reaching from prediction over simulation, condition monitoring, and failure detection to controller design. However, the quality of the results for all these goals depends on the model quality, that is, the accuracy. The smaller the difference between the model and the real process is, the more accurate a prediction, control loop, and so forth can be.

All model types consist of a model *structure* and a number of *parameters*. The *structure* defines the underlying equations, assumptions, and the resolution (granularity, order) of the model. One model structure can often be used for many technical processes with the same *qualitative* behavior. The *parameters* (e.g., coefficients of polynomials, differential, or difference equations) can be adjusted for getting the desired *quantitative* results. The task of finding the parameter values which minimize the difference between real measurements and simulation results of the model (for the same inputs and other conditions) is called *system identification*, *process identification,* or *parameter identification*.

There are a lot of tools which can be used for parameter identification; an overview is given in [1]. Probably the most frequently used software is Matlab [2]. There are many toolboxes for Matlab which support process identification for a lot of different kinds of models. However, for models which have *not* been implemented in Matlab, usually other identification tools appropriate to the other modeling software must be used. This often results in suboptimal identification results as, for example, due to time reasons, only simple tuning methodologies are used, often just manual adjustments in a trial-and-error manner.

Recently, higher level process identification software has been presented, which neither provides *new* process identification methods nor *replaces* existing identification software. Instead, it sets a coordination view on top of them [1]. Besides process identification, this software has its focus on virtual sensor design and can thus be used not only on standard PCs in offline mode, but also on embedded PCs in online mode. In the state at the publication of [1] it was not possible to use external models in that software. The software has now been extended to support different external model types (e.g., Matlab and Python scripts) additionally to "internal" model types (strictly speaking realized as Java plug-ins) like static equations, difference equations, or discrete time and continuous-time transfer functions. This allows using the identification software for already existing models that have been implemented in external software without the need to convert these models to new software's format. Additionally, it is even possible to combine external models (cosimulation). For example, the parameters of a Matlab script model can in that way easily be optimized simultaneously with parameters of a Python script model.

The article is structured as follows. In Section 2, the basic system identification procedure, which is the background of the software, is presented. An overview about the high level identification software is given in Section 3 with the focus on the integration of external models. In Section 4, an application example (case study) is presented in detail. In that example, two different model types are used to model the same physical object. The example is the thermal modeling of an actuator strut that can be found in similar form in many machine tools. Although the models have been implemented in different software, the new higher level software has been used for both models to coordinate the process identification. The section compares both the quality of the models and the effort for creating them. Finally, conclusions are drawn (Section 5).

The abbreviations and symbols used in this article are presented in Notations.

2. Identification Procedure

In this section, the identification procedure is presented, which is supported by the software. It is based on [1, 3, 4].

Let $\underline{y}(k)$ be a vector of M time series where k is the index of the sampling instant. Each element of the vector represents a time series for one physical quantity. More concretely, $\underline{y}_{\text{model}}(k)$ is the vector of the outputs of the model and $\underline{y}_{\text{measurement}}(k)$ the vector of measured time series for the same physical quantities and under the same conditions and inputs. The difference between the model outputs and the corresponding measurements is mostly evaluated using the sum of squared error (SSE)

$$\underline{\text{SSE}} = \left[\text{SSE}_1 \;\; \text{SSE}_2 \;\; \cdots \;\; \text{SSE}_N\right]^T, \tag{1}$$

$$\text{SSE}_i = \sum_{j=1}^{N} \left(y_{\text{model},i}(k) - y_{\text{measurement},i}(k)\right)^2, \tag{2}$$

where N is the number of recorded time steps (sampling instants), $\underline{\text{SSE}}$ is the vector of squared errors, and i is the row index in the vectors $\underline{y}_{\text{model}}(k)$, $\underline{y}_{\text{measurement}}(k)$, and $\underline{\text{SSE}}$. Another often used measure is the root-mean-square error (RMSE)

$$\begin{gathered}\underline{\text{RMSE}} = \left[\text{RMSE}_1 \;\; \text{RMSE}_2 \;\; \cdots \;\; \text{RMSE}_N\right]^T, \\ \text{RMSE}_i = \sqrt{\frac{\text{SSE}_i}{N}}.\end{gathered} \tag{3}$$

In general, there are two main strategies of parameter identification.

(1) The first one is the *analytical computation* of the optimal parameters for a given model structure according to given measurements. For that purpose, there are mathematical strategies like least squares method, instrumental variable method, maximum likelihood method, and so forth [5, 6]. These methods deliver the optimal parameters for a given optimization criterion (usually $\underline{\text{SSE}}$ with $M = 1$) after a finite number of computational steps. However, this is only possible for certain model structures, for example, linear models, polynomials for static models, or linear difference equations for dynamic models, mostly models that are linear in their parameters. Most text books as well as publications on process identification focus on analytical methods [5, 6].

(2) On the other hand, there is the possibility of "training" models using *iterative optimization methods*. In this case, the model is simulated again and again with varied parameters and the difference ($\underline{\text{SSE}}$) between the model output and the measurements is evaluated. There are a lot of optimization strategies like Monte Carlo optimization, simulated annealing, Levenberg-Marquardt, gradient-based search, and evolutionary algorithms. Most such algorithms take less assumptions on the model structure than analytical methods and can therefore be used for a wide variety of model types or even unknown (black-box) models. Therefore, they are the usual way of process identification for complex model types like finite element models [7, 8]. But also for simple models, where analytical methods exist, such optimization methods are sometimes used; a well-known example is the Matlab System Identification Toolbox [9]. However, these methods do not guarantee to deliver the globally optimal results (due to local optima), especially not in finite time.

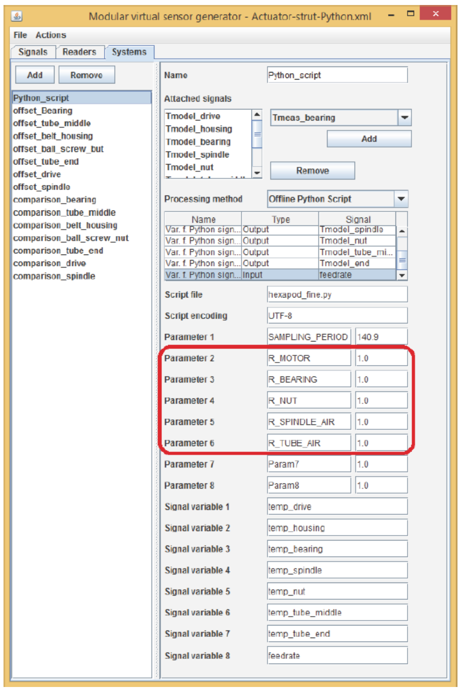
Both types of identification procedures are supported by the new software to give a maximum of freedom to the user. It is also possible to combine both strategies. For example, linear model parts can first be identified analytically, and more complex or even black-box parts can be identified via evolutionary algorithms, afterwards.

3. Software Overview

A first description of the developed software has been given in [1]. The coordinator manages a modular model where the subsystems are connected via "signals," as scientifically known from "systems theory."

Most functionality is provided to that core via *plug-ins*. There are currently three types of plug-ins:

(1) *System-processing plug-ins* provide model type specific functions, especially model simulation, parameter identification, and inversion.


Modular virtual sensor generator - Actuator-strut-Python.xml
File
Actions
Signals
Readers
Systems
Add
Remove
Add
Remove

Python_script
offset_Bearing
offset_tube_middle
offset_belt_housing
offset_ball_screw_but
offset_tube_end
offset_drive
offset_spindle
comparison_bearing
comparison_tube_middle
comparison_belt_housing
comparison_ball_screw_nut
comparison_tube_end
comparison_drive
comparison_spindle

Name	Python_script
Attached signals	Tmodel_drive, Tmodel_housing, Tmodel_bearing, Tmodel_spindle, Tmodel_nut
	Tmeas_bearing
Processing method	Offline Python Script

Name	Type	Signal
Var. f. Python sign...	Output	Tmodel_spindle
Var. f. Python sign...	Output	Tmodel_nut
Var. f. Python sign...	Output	Tmodel_tube_mi...
Var. f. Python sign...	Output	Tmodel_end
Var. f. Python sign...	Input	feedrate

Script file	hexapod_fine.py	
Script encoding	UTF-8	
Parameter 1	SAMPLING_PERIOD	140.9
Parameter 2	R_MOTOR	1.0
Parameter 3	R_BEARING	1.0
Parameter 4	R_NUT	1.0
Parameter 5	R_SPINDLE_AIR	1.0
Parameter 6	R_TUBE_AIR	1.0
Parameter 7	Param7	1.0
Parameter 8	Param8	1.0
Signal variable 1	temp_drive	
Signal variable 2	temp_housing	
Signal variable 3	temp_bearing	
Signal variable 4	temp_spindle	
Signal variable 5	temp_nut	
Signal variable 6	temp_tube_middle	
Signal variable 7	temp_tube_end	
Signal variable 8	feedrate	

FIGURE 1: Screenshot of the parameter selection for a script (variable names and values). The physical meaning of the parameters is explained in Section 4.2.

(2) *Reader plug-ins* are responsible for getting measurement data into the software—for example, from files, data bases, or (in online mode) directly from sensors—that is used to compare the model outputs with the real modeled system.

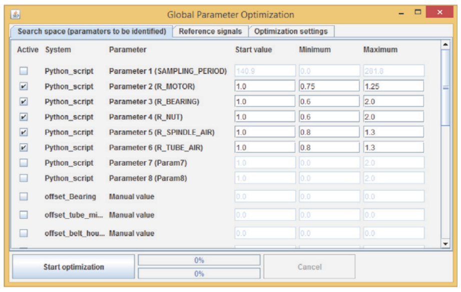
(3) *Visualization plug-ins* are used to visualize simulation results, for example, graphically or as tables, or for writing output data into files.

Besides the management of the modular model and system identification work flow the core contains standard optimization methods that are independent of the model type and can thus be used for all model types together.

One of the reasons for the plug-in concept is the ability to extend the identification software by functionality of other software products using "wrapper plug-ins." That concept has now been used to integrate models that have been implemented as Matlab or Python scripts. For the end user of the software, the following work flow results.

(1) Usually, models that have been written as scripts contain adjustable parameters as *variables* of the software. If that is not the case, variables for uncertain parameters that are to be identified have to be created in the script to use the identification coordinator.

(2) After that, in the GUI (graphical user interface) of the identification coordinator, a system of type "Matlab script" or "Python script" is created. In the corresponding dialog the used script file name and the names of the parameters that have to be optimized are edited (Figure 1). It is also possible to set parameter values manually from the software, which is helpful if the same script is used in different contexts.


Global Parameter Optimization
Search space (paramaters to be identified)
Reference signals
Optimization settings
Start optimization
0%
0%
Cancel

Active	System	Parameter	Start value	Minimum	Maximum
☐	Python_script	Parameter 1 (SAMPLING_PERIOD)	140.9	0.0	281.8
✓	Python_script	Parameter 2 (R_MOTOR)	1.0	0.75	1.25
✓	Python_script	Parameter 3 (R_BEARING)	1.0	0.6	2.0
✓	Python_script	Parameter 4 (R_NUT)	1.0	0.6	2.0
✓	Python_script	Parameter 5 (R_SPINDLE_AIR)	1.0	0.8	1.3
✓	Python_script	Parameter 6 (R_TUBE_AIR)	1.0	0.8	1.3
☐	Python_script	Parameter 7 (Param7)	1.0	0.0	2.0
☐	Python_script	Parameter 8 (Param8)	1.0	0.0	2.0
☐	offset_Bearing	Manual value	0.0	0.0	0.0
☐	offset_tube_mi...	Manual value	0.0	0.0	0.0
☐	offset_belt_hou...	Manual value	0.0	0.0	0.0

FIGURE 2: Screenshot of the form for selection of the parameters to be identified with their lower and upper limits. The physical meaning of the parameters and their limits is explained in Section 4.2.

(3) The needed other parts for process identification are specified. This contains the selection of an appropriate "reader" (e.g., file format, data base) for the measurement data that is to be used, as well as—if necessary—additional preprocessing steps or further model parts that are not part of the script.

(4) Since scripts can usually contain all thinkable types of models or combinations of them, they are treated as black-box in the current implementation. The Matlab and Python plug-ins provide therefore no model-specific optimization methods. Because of this, the window for iterative standard parameter optimization can be opened. The parameters that shall be optimized automatically are selected together with their expected bounds and—if needed for the optimization method that is chosen later—start values (Figure 2).

(5) In the next step, the optimization criterion is chosen. This is a set of signal pairs (usually a pair of one measurement signal $y_{\text{measurement}}$ and one simulation output signal y_{model}) together with a weighting factor for each pair. Let $\underline{w}$ be the vector of weighting factors with size M; then the error

$$e = \underline{w} \cdot \underline{\text{SSE}} \tag{4}$$

is the evaluated criterion, with $\underline{\text{SSE}}$ taken from (1).

(6) Now, the identification method is chosen. Only model independent identification methods are provided by the core, for example, Monte Carlo simulation, simulated annealing, and an evolutionary algorithm.

(7) Finally, the parameter optimization can be started. The possible stop criteria are currently a defined number of iteration steps and a time limit. A quality limit (i.e., desired value of e) could be added easily. The parameter optimization finishes when (at least) *one* of the specified criteria is reached.

The coupling with Matlab is realized with the "Matlab Engine API for Java" [10] and the coupling with Python by integrating the open-source Jython interpreter [11]; see Figure 3. These

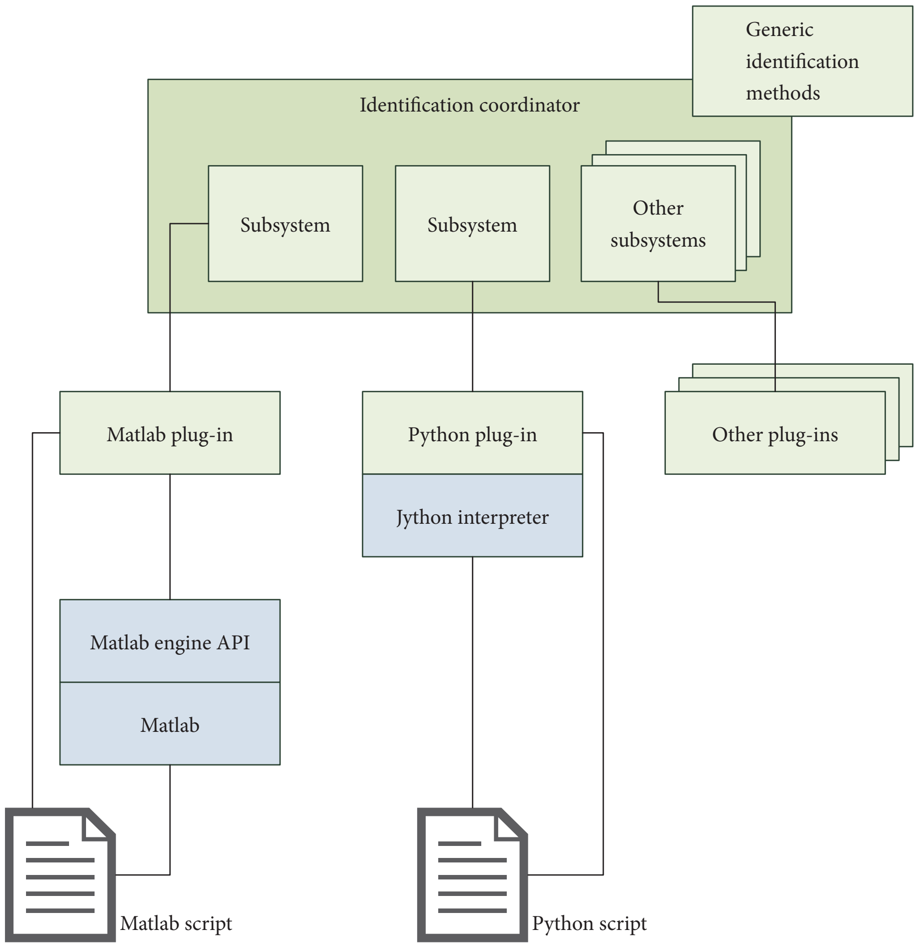


Figure 3: Integration of external scripts in the identification coordinator.

two solutions represent also two of the three main possible ways of integrating external software into a Java plug-in:

(1) Use an API (application programmer's interface) of the external software to directly communicate with the external software for setting parameter values, starting simulation, and exchanging signals.

(2) Provide an *interpreter or simulator* for the external models as a plug-in.

(3) Start a simulation as a *batch job*, that is, via command line.

The last type of coupling should only be chosen if the other two solutions are not possible or if the required libraries are too large for the application (e.g., on embedded systems), because starting a simulation as a batch job is usually slower, especially if the start of the external simulation software needs significant time.

4. Case Study: Actuator Strut

In this section, the concept of the software is explained using a practical application example, that is, the comparison of two model types for the same modeled object.

The system to be modeled is an actuator strut. This example is taken from the collaborative research center Transregio 96, in which the software for parameter identification has been created. Actuator struts are typically used in machine tools, where for each axis (x, y, and z) one actuator strut is used to move either the work piece or the main spindle with the tool. Figure 4 shows the modeled actuator strut which has been separated from the overall machine for instrumentation and experimentation. It has only one relevant way of movement: the spindle can move out of and into the tube. Since one spindle end is fixed at the drive end ("bearing"), the spindle itself does not move, but the tube does (around the spindle). However, this has no influence on the thermal model as it is only a question of a coordinate transformation.

From the research project's perspective, the interesting quantity is the thermal "deformation" which is here only relevant in one dimension (axial), that is, the change of the length of the actuator strut due to temperature changes. This is interesting because thermal deformation in machine tools leads to less accuracy of production processes like milling or drilling. The thermal deformation is a function of the temperature field in the actuator strut. Therefore, the possibility of computing the temperature field is practically

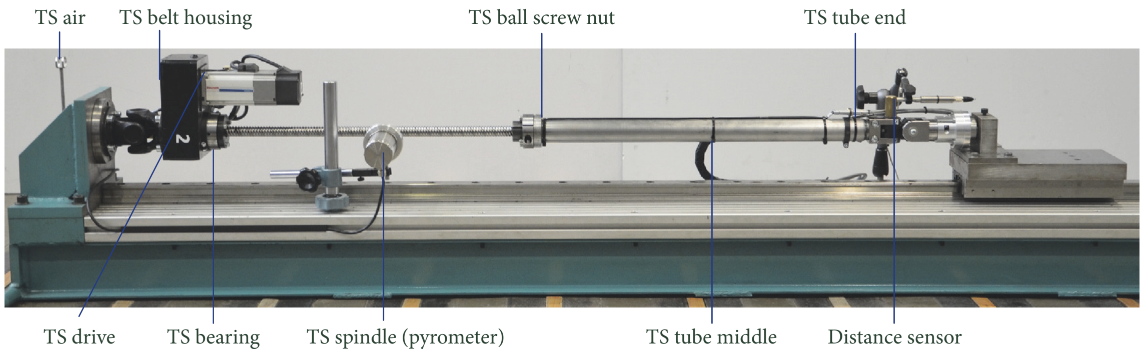


FIGURE 4: Actuator strut with measurement points. “TS” = “temperature sensor.”

enough to estimate the change of the length that can be measured only with much higher costs than the temperature.

Since the object is long and thin and only at one end fixed in x direction, only one direction (axial) is important for practical use. Assuming that for each point x on the axial direction the strut has the same temperature $\vartheta(x, t)$ for each y and z, the change Δl of the length can be estimated from

$$\Delta l = \int_0^l \alpha(x)\, \Delta\vartheta(x)\, \mathrm{d}x, \tag{5}$$

where $\alpha(x)$ is the linear heat expansion coefficient for the substance at point x, $\Delta\vartheta(x)$ the change of the temperature at point x compared to its initial value, and l the initial length [12].

The heat flow in a solid is described by the heat equation

$$c \cdot \left(\frac{\partial^2\vartheta}{\partial x^2} + \frac{\partial^2\vartheta}{\partial y^2} + \frac{\partial^2\vartheta}{\partial z^2}\right) = \frac{\partial\vartheta}{\partial t}, \tag{6}$$

where c is a constant (thermal diffusivity). However, this three-dimensional partial differential equation is too complex for practical simulation. At the same time, it is even not sufficient for describing the thermal behavior of the actuator strut, because of missing consideration of convection or heat radiation at surfaces. Therefore, several simplified model structures are available to approximate the behavior. Two of them will be used in the models described below. The *input* quantity of the models is the feed-rate (speed) and the *outputs* are the temperatures at the measured points. The deformation (change of length) is left out of consideration in the models but could be added using (5).

Starting from a thermal steady state, temperature curves (step responses) have been measured at several positions at the strut. For gaining these measurements, the actuator strut has been moved periodically with a constant feed-rate (velocity) for a duration between 80 and 170 minutes. Measurements have been taken for the feed-rates 20 m/min, 25 m/min, and 30 m/min. The sensor positions are shown in Figure 4 and the measurement results in Figure 5.

In the following, it is shown how two models are set up using the data for a feed-rate of 20 m/min. The parameters are estimated using the new software. The quality of each model is evaluated by simulating the model for the feed-rates 20 m/min, 25 m/min, and 30 m/min and comparing the outputs with the appropriate measurements. The samples are nearly equidistant, but for technical reasons the sampling period depends on the feed-rate and is thus not equal for the three cases. For the case $v = 20$ m/min, the sampling period is 2.35 min (140.9 s), for $v = 25$ m/min 2.48 min (148.7 s), and for $v = 30$ m/min 2.64 min (158.3 s).

4.1. Model 1: Set of Transfer Functions. The measured temperature curves of the experiments shown in Figure 5 are typical step responses of nonintegrating systems; that is, the influence of the speed on each temperature measurement point can be modeled either by a higher order dynamic system, a series of first-order systems, or (roughly) a first-order system with a time delay (FOPTD). The latter has the advantage that the parameters can be interpreted intuitively (but not physically) by comparing them with the step response. The transfer function of a FOPTD model is

$$G_i(s) = \frac{\Theta_i(s)}{V(s)} = \frac{K_i}{1 + T_i s} e^{-sL_i}, \tag{7}$$

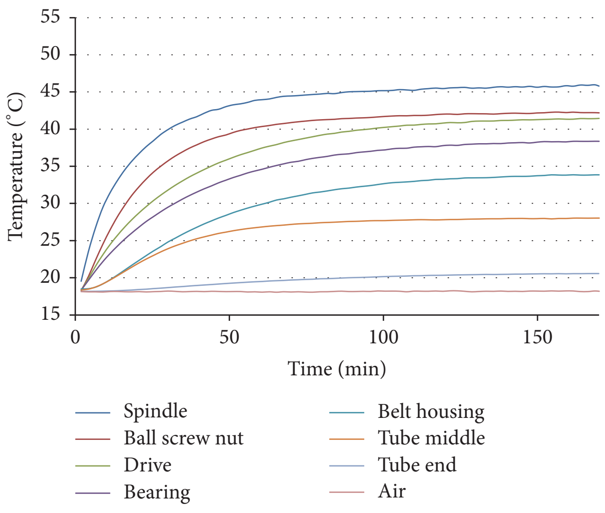
where K_i is the proportional action coefficient, T_i the time constant (first-order lag), L_i the time delay or dead time, and s the Laplace variable (complex frequency of the Laplace transform [13]). $\Theta_i(s)$ is the Laplace transform of a (scalar) output (one temperature measurement point $\vartheta_i(s)$). $V(s)$ is the Laplace transform of the input (feed-rate $v(t)$). i is the index of the measurement position. The transfer function (7) is equivalent to the differential equation

$$\vartheta_i(t) + T_i \frac{\mathrm{d}\vartheta_i(t)}{\mathrm{d}t} = K_i \cdot v(t - L_i), \tag{8}$$

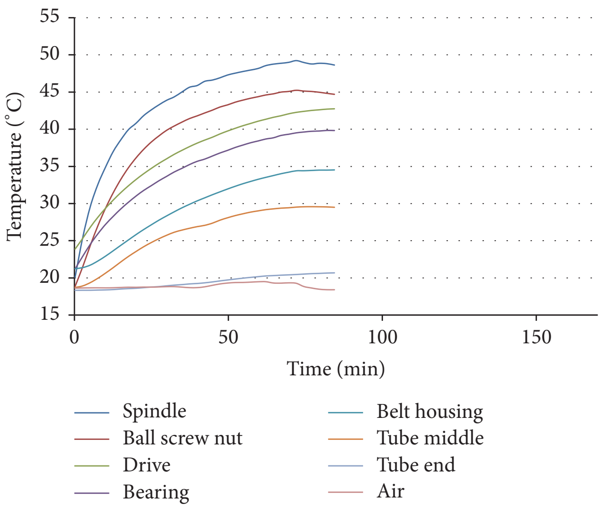
where t is the (continuous) time.

Even this simple model type has a relatively complex modular structure (see Figure 6), because for each measurement point a separate FOPTD model has to be set up and the initial temperature of the measurements has to be subtracted. The latter is the case, because for the input (speed) equal to zero a linear model like an FOPTD model always delivers zero as output.

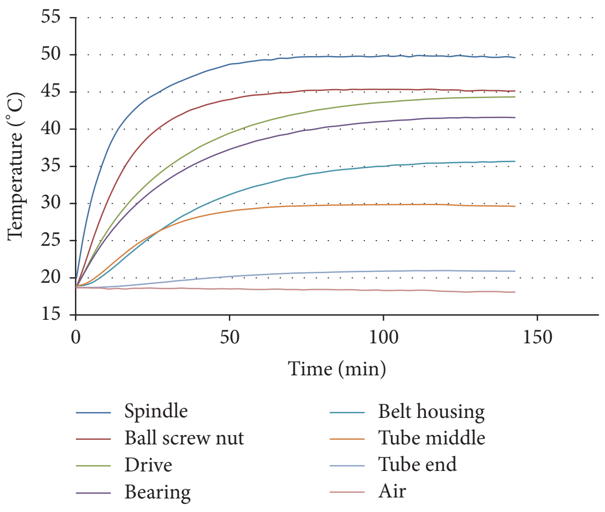
Since there are seven measurement points and each FOPTD model contains three parameters, the overall model has totally 21 parameters.



(a) $v = 20$ m/min



(b) $v = 25$ m/min



(c) $v = 30$ m/min

FIGURE 5: Measurements used for parameter identification.

The values of the parameters K_i and T_i can be identified using the least squares method [5, 6, 14]. For that purpose, first the coefficients a_{m1} and b_{m0} of a discrete difference equation

$$\vartheta_i[k] + a_{m1} \cdot \vartheta_i[k-1] = b_{m0} \cdot v[k - \tilde{\tau}] \qquad (9)$$

or the equivalent transfer function

$$G_d\left(z^{-1}\right) = z^{-\tilde{\tau}} \frac{b_{m0}}{1 + a_{m1} z^{-1}} = z^{-\tilde{\tau}} \frac{K_i \cdot (1 - m_1)}{1 - m_1 z^{-1}} \qquad (10)$$

with the discrete time delay $\tilde{\tau} = L_i / T_s$ and

$$\begin{aligned} m_1 &= \exp\left(-\frac{T_s}{T_i}\right), \\ a_{m1} &= -m_1, \\ b_{m0} &= K_i \cdot (1 - m_1) \end{aligned} \qquad (11)$$

are computed. z is the variable of the z-Transform [13] and T_s the sampling period. Then, the continuous-time parameters K_i and T_i can be computed from them using

$$\begin{aligned} \widehat{K}_i &= G_d(1) = \frac{b_{m0}}{1 + a_{m1}}, \\ \widehat{T}_i &= -\frac{T_s}{\ln m_1} = -\frac{T_s}{\ln(-a_{m1})}. \end{aligned} \qquad (12)$$

The discrete time delay $\tilde{\tau}$ (and thus also the continuous-time time delay L_i) cannot be computed using the least squares method. Therefore, for each integer value of $\tilde{\tau}$ in the interval $[0, 20]$ the model parameters have been computed (automatically) and the one with the smallest mean quadratic error has been used.

This model type is not provided via external software as it is relatively simple. The model structure, the appropriate version of the least squares algorithm, and the loop of time

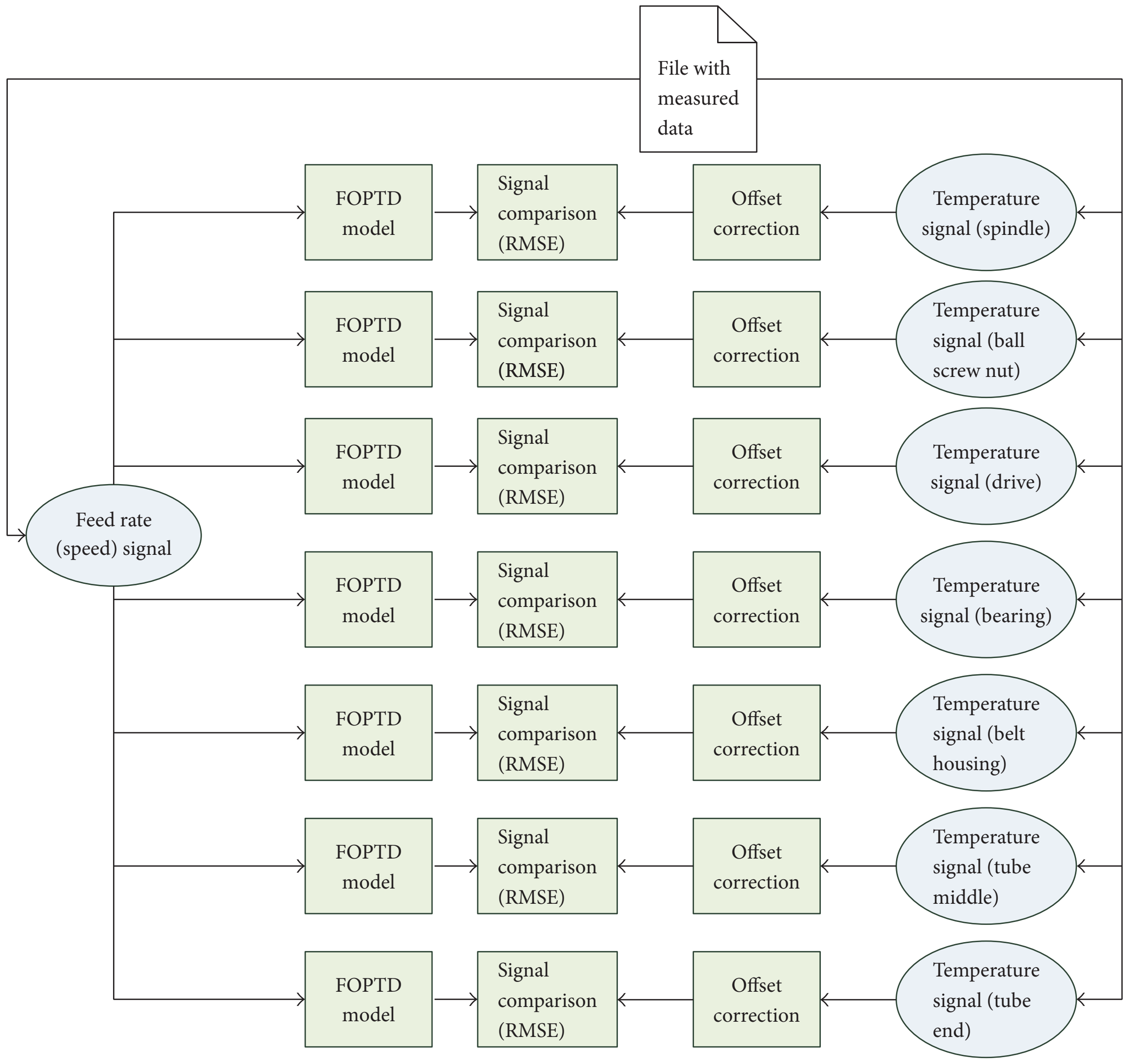

File with measured data
FOPTD model
Signal comparison (RMSE)
Offset correction
Temperature signal (spindle)
Temperature signal (ball screw nut)
Temperature signal (drive)
Temperature signal (bearing)
Temperature signal (belt housing)
Temperature signal (tube middle)
Temperature signal (tube end)
Feed rate (speed) signal

Figure 6: Structure of the model consisting of FOPTD models.

delays have been directly programmed as a Java plug-in without a link to an external software product.

Table 1 shows the identified values of the parameters for each measurement point for the feed-rate 20 m/min.

The validation cases show much larger values of RMSE than the optimized case. This is a hint that the true modeled object is nonlinear. In the case $v = 25$ m/min, the biggest differences occur for the drive, because it had not fully cooled down before the experiment started. For $v = 30$ m/min all values of RMSE become much larger. This can be explained with the nonlinearity, as according to the measured data the final temperature does not change significantly although the final temperature of the model output is proportional to v due to the superposition principle of linear models.

Positions that are far away from any heat source show the largest time constants, the highest delay, and the smallest proportional action coefficient. The curve of the belt housing is typical for a higher order process due to the long way from the heat sources to the measurement point. Since the model is not of higher order, the time delay is in that case larger. Also the tube is reached relatively slowly by the heat, resulting in long time constants and a time delay.

Since the delay can only be estimated as an integer multiple of the sampling period, further deviations occur. However, since the changes of the sampling period are below 13%, this influence can be neglected compared to large influence of the nonlinearity. Also the importance of the influence of the air temperature is minor.

4.2. Model 2: Electric Equivalent Circuit Diagram

4.2.1. Model Overview. In this section, the actuator strut is modeled using an electric equivalent circuit diagram (based on electrothermomechanical analogies), shown in a simplified form in Figure 7. In an electric equivalent circuit diagram, each relevant physical component (with index i) of the actuator strut is modeled by a node with a heat capacity C_i and a power loss $\dot{Q}_i$ (heat source). Between each pair of nodes

Table 1: FOPTD models for temperature sensors. Optimized for $v = 20$ m/min (sampling period 2.35 min), other cases as validation without new parameter optimization.

Number	Sensor	Delay L in min	Prop. act. coeff. K in Kmin/m	Time constant T in min	RMSE 20 m/min in K	RMSE 25 m/min in K	RMSE 30 m/min in K
8	Spindle	0	1.33	22.39	0.66	2.49	7.96
5	Ball screw nut	0	1.24	28.25	0.74	1.78	7.60
3	Drive	0	1.20	39.06	0.45	5.59	7.05
4	Bearing	0	1.05	42.52	0.40	2.48	5.20
2	Belt housing	4.70	0.85	57.53	0.51	1.38	3.83
6	Tube middle	2.35	0.52	41.99	0.48	0.42	2.50
7	Tube end	11.74	0.15	93.04	0.08	0.19	0.56
	Mean of all	—	—	—	0.47	2.05	4.96

(i, j) there is thermal conductance g_{ij}. If two nodes have no direct contact to each other, the conductance is zero.

Each node i of the P nodes is modeled with the first-order ordinary differential equation

$$C_i \cdot \frac{\mathrm{d}\vartheta_i(t)}{\mathrm{d}t} = \dot{Q}_i + \sum_{j=1, j\neq i}^{P} g_{ij} \cdot \left(\vartheta_j(t) - \vartheta_i(t)\right). \tag{13}$$

However, in practice, the heat flows $\dot{Q}_i(t)$, which are the inputs of the model (13), are not known. Instead, the feedrate $v(t)$ is known and it is assumed that the heat flows $\dot{Q}_i(\mathrm{t})$ are a function of $v(t)$. In the simplest case, a proportional dependency can be assumed; that is,

$$\dot{Q}_i(t) = q_i \cdot v(t) \tag{14}$$

with a proportionality factor q_i (in J/m).

4.2.2. Parameters of the Model. The model contains three types of parameters: a heat capacity C_i and a velocity-dependent heat loss q_i for each node and heat conductance g_{ij} between each pair of connected nodes. Nodes that are not connected have a heat conductance g_{ij} of 0. The heat capacity of the air can be treated as infinity.

Since g_{ii} is not needed and $g_{ij} = g_{ji}$, it is enough to estimate the heat conductance g_{ij} with $i < j$, that is, $((P-1)\cdot P)/2$ heat conductance parameters. Together with the P heat capacities and P heat losses, the total number of parameters can be computed to $2\cdot P + ((P-1)\cdot P)/2$. The model of Figure 7 contains 19 nodes, so the number of parameters is 209.

4.2.3. Parameter Reduction. The whole model can be represented by a system of first-order differential equations according to (13). Therefore, it is possible to create an equivalent state-space model and use analytical methods for parameter identification [15, 16]. However, the analytical process identification methods for state-space models are quite complex, because not only are there many parameters to be estimated (here 209), but also the state has to be estimated, here the temperatures of all model nodes—also these that are not measured at the real machine. Additionally, in the present example the model has been implemented as a Python script (see Figure 8) and the goal is to reuse this script and not to convert it into a state-space model of another type of software. In that way, this script serves as an example for the integration of arbitrary external scripts onto the identification coordinator. Because of that, only standard optimization methods are used. However, to identify 209 parameters simultaneously, especially with standard optimization methods, is practically hopeless because of the huge search space. Fortunately, the number of parameters to be identified can be heavily reduced using a priori knowledge. In particular, the 5 most uncertain parameters have been determined and only these are optimized by the optimization algorithm. All other parameters are set to values that have been estimated using physical knowledge. The strategy of finding the 5 most uncertain parameters and how the others have been estimated is described in Appendix.

The remaining 5 parameters are "ratio parameters"; that is, they have no physical meaning but are factors that are multiplied with theoretically estimated physical parameters. The ratio parameter r_m is related to the motor heat loss, r_b to the heat loss of the bearing, r_n to the heat loss of the ball screw nut, r_{sa} to the heat conductance between the spindle and the air, and r_{ta} to the heat conductance between the tube and the air. See Appendix for details.

The limits of the ratio parameters can be computed from the uncertainties shown in [4]; that is, the uncertainty is +200/−70% for the power loss of ball screw nut and bearing, +30/−20% for heat conductance with free convection and radiation, and ±25% for the motor power loss. Therefore

$$\begin{gathered} r_m \in [0.75, 1.25], \\ r_b, r_n \in [0.6, 2.0], \\ r_{\mathrm{sa}}, r_{\mathrm{ta}} \in [0.8, 1.3]. \end{gathered} \tag{15}$$

In conclusion, the Python script has the five parameters r_n, r_b, r_m, r_{sa}, and r_{sb} which are multiplied with the theoretically estimated model parameters according to Table 5.

4.2.4. Evaluation. Each parameter r_i has been optimized in the given interval using Monte Carlo optimization. This optimization strategy is relatively slow but avoids sticking in local minima. The results are presented in Table 2. As for the FOLPD models, the parameters have been optimized for

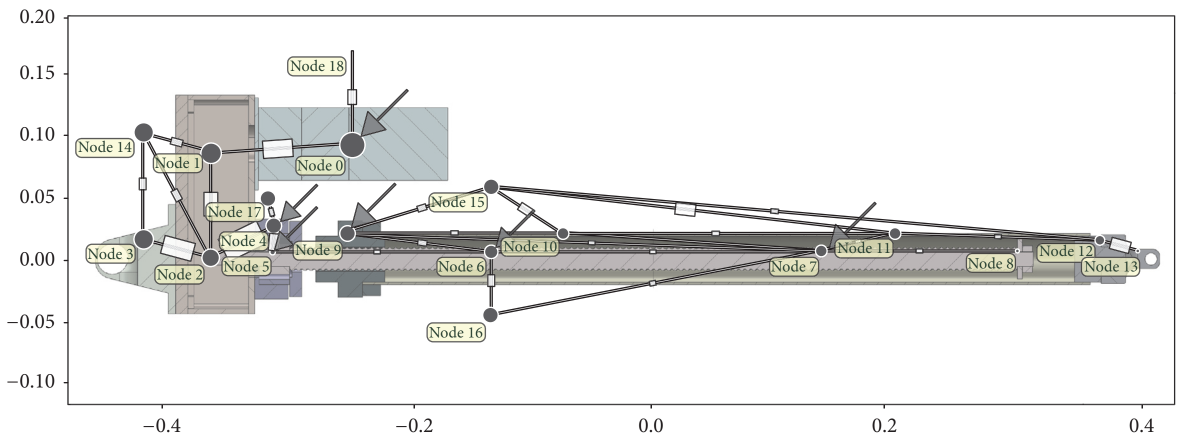


Figure 7: Electric equivalent circuit diagram of the actuator strut. “Resistors” represent thermal conductance. The circles are simplified visualizations for more complex electric circuits consisting of a parallel circuit of a current source (heat loss) and a (heat) capacity between the node and “electric ground.” The size of a circle correlates with the size of the capacity, the size of an arrow correlates with the size of the conductance.

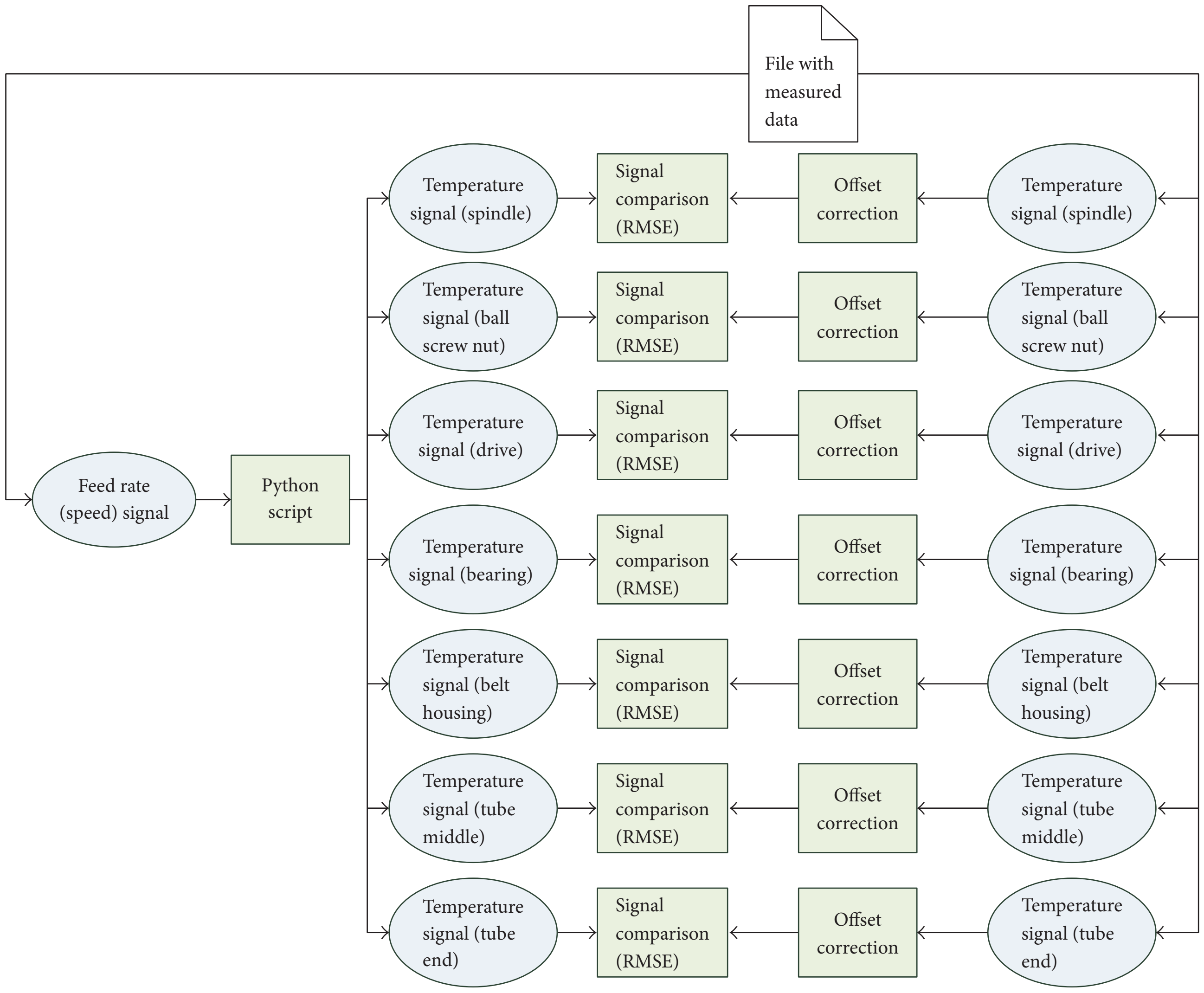

File with measured data
Feed rate (speed) signal
Python script
Temperature signal (spindle)
Temperature signal (ball screw nut)
Temperature signal (drive)
Temperature signal (bearing)
Temperature signal (belt housing)
Temperature signal (tube middle)
Temperature signal (tube end)
Signal comparison (RMSE)
Offset correction

Figure 8: Systems and signals for electric equivalent circuit as a Python script.

Table 2: Optimal parameters of the equivalent circuit model for different optimization goals (optimized for $v = 20$ m/min data). Optimization time 60 min per case and Monte Carlo optimization (about 7600–9200 iterations per case).

Optimized for sensor	r_n	r_b	r_{sa}	r_{ta}	r_m	SSE 20 m/min in K^2	RMSE 20 m/min in K	RMSE 25 m/min in K	RMSE 30 m/min in K
Spindle	1.13	1.68	1.25	1.26	0.96	1.21	0.13	2.07	7.30
Ball screw nut	0.88	1.21	1.16	1.25	1.23	1.90	0.16	1.83	7.30
Drive	0.86	1.02	1.17	1.06	1.24	48.52	0.81	4.59	6.35
Bearing	1.74	0.70	1.13	1.12	0.92	6.10	0.29	2.40	4.93
Belt housing	1.58	0.76	1.00	1.03	0.77	6.79	0.30	2.03	4.13
Tube middle	0.96	0.99	1.07	0.94	1.06	0.74	0.10	0.29	2.43
Tube end	0.61	1.75	0.85	0.99	1.15	5.48	0.27	0.45	0.79
Mean of all	0.89	0.71	0.88	1.14	1.16	268.12	1.73	2.34	4.79

the data of $v = 20$ m/min and the results of the unchanged model for $v = 25$ m/min and $v = 30$ m/min are given for comparison. The accuracy of the model is quite similar to the set of transfer functions. Especially regarding the validation cases ($v = 25$ m/min and $v = 30$ m/min) the differences are negligible. The reason is that both models are linear and thus suffer in an equal manner from the inability to model the nonlinear behavior of the physical process.

It is evident that although for each simulation the same model structure of the overall machine has been used, the optimal parameters are very different depending on the quantity to be optimized. However, it is often possible that different settings produce the same output—which is one reason for this diversity. Others are given later in this section.

As explained before, only 5 parameters have been optimized with optimization algorithms while in Section 4.1 for the set of transfer functions 21 parameters have been optimized. Nevertheless, in the transfer function model only 3 parameters have an influence on one output signal while in the equivalent circuit model all 5 parameters have an effect on all output signals. These can be interpreted as degrees of freedom. Due to the five parameters, there is more freedom in the equivalent circuit model for adjusting the signal behavior at a given point as with the three parameters of the FOPTD model. Therefore, the accuracy of the equivalent circuit model is mostly higher than the accuracy of the FOPTD models when optimized for one signal to be as close as possible to the appropriate measured signal. However, there are also several shortcomings of this model type:

(i) The effort for making the model structure, implementing the script, computing the start values of the 209 parameters, and identifying the 5 most unknown parameters is much higher.

(ii) The time for optimization is much higher (60 minutes for the Monte Carlo algorithm with the numerous Python script calls versus less than 1 second for the least squares method with the FOPTD models) since iterative optimization methods have to be used instead of the least squares method.

(iii) If not only one signal but the average of the RMSEs of all 7 measurement points is optimized, then the model is much worse than the set of independent FOLPD models. One reason is that in the set of transfer functions 21 parameters have been optimized but for the equivalent circuit model only 5. Additionally, the transfer functions are not coupled and can thus be optimized independently from each other while in the equivalent circuit model each parameter has an influence on every output signal.

(iv) While one advantage of the equivalent circuit model is the physical interpretability of the parameters (capacities, heat conductivities, and heat losses), it is a wrong conclusion that the optimizer would bring the parameter values to the real physical parameters. Instead, it chooses *any* values that minimize the difference between measurements and simulation outputs, even if they are not physically correct but a result of the simplified model structure (a finite number of mass points) or unmodeled disturbances. The extreme differences between the optimal values of the parameters for the different optimization criteria (measurement points) confirm that. A further confirmation of that problem is the poor model quality when not only one signal but the average of all RMSEs is optimized. The conclusion of this problem is that the advantage of the equivalent circuit model—where the parameters are physically interpretable—is only working if the model structure is sufficiently correct and all parameters can be computed without parameter identification techniques. As soon as parameter identification is used, the identified parameters have no better physical meaning than the parameters of the phenomenological FOPTD model.

Nevertheless, the choice of a model type always depends on the concrete application and the questions to be answered.

5. Conclusions

There are two main goals of this article. The first is the presentation of a higher level software for parameter identification that allows now the inclusion of models that have been realized with external software without the need for remodeling in the new software. In particular, this has been realized for Matlab and Python scripts, demonstrating the generic concept. Due to the plug-in-based architecture, it is possible to extend the software for coupling a lot of other simulation software without changing the core of the identification coordinator. Additionally, this allows reusing established simulation and modeling software with all their specific features.

The second goal of this article is the detailed presentation of a case study where two models of an actuator strut are compared with respect to parameter identification. A set of transfer functions has been compared to an equivalent circuit model. Both have their advantages and disadvantages regarding reachable accuracy, modeling effort, identification effort, and physical interpretability. Although with an equivalent circuit model a higher accuracy can be reached for single output signals, this results in less physical interpretability and less quality when all output signals are optimized. Both models have been identified using the new identification coordinator software such that the reading of measurement data and signal comparison "infrastructure" could be reused, saving time for parameter identification.

Appendix

Parameter Reduction for the Equivalent Circuit Model

This appendix gives some complementary details for Section 4.2.3 about the parameter reduction of the equivalent circuit model.

Many of the 209 parameters are known to be negligible, that is, approximately zero or infinity. This is the case for the heat conductance between nodes that are not directly connected as well as for the heat capacity of the air (that can be treated as infinity) and the power loss of nodes without friction. After excluding these parameters, the model contains only the remaining 14 heat capacities, 6 (speed-dependent) power losses, and 28 heat conductance parameters, that is, totally 48 parameters. The initial (theoretically computed) values of all parameters (that are not zero) can be found in Tables 3 and 4.

The number of parameters to be optimized (i.e., the search space) has been reduced significantly, but it is still too large for standard optimization methods. Therefore, the next step is to find parameters that can be computed from physical knowledge with relatively large accuracy.

The heat capacities of the nodes are relatively certain (±7% according to [4]), because they can be computed from the product of their mass and substance-specific heat storage capacity. If necessary, the mass can be computed from the product of the volume and the density; the volume can often be taken from a CAD model. For components that consist of only one material (substance), the computed capacities are very certain. Since the model structure represents the "real" physics (see (6)) only very roughly, this accuracy is sufficient as more accurate parameters would not result in a significantly more accurate description of the real thermal behavior. The heat capacity of the drive (motor) is more uncertain, because it consists of many parts of different substances, but the motor is less relevant for the thermoelastic deformation of the actuator strut, because it is only at one, relatively small, point connected to the strut; compare (5).

Table 3: Heat capacities and feed-rate-dependent heat losses of the nodes (computed or estimated without parameter identification).

Node number i	Meaning	Heat capacity C_i in J/K	Heat loss factor q_i in J/m
0	Drive	982.08	0.533
1	Belt housing drive	657.40	0
2	Belt housing spindle	694.97	0
3	Shoulder joint	435.41	0
4	Bearing seat	330.66	0.303
5	Spindle clamping end	50.73	0.454
6	Spindle lower part	253.65	0.306
7	Spindle upper part	253.65	0.306
8	Spindle free end	24.92	0
9	Ball screw nut	406.48	0.408
10	Tube lower part	243.51	0
11	Tube upper part	243.51	0
12	Tube joint end	179.43	0
13	Hand joint	14.35	0
14	Air belt housing	∞	0
15	Air tube	∞	0
16	Air spindle	∞	0
17	Air bearing	∞	0
18	Air drive	∞	0

The heat conductance g_{ij} is a priori less accurate than the heat capacities C_i, because an equivalent circuit model assumes each node to be a mass point and not a three-dimensionally extended object. For homogenous uniform heat exchange the equation

$$g = \lambda \cdot \frac{A}{l} \tag{A.1}$$

with distance l, (mean) cross-sectional area A, and thermal conductance λ holds. For more complex geometries, the approximations from [12] are used. The heat conductance between two steel components that are mounted together is relatively certain (about ±10% [4]).

Large uncertainties exist regarding conductance at rolling contacts (+100/−40%), conductance to the air (+30/−20% without forced convection), and power losses (+200/−70% for rolling contacts and ±25% for electric units like motors) [4]. The values of these parameters have been taken from [12].

The heat conductance between the spindle and the bearing or ball screw nut can be neglected as it is small compared to the heat loss due to friction at this positions. There are also some other uncertain conductance (heat conductance of motor to air, etc.) but their influence on the thermal deformation (thermal expansion of the strut) is much smaller.

TABLE 4: Heat conductance g_{ij}. All the other 162 combinations are not connected and treated as zero.

Number	Node i	Meaning of node i	Node j	Meaning of node j	Heat conductance g_{ij} in W/K
#0	0	Drive	1	Belt housing drive	1.97
#1	0	Drive	18	Air drive	0.38
#2	1	Belt housing drive	2	Belt housing spindle	1.41
#3	1	Belt housing drive	14	Air belt housing	0.27
#4	2	Belt housing spindle	3	Shoulder joint	2.74
#5	2	Belt housing spindle	4	Bearing seat	3.17
#6	2	Belt housing spindle	14	Air belt housing	0.25
#7	3	Shoulder joint	14	Air belt housing	0.25
#8	4	Bearing seat	5	Spindle clamping end	0.95
#9	4	Bearing seat	17	Air bearing	0.06
#10	5	Spindle clamping end	6	Spindle lower part	0.05
#11	6	Spindle lower part	7	Spindle upper part	0.03
#12	6	Spindle lower part	9	Ball screw nut	0.13
#13	6	Spindle lower part	10	Tube lower part	0.04
#14	6	Spindle lower part	16	Air spindle	0.25
#15	7	Spindle upper part	8	Spindle free end	0.05
#16	7	Spindle upper part	9	Ball screw nut	0.13
#17	7	Spindle upper part	10	Tube lower part	0.07
#18	7	Spindle upper part	11	Tube upper part	0.04
#19	7	Spindle upper part	16	Air spindle	0.06
#20	9	Ball screw nut	10	Tube lower part	0.10
#21	9	Ball screw nut	15	Air tube	0.19
#22	10	Tube lower part	11	Tube upper part	0.05
#23	10	Tube lower part	15	Air tube	0.45
#24	11	Tube upper part	12	Tube joint end	0.10
#25	11	Tube upper part	15	Air tube	0.45
#26	12	Tube joint end	13	Hand joint	0.83
#27	12	Tube joint end	15	Air tube	0.06

TABLE 5: Uncertain parameter influences.

Influenced model parameters	Physical parameter	Ratio variable
q_0	θ_m	r_m
q_6, q_7, q_9	θ_n	r_n
q_4, q_5	θ_b	r_b
$g_{\#14}, g_{\#19}$	θ_{sa}	r_{sa}
$g_{\#21}, g_{\#23}, g_{\#25}, g_{\#27}$	θ_{ta}	r_{ta}

Therefore, up to here the remaining parameters are 6 power losses and 6 heat conductance parameters, that is, totally 12 parameters. However, it is possible to reduce the number of parameters further.

Let $\boldsymbol{\theta}$ be the set of *physical* parameters and $\boldsymbol{\theta}_u \subset \boldsymbol{\theta}$ the set of significantly uncertain physical parameters. Also, let $\boldsymbol{\Theta}$ be the set of *model* parameters and $\boldsymbol{\Theta}_u \subset \boldsymbol{\Theta}$ the set of significantly uncertain model parameters; see Figure 9. Then, each model parameter $\Theta \in \boldsymbol{\Theta}$ is a function of $\boldsymbol{\theta}$ and also each unknown model parameter $\Theta \in \boldsymbol{\Theta}_u$ is a function of $\boldsymbol{\theta}$, but each known model parameter $\Theta \in (\boldsymbol{\Theta} \setminus \boldsymbol{\Theta}_u)$ is only a function of $(\boldsymbol{\theta} \setminus \boldsymbol{\theta}_u)$. This is relevant as it can help to reduce the number of parameters in an optimization task, if $|\boldsymbol{\theta}_u| < |\boldsymbol{\Theta}_u|$, which happens if several model parameters depend on the same physical parameter.

In the present example, there are five uncertain *physical* parameters that are represented by a variable θ: the motor heat loss θ_m, the heat loss between bearing and spindle θ_b, the heat loss between ball screw nut and spindle θ_n, the heat conductance θ_{sa} between spindle and air, and the heat conductance θ_{ta} between tube and air. Therefore, $\boldsymbol{\Theta}_u = \{r_m, r_n, r_b, r_{\text{sa}}, r_{\text{ta}}\}$ and $\boldsymbol{\theta}_u = \{q_0, q_4, q_5, q_6, q_7, q_9, g_{\#14}, g_{\#19}, g_{\#21}, g_{\#23}, g_{\#25}, g_{\#27}\}$. The relationship between the physical parameters and the model parameters is shown in Table 5 and Figure 10. If the model parameters can be computed from the physical parameters, it is enough to identify the 5 physical parameters instead of 12 model parameters.

Each physical parameter can be roughly estimated from theoretical backgrounds. This estimation is called θ_{estim}. Similarly, each model parameter can be estimated as Θ_{estim}. Assuming a proportionality between physical parameter and model parameter, the equation

$$r = \frac{\Theta_{\text{opt}}}{\Theta_{\text{estim}}} = \frac{\theta_{\text{opt}}}{\theta_{\text{estim}}} \qquad \text{(A.2)}$$

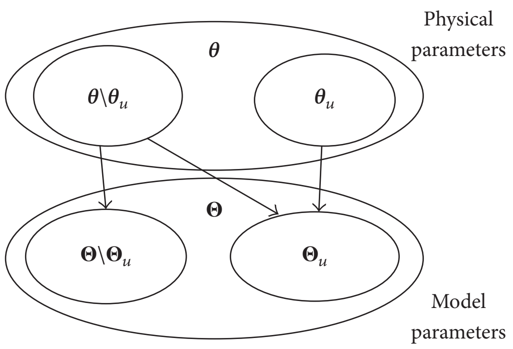


Figure 9: Parameter sets of known and unknown physical parameters and known and unknown model parameters.

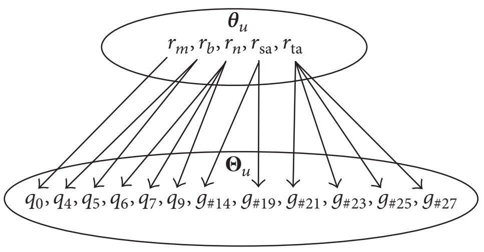

$\boldsymbol{\theta}_u$
$r_m, r_b, r_n, r_{sa}, r_{ta}$
$\boldsymbol{\Theta}_u$
$q_0, q_4, q_5, q_6, q_7, q_9, g_{\#14}, g_{\#19}, g_{\#21}, g_{\#23}, g_{\#25}, g_{\#27}$

Figure 10: Parameter sets of unknown physical parameters and unknown model parameters.

holds, where θ_{opt} and Θ_{opt} are the optimal values of the physical and model parameters, respectively, found by parameter identification. Therefore, the "ratio variable" r can be a replacement of the physical variable to be optimized, which has the advantage that it is 1 at the optimization beginning and the value of the physical parameter needs not to be computed explicitly. If $r = 1$ after optimization, the initial estimation was correct.

Thus, only 5 (of the original 209) parameters have to be identified by parameter identification techniques and all other parameters are either known with relatively high certainty or can be computed from these five "ratio parameters."

Notations

Abbreviations

FOPTD:	First order plus time delay
CAD:	Computer-aided design
PC:	Personal computer
RMSE:	Root-mean-square error
SSE:	Sum of squared errors.

Symbols

α:	Linear heat expansion coefficient
$\Delta\vartheta$:	Change of temperature
Δl:	Change of length
λ:	Thermal conductance
$\tilde{\tau}$:	Discrete time delay
$\vartheta, \vartheta_i, \vartheta_j$:	Temperature (of node i or j, resp.)
$\boldsymbol{\theta}$:	Set of physical parameters
θ_b:	Heat loss between bearing and spindle
θ_{estim}:	Estimated value of a physical parameter
θ_m:	Motor heat loss
θ_n:	Heat loss between ball screw nut and spindle
θ_{opt}:	Optimal values of a physical parameter
θ_{sa}:	Heat conductance between spindle and air
θ_{ta}:	Heat conductance between tube and air
$\boldsymbol{\theta}_u$:	Set of significantly uncertain physical parameters
$\boldsymbol{\Theta}$:	Set of model parameters
Θ_{estim}:	Estimated value of a model parameter
$\Theta_i(s)$:	Laplace transform of temperature ϑ_i
Θ_{opt}:	Optimal values of the model parameters
$\boldsymbol{\Theta}_u$:	Set of significantly uncertain model parameters
A:	(Mean) cross-sectional area
a_{m1}:	Coefficient of a discrete difference equation
b_{m0}:	Coefficient of a discrete difference equation
c:	Thermal diffusivity
C_i:	Heat capacity
e:	Error or Euler-Mascheroni constant
g, g_{ij}:	Heat conductance (between nodes i and j)
$G_d(z^{-1})$:	Discrete time transfer function
$G_i(s)$:	Continuous-time transfer function
i, j:	Row index in a vector
k:	Index of the sampling instant
K_i:	Proportional action coefficient
$\widehat{K}_i$:	Estimation of K_i
l:	(Spatial, initial) length or distance
L_i:	Time delay or dead time
M:	Number of time series in a vector
m_1:	Auxiliary variable
N:	Number of recorded time steps (sampling instants)
P:	Number of nodes in an electrical circuit model
q_i:	Velocity-dependent heat loss
$\dot{Q}_i$:	Heat flow
r, r_i:	Ratio variable
r_b:	Ratio parameter of bearing heat loss
r_m:	Ratio parameter of motor heat loss
r_n:	Ratio parameter of nut heat loss
r_{sa}:	Ratio parameter of θ_{sa}

r_{ta}:	Ratio parameter of θ_{ta}
$\underline{\text{RMSE}}$:	Vector of root-mean-square errors
RMSE_i:	ith element in $\underline{\text{RMSE}}$
s:	Laplace variable (complex frequency)
$\underline{\text{SSE}}$:	Vector of squared errors
SSE_i:	ith element in $\underline{\text{SSE}}$
t:	(Continuous) time
$[\cdot]^T$:	Transposed matrix or vector
T_i:	Time constant (first-order lag)
$\widehat{T}_i$:	Estimation of T_i
T_s:	Sampling period
v:	Feed-rate
$V(s)$:	Laplace transform of $v(t)$
$\underline{w}$:	Vector of weighting factors
x, y, z:	Coordinates
$\underline{y}(k)$:	Vector of M time series
$\underline{y}_{\text{model}}(k)$:	Vector of the outputs of a model
$\underline{y}_{\text{measurement}}(k)$:	Vector of measured time series
z:	Variable of the z-transform.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work has been funded by the German Research Foundation in the Collaborative Research Center Transregio 96.

References

[1] B. Hensel and K. Kabitzsch, "Generator for modular virtual sensors," in *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*, Berlin, Germany, 2016.

[2] https://de.mathworks.com/products/matlab.html.

[3] B. Hensel, T. Wagner, A. Gellrich, K. Kabitzsch, and B. Kauschinger, "Holistic ontology-based assistance system for efficient process model parameter identification," *Journal of Computational Engineering*, vol. 2015, pp. 1–14, 2015.

[4] B. Kauschinger and S. Schroeder, "Uncertain parameters in thermal machine-tool models and methods to design their metrological adjustment process," *Applied Mechanics and Materials*, vol. 794, pp. 379–386, 2015.

[5] T. S and P. Stoica, *System Identification*, Prentice Hall, Hemel Hempstead, UK, 1989.

[6] R. Isermann and M. Münchhof, *Identification of Dynamic Systems*, Springer, Berlin, Germany, 2011.

[7] M. Sanayei and P. Rohela, "Automated finite element model updating of full-scale structures with PARameter Identification System (PARIS)," *Advances in Engineering Software*, vol. 67, pp. 99–110, 2014.

[8] H. Wernsing and C. Büskens, "Parameter identification for finite element based models in dry machining applications," *Procedia CIRP*, vol. 31, pp. 328–333, 2015.

[9] https://de.mathworks.com/products/sysid.html.

[10] https://de.mathworks.com/help/matlab/matlab-engine-api-for-java.html.

[11] http://www.jython.org/.

[12] G. Jungnickel, *Simulation of thermal behavior of machine tools – Modelling and parameterization*, K. Großmann, Ed., Chair for Machine Tools, Dresden University of Technology, 2010.

[13] , *Control System Fundamentals*, W. S. Levine, Ed., CRC Press, Boca Raton, Fl, USA, 2000.

[14] K. J. Åström and B. Wittenmark, *Adaptive Control*, Reading: Addison-Wesley Publishing Company, 1989.

[15] S. J. Qin, "An overview of subspace identification," *Computers and Chemical Engineering*, vol. 30, no. 10-12, pp. 1502–1513, 2006.

[16] R. Ding and L. Zhuang, "Parameter and state estimator for state space models," *The Scientific World Journal*, vol. 2014, Article ID 106505, 2014.