An Artificial-Intelligence-Based Method to Automatically Create Interpretable Models from Data Targeting Embedded Control Applications

Jens S. Buchner [*] **Sebastian Boblest** [*] **Patrick Engel** [*]
Andrej Junginger [*] **Holger Ulmer** [*]

[*] *ETAS GmbH, Borsigstraße 24, 70469 Stuttgart,*
e-mail: jens.buchner@etas.com.

Abstract: The development of new automotive drivetrain layouts requires modeling of the involved components to allow for ideal control strategies. The creation of these models is both costly and challenging, specifically because interpretability, accuracy, and computational effort need to be balanced. In this study, a method is put forward which supports experts in the modeling process and in making an educated choice to balance these constraints. The method is based on the artificial intelligence technique of genetic programming. By solving a symbolic regression problem, it automatically identifies equation-based models from data. To address possible system complexities, data-based expressions like curves and maps can additionally be employed for the model identification. The performance of the method is demonstrated based on two examples: 1. Identification of a pure equation based model, demonstrating the benefit of interpretability. 2. Creation of a hybrid-model, combining a base equation with data-based expressions. Possible applications of the method are model creation, system identification, structural optimization, and model reduction. The existing implementation in ETAS ASCMO-MOCA also offers a high efficiency increase by combining and automating the two procedural steps of embedded function engineering and calibration.

Keywords: Nonlinear and optimal automotive control; Automotive system identification and modeling; Modeling, supervision, control and diagnosis of automotive systems.

1. INTRODUCTION

To develop control strategies, like, e.g., model-predictive control for components of physical systems, mathematical models describing these components are required. The industrial-scale application of these control strategies, by implementing them into embedded control units, consequently raises various constraints to the employed models. These constraints are accuracy, interpretability, computational effort, memory usage, calibration effort, and generalizability to similar systems.
Purely statistical modeling techniques, like neural networks or Gaussian process models are designed to robustly reach high modeling accuracy. The accuracy is defined by a suitable statistical metric like, for example, root mean squared error. Though employing the same type of metric, to learn equation-based models is less straight forward. This is mainly due to the symbolic representation of equations which requires to search for the most suitable expression in a large combinatorial and discrete space. In turn, the symbolic representation allows for an intuitive interpretation by experts. In addition fundamental dependencies of a system may be represented by equation-like expressions, e.g. like in the example shown in section 4.1. This can lead to a comparably small number of degrees of freedom to parameterize the model and with that it reduces the calibration effort and memory usage. For neural networks and Gaussian processes reduction of the computational efforts is typically addressed by post-processing techniques like pruning and model compression, respectively. In the method put forward in this study, reducing the computational efforts is addressed mainly by three approaches which are explained in detail in the referenced sections: First, by choosing functions of low complexity (section 3.1). Second, by penalizing computational complexity (section 3.6). Third, by allowing for an educated choice from a set of Pareto-optimal models (section 2.5).
However, an accurate representation of all effects which contribute significantly to the target quantity to-be-modeled is crucial. For this, purely equation-based models may not be suited to reach the required accuracy while preserving interpretability. Hence, this may raise the need for the usage of hybrid-model structures which extend the equation-based models by data-based parts (e.g. as employed by Sequenz et al. (2012)).
To validate a model's accuracy, an appropriately measured dataset of the modeled system is required in any case. In this study, the possibility to automatically create models meeting the above constraints from a measured dataset is demonstrated employing artificial intelligence techniques. To reach the goal of equation-based models, the underlying problem to be solved is defined as the symbolic-regression problem. The applied method, however, needs

to be open to create hybrid models as well. Symbolic regression problems can be solved by various approaches (Dong et al. (2015)). Several methods show promising results by focusing purely on equation-based models. McConaghy (2011) employs pathwise regularized learning to prune a huge set of candidate basis functions down to compact models, while Brunton et al. (2016) combined sparsity-promoting techniques with machine learning. In turn, Horesh et al. (2016) and Austel et al. (2017) use mixed-integer non-linear programming to minimize complexity along with solving the symbolic regression problem. A technique, which is well suited for addressing problems which can be solved by solutions of a clearly confined functional space was made available by Sahoo et al. (2018). The authors embed symbolic equations into a neural network framework to enable the application of gradient based training methods. Recently, Iten et al. (2020) and Udrescu and Tegmark (2020) addressed the symbolic regression problem by employing neural networks helping to identify system properties like symmetries. For the given challenge of using one method to identify purely equation-based models along with hybrid-models, the artificial intelligence technique of genetic programming was identified to be most appropriate. Since Koza (1992) put forward a seminal book on genetic programming, ongoing research has been made. Most prominently Schmidt and Lipson (2009) addressed problems with dependent multi-dimensional outputs by defining cost-functions based on partial derivatives. By that the authors demonstrated the applicability of genetic programming for the identification of conservation laws from data. Lately, Martins et al. (2018) addressed the problem of exponential growth and Iba et al. (2018) combined genetic programming with relevance vector machines to identify solutions based on linear combination of basis functions.
The algorithm (section 3) employed by the method (section 2) described in this study extends the method of Koza (1992) by two major improvements. First, being inspired by Topchy and Punch (2001) a second-order local optimization method was introduced. Second, data-based model types – represented by SGR models (Priber (2003)) – are employed to allow for the creation of hybrid models. In section 4 the performance of the method is demonstrated in two real-world control engineering applications.

2. METHOD

To automatically create models from data, employing the method demonstrated in this study, involves a workflow which is described in the following.

2.1 Data Acquisition

To create a model for a specific system of interest, an appropriate dataset has to be acquired by a measurement. The dataset has to include the quantity of interest characterizing the system's behavior (output) along with all quantities influencing the latter (inputs). The dataset requires a statistically sufficient representation of the system, which can for instance be achieved by a design-of-experiment. The measurement needs to cover all operation points which are to be represented by the model. Additionally, the input quantities should be varied such that all significant variations in the observed system's output are taken into account. The dataset size should allow for a meaningful split into training and test data. The current implementation only fully supports static measurements.

2.2 Data Import and Preparations

The acquired data are prepared such that they are: (i) Clean: All values are plausible, no not-a-number values are given, and no errors from defect measurement devices are included. (ii) Labeled: For all values included in the data the actually measured quantity (also called label) is defined along with its units. (iii) Split: The dataset is split into a training and test dataset, with both subsets still fulfilling the mentioned requirements.

2.3 Problem Statement and Algorithm Configuration

First, the actual symbolic regression problem is defined by setting the target (output) quantity. Additionally, prior knowledge is employed by the selection of all input quantities which are expected to have a possibly significant influence on the output quantity. Second, the algorithm (section 3) is configured by choosing it's settings.

2.4 Algorithmic Execution

The symbolic regression problem is solved by executing the algorithm as depicted in Fig. 1. The iterative process is carried out until a predefined termination criterion is reached, or a user interrupts. The latter may occur on the basis of the results of the last iteration.

2.5 Model Choice

Once the algorithm is finished, its results are made available. These results are the models which are part of the Pareto front defined in the metric space spanned by the models' *Fitness Method* and complexity (section 3.6). By selecting a model from the Pareto front, a user can make an educated choice involving system knowledge and can balance the model accuracy and complexity.

2.6 Statistical Analysis

The performance of the model is assessed by carrying out a statistical analysis of data residuals. This means the accordance of data obtained from the measurement and from the model-prediction is evaluated using the *Fitness Method* and linear regression analysis. This is carried out both for the training data and the test data to judge on the optimization result, to exclude overfitting, and assess the generalization behavior of the model in the input quantity space.

2.7 Structural and Semantic Analysis

Human interpretability of the results is a structural advantage of symbolic regression methods compared to purely statistical modeling techniques (such as Gaussian processes or artificial neural networks). This step provides the possibility to make use of this advantage and to give detailed inside in the modeled system's behavior. The

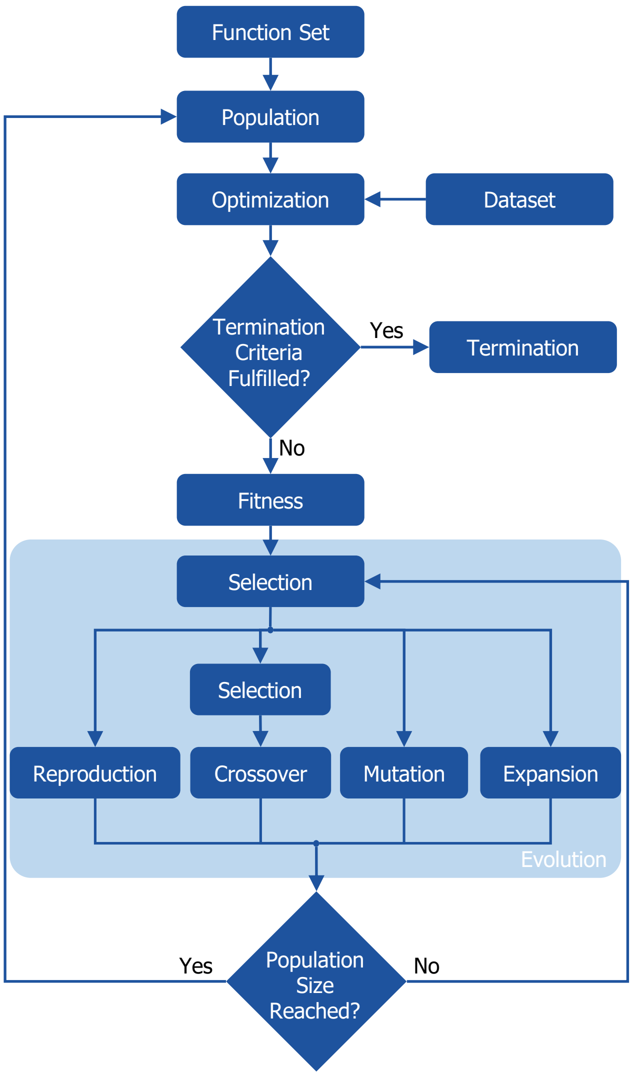

Fig. 1. Algorithm: Flow-chart depicting the main algorithmic processing steps.

model is analyzed both on a structural and semantic level, by following the actual computational steps and by making sense of the symbolic expressions, respectively. The same is carried out for the parameters involved in the model, by assessing their actual numeric values and by graphical inspections of curves and maps.

3. ALGORITHM

To solve the symbolic regression problem as stated (section 2.3), the algorithm described in this section is employed. It is a modified version of genetic programming (Koza (1992); Schmidt and Lipson (2009)), which was extended by two main aspects: First by employing data-based function types such as curves and maps and second by carrying out a second-order local optimization procedure. The major algorithmic steps are described below and depicted in Fig. 1. Being used at many points during the course of the algorithm, stochastic selection is fundamental to the algorithm. The implementation realizes stochastic selection deterministically to ensure reproducibility of algorithmic results. However, the underlying probability distributions are still sampled as they are defined. If not mentioned differently, this distributions are uniform distributions defined on a corresponding set of values. Depending on the context, this set is either real- or integer-valued.

3.1 Function Set

The algorithm relies on a set of functional expressions (Tab. 1) which are employed during the course of the algorithm. The noted curves and maps are realized in the implementation by SGR models (Priber (2003)). With that, the interpolated values of a curve or map are given by its parameters $a_1, ..., a_m$. The user brings-in prior- and system-knowledge by selecting a subset of the functions.

Table 1. Functional expressions being employed. x and y represent the input values, respectively. In case a_i are given, they are constant parameters of each function which are degrees of freedom with respect to the local optimization (section 3.4).

Functional Expressions	Complexity
$x+y$, $x-y$, $x*y$, x/y	2
x^2, $\sqrt{\lvert x\rvert}$	5
$\lvert x\rvert^{a_1}$	8
$\sin(a_1x)$, $\cos(a_1x)$, $\tan(a_1x)$	8
$\exp(a_1x)$	8
$\log(\lvert x\rvert)$	5
$\min(x,y)$, $\max(x,y)$	2
$\text{gaussian}(x,a_1,a_2)$	11
$\text{sigmoid}(x)$	5
1D Curve with 10 sampling points	20
2D Map with 3x3 sampling points	18
2D Map with 5x5 sampling points	50

3.2 Population

Based on the defined function set and the related parameters, a population – a set of models having *Population Size* elements – is created on a stochastic basis.
Starting with this step, the models are represented as directed graphs with a tree structure (Augusto and Barbosa (2000)). While the top node of each graph defines its output, the non-terminal nodes are in general given by the functions of Tab. 1. The terminal nodes are either constant parameters or input quantities.
The graph creation process is carried by the different methods listed below and can optionally be strictly constrained by the graphs' properties depth, size, and complexity.
In the following each model in the population is denoted as

$$F_i(\boldsymbol{a_i}, \boldsymbol{x}_1, ..., \boldsymbol{x}_{n_i}). \tag{1}$$

With i indexing the population members, $\boldsymbol{a_i}$ representing the parameters associated with this member, $\boldsymbol{x}_j$ representing the values for input j, and n_i being the number of inputs of the given graph. During the course of the algorithm the population evolves from the evolution operations (section 3.8). Each iteration step in the main loop of the algorithm (Fig. 1) corresponds to one generation.
Corresponding settings with default values in brackets:
Population Size: Number of models contained in each population. [100]
Population Creation Method: The graphs in the initial population are created with one of the following three

methods. [*half-and-half*]: (i) *full*: Each graph is created such that it has a depth of *Population Init Depth Max* in all branches. (ii) *grow*: Each graph is created such that at least one branch has a depth of *Population Init Depth Max.* (iii) *half-and-half*: One half of the population is created with full method and the other half of the population is created with the grow method. The employed maximum depth value is determined from a uniform distribution within *Population Init Depth Min* and *Population Init Depth Max*, for each graph individually.
Population Init Depth Min: Minimum init. depth. [1]
Population Init Depth Max: Maximum init. depth. [3]
Max Program Complexity: No graphs are created (initially and during evolution) having a larger complexity. A graphs' complexity is an integer given by the sum of complexities of all nodes (Tab. 1) the graph is made of. [not employed]
Max Program Depth: No graphs are created (initially and during evolution) having a larger depth. [not employed]
Max Program Size: No graphs are created (at any time) having a larger size (number of nodes). [50]

3.3 Dataset

The training dataset is incorporated for the computation of each graph's output. The dataset consists of m inputs $\boldsymbol{x}_1, ..., \boldsymbol{x}_m$ and the output y_{meas}. Each model output $y_{\mathrm{mod},i}$ is computed by

$$y_{\mathrm{mod},i} = F_i(\boldsymbol{a_i}, \boldsymbol{x}_1, ..., \boldsymbol{x}_{n_i}). \tag{2}$$

It has to be mentioned that the inputs employed for an individual model $\boldsymbol{x}_1, ..., \boldsymbol{x}_{n_i}$ are a subset of the inputs contained in the dataset.

3.4 Optimization

The output values of each graph are fitted to the measured output values. This is reached by solving the local optimization problem

$$\min_{\boldsymbol{a_i}} \mathcal{OF}_i(y_{\mathrm{mod},i}, y_{\mathrm{meas}}). \tag{3}$$

$\mathcal{OF}_i$ denotes the objective function defined in equation (5). Topchy and Punch (2001) employed a first-order optimization procedure to increase the efficiency of genetic programming for symbolic regression tasks. The algorithm presented in this paper allows to use various second-order optimization techniques as listed below. All employed optimization techniques are iterative techniques. They are either carried out until their specific convergence criteria is fulfilled or a maximum number of iterations is reached. As a matter of principle, the values in $\boldsymbol{a_i}$ are modified during the course of the local optimization procedure. If curves or maps (Tab. 1) are employed, this necessarily leads to a potential readjustment of their input value range and consequently their support points. Hence, the latter are redefined in each iteration to equidistantly sample the input values' range.
Corresponding settings with default values in brackets:
Optimizer Method: Local optimization method [Levenberg-Marquardt]: Levenberg-Marquardt method, Trust-region reflective, dogbox algorithm.
Optimizer Iterations: Maximum number of iterations the local optimization should be carried out. [10]

3.5 Termination criteria fulfilled?

The algorithm stops if one of the following criteria holds. For one of the models in the population the value obtained by the $\mathcal{OF}_i(y_{\mathrm{mod},i}, y_{\mathrm{meas}})$ is smaller than the termination threshold. The maximum number of generations is reached.
Corresponding settings with default values in brackets:
Termination Threshold: Threshold employed for termination. [not employed]
Number of Generations: Max. number of generations. [50]

3.6 Fitness

For each model in the population its fitness $\mathcal{F}_i$ is computed as

$$\mathcal{F}_i(F_i, y_{\mathrm{meas}}) = \frac{1}{1 + \mathcal{F}_i^*(F_i, y_{\mathrm{meas}})}. \tag{4}$$

The raw-fitness $\mathcal{F}_i^*$ is defined as:

$$\mathcal{F}_i^*(F_i, y_{\mathrm{meas}}) = \mathcal{OF}_i(y_{\mathrm{mod},i}, y_{\mathrm{meas}}) + \mathcal{P}_i(F_i), \tag{5}$$

with the objective-function $\mathcal{OF}_i(y_{\mathrm{mod},i}, y_{\mathrm{meas}})$ given by the different fitness methods listed below, and the penalty

$$\mathcal{P}_i(F_i) = p\mathcal{C}_i \tag{6}$$

Here, p represents the *Parsimony Coefficient* and $\mathcal{C}_i$ is the graph's complexity. $\mathcal{F}_i$ is a crucial quantity in the following steps, hence it needs to be emphasized that the meaning of a better fitness is related to a higher value of $\mathcal{F}_i$.
Corresponding settings with default values in brackets:
Fitness Method: objective-function employed for fitness computation. [RMSE]: RMSE: Root mean squared error. MSE: Mean squared error. ABS: Normalized absolute difference, which is the mean of the L^1-norm.
Parsimony Coefficient: Factor to weight the impact of the complexity penalty. [1]

3.7 Program Selection

The actual structural optimization of the graphs starts with selecting graphs from the population. The selection is done on the basis of $\mathcal{F}_i$, with one of the following methods:
Tournament: From the given population *Tournament Size* graphs are randomly drawn and the one with the lowest fitness is selected.
Fitness-based: A probability distribution is derived from the fitness of all graphs in the current population [Koza (1992)]. This distribution is sampled to select graphs. The fitter a graph is, the more likely it will be selected.
Greedy Overselection: The population is divided into a high-fitness group and a low-fitness group. For both groups fitness-based selection is used. The high-fitness group is used with *Probability Top*. Else, the low-fitness group is chosen.
Multi-Tournament: This method works as Tournament. With a selectable probability, however, $\mathcal{F}_i/\mathcal{C}_i$ is used as a selection criterion instead of just $\mathcal{F}_i$.
Program Selection Method: As described above. [*Tournament*]
Tournament Size: Number of programs to select for a tournament. [5]
Fraction Top: The fraction of the population defines the size of the high-fitness group. The group is filled with the graphs having the highest fitness. [0.1]

Probability Top: Probability to employ the high-fitness group for fitness-based selection. [0.9]
Probability Fitness: Probability to employ $\mathcal{F}_i/\mathcal{C}_i$ instead of $\mathcal{F}_i$. [0.5]

3.8 Evolution

The evolution step comprises the graph-modification operations of reproduction, expansion, mutation, and crossover. It is the actual structural optimization step and is carried out with the graph selected in the previous step. The probabilities for the different techniques determine how likely they are to be applied to this graph.
Corresponding settings with default values in brackets:
Reproduction Probability: Probability to carry out the reproduction operation. [0.05]
Expansion Probability: Probability to carry out the expansion operation. [0.05]
Mutation Probability: Probability to carry out the mutation operation. [0.05]
Crossover Probability: Probability to carry out the crossover operation. [0.85]

Reproduction Reproduction of a graph is taking it over as-is to the population of the next generation. This operation ensures that graphs with a good fitness are transferred unchanged to the next generation.

Expansion This relatively new method [Islam. et al. (2018)] is carried out by repeatedly (i) randomly selecting a terminal node of the graph, (ii) creating a new random graph with depth two, (iii) replacing the terminal node by the new graph. These steps are repeated for a fixed number of times. If at any time the expanded graph has a better fitness than the best program in the population, it is taken over to the next generation. Else, the original graph is taken over as-is.

Mutation A mutation of a graph is carried out with three different operations, each of them being applied to a randomly selected node of the graph: 1. Point: The selected node is replaced by a random node with the same number of inputs. 2. Hoist: The selected node of the graph is replaced by a sub-tree of itself. 3. New Tree: The selected node and all sub-nodes are replaced by a newly created graph with a maximum depth of three. How likely which method is applied is determined by the individual probability listed below.
Corresponding settings with default values in brackets:
New Tree: Probability for the new tree operation. [0.5]
Hoist: Probability for the hoist mutation operation. [0.25]
Point: Probability for the point mutation operation. [0.25]

Crossover The crossover operation is the recombination of two graphs to find an even fitter one. The operation is carried out by the following steps: (i) The previously selected graph is defined as the target graph. (ii) A second graph - the source graph - is selected exactly as described in section 3.7. (iii) In both graphs, one node with all its sub-nodes is selected randomly as a branch to-be-exchanged. (iv) The branch in the target graph is replaced by the branch of the source graph. The resulting graph is taken over to the population of the next generation.

3.9 Reached Population Size?

The evolution operations are applied until the population of the next generation has *Population Size* number of members.

3.10 Implementation

The method and workflow as described in section 2 and 3 has been implemented in ETAS ASCMO-MOCA. The latter is a software product of ETAS which is delivered as a desktop application. The basic functional scope of ASCMO-MOCA is to enable efficient parameter optimization of physical models, which are employed in control applications. This functionality has now been extended by the method described here. For this study the latest version of the tool (V5.4 Beta 1, refresh 17, build 42754) has been employed. All computations have been carried out on a standard laptop computer (Intel(R) Core(TM) i7-8650 CPU @ 1.90GHz, 32 GB RAM) on a single core.

4. RESULTS

In the following, the method described in section 2 is applied to two exemplary cases: first, modeling a power-loss of an electric machine (section 4.1) and second creating a torque-model of a gasoline engine (section 4.2). Both cases are real engineering applications which involve specific system properties to be taken into account and thus gave rise to different solution approaches.

4.1 Power-Loss Model of an Electric Machine

With increasing maturity, the control of electric machines in automotive powertrains requires to consider even higher-order effects influencing the system's performance. Such effects are, for instance, power-losses involved with different components of the system, e.g., the winding head in this case. Additionally, upper latency limits in the range of micro-seconds demand models which are minimal with respect to computational effort and memory demand.

System Setup For the given system, measurement of stationary operation points have be carried out. The quantities listed in Tab. 2 have been observed by equidistant variations within their range of operation. The dataset

Table 2. Quantities observed for the power loss of an electric machine.

Quantity	Description	Unit	Min.	Max.	Type
I_{D}	Direct Current	A	-525	-1	In
I_{Q}	Quadratic Current	A	1	525	In
T_{rtr}	Rotor Temperature	°C	-20	150	In
n_{mtr}	Motor Speed	rpm	1	16000	In
P_{mod}	Power loss	W	0	4558	Out

contains 24684 datapoints which have been acquired. The dataset was split into 10% training and 90 % testing data, due to a significant high redundancy in the data. The splitting was done by randomly selecting datapoints.

Algorithmic Settings and Execution Compared to the default algorithmic settings listed in section 3, the following settings have been changed. The employed functions were $x + y$, $x - y$, $x * y$, x/y, $\sqrt{|x|}$, $|x|^{a_1}$, $\log(|x|)$, and $\exp(a_1 x)$. *Population Size* was set to 1000 and *Parsimony Coefficient* to 0.1. After 6 iterations corresponding to 13 minutes of computation, the model provided in (7) was chosen from the Pareto-front. The RMSE for the training dataset is 0.029 W. The total amount of time spend for model creation and analysis was $\approx$ 1 h.

Resulting Model The model which was identified is

$$P_{\mathrm{mod}} = a_1 I_{\mathrm{D}}^2 + a_2(a_3 I_{\mathrm{Q}} + I_{\mathrm{Q}}^2) \tag{7}$$

with the corresponding parameters listed in Tab. 3. Equation (7) obviously fulfills the required property of being interpretable. For instance, the absence of n_{mtr} and

Table 3. Parameter values according to (7).

Parameter	Value
a_1	0.0083
a_2	0.0083
a_3	0.0354

P_{mod} directly determines their insignificance with respect to P_{mod}. Additionally, by only having three calibration parameters (Tab. 3), it is memory efficient and fast to calibrate in future applications. Since the expression can be computed only with few arithmetic operations it is also computationally efficient.

Model Performance The accuracy of equation (7) was evaluated by comparing y_{meas} and $y_{\mathrm{mod},i}$ for the testing data. The result is depicted in Fig. 2, demonstrating that the accuracy achieved with the training data is preserved for the testing data. Comparing the RMSE of 0.029 W to the value range of P_{mod} results in a relative model accuracy of 0.00064%.

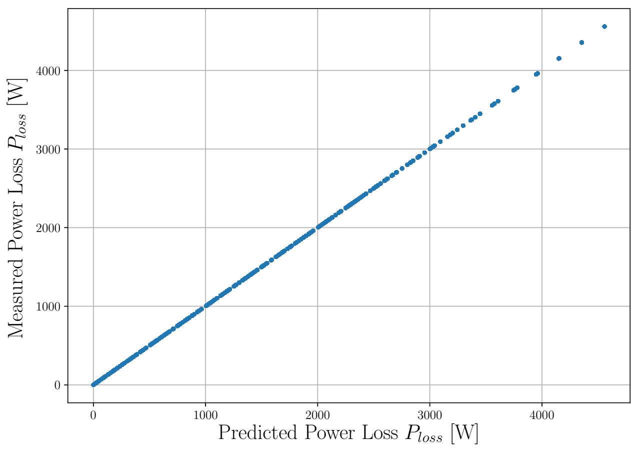

Fig. 2. Regression plot comparing measured y_{meas} and the modeled output $y_{\mathrm{mod},i}$ obtained from (7) for the testing data. The RMSE is 0.029 W with a coefficient of determination r^2 of 0.9999999.

4.2 Torque Model of an Internal Combustion Engine

The ongoing improvement of existing powertrain technologies, like internal combustion engines, naturally leads to an increase in complexity of the models employed for the associated control strategies. This manifests both in the models' structure and in the number of calibration parameters. Each increasing costs per-piece and development costs respectively, when using the models in embedded controllers. Thus, reducing the complexity is of interest in this context. The suitability of the given method for this task is demonstrated on the example of a torque model of a gasoline engine.

System Setup 2813 measurements at stationary operation points have been carried out for the quantities listed in Tab. 4, with equidistant variations in their range of operation. The resulting dataset was split into 70% training and 30% testing data, employing random selection.

Table 4. Quantities observed for the torque model of an internal combustion engine.

Quantity	Description	Unit	Min.	Max.	Type
n_{eng}	Engine Speed	rpm	597	5997	In
r_{l}	Relative Load	%	13	86	In
φ_{ign}	Ignition Angle	°Crank	-27	61	In
M_{eng}	Engine Torque	Nm	-55	310	Out

Algorithmic Settings and Execution The following algorithm settings were changed from their default values (section 3). For the given system abstraction and inputs, a purely equations-based description was not expected to be suitable. Hence, the employed functions were set as $x+y$, $x-y$, $x*y$, x/y, 1D Map with 10 sampling points, 2D Map with 5x5 sampling points. *Population Size* was set to 15, *Parsimony Coefficient* to 0.1, *Max Program Complexity* to 200, *Termination Threshold* to 1 Nm, and *Population Init Depth Max* to 5. After 10 iterations corresponding to 12 minutes of computation the model provided in Fig. 3 was identified from the Pareto-front, having an RMSE of 1.15 Nm for the training dataset. The total amount of time spent for model creation and analysis was about 1.5 h.

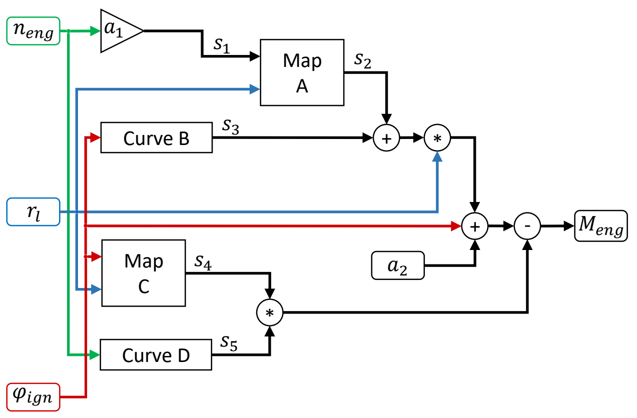

Fig. 3. Causal model of M_{eng}, which has been identified by the algorithm.

Resulting Model The model which has been identified by the algorithm can directly be translated to a causal model as sketched in Fig. 3. The values of scalar parameters included in this model are listed in Tab. 5. The corresponding maps and curves are depicted in Fig. 4, 5, 6,

Table 5. Parameter values according to Fig. 3.

Parameter	Value
a_1	6.773
a_2	-79.18

and 7. The shown curves and maps are obtained directly from the optimization procedure without any manual readjustment. All maps are widely found to be smooth and locally monotonic and by that fulfill a requirement, which is highly rated by control engineers (e.g. Sequenz et al. (2012)). It needs to be stressed that this property was achieved without applying any smoothness regularization. The penalization of complexity in equation (5) enforces the complexity of the system description to be as low as possible under the given constraints. Hence, redundancy in the system description is reduced. For curves or maps the smoothness emerges as a property of the system part which is represented by it.
Though not being as concrete as (7), the model in Fig. 3 still offers the interpretation advantages which are associated with causal models. This holds for the computational structure, the semantic meaning of nodes, as well as the parameter values.

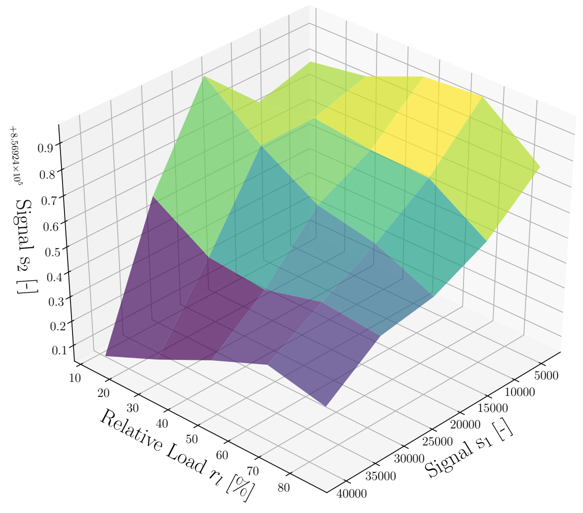

Fig. 4. Plot of Map A as employed in the torque model (Fig. 3).

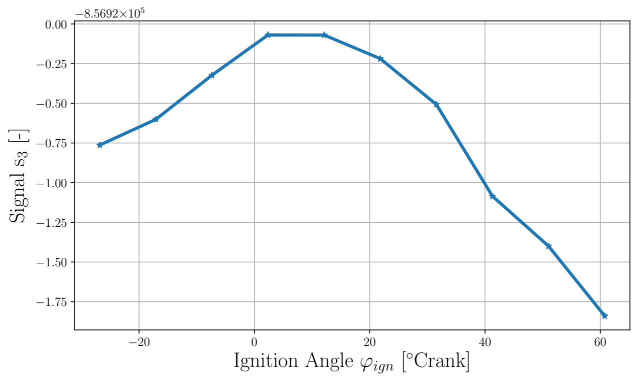

Fig. 5. Plot of Curve B as employed in the torque model (Fig. 3).

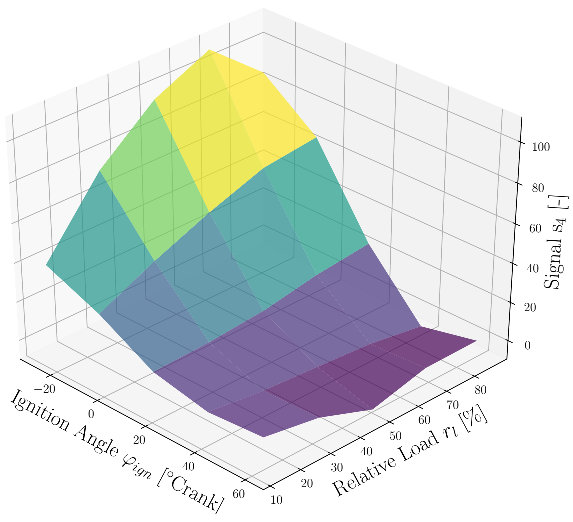

Fig. 6. Plot of Map C as employed in the torque model (Fig. 3).

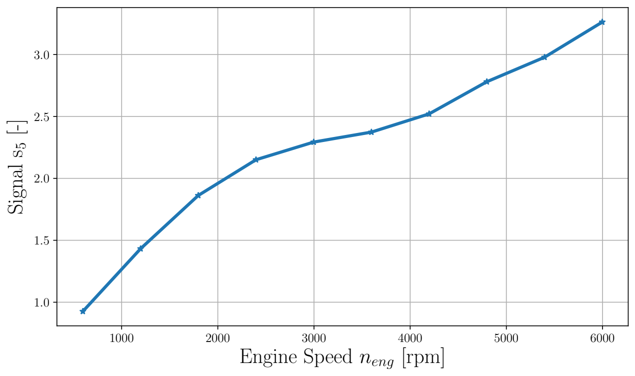

Fig. 7. Plot of Curve D as employed in the torque model (Fig. 3).

Model Performance The torque-model's accuracy was assessed by comparing y_{meas} and $y_{\text{mod},i}$ for the testing data (Fig. 8). Both the RMSE and r^2 lead to the conclusion that the system is reliably represented by the model in Fig. 3.

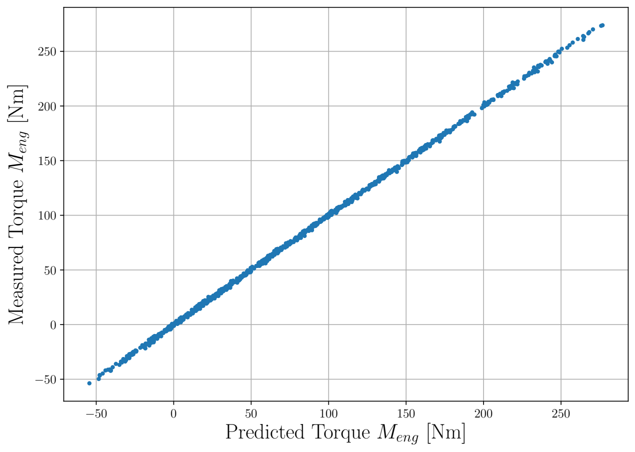

Fig. 8. Regression plot comparing the measured testing data and the prediction by Fig. 3. The RMSE is 1.19 Nm with a coefficient of determination r^2 of 0.9998.

A reference model is delivered along with ASCMO-MOCA which achieved the same accuracy for the given dataset. This model, however, has 118 calibration parameters compared to 72 in Fig. 3, corresponding to a reduction of degrees of freedom by 39%.

5. CONCLUSION AND OUTLOOK

In this study a novel method to automatically create models from data is described. The method automatizes the steps of control function engineering and calibration at once, while preserving the possibility to bring-in system expert knowledge at decisive points. Hence, it is specifically suited for embedded control applications.
The feasibility of the approach is successfully demonstrated with two examples. First, by creating an equation-based model which is able to describe the system in a compact manner with 0.00064% accuracy. Second, by identifying a causal model which leads to results having the same accuracy as a reference model, along with a significant reduction of calibration parameters. The computation times and engineering efforts involved with the two examples demonstrate the efficiency gain raised by employing the method.
Although stemming from real-world engineering problems, the examples are small with respect to the number of input quantities and dataset size. The algorithmic design determines that addressing larger problems, in the previous sense, causes higher computation time and more development efforts. This, however, also holds for other – both manual and automated – modeling attempts. The given method also offers straight-forward options for parallel computation which can reduce the drawback of high computation times. Besides creating models for predictive control, possible applications of the method are model compression, model run-time optimization, model structure optimization, system identification, model reduction, and data analytics.
Enhancement of the method may be achieved by improving the complexity estimation for the different function types by using empirical results of the respective computation efforts for a specific embedded device. Additionally, extending the method to time-series data, along with including derivative operators and time-shifts to the function set, is mandatory for creating models of dynamic systems. Most promising tackling generalization by employing various datasets can significantly increase the benefit created by this method.

ACKNOWLEDGEMENTS

We would like to thank Markus Hanselmann, Thilo Strauß, Matthias Werner, Fabian Jansen, and the ASCMO team for the fruitful discussions.

REFERENCES

Augusto, D.A. and Barbosa, H.J.C. (2000). Symbolic regression via genetic programming. In *Proc. on the 6th Brazilian Symp. on Neur. Networks*, volume 1, 173–178.

Austel, V., Dash, S., Gunluk, O., Horesh, L., Liberti, L., Nannicini, G., and Schieber, B. (2017). Globally optimal symbolic regression. In *Interpretable ML Symposium, 31st Conference on Neural Information Processing Systems (NIPS 2017)*, 1–6.

Brunton, S.L., Proctor, J.L., and Kutz, J.N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15), 3932–3937.

Dong, X., Dong, W., Yi, Y., Wang, Y., and Xu, X. (2015). The recent developments and comparative analysis of neural network and evolutionary algorithms for solving symbolic regression. In *Intelligent Computing Theories and Methodologies*, 703–714. Springer International Publishing, Cham.

Horesh, L., Liberti, L., and Avron, H. (2016). Globally optimal MINLP formulation for symbolic regression. Technical report, IBM Research Division.

Iba, H., Feng, J., and Izadi Rad, H. (2018). GP-RVM: Genetic programing-based symbolic regression using relevance vector machine. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 255–262.

Islam., M., Kharma., N., and Grogono., P. (2018). Expansion: A novel mutation operator for genetic programming. In *Proceedings of the 10th International Joint Conference on Computational Intelligence - Volume 1: IJCCI,*, 55–66. INSTICC, SciTePress.

Iten, R., Metger, T., Wilming, H., del Rio, L., and Renner, R. (2020). Discovering physical concepts with neural networks. *Physical Review Letters*, 124, 010508.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Martins, J.F.B.S., Oliveira, L.O.V.B., Miranda, L.F., Casadei, F., and Pappa, G.L. (2018). Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, 1151–1158. ACM, New York, NY, USA.

McConaghy, T. (2011). *FFX: Fast, Scalable, Deterministic Symbolic Regression Technology*, volume Genetic Programming Theory and Practice IX of *Genetic and Evolutionary Computation Series*, chapter 13, 235–260. Springer New York, New York, NY.

Priber, U. (2003). Smoothed grid regression. In *13. Workshop Fuzzy Systeme 2003*, 159–172.

Sahoo, S.S., Lampert, C.H., and Martius, G. (2018). Learning equations for extrapolation and control. In *Proc. 35th International Conference on Machine Learning, ICML 2018*, volume 80, 4442–4450. PMLR.

Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923), 81–85.

Sequenz, H., Keller, K., and Isermann, R. (2012). Zur Identifikation mehrdimensionaler Kennfelder für Verbrennungsmotoren. *Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs- und Informationstechnik*, 60(6), 344–351.

Topchy, A. and Punch, W.F. (2001). Faster genetic programming based on local gradient search of numeric leaf values. In *Proc. of the 3rd Annual Conf. on Genetic and Evolutionary Comp.*, GECCO'01, 155–162. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Udrescu, S.M. and Tegmark, M. (2020). Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16).