2018-01-1458

Automatic Calibrations Generation for Powertrain Controllers Using MapleSim

Abstract

Modern powertrains are highly complex systems whose development requires careful tuning of hundreds of parameters, called calibrations. These calibrations determine essential vehicle attributes such as performance, dynamics, fuel consumption, emissions, noise, vibrations, harshness, etc. This paper presents a methodology for automatic generation of calibrations for a powertrain-abstraction software module within the powertrain software of hybrid electric vehicles. This module hides the underlying powertrain architecture from the remaining powertrain software. The module encodes the powertrain's torque-speed equations as calibrations. The methodology commences with modeling the powertrain in MapleSim, a multi-domain modeling and simulation tool. Then, the underlying mathematical representation of the modeled powertrain is generated from the MapleSim model using Maple, MapleSim's symbolic engine. Maple is further used to manipulate the powertrain equations to produce the representation required for calibrations extraction. The methodology has been applied successfully in a research project with a large automotive OEM (Original Equipment Manufacturer), leading to significant improvements in the calibrations generation process. It has since been integrated into the OEM's model-based development process.

Introduction

Motivation

Model-based design is a prevalent paradigm in development of embedded control systems across industries. As with any development paradigm, its industrial success depends largely on the practicality of supporting tools [1]. Matlab/Simulink has been a *de facto* standard for design and analysis of embedded automotive systems. While both controller design and plant modeling can be performed within Matlab/Simulink, tools exist that are better suited for plant modeling and analysis than Matlab/Simulink. *MapleSim* [2] by Maplesoft is one of them. MapleSim is a tool for modeling and simulation of physical systems, with the powerful underlying symbolic mathematical engine *Maple*. It offers the flexibility of component-based modeling that, combined with the Maple's symbolic capabilities, enables the development of highly accurate and customizable models, leading to reduced development costs. Furthermore, MapleSim's symbolic capabilities, including symbolic simplification and symbolic optimization of generated code, enable complex models to be simulated at speeds that allow real-time simulation for Hardware-In-the-Loop testing. The tool has been used in different industries, including safety critical industries [3]. Furthermore, the tool has been applied in powertrain modeling and analysis [4, 5, 6]. For example, [4] uses MapleSim/Maple for modeling and rapid prototyping of a powertrain.

In model-based development, the calibration process makes up a significant portion of overall development efforts [7]. The calibration process deals with tuning system parameters—*calibrations*—to meet multiple requirements. Within the powertrain controls, calibrations typically reflect parameters (e.g., filters' parameters, delays, thresholds, etc.) that are used for fine-tuning performance, fuel-efficiency, safety and drivability of vehicles. Our work focuses on generating calibrations for a module of the Hybrid Powertrain Controller (HPC), the main controller within the powertrain control software of a hybrid electric vehicle. The module captures the powertrain's equations of motion expressing a desired set of torques as a function of a different set of torques and speeds. The coefficients in these equations are called *Torque Speed Coefficients calibrations* or TSC cals. The module effectively represents a *powertrain-abstraction layer* (in software engineering terms, a *hardware hiding module* [8]) within the powertrain software that hides the deployed powertrain architecture from the remaining powertrain software, allowing for a high degree of software reuse across different powertrain architectures in multiple vehicle designs.

Our approach for automatic calibrations generation of the TSC cals for the HPC controller is as follows. First, a model representing a low-fidelity, rigid body dynamics of the powertrain is developed in MapleSim from the powertrain schematics. Then, MapleSim's underlying computation engine, Maple, is used to automatically generate symbolic equations representing the behavior of the system. Maple is further used to manipulate the equations to produce a form suitable for easy extraction of the relevant coefficients representing the desired calibrations. We note that only steady-state responses are considered: the models corresponding to each of the plausible modes of operation of a powertrain are considered without analyzing the transient responses as the calibrations of interest do not depend on the transient response of the powertrain.

Prior Work

Model calibration is the process of adjusting parameters of a model so that it complies with a reference system [9]. The reference system can be either the actual physical system or a trusted reference model. MapleSim and Maple have been previously used in the calibration phase of a lithium-ion battery modeling [10] to estimate the parameters of a MapleSim model of the lithium-ion battery. In that work, the homotopy optimization method of [11] was used to match the model's input/output behavior to experimental data. Our work, however, does not involve a parameter estimation approach that uses an optimization method to minimize the difference between the model's input/output behaviour and experimental data. Instead, our reference model is a low-fidelity model of a physical powertrain. The model's underlying mathematical representation is then generated automatically using Maple. Maple is further used to manipulate the representation to an appropriate set of equations so that the coefficients of the equations represent the calibrations of interest.

Contributions

This paper presents a novel approach in the generation of the TSC calibrations for powertrain controllers. The generation is highly automatic: after the model of a powertrain architecture is developed in MapleSim, a generic script template is adapted for the specific architecture to pick out the relevant coefficients from the (automatically generated) mathematical representation of the modeled system. The practicality of the approach has been validated in an industrial setting: the calibrations generation process has been integrated into the existing model-based development process of a large automotive OEM (Original Equipment Manufacturer) proving to be superior to the previously used calibrations generation process. Moreover, although our work focuses on the calibration phase of controller synthesis in the model-based development using a low-fidelity plant model of the powertrain developed from its schematics, this MapleSim model can be reused in plant modeling and analysis after being appropriately augmented with additional details to provide a higher-fidelity plant model of the powertrain model. It can then be used in a number of applications, including high-performance real-time hardware simulations, thanks to Maple's powerful symbolic engine.

In the following sections, we first summarize the capabilities of the MapleSim/Maple environment. Then, the calibrations generation methodology is presented and demonstrated on a simple transmission, and its industrial application on a real-world powertrain of a hybrid electric vehicle is illustrated. Further, the integration of the methodology into the existing model-based development process of our industrial partner is detailed. Finally, we offer some useful insights into benefits and limitations of the methodology, and then conclude the paper with avenues for future work.

Tools

The Matlab/Simulink environment offers excellent capabilities for modeling, simulation and analysis of embedded systems. Once a physical system is decomposed into block diagram structures with causal interactions, it can be efficiently modeled within the Matlab/Simulink framework. Often, a significant effort in terms of analysis and analytical transformations is needed to obtain a model in this form. Furthermore, this effort is error-prone. In order to allow reuse of component models, the equations could be stated in a neutral form without making assumptions on the order of computations. This, so-called *acausal modeling*, allows for describing a system's physical structure and essence as opposed to an algorithmic procedure or structure. Acausal modeling languages allow use of an object-oriented representation that permits an intuitive definition of a system model by graphically describing its topology—components can be connected and their relationship defined without a need to make a decision regarding which signals are inputs and which are outputs.

While Matlab/Simulink supports *causal modeling* essentially describing system behaviour—with a main emphasis on what the system does—acausal modeling tools like MapleSim, Dymola, SimulationX and AMESim emphasize what the system represents. It should be noted that Matlab/Simulink's framework also recently provided a means for acausal modeling, using Simscape. However, the core benefit of using MapleSim for the purpose of this work is its underlying symbolic engine, Maple, which ultimately provides the means to retrieve model's equations in their symbolic form, together with the seamlessly integrated environment (between Maple and MapleSim) to efficiently manipulate and analyze them—which Simscape does not. As a consequence, MapleSim, together with Maple, could be a great addition to an already existing Simulink-based toolchain. It provides connectivity to Simulink, whether it is achieved by exporting models into S-functions or using Functional Mockup Interface (FMI).

Within MapleSim, engineers create the system diagram on-screen and the model equations are generated automatically. The equations can be viewed and manipulated within the Maple environment, where engineers can take advantage of Maple's standard packages for dynamic system analysis, optimization, statistics, and more. The Maple programming language can be used to further manipulate and create applications around developed models. MapleSim builds on Maple's numeric and symbolic computation to perform simulations of complex models. The equations generated from the model are simplified symbolically and higher index DAEs (Differential Algebraic Equations) are reduced using Maple's index reduction algorithms, which also remove redundant equations and flag inconsistencies within the model. The simplified model is numerically simulated, with compilation options for further gain in speed of execution. MapleSim has the ability to convert models to C code using Maple's code generation and optimization tools. Maplesoft also offers numerous toolboxes for export of models into various simulation environments such as Simulink, LabVIEW (a graphical programming environment for measurement, testing and modeling of control systems) and ControlDesk (dSPACE's experiment software for electronic control unit (ECU) development and validation).

Calibrations Generation

In this section we will describe the methodology used to obtain the TSC calibrations. First, the problem is precisely defined and the limitations of the previous solution are explained. Then, the steps of the proposed methodology are described using a simplified transmission.

Problem Statement

The main controller within the powertrain software of electrified vehicles of our industrial partner, a large automotive OEM, optimizes the vehicle's energy consumption (fuel economy) and drivability. The controller is called the Hybrid Powertrain Controller (HPC) and is implemented in Matlab/Simulink. A module of the HPC captures powertrain's dynamics relating the powertrain's speeds and torques. The module represents a powertrain-abstraction layer within the controller—it encodes the powertrain architecture using a set of calibrations, so-called Torque Speed Coefficients calibrations or TSC cals, so that the rest of the powertrain software is not aware of the underlying powertrain architecture and can therefore be reused throughout different powertrain architectures.

In the original calibrations generation process of the OEM, for a given powertrain architecture, the powertrain's dynamics relating powertrain's speeds and torques were represented using symbolic, *manually derived* equations within hand-coded Matlab's scripts. Using Matlab's symbolic solver, the TSC calibrations were then generated as relevant coefficients from these equations. The approach suffered from a number of issues. The manual derivation of speed-torque dependencies for each powertrain configuration using schematics is a tedious, time-consuming, and error-prone process. An in-house tool was used to validate the calibrations derived manually. However, the tool had many drawbacks that eventually rendered it inadequate for use in the calibrations generation process. Most notably, the tool lacked proper debugging facilities. Furthermore, the tool did not provide support for all typical powertrain components (e.g., one-way clutches were not natively supported), and was unintuitive to use.

Given the aforementioned drawbacks of the described manual calibrations generation process, we seek for a highly automated calibrations generation methodology whose application will result in a substantial reduction of modeling and analysis efforts, as well as significantly decrease the potential for errors.

Methodology

This section presents our proposed methodology to tackle the presented problem. We use a four-speed automatic transmission from [12] as a simple running example.

First, we define the inputs to the methodology. The schematic of the powertrain is an input to the methodology. The schematic of the example four-speed automatic transmission is shown in Figure 1. The transmission consists of five clutches, each of which can be either locked or unlocked. This allows for different gear ratios between the input and the output of the transmission which, in turn, allow for different modes of operation. PG_n represents a planetary gear composed of ring, carrier, and sun gears (R_n, C_n, and S_n, respectively). Clutches C_{123}, C_{34}, and CR can directly connect the respective gears to the engine (*input* in the figure). Clutches $C1R$ and C_{24} are used to ground the gears by locking them to the transmission casing.

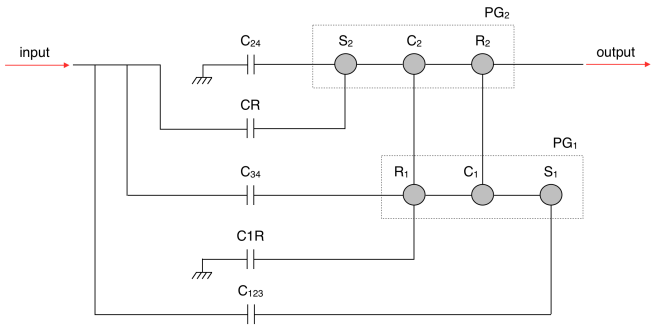

Figure 1. Schematic of the four speed automatic transmission

Next, the desired calibrations are defined for each mode. The four-speed automatic transmission has five modes of operation:

- $C1R$ and C_{123} locked; other clutches unlocked,
- C_{24} and C_{123} locked; other clutches unlocked,
- C_{34} and C_{123} locked; other clutches unlocked,
- C_{34} and C_{24} locked; other clutches unlocked,

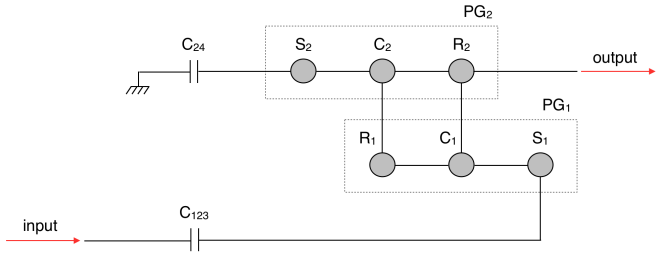

Figure 2. Schematic of Gear 2

- $C1R$ and CR locked; other clutches unlocked.

For conciseness of exposition, we illustrate how calibrations are defined on only one mode of the four speed transmission, the mode Gear 2. As shown in Figure 2, Gear 2 is achieved by locking clutches C_{24} and C_{123} while the rest of the clutches remain unlocked. Let us assume that the desired calibrations for Gear 2 are defined as coefficients in the following equations where T_i represents the input torque, T_x represents the torque at clutch x, and T_o represents the output torque.

$$T_o = G2\_ToFromTi \cdot T_i + G2\_ToFromTC1R \cdot T_{C1R} + G2\_ToFromTC34 \cdot T_{C34} + G2\_ToFromTCR \cdot T_{CR} \quad (1)$$

$$T_{C123} = G2\_TC123FromTi \cdot T_i + G2\_TC123FromTC1R \cdot T_{C1R} + G2\_TC123FromTC34 \cdot T_{C34} + G2\_TC123FromTCR \cdot T_{CR} \quad (2)$$

$$T_{C24} = G2\_TC24FromTi \cdot T_i + G2\_TC24FromTC1R \cdot T_{C1R} + G2\_TC24FromTC34 \cdot T_{C34} + G2\_TC24FromTCR \cdot T_{CR} \quad (3)$$

Now that the inputs to the methodology (the powertrain schematic, modes and desired calibrations) have been defined, we outline the steps of the proposed methodology.

1. *Modeling Powertrain in MapleSim.* In the first step of the methodology, the powertrain architecture is modeled from its schematic using MapleSim. Figure 3 contains the (non-linear) MapleSim model of the schematic powertrain from Figure 1. In the figure, T_1 and T_2 are the input and output torques, respectively, and I_1 represents the input inertia.
2. *Generating Powertrain Modes.* The MapleSim models corresponding to different modes of the powertrain are generated manually from the model in Figure 3 by locking/unlocking clutches. A locked clutch corresponds to a rigid connection and an added torque sensor, while unlocking a clutch includes removing the clutch and the connections on either side of the clutch. The MapleSim model of the Gear 2 mode of the four-speed transmission is shown in Figure 4. In the figure, T_I and T_O represent the input and output torques, respectively; TS_1 and TS_2

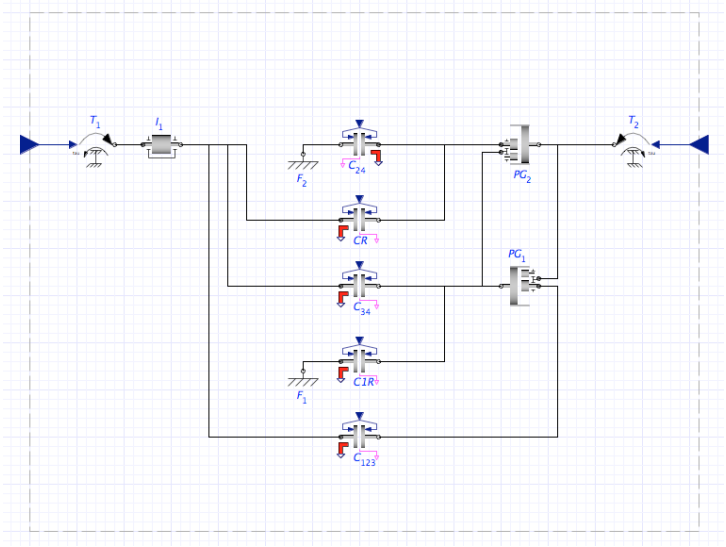

Figure 3. MapleSim model of four speed automatic transmission

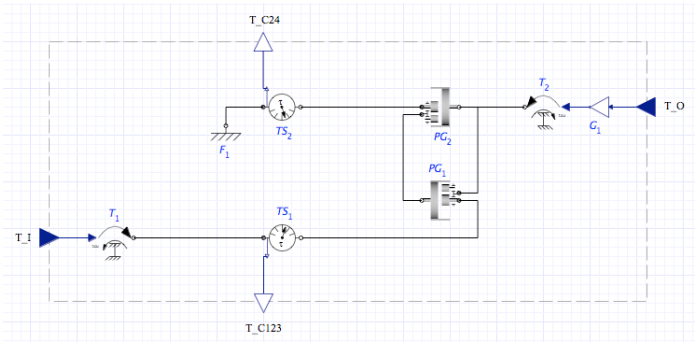

Figure 4. Linearized model for Gear 2

are torque sensors; F_1 represents ground, while G_1 represents a gain.

3. *Generating Equations and Extracting Calibrations.* Equations defining the dynamics of the powertrain in a mode are automatically generated from the corresponding MapleSim model, in a fully symbolic and parametrized form, using Maple. Each of the MapleSim models representing a mode of a given powertrain has a Maple script attached to it. The script automates the manipulation of the corresponding mode's equations to a form needed to retrieve the desired calibrations. For Gear 2, the equations governing the behaviour of the model of Figure 4 are manipulated into the form given by Equations 1, 2, and 3. We next explain the Gear 2 script in more detail, focusing on its most important parts.

    a. First, Maple's `GetEquations` command in Figure 5 generates a mathematical representation of a physical system represented using MapleSim—the command retrieves the set of equations underlying the MapleSim system from Figure 4.

```
> #The LinkModel() command creates an object that allows access to the
  MapleSim model.
> restart;
  A := MapleSim:-LinkModel('filename' = "4 Speed Gear 2.msim"):
  A:-SetSubsystemName("Speed"):
  ret          := A :- GetEquations('output' = 'all', 'filter' = {},
  'simplify' = true):
```

Figure 5. Linking to the MapleSim model and obtaining equations

    b. The speed and torque variables of interest are defined (the variables are first renamed for clarity and aesthetic reasons)—see Figure 6. In this example, only torques are required.

```
> #The map (fcn, expr) command applies the function fcn to the elements of
  the expression expr. In the instance below, varsInterest consists of all
  of the elements on the right hand side of the varSubstitution list.
> varSubstitutions := [T_I(t)    = Ti(t),
                       T_O(t)    = To(t),
                       T_C1R(t)  = TC1R(t),
                       T_C123(t) = TC123(t),
                       T_C24(t)  = TC24(t),
                       T_C34(t)  = TC34(t),
                       T_CR(t)   = TCR(t)]:

  varsInterest := map(rhs,varSubstitutions):
```

Figure 6. Variables of interest

    c. The set of equations generated in a) is reduced to only include the equations containing the variables of interest defined in Figure 6. The corresponding part of the script and the resulting equations are shown in Figure 7.

```
> #The select(condition, expression) command selects the operands of the
  expression which satisfy the condition.
> #The expand(expression) command distributes products over sums in the
  expression.
> #The subs(a=x, expression) command substitutes a for x in the expression.
> equations := [op(select(has,expand(subs(varSubstitutions,DAEs union
  Definitions)),varsInterest))]:
> equationsVector := convert(equations,Vector);
```

$$
equationsVector := \begin{bmatrix}
-\frac{S1\,PG2\_sun\_tau(t)\,R2}{R1\,S2} - \frac{S1\,PG2\_sun\_tau(t)}{R1} + Ti(t) = 0 \\
PG1\_carrier\_tau(t) + \frac{R2\,PG2\_sun\_tau(t)}{S2} - To(t) = 0 \\
PG1\_carrier\_tau(t) + \frac{R2\,PG2\_sun\_tau(t)}{S2} + PG2\_sun\_tau(t) + Ti(t) = 0 \\
G1\_u(t) = To(t) \\
G1\_y(t) = To(t) \\
T1\_tau(t) = Ti(t) \\
T2\_tau(t) = To(t) \\
TC123(t) = Ti(t) \\
TC24(t) = PG2\_sun\_tau(t) \\
TS1\_RO(t) = Ti(t) \\
PG1\_sun\_tau(t) = Ti(t) \\
T1\_flange\_tau(t) = -Ti(t) \\
T2\_flange\_tau(t) = -To(t) \\
TS1\_CB\_u(t) = Ti(t) \\
TS1\_CB\_y(t) = Ti(t) \\
TS1\_TS\_tau(t) = Ti(t) \\
PG1\_PG3P\_sun\_tau(t) = Ti(t) \\
TS1\_TS\_flange\_a\_tau(t) = Ti(t) \\
TS1\_TS\_flange\_b\_tau(t) = -Ti(t) \\
TS1\_flange\_a\_tau(t) = Ti(t) \\
TS1\_flange\_b\_tau(t) = -Ti(t)
\end{bmatrix}
$$

Figure 7. The model equations

    d. The solving variables are then chosen ($TC123$, $TC24$, and To):

```
solvingVars := [TC123(t),TC24(t),To(t)];
```

    e. Using Maple's `solve` command, the equations are solved symbolically (Figure 8) for the defined solving variables and intermediary variables (not including the variables of interest that are not solving variables—i.e., TCR, $TCR1$, $TC34$ and Ti). Next, the relevant dependencies for the solving variables are extracted from the resulting equations (Figure 9). The resulting dependencies of interest for the

solving variables T_{C123}, T_{C24}, and T_o, shown as matrices, can be seen in Figure 10.

```
> #The solve(equations, variables) command solves a set of equations for a set of
  variables.
> #The indets(expr,typename) command returns a set containing all of the indeterminates
  of the expr that are of type typename.
> #The minus and union commands are the set operations complement and union
  respectively.
> #The op(expr) command extracts operands from the expression expr.
> #The dependentAccelerations vector is not needed for this example, however, since the
  script is generic, it is included in the code.
> dependentAccelerations := {ni_dot(t),no_dot(t)};
> solutions := solve(equations,[op(indets(equations,function) minus {op(varsInterest)}
  union {op(solvingVars)} minus dependentAccelerations)]): solutionsVector := convert
  (solutions, Vector);
```

$$solutionsVector := \begin{bmatrix} G1\_u(t) = -\frac{Ti(t)\,(R1\,S2 + R2\,S1 + S1\,S2)}{S1\,(S2 + R2)} \\ G1\_y(t) = -\frac{Ti(t)\,(R1\,S2 + R2\,S1 + S1\,S2)}{S1\,(S2 + R2)} \\ T1\_tau(t) = Ti(t) \\ T2\_tau(t) = -\frac{Ti(t)\,(R1\,S2 + R2\,S1 + S1\,S2)}{S1\,(S2 + R2)} \\ TC123(t) = Ti(t) \\ TC24(t) = \frac{Ti(t)\,R1\,S2}{S1\,(S2 + R2)} \\ TS1\_RO(t) = Ti(t) \\ To(t) = -\frac{Ti(t)\,(R1\,S2 + R2\,S1 + S1\,S2)}{S1\,(S2 + R2)} \\ PG1\_carrier\_tau(t) = -\frac{(R1 + S1)\,Ti(t)}{S1} \\ PG1\_sun\_tau(t) = Ti(t) \\ PG2\_sun\_tau(t) = \frac{Ti(t)\,R1\,S2}{S1\,(S2 + R2)} \\ T1\_flange\_tau(t) = -Ti(t) \\ T2\_flange\_tau(t) = \frac{Ti(t)\,(R1\,S2 + R2\,S1 + S1\,S2)}{S1\,(S2 + R2)} \\ TS1\_CB\_u(t) = Ti(t) \\ TS1\_CB\_y(t) = Ti(t) \\ TS1\_TS\_tau(t) = Ti(t) \\ PG1\_PG3P\_sun\_tau(t) = Ti(t) \\ TS1\_TS\_flange\_a\_tau(t) = Ti(t) \\ TS1\_TS\_flange\_b\_tau(t) = -Ti(t) \\ TS1\_flange\_a\_tau(t) = Ti(t) \\ TS1\_flange\_b\_tau(t) = -Ti(t) \end{bmatrix}$$

Figure 8. Solving the equations

```
> #Select equations of interest from the solutions Vector
> res_TC123 := expand(select(x->lhs(x) = TC123(t), op(solutions)));
```

$$res\_TC123 := [TC123(t) = Ti(t)]$$

```
> res_TC24 := expand(select(x->lhs(x) = TC24(t), op(solutions)));
```

$$res\_TC24 := \left[TC24(t) = \frac{Ti(t)\,R1\,S2}{S1\,(S2 + R2)}\right]$$

```
> res_To := expand(select(x->lhs(x) = To(t), op(solutions)));
```

$$res\_To := \left[To(t) = -\frac{Ti(t)\,R1\,S2}{S1\,(S2 + R2)} - \frac{Ti(t)\,R2}{S2 + R2} - \frac{Ti(t)\,S2}{S2 + R2}\right]$$

Figure 9. Obtaining the dependencies of interest

f. Finally, the parameter values for the gear ratios, inertias, and damping are substituted in to obtain the numeric calibration coefficients. For instance, the calibration value for T_o from T_i is -1.45 (i.e. $T_o = -1.45 \cdot T_i$), as shown in Figure 11.

The script is generic and self-documenting. For other modes of the same architecture, the only trivial change is the choice of solving variables.

4. *Printing.* The same Maple script generates the calibrations in a ".m" (Matlab file) format as seen in Figure 11. These numerical calibrations can then used by a Simulink controller model for code generation, compilation, and eventual real-time execution of the controller's functionality.

```
> #Rewrite the equations of interest (that show the dependencies) as matrices.
> sVars_TC123 := [Ti(t),TC1R(t),TC24(t),TCR(t)]:
> A_TC123,b_TC123 := LinearAlgebra:-GenerateMatrix([rhs(op(res_TC123))-lhs(op(res_TC123)
  )=0],sVars_TC123):
> b_TC123 = convert(customSimplify(A_TC123),Vector),convert(sVars_TC123,Vector);
```

$$\left[TC123(t)\right] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} Ti(t) \\ TC1R(t) \\ TC24(t) \\ TCR(t) \end{bmatrix}$$

```
>
> sVars_TC24 := [Ti(t),TC1R(t),TC34(t),TCR(t)]:
> A_TC24,b_TC24 := eval(LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_TC24))-lhs(op
  (res_TC24))=0]),sVars_TC24)):
> b_TC24 = convert(customSimplify(A_TC24),Vector),convert(sVars_TC24,Vector);
```

$$\left[TC24(t)\right] = \begin{bmatrix} \frac{R1\,S2}{S1\,(S2 + R2)} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} Ti(t) \\ TC1R(t) \\ TC34(t) \\ TCR(t) \end{bmatrix}$$

```
>
> sVars_To := [Ti(t),TC1R(t),TC34(t),TCR(t)]:
> A_To,b_To := eval(LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_To))-lhs(op(res_To)
  )=0]),sVars_To)):
> b_To = convert(customSimplify(A_To),Vector),convert(sVars_To,Vector);
```

$$\left[To(t)\right] = \begin{bmatrix} \frac{(-R2 - S2)\,S1 - R1\,S2}{S1\,(S2 + R2)} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} Ti(t) \\ TC1R(t) \\ TC34(t) \\ TCR(t) \end{bmatrix}$$

Figure 10. Relevant dependencies for Gear 2

```
% ========================================================================
% Gear 2 Torque Equations
% ========================================================================

% Dependent Variables:  Ti, TC1R, TC34, and TCR

G2_ToFromTi                          = single(      -1.4500000000 );
G2_ToFromTC1R                        = single(       0.0000000000 );
G2_ToFromTC34                        = single(       0.0000000000 );
G2_ToFromTCR                         = single(       0.0000000000 );

% Dependent Variables:  Ti, TC1R, TC34, and TCR

G2_TC123FromTi                       = single(       1.0000000000 );
G2_TC123FromTC1R                     = single(       0.0000000000 );
G2_TC123FromTC34                     = single(       0.0000000000 );
G2_TC123FromTCR                      = single(       0.0000000000 );

% Dependent Variables:  Ti, TC1R, TC34, and TCR

G2_TC24FromTi                        = single(       0.4500000000 );
G2_TC24FromTC1R                      = single(       0.0000000000 );
G2_TC24FromTC34                      = single(       0.0000000000 );
G2_TC24FromTCR                       = single(       0.0000000000 );
```

Figure 11. Calibrations for Gear 2

Industrial Application

Application on Real-World Powertrain

The proposed methodology is scalable to most conventional or hybrid powertrain architectures that are currently used in the automotive industry. It has already been successfully implemented on three industrial hybrid powertrain configurations. To illustrate the methodology on a real-world application, we apply the methodology to an industrial powertrain configuration, a two-mode *Electrically Variable Transmission (EVT)*. However, for confidentiality reasons, we disclose neither the powertrain schematics, nor the details of all the steps of the methodology applied on it. We merely intend to show that the methodology scales to production-scale applications. Moreover, its benefits are far more obvious on the real-world example given that the relevant speed-torque dependencies are far more complex than for the simple example of the four-speed automatic transmission.

We focus on Step 3 of the methodology (generating relevant equations). The script described in the previous section (for the four-speed automatic transmission), only slightly modified, is used to generate

equations for the EVT. The major difference between the two scripts is the choice of solving variables, as highlighted in Figure 12. This configuration is much more involved than the simple example of the four-speed automatic transmission. It includes 5 relevant speed variables, 7 torques, 2 accelerations, 8 inertias, and 5 dampeners. The `diff` command is used to calculate the partial derivative of the speed variables `n3(t)` and `n5(t)`) with respect to time, resulting in accelerations `n3_dot(t)` and `n5_dot(t)`), respectively.

```
> torqueSubs := [EVT_T1(t)  = T1(t),
                 EVT_T2(t)  = T2(t),
                 EVT_T3(t)  = T3(t),
                 EVT_T4(t)  = T4(t),
                 EVT_T5(t)  = T5(t),
                 EVT_T6(t)  = T6(t),
                 EVT_T7(t)  = T7(t)]:

  accelSubs  := [EVT_n3_dot(t) = n3_dot(t),
                 EVT_n5_dot(t) = n5_dot(t),
                 diff(n3(t),t) = n3_dot(t),
                 diff(n5(t),t) = n5_dot(t)]:

  solvingVars := [T7(t), T6(t), T4(t)];
  dependentAccelerations := {n3_dot(t), n5_dot(t)};
```

Figure 12. Variables of interest and solving variables

Some of the equations that define the behaviour of one of the modes of the EVT are shown in Figure 13. In the equations, Ti, ni, and ni_dot refer to torque, speed, and acceleration variables at node i, respectively. I_i, B_i, and β refer to inertia, dampening, and gear ratios, respectively.

```
res_T4_T7 := expand(select(x->lhs(x) = T4(t),  op(solutions))):
sVars_T4_T7 := [T1(t),T2(t),n6(t),n3(t),n5(t),n3_dot(t),n5_dot(t),T6(t),T3(t),T7(t)]:
A_T4_T7,b_T4_T7 := eval(LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_T4_T7))-lhs(op(res_T4_T7))=0]),sVars_T4_T7)):
b_T4_T7 = convert(customSimplify(A_T4_T7),Vector),convert(sVars_T4_T7,Vector);
```

$$[T4(t)] = \begin{bmatrix} \frac{I_2+I_3}{(1+\beta)I_4} \\ 0 \\ 0 \\ \frac{(-I_2-I_3)B_1+B_3I_4}{I_4} \\ \frac{(I_2+I_3)B_1\beta}{(1+\beta)I_4} \\ 0 \\ \frac{(I_2+I_3)\beta}{1+\beta} \\ 1 \\ 0 \\ \frac{-(1+\beta)^2 I_4 - I_2 - I_3}{I_4(1+\beta)^2} \end{bmatrix} \cdot \begin{bmatrix} T1(t) \\ T2(t) \\ n6(t) \\ n3(t) \\ n5(t) \\ n3\_dot(t) \\ n5\_dot(t) \\ T6(t) \\ T3(t) \\ T7(t) \end{bmatrix}$$

```
sVars_T7 := [T1(t),T2(t),n5_dot(t),n3_dot(t),T4(t),n5(t),n3(t)]:
A_T7,b_T7 := eval(LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_T7))-lhs(op(res_T7))=0]),sVars_T7)):
b_T7 = convert(customSimplify(A_T7),Vector),convert(sVars_T7,Vector);
```

$$[T7(t)] = \begin{bmatrix} 1+\beta \\ 0 \\ \beta(1+\beta)I_4 \\ -(1+\beta)^2 I_4 \\ 0 \\ B_1\beta(1+\beta) \\ -(1+\beta)^2 B_1 \end{bmatrix} \cdot \begin{bmatrix} T1(t) \\ T2(t) \\ n5\_dot(t) \\ n3\_dot(t) \\ T4(t) \\ n5(t) \\ n3(t) \end{bmatrix}$$

Figure 13. Equations defining a specific mode

Integration into Existing Process

The calibrations generation methodology has been integrated into the model-based development process of our industrial partner. The resulting process is depicted using the process flowchart in Figure 14. It consists of the following phases:

1. Model the powertrain in MapleSim from its schematic.

2. For each mode of the configuration, the required torque and speed equations governing its behaviour are generated manually, using the laws of motion. In parallel with the manual derivation, the torque and speed equations are automatically generated from the MapleSim model of each of the modes using the methodology presented in this paper.

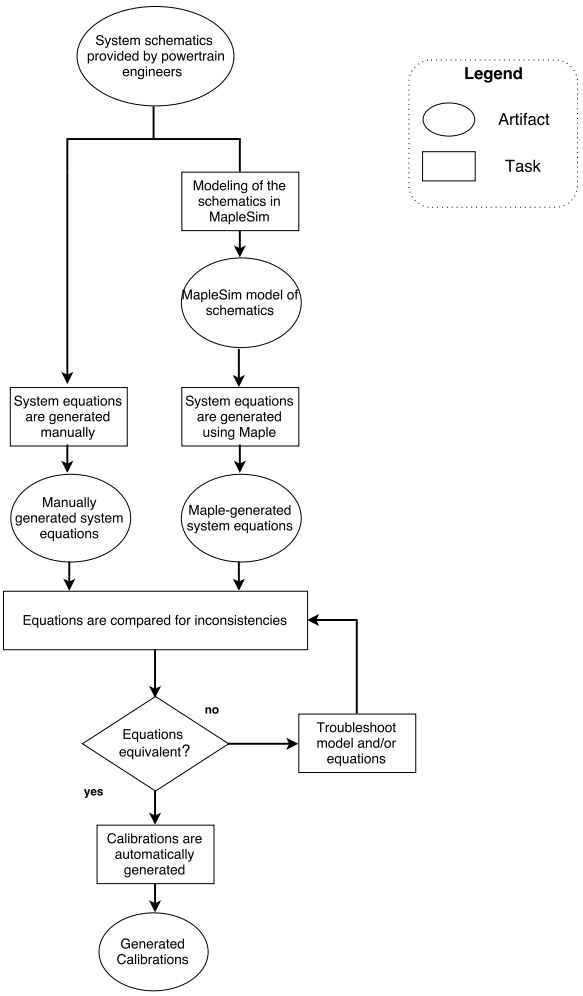

Figure 14. Calibrations generation process flow chart

3. The manually derived equations are compared to the automatically generated equations. A mismatch indicates an error in either manual derivation or automatic generation. The engineer then troubleshoots, and corrects any issues. Once the equations are identical, calibrations are automatically generated from the retrieved equations.

In summary, our methodology is currently employed in the existing calibrations generation process as part of *calibrations validation*: the manually derived equations representing the behaviour of a powertrain are validated against equations generated using MapleSim/Maple in our methodology. If the equations match, the calibrations are automatically printed into an ".m" file.

For the two-mode EVT from the previous section representing a real-world industrial example, the calibrations generation process as depicted in Figure 14 takes one day. In comparison, the application of the original calibrations generation process that used an in-house tool for validation took five days. Moreover, for more complex models, which may include a higher number of clutches, planetary gears, inertias, etc., the presented calibrations generation methodology shows even greater reduction in modeling and validation efforts, while being far less error-prone. It is worth noting that, for the two-mode EVT, the methodology requires insignificant computing resources. More pre-

cisely, the `solve` command, by far the most resource-intensive constituent of the methodology, takes only 63 ms of CPU time and 7.56 MB of memory on an Intel Core i5-3570K quad-core 3.4 GHz machine with 16 GB of RAM to solve the system of 258 equations underlying the EVT. Since the `solve` command can easily handle large systems of thousands of equations, the methodology is easily scalable to considerably more complex powertrains.

Finally, we note that the current application of the methodology as shown in Figure 14 represents merely part of a transition leading to a new calibrations generation process relying solely on the methodology: after the engineers get used to using the tool and increase their confidence in the modeling and calibrations generation using MapleSim/Maple as proposed in this paper, the process from Figure 14 will collapse to our methodology.

Discussion

The introduction of a symbolic physical modeling tool into the controller synthesis process of a large OEM provides significant reduction in the calibrations generation efforts, while providing greater confidence in the correctness of the generated calibrations. Once the calibrations have been generated for a given powertrain architecture, in the case when the architecture undergoes modifications, engineers can easily change the MapleSim model accordingly, and re-run the Maple script (potentially, slightly modified) instead of manually re-analyzing a model and re-calculating the calibrations. In a similar manner, the Maple script requires only trivial modifications to be applied to it in order to be used for the calibrations generation for a different architecture.

The models used for calibrations generation in this paper are high-level models of powertrain transmissions: they depict only mechanical, rigid-body dynamics of a transmission incorporating basic models of clutches, gear sets, and shafts, including their corresponding inertia and damping parameters. For this particular application, higher fidelity is not required. However, MapleSim can also be used in plant modeling and analysis—in fact, this is exactly where it excels. More precisely, the high-level model of a powertrain, as used in this paper for controller synthesis, can be used as an initial model to be refined into higher-fidelity MapleSim models of the plant that would then be used in real-time simulations, sensitivity analysis, parameter estimation, HIL (Hardware-in-the-Loop) testing, etc. This would allow for a tighter integration of tools in a company's toolchain.

Further, given the complementary roles of MapleSim and Simulink, as well as the ubiquity of Simulink, the integration of MapleSim with Simulink-based toolchain is very important. MapleSim and Simulink can be integrated using Maplesoft's *MapleSim Matlab Connector* [13]. The tool is capable of exporting MapleSim models to Simulink: it generates S-function blocks from MapleSim models so that they can then be included within Simulink models. Furthermore, Maplesoft's *MapleSim Connector for FMI* [13] provides a means for simulation/co-simulation with numerous free and commercial symbolic/numeric physical modeling tools for seamless integration with the existing toolchain using the FMI (Functional Mockup Interface) standard [13].

The correctness of the calibrations generated using the methodology depends on the correctness of symbolic transformations performed by Maple's engine. Having a symbolic model (and Maple's symbolic engine) at hand provides the means for automatic verification and validation of transformations of symbolic equations to calibrations using automated theorem provers (e.g., *PVS* [14], *Coq* [15]). Preliminary work on this topic appears in [16]. Using this method could help to eliminate tool related single-points of failure in the calibrations generation process for safety critical applications.

Conclusions and Future Work

A novel methodology for automatic generation of calibrations for powertrain controllers was presented. The methodology has been shown to significantly reduce the time required to generate calibrations for a module of the main powertrain controller within hybrid electric vehicles. Additionally, it reduces human error, which is a common source of error present in the manual calibrations generation process. As the approach is based on the Maple/MapleSim environment, integration with existing model-based development processes and tool chains is made easier with tools such as Maplesoft's MapleSim Matlab Connector and MapleSim Connector for FMI. In the event that the methodology is not completely integrated, it still provides a powerful tool for independent verification of manually generated calibrations.

One of the drawbacks of the methodology is the learning curve of the Maple/MapleSim environment. A non-shallow understanding of it is required to adapt the scripts and models to new powertrain architectures. To remedy this issue, the following steps can be taken in future work:

1. Generating (linear) MapleSim models representing modes of operation of a powertrain from a generic, non-linear MapleSim model of the powertrain is currently done manually. This process could be automated by modifying the underlying code of the non-linear MapleSim model to generate all the feasible modes of operation automatically.

2. The scripts can be improved to further automate the process. A simple user interface could be created that would allow a user to provide only the MapleSim model of the powertrain and then select the calibrations that need to be extracted.

As already mentioned, verification of the generated calibrations is a topic that needs to be investigated. The methodology can be modified to automatically generate theorems that prove the correctness of the calibrations and the processes that generate them. These theorems can then be proven via automated theorem provers such as PVS and Coq.

References

[1] Hunt, A. and D. Thomas (2002). Ubiquitous automation. *IEEE Software 19*(1), 11.

[2] McPhee, J. (2012, 19 July). MapleSim in engineering research. In *3rd Annual MapleSim Academic Summer Workshop*, McMaster University, Hamilton.

[3] Lawford, M. and A. Wassyng (2012). Formal verification of nuclear systems: Past, present, and future. *Information & Security 28*(2), 223.

[4] Asl, H. A. (2014). *Acausal Powertrain Modelling with Application to Model-based Powertrain Control*. Ph. D. thesis, University of Waterloo.

[5] Saeedi, M. (2010). A mean value internal combustion engine model in MapleSim. Master's thesis, University of Waterloo.

[6] Asl, H. A., N. L. Azad, and J. McPhee (2012). Modeling torque converter characteristics in automatic drivelines: Lock-up clutch

and engine braking simulation. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 359–367. American Society of Mechanical Engineers.

[7] Guerrier, M. and P. Cawsey (2004). The development of model based methodologies for gasoline IC engine calibration. Technical report, SAE technical paper.

[8] Parnas, D. L., P. C. Clements, and D. M. Weiss (1985). The modular structure of complex systems. *IEEE Transactions on Software Engineering SE-11*(3), 259–266.

[9] Hofmann, M. (2005). On the complexity of parameter calibration in simulation models. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology 2*(4), 217–226.

[10] Masoudi, R., T. Uchida, and J. McPhee (2015). Parameter estimation of an electrochemistry-based lithium-ion battery model. *Journal of Power Sources 291*, 215–224.

[11] Dunlavy, D. M. and D. P. O'Leary (2005). Homotopy optimization methods for global optimization. Technical Report SAND2005-7495, Sandia National Laboratories, United States Department of Energy, Albuquerque, New Mexico.

[12] Mallela, S. (2013). Design, modeling and control of a novel architecture for automatic transmission systems. Master's thesis, University of Minnesota.

[13] Fritzson, P. and O. Rogovchenko (2012). Introduction to object-oriented modeling, simulation and control with Modelica. In *Proceedings of 6th MODPROD Workshop on Model-Based Product Development. Linköping University, Sweden*.

[14] Owre, S., J. M. Rushby, and N. Shankar (1992). PVS: A prototype verification system. In *International Conference on Automated Deduction*, pp. 748–752. Springer.

[15] Bertot, Y. and P. Castéran (2013). *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.

[16] Korobkine, A. O. (2002). Model-Based Visual Tracking via Maple Code Generation. Master's thesis, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada.

Contact Information

Alexandre Korobkine is a Senior Project Engineer working for General Motors, where he specializes in designing and developing algorithms servicing operation and safety of powertrain controller modules. E-mail: alexandre.korobkine@gm.com

Romi Boimer received a Bachelors of Engineering and Management from McMaster University in 2016. She is currently working towards a Masters degree in Computer Science (AI/HCI concentration) from Stanford University, California. She is also a Software Development Engineer at Amazon Web Services, Palo Alto, California. Her research interests include robotics, AI, and metaheuristic optimization methods. E-mail: rbboimer@amazon.com

Vera Pantelic is working as a Senior Principal Research Engineer with the McMaster Centre for Software Certification, and McMaster Institute for Automotive Research and Technology (MacAUTO), McMaster University. Her research interests include development and certification of safety-critical software systems, model-based design, and supervisory control of discrete event systems. E-mail: pantelv@mcmaster.ca

Syed Asim Shah is a Research Engineer and a Master's student working at the McMaster Institute for Automotive Research and Technology (MacAUTO), McMaster University. His research interests include safety-critical systems, autonomous vehicle safety, fault tolerant architectures and vehicle software optimization. E-mail: shahsa4@mcmaster.ca

Mark Lawford is the Associate Director of the McMaster Centre for Software Certification, and one of its founders. He is a Professor in the Department of Computing and Software, and has more than 15 years experience integrating formal techniques with practical industrial software engineering applied to safety-critical real-time control systems. E-mail: lawford@mcmaster.ca

Carlos L. Castillo is an Electrified Powertrain Control Engineer at FCA USA LLC. His research interests are Robust Control, System Identification, Self-driving Vehicles, Machine Learning and Convolutional Neural Networks. E-mail: carlos.l.castillo@fcagroup.com

Dr. Feisel Faz Weslati heads the FCA Electrified Powertrain Controls Engineering group. His team is responsible for developing the supervisory control strategies and algorithms for FCA global portfolio of hybrid/electric vehicle. Dr. Weslati has a varied industrial experience in the areas of hybrid systems, NVH, active chassis controls and robotics. He holds a PhD and 2 Masters in Mechanical Engineering and Engineering Management.