

Simplified modelling of electric and hybrid vehicles

A learn-by-using tutorial – November 2024

Massimo Ceraolo – University of Pisa – Italy

Massimo.ceraolo@unipi.it

Table of Contents

1	About this document.....	2
1.1	General	2
1.2	Note of caution	2
2	Introductory stuff	3
2.1	Acronyms	3
2.2	Language and Tools	3
2.3	Model and data (txt) files	3
2.4	Prerequisites to this document	5
2.5	About this document and library	5
3	My first EV model	5
3.1	My first simulation: the Sort1 Cycle	7
3.2	Details on DragForce model	10
3.3	Simulating NEDC	14
3.4	Details of Driver model	15
4	Easing vehicle switch (use of data record)	17
5	Map-based EV models	18
5.1	The idea of map-based electric drives.....	19
5.2	Flexible use of maps from file.....	20
5.3	EV model at work.....	21
5.4	Modelling batteries	28
5.5	Map-based DC-interfaced electric drive implementation	33
6	Simplified electric drives.....	34
6.1	Constant voltage/frequency asynchronous drive	34
6.2	Synchronous machine drive	40
7	Map-based HEV models.....	46
7.1	HEV's resume.....	46
7.2	SHEV basic model	48
7.3	SHEV with SOC closed-loop control.....	51
7.4	SHEV with On/Off control.....	53
7.5	Simplified SHEV with ON/OFF control	55
7.6	PSD-HEV model.....	58
7.7	Simulation "PSecu1" (power filter).....	60
7.8	Simulation "PSecu2" (power-filter and SOC-loop control).....	66
7.9	Simulation "PSecu3" (with power-filter SOC control and ON/OFF)	70
8	Some submodels of EV and HEV models	73
8.1	OneFlange models	73
8.2	ICE models	76
8.3	Genset.....	76
8.4	GensetOO	80
9	Map-based support models	81
9.1	CombiTable1Factor and CombiTable2Factor models	81
9.2	EfficiencyCT block	84

9.3	EfficiencyLF block.....	87
9.4	LimTorqueFV block	88
9.5	LimTorque (includes LimTorqueFV behaviour)	90
9.6	ConstPg model.....	91
10	EV-HEV simulation concluding remarks.....	93
11	Testing individual models	93
11.1	LimTorque.....	93
11.2	OneFlange.....	94
11.3	TwoFlange.....	103
11.4	ICE	104
11.5	Genset.....	110
12	Appendix 1: Content of EVmaps.txt	116
13	Appendix 2: Efficiency.m	117
14	Appendix 3: Some info on EHPTlib architecture.....	117
14.1	Inheritance of ICE models.....	118
14.2	Inheritance of OneFlange models.....	122
14.3	Inheritance of Gensets.....	123
14.4	Inheritance of ECUs	123
15	References	123

1 About this document

1.1 General

This document is an update of a webbook chapter available from the OpenModelica web site from the url:

<http://omwebbook.openmodelica.org/SMEHV>

Since updating the quoted document involves work of people other than myself, I decided to keep updated this copy to be distributed along the library via GitHub.

This document is a tutorial for the usage of library <https://github.com/ceraolo/EHPTlib>. It makes extensive usage of the examples of use of this library provided in <https://github.com/ceraolo/EHPTexamples>.

The current version is for release 3.0.0 of the library, which was a complete restructuring over previous 2.x.x. Inheritance is now used extensively, and the definition of models and corresponding masks for parameters is much more consistent now.

Several simulation models testing individual EHPTlib models have been added. This helps users use the provided models, since they can see the expected behaviour at least in some cases.

Furthermore, an additional SHEV models OO_noMap has been added, which can show very easily the effects of different genset sizes on the hybrid behaviour.

Correspondingly, this document has grown from 78 pages to 124.

1.2 Note of caution

EHPTlib is periodically updated and improved.

Even though I try to keep this document updated, it would be too demanding updating all the involved plots whenever small changes are produced e.g. by changing some model data either set online or read from a txt file.

I update figures and plots only when very significant differences occur.

You, the reader, are therefore invited not to expect always to get exactly the same results as shown here, when running the examples. Small differences can be present because of differences in data, but the utility of all the plots is retained.

2 Introductory stuff

2.1 Acronyms

EHPT Electric and Hybrid Power Train

MSL Modelica Standard Library (in its rel. 4.0.0, except when otherwise specified)

MS Modelica specifications (in its rel. 3.5 except when otherwise specified)

2.2 Language and Tools

This chapter is written taking as reference MS 3.5 and MSL 4.0.0.

All the examples provided are checked with two Modelica tools:

- Dymola, a very well-known commercial Modelica tool, in its rel. 2023x
- OpenModelica the most complete open-source Modelica simulation tool, in its rel. 1.23.1, 64bit Windows version.

OpenModelica is available for free, under different platforms (Windows, Mac, Unix) from its official site, <http://www.openmodelica.org>.

Nearly all the Modelica tool pictures included in this chapter are taken from OpenModelica. The appearance of that tool might slightly change from a computer to another, even running Windows, because the OpenModelica editor, OMEdit, is highly customisable, and my customization could have a slightly different appearance than readers'.

Once one has run a simulation using OpenModelica, the output can be analysed in other computers, even not having OpenModelica installed, especially if the CSV format is used as an output. OpenModelica CSV outputs can be conveniently read, and the result plotted and post-processed using the freeware PlotXY program, which can be downloaded from:

<http://ceraolo-plotxy.ing.unipi.it/default.htm>

This program is continuously updated, and the most recent version is dated 2023. Plots with PlotXY are high quality and can be exported in several ways. Post-processing with PlotXY includes:

- algebraic combination of curves (sums, products, etc. with possible usage of parentheses)
- Fourier analysis to find and plot harmonics
- integral of variables (e.g., to find energies from powers).

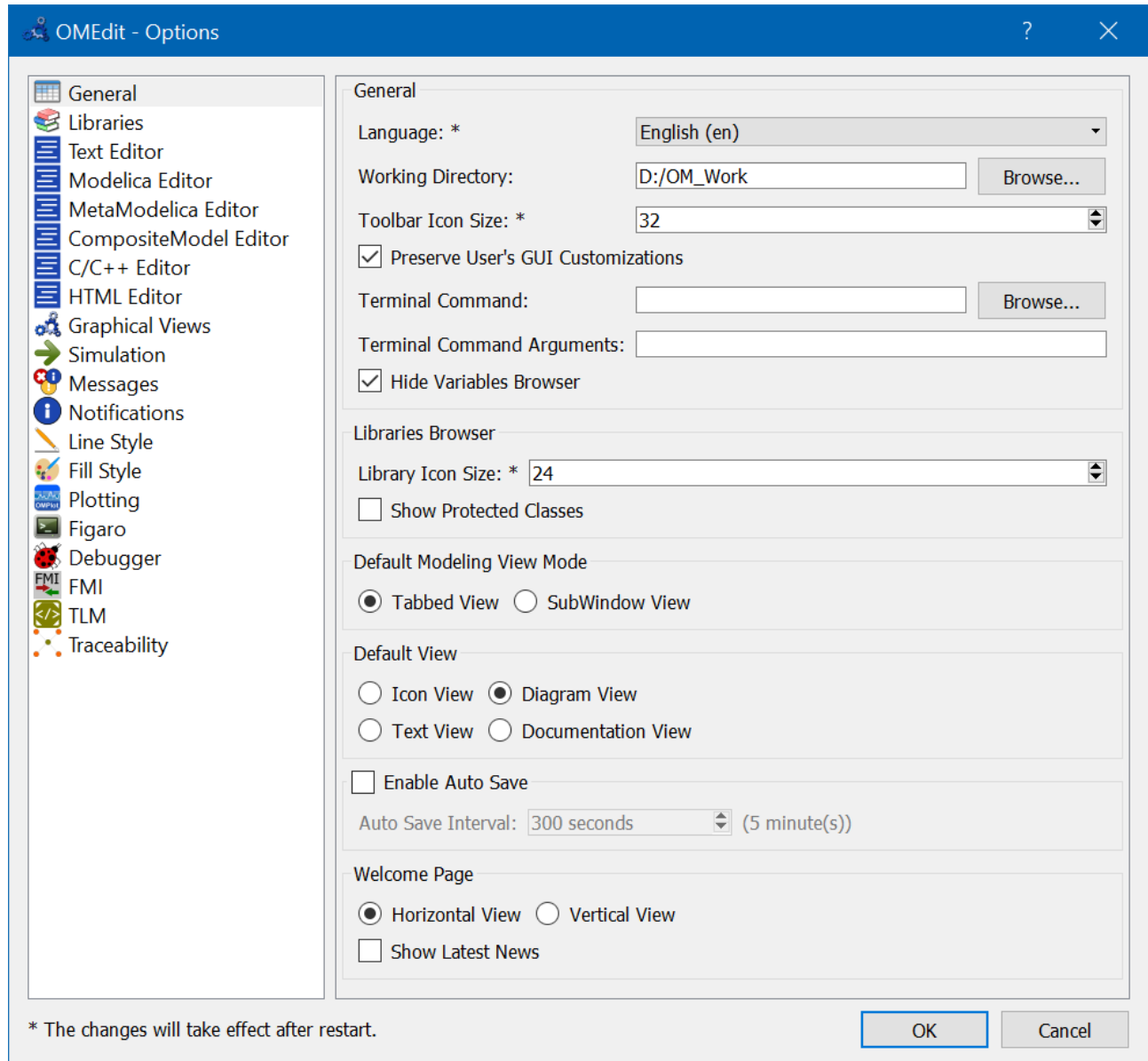
2.3 Model and data (txt) files

Before starting the reader should have these files available libraries EHPTlib and EHPTexamples installed, or available on their hard disks.

Several models of EHPTlib and EHPTexamples libraries make usage of text files. They are embedded in the libraries in the “Resources” folder.

The user can leave them as they are, change them, add new files with different names in that folder, or use different versions of txt files, specifying the right pathname in the models that make usage of them.

A position in which txt files can be put and are always found by OMEdit is the “working directory”, as defined in the Options window, such as:



If for instance we want a personalised version of friving cycle instead of the already provided Sort1, or NEDC or WLTC3, it can be put in a file named MyCycle.txt, which can be put in the working directory “D:/OM_Work” in the example of the screenshot here above).

In the model that is about to use it, we must substitute the built-in cycle name, e.g.

```
Modelica.Utilities.Files.loadResource("modelica://EHPTexamples/Resources/NEDC.txt")
```

with

“MyCycle.txt”

2.4 Prerequisites to this document

This document is written having in mind a technical learner, typically an Engineering student at a bachelor level. Any learner that has at least the technical knowledge of a bachelor student in Engineering should be able to understand it easily. It could be also useful to any full engineer that wants to learn something about electric and hybrid vehicles, somehow “hands-on”. In fact, Modelica offers learners an experience similar to the one we could have in a laboratory making tests on a full vehicle. With the advantage that we can measure everything without having to build complex measuring chains, and we can change everything in the physical and control parts of our system without having to spend a fortune or risk an explosion.

Regarding Modelica, only some very basic experience with this language and MSL is required; one should be able to use a simulation tool, however. It is best if that simulation tool is the open-source OpenModelica (through the GUI interface offered by OMEdit), since all the pictures and the models presented are represented using OpenModelica’s OMEdit.

2.5 About this document and library

Modelling of road vehicles is a highly multidisciplinary task. It may involve modelling with rotational mechanics, translational mechanics, electrical machines and converter, hydraulic components and systems, and more.

In particular, modelling electric and hybrid vehicles cannot absolutely avoid at least interfacing the electrical and mechanical domains, and their control.

Because of this, vehicle modelling and simulation in general, and electric and hybrid vehicle modelling in particular, are a field of engineering in which Modelica excels, because of its inherent capability to simulate multi-domain systems, from the ground up.

That is the rationale behind the choice of writing this chapter.

This chapter will not provide very detailed (thus huge) models, but instead models with a calibrated level of detail, which can help learning the basics and constitute a solid basis for building more compete and realistic versions of them.

3 My first EV model

The core of this chapter is simulation of EV’s and HEV’s *power trains*. All of the vehicle that is around the power train (chassis, auxiliary systems) will be dealt with only marginally.

A very basic representation of the vehicle’s power train, using whenever possible MSL components and using a Modelica diagram, is in figure 1. It is a Modelica *simulation model*, according to MS sect 1.2.

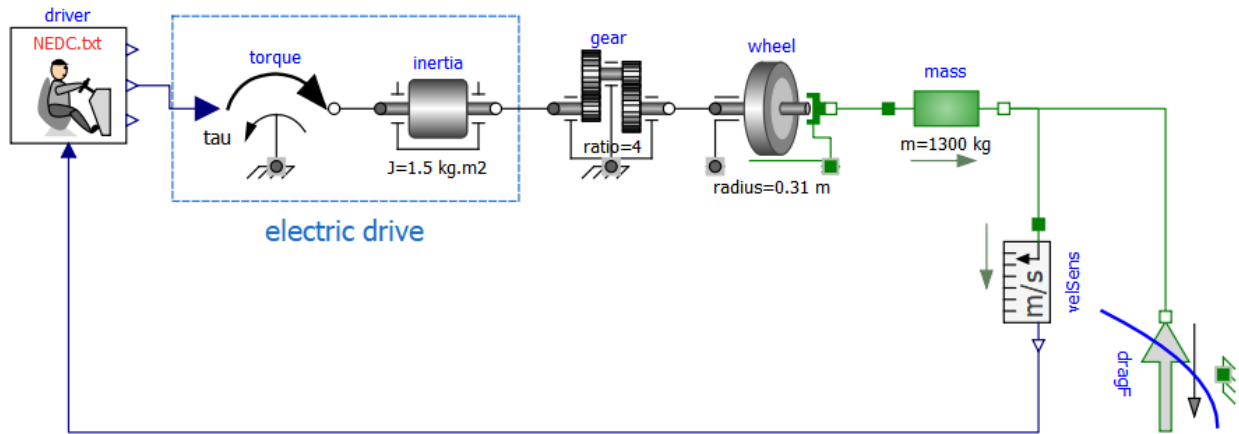


Figure 1. A first, very simple, EV Model (EHPTexamples.EV.FirstEV).

Let us now explain the graphical elements (submodels of the simulation model in figure) shown, starting from its left side.

- driver. It is impossible to simulate a vehicle without including in the simulation a driver model, unless we intend to consider a self-driving vehicle, which is totally out of the scope of this chapter. We will use a very simple driver model in this book, which are created on purpose, since they are not available in MSL. This driver model is described in section 3.4. It is designed to follow a kinematic drive, whose name is displayed in its icon ("Sort1.txt" in the example above). It is a *block*, i.e a Modelica model without any physical connector, which generates signals that should be interpreted to be torque signal: accelerator (above), brake (below), combined (midrange, connected to power train in the figure above).
- torque. This is the Modelica.Mechanics.Rotational.Sources.Torque. It plays the important role of connecting the "cyber" part of our simulation model to its "physical" part, in the direction cyber->physical; another connection between the two worlds, in the opposite direction, is made by the speed sensor at the right side of the picture above. The combination torque-inertia here simulates a power train that adds power to the signal issued by the driver. It could be for instance an electric drive, composed by an electronic converter and an electric motor.
- inertia. This is the inertia of the electric drive, e.g. the rotational inertia of the rotor of an electric motor present in the power train.
- electric drive. In this vehicle simplified model, the torque-inertia pair constitutes the electric drive, which receives as input a torque signal, positive when accelerating and negative when braking, and actuates it, considering the dynamics induced by the inertia. The electric drive absorbs or delivers power. This simple model does not show power conversion into electricity.
- gear. Commonly EV's have only a fixed ratio reduction gear. That is the role of gear in the above picture.
- wheel. This is an ideal rolling wheel. Note that in real-life cases wheels do not roll ideally. They have some *slip*, which is below 5% on good tyre-to-tarmac contact, and accelerations below 5-7 m/s². Since this book describes simplified vehicle modelling and simulation, we always consider ideal rolling; the accelerations we will see will be, indeed, well below the 5 m/s² threshold.
- mass. A very important part, though very simple, is the Modelica.Mechanics.Rotational.Components.Mass model called *mass*. It is the vehicular mass, in the example in figure 1400 kg. At its flange_a it is subject to propulsive force due to power train torque, at its flange_b to the resistance force to movement of the vehicle *dragF*.

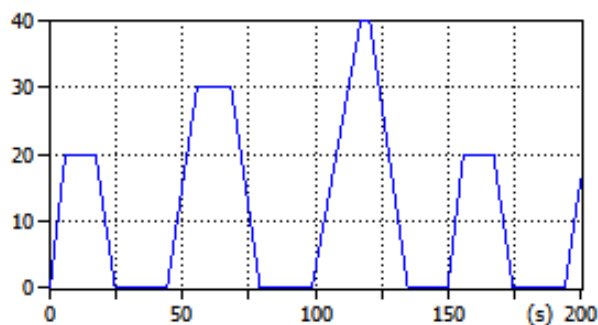
- velSens. As mentioned earlier, in terms of the “cyber-physical” simulation environment, which is Modelica, this sensor (as all sensors) creates a communication link between the physical and cyber parts of our simulation model. Looking at things in another way, it plays the role of a vehicle dashboard. The driver has in mind the kinematic cycle he wants to follow, and to do this he acts a closed-loop controller. Therefore, he looks at the actual speed through the dashboard (which in our mind displays the speed measured by velSens) compares it with the desired kinematic cycle, and acts on its commands (brake, accelerator) so that to reduce or nullify the error.
- dragF. This an object describing the vehicle resistance to movement. Since this force requires some rather sophisticated code to avoid unexpected effects, this is not a MSL model. It is instead a model extended from Modelica.Mechanics.Translational.Interfaces. PartialFriction. A description of this model will be given in sect. 3.2

3.1 My first simulation: the Sort1 Cycle

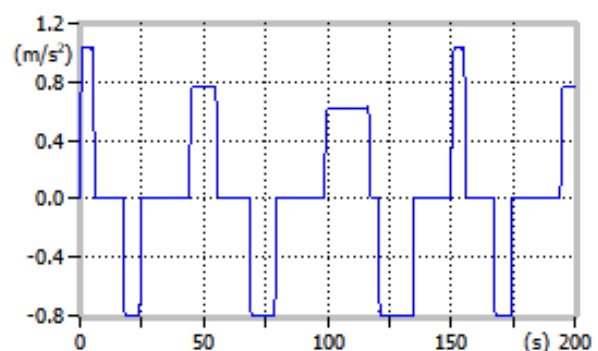
I imagine the reader wishing to see our first simulation model at work, even before knowing all the details of its component models.

Therefore, we introduce here a very simple driving cycle, which is able to show the basic characteristics of an electric vehicle motion. Although simple, the cycle we are going to introduce is standard. It is a standardised cycle used for urban bus testing, called “Sort1”. Below you can see how this cycle is defined in a text file (“Sort1.txt”), its plot over time, and the resulting acceleration.

```
#1
# Standardised Sort1 cycle
# columns: time, speed (km/h)
double Cycle(13,2)
  00.00    0.00
  05.39    20.0
  17.22    20.0
  24.17    0.00
  44.17    0.00
  54.99    30.0
  68.37    30.0
  78.79    0.00
  98.79    0.00
  116.71   40.0
  120.60   40.0
  134.49   0.00
  150.00   0.00
```



(file wbEHVpkg.EV.FirstEV_res.mat; x-var time)
driver.driveCyc.y[1]



(file wbEHVpkg.EV.FirstEV_res.mat; x-var time) mass.a

Figure 2. The sort1 cycle (left) and its resulting acceleration (right).

Note that these plots contain also the file name, so the reader can reproduce them by running the model with OMEdit, the GUI tool to use OpenModelica compiler. They will get virtually the same curves above, even though the visual aspect will be slightly different.

For instance, their OMEdit version could appear as follows:

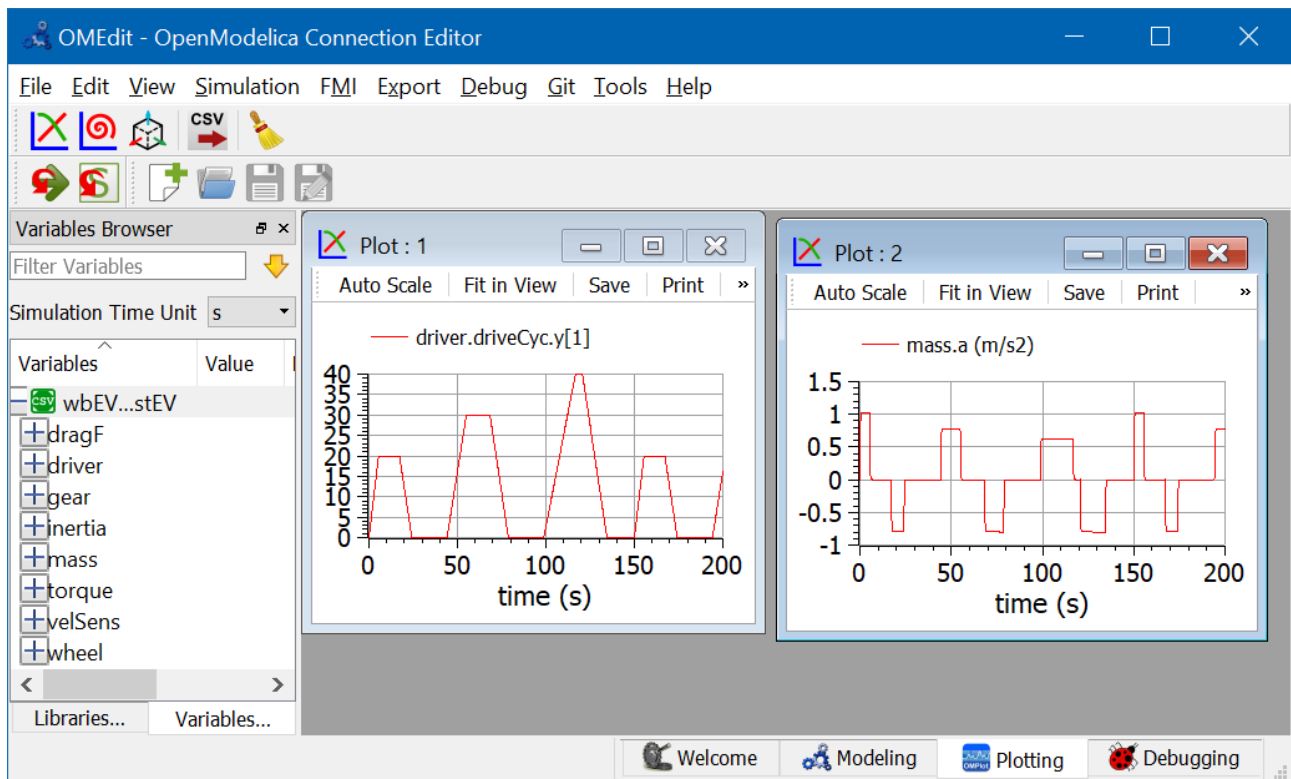


Figure 3. Plots corresponding to figure 2, but obtaining through OpenModelica.

At the left –hand side of this screenshot you see the list of variables, at the right side the two plots. Above, menus and the toolbar.

To select a cycle in one of the supplied models, the reader has just to tell the driver the cycle name. E.g. for the sort1 cycle the driver's parameter window under OMEdit might appear as follows:

OMEdit - Component Parameters - driver in wbEVPkg.EV.FirstEV

Parameters

General Modifiers

Component

Name: driver

Class

Path: wbEVPkg.SupportModels.PropDriver

Comment: Simple Proportional controller driver

Parameters

CycleFileName "Sort1.txt" ... Drive Cycle Name ex: "sort1.txt"

k 1000 ... Controller gain

yMax 100000.0 ... Max output value (absolute)

extrapolation Modelica.Blocks.Types.Extrapolation.Periodic ... Extrapolation of data outside the definition range

OK Cancel

Figure 4. The parameter dialog box for the proposed driver model.

The Sort1 cycle, in its period which is 140s, has three trapezoids, which are scaled progressively: the acceleration reduces from the first trapezoid on, the maximum speed raises.

The above plots, as well as plots of many other quantities, can be obtained simulating an EV using the simulation model shown in figure 1, with the addition of two power sensors, giving rise to the model shown in figure 5.

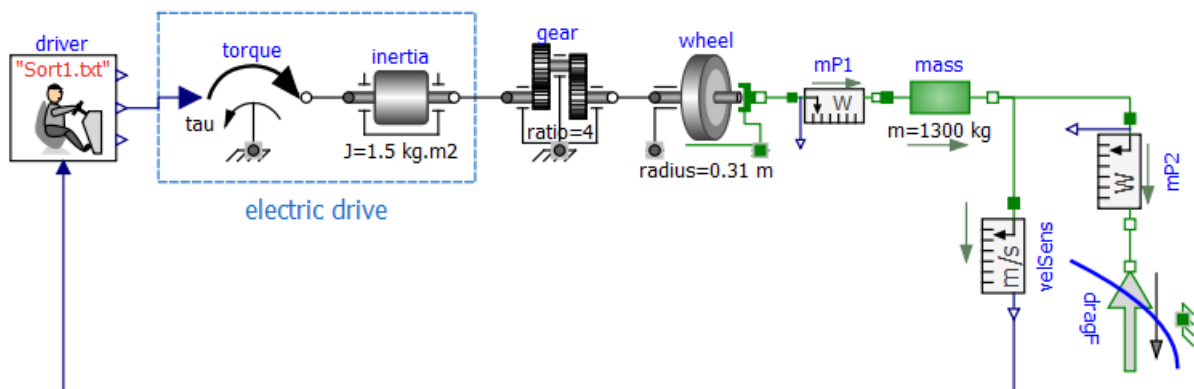


Figure 5: Model EHPTexamples.EV.FirstEVpow.

Running this model we could for instance have a look at two propulsion powers pP1 and pP2, shown in conjunction with the cycle. This can be done using PlotXY, which allows twin vertical axes (speed, green, is against the right vertical axis):

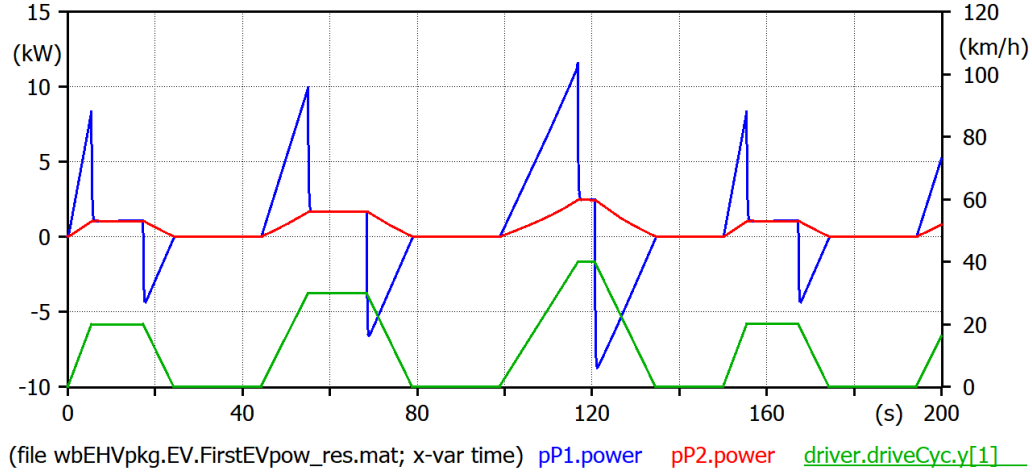


Figure 6 Powers under the Sort1 cycle and vehicle speed.

When speed is constant pP1 power equals pP2's. During transients the two differ, and the difference is the accelerating torque. The areas between red and blue curves are kinetic energies: when the red curve is above the blue ones, we are accumulating kinetic energy in the vehicle's mass, when the position is reversed, that energy is given back to the power train. The power train shown in figure 5 is totally reversible and has no loss, i.e. it can be assumed to be an *ideal EV power train* since, in addition to not having losses, is able to fully recover braking energy. Therefore, at the end of the simulation the net energy delivered by the electric drive is the area below red curve, which equals the area below the blue one (if we consider as initial and final two points in which vehicle speed is zero, i.e. time=0 and time=180s).

Drag force needs special consideration and will be discussed in the next section.

3.2 Details on DragForce model

According to vehicular mechanics, the resistance to movement (or *drag*) force R for horizontal paths is usually expressed as follows:

$$R = fmg + \frac{1}{2}\rho SC_x V^2 \quad (1)$$

The first term, containing the rolling resistance factor f , the mass m and gravity acceleration g , simulates rolling resistance; the second one simulates aerodynamic drag. In it, ρ is the air density, S is the cross-sectional vehicle area, C_x the longitudinal drag coefficient, V the vehicle speed. Indeed using the (1) in simulations is incorrect; we must instead use the expressions shown in (2).

$$\begin{cases} R = F_{appl} & \text{if } V = 0 \text{ and } F_{appl} < fmg \\ R = fmg + \frac{1}{2}\rho SC_x V^2 & \text{otherwise} \end{cases} \quad (2)$$

This requires special code: when V crosses zero, if the vehicle is pushed with a force lower than fmg , it stays stuck at $V=0$. If instead of (2) we used (1) when the vehicle is standstill horizontally and no torque is applied to the wheel hubs, the vehicle would move backwards!

More in depth

Note that when accelerating or braking the vertical load of front and rear wheels are different from each other. For instance, when braking the load on front wheels is higher. The first term in (1) refers to the total load on the four wheels, whatever its distribution. This is correct whenever all the wheels are nearly rolling, that occurs in normal conditions, except when we have very high accelerations (e.g. more than 5-7 m/s² on

good tyre-to-tarmac contact). Therefore, the models in this chapter are valid if we avoid very strong accelerations, such that road-to-tarmac slips larger than 5-8% occur.

Equations (2) describe a behaviour that occurs whenever there is the switch from stationary grip to dynamic grip. E.g., in vehicles, in clutches and brakes. Therefore, a model for the dragForce able to operate correctly can be obtained using something already available in MSL.

The model EHPTexamples.SupportModels.DragForce is built slightly modifying Modelica.Mechanics.Translational.Components.Brake to introduce equations (2).

The reader could look at the whole code if he wants, but it is not very important. Here we just comment a few rows of code.

First we define protected parameters A and B as follows:

```
protected
  parameter Real A = fc * m * g_n;
  parameter Real B = 1 / 2 * rho * S * Cx;
```

They are the two coefficients in equation (1)

Then we define the drag force, according to the R formulas in equation (2):

```
f - B * v ^ 2 * Sign = if locked then sa * unitForce else f0 *
  (if startForward then
    Modelica.Math.tempInterpoll(v, [0, 1], 2)
  else if startBackward then
    -Modelica.Math.tempInterpoll(-v, [0, 1], 2)
  else if pre(mode) == Forward then
    Modelica.Math.tempInterpoll(v, [0, 1], 2)
  else
    -Modelica.Math.tempInterpoll(-v, [0, 1], 2));
```

The part right of ‘=’ is just copied from Translational.Components.Brake, without even try to understand all details. The part left of ‘=’ is just equation (1), to which we added “Sign” that is the sign of V and considers whether the movement is forwards or backwards.

Finally, the brake model contains the variable free, which decouples the two brake parts when we are not braking. In our case this would occur when the considered wheel is not in contact with the asphalt, e.g. for a car “jump”. But our model is not able to simulate such things. We consider therefore wheels always in contact with the soil, and add the following row in DragForce:

```
free = false;
```

3.2.1 Numerical values

In the simulations of section 3 as numerical values of the parameter connected to drag force we have used, and will be using the following ones, corresponding to a typical small car at typical air pressure:

$m = 1300 \text{ kg}$ $\rho = 1.226 \text{ kg/m}^3$, $S = 2.2 \text{ m}^2$, $f = 0.014$, $C_x = 0.26$.

To see our dragForce at work, we can have a look at its flange force while still simulating the model shown in figures 1 and 5.

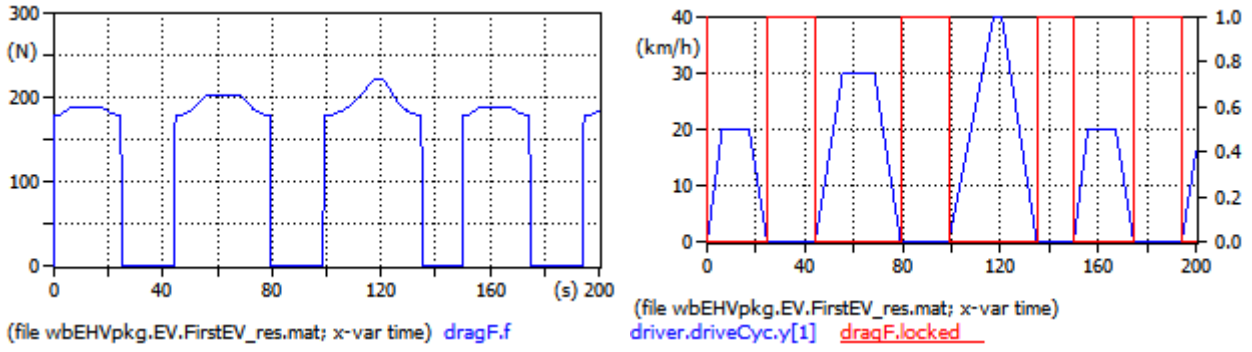


Figure 7. Drag force example, for a vehicle operating under the Sort1 Cycle (model EHPTexamples.EV.FirstEV).

In the left plot we see that the part of drag force not depending on speed prevails, being around 180 N; while the part depending on speed becomes more and more significant when we move from the first to second and third trapezoid. This is reasonable, since typically in cars the aerodynamic part of drag force becomes prevalent starting from speeds around 80 km/h. With our parameters this critical speed is found from:

$$fmg = \frac{1}{2} \rho S C_x V_{crit}^2$$

which gives $V_{crit}=81$ km/h.

In the right plot we see two curves: the red one must be read on the left vertical axis and contains speed. The green one is a boolean variable indicating when the drag force understands the vehicle to be stopped, and its value is to be read on the right vertical axis (1 means true, 0 false). When `locked=true` the first of (2) is valid, otherwise the second one applies.

3.2.2 Adding slope

In the previous subsections we introduced drag force without the effects of slope. When we have non-horizontal paths, we must modify it. Equations (2) become (α is the angle with reference to the horizontal plane):

$$\begin{cases} R = F_{appl} & \text{if } V = 0 \text{ and } F_{appl} < fmg \\ R = fmg \cos \alpha + mg \sin \alpha + \frac{1}{2} \rho S C_x V^2 & \text{otherwise} \end{cases} \quad (3)$$

In this case the name “drag force” is not so correct anymore, since R with negative α can be negative, i.e. favour vehicle movement, but we keep that name for simplicity.

Adding slope requires information about the path slope to be transferred to the new, slope-aware, enhanced drag force component we want to create.

In EHPTexamples this is done through a txt file. An example txt file for angles is supplied carrying the name Angle1.txt, and has the following content:

#1

First column: position, second column: angle (rad)

double Angle(8,2)

```
0.00    0.0
100.    0.
```

110.	0.1
200.	0.1
220.	-0.1
300.	-0.1
310.	0.0
400	0.

In EHPTexamples we have also a ready model named FirstEVAngle. The reader is prompted to try it, remembering that for this kind of simulations two txt files must be supplied: the cycle, inside driver (for instance the provided TestAngle.txt) and the angles, inside drag force (for example the provided Anelw1.txt).

We also note that models with angles are only partially implemented: typically we will not see the vehicle go to a perfect stop, when the slope is large, differently from the case of horizontal movement.

Consider for instance the supplied model FirstEVAngle with “Sort1.txt” as driving cycle and “Angle1.txt” as angle profile. The following plots show the vehicle speed (desired and actual) and the variable dragF.locked. We see that when time is between 80s and 100s, speed is nearly zero, but the vehicle is not completely stopped, and “locked” variable remains false.

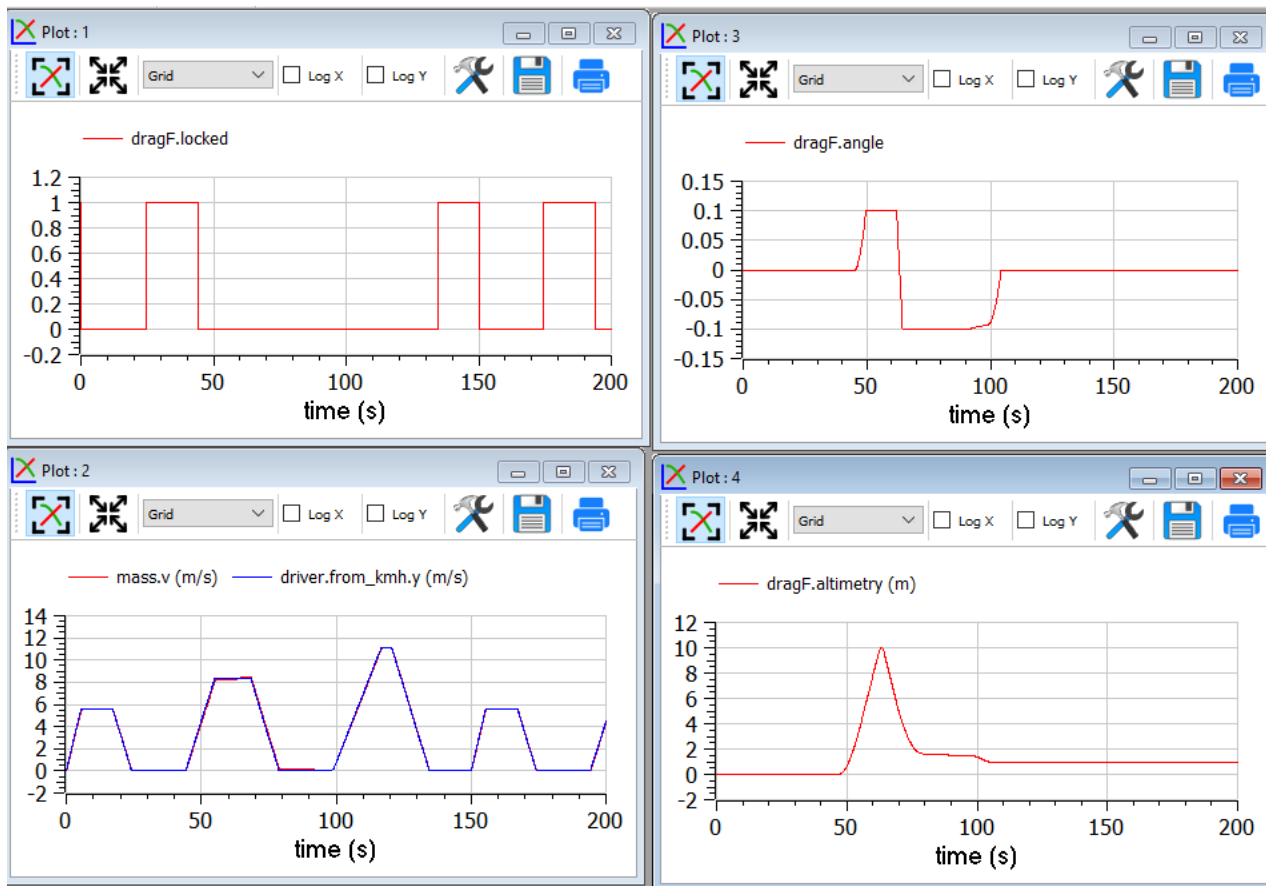


Fig. 8. Example when a negative slope impedes the vehicle to come to a complete stop.

Fixing this requires implementing a totally different driver's model, since the driver in these cases must explicitly push on the brake pedal, often requesting a strong negative torque. Details on this are shown in sect 3.4.2.

This enhanced driver model is out of the scope of the current version of EHPTlib library.

3.3 Simulating NEDC

NEDC (New European Driving Cycle) Cycle is richer than Sort1, and is thought for cars, not buses. It has an urban part, constituted each by three near-trapezoids, which is repeated four times, and a final extra-urban part.

To simulate this cycle the file name “NEDC.txt” must be chosen in the driver’s parameter window, as shown earlier for Sort1 cycle. This cycle looks as follows:

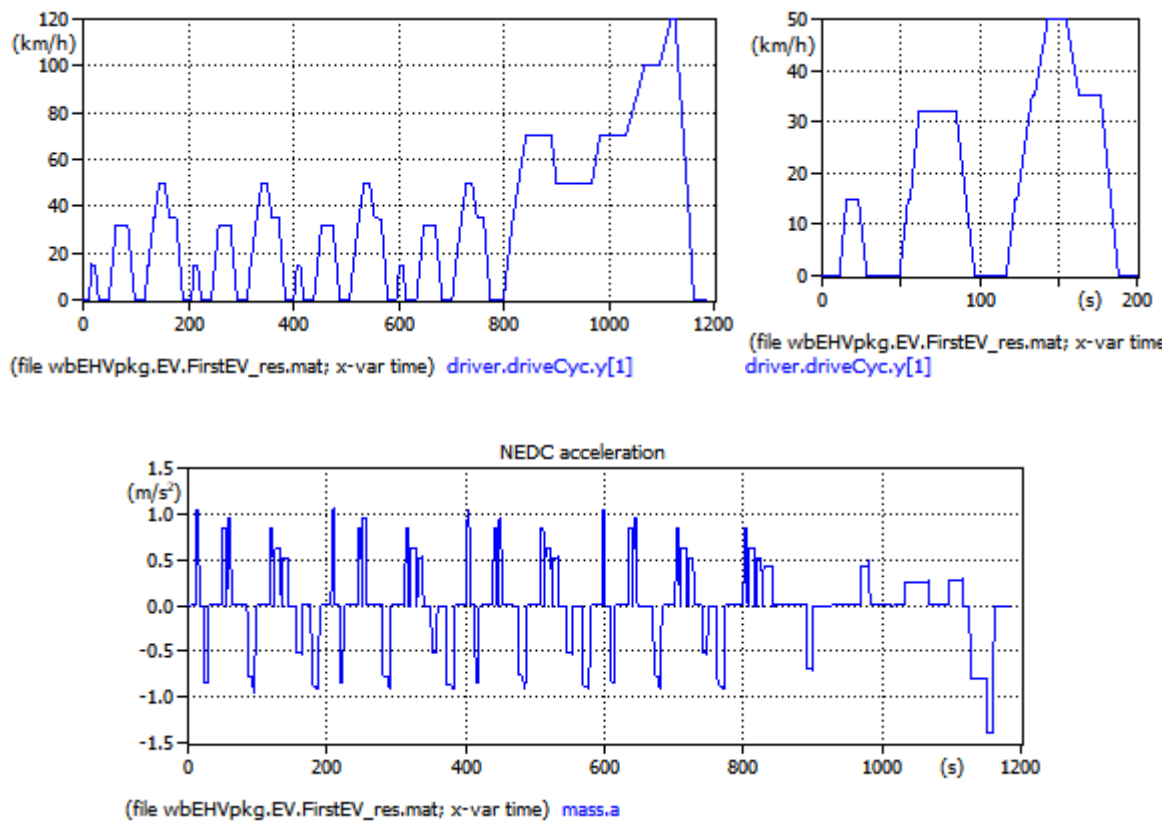


Figure 9. The NEDC cycle (top) and its resulting acceleration (bottom). Model EHPTexamples.EV.FirstEV. To reproduce select “NEDC.txt” in driver’s parameters dialog and use simulation time=1200s.

The acceleration plot shows very rapid acceleration changes even during ramps of the near-trapezoid shapes. These are since this cycle was thought for conventional vehicles, in which during acceleration gears are switched, and during switching speed has a plateau (see zoom above) where acceleration goes to zero or so.

These plateaux are meaningless for EVs and those working with these vehicles usually use modified versions of the cycle not having these plateaux.

A final note. NEDC cycle is very old. A new family of cycles has been developed as a worldwide effort, called WLTC (Worldwide Light-vehicle Test Cycles). They are planned to be progressively introduced in some countries, in particular in Europe, starting from September 2017 ¹.

Also for the NEDC cycle the power plots as those in fig. 6 can be plotted, as per the following figure 10. Naturally actual powers depend not only on the kinematic cycle, but also on the vehicle parameters: mass, drag force.

¹ <https://www.vda.de/en/topics/environment-and-climate/exhaust-emissions/wltp-worldwide-harmonized-light-vehicles-test-procedure.html>

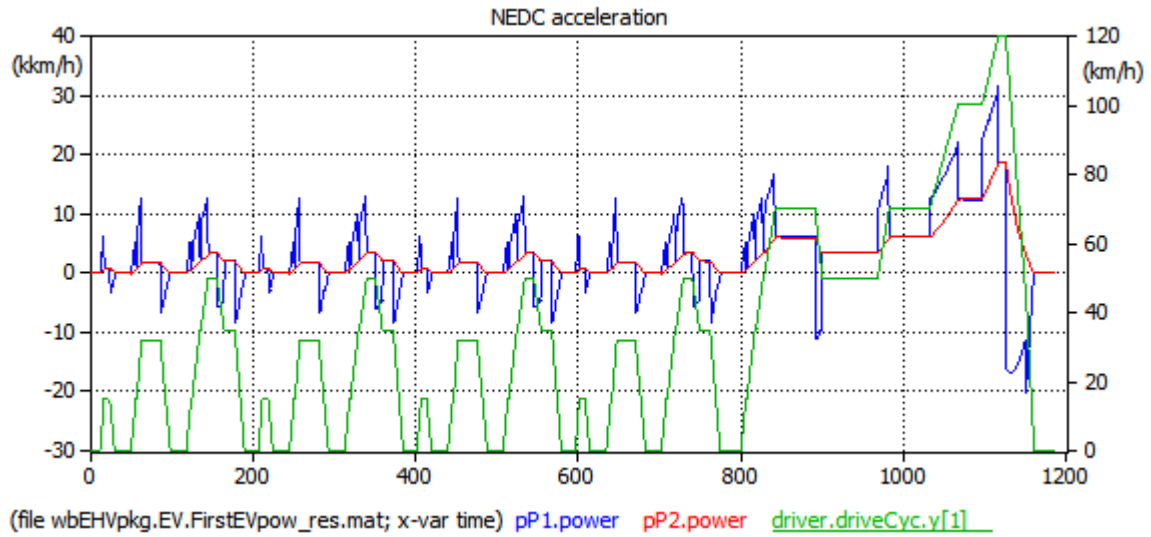


Figure 10. Some powers for a vehicle operating under the NEDC cycle. Model EHPTexamples.EV.FirstEVPow.

To reproduce select “NEDC.txt” in driver’s parameters dialog and use simulation time=1200s.

The peak power now reaches 30 kW inclusive of accelerating power, while the power needed to overcome motion resistance has a peak of 18.8 kW in the final part of the cycle, reaching 120 km/h.

3.4 Details of Driver model

The driver is a man, and therefore to simulate him models can be very complex. However, for our first EV, the driver has just to follow the given driving cycle, closely enough. In addition, this chapter is not to study people, but machines, i.e. our EVs and HEVs. Therefore, here a very simple driver model can be used. The driver model used is shown in the following figure:

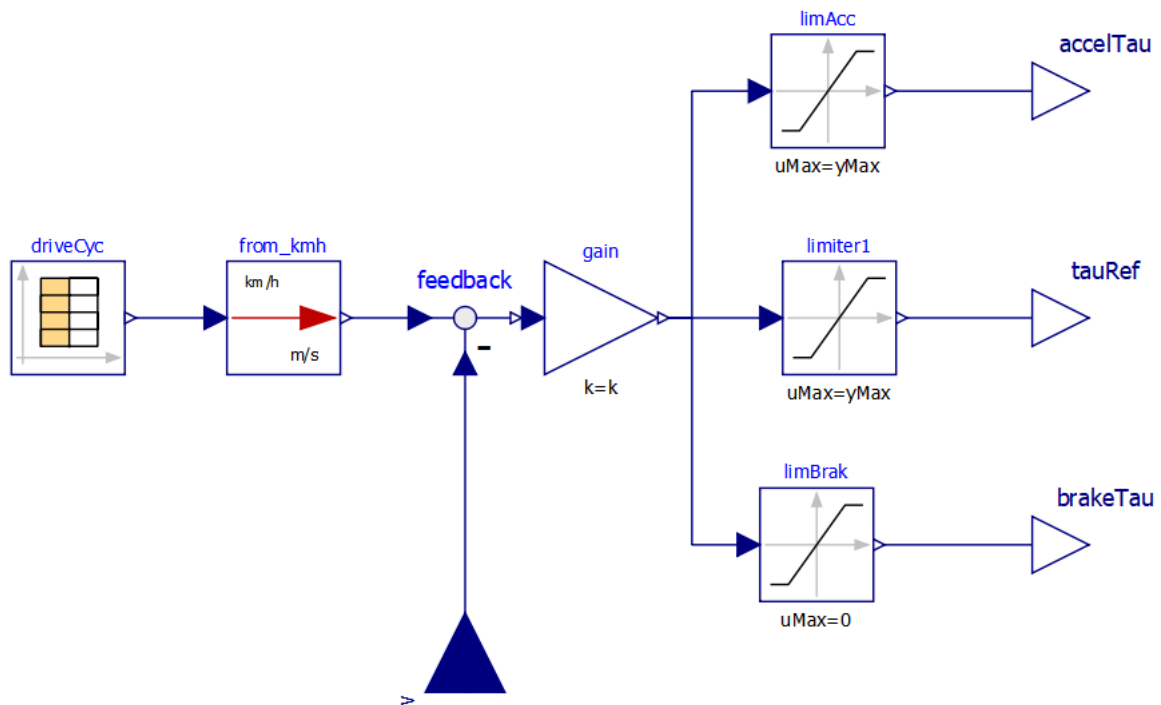


Figure 11. Proportional Driver model diagram.

It is called `PropDriver` because it implements a simple proportional control. A real driver commands propulsion through acceleration and brake pedals. In a simplified way, we can say that a positive

torque (i.e. so that the power train delivers power) is requested acting on the accelerator pedal, while a negative one is requested while acting on the brake. This is how the above `PropDriver` works. The two signals are also combined into a unique torque signal, named `tauRef`.

Signal `yMax` is imputed from the component GUI. Signal `accelTau` is limited between 0 and `yMax`, `tauRef` between `-yMax` and `yMax`, `brakeTau` between 0 and `yMax`.

Considering the situation more in detail, we must say that accelerator can require negative torque as well, when the foot is released with the vehicle running at a given speed. This occurs naturally in a conventional vehicle (internal-combustion engine based) since when gas is zeroed releasing the accelerator pedal, inner ICE losses cause a mild negative torque to be delivered. This natural behaviour helps smooth driving, since it allows moderate braking without requiring displacing a foot from a pedal into another. Therefore, it is replicated artificially in modern EV controllers.

Just as an example, we can consider the technique to perform this *accelerator release braking* similar to the one published in [2].

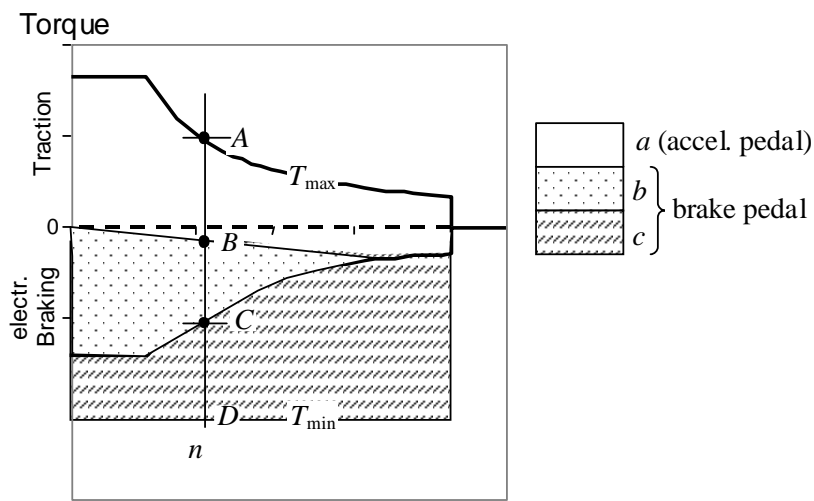


Figure 12. Operating regions showing a possible logic to manage pedal signals, and to get *accelerator release braking*.

According to that logic, an effective way of interpreting the driver commands, the zone between maximum and minimum electric drive torques T_{max} and T_{min} is split into three sub-zones in which the driver commands have different interpretations.

The zone *a* is limited by the upper part of the operating region and a straight line that passes in the point characterised by null torque and speed. At any generic Electric Drive speed n all points of segment *AB* correspond to different apertures of gas pedal position (GPP): for its maximum aperture, the maximum drive train torque T_{max} is requested; for minimum aperture, a negative torque (then a light braking action) is requested whenever $n > 0$; for intermediate apertures, linear interpolation is used.

Zone *b* (segment *BC*) is still available for stronger electrical braking action. Finally zone *c* (segment *CD*) adds mechanical braking.

Inside zone *a* there are parts with positive and part with negative torques. The latter realise the mentioned acceleration release braking. So, a totally released accelerator pedal corresponds to zero torque only at the drive zero speed; that torque becomes negative at higher speeds to get moderate braking action.

When the user moves his foot to the braking pedal, in the first part of its span it will cover zone *b*, then zone *c* in which mechanical braking is added.

Practical implementation of this technique of interpreting driver's signals is not provided in the files distributed along with this chapter. In them we will just use the combined torque signal `tauRef` in figure 11. Indeed, evaluation of vehicle's performance can be done not going into this detail, as will be seen in the remainder of the chapter.

3.4.1 *Proposed activity*

The reader is prompted to modify the driver model as shown in figure 11, in such a way to implement the braking logic discussed in Fig. 12. The output to be connected to the torque input in figure 1 can still be the central output, which contains all torque requests; evaluation of `accelTau` and `brakeTau`, can be useful to evaluate the effects of the implemented accelerator release braking, and to fine-tuning it, e.g. counting the number of times in which the driver moves his foot from accelerator to brake pedals in several cycles, to try to reduce it to a minimum.

3.4.2 *When the vehicle stops*

The very final part of real-life braking action needs some comment. A poor driver typically keeps a constant position of the brake pedal until stop; this causes a large jerk to occur at the final instant of the movement, since the braking force drops to zero near instantly when the vehicle comes to a stop, mainly by effect of the instantaneous change in the brake's torque when speed reaches zero. A wiser driver, instead, reduces the force acting on the brake pedal when speed is near to zero, so significantly reducing jerk at the time when speed becomes zero. Only the step due to the drag force remains (see eq. (2) where, in horizontal road, $F_{appl} = 0$), but this is much smaller than the previous case.

In EHPTlib, our PropDriver is a very simple model which tries to simulate the wiser driver, because PropDriver generates negative torque proportional to the speed error, which tends to go to zero near vehicle stop.

The situation is more complicated when the road has a negative slope (we are moving downhill). In this case, it may well happen that the "drag" force in eq. (3) is negative (i.e., tends to accelerate), in particular near zero speed, and therefore with the PropDriver, the simulated vehicle never comes to a complete stop. In real life, when the vehicle stops in a downhill route the driver will push the brake pedal stronger than in case of a horizontal one.

Examples of this downhill behaviour are shown in sect. 3.2.2. Despite of this glitch, the simulations with negative slope still show useful results in terms of accelerations, speeds, forces, etc..

4 **Easing vehicle switch (use of data record)**

If we want to simulate a new vehicle using "FirstEV", we have to manually change the parameters describing model's behaviour: the drivers's control constant, engine inertia, gears ratio, etc. It is much easier to re-formulate the model in such a way that switching a vehicle involves just dragging a "data" component, containing all the model's parameters. This is easily done using a Modelica record. Consider model "EVdata", whose diagram is as shown in figure 13.

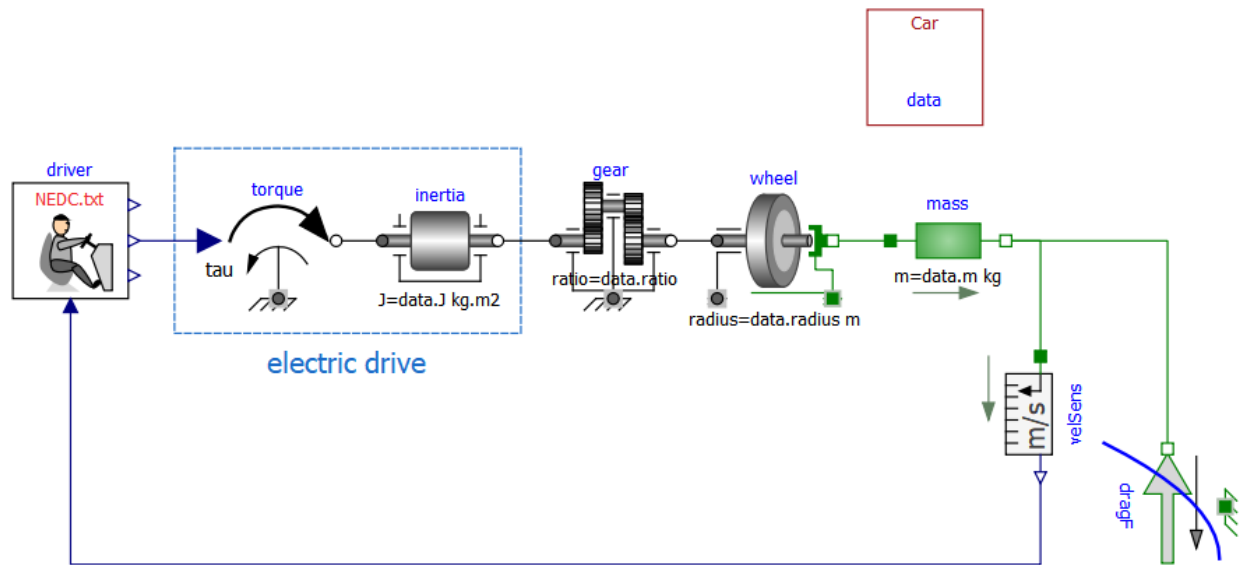


Figure 13. A Model allowing easy vehicle switching (EHPTexamples.EV.EVdata).

This model is formulated so that the parameters are all encapsulated into the record named “Car”, whose instance is named “data”. Changing from a vehicle to another requires just deleting “Car” model and dragging a different record, for instance the supplied “Bus” model. This new instance of the record carrying vehicle data must be named “data”.

The examples supplied contain the following numerical data:

<pre> record Bus parameter Real m = 16000; parameter Real ratio = 6; parameter Real radius = 0.5715; parameter Real J = 5; parameter Real Cx = 0.65; parameter Real S = 6; parameter Real fc = 0.013; parameter Real rho = 1.226; parameter Real kContr = 1000; end Bus; </pre>	<pre> record Car parameter Real m = 1300; parameter Real ratio = 3.905; parameter Real radius = 0.31; parameter Real J = 1.5; parameter Real Cx = 0.26; parameter Real S = 2.2; parameter Real fc = 0.014; parameter Real rho = 1.226; parameter Real kContr = 100; end Car; </pre>
---	---

The reader is invited to perform several simulations changing vehicle behaviour, using these data records.

The reader can also modify other supplied models to allow also for them “easy vehicle switch” using record.

5 Map-based EV models

The EV model used earlier was to understand the very basic behaviour of an EV power train, but has too many limitations for practical usage. In particular:

- it does not consider any battery model and behaviour
- it does not consider power train losses
- it does not consider that braking can occur with either power train or mechanical brakes
- it does not consider that power trains have limits in the torque and power they can deliver or absorb.

Before presenting a model overcoming these limitations, in the next section we first describe what we intend for a map-based electric drive.

5.1 The idea of map-based electric drives

There are several cases in which it is not important to model all the details of a physical system. We can therefore use simplified models, which do not reproduce fast transients, but are enough for the scope of EV and HEV simulation: evaluate torques, forces, powers, battery SOC, etc.

As regards EVs, the component that we will simulate with maps (map-based component) is the *electric drive*.

Note that an electric drive is not just a machine: it is an electrical machine, its feeding converter (often called Power Processing Unit, or PPU) and the related control. Figure 14 shows the idea of electric drives, when they are fed from a DC source, which is the case of electric vehicles.

Two typical electric drives are:

- DC drives: a converter feeds a DC machine (fig. a)
- three-phase drives: a voltage-sourced converter (VSC) feeds a three-phase machine (fig b). The machine, in turn, can be induction (asynchronous) or synchronous; in the latter case for EVs the most common case is a permanent-magnet synchronous machine.

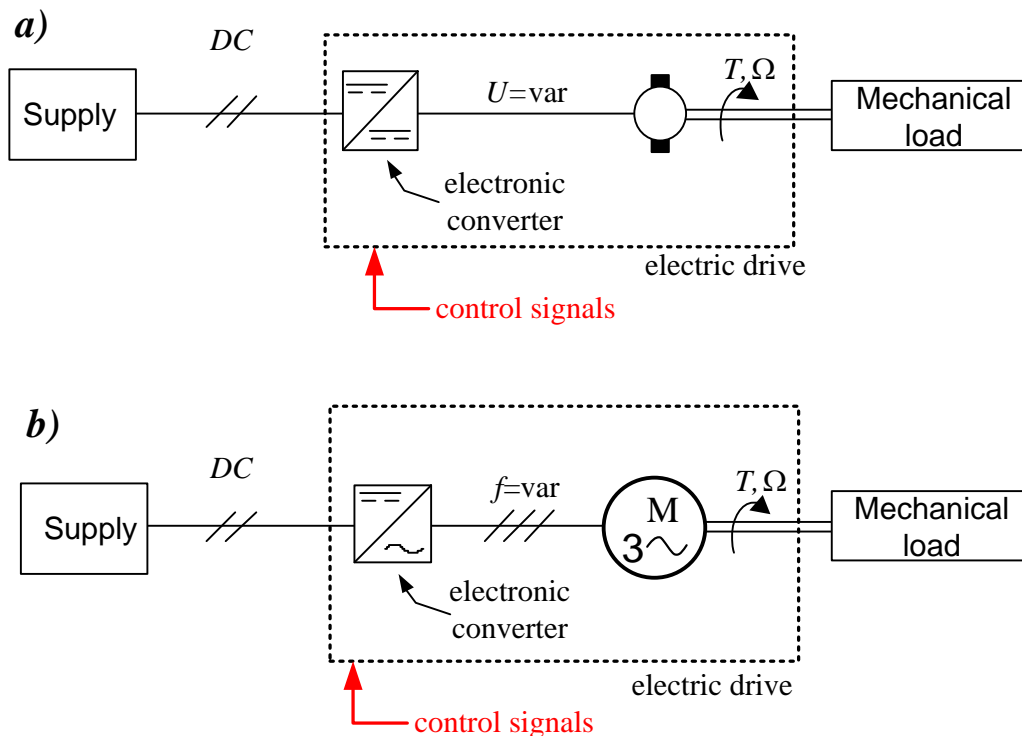


Fig. 14: DC-fed Electric drives: based on a DC machine (a)) or an AC machine (b)).

In several cases all electric drives for simplified EV simulation purposes can just model:

- the inertial behaviour of machine rotor
- the torque and power limits
- the energy losses evaluated through maps: a fixed efficiency for every possible operating point
- the capability to respond to control signals.

Regarding torque and power limits, often as a first approximation just symmetric limits, a base speed, and a maximum speed can be considered as represented in figure 15.

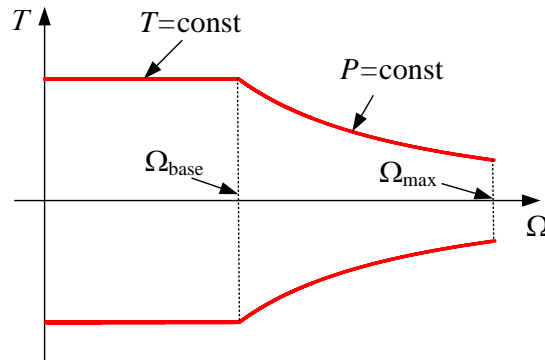


Figure 15: the operating region of an electric drive, showing torque-limited and power-limited zones.

All points between the two red curves and the vertical axes $\Omega=0$ and $\Omega=\Omega_{\max}$ are acceptable operating points for the drive. Each of them implies a given efficiency: ratio of mechanical to electrical powers when the machine is operating as a motor (positive torques), or electrical to mechanical when it is operating as a generator (negative torques i.e., for a vehicle regenerative braking). Many electric drives can be modelled this way: the distinction between DC drive, Induction drive, and PM-SM (permanent-magnet synchronous machine) drive disappears.

In SMEHV library we propose two map-based models of electric drives, which illustrate this kind of modelling: the operating region as in figure 15, with efficiency values inside that region, the electric motor inertia, control. A detailed explanation of this model is in sect.5.5.

The models are OneFlange and OneFlange2. OneFlange2 does the same of One Flange, but in addition it allows to implement max and minimum torque as a function of the angular speed, through curves supplied via an array taken from an input file. Wit OneFlange2, therefore, we do not need to have exact the shape of limits as shown in figure 15. In particular, they can be asymmetrical, since often during regenerative braking we can afford lower torques and powers than those allowable while in traction.

5.2 Flexible use of maps from file

“maps”, for specific consumption, limits efficiencies, can be defined through static values, such as max torque and max power in figure 15.

This is sufficient for the simplest cases.

For more complicated cases they are better introduced through matrices, which in turn can be defined as input arrays, or taken from a file.

EHPTlib allows all these techniques:

- maximum torque and power can be generally set through input so that shapes such as in figure 15 are automatically generated
- electric machine efficiencies and internal combustion specific consumptions can be either entered through matrices inputted as mask parameter

- maximum and minimum torque limits, efficiencies, specific consumptions can also be taken as matrix (that we often call “maps”) from an input txt file.

When maps are taken from a file, multipliers are useful to reuse a map for a different vehicle.

Take for instance a consumption map. Factors `scSpeedFactor` and `scTorqueFactor` are defined, which multiply the computed speed and torque before entering the table, `scConsFactor` multiplies the table output before further processing.

Consider for instance the following map:

```
# First row: (from column 2) speed (rpm)
# First column (from row 2): torque (Nm)
# body: spec. consumption (g/kWh).
double iceSpecificCons(10 6)
0.      100      200      300      400      500
10      630      580      550      580      630
20      430      420      400      400      450
30      320      325      330      340      350
40      285      285      288      290      300
50      270      265      265      270      275
60      255      248      250      255      258
70      245      237      238      243      246
80      245      230      233      237      240
90      235      230      228      233      235
```

If I want to use a map from a file with the same shape as this for a vehicle having max speed=1000 rpm, max torque=200 Nm, max consumption 500g/kWh, I will use:

```
scSpeedFactor=60/(2*pi)*500/1000,
scTorqueFactor=90/200,
scConsFactor=500/630.
```

Note that internally speeds are computed in rad/s; since here in the table here they are in rpm, I must include rad/s to rpm conversion.

If, instead, I want to use exactly this map, I will use:

```
scConsFactor=scTorqueFactor=1, scSpeedFactor=60/(2*pi)
```

In general, all input and output factors for matrices are not used when data is not taken from a txt file.

5.3 EV model at work

5.3.1 Basic description

In the model presented in this section, we will remove the limitations listed in sect. 5. Nevertheless, it will not be very detailed for two reasons:

- to understand things, it is often better to have as simple models as possible
- to allow simulation of the large timespans required for vehicle simulations (minutes or tens of minutes) it is not wise to go into minute details such as commutation of valves inside vehicle power electronic converters.

The model we propose has the diagram shown in figure 16.

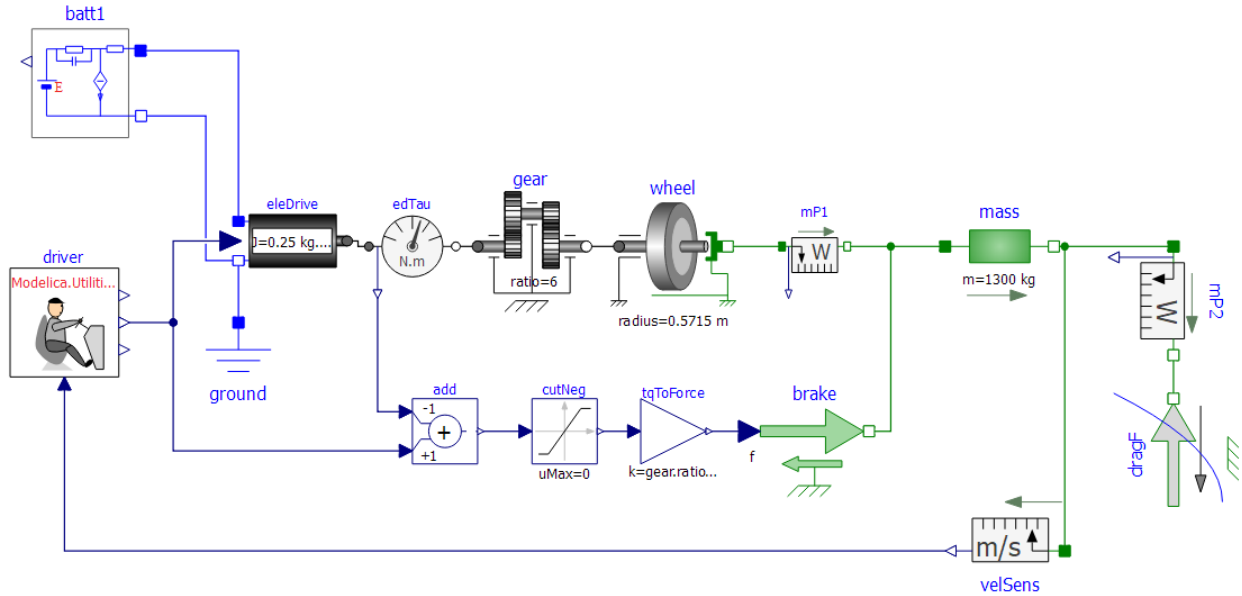


Figure 16: Map-based EV model diagram (Model EHPTexamples.EV.MBEV).

This model requires, to run, the power train efficiency to be specified directly on the eleDrive mask, or on a txt file. To reproduce the results shown here the reader can use the provided “EVmaps.txt”. How to link a txt file with a model is explained in section 2.3.

Some comments about the added components:

1. the battery model will be discussed in section 5.4
2. the drive train (eleDrive) model is a map-based electric drive model, as introduced in sect. 5.1, and discussed more in detail in in sect. 5.5
3. the brake action is simulated by just adding a braking force to the mass’s flange_b. A more sophisticated brake model exists in MSL (Modelica.Mechanics.Translational.Components.Brake), but it was not chosen because in this book we prefer simplicity to completeness.

Regarding the above point 3, it is interesting to note that sum of torque is automatically made by any Modelica tool (here OpenModelica), since torque is a flow variable and flow variables are automatically algebraically summed up when several connectors are connected to each other. Here, the requested braking force is non-zero only when electric braking from eleDrive is insufficient.

5.3.2 Sample simulations

Simulation of NEDC cycle using the model in figure 16, with mapsOnFile=true inside eleDrive, gives the results shown in the following pictures (fig. 17; add.u2 is the reference torque, requested by the driver).

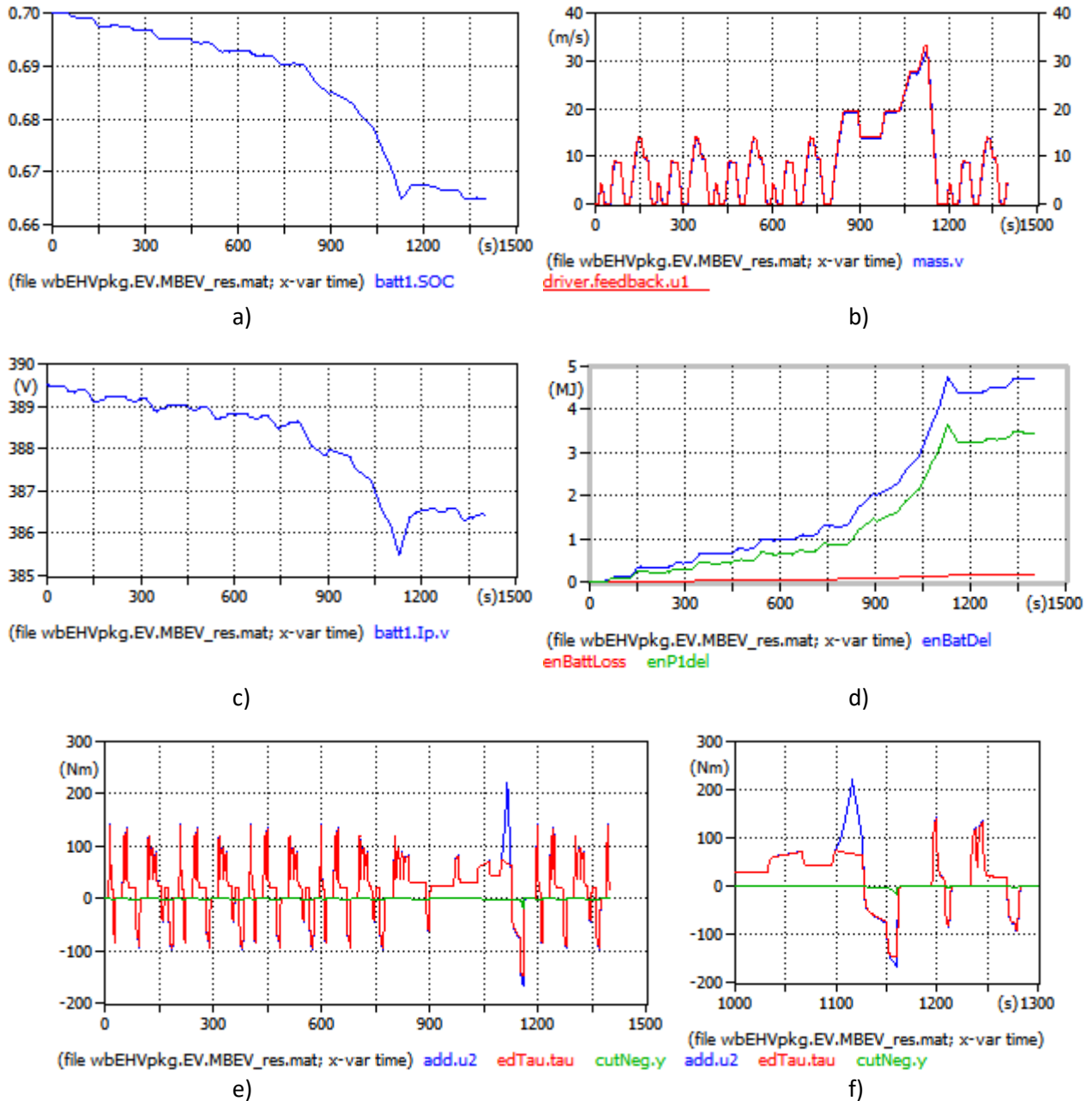


Figure 17. Plots related to the model in figure 16 operating under NEDC cycle.

Plot a) contains the battery Stat-of-charge (SOC)

Plot b) compares actual (red) and desired speed, and shows that they are very near to each other.

Plot c) contains the battery voltage

Plots e) and f) show torque request (blue), torque delivered by the electric drive (red) torque corresponding to the mechanical braking force (blue). We can note that:

- between $t=1100$ and 1126 s not all the requested torque can be delivered. This is because the max power= 22 kW is hit. This is common in vehicles, whenever drivers request torques that are larger than maximum available (due to torque or power limitations), but in this case has no practical effects on the desired speed
- negative torques are nearly always obtained with electric braking; mechanical braking here occurs only at the final deceleration of the extra-urban part (plot zoom in the lower right corner).

In practice, however, real EV's use more mechanical braking. Compare with figure 62 in sect. 9.4.1

Plot d) shows energies. In this case energies are obtained using direct equation writing, to avoid the diagram to become too crowded. They are simply the following ones, in model MBEV:

```
Modelica.SIunits.Energy enBatDel, enP1del, enBattLoss;
equation
der(enBatDel) = (batt1.p.v - batt1.n.v) * batt1.n.i;
der(enP1del) = mPl.power;
der(enBattLoss) = batt1.powerLoss;
```

The variable `enBatDel` is the energy delivered by the battery, while `enP1del` is the energy supplied to the left flange of mass: the two powers differ from each other because of the non-unity efficiency of the power train `eleDrive`. If we select, in this component, “`mapsOnFile`”=false, and leave the default `effTable` (which means unity efficiency) the two curves overlap.

Using this simulation, we can evaluate the Wh per km this vehicle requires under the given cycle. We must remember that a “rule-of-thumb” for common usage is that the required energy is around 100 Wh per tonne for a car operating mainly in urban areas.

Here we have, dividing the final value of `enBatDel` for the final value of `mass.s`, taking into due consideration the units of measure, 116 Wh/km, which is good for a 1300 kg car.

If we repeat the simulation using the provided WLTC class 3 cycle (“`WLTC3.txt`”) for its duration (1800 s) we find 133 Wh/km, which confirms what we already knew that WLTC3 is more severe than NEDC, and corresponds very well to the 100 Wh/(tonne.km) rule-of-thumb value.

The model `EleDrive` in figure 16 is a component able to operate at a maximum permissible torque and maximum permissible power. These limits can be plotted in a torque-speed diagram, in which we can put also the points in which the vehicle operates under a given cycle. Figure 18 Contains a parametric plot showing the vehicle operating points under NEDC, inside the operating map.

The red curves are the maximum and minimum allowed torques (cf. fig. 15). The curve requested by the driver curve stays most of the time inside and is shown in blue; when it should have gone beyond, it is cut and shown in green.

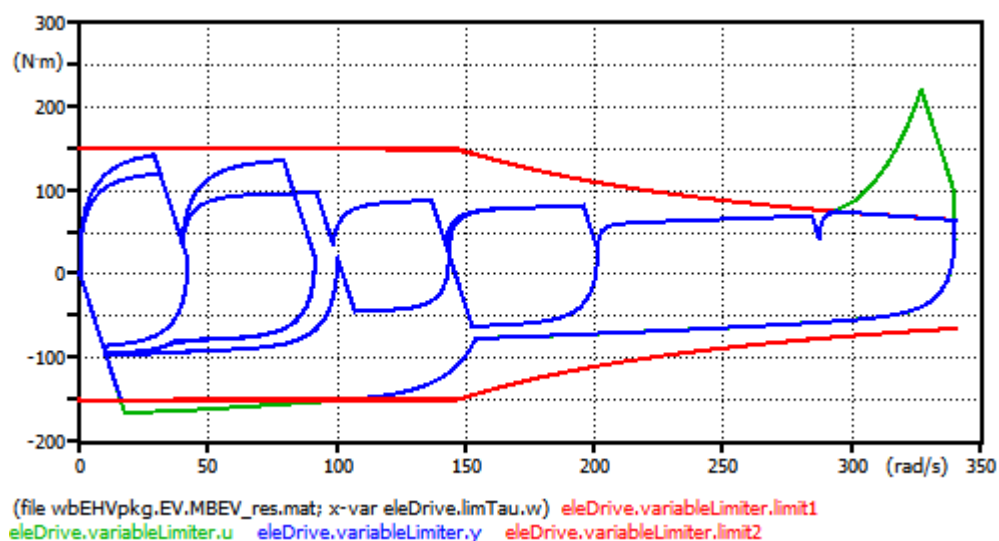


Figure 18. Actual torques (blue) shown inside the limits (red).

A typical EV study consists of the evaluation of the actual operating points (i.e. the points belonging to the green curve in the diagram above), in a diagram plan containing the map efficiency: in case they stay too often in zones having low efficiency, something must be changed in the power train design.

More on the efficiency maps in sect.8.

The dialog to introduce drive-train parameters is as in figure 19. Note that writing the efficiency map directly is not that practical, therefore is written into a file. In the example provided the file is the provided “EVmaps.txt” and the variable name stored in that file is “effTable”.

OMEdit - Element Parameters - eleDrive in EHPTexamples.EV.MBEVdata

Parameters

General Modifiers

Component
Name: eleDrive

Class
Path: EHPTlib.MapBased.OneFlange
Comment: Simple map-based model of an electric drive

Parameters

powMax	22e3	W	Maximum mechanical power
tauMax	200	N.m	Maximum torque
uDcNom	100	V	nominal DC voltage
wMax	1000	rad/s	Maximum drive speed
J	data.J	kg.m2	Rotor's moment of inertia
mapsOnFile	true		= true, if tables are taken from a txt file
mapsFileName	EVmaps.txt		File where matrix is stored
effTableName	effTable		Name of the on-file efficiency matrix
effTable	[0, 0, 1; 0, 1, 1; 1, 1, 1]		rows: speeds; columns: torques; both p.u. of max

OK Cancel

Figure 19. Dialog box to set parameter of `eleDrive` component (`OneFlange` class).

Just to be more explicit, the provided EVmaps.txt has the following aspect:

```
#1
# efficiency - rows:speeds pu, columns:torques pu
#
```

```
double effTable (6,6)
    0.00  0.00  0.25  0.50  0.75  1.00
    0.00  0.75  0.80  0.81  0.82  0.83
    0.25  0.76  0.81  0.82  0.83  0.84
    0.50  0.77  0.82  0.83  0.84  0.85
    0.75  0.78  0.83  0.84  0.85  0.87
    1.00  0.80  0.84  0.85  0.86  0.88
```

Note that if `mapOnFile=false`, the drive will have unit efficiency.

The efficiencies in the file are with reference to values of rotational speed, p.u. of `wMax` and torque, p.u. of `tauMax` (see dialog box in figure 19).

It is interesting to evaluate the amount of energy recovered when braking, that would otherwise be lost. To do this, the model can be slightly modified in a graphical way, or with addition of the following code:

```
Modelica.SIunits.Energy enBraking;
der(enBraking) = if pP1.power > 0 then 0 else -pP1.power;
```

This braking energy is compared with `pP1` energy in the plots in figure 20:

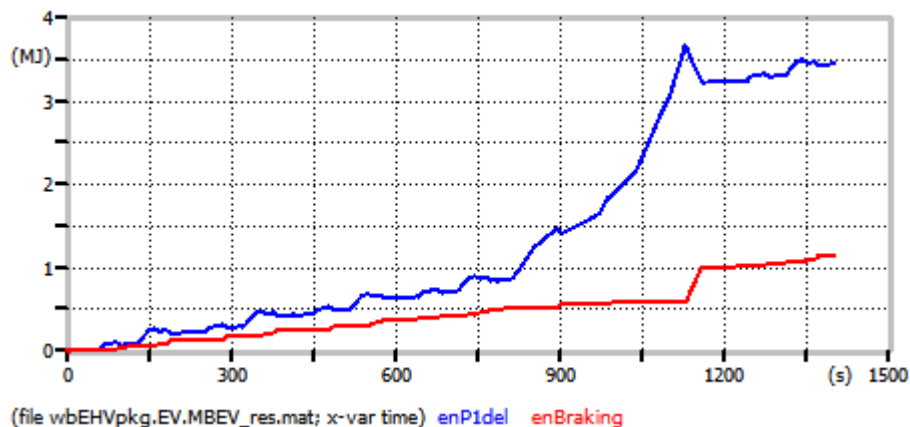


Figure 20. Braking Energy (green) compared with total energy (red) to vehicle mass.

In compliance with what said in section 4, also for MBEV there is a “data” version, which eass model switching, called Model EHPTexamples.EV.MBEVdata.

5.3.3 Comparison of Tabular and Formula-based loss computation

Since October 2022 the electric drive can be implemented using one of the three available models:

- OneFlange. It allows specifying maximum torque and power
- OneFlange2 contains an improvement onto OneFlange: here the user can implement max and minimum torque as a function of the angular speed, through curves supplied via an array taken from an input file. This allows more flexible definition of limits than just specifying max torque and power. This model is not parameter compatible with OneFlange. This has caused this new model to be introduced
- OneFlange2LF. This is a variation of OneFlange2: here efficiency is computed by means of a mathematical formula, instead than using a table read from a file. Details on this difference in sect. 9.2 and 9.3. This model is not parameter compatible with OneFlange and OneFlange2. This has caused this new model to be introduced

In the following plots we compare the losses as computed with map and formula. The considered map is the one reported in sect. 11, while the parameters of the loss formula are those used in sect. 13.

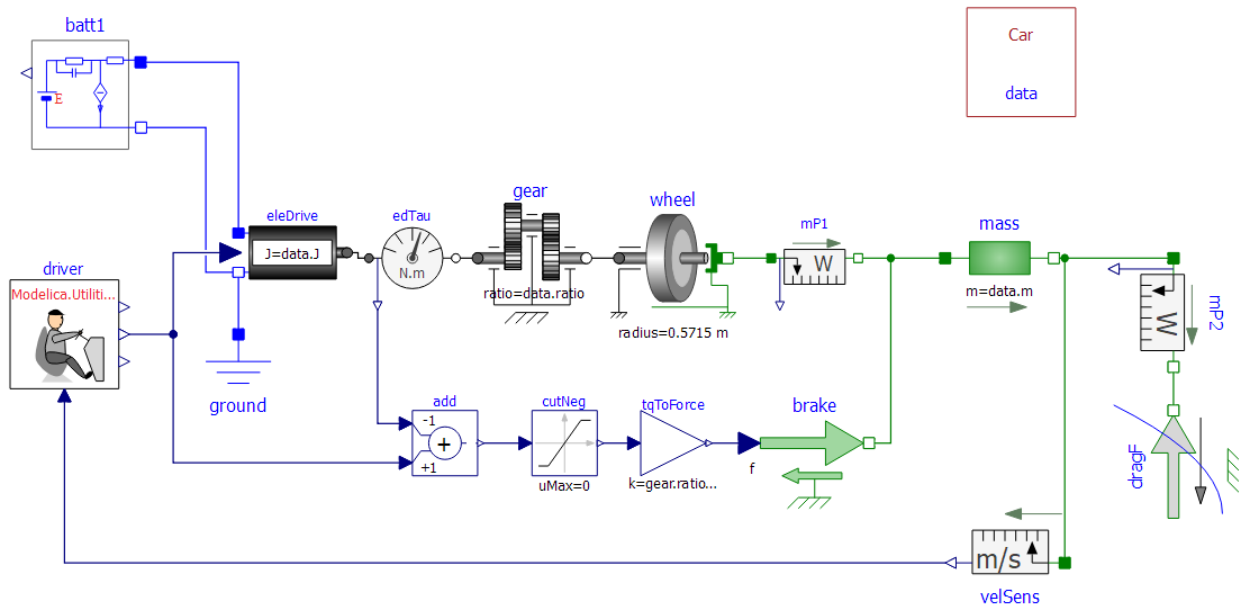


Figure 21: Map-based EV model diagram using data record and loss formulas (Model EHPTexamples.EV.MBEVdataLF).

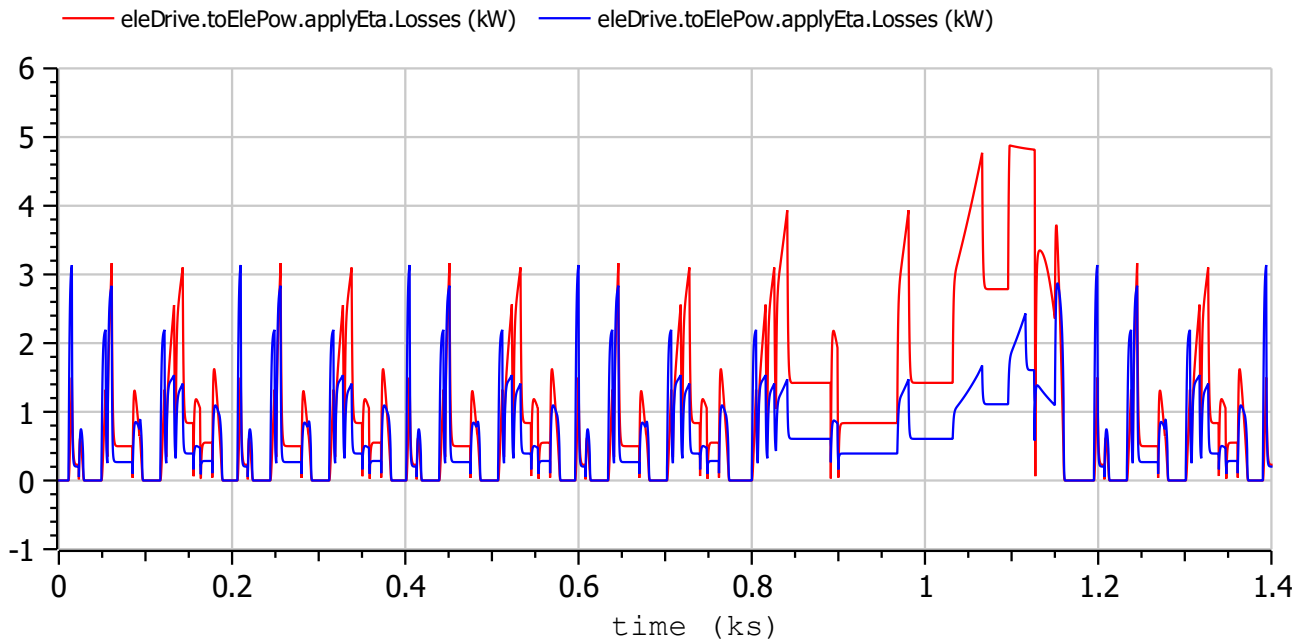


Figure 22: Comparison of losses as computed in models MB EHPTexamples.EV.MBEVdata (red) (blue) and EHPTexamples.EV.MBEVdataLF (blue).

5.3.4 Model limitations

The proposed EV model is simple and thought for teaching.

Many more additions could be made for industrial production models either in terms of completeness (e.g. some vehicles have switching gears, which is not included in our model), or

accuracy: being map-based, this model is not able to depict the machine currents and other electric quantities in detail.

5.3.5 Proposed activity

Modify MBEV in such a way that all braking energy is lost in brakes. This will give a better picture than that in figure 20 of the advantages of braking energy recovery.

5.4 Modelling batteries

Our battery model has the structure as shown in figure 23 that replicates, with additions (in red), our models' icon. It is rather directly drawn from paper [1] which, published in 2000, has had a wealth of 274 citations and 4321 full text views since then², and many papers have appeared in subsequent years that replicated the approach used there. Although the original paper [1] referred to lead-acid batteries, that same model has proved to be adequate also for other kind of batteries, such as lithium based [6].

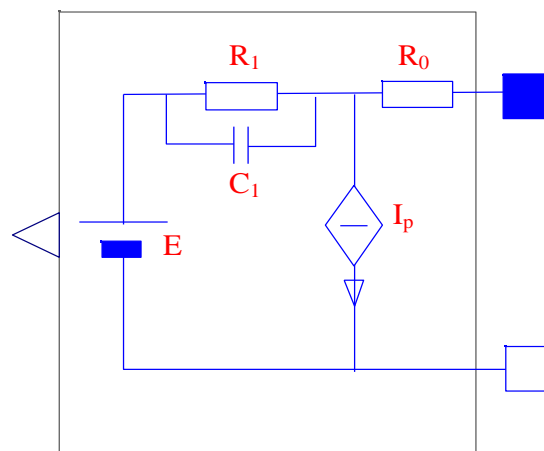


Figure 23 The icon of our battery model (EHPTexamples.SupportModels.Batt1).

The model consists of an electromotive force E , an R-C network, a parasitic current I_p . The charge stored in the battery is the integral of current flowing through E , while the integral of I_p is charge that is lost.

In paper [1] E is a complex function of the battery state-of-charge SOC, and I_p is a function of battery voltage and temperature. However, this chapter is not a study about batteries. We can add here two simplifying hypotheses:

- we take E as proportional to the stored charge. Therefore E itself can be substituted by a capacitor, having a very, very large capacitance, and a voltage equal to the so called “open circuit voltage (OCV)”, i.e. the voltage that can be measured when the battery is disconnected from the outside circuit, and has stayed disconnected for a while. Therefore, that capacitor will still partially charged even when the battery is discharged.
- I_p is taken as being a fixed fraction of the absolute value of the current flowing through terminals.

The diagram of the model we propose, is then as follows:

² As reported by Google Scholar on June 2017

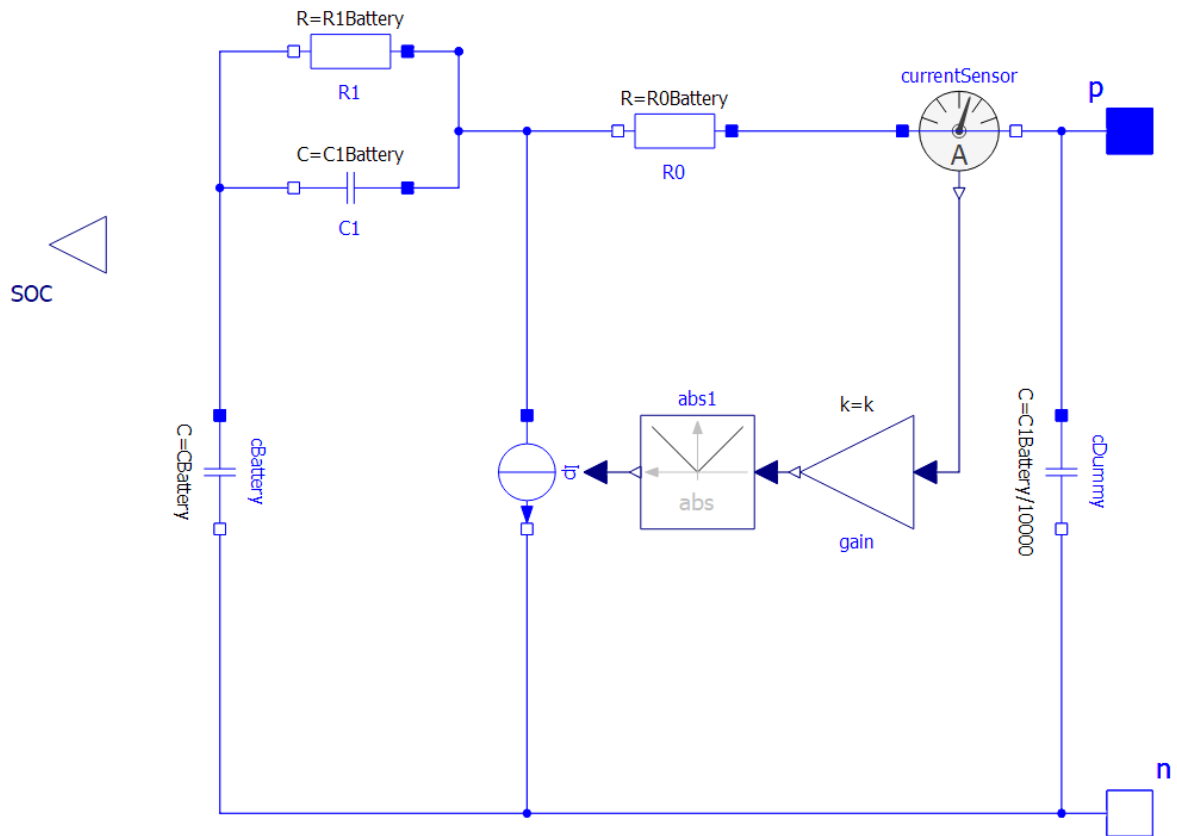



Figure. 24. The diagram of our battery model (EHPTlib.SupportModels.Miscellaneous.Batt1).

Both internal capacitance $cBattery$ and parasitic current i_p are automatically computed at initialization stage by the model itself, based on the user's data.

In this model, in fact, it is used the golden rule to ask the users, whenever possible, data in the form they are used to, and compute internally the data that are to be used during simulations. Below the tabs on the parameter window are shown.


OMEdit - Component Parameters - batt1 in wbEVPkg.EV.MBEV
?
×

Parameters

General
Cell data
Battery pack data
Modifiers

Component

Name: batt1

Class

Path: wbEVPkg.SupportModels.Batt1

Comment: Battery model based on one R-C block in its electric circuit

Initialization

Ubat.start ☒ V

SOC parameters

SOCMin	<input type="text" value="0"/>	Minimum state of charge
SOCMax	<input type="text" value="1"/>	Maximum state of charge
SOCInit	<input type="text" value="0.7"/>	Initial state of charge

Parameters related to losses

efficiency	<input type="text" value="0.85"/>	Overall charging/discharging energy efficiency
iCellEfficiency	<input type="text" value="0.5 * ICellMax"/>	A Charging/discharging current the efficiency refers to

OK

Cancel

OMEdit - Component Parameters - batt1 in wbEVPkg.EV.MBEV

Parameters

GeneralCell dataBattery pack dataModifiers

Parameters

QCellNom

100 * 3600

C

Nominal electric charge

ECellMin

3,3

V

Minimum open source voltage

ECellMax

4,15

V

Maximum open source voltage

ICellMax

10 * QCellNom / 3600.0

A

Maximum admissible current

Electric circuit parameters

R0Cell

0.05 * ECellMax / ICellMax

Ohm

Serial resistance "R0"

R1Cell

R0Cell

Ohm

Serial resistance "R1"

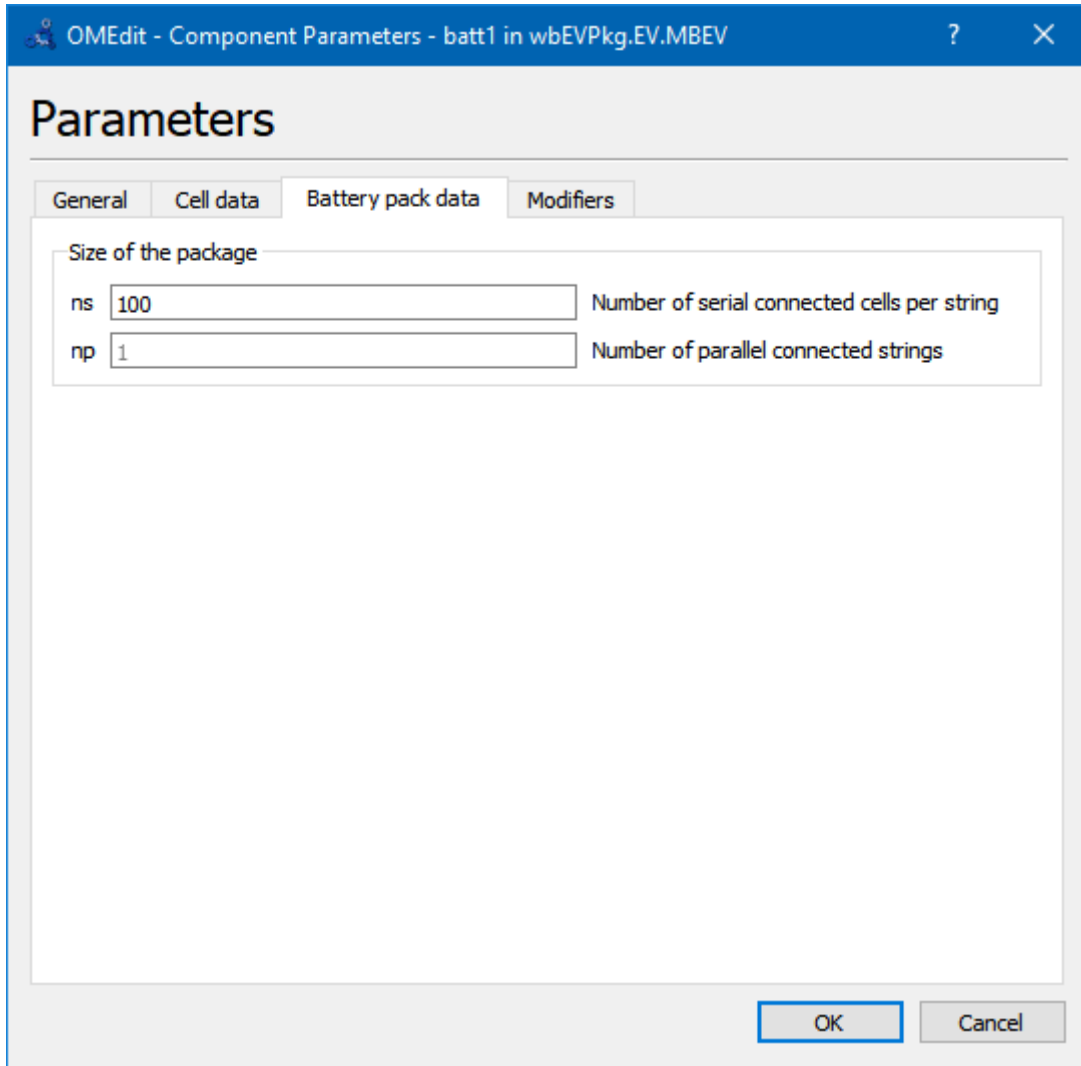
C1Cell

60 / R1Cell

F

Capacitance in parallel with R1

OKCancel



Battery pack data assume that the pack is made by parallel connection of np strings, each of which made up by a serial connection of ns cells.

The battery model gives as output the StateOfCharge SOC, defined by:

$$SOC = \frac{Q_s}{C} = 1 - \frac{Q_e}{C} \quad (4)$$

where Q_s is the “stored charge”, while Q_e the “extracted charge” (starting from full battery). The definition involving Q_e is more useful because the condition of battery full is more clearly defined than battery empty [1, 6]. However SOC is internally used in the model, the user may think of eq. (4) considering:

$$Q_s = \int_{t_{\text{empty}}}^t I_E dt \quad Q_e = \int_{t_{\text{full}}}^t -I_E dt$$

and t_{empty} and t_{full} are time at which the battery is fully empty or full, I_E is the battery current entering its positive terminal.

The reader is however invited to have a look at the code to gain experience in reading Modelica code.

In one of the two the provided battery models, in addition to what shown here, a very small capacitance, named “cDummy”, has been added across the battery terminals. Being very small, this does not influence the results, but facilitates convergence in special circumstances.

5.4.1 Proposed activity

The reader could enhance the battery model substituting the capacitor with a SOC –dependant EMF. He could first use a linear dependence, so that the behaviour of the fixed capacitor is reproduced. Then, a non-linear function, e.g. obtained using `Modelica.blocks.Tables.CombiTable1D`.

5.5 Map-based DC-interfaced electric drive implementation

We already discussed the concept of map-based models. We use two map-based models of an electric drive, one which sees <real inputs, another connectors. Let us consider the first one as shown in figure 25 (EHPTlib.MapBased.OneFlange).

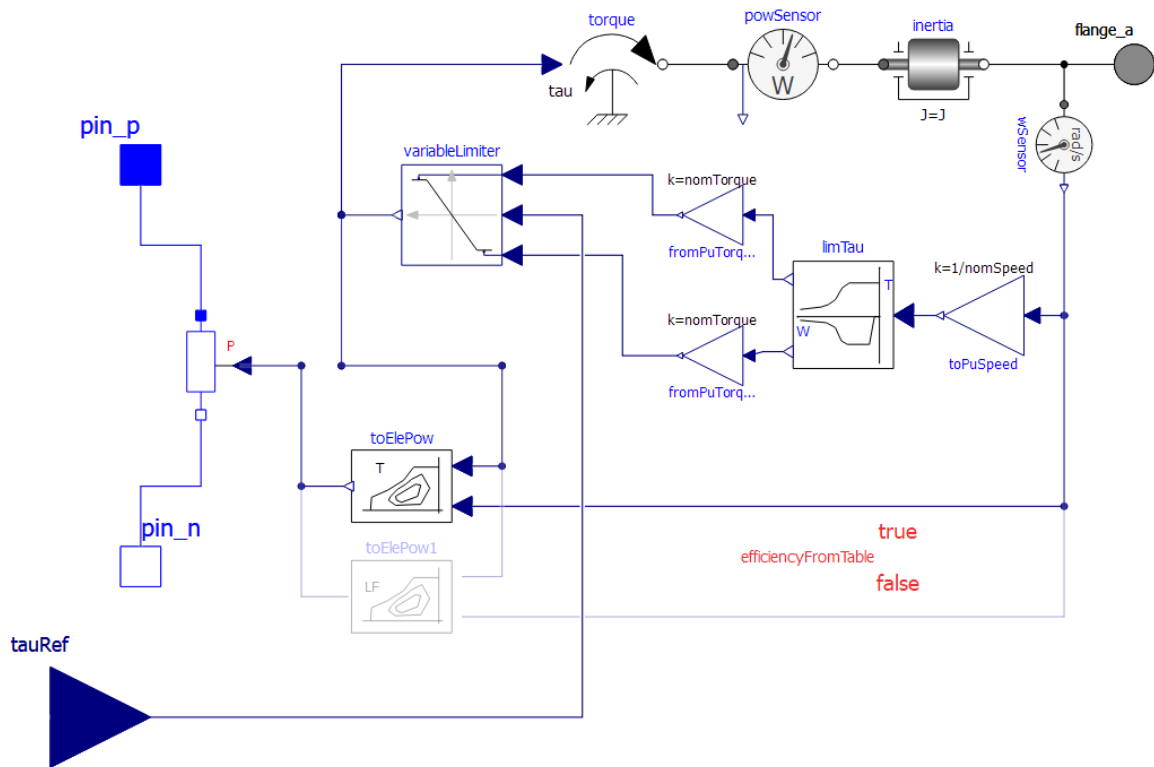


Figure 25. Diagram of our real-input DC-interfaced Electric Drive model (EHPTlib.MapBased.OneFlange).

Before analysing this model as a whole, the reader is prompted to have a look at the illustration of the utility blocks and models used here (`LimTorque`, `EfficiencyCT`, and `EfficiencyLF`) and model `ConstPg`, as found in sect 9.

Block `limTau` of class `LimTorque` avoids the requested torque `tau` to overcome its maximum allowable; block `toElePow` of class `EfficiencyCT` or `EfficiencyLF` depending on flag `efficiencyFromTable` computes the electric power from the mechanical inputs, considering the wished efficiency entered a `CombiTable` or a `LossFormula` respectively.

Finally the variable resistor model `pDC` of class `ConstPg` at the left side of the picture, is an explicitly created model to interface map-based mechanical models with the corresponding electric circuit.

The behaviour of the model is rather straightforward: it tries to apply the input torque `tauRef` to the rotor inertia; before doing this, it evaluates whether the requested torque overcomes the inherent limits of the drive, which are also a function of the shaft angular speed. These limits are applied through block `limTau`. The generated mechanical torque is then subjected to the block

toElePow, which applies to it the component's efficiency and so determines the electric power to be requested from the DC input, through pDC.

6 Simplified electric drives

6.1 Constant voltage/frequency asynchronous drive

Constant voltage/frequency control is a common, easy-to implement control of asynchronous drives.

The principle of operation of an asynchronous drive is described in figure 26, taken from [4].

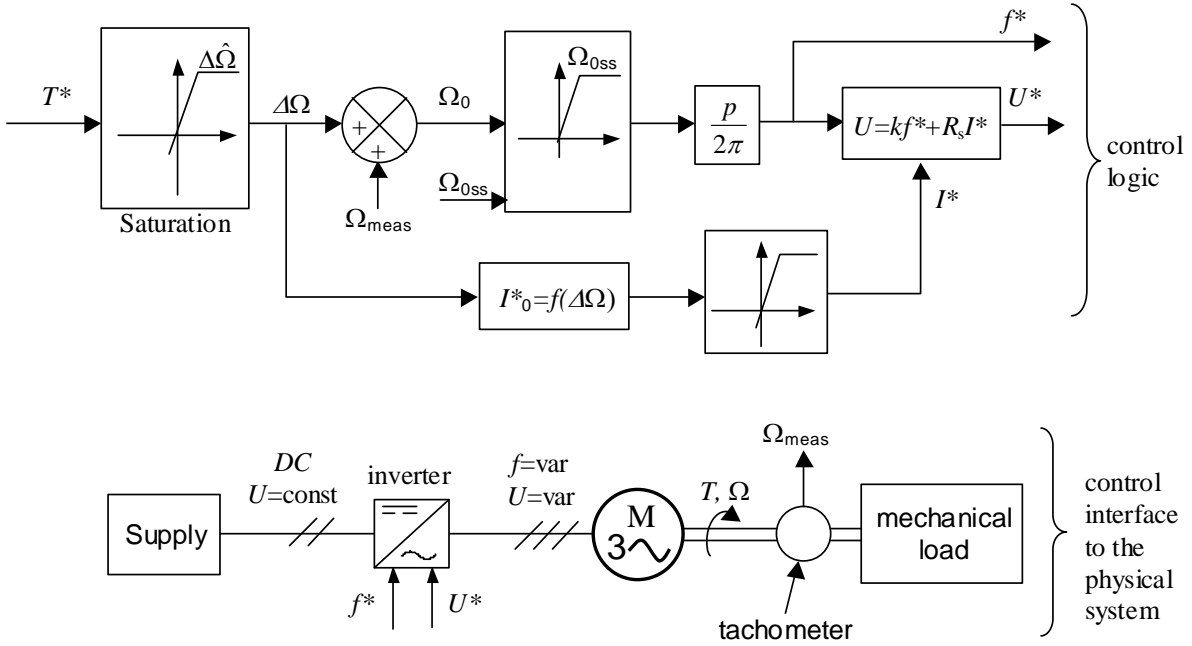


Figure 26. A general representation of a hybrid power train.

This is implemented in EHPTlib code following this diagram, with the following model:

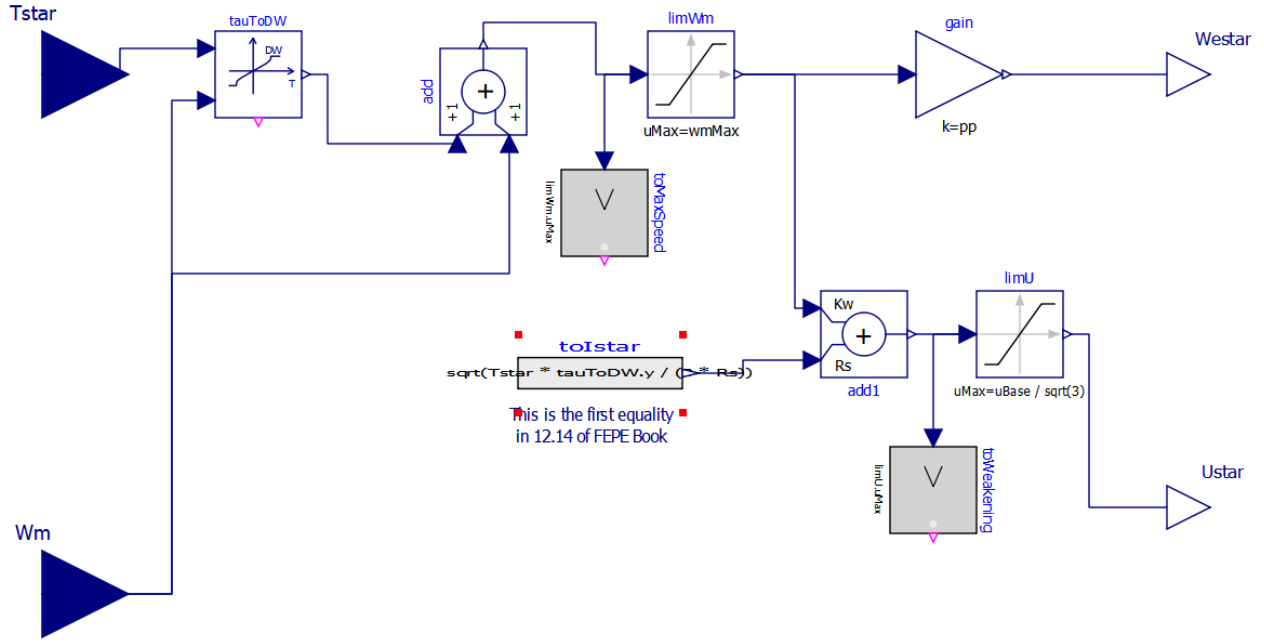


Figure 27. Diagram of control logic for ASMA constant voltage/frequency control (model EHPTlib.ElectricDrives.ASMARelated.ControlLogic)

This diagram follows quite closely the one illustrated in figure 26. In particular:

- block τ_{toDW} receives the torque requested and converts it into slip speed $\Omega_0 - \Omega$. Note that if the torque request is beyond the maximum allowed by the machine, the output slip speed is equal to $\Omega_0 - \hat{\Omega}$, where $\hat{\Omega}$ is the speed value corresponding to the maximum torque.
- we have two saturations: $limW_m$ that blocks any further frequency rise, when we have reached the maximum operational speed: this is the control logic parameter named w_{max} , and $limU$, which limits the maximum voltage to the U_{base} value set in the control logic parameters. Note that u_{base} is reached when w_{base} frequency is reached.
- As usual, letter “w” or “W” indicate angular speed or angular frequency. To avoid confusion, here “we” indicates angular frequency (“e” stands for electric”) while “wm” indicates angular speed (“e” stands for mechanical).
- the current reference inputted to $add1$ block is a non-linear function of $\Delta\Omega$ implemented using the very formula of eq. 12.14 of [4].

6.1.1 Starting Asynchronous machines

In EHPTlib as a simple example to evaluate how control Logic works, the starting of the MSL asma machine, with default parameters is obtained with a full start from constant-voltage.-constant-frequency mains source, or using an electric drive using the constant voltage/frequency logic. The model diagram is shown in figure 28.

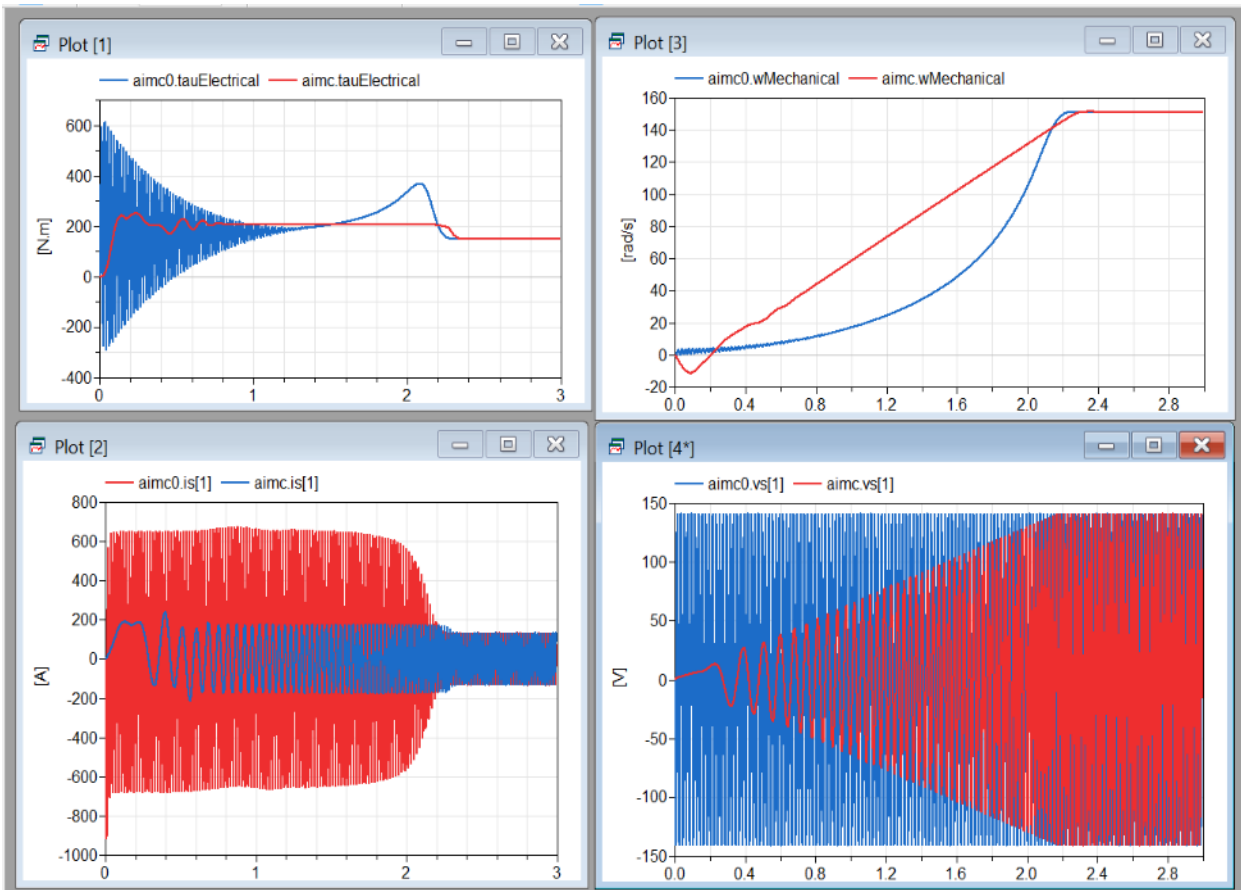


Fig. 29. Some curves obtained simulating the diagram shown in figure 28.

The top-left picture shows torques. The blue curve represents the torque when starting from mains. We see huge oscillations due to the fact that initially the three currents are not a balance set of three-phase currents; the drive torque is quite near to be constant. More constant could be obtained with field oriented control, which however is out of the scope of this tutorial and library. Top-right we see the angular speed: with the used parameters both simulations reach final regime in the same time.

Bottom-left we see the current. It is seen that the starting from mains requires much higher currents than the other case, which also means that the machine cannot stay at low speed for long times: it would heat and eventually burn; moreover it operates at very low efficiency. The drive, instead requires much lower currents and is such that the drive could stay at any speed between zero and the maximum one, without thermal issues. Finally, the bottom-right plots compare the phase voltages in the two cases.

6.1.2 Torque following

Obviously, electric drives are not used for just starting machines, but also to control their operation. In EHPTexamples an example is proposed for using the drive to follow a torque request.

In this model (EHPTexamples.ElectricDrives.tqFollowing) the drive is requested to follow a torque signal.

The proposed model is as in figure :

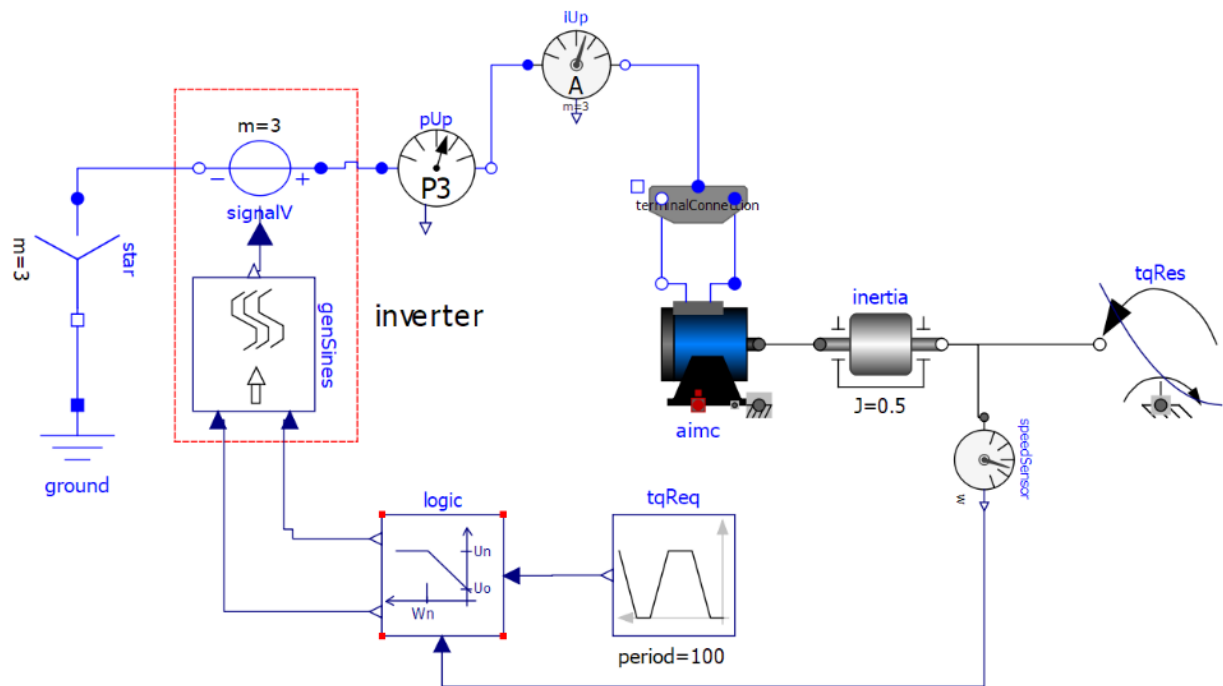
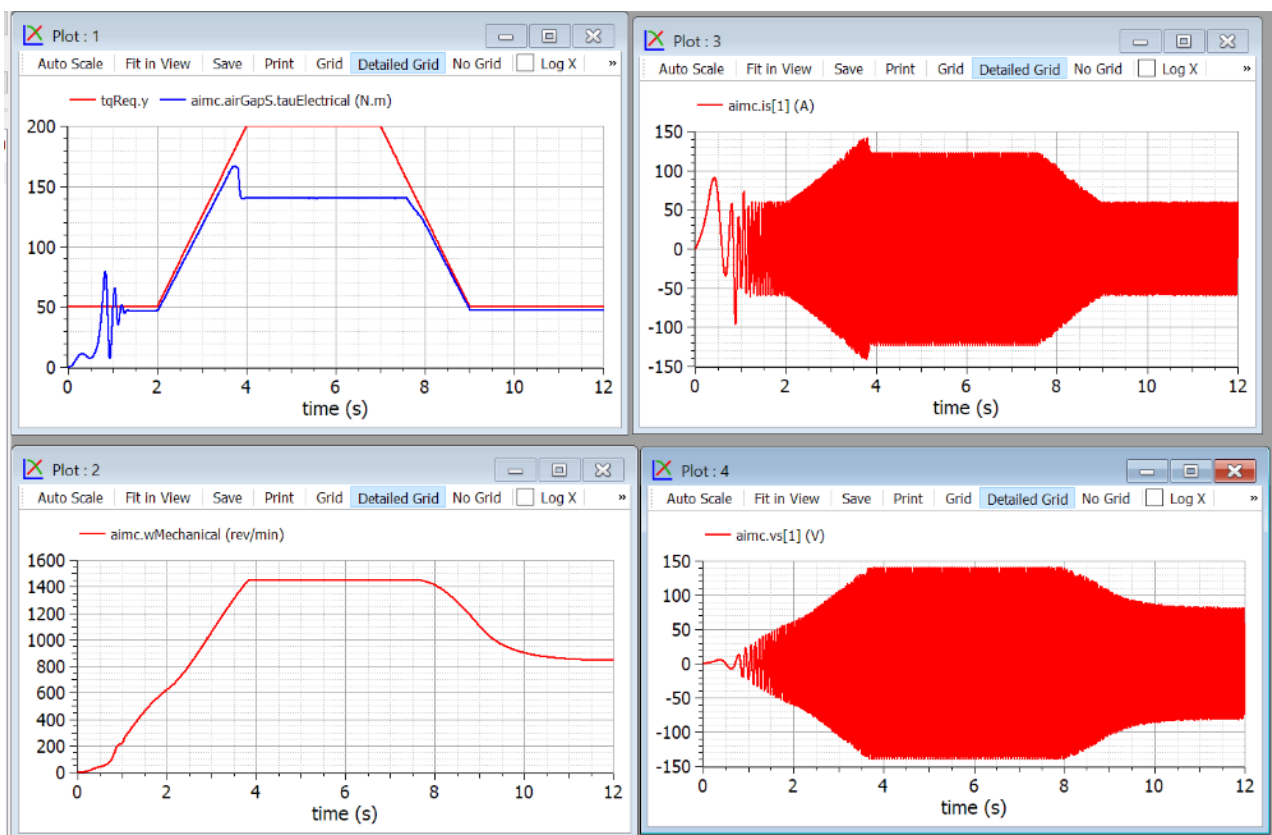


Figure 30. Diagram used to show torque-following logic.
(EHPTlib.electricDrives.SmaTqFollowing))

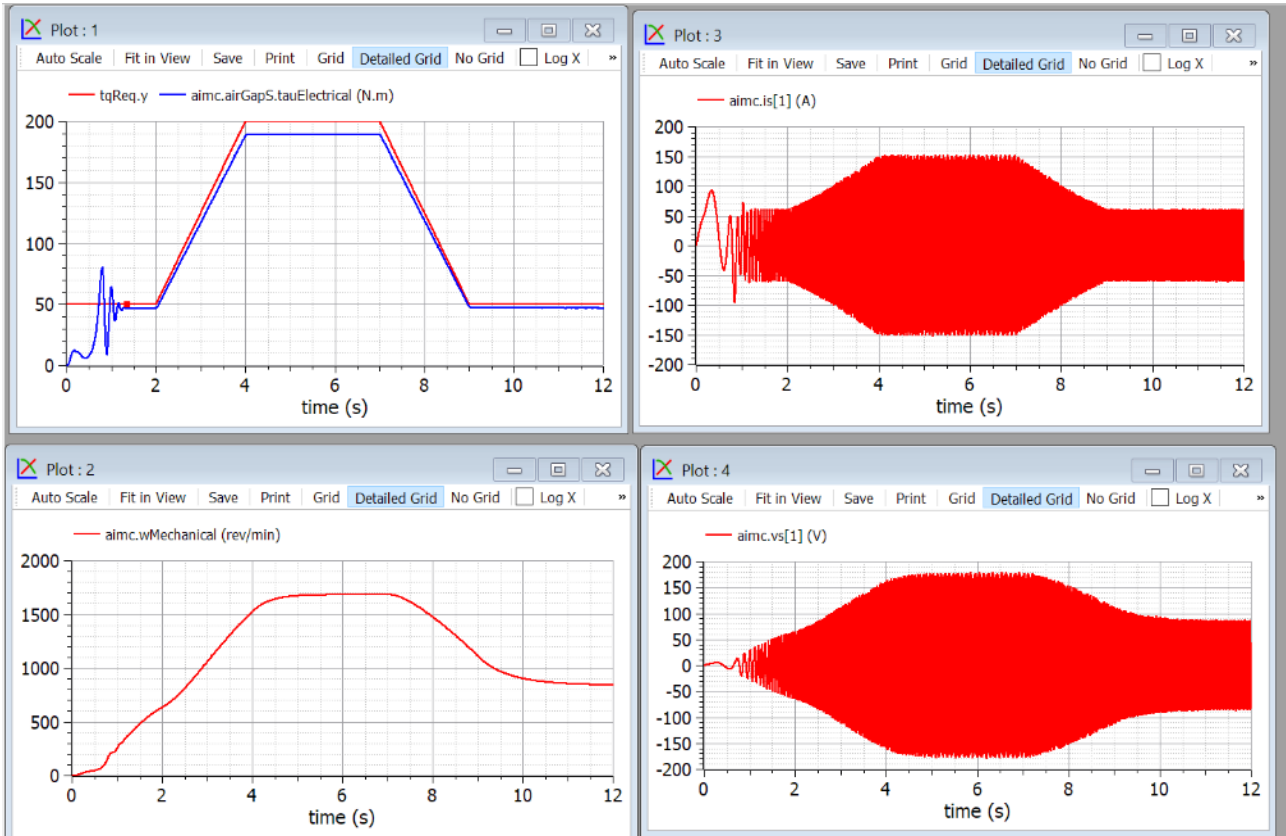
Some significant plots follow:



Note that the machine cannot follow torque in the central region, because max voltage and max speed are reached, as can be seen looking at variables `logic.toWeakening.y` and `logic.toMaxSpeed.y`. voltage limitation is reached, and no torque weakening logic is included here.

The torque is followed much better if we change in the control logic `uBase` and `weBase` and `wmMax` bringing them to 200 V rms (phase) 500 rad/s and 314 rad/s respectively.

The corresponding plots are as follows:

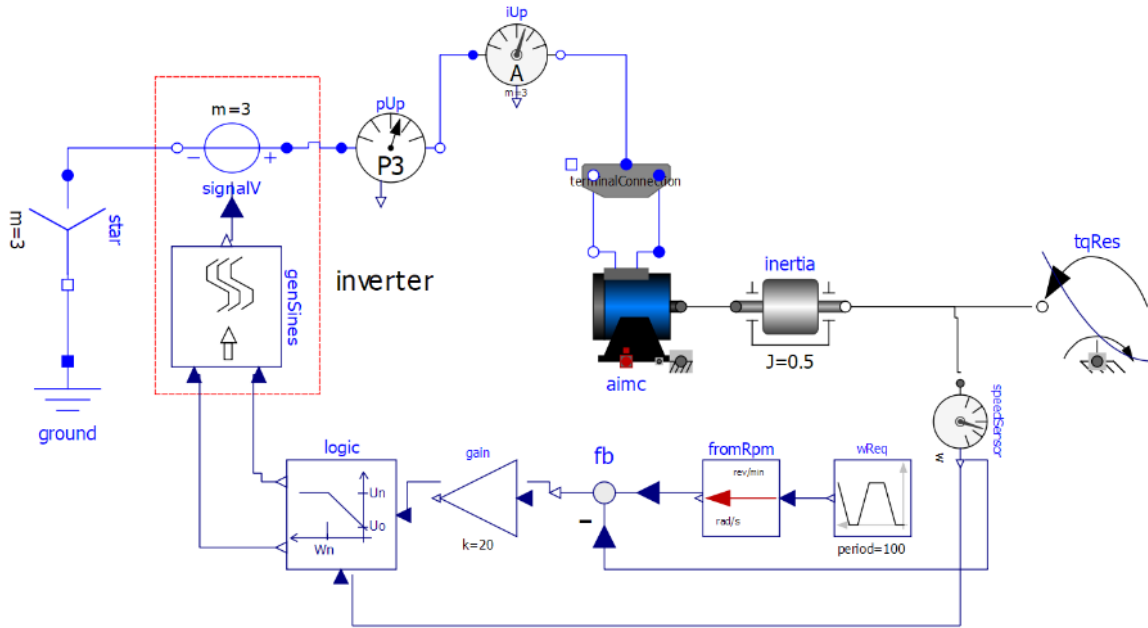


variables `logic.toWeakening.y` and `logic.toMaxSpeed.y`.voltage remain false.

Note that torque is not exactly followed because we used an open-loop control. Closed loop control of torque is rare. Very often when need to control speed, so loop is closed on speed. as per the following example. In vehicles also we have closed loop speed control, where we consider the controller to be the driver.

6.1.3 Speed following

Now we simulate a closed-loop speed control. The model is as follows:



Here we see that mechanical torque follows the requested signal; since we used a simple proportional controller, some error is still present. This can be reduced significantly by enlarging (but not too much to avoid instability or lack of convergence) the control gain or adding an integral part to control (in this case the steady-state error reaches zero).

6.2 Synchronous machine drive

Today many electric drives are implemented using permanent-magnet synchronous machines.

To have a mildly detailed idea of how they work, here control blocks able control torque of both isotropic and anisotropic machines are proposed. They can operate to keep under control the machine's terminal voltage (producing negative direct-axis currents I_d), and terminal current, reducing the delivered torque if delivering the requested one would imply overcoming the given machine current limits.

The following blocks are provided:

- ToPark and FromPark: they convert phase quantities into Park's and vice-versa
- MTPAi, determines the currents I_d and I_q to implement the Maximum Torque Per Ampere control, adequate for isotropic machines. In this case below base speed I_d is always zero
- MTPAa, determines the currents I_d and I_q to implement the Maximum Torque Per Ampere control, adequate for anisotropic machines. In this case below base speed I_d for anisotropic machines is different from zero. MTPAi behaviour is obviously attainable using MTPAa, imposing equal values for I_d and I_q . Two different blocks are provided, since for simplified analysis and teaching MTPAi is easier to understand
- MTPAal, a modified version of MTPAa, that is able also to limit stator current so that to avoid the machine to be overloaded.

6.2.1 Simulation of *EHPTlib.ElectricDrives.TestingModels.SmaDriveFW*

In this simulation the permanent-magnet synchronous machine of the MSL (with the default parameters) is controlled using the MTPA controller.

The model diagram is shown in figure:

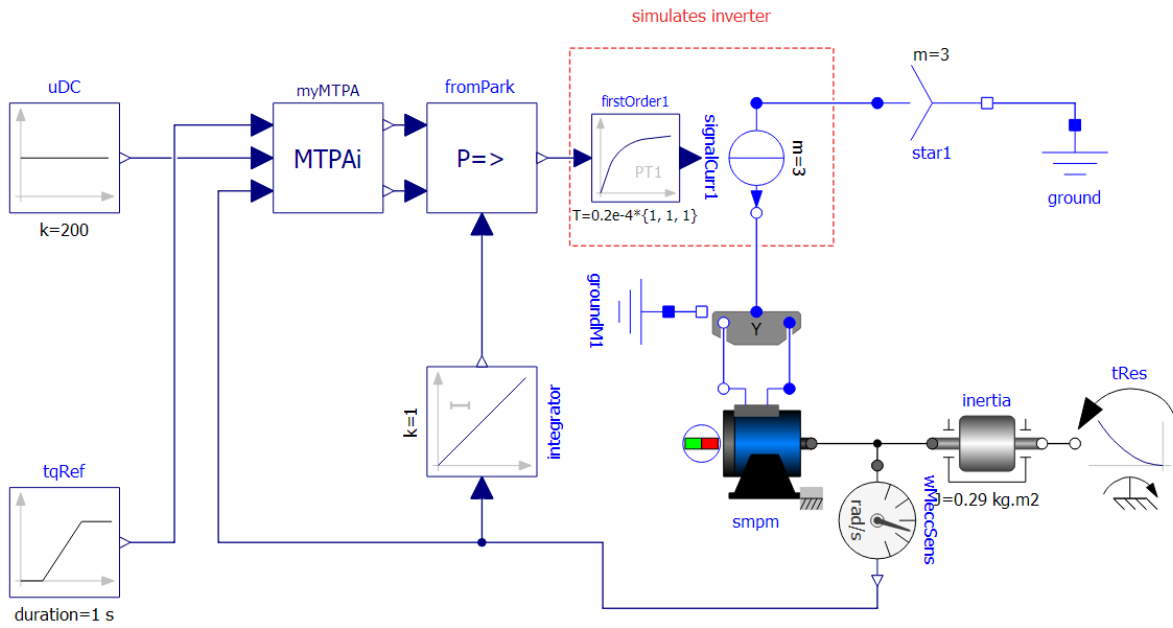


Fig. 31: Starting a Permanent-magnet synchronous machine using park components of stator current (ElectricDrives.TestingModels.SmaDriveFW).

As in real life, the controller must be calibrated on the controlled machine. This is also done in this model, as shown in the following dialog window:

OMEdit - Component Parameters - myMTPA in wbEHPTlib.ElectricDrives.TestingModel... ? X

Parameters

General Modifiers

Component

Name: myMTPA

Class

Path: wbEHPTlib.ElectricDrives.SMArelated.MTPAi
Comment: MTPA logic for an isotropic machine

Parameters

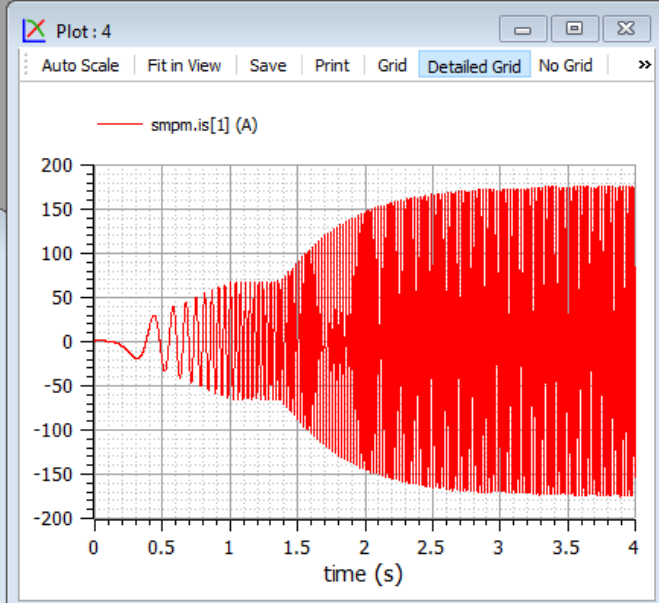
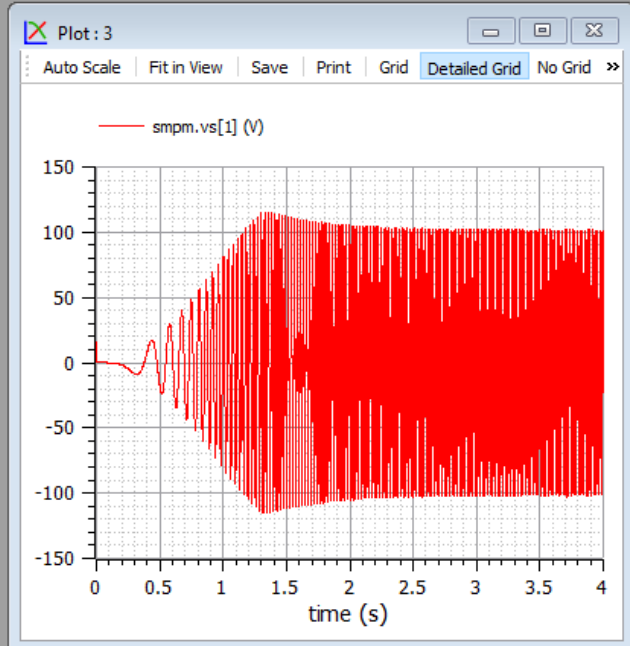
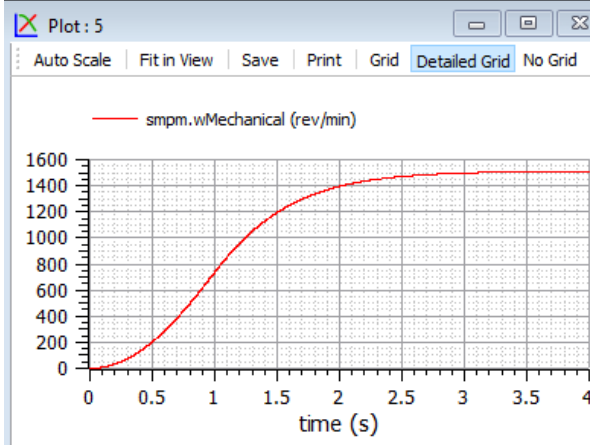
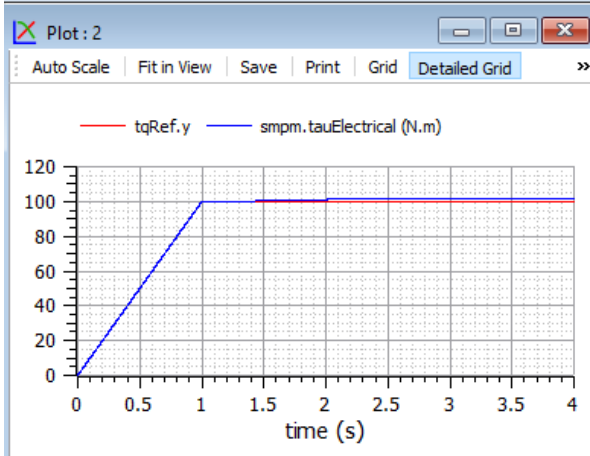
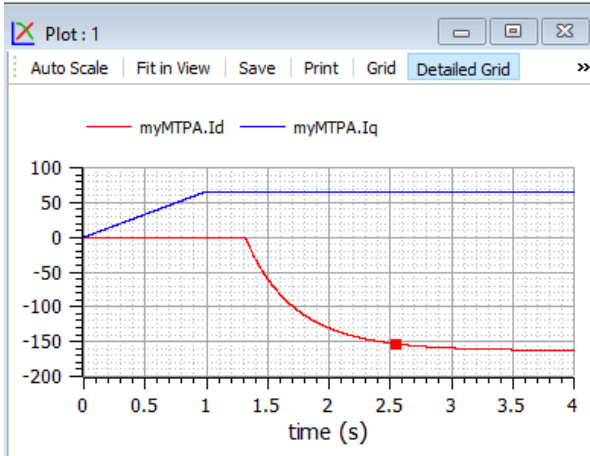
Ipm	smpm.permanentMagnet.Ie	A	
pp	smpm.p		Pole pairs
Rs	smpm.Rs	Ohm	Stator resistance
Ld	smpm.Lmd	H	Direct-axis inductance
Umax	100	V	Max rms voltage per phase to the motor

Initialization

gammaStar.start	<input type="checkbox"/>	0	deg	
IparkFF.start	<input type="checkbox"/>	0	A	Ipark amplitude FullFlux (i.e. before flux weakening evaluation)
Ipark.start	<input type="checkbox"/>	70	A	Ipark amplitude ($=\sqrt{Id^2+Iq^2}$)

OK Cancel

Some interesting results are shown in the following figure:



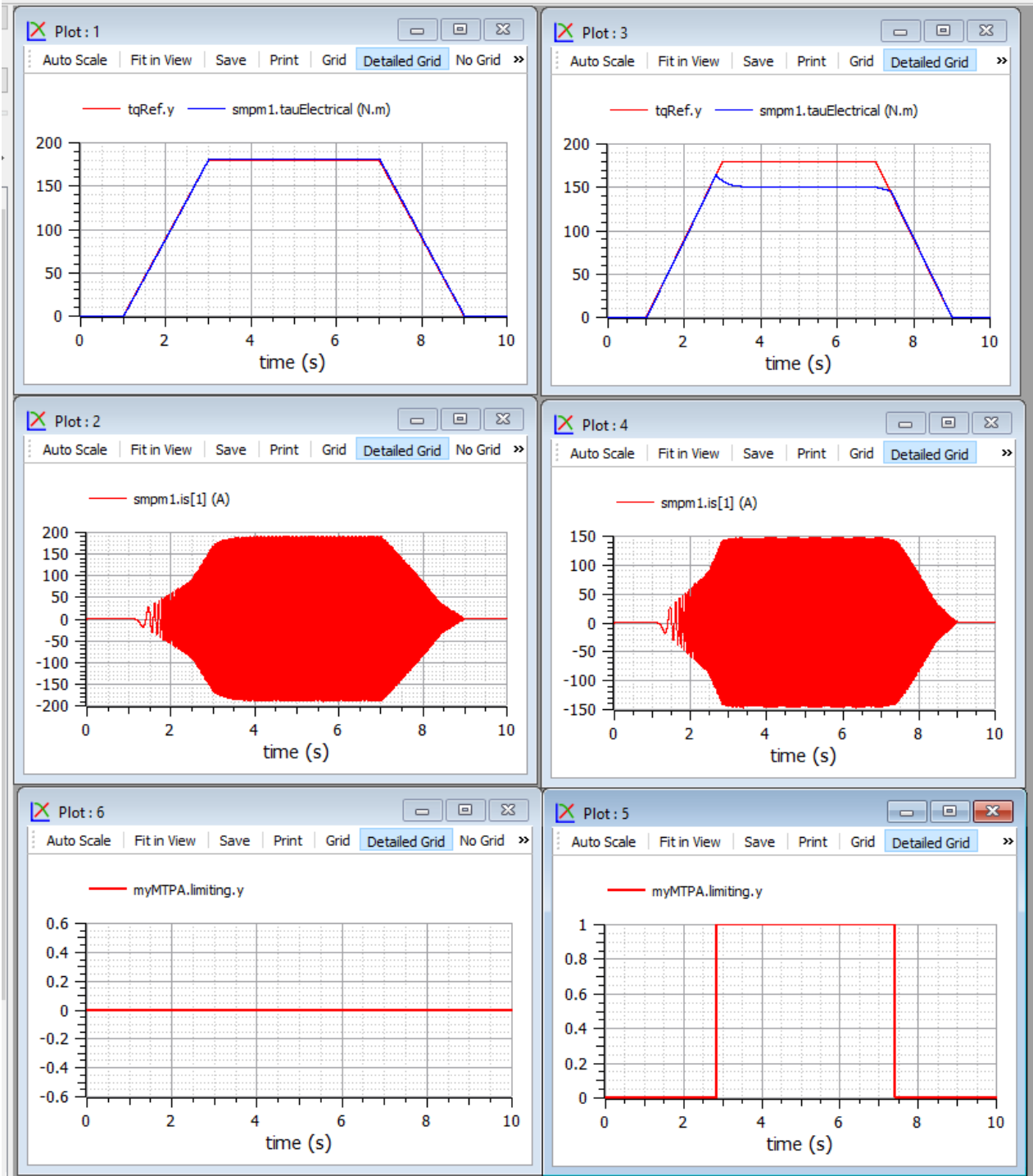
The U_{dc} input to myMTPA is the actual DC voltage. Internally the block limits the maximum line-to-line peak voltage to the UDC; which is what required by state-of-the art Space-Vector PWM control. From the output it can be checked that the machine voltage is limited to 115.5 V (peak, which corresponds to 200V line-to-line (peak)).

6.2.2 Simulation *EHPTlib.ElectricDrives.TestingModels.SmaDriveLim*

In this simulation the drive is subject to a torque request that would imply the machine current to overcome the maximum allowed. The controller MTPAal is used that is able to control also that the current does not overcome the maximum allowed. In this simulation this controller controls the MSL PMSM machine, that is isotropic. When satisfying torque request and voltage control cannot be attained while keeping current below the maximum allowable value, i_q is reduced to reduce torque.

Intervention of torque limitation to limit current is indicated by the status of the output of block “limiting” inside MTPAaI.

In the following figure the plots at the left-hand side are drawn setting the limiting current to 200 A rms. Here current limitation does not intervene. In the right plots, instead the current limitation is set to 100A rms, and intervenes effectively: it keeps current under control, at the expenses of some torque reduction.



6.2.3 Simulation of *EHPTlib.ElectricDrives.TestingModels.SmaAllSpeeds*

Permanent Magnet Synchronous Drives have a first zone, up to the so-called *base speed*, in which voltage is progressively raised as speed increases. When the maximum allowable voltage is reached,

it is clamped to that maximum, and the constant-voltage constraint determines the control and the split of the stator current in its two components on direct and quadrature axes.

If, during this transient, the stator current amplitude is kept constant, the active power firstly grows linearly, then, after base speed, initially still increases, but with a lower slope, reaches a maximum, then decreases. Before power reaches its maximum the machine absorbs reactive power (it is in the inductive zone), beyond that maximum it delivers reactive power (capacitor zone).

All this can be seen in the SmaAllSpeeds simulation whose diagram is shown in figure 32.

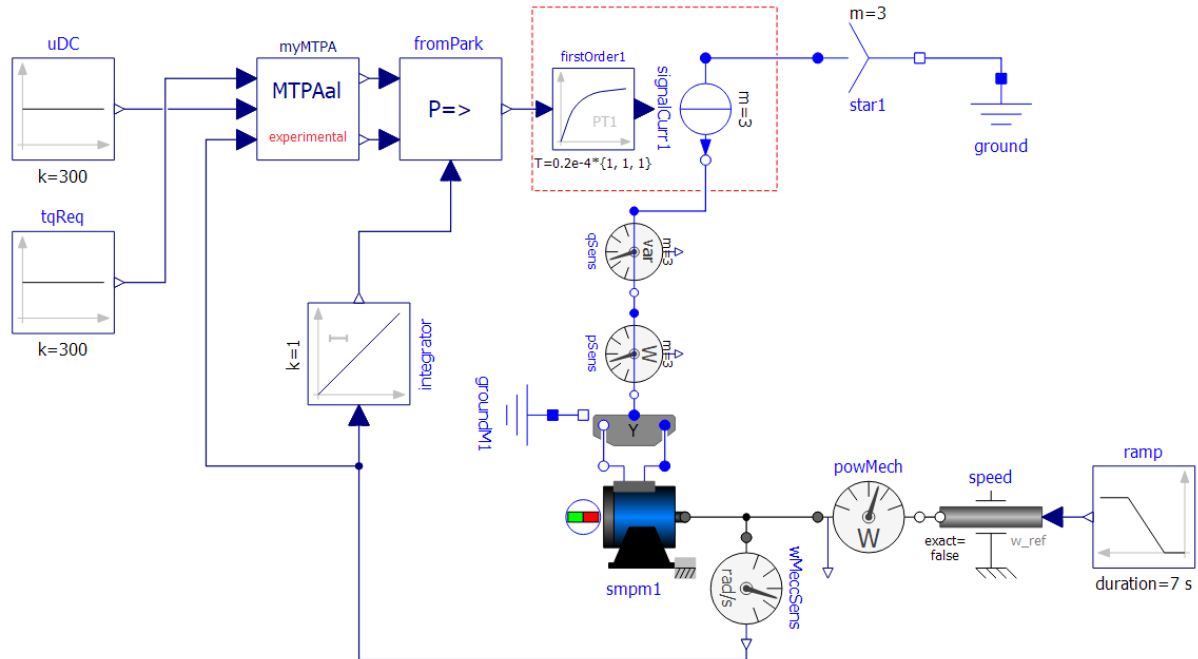
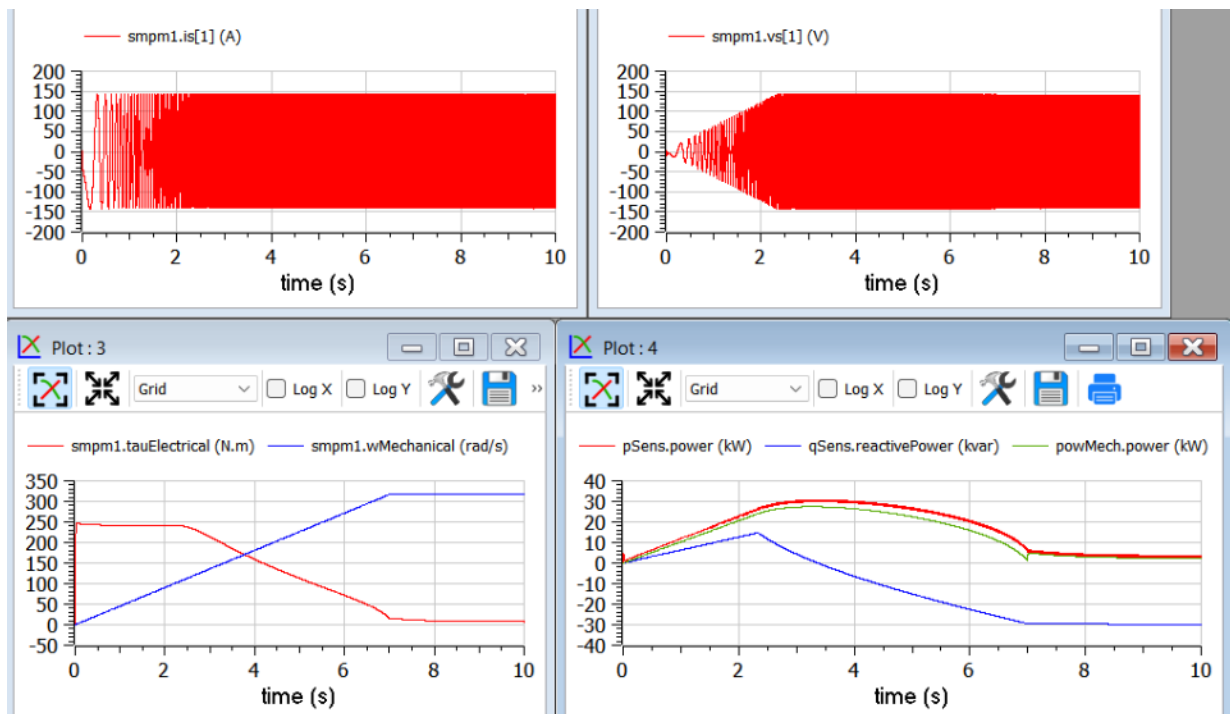


Fig. 32: diagram of the model ElectricDrives.TestingModels.SmaAllSpeeds.

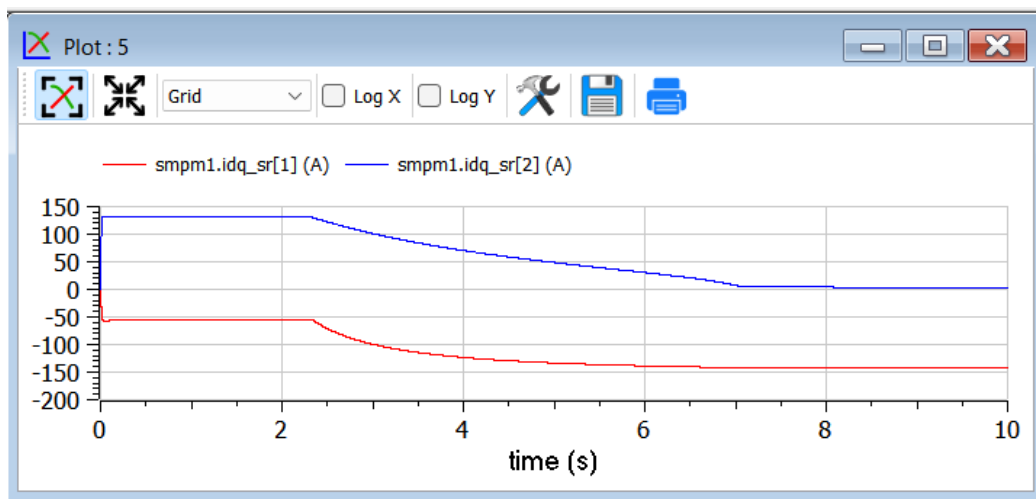
Some results of its simulation which are shown in the following figure.



Some comment on the plots:

- The top-left plot shows that the MTPAal control is effective in keeping the current constant and equal to its nominal value (100 A rms);
- the top-right plot shows the voltage initially linearly increasing (and with frequency proportional to speed), up to when the maximum allowed (here 100 V rms) is reached; then the voltage is effectively clamped to the maximum allowed;
- the bottom-left diagram shows that the electrical torque actually developed is always lower than the requested value i.e. 300 Nm. In fact, the control gives priority to the need to keep the current within limits, and reduces torque whenever this is necessary for this purpose;
- the bottom right diagram shows the active power first growing linearly (up to the base speed, i.e. the point in which the voltage stops increasing linearly, then grows slower, reaches a maximum, then decreases. The reactive power initially grows linearly in the inductor region (power factor constant); after overcoming the base speed, it begins to reduce; reaches zero exactly when active power is at its maximum, then becomes negative (capacitive operation of the PMSM). The figure also shows the mechanical power (green). The difference between red and green curve during acceleration is mainly due to the inertia torque times rotational speed; in the final seconds, when speed is constant, the two curves are much nearer, the difference being due to power losses in the machine resistances.

The following plot finally shows how the I_d and I_q components of stator currents are shaped, to fulfil all the control purposes:



Up to base speed I_d (red) is slightly negative. If the machine were isotropic, in this region I_d would have been zero. Then it becomes more negative to keep voltage under control (to weaken flux).

The quadrature current I_q stays all the time at the values allowed by the limitation on the stator current, given the needs of controlling voltage which determine I_d and have priority.

7 Map-based HEV models

7.1 HEV's resume

The reader is supposed to have some basic knowledge of what hybrid vehicles are and how they are structured. Therefore here we give just a brief resume.

According to ISO/TR 8713, a hybrid vehicle is a vehicle which has both a Rechargeable Energy Storage System (RESS) and a fueled power source for propulsion.

This corresponds to the following graphical representation:

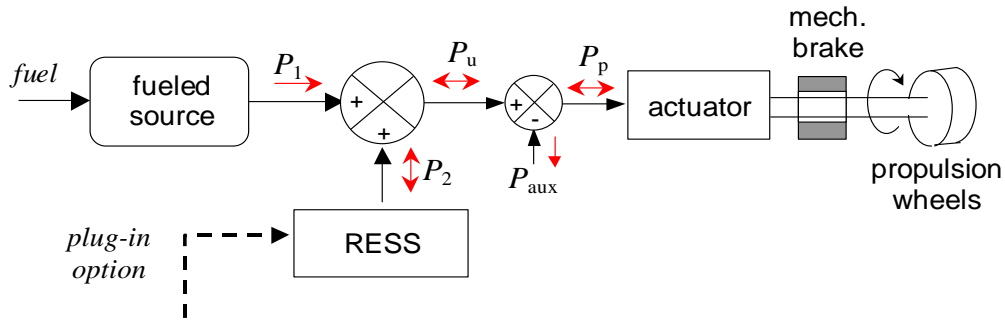


Figure 33. A general representation of a hybrid power train.

In which the red arrows indicate possible power directions. In the figure we use the following symbols:

P_1 : fuelled source power (positive)

P_2 : RESS power (positive when contributing to propulsion, negative when absorbing power into RESS)

P_u : useful power

P_{aux} : power “delivered” absorbed by auxiliaries. In reality auxiliaries absorb power, so P_{aux} is always a negative number as indicated by the red arrow. It is the sum of any load other than propulsion, e.g. lamps, GPS, radio, etc.

P_p : propulsion power: positive when in traction, negative when regen-braking.

The plug-in option is often provided to recharge RESS from the mains. This will not be considered in the remainder of this chapter.

We might say that the scheme in figure 33 is *cyber-physical*: the part left of the actuator is represented in terms of the block diagrams usually used to analyse control systems, and good at describing general fluxes, the part right of the actuator is more physical (it represents physical objects).

The scheme in figure 33 is rather general and accommodates several architectures. For instance, the fueled source could be an internal combustion engine (the most common case), a gas turbine, a fuel-cell generation system, etc. In this chapter we will only considered as fueled sources internal combustion engines.

Even while sticking to internal combustion engines, there are many ways in which this abstract representation can be implemented. Often they are classified as follows:

- Parallel Hybrid Vehicles (PHEV). Here there is a mechanical path from the ICE to wheels. Power contribution from RESS is added or subtracted to this main mechanical path
- Series Hybrid Vehicles (SHEV) Here the mechanical power produced by the ICE is first converted into electricity; then additional power from RESS is algebraically summed to this power, to generate the full propulsion power.
- Complex Hybrid Vehicles (CHEV): solutions which cannot be classified in the previous two architectures, e.g. in cases in which there is a parallel and a series power path.

In this chapter we present models of all these architectures. In the third group we choose a Power-Split-Device based architecture, which is the scheme used by the best-selling hybrid vehicle ever³.

Note that since every HEV has two possible sources of electricity, there is an additional degree of freedom in comparison to conventional vehicles: at each time we can decide which part of the needed propulsion power is to be delivered by the fueled source, and which from the RESS.

³ The Toyota Prius that, according to [11], has sold between 1997-2013, over 3 million cars worldwide.

Deciding this share is a complex operation since a lot of issues must be solved, related to the fact that the RESS can get discharged (and so cannot deliver power anymore), fully charged (cannot absorb power anymore), the load to be coped with, the useful power, is a function of time, the choices we make at a given time t influence what will happen in the future, and so on. This complex task is performed by a control system usually called EMS (Energy Management System), which takes as input the drivers commands and operates the power train in such a way to satisfy the propulsion and auxiliary power requests, while trying to maximise an performance index, typically the average efficiency (thus minimising the fuel consumption).

7.2 SHEV basic model

The SHEV model proposed here has the following diagram:

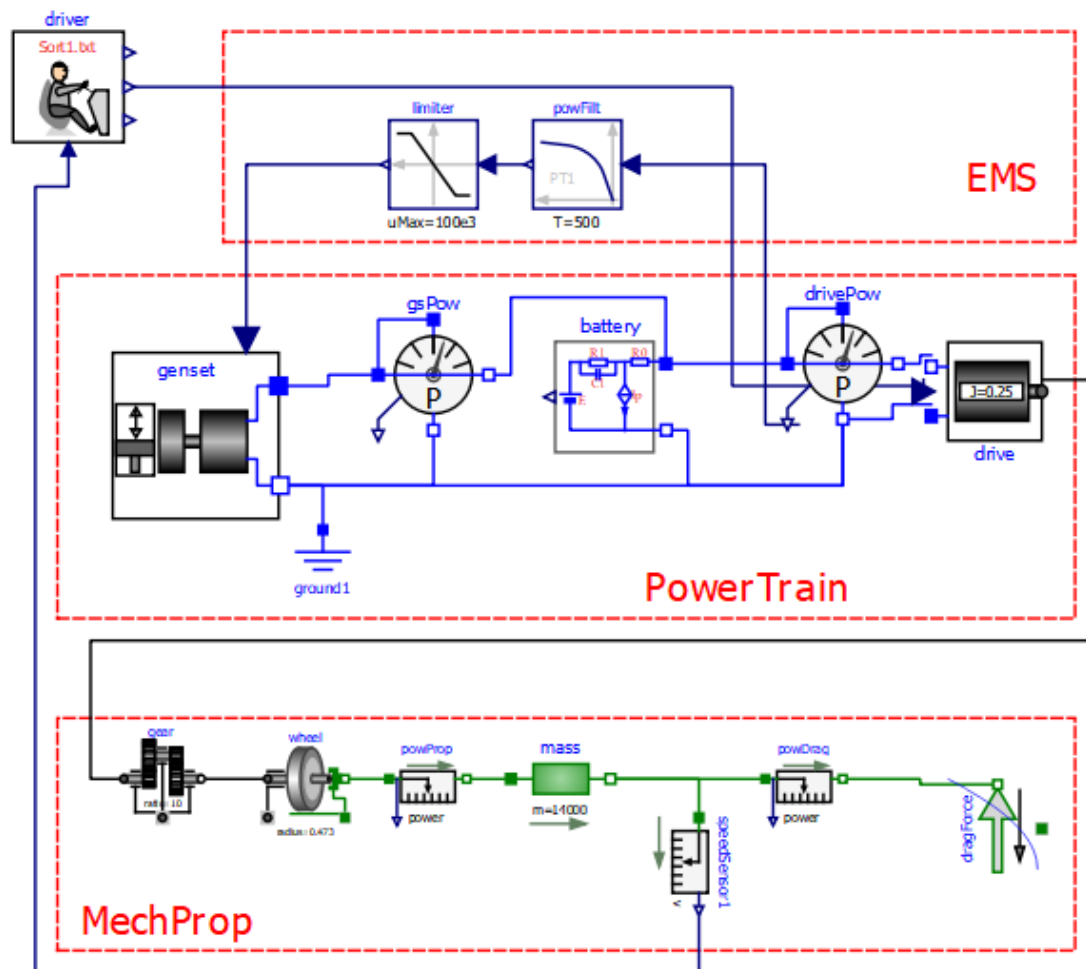


Figure 34 Diagram of the EHPTexamples.SHEV.SHEVpowerFilt model.

This model requires, to run, some data to be specified on a txt file. To reproduce the results shown here the reader can use the provided "SHEVmaps.txt". How to link a txt file with a model is explained in section 2.3.

Note on genset parameters.

The internal ICE data are supplied through maps to be provided through a txt file. The values explicitly set through the genset Parameters dialog box refer to the internal generator (except **wlceStart**). Any change on these should be made considering joint changes in the ICE maps.

The lower part is the physical mechanical propulsion part used in the previous EV models.

In the central part of the picture the Power Train is detailed.

It contains a `genset`, which generates electricity, the battery `batt`, constituting the RESS in figure 33, a map-based electric drive `drive` of the same type used in section 4.

The upper port contains the EMS (whose need was discussed at the end of section 7.1), here having a very simple structure: it requires the `genset` to deliver the average propulsion power: in this way battery operates as a “power filter”: its delivers fast components of the propulsion power, leaving the average to the `genset` (i.e. the fueled source). Naturally, EMS must take into account the torque requests from the driver, which in turn tries to follow a given driving cycle, as seen when discussing pure (non-hybrid) electric vehicles.

Series Hybrids are most suited for urban trips, in which they allow important downsizing of the Internal combustion engine. Therefore here we test that hybrid using, as mass and resistance to movement data, numbers adequate for a bus: mass is 14 tonnes, cross sectional area 6 m², Cx 0.65.

The following figure 35 shows some results, obtained when simulating a few repetitions of a Sort1 cycle that, as already noted, was conceived for buses. These results can be replicated simply by running `EHPTexamples.SHEV.SHEVpowerFilter` using the default data.

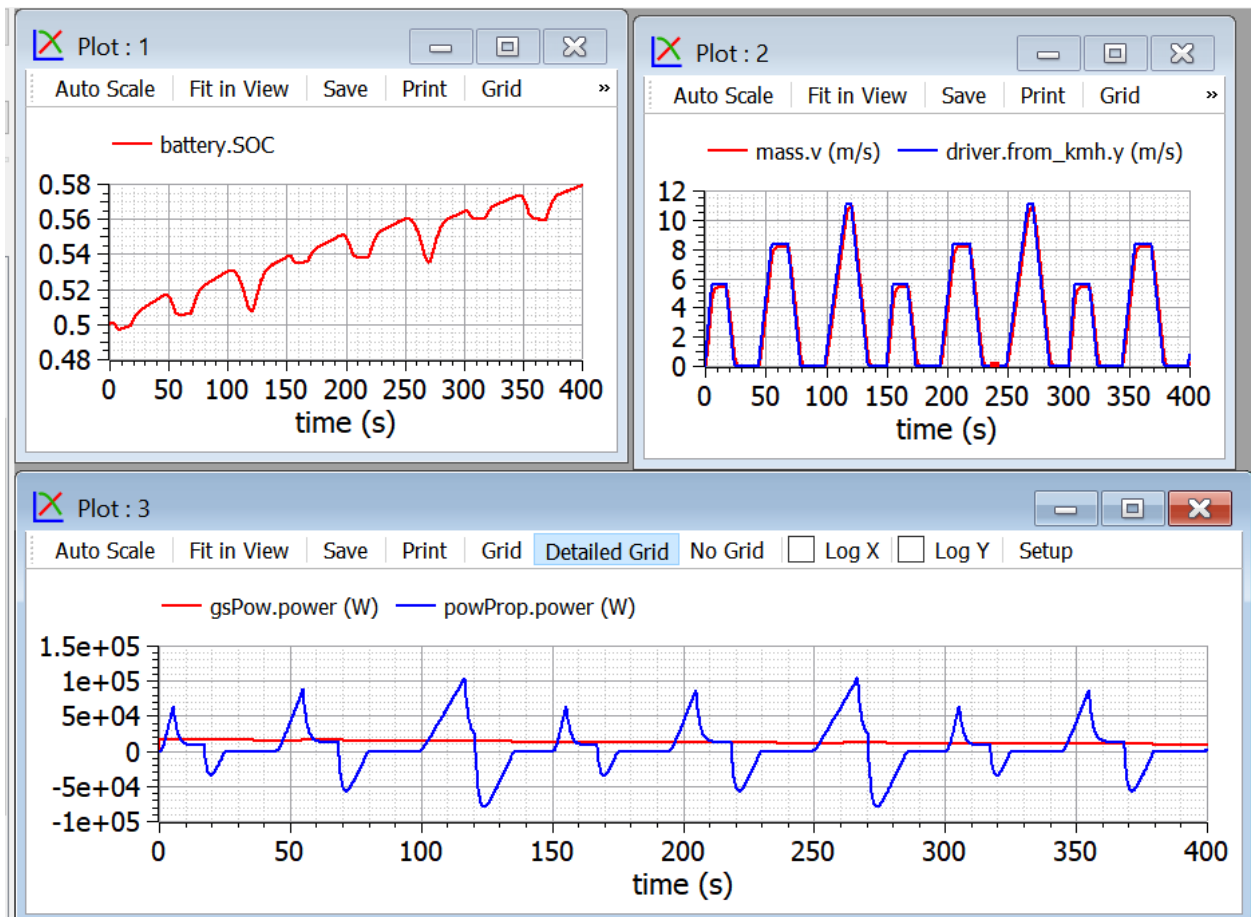


Figure 35. Plots of some Sort1 repetitions using the system in figure 34.

Note that the peak power required by this cycle is around 120 kW; using this hybridisation, the power required from the ICE is around only 7 kW.

The reader can reproduce these results selecting in the `driver` the Sort1 cycle as parameter.

However, buses, from time to time, are requested to cover also extra-urban routes, e.g. when going into the deposit at the end of the day or when they are to be transferred from a town to another. Therefore it might be of use to evaluate also the performance of our hybrid bus also under the NEDC cycle.

This is easily done just changing the cycle (in the driver's parameter) and the simulation duration. Some results, which can obviously be reproduced by the reader (he has just to change the cycle from sort1.txt into NEDC.txt and the simulation duration), are as follows:

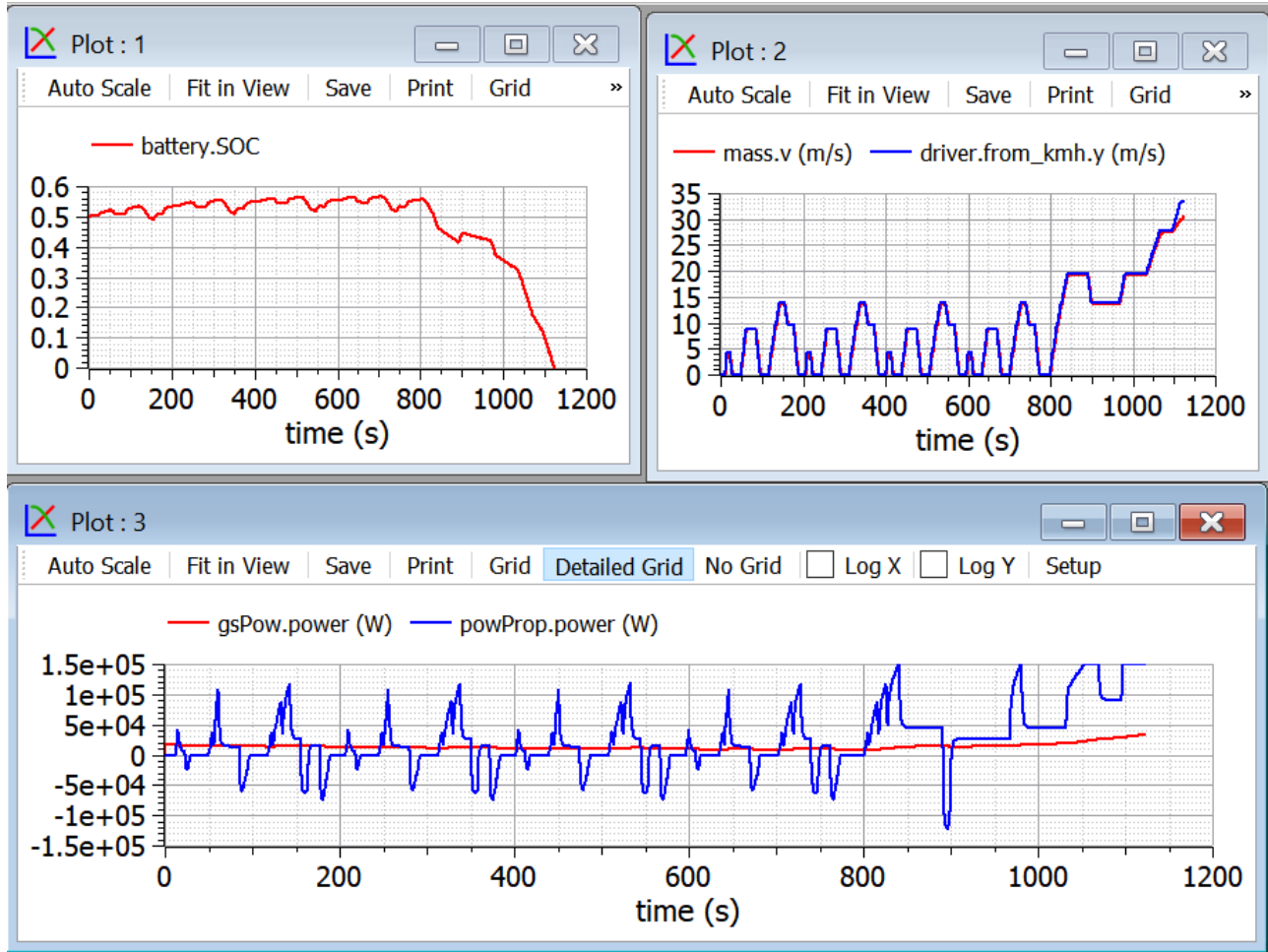
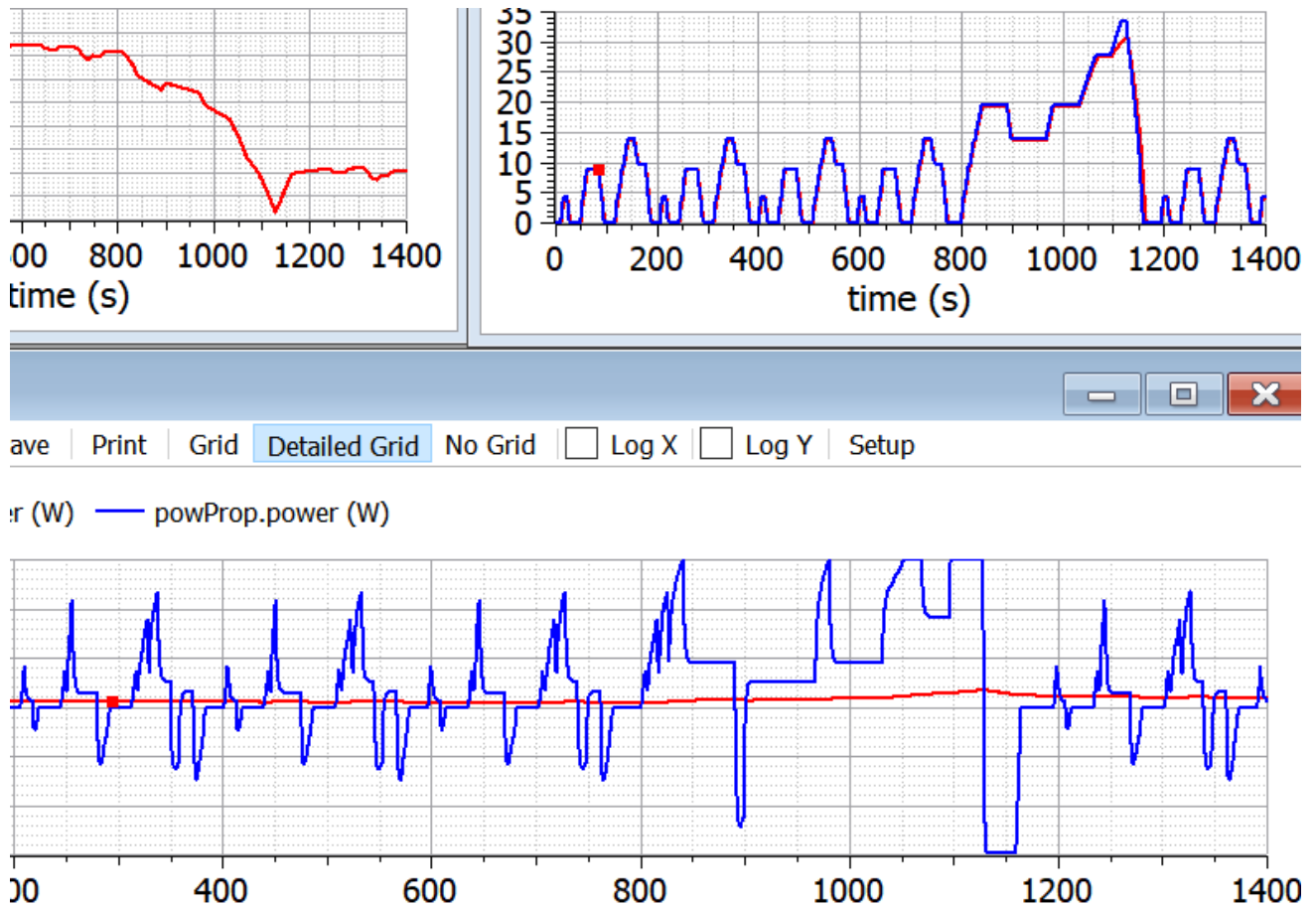


Figure 36. Plots of NEDC cycle using the system in figure 34.

In this case, simulation stops due to the battery becoming totally discharged. If simulation of figure 36 is repeated using a 50 Ah battery instead of one having 25 Ah (and having $I_{max}=1000$ A), the following results are obtained:



Powers, especially when the vehicle is in the extra-urban part, are much higher. At the highest speeds the power train hits its max power limit (150 kW)⁴, and actual speed is lower than the expected one.

Notwithstanding a much larger battery that what adequate for Sort1, this simulation reveals a weakness of our model: the SOC drops significantly. This is because we ask the ICE to deliver the average power train power, as computed by a simple power filter. I.e., we have no correction factors for SOC departure from its reference value. The situation is improved in the next section, where it is shown that even NEDC can be overcome with the 25 Ah battery, with a better control.

7.3 SHEV with SOC closed-loop control

The problem of SOC control, shown in the previous section, can be addressed by adding in the EMS a correction term to the ICE requested power, according to the diagram of figure 37.

⁴ In the SHEV simulations the parameter `mapsOnFile` of drive has been set to false, and therefore drive has unit efficiency.

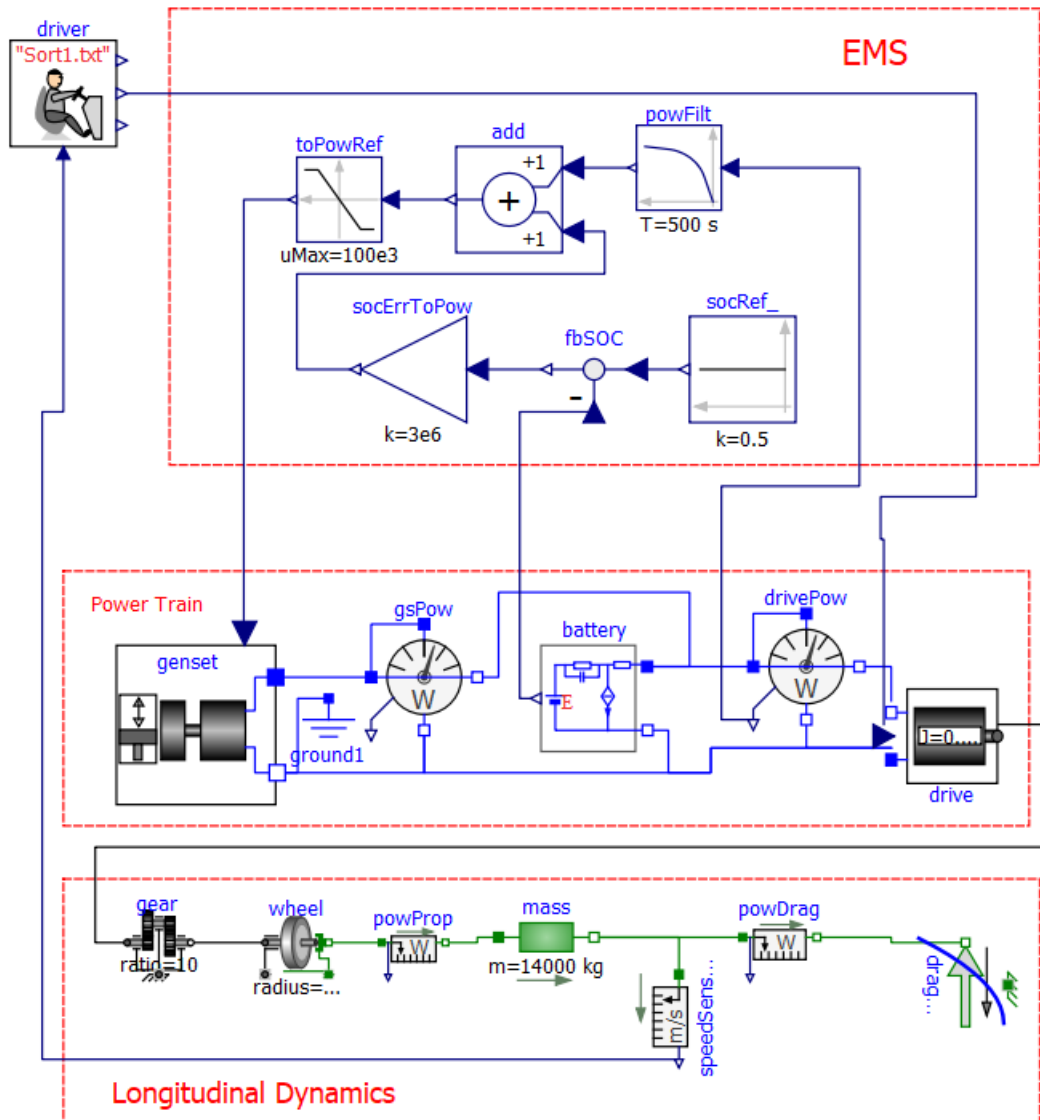


Figure 37: Diagram of the EHPTexamples.SHEV.SHEVpowerFiltSoc model (Series Hybrid Electric Vehicles, with SOC control).

This model requires, to run, some data to be specified on a txt file. To reproduce the results shown here the reader can use the provided "SHEVmaps.txt". How to link a txt file with a model is explained in section 2.3.

Some plots that can be obtained running the model whose diagram is in figure 37, and the battery is again set to 25 Ah, $I_{max}=500$ Ah, and the cycle is NEDC, are as follows:

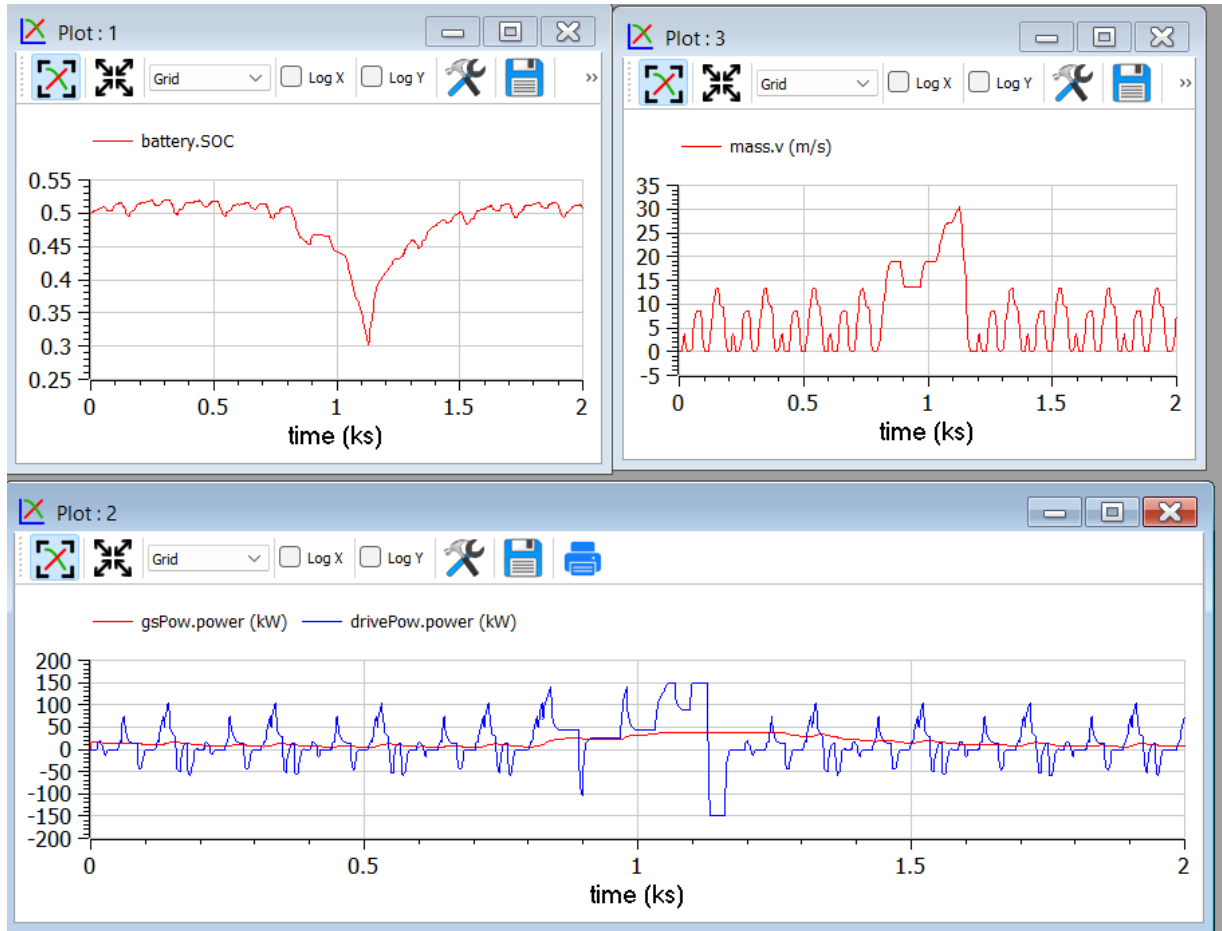


Figure 38. Plots of NEDC cycle using the system in figure 37.

Here, the SOC drops much less than the previous case, and continues to slowly recover, beyond $t=1200$ s, and go near to the reference set. The SOC set-point is `socRef`, the speed with which SOC is corrected to stay near the set point is the value of constant in `socErrToPow`. Note that the genset power stays largely constant between 80 s and 1300, because the max generator torque is reached (set to 500 Nm).

This simulation shows that, using proper SOC control, the 25 Ah battery is correct for this vehicle subject also to the cycle NEDC, much heavier than Sort1.

7.3.1 Proposed activity

In case the bus has a plug-in architecture, it may be useful to discharge more the battery near the end of daily service, to take advantage of the cheaper mains recharge. This can be obtained by a driver command “prepare for night”, which changes the SOC set-point. Similarly, if the bus is allowed to go in a special city centre in EV mode (ICE shut-off), maybe the driver wants the battery to prepare for this run with a command “prepare for ZEV”.

The reader is prompted to modify the model giving this further level of control to the driver. Indeed in modern vehicles, in addition to accelerator and brake pedals, it is often possible for drivers to choose also the *mode* in which he want to drive (sports, economy, etc.) this additional command we are suggesting falls within the family of additional driver commands.

7.4 SHEV with On/Off control

In the model shown in the previous section, it has been noted that the ICE power is below 10 kW. The ICE size, however, is usually much larger than this, for instance to allow sustained driving at a

“motorway” speed (e.g. 80 km/h) for instance during transfers to deposit or to another town (remember that our vehicle is a bus). With this requisite the ICE power should be at least 60 kW, as can be checked using equation (1) and possibly a constant-speed simulation with a high `socErrToPow` constant.

Just to have an idea on how an ON/OFF strategy could look like and work, we can simulate the provided SHEV_OO, which has the aspect shown in figure 39.

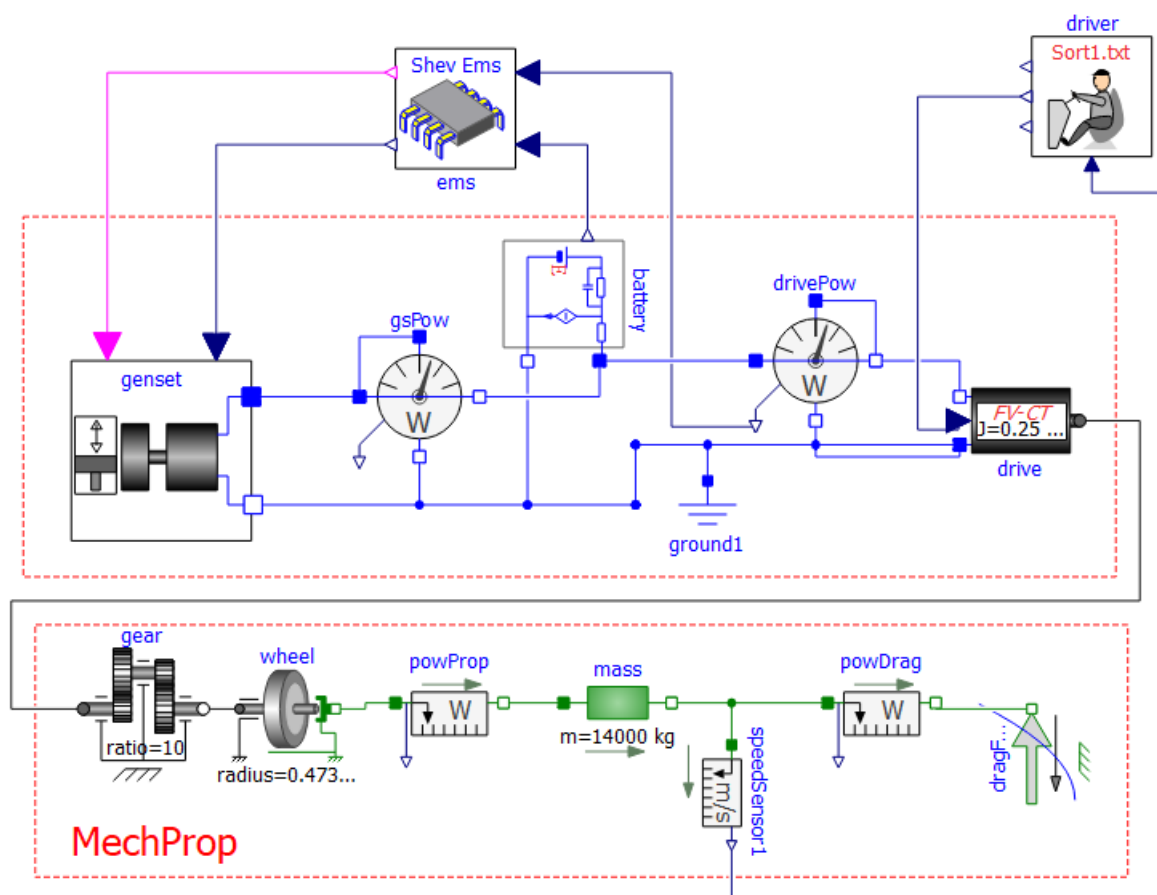


Figure 39: Model of the proposed Series Hybrid Electric Vehicles, with SOC control and ON/OFF.

This model requires, to run, some data to be specified on a txt file. To reproduce the results shown here the reader can use the provided “SHEVmaps.txt”. How to link a txt file with a model is explained in section 2.3.

The plots in figure 40 show some results with the Sort1 cycle repeated many times, using for the driver the extrapolation “Periodic”.

To have a fair comparison, the final SOC of the OO simulation is made equal to the other case, acting on `ems.socRef`. After a few attempts the value of 0.58 was chosen.

The SOC feedback constant is set to 30 and 300 kW for the PowerFiltSOC and SHEV_OO respectively. The start value of powerFilter in both cases was set to 10 kW. Namely variables ems.powFilt.yStart (OO) and powerFilt.yStart (non OO) were set to 10 kW.

Red curves refer to the ON/OFF operation, while blue ones are from the SHEVPowerFiltSoc model, and are reported here for comparison.

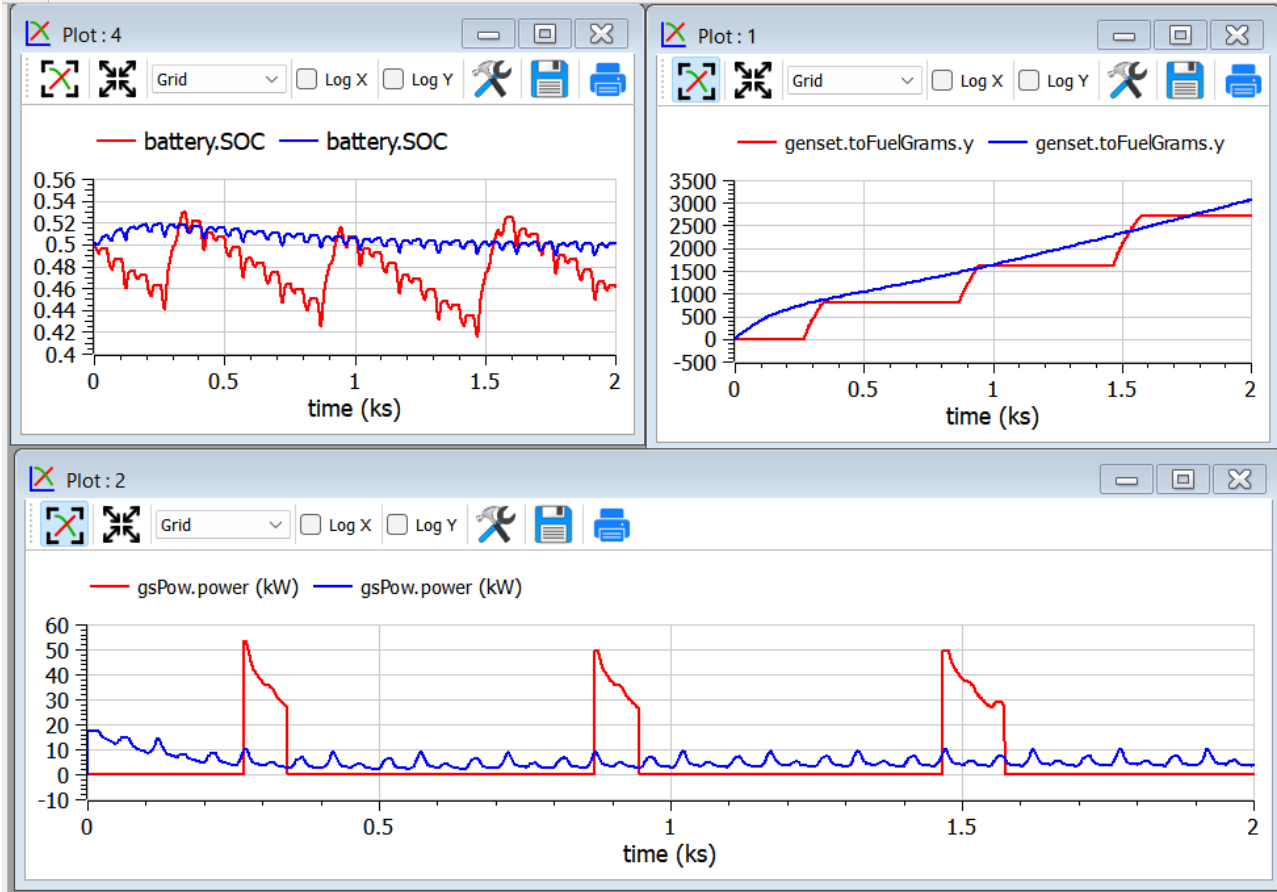


Figure 40. Comparison of SHEV behaviour with different control strategies (SHEVpowerFiltSOC, blue, and SHEV_OO, red Sort1)

The plots show that the fuel consumption with ON/OFF is much lower than the other.

7.4.1 Proposed activity

In the previous section it has been seen that the proposed ON/OFF control reduces fuel consumption.

However the consumption depend on the switch-on and switch off thresholds chosen, and on the used cycle. In particular, the advantages are much more with Sort1 Cycle, which has a very low average power than with NEDC.

The reader is therefore prompted to change the cycle and the thresholds, to see their effect on consumption. Remember that comparisons are to be made at equal initial and final SOC.

7.5 Simplified SHEV with ON/OFF control

Model SHEV_OO does the best to be realistic, even with still within a simplified approach to model series-hybrid power trains.

A simplified version of it is in OO_noMaps, which is more manageable, since it allows seeing significant changes, e.g. the effect of changing genset power, without having to change too many parameters.

If run with the default parameters, we get for instance the following plots for the same variables displayed in figure 40:

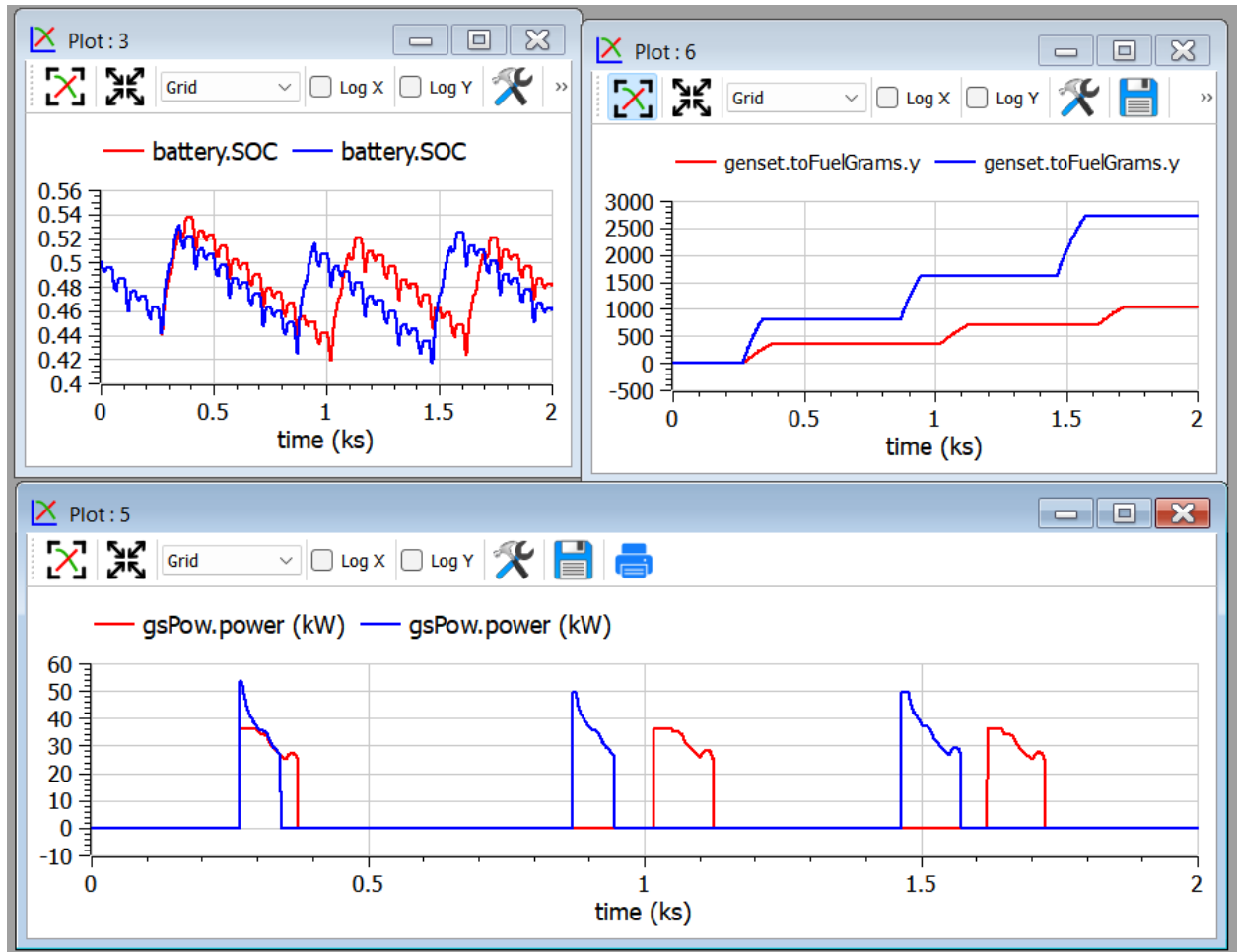
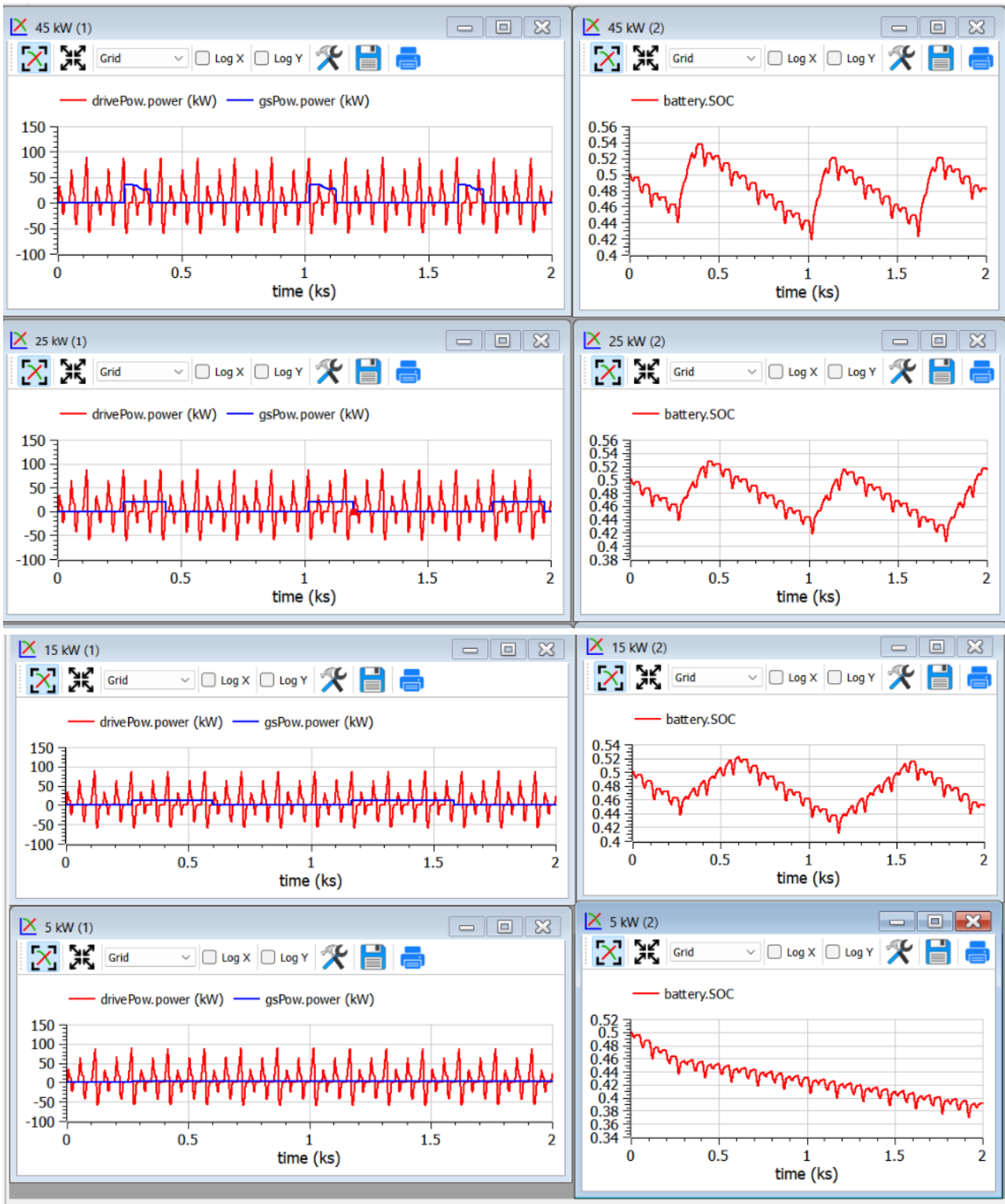


Figure 41. Comparison of SHEV behaviour with different control strategies (SHEV_OO, blue, and OO_noMaps, red.)

In the following figure we just compare gsPow for models SHEV_OO and OO_noMaps:

It is interesting to see the effects of progressively reducing the genset power. Below we see what occurs when we reduce it from 45 kW to 25, 15, 5 kW:



While we reduce the genset power, the ON times will grow progressively. Below around 6 kW, the genset will stay ON all the time, with the given cycle, and SOC cannot be sustained: the genset behaviour becomes like that of a BEV; in this case the SHEV becomes a range-extender EV.

7.6 PSD-HEV model

7.6.1 Background

While a SHEV can be considered a variation of a battery EV, with the addition of the fuelled source, a PHEV can be considered a variation of a conventional Vehicle, with the addition of an electric machine. In both cases, the smaller the addition, the nearer the vehicle to the original one (battery EV and conventional ICE based, respectively).

To create a simulation model of a PHEV therefore, we should start from a simulation of a conventional vehicle, with gears switching. This is not done in the current version of this model however, to reduce its complexity and length.

Instead, as an example of a general purpose hybrid vehicle, we will discuss a complex hybrid architecture, often called Series-parallel architecture, which is based on a planetary gears, called Power Split Device (PSD).

This architecture is installed on-board very successful commercial hybrid vehicles (see note 3).

The implementation of this architecture we consider is shown in fig. 42. T 's are torques, Ω 's are angular speeds, P 's are powers.

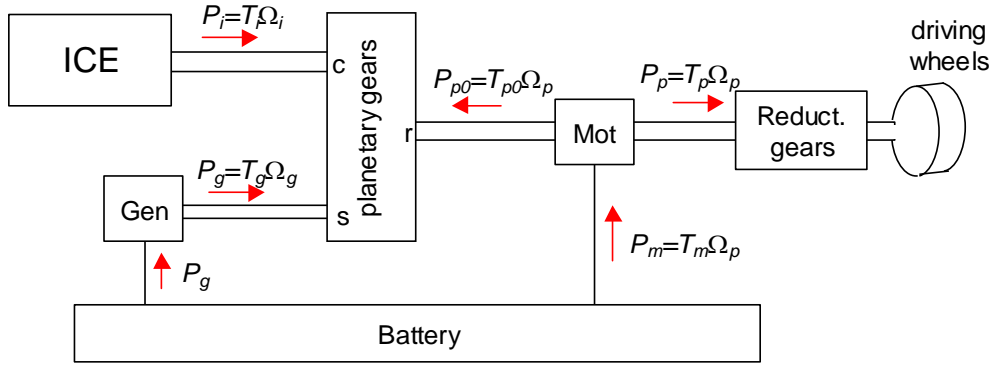


Figure 42. The considered PSD-based power train architecture.

The core of the power train is the planetary gears, which has an inner structure of the kind shown in figure 43.

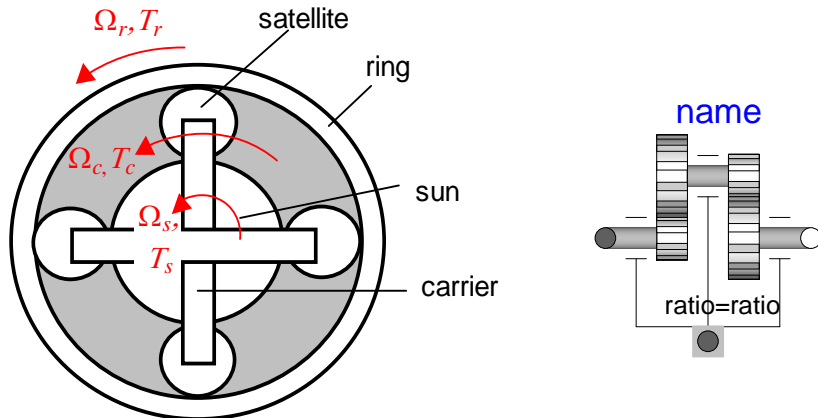


Figure 43 PSD (idealized) construction and corresponding MSL model (idealPlanetary).

It is nice to know that a planetary gear of this kind is already present in MSL. Its icon is shown in fig. 43 as well, in its right part.

A simple analysis of kinematic and static relations of this device shows that two static and one kinematic equation exist. They can be expressed in different, equivalent forms. If we indicate with ρ the ratio between ring and sun radius and with $\sigma=1/\rho$, we can equivalently write, as kinematic equations:

$$\Omega_r = (1 + \sigma)\Omega_c - \sigma\Omega_s \quad \Omega_s = (1 + \rho)\Omega_c - \rho\Omega_r \quad \Omega_c = \frac{1}{1 + \sigma}\Omega_r + \frac{1}{1 + \rho}\Omega_s$$

And as static equations:

$$\begin{cases} T_r = \rho T_s \\ T_c = -(1 + \rho)T_s \end{cases} \quad \begin{cases} T_s = \sigma T_r \\ T_c = -(1 + \sigma)T_r \end{cases} \quad \begin{cases} T_r = -\frac{1}{1 + \sigma}T_c \\ T_s = -\frac{\sigma}{1 + \sigma}T_c \end{cases}$$

Because of these relations, once one angular speed is known the PSD geometry determines the ratio between the other two; once a torque is known, the geometry of PSD determines the other two.

The above formulas are written according to the sign conventions used in MSL: torques are the torques acting from the outside towards the PSD, and when they have the same sign as the corresponding angular speed, positive mechanical power is transferred from the outside towards the PSD.

In the architecture shown in figure 42 the internal combustion engine is flanged to the PSD carrier, the electric drive called *generator* “Gen” to the sun, the output shaft, to which the electric drive called *motor*, “Mot” is connected, to PSD’s ring.

The architecture shown in figure 42 allows to choose the power to make flow in the battery, independently of the vehicle operation point. If this power is zero, we use a mode which we call “electric balance” mode, or CVT mode, since in this case we use the whole architecture as an intelligent Continuously-Variable Transmission (CVT). Otherwise, we can use some strategy to control the battery SOC, in a way similar to what done above with SHEV’s.

“Gen” and “Mot” are reversible electric drives, since both can operate in motor and generator mode. However during vehicle traction and with all PSD speeds positive, Gen actually operates as a generator and Mot as a motor; the opposite occurs when, being the speeds still positive, the vehicle is regenerative braking and PSD output power is reversed.

7.6.2 ICE model

Our PSD model requires a map-based ICE model, not previously used. Our ICE model’s diagram is shown in figure 44.

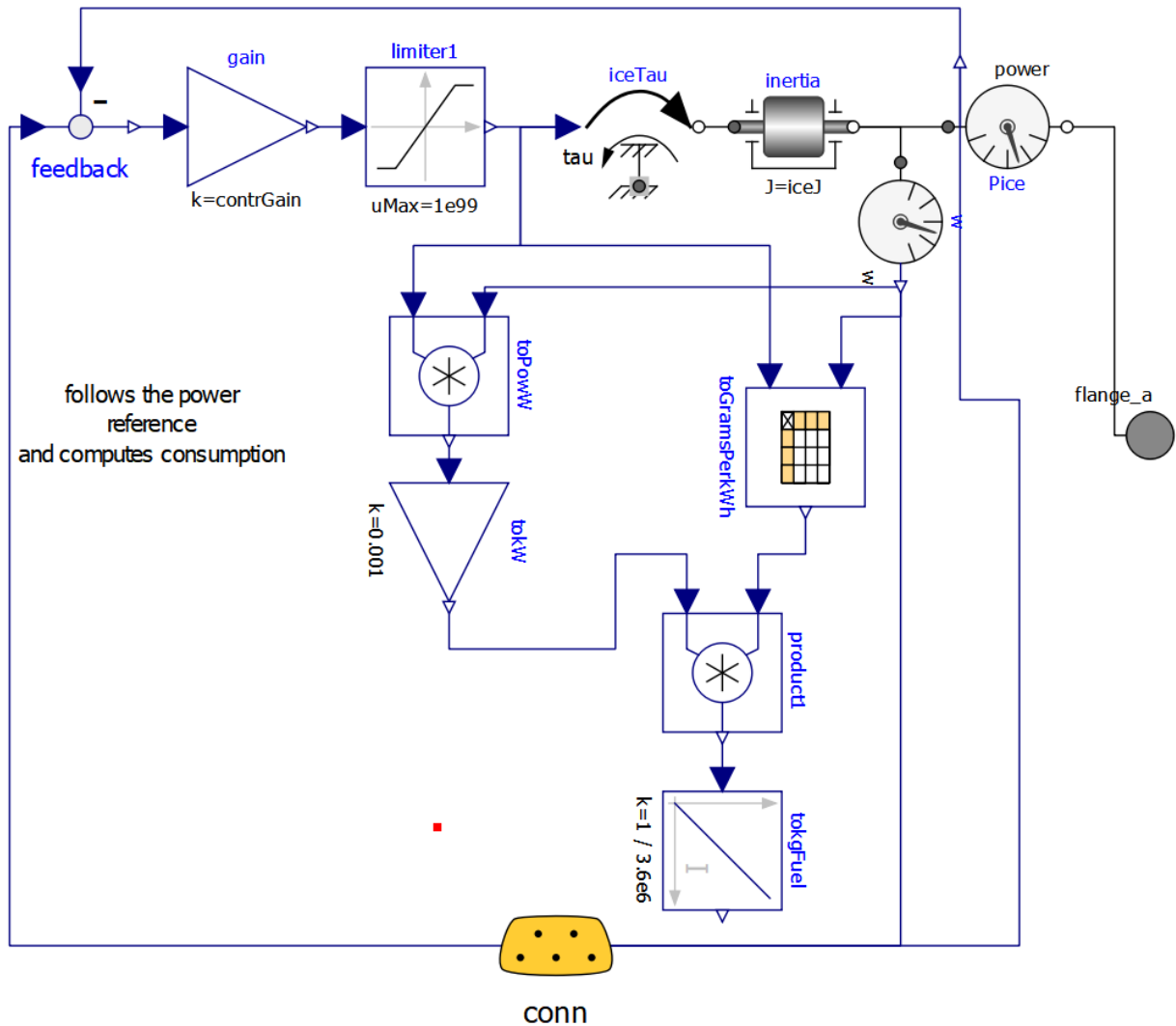


Figure 44. ICE model diagram (EHPTlib.MapBased.IceConnP)

The ICE dynamics is modelled by `inertia`; its fuel consumption by a map, read from a txt file (e.g. in the provided `PSDMaps.txt`). This model has a simple inner control logic, which tries to follow the input signal, sent through the expandable connector `conn`, containing a torque request.

Here, by simplicity, the maximum torque is limited by a constant value. More realistically, it should be limited using a variable limit, whose value is function of the rotational speed.

7.7 Simulation “PSecu1” (power filter)

In a way totally similar to SHEV basic model previously discussed now we show a logic in our PSD power train which tries to make the ICE deliver the average power needed for propulsion.

The principle diagram in figure 42 in our Modelica code becomes the diagram shown in figure 45.

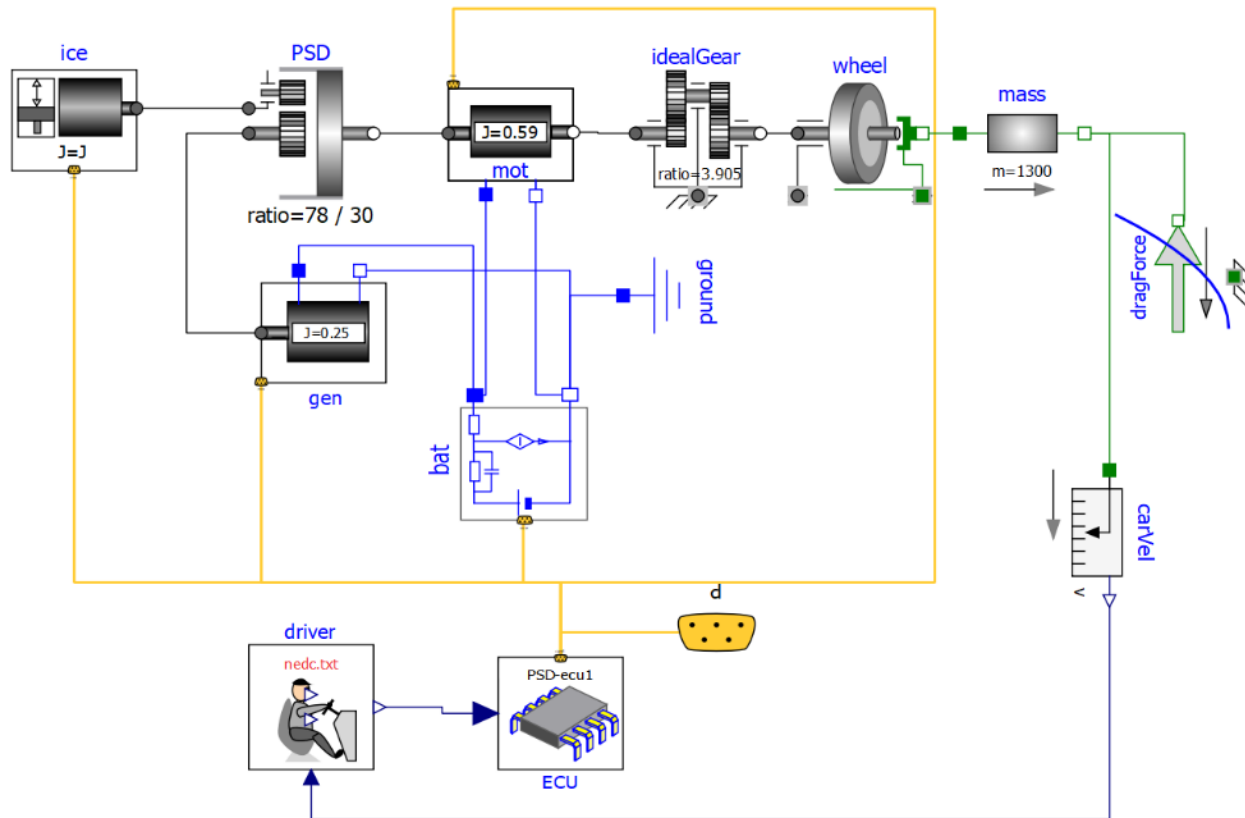


Figure 45. Modelica Diagram of the proposed PSD-based hybrid drive model (EHPTexamples.PSD.PSecu1).

This model requires, to run, some data to be specified on a txt file. To reproduce the results shown here the reader can use the provided “PSDmaps.txt”. How to link a txt file with a model is explained in section 2.3.

The reader should easily distinguish the elements in figure 42, with the addition of the control part, constituted by driver and Electronic control unit (ECU).

To reduce the wiring complexity, the various submodels’ signals are connected to each other by means of a bus (yellow).

To do this one has to:

- 1) decide with signals must flow in the bus and decide unique names since the various signals are distinguished by their name
- 2) create a specialised version of the various components, which are able to exchange their signals though expandable connectors, using the unique naming standard. The reader can easily see that the components in the above diagram are directly derived from the ones used above, with some modifications to allow connection though expandable connectors
- 3) create an Electronic Control Unit which manages the signals and implements the vehicle control logic.

The signals that will pass through our signal bus are listed in the following table:

<i>Name</i>	<i>sender</i>	<i>users</i>	<i>comment</i>
batPowDel	bat	<i>disp</i>	Electric power delivered by the battery
batSOC	bat	controller	Battery state-of-charge
genTauLim	gen	ice	maximum generator torque at its actual speed
genTauRef	ECU	gen	
genPowDel	gen	<i>disp</i>	generator power delivered through its mechanical flange
genW	gen	controller	generator angular velocity
icePowDel	ice	<i>disp</i>	ICE power delivered through its mechanical flange
icePowRef	ECU	ice	ICE reference power
iceW	ice	<i>disp</i>	ICE angular velocity
iceON	ECU2	<i>disp</i>	ICE ON/OFF state (°)
motPowDelB	mot	contr	Mot power delivered through its flange B
motPowDelAB	mot	<i>disp</i>	Sum of Mot powers delivered through flanges A and B)
motTauRef	ECU	mot	
motW	mot	<i>disp</i>	Mot angular velocity

(°) used only by ECU2, in simulations with ON/OFF

Several names are composed by two or three parts. The first part indicates the component to which the variable refers, (for instance "ice") the central one which kind of variable it is (for instance "tau" means torque), finally the(optional) third part indicates info about de variable for instance "del" means delivered, i.e. for a power a positive value indicates that the power is delivered to the outside by the component the variable belongs to; "ref" indicates a reference value (a variable that is used as a reference by some controller) "Lim" a limit, or maximum, value.

Just to have an idea of the changes needed to make specialised, connector of components in the following figures we see first the component `OneFlangeFVCT`, then its specific-connector-based version made with signal names related to PSD power trains, `OneFlangeFVCTConn`.

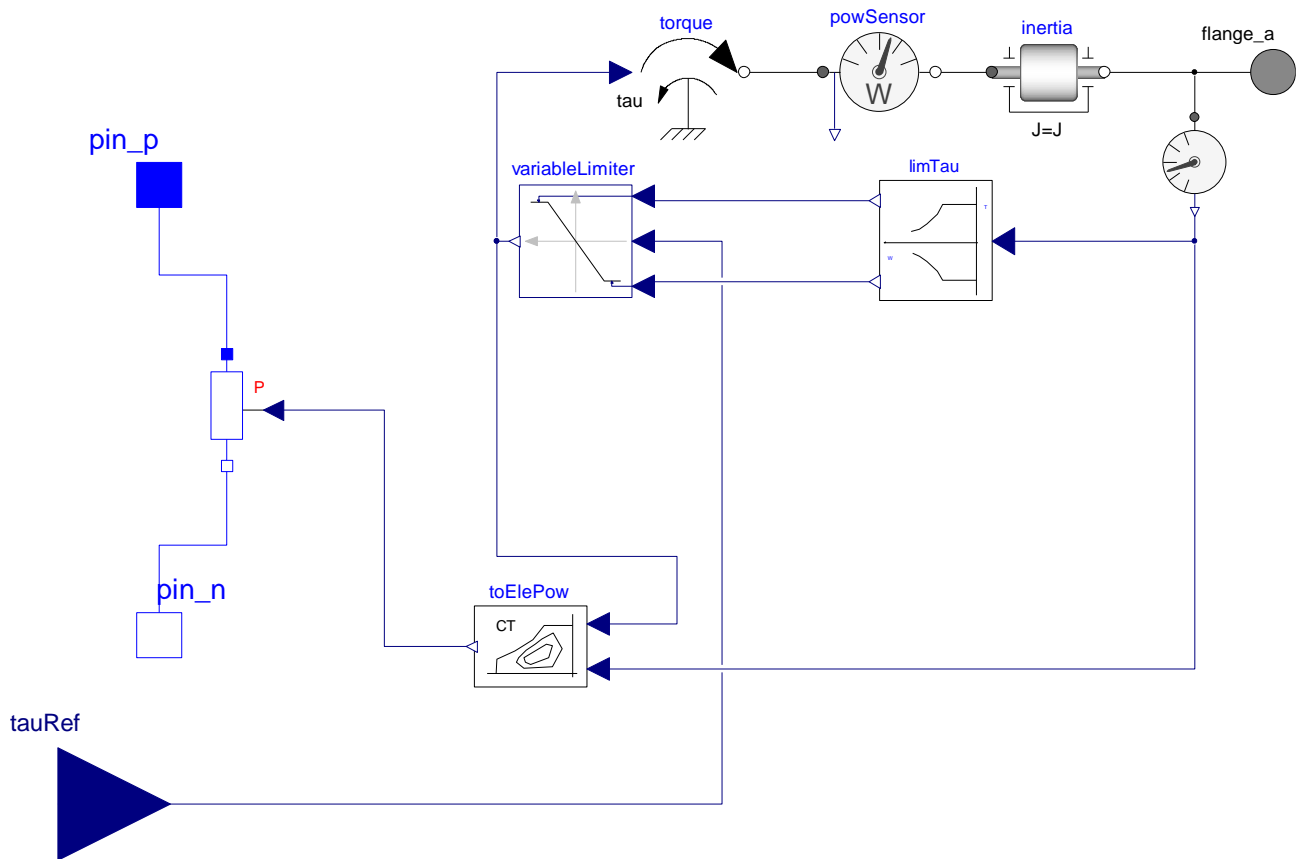


Figure 46. OneFlange model of an electric drive Input signal is reference tau τ_{ref} (EHPTlib.OneFlangeFVCT).

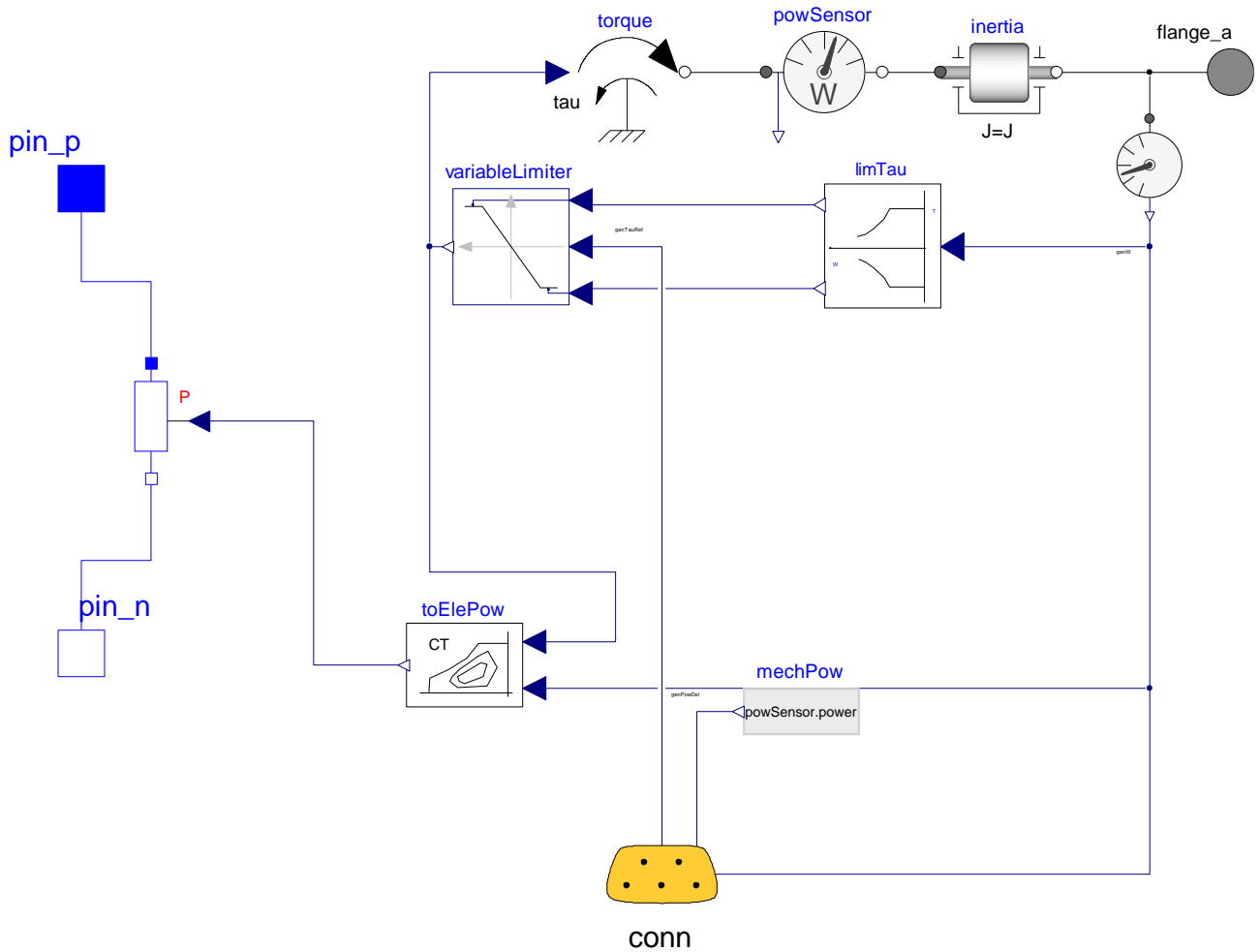


Figure 47. PsdGenConn Model of the “generator” machine of a PSD drive. It can be seen as a specialized version of the OneFlangeModel in figure 46, with a connector and signal names related to PSD generator.

The general arrangement is the same, but the torque reference instead of being taken from an input signal, it is taken from the expandable connector, where it is called `genTauRef`.

Moreover, an additional signal is outputted, which can be exploited by any other model connected to the same bus, which is the mechanical output power and named `genPowDel`.

Now we will use the model whose gist has been described with three different control strategies, starting from the simpler moving towards the more complex.

7.7.1 Proposed activity

The reader can enhance the supplied `IceConnP` model, making the maximum torque function of rotational speed. He can follow, as an example, the implementation of this kind of torque limitation used in the provided model `wbEHPTLim.MapBased.ECUs.GMS`.

Here we control the vehicle in such a way that the ICE delivers the average power needed for propulsion, and delivers it at its optimal speed, i.e., at the speed corresponding, for the given power, at the minimum fuel consumption. This logic implemented in the EMS, which has the scheme shown in figure 48.

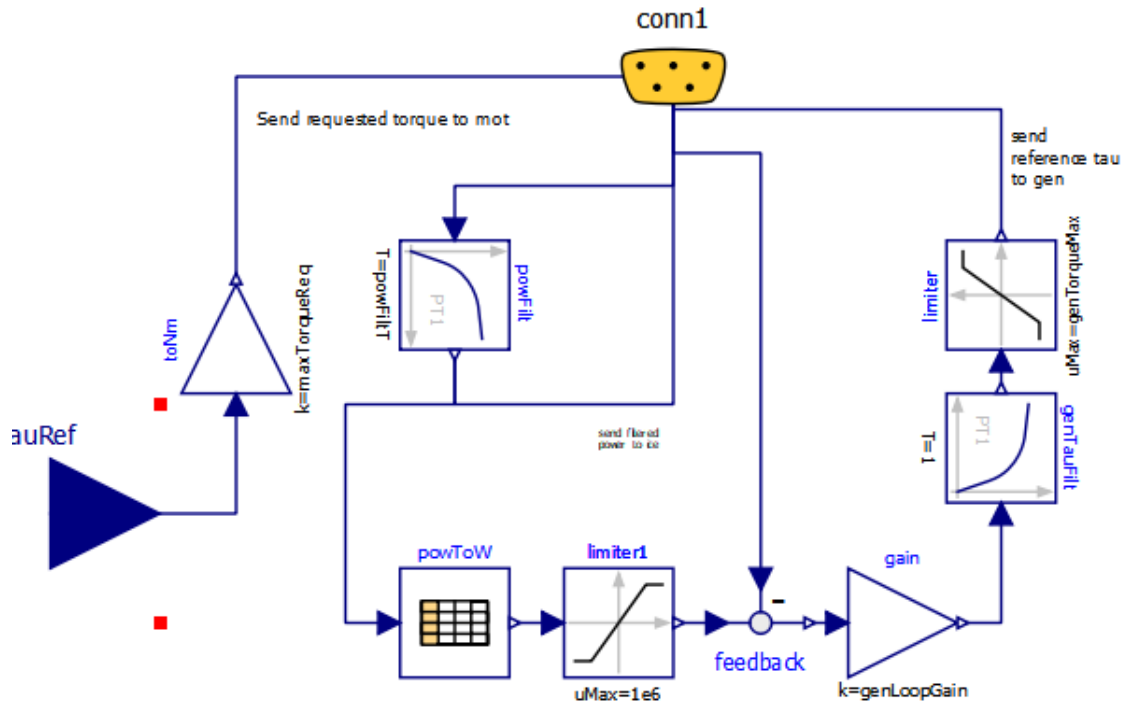


Figure 48. Diagram of “ECU1” version of PSD-HEV Energy Management System (EHPTlib.MapBased.ECUs.Ecu1).

In this scheme:

- the received tau reference is sent back to connector as the signal `motTauRef` with just a scale adaptation. this way the motion control is attribute to mot, which is requested a torque directly depending on the torque request from the driver
- to decide the power the ICE has to deliver, through conn, the propulsion power signal `icePowRef` is received, filtered through a first-order block, and the filtered back sent to ICE as a power request
- the control of the ICE speed is attributed to the generator. Starting from the ICE power request, through the table `powToW` the corresponding optimal speed is determined, and used for a reference of a closed loop control, which determines the gen torque to be generated, `genTauRef` to try to obtain the wanted ICE speed.

It is ICE responsibility, through its internal control, to deliver the power the EMS requires. This is done inside the ICE block having the diagram shown in figure 44.

The reader is invited to perform the simulation of this model using the provided PSD.Psecu1 model. Just to give an idea of the result, some pictures are shown below:



Figure 49 Some plots obtained when simulating the model in figure 45 (EHPTexamples.PSD.PSecu1), controlled with the ECU shown in figure 48 (ECU1) subject to NEDC cycle.

The plots show that the vehicle closely follows the requested NEDC cycle, with the ICE delivering (*icePowDel*) a smoothed version of the power needed for propulsion (*icePowDelAB*).

However, this is an open-loop control of the power that, in this case, causes a non-negligible shift in the battery SOC *d.batSOC*. In fact this cycle has a larger power request in its last part, the suburban section of the cycle, which is felt with delay by the averaging block, and never corrected.

Therefore it appears necessary to enhance the control logic, so that SOC is kept under better control.

7.8 Simulation “PSecu2” (power-filter and SOC-loop control)

To enhance the previous simulation, the EMS is modified as follows and called ECU2. Note that no other component has changed.

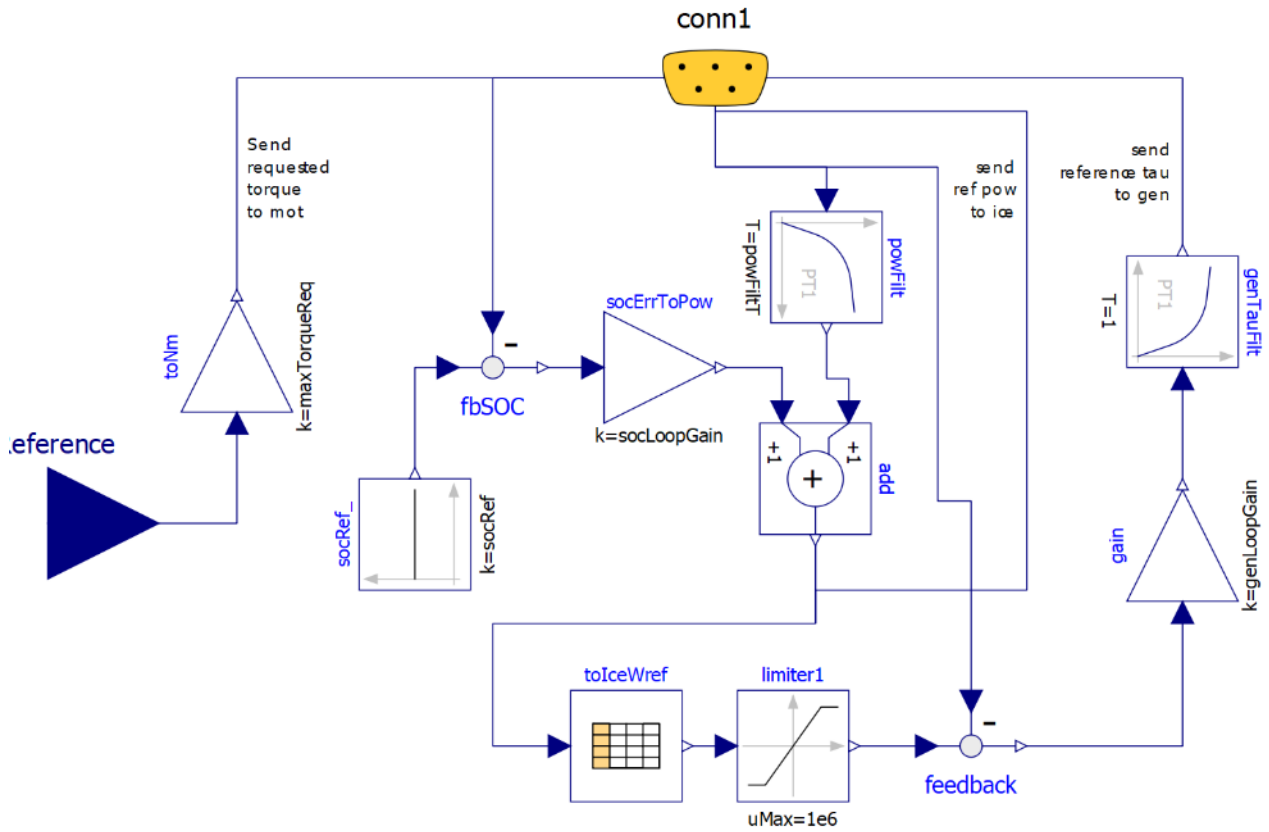


Figure 50. Modelica diagram of the proposed EHPTlib.ECUs.Ecu2 model.

A comparison with the diagram in figure 48 shows the difference: now the requested power is determined by the block `add`, which has two contributions: in addition to the previous one, consisting in the averaged propulsion power, we have a signal proportional to the SOC error.

The corresponding results are shown in figure 51.

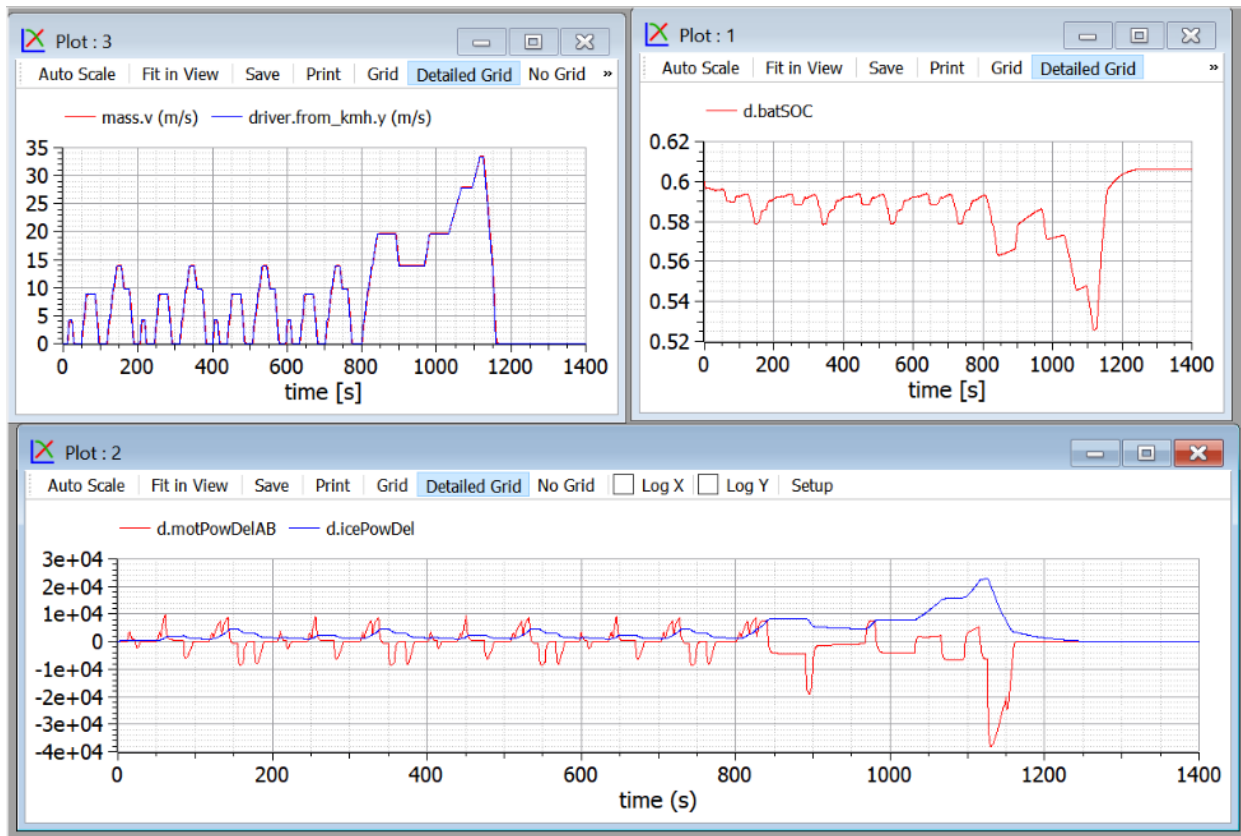


Figure 51. Some plots obtained when simulating the model in figure 45, controlled with the ECU shown in figure 50 (EHPTexamples.PSD.PSecu2), subject to NEDC cycle.

We see that now SOC is kept under much better control.

The control we implemented here is closely determined by the action of the two terms of the `add` block in `Ecu2`. We can enlarge or reduce the ice power and battery SOC fluctuations. With large values of the parameter `powFiltT` (the time constant in the block `powFilt`) and small values of parameter `socLoopGain` (the gain of the `socToErrPow` block) we will have a rather constant ICE speed, and large power and energy fluctuations in the battery, which needs to be sized accordingly. On the opposite side, smaller time constants of the power filter and large gains in the `socToErrPow` block will help keep the battery at a nearly constant level. In this case the battery can be smaller.

In extreme cases, we could design a vehicle with a very small battery, in which the propulsion power is nearly totally given by the `ice`, while during transients `gen` and `mot` supplied powers that are nearly one the opposite of the other (`gen` generates the power `mot` uses) to allow the PSD operating point to be changed, and the PSD architecture operates more like a Continuously Variable transmission than a true hybrid. This extreme choice, however, is not advisable since it would not leave room for regenerative braking.

To see these things in practice, the reader is advised not to tamper with `ecu2` model directly, but, instead using its parameter dialog box, which has the following appearance:

OMEdit - Component Parameters - ECU in wbEVPkg.PSD.PSecu2

Parameters

General Modifiers

Component

Name: ECU

Class

Path: wbEHPTlib.MapBased.ECUs.Ecu2

Comment: Power Split hybrid power train controller, with SOC control, without ON/OFF

Parameters

genTorqueMax	80	...	N.m	maximum absolute value of gen torque
socRef	0.6	...		Target value of SOC
socLoopGain	1e5	...	W	soc loop gain
maxTorqueReq	80	...	N.m	Torque request (Nm) that corresponds to 1 from driver
genLoopGain	1.0	...	N.m/(rad/s)	Control gain between ICE speed error and gen torque
powFiltT	60	...	s	Power filter time constant (s)

OK Cancel

Acting on it, we change the value of the instance of `ecu2` we use. The reader is prompted to do this while doing the following *proposed activity*.

7.8.1 Proposed activity

The proposed model is for teaching purpose and not for production. Therefore is not very robust. However, starting from the working version, the reader is invited to play with `socLoopGain` and `powFiltT` in `ECU2`, to try to obtain the behaviour described in the comment just above here. In particular it is given the task to size the control so that a small battery (just around 500 Wh, e.g. 100 cells of the type used in our models but having 5 Ah nominal capacity) is sufficient to cover NEDC, with SOC staying between 0.2 and 0.8.

To avoid entering operating zones which create numerical difficulties, it is suggested to make small variations at each simulation and to save models which better the previous ones, before making additional attempts.

As an example we can show what happens with a small battery: 5Ah-100 cells, which means for a nominal voltage of 1.2 V 240Wh: a very small battery. Using this battery, `powFiltT`= 10s, `socLoopGain`= 20 kW and `SOCinit`=`SOCref`=0.6: we get the following curves:

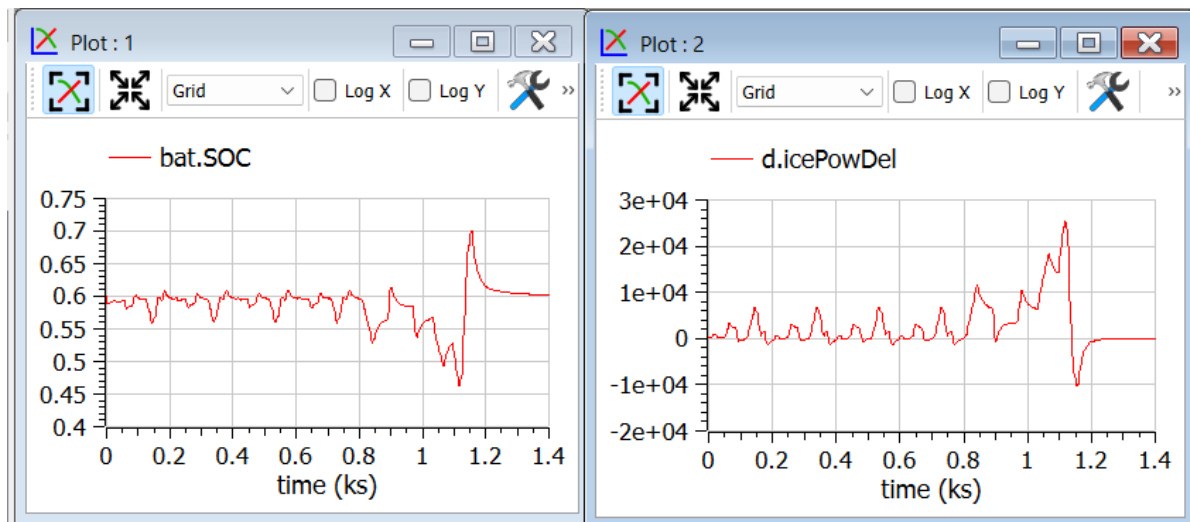


Figure 52. Some plots related to a small-battery proposed activity for PSD-hybrid.

In this case we privilege a small battery at the expenses of large ICE power variations.

Note that in the final part of our plots, when we have a strong braking action, the battery gets filled over the reference value of 0.6.

In case of even stronger braking, e.g. when running downhill, this small battery gets filled and part of the energy must be dispersed into heat using mechanical brakes.

Since mechanical braking is not present in the proposed models, if we try to further reduce the battery, during last braking the battery reaches SOC=1, and simulations stop since the power train cannot brake anymore in absence of dissipative brakes.

The model in the package that solves this Proposed Activity is called PSecu2PA.

7.9 Simulation “PSecu3” (with power-filter SOC control and ON/OFF)

Consider the results shown in the previous section 7.8. From them, we see that on one hand, with the used battery size, battery SOC under NEDC varies a little; on the other during the urban part of it, the average power, which becomes the ICE power, is rather small, and therefore implies low ICE efficiency.

Because of this it is in this case advisable to envisage an ON/OFF strategy which automatically shuts-off the engine to avoid it to be too light loaded, in a way that is very similar to those seen for SHEV's.

To attain this control, a further modification of the EMS is made, giving rise to the Ecu3, whose diagram is shown in the following figure:

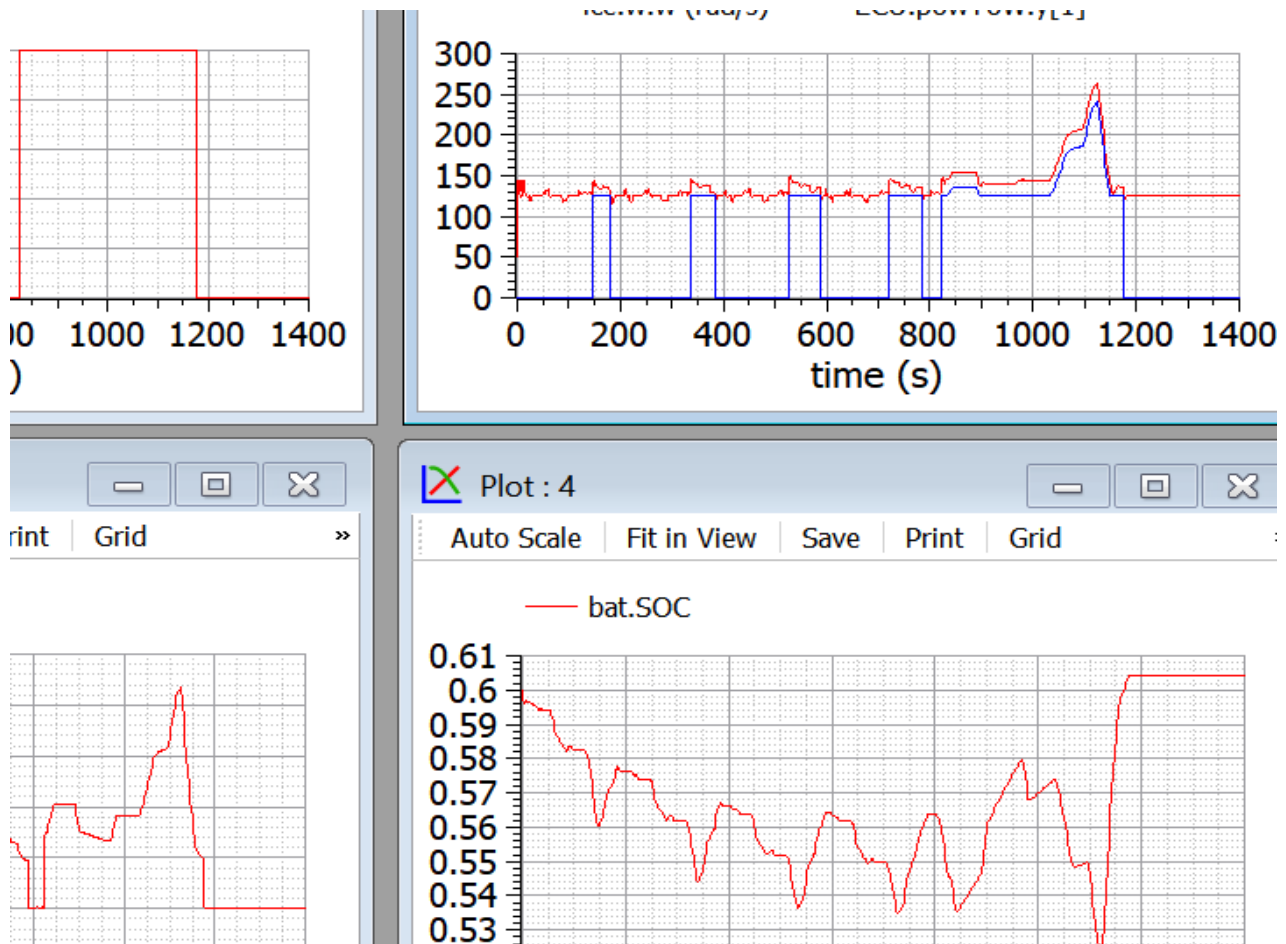


Figure 54. Plots from EHPTexamples.PSD.PSecu3 model.

It is easy to see that the ON/OFF strategy allows higher ICE powers when in ON state, and the SOC control is still guaranteed. The top-right picture shows that when ON the ICE operates near its optimal speed.

Ice is able to give information about the consumption in this case, compared with case in sect. 7.8. This comparison must be made having the care of choosing control variables which allow to have in one case the battery more discharged than in the other, otherwise to comparison would be unreliable. So, to obtain the following pictures (fig. 55), some slight modifications on the control signals are made to obtain this SOC equivalence ($ecu.SOC_{ref}$, for the case with ON/OFF, changed to 0.65).

The results obtained playing with the supplied models are interesting, and showing below: the two simulations start at the same SOC, and reach the same SOC again for $t=1250$. At this time the fuel consumed with ON/OFF is 0.356 kg, in comparison to 0.449 kg without it, implying a consumption reduction by over 20 %

It is reasonable to expect that using different thresholds the results can be even better. Naturally, for production usage, before deciding the ON/OFF thresholds several different cycles must be considered, while optimising on just a single NEDC is nearly a non-sense.

l.batSOC

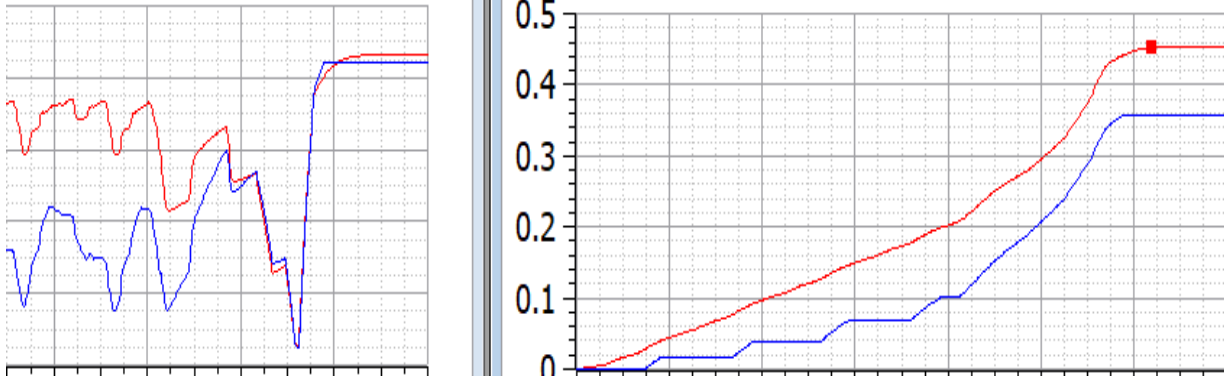


Figure 55 Fuel consumption comparison between non-ON/OFF and ON/OFF.

To replicate run EHPTexamples.PSD.PSecu2 and EHPTexamples.PSD.PSecu3 using the supplied parameters.

8 Some submodels of EV and HEV models

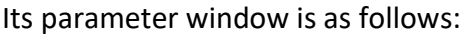
The library contains several map-based models. Here we propose just a short description of some of them.

8.1 OneFlange models

We have two OneFlangeModels, derived from PartialOneFlange (OneFlange and OneFlangeConn). They differ from how the requested torque is sent into them, from either a Real input or a connector. Operation of OneFlange models can also be customised through the choice of internal flags: limitsOnFile, efficiencyFromTable, effMapOnFile.

Their differences easily understood looking at the information windows of the corresponding models, and on the provided examples, as discussed in section 11.2.

The diagram of PartialOneFlange is as follows:



OMEdit - Parameters - EHPTlib.MapBased.Partial.PartialOneFlange

Parameters

General Modifiers

Class

Path: EHPTlib.MapBased.Partial.PartialOneFlange

Comment: Partial map-based one-Flange electric drive model

General parameters

J: 0.25 kg.m² Rotor's moment of inertia

uDcNom: 100 V Nominal DC voltage

limitsOnFile: false = true, if torque and speed limits are taken from a txt file, otherwise they are tauMax and wMax

wMax: 3000 rad/s Maximum speed (for efficiency and torque limitation when limitsOnFile=false)

powMax: 22000 W Maximum mechanical power (used for efficiency and torque limitation when limitsOnFile=false)

tauMax: 80 N.m Maximum torque when limitsOnFile=false

Torque limitation related parameters

tlTorqueFactor: 1 factor applied to input torque for limits they are from file

tlSpeedFactor: 60/(2*pi) factor applied to input speed for limits when they are from file

limitsFileName: "noName" File where efficiency table matrix is stored

tauLimitsMapName: "noName" Name of the on-file lower torque limit

Efficiency related parameters

efficiencyFromTable: ☒ =true if efficiency if from a table (either online or from a file); otherwise use a the built-in loss formula

effMapOnFile: ☐ = true, if tables are taken from a txt file

effMapFileName: "noName" File where efficiency table matrix is stored

eTorqueFactor: 1 factor applied to input torque for efficiencies when they are from file

eSpeedFactor: 60/(2*pi) factor applied to input speed for efficiencies when they are from file

effTableName: "noName" Name of the on-file efficiency matrix

effTable[3, 3]: [0, 0, 1; 0, 1, 1; 1, 1, 1] rows: speeds; columns: torques; both PU of wMax and tauMax

Loss-formula parameters

A: 0.006 fixed losses

bT: 0.05 torque losses coefficient

bW: 0.02 speed losses coefficient

OK Cancel

Some fields are automatically greyed-out when they become inactive and vice-versa.

When `limitsOnFile=false` block `limTau`, imposes just a fixed limit in the output torque and power, according to the following rule:

*If we call w the angular speed When $w < w_{Max}$ the output is τ_{Max} if $\tau_{Max} * w < P_{nom}$, otherwise it is P_{nom}/w . But if w is larger than w_{Max} τ_{Max} is rapidly falling to zero (reaches zero when speed overcomes w_{Max} by 10%).*

In this way torque is limited to τ_{Max} up to base speed, then to the value such that P_{max} limit is involved; after w_{Max} the limit goes rapidly to zero. Variable state identifies what limit is in force:

state=0 below base speed;

state=1 before w_{Max} ;

state=2 slightly above w_{Max} (torque rapidly brought to zero)

state=3 above w_{Max} , torque brought to zero;

The determination of electric power is done inside `toElePow` or `toElePow1` and blocks, which multiply its two inputs and then applies to the result, which is the requested mechanical power, the drive efficiency to determine the electric power to request from the DC source.

Blocks `toElePow` or `toElePow1` are conditional: only one of them is active, depending on the value of `efficiencyFromTable`.

8.2 ICE models

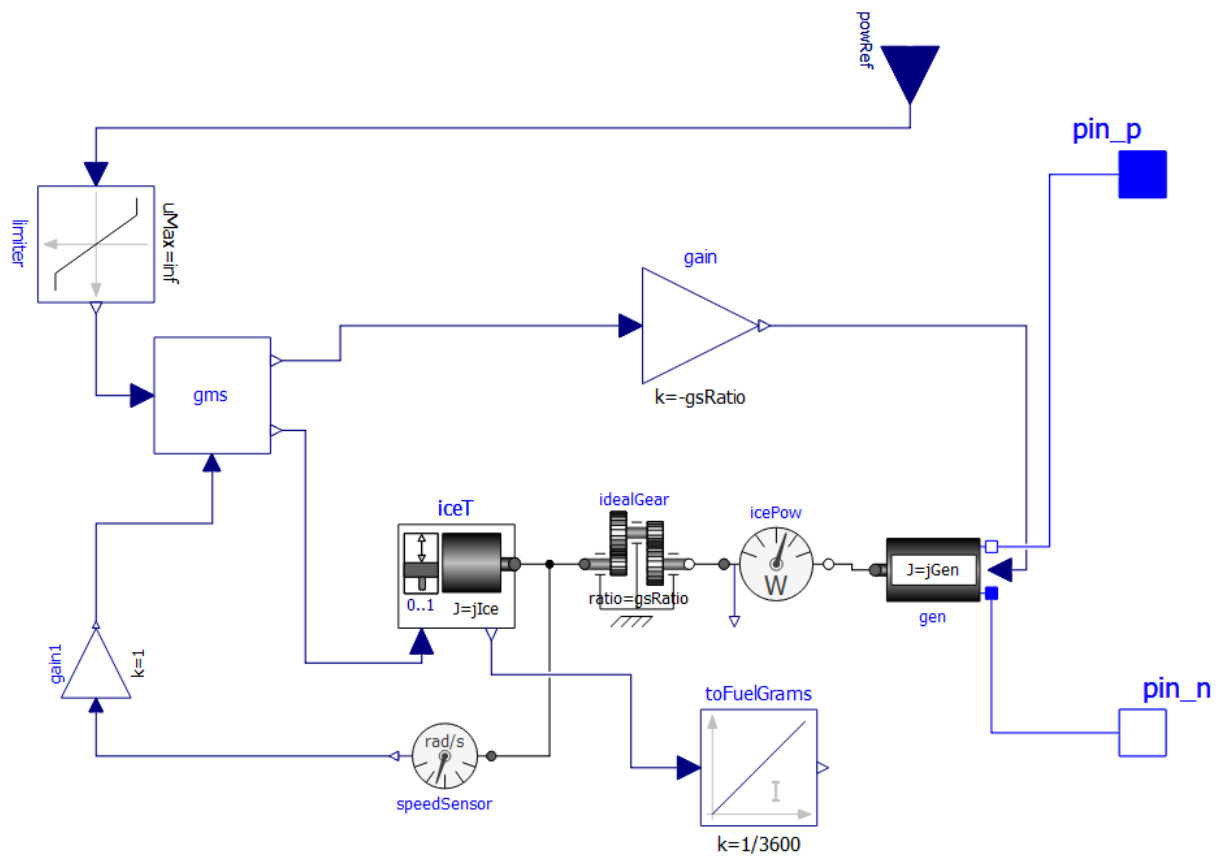
The library contains several map-based models for Internal Combustion engines.

- `IceP`, which receives as input a power request. It tries to deliver the requested power at its output mechanical flange
- `IceConnP` is similar to `IceP`, but it exchanges signals with its outside through an expandable connector
- `IceConnPOO` is similar to `IceConnP`, but allows the ICE to be switched On and Off.
- `IceT`, which receives as input a torque request. It tries to deliver the requested power at its output mechanical flange
- `IceT01` is like `iceT`, but its input torque is not expressed in newton; it is a normalised value between 0 and 1: a value of 1 indicates the maximum allowable torque.

All these models indirectly inherit from `PartialIceBase`. The full taxonomy of ICE based models is discussed in section 14.1

8.3 Genset

Genset models has the following diagram:



It contains an ICE-based generator, an ideal gear to adapt speed, and an electric drive (acting as a generator). Ice and Drive are map-based.

The library provides several ICE and drive models; in Genset It has been chosen to use the IceT01 model for ICE, and OneFlange for the drive, acting as generator.

The user data window has to tabs. Whose appearance is as follows:

OMEdit - Parameters - EHPTlib.MapBased.GenSet
?
X

Parameters

General
Map related parameters
Modifiers

Class

Path: EHPTlib.MapBased.GenSet
Comment: GenSet GMS+GEN+SEngine

Parameters

gsRatio
1
IdealGear speed reduction factor

throttlePerWerr
100/maxGenW
internal throttle controller proportional gain

uDcNom
100
nominal DC voltage (only order of magnitude needs to be right)

mapsOnFile
☒

constOptimalSpeed
100
rad/s

Optimal speed when mapsOnfile=false

ICE parameters

jIce
0.1
kg.m2

ICE moment of inertia

wIceStart
167
rad/s

constFuelConsumption
200
Fuel consumption in g/kWh when mapsOnfile=false

Generator parameters

jGen
0.1
kg.m2

Generator moment of inertia

maxGenW
1e6
rad/s

Max generator angular speed when not mapsOnFile

maxGenPow
100e3
W

Max mechanical power of the internal generator when not mapsOnFile

maxTau
200
N.m

Max torque between internal ICE and generator when not mapsOnFile

constGenEfficiency
0.85
Gen efficiency when mapsOnfile=false

OK
Cancel

OMEdit - Parameters - EHPTlib.MapBased.Genset

Parameters

General Map related parameters Modifiers

Parameters

mapsFileName File containing data maps

GMS parameters

optiSpeedName Name of the on-file specific consumption variable

osInFactor Factor before inputting pRef into map from txt file

osOutFactor Factor after reading optiSpeed from txt file

ICE parameters

specConsName Name of the on-file specific consumption variable

scTorqueFactor Torque multiplier for specific consumption map

scSpeedFactor Torque multiplier for specific consumption map

scConsFactor Output multiplier for specific consumption map

iceTauLimitName Name of the on-file max ICE torque matrix

tlIceTorqueFactor Torque multiplier for torque limit map

tlIceSpeedFactor Speed multiplier for torque limit map

Gen parameters

efficiencyName Name of the on-file generator efficiency matrix

eTorqueFactor Factor applied to input torque for efficiencies when they are from file

eSpeedFactor Factor applied to input speed for efficiencies when they are from file

genTauLimitName Name of the on-file max ICE torque matrix

tlGenTorqueFactor Torque multiplier for torque limit map

tlGenSpeedFactor Speed multiplier for torque limit map

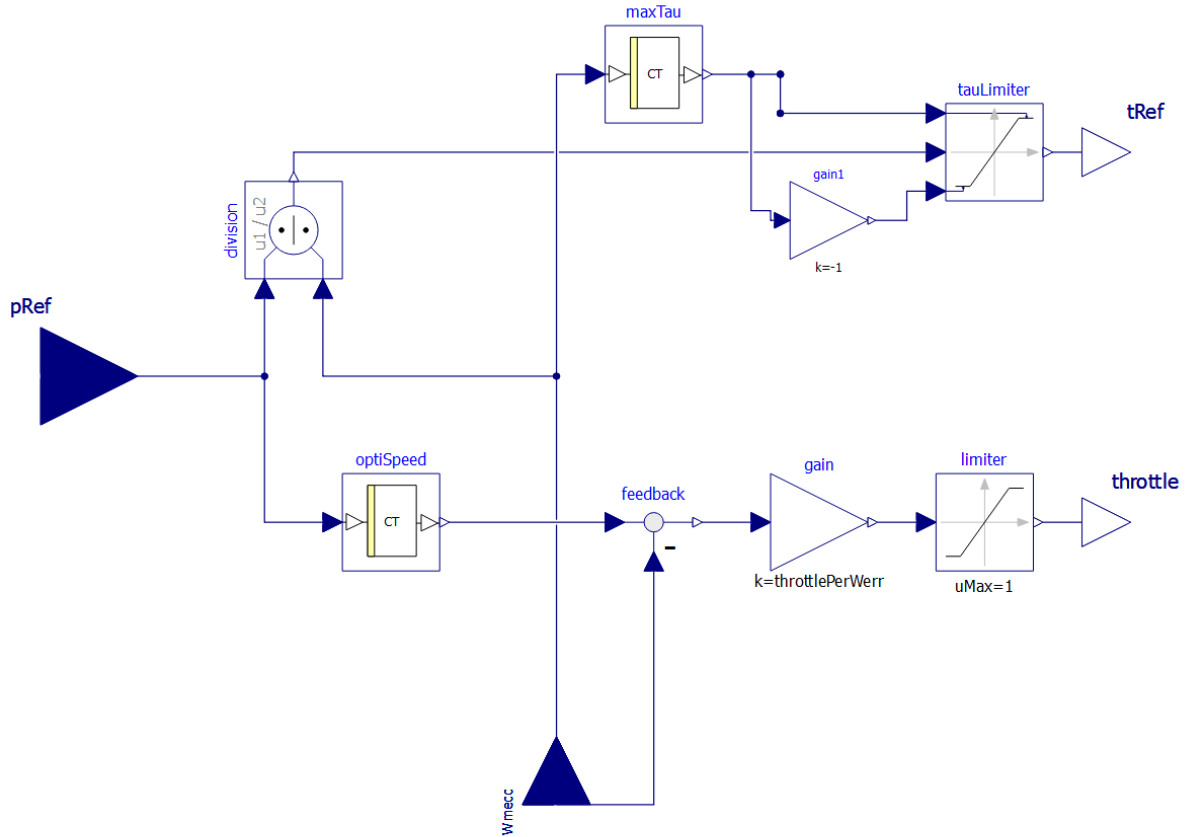
OK Cancel

Some fields are automatically greyed-out when they become inactive and vice-versa.

Explanation of all these parameters is given in the info-wondow of Genset model.

8.3.1 GMS

GMS has the following diagram:

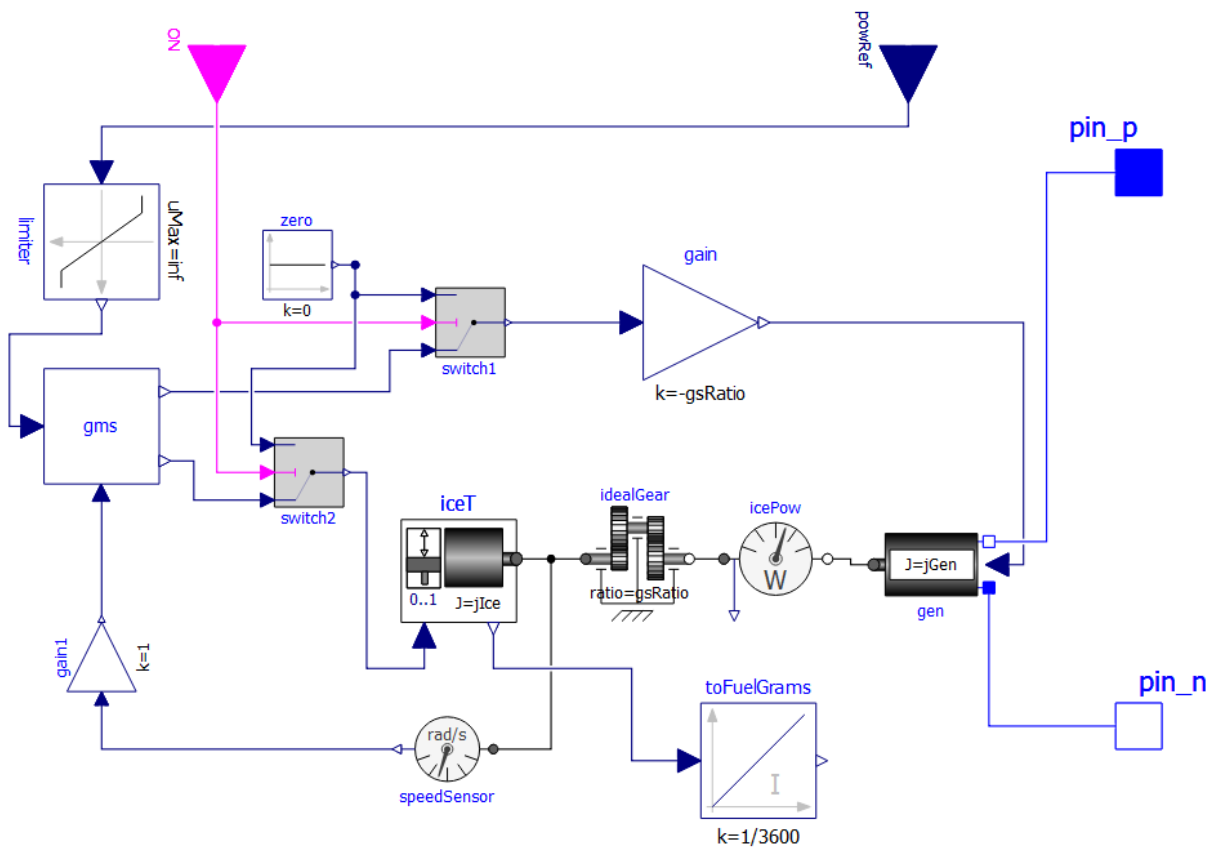


From the input power, it determines its optimal share into its two factors torque and speed, considering also the allowable torque limits. Data from matrices, either from a file or not are read through the CombiTable1Factor (see sect. 9.1)

8.4 GensetOO

It is similar to Genset, but allows ON-OFF operation of the engine: the engine is switched OFF when the input boolean signal requires it; when OFF, it does not generate torque nor power, nor consume fuel.

The diagram is as follows:



The library provides several ICE and drive models; in GensetOO, as in Genset, it has been chosen to use the IceT model for ICE, and OneFlange for the drive, acting as generator.

The user data window is identical to that of Genset.

9 Map-based support models

9.1 CombiTable1Factor and CombiTable2Factor models

We've said several times that, to give our models extensive description of the devices they simulate, we often resort to txt files containing matrices. These can tell for instance the fuel consumption as a function of rotational speed and torque, maximum torque as a function of speed, etc.

When taking data from a txt file we would like to be somehow free in choosing how to write them.

Consider for instance the following matrices:

```
# First column: speed (rad/s), 2nd column: torque (Nm)
double minMaxGenTau (9,3)
0      -0.0  0
70     -0.0  0
80     -380. 380
100    -620. 620
120    -800. 800
170    -800. 800
220    -670. 670
230    -650. 650
240    -570. 570
```

```

# motEffTable
# efficiency - rows:speeds pu, columns:torques pu
double motEffTable (6,6)
    0.00  0.00  0.25  0.50  0.75  1.00
    0.00  0.75  0.80  0.81  0.82  0.83
    0.25  0.76  0.81  0.82  0.83  0.84
    0.50  0.77  0.82  0.83  0.84  0.85
    0.75  0.78  0.83  0.84  0.85  0.87
    1.00  0.80  0.84  0.85  0.86  0.88

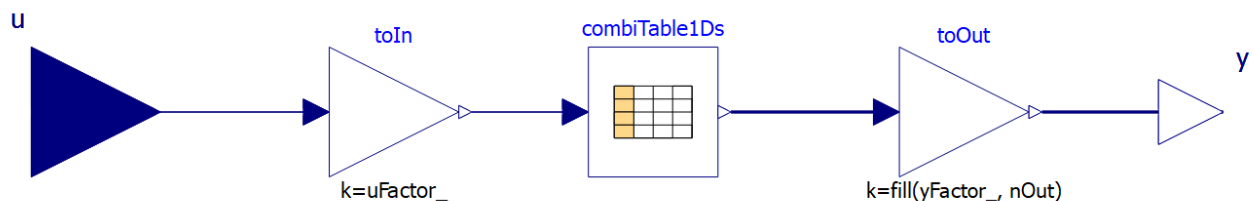
# specificCons (=specific consumption)
# First row: (from column 2) speed (rad/s), 1st column (from row 2): torque; body: spec.
consumption.
double specificCons(10,9)
    0      0      80      100      120      170      220      230      240
    0      999     999     999     999     999     999     999     999
    100     280     280     280     280     280     280     280     280
    200     215     213     219     225     231     252     255     260
    300     210     208     208     207     214     228     230     237
    400     213     204     204     199     204     218     218     222
    500     213     205     205     198     202     213     214     215
    600     213     213     213     198     205     212     213     214
    700     213     213     213     198     294     210     211     214
    800     213     213     213     213     213     210     211     214

# optispeed
# First column Power (W); second column optimal speed (rpm)
double optiSpeed(9,2)
    0      800
    20000   850
    40000  1100
    60000  1250
    80000  1280
    100000 1340
    120000 1400
    140000 1650
    160000 2130

```

In minMaxGenTau we wrote speeds in rad/s, in SpecificCons in rpm. In motEffTable we wrote both speeds and torques in per unit (pu), i.e. normalised to some factor.

To allow this flexibility, in EHPTlib, all matrix inputs and outputs are multiplied by pre-defined factors. This is done defining modified versions of MSL's CombiTable1Ds and CombiTable2Ds, called CombiTable1Factor and CombiTable2Factor, whose diagrams are as follows:



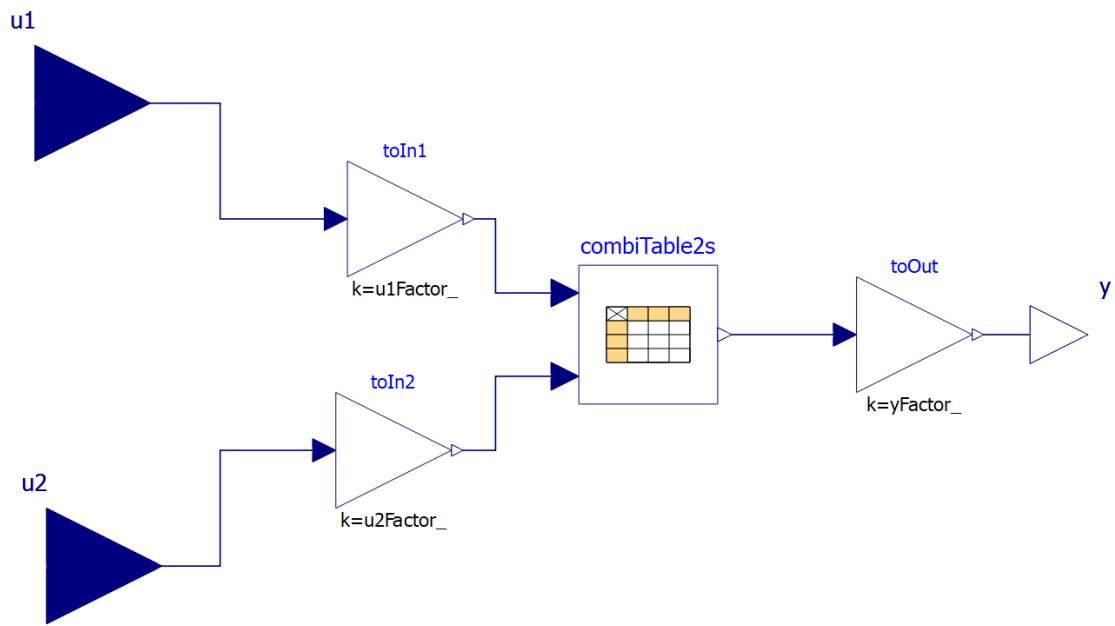


Figure 56: CombiTable1Factor (above) and CombiTable2Factor (below).

Using them:

- `minMaxGenTau` is read with factors all being one (data in table are already from the SI)
- `motEffTable` is read using for speed $\text{factor} = 1/\text{nomSpeed}$, where `nomSpeed` is the speed, in rad/s, which should correspond to 1 in the table, and for torque $\text{factor} = 1/\text{nomTorque}$, where `nomTorque` is the torque which should correspond to 1 in the table; if the needed output is efficiency, as in OneFlange models, the output factor is 1.
- `specificCons` is read using all unity factors
- `optiSpeed` is read using as input factor $60/(2*\pi)$

There can be a more advanced usage of these factors.

Imagine that we have a consumption map for a 100 kW ICE, and we want instead to use a larger engine that has twice the torque, the same speed range, and consumption reduced by 20%, and we know that the consumption shape is the same for these two engines, if normalised for speed, torque and maximum consumption. In other words, using appropriate input and output scales the shape is the same.

Instead of rewriting the map on the txt file, we can use factors to fool the software: it will use the map of a 100 kW ICE, applying, linearly scaled to the 200 kW one. With the data of this example, we can obtain this using the following factors:

Torque factor=0.5

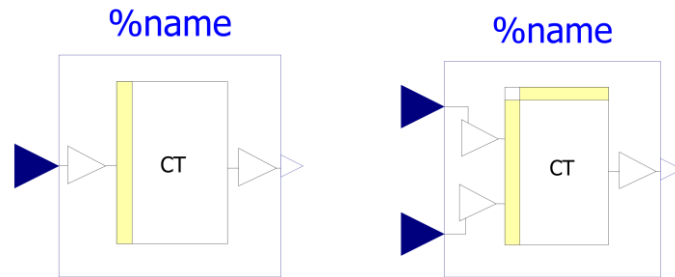
Speed factor=1

Output (specific consumption) factor=0.8.

A final note on `CombiTable2Factor` and `CombiTable2Factor`. They call the MSL `CombiTable1DS` and `CombiTable2DS`, which implement matrixes that can either be directly inputted or read from files.

When they are directly inputted they are typically very simple matrices. For this, in this case all factors are forced to be one. In other words, input factors are used only when reading matrices from a txt file.

Icons of CombiTable2Factor and CombiTable2Factor are drawn trying to show their internal architecture:



We showed them here, to help understanding diagrams introduced later, that make usage of these models.

9.2 EfficiencyCT block

This block (the last “CT” stand for CombiTable) having pathname `EHPTlib.SupportModels.MapBasedRelated.EfficiencyCT`, receives as input a power train angular seed and torque, and outputs the corresponding power computed as the product of them, diminished by the internal losses evaluated by means of an efficiency map. Usually efficiency maps are known for families of components, in a normalised way. Because of this, a normalised map approach is used here as well. The diagram is as follows:

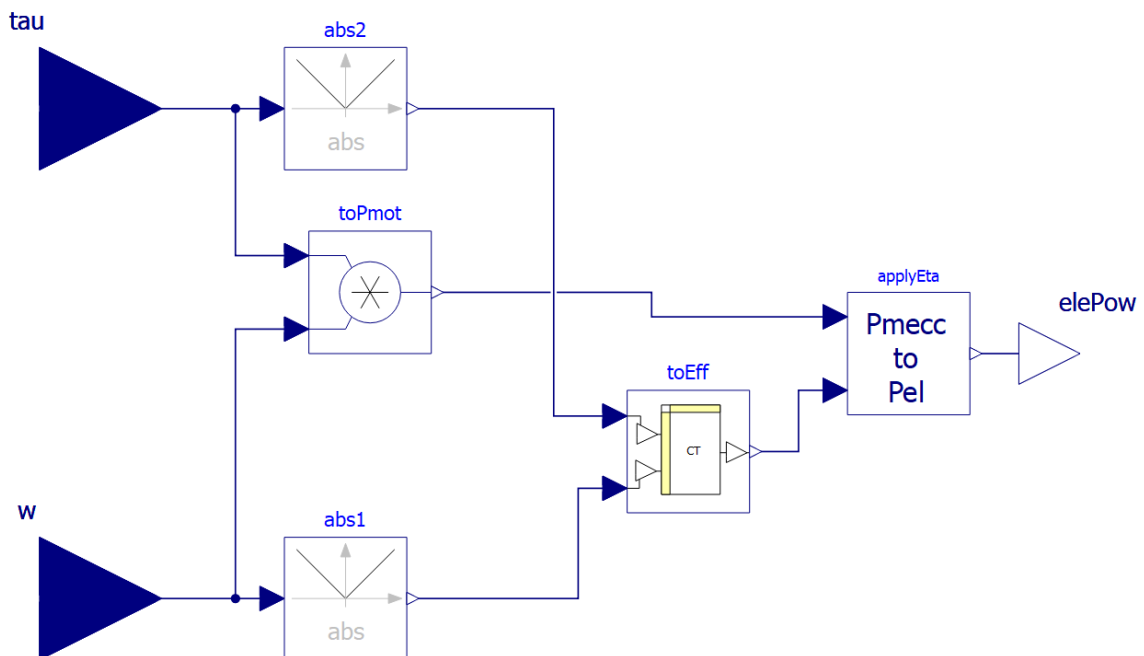


Figure 57 Diagram `EHPTlib.SupportModels.MapBasedRelated.EfficiencyCT` block.

The model uses for the efficiency table a CombiTable2D component. That table can be used passing the matrix on-line or reading it from a file. In the provided form, it is passed on-line.

The numerical data inserted as an example in the model correspond to the following efficiency map, not drawn from a real case, but reproducing a somewhat reasonable trend.

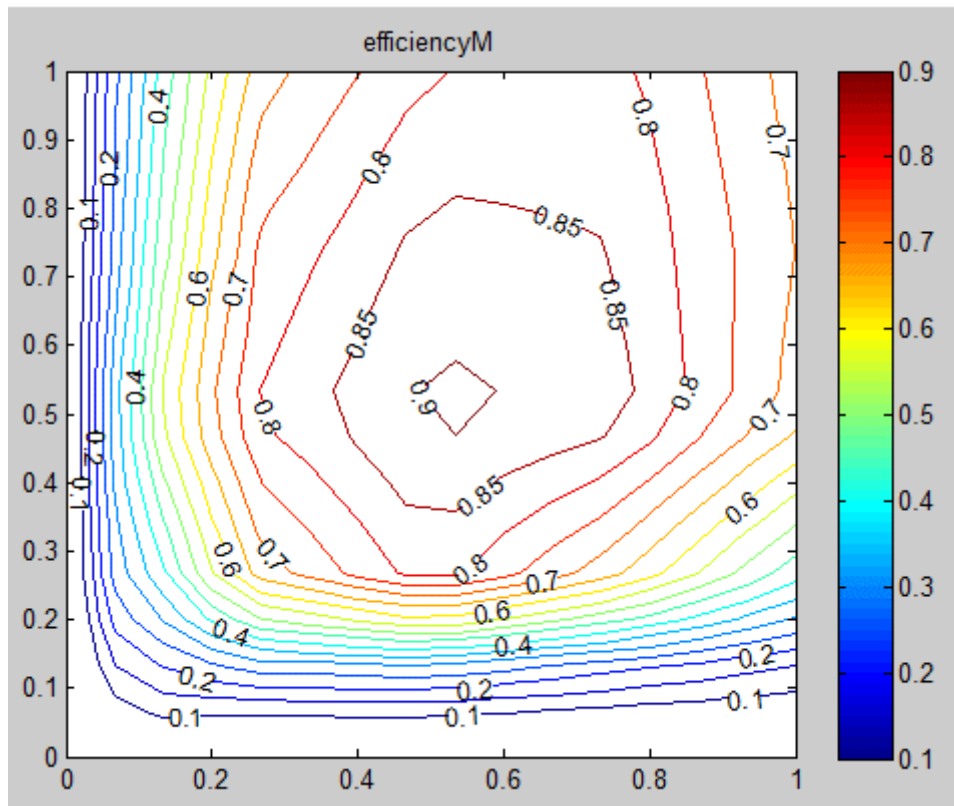
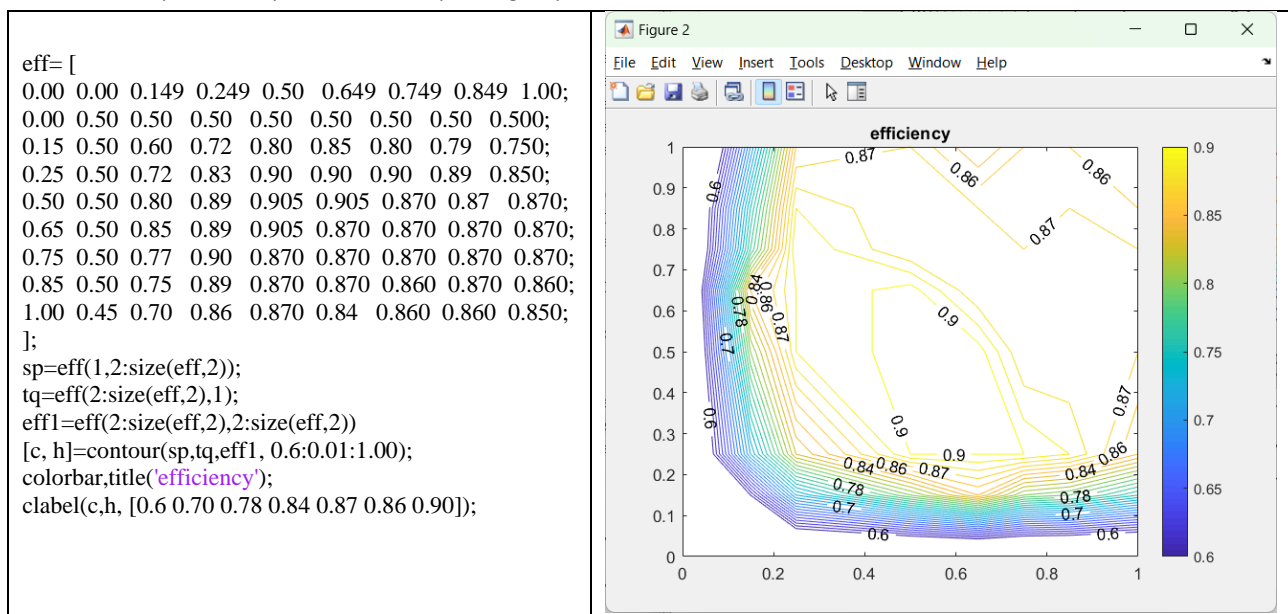


Figure 58. An example efficiency map

Drawing this kind of maps from numerical data is not a very easy task. If the user uses Scilab or Matlab, we can do this taking advantage of the “contour()” function.

Below, an example of a script and the corresponding map. The initial matrix is written un the Modelica format for Table2D matrices.



The use of efficiency maps, though being very common and useful to graphically analyse things, has important drawbacks. The biggest difficulty is that on the horizontal and vertical axes (i.e. axes $\Omega=0$ and $T=0$) actual efficiencies are zero, since mechanical power is zero, but losses are not zero. To use efficiency maps in EfficiencyCT efficiencies on these axes cannot be set exactly to zero, to avoid division by zero, but to small values, e.g. 0.01. The map in figure 58 was drawn using exactly $\text{eff}=0.01$ on the axes.

When the efficiency map is read from a file, it has the appearance on the following two examples:

```
# efficiency - row1 (u2): speeds pu, column1 (u1) torques pu
#
double nEffTable (9,9)
0.00  0.00  0.15  0.25  0.50  0.65  0.75  0.85  1.000
0.00  0.50  0.50  0.50  0.50  0.50  0.50  0.50  0.500
0.15  0.50  0.60  0.75  0.80  0.85  0.80  0.79  0.780
0.25  0.50  0.75  0.89  0.905 0.905 0.90  0.89  0.875
0.50  0.50  0.80  0.905 0.905 0.905 0.870 0.87  0.870
0.65  0.50  0.85  0.905 0.905 0.870 0.870 0.870 0.870
0.75  0.50  0.80  0.90  0.870 0.870 0.870 0.870 0.870
0.85  0.50  0.79  0.89  0.870 0.870 0.870 0.870 0.870
1.00  0.50  0.78  0.875 0.870 0.84  0.870 0.870 0.870

# efficiency - row1 (u2): speeds (rpm), column1 (u1) torques (Nm)
#
double effTable (9,9)
00  0300  0500  1000  1300  1500  1700  2000  3000
00  0.60  0.50  0.50  0.50  0.50  0.50  0.50  0.500
105 0.60  0.60  0.75  0.80  0.85  0.80  0.79  0.780
175 0.60  0.75  0.89  0.905 0.905 0.90  0.89  0.875
350 0.60  0.80  0.905 0.905 0.905 0.870 0.87  0.870
455 0.60  0.85  0.905 0.905 0.870 0.870 0.870 0.870
525 0.60  0.80  0.90  0.870 0.870 0.870 0.870 0.870
595 0.60  0.79  0.89  0.870 0.870 0.870 0.870 0.870
700 0.60  0.78  0.875 0.870 0.84  0.870 0.870 0.870
```

In the case of `nEffTable`, data on the first row and column are intended to be normalised: the block receives as input actual torque and speed, that are normalised by means $1/\tau_{\text{Max}}$ and $1/w_{\text{Max}}$ of the gains shown in figure 57.

In the case of `effTable`, data on first row and column are intended to be in rpm and Nm respectively. Values of τ_{Max} and w_{Max} must be chosen accordingly, e.g. $\tau_{\text{Max}}=1$ and $w_{\text{Max}}=60/(2\pi)$ (in this way the speed read by the physical part of the model, which is in rad/s) is converted in rpm before being entered in the table first row).

Another way to introduce efficiencies in map-based models, is to introduce formulas to compute efficiencies. This is discussed in the next section.

9.2.1 Proposed activity

The reader is prompted to modify the model so that the `effTable_` data are read from a file. The file name is to be passed as a parameter to the block.

9.3 EfficiencyLF block

This block (the last “LF” stands for Loss Formula) receives as input a power train angular speed and torque. In this case efficiency is computed through losses (fig 59).

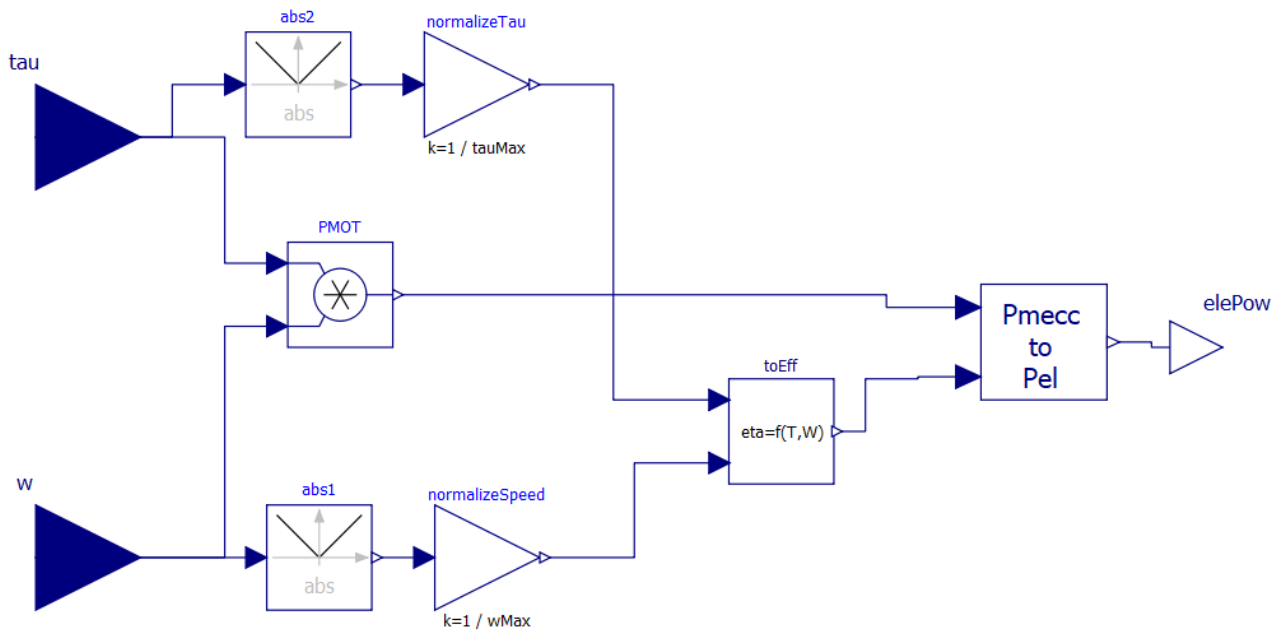


Figure 59. Diagram of the EfficiencyLF block.

Here, inside toEff, losses are computed through a loss formula (not a map, hence the model’s name); toEff has the following code:

```
block ComputeEffFromPu "adds drive losses function of W/Wmax and T/Tmax"
  parameter Real A = 0.006 "fixed losses";
  parameter Real bT = 0.05 "torque losses coefficient";
  parameter Real bW = 0.02 "speed losses coefficient";
  parameter Real bP = 0.05 "power losses coefficient";
  Real lossesPu "losses pu of Wmax*Tmax";
  Modelica.Blocks.Interfaces.RealInput Wu "angular speed, pu of Wmax" annotation (
    Placement(transformation(extent = {{-138, -80}, {-98, -40}})));
  Modelica.Blocks.Interfaces.RealOutput y "computed efficiency" annotation (
    Placement(transformation(extent = {{96, -10}, {116, 10}})));
  Modelica.Blocks.Interfaces.RealInput Tu "Torque pu of Tmax" annotation (
    Placement(transformation(extent = {{-138, 40}, {-98, 80}})));
equation
  lossesPu = A + bT * Tu ^ 2 + bW * Wu ^ 2 + bP * (Tu * Wu) ^ 2;
  y = Tu * Wu / (Tu * Wu + lossesPu);
end ComputeEffFromPu;
```

Using the default parameters, the obtained efficiency maps, respectively for traction and braking are as follows: (those having matlab can use the function efficiency.m reported in Appendix using the row efficiency(0.006, 0.05, 0.02, 0.05, [0.01 1], [0.01 1]));

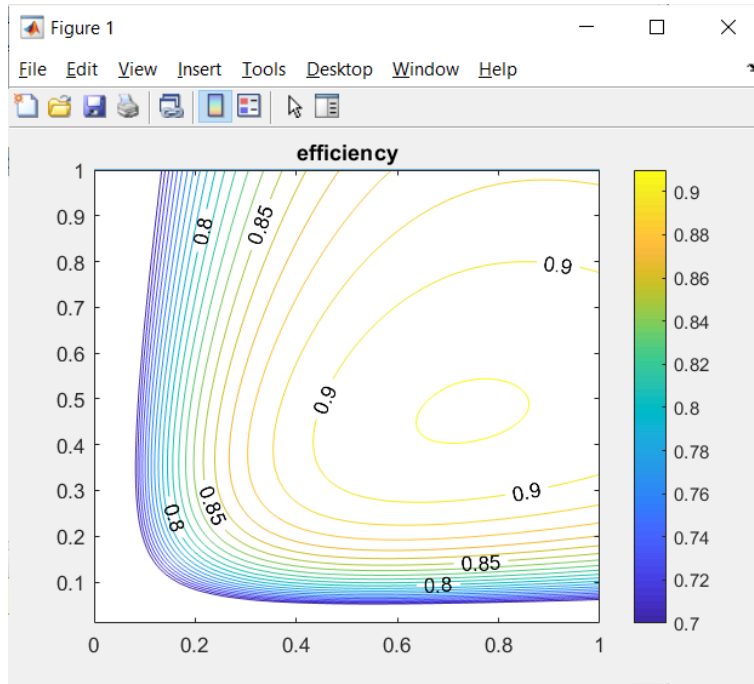


Figure 60. EfficiencyMap with the proposed loss formula.

9.4 LimTorqueFV block

This block receives as input a torque request and measured speed; outputs the limited torque, i.e. the torque request if it is within limits, otherwise the maximum (or minimum) available torque (“FV” in the name stands for “Fixed Values”; compare with next model LimTorqueCT, in the next section). The limiting torque is computed by limiting absolute value of power and torque as per the region shown in the model’s icon, which is as follows:

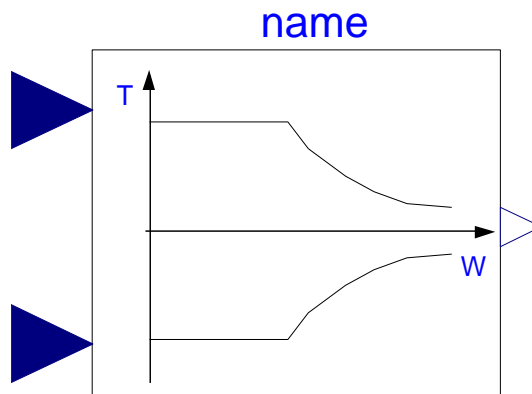


Figure 61. Icon of the LimTorqueFV.

This is a typical simplified operating region of a drive train: it contains a maximum torque limit, a maximum power (the two hyperbolas), a maximum speed.

9.4.1 Proposed activity 1

The model `tauLim` has two significant limitations:

- It considers only symmetrical curves around the horizontal axis, i.e. equal values for positive and negative limiting torques.
- It does not consider automatic dropping of regions having negative efficiencies.

A more realistic map will have the shape shown in figure 62.

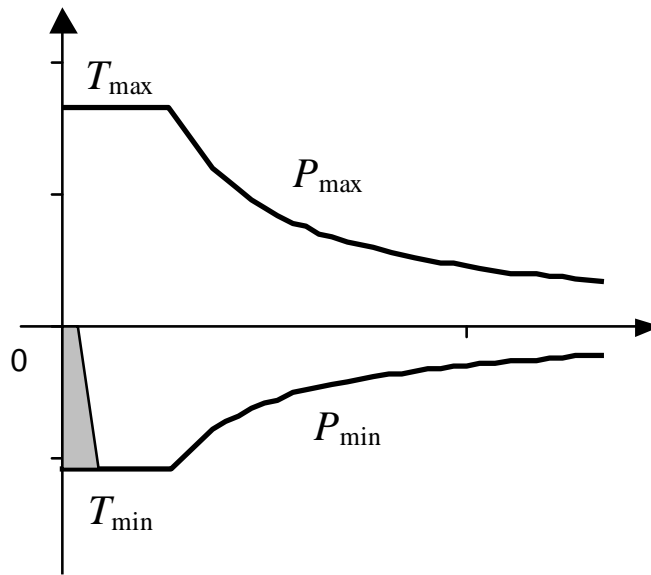


Figure 62. Improved torque limitation region.

In this figure not necessarily $|P_{\max}| = |P_{\min}|$ nor $|T_{\max}| = |T_{\min}|$

Moreover a “prohibited region”, greyed, is present, where it is not convenient to operate the electric drive. In fact, in this region we can have negative efficiency. This is due to the fact that the mechanical power is lower than the electric losses to exchange that power. Instead of using that region, then, it is convenient to mechanical braking.

The code for `TorqueFV` included in the `EHPTlib` is very simple:

```
block LimTorqueFV
  "Defines torque limits based on fixed max values of torque and power"
  Modelica.Blocks.Interfaces.RealInput w annotation (
    Placement(transformation(extent = {{-140, -20}, {-100, 20}})), iconTransformation(extent = {{-140, -20}, {-100, 20}}));
  Modelica.Blocks.Interfaces.RealOutput yH annotation (
    Placement(transformation(extent = {{100, 50}, {120, 70}})));
  parameter Modelica.Units.SI.Power powMax=50000
    "Maximum mechanical power";
  parameter Modelica.Units.SI.Torque tauMax=400 "Maximum torque ";
  parameter Modelica.Units.SI.AngularVelocity wMax= 1500 "Maximum speed";
  Integer state "=0 below base speed; =1 before wMax; =2 in w limit, =3 above wMax";
  //0 or 1 if tauMax or powMax is delivered; =2 or 3 if w>wMax
protected
  parameter Real alpha = 0.10 "fraction of wMax over which the torque is to be brought to zero";
public
  Modelica.Blocks.Interfaces.RealOutput yL annotation (
    Placement(transformation(extent = {{100, -70}, {120, -50}})));
algorithm
  if w < powMax / tauMax then
    state := 0;
    yH := tauMax;
  else
    state := 1;
    yH := powMax / w;
  end if;
  //over wMax the torque max is to be rapidly brought to zero
  if w > wMax then
    if w < (1 + alpha) * wMax then
      state := 2;
      yH := powMax / wMax * (1 - (w - wMax) / (alpha * wMax));
```

```

else
    state := 3;
    yH := 0;
end if;
end if;
yL := -yH;
end LimTorqueFV;

```

The reader is prompted to enhance this code, to give it more flexibility, allowing to generate a region of the type show in figure 62.

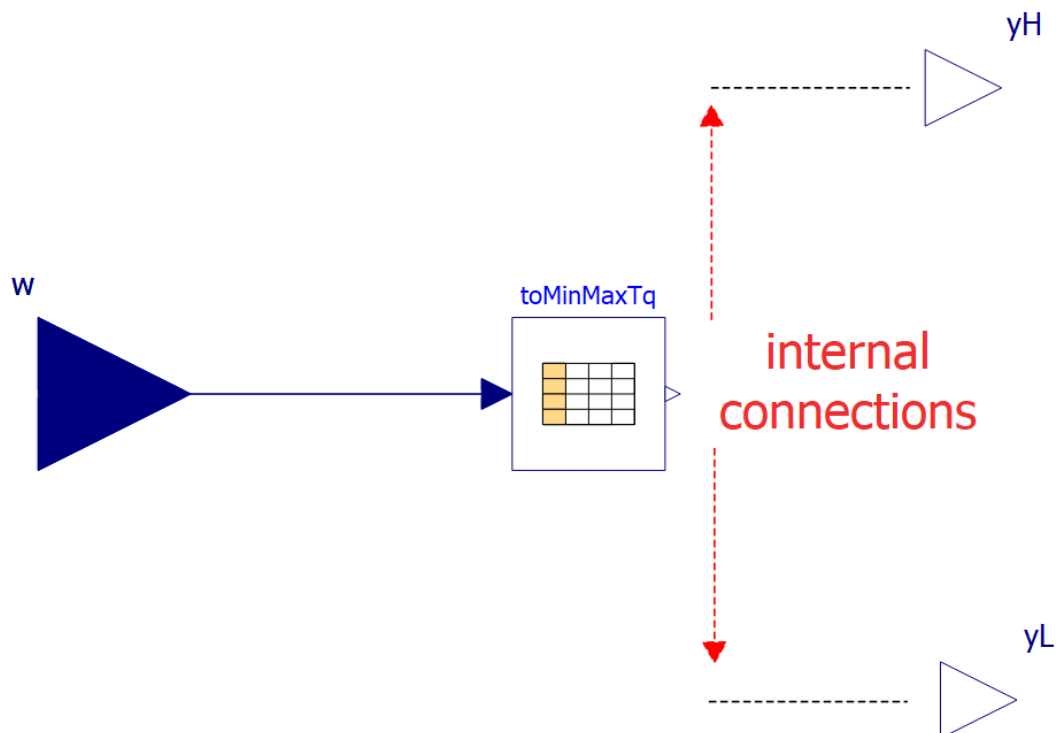
9.4.2 Proposed activity 2

This model is written for positive speeds. The user is prompted to modify it in such a way that it can operate also with negative speeds. The operating region therefore becomes four-quadrant. In case this activity is done after Proposed activity 1, the improvements there made can be included also in activity 2.

9.5 LimTorque (includes LimTorqueFV behaviour)

A much larger flexibility in defining torque limits is to allow them to be introduced through a CombiTable, which in turn can receive data from a file. This is done in the block LimTorque, which also allows input torques to be inputted normalised or with SI input quantities.

This block has the following diagram:



This block **includes** the behaviour of LimTorqueFV: if the parameter `limitsOnFile=true`, limits are taken from maps on file, otherwise they are fixed maximum torque and power, as per LimTorqueFV. Below the input mask in the two cases. You can see that the unneeded parameters are automatically greyed.

OMEdit - Parameters - EHPTlib.SupportModels.MapBasedRelated.LimTorque

Parameters

General Modifiers

Class

Path: EHPTlib.SupportModels.MapBasedRelated.LimTorque

Comment: Defines torque limits based on values from a CombiTable

Parameters

limitsOnFile	false		= true, if max torque is taken from a txt file
powMax	50000	W	Maximum mechanical power
tauMax	400	N.m	Maximum torque
wMax	1500	rad/s	Maximum speed
limitsFileName	"noName"	...	File where matrix is stored
limitsTableName	"noName"		Name of the on-file maximum torque as a function of speed

OK Cancel

OMEdit - Parameters - EHPTlib.SupportModels.MapBasedRelated.LimTorque

Parameters

General Modifiers

Class

Path: EHPTlib.SupportModels.MapBasedRelated.LimTorque

Comment: Defines torque limits based on values from a CombiTable

Parameters

limitsOnFile	true		= true, if max torque is taken from a txt file
powMax	50000	W	Maximum mechanical power
tauMax	400	N.m	Maximum torque
wMax	1500	rad/s	Maximum speed
limitsFileName	"noName"	...	File where matrix is stored
limitsTableName	"noName"		Name of the on-file maximum torque as a function of speed

OK Cancel

9.6 ConstPg model

This model has the purpose to interface map-based mechanical models with the corresponding electric circuit. Its diagram is as follows:

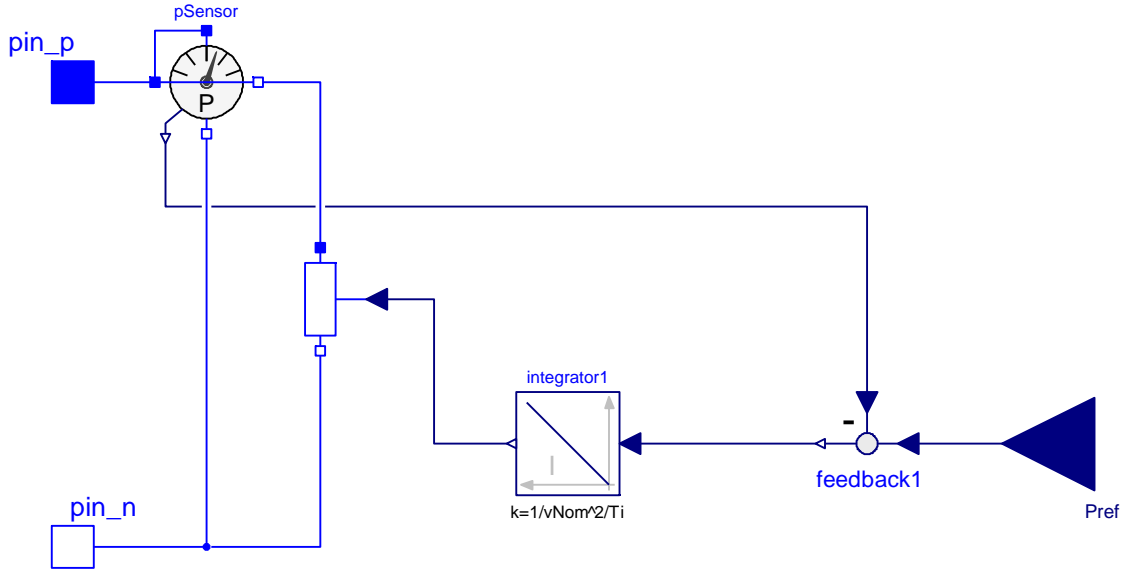


Figure 63. Diagram of EHPTlib.SupportModels.MapBasedRelated.ConstPg.

It is based on a variable conductor, whose value is controlled so that the absorbed power equals the requested power, using a simple integral open.-loop controller.

It is very important to note that when this is connected to a real DC circuit, not always a solution can be found.

Consider, for instance that it is connected to a DC circuit which has a thévenin equivalent with electromotive force E_{th} and inner resistance R_{th} . The situation is as shown in figure 64.

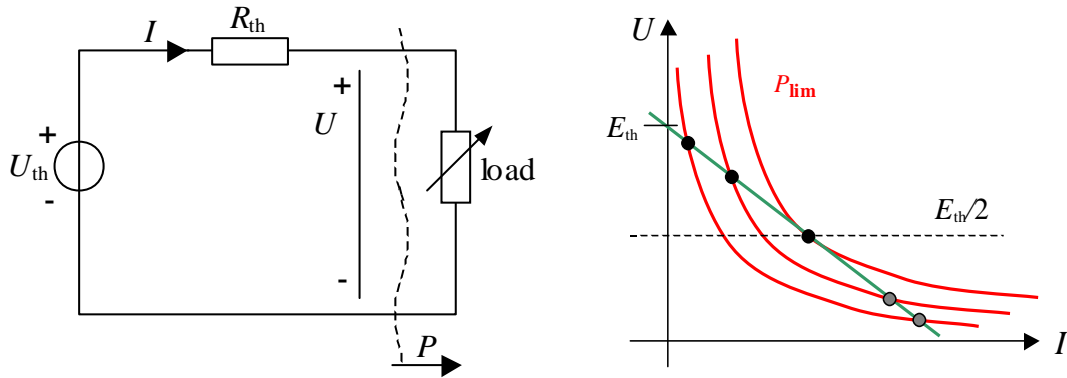


Figure 64. Duplicity of equilibrium point when a Thevenin DC source is loaded with a constant power load.

In the right part of the figure different load powers correspond to different hyperbolas. The green straight line correspond to the operating points compatible with the source DC circuit. It is apparent that beyond P_{lim} no intersection exists between red and green curves, i.e. that power cannot be delivered. When $P < P_{lim}$ there exist two possible operation points, of which usually only the one with the higher voltage is acceptable.

This means that this kind of interface requires some attention that the requested power is not too much, and that the system initial condition is such that the wanted point (usually the upper one, as I said) is found by the solver.

Note that it is $P_{lim} = E_{th}^2 / (4R_i)$.

9.6.1 *Proposed activities*

1. The proposed model has as parameter the integrator constant. The provided formula contains also the nominal conductor voltage v_{Nom} . Why?
2. The integrator should have an initial value for output. Try to envisage which could be best and enhance the model to include this initial condition. Maybe a reference power p_{Nom} could be added as a parameter.
3. Try to insert the model in simple DC circuits to see where and when an operating point is found and when it is not. First you can use a circuit composed by fixed voltage in series with a fixed resistance. Then you can use more complicated circuits.

10 EV-HEV simulation concluding remarks

Specialised programs exist to simulate vehicle power trains (e.g. [10]). Also rich commercial libraries are available.

This chapter had not the ambition of discussing industrial-grade simulators. However, through simple models, all based just on MSL, and progressive exercises, it has the purpose to allow the reader to get to know the main issues and opportunities of simulating EVs and HEVs using Modelica and Open source simulators such as OpenModelica. At least OpenModelica and proposed models (all of which can be run through OpenModelica) are completely open, so one can understand, and modify, everything, up to the tiniest detail.

Indeed, on several occasions I felt that commercial programs, although powerful and reliable, have the important drawback of not allowing to understand all the details of the inner models.

Not always the more complex models are the best ones. For preliminary analyses, and evaluations, simpler ones allow to understand more easily the basic phenomena.

I hope that the reader that has got here has understood a lot regarding EV and HEV simulation, as well as HEV control and energy management.

11 Testing individual models

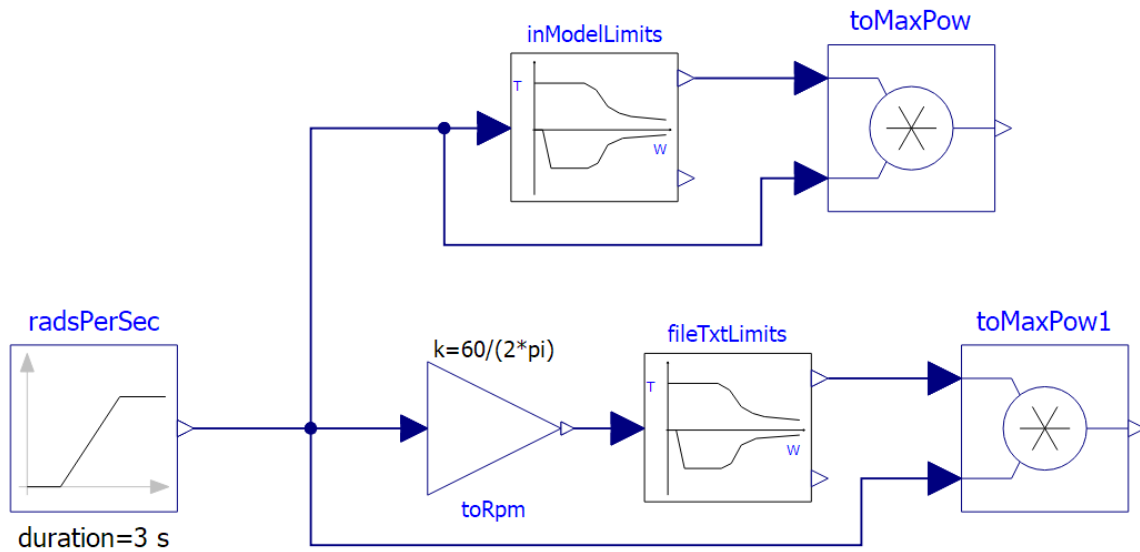
EHPTlib.MapBased contains folder **TestingModels** in which simple tests of some of the map-based components are proposed. All these contain some explanation in their information perspective.

Here we show just some example results

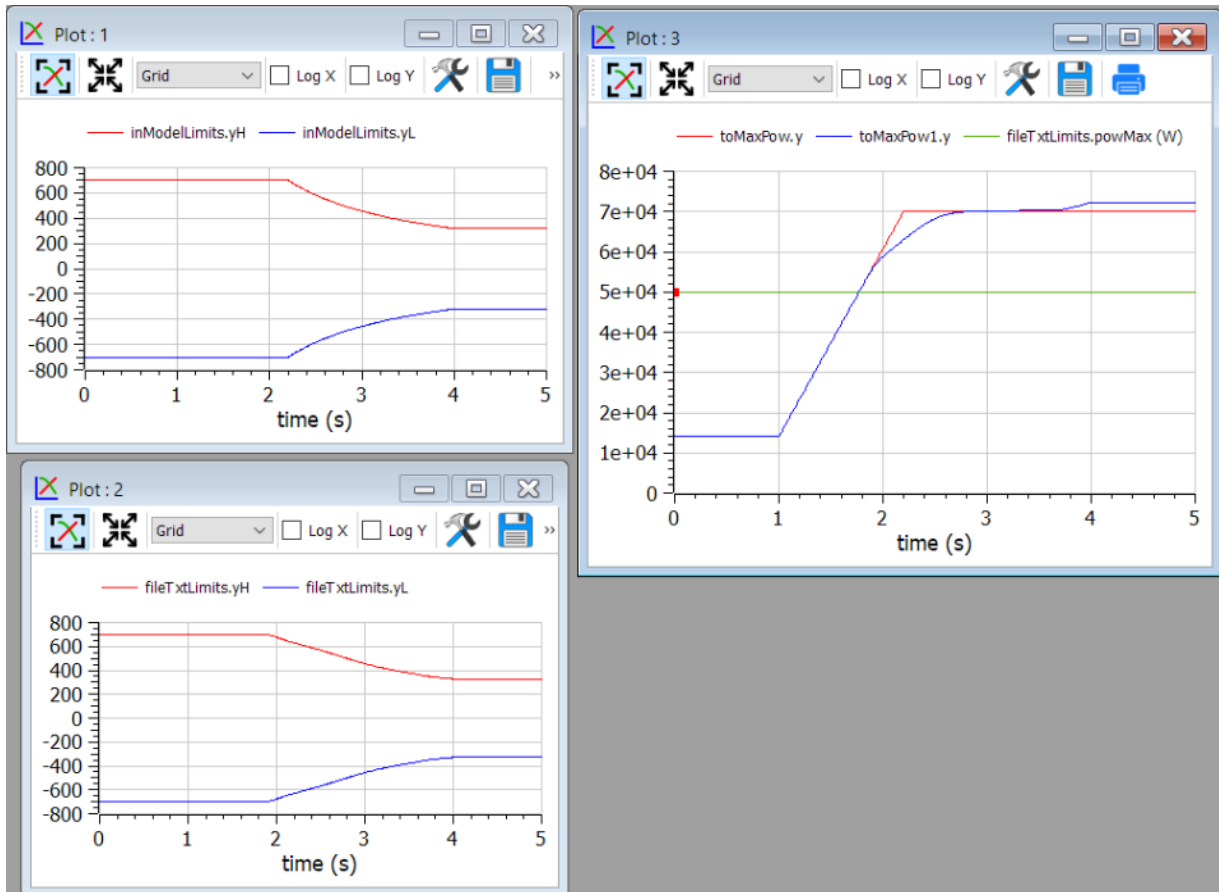
11.1 LimTorque

This tests just the usage of block LimTorque, which allows fixed limits to be defined or taken from a file. When they are from a file, the fixed values w_{Max} , pow_{Max} , tau_{Max} , have no effect on the output.

Diagram:

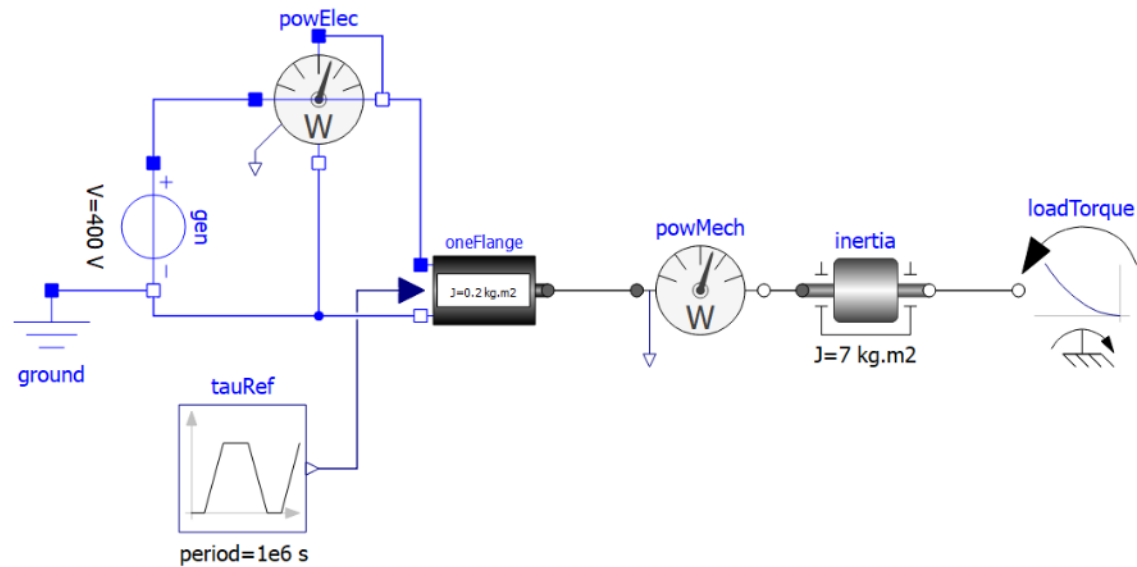


Some outputs:



11.2 OneFlange

We have several models, all with the same diagram, which is as follows:



The machine has a maximum power of 70 kW, and maximum torque of 400 Nm.

The individual models differ on how the data are inputted or generated:

- Efficiency can be given either in terms of a table or from a formula. The table can be defined online or taken from a file; in the latter case input data from a file can be subject multiplication by a factor (eTorqueFactor and eSpeedFactor respectively)
- Torque limits can be fixed or taken from a file; in the latter case input and output data from a file can be subject multiplication by a factor (tlSpeedFactor and tlTorqueFactor)

These options are selected through flags on the input mask. Flags are dynamic: when they are not meaningful they are de-activated.

The input mask (with the data for TestOneFlange1) is as follows:

OMEdit - Element Parameters - oneFlange in EHPTlib.MapBased.TestingModels.TestOneFlange.TestOneFlange1

Parameters

General | **Modifiers**

Name: oneFlange
Comment:

Class
Path: EHPTlib.MapBased.OneFlange
Comment: Simple map-based model of an electric drive

General parameters

J: 0.2 kg.m² Rotor's moment of inertia

uDcNom: 400 V Nominal DC voltage

limitsOnFile: ☐ = true, if torque and speed limits are taken from a txt file, otherwise they are tauMax and wMax

wMax: 200 rad/s Maximum speed (for efficiency and torque limitation when limitsOnFile=false)

powMax: 70 kW Maximum mechanical power (used for efficiency and torque limitation when limitsOnFile=false)

tauMax: 400 N.m Maximum torque when limits are not from an un-normalised file

Torque limitation related parameters

tTorqueFactor: 1 factor applied to input torque for limits they are from file

tSpeedFactor: $60/(2\pi)$ factor applied to input speed for limits when they are from file

limitsFileName: "noName" File where efficiency table matrix is stored

tauLimitsMapName: "noName" Name of the on-file lower torque limit

Efficiency related parameters

efficiencyFromTable: ☐ =true if efficiency if from a table (either online or from a file); otherwise use a the built-in loss formula

efMapOnFile: ☐ = true, if tables are taken from a txt file

efMapFileName: "noName" File where efficiency table matrix is stored

eTorqueFactor: 1 factor applied to input torque for efficiencies when they are from file

eSpeedFactor: $60/(2\pi)$ factor applied to input speed for efficiencies when they are from file

efTableName: "noName" Name of the on-file efficiency matrix

efTable[3, 3]: [0, 0, 1; 0, 1, 1; 1, 1, 1] rows: speeds; columns: torques; both PU of wMax and tauMax

Loss-formula parameters

A: 0.006 fixed losses

bT: 0.05 torque losses coefficient

OK Cancel

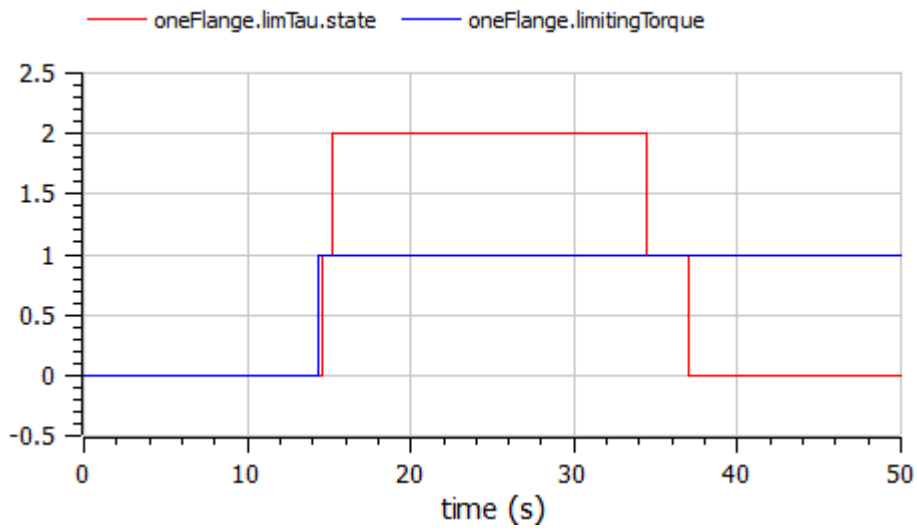
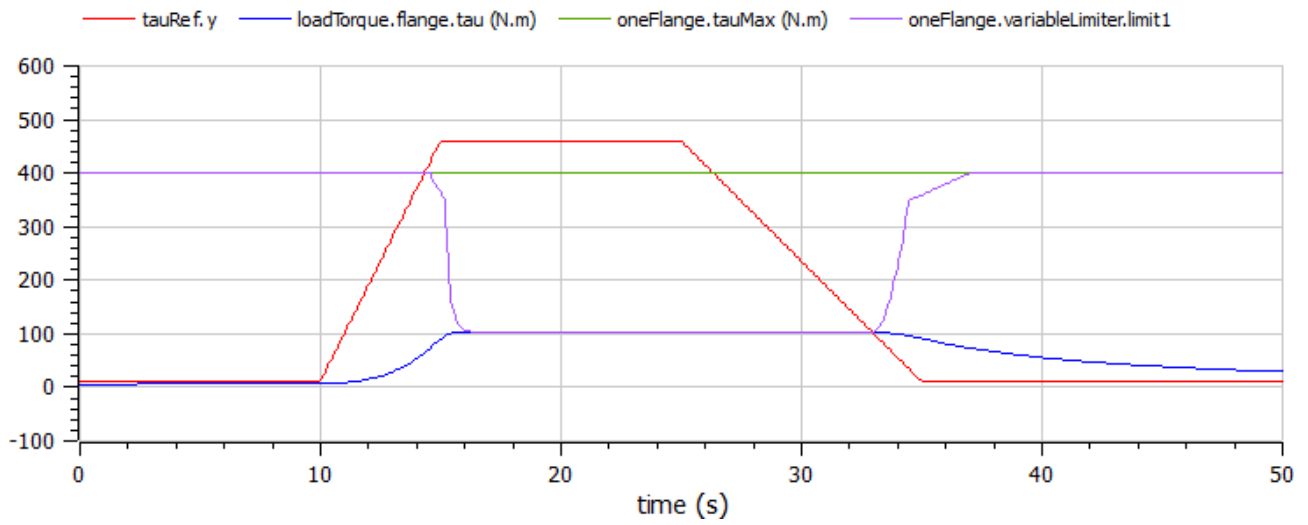
11.2.1 TestOneFlange1

It uses:

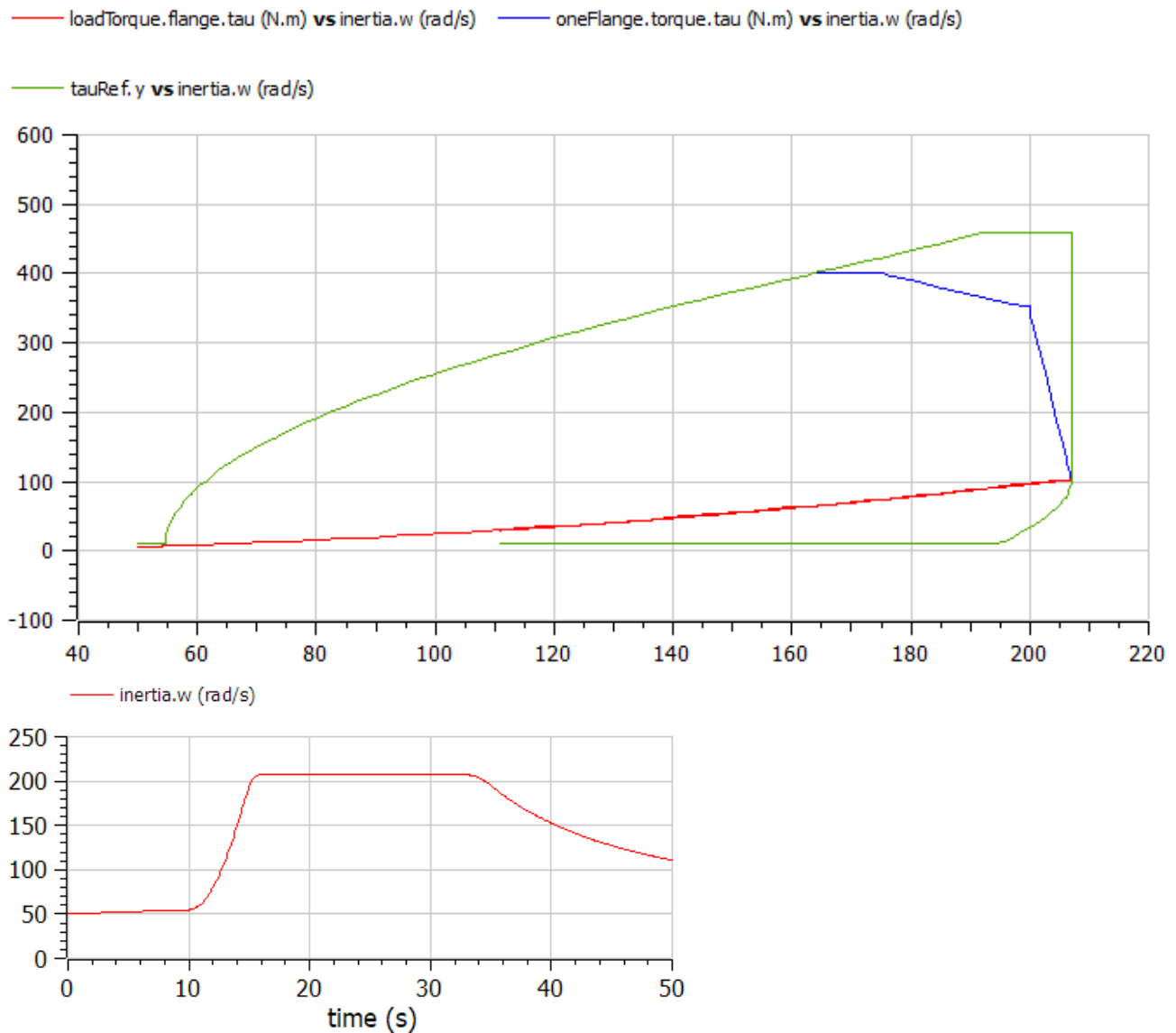
- Fixed torque limits
- Loss formula

Some plots follow, as proposed and explained the model info.

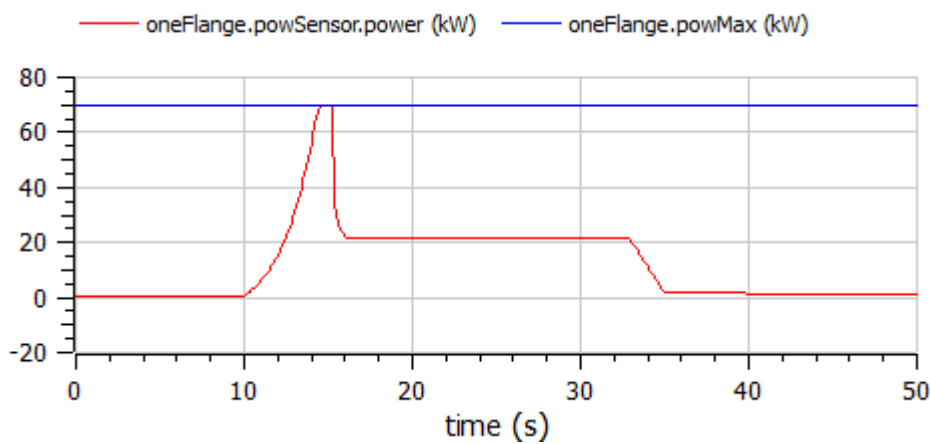
First suggested group of plots:

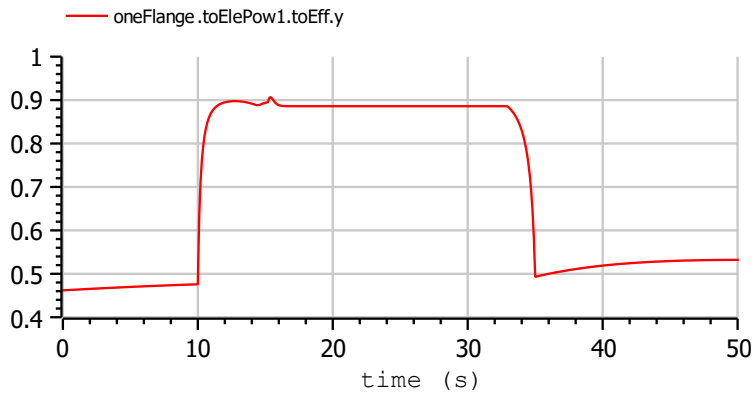


Second suggested group of plots:



Further Suggested Plots:





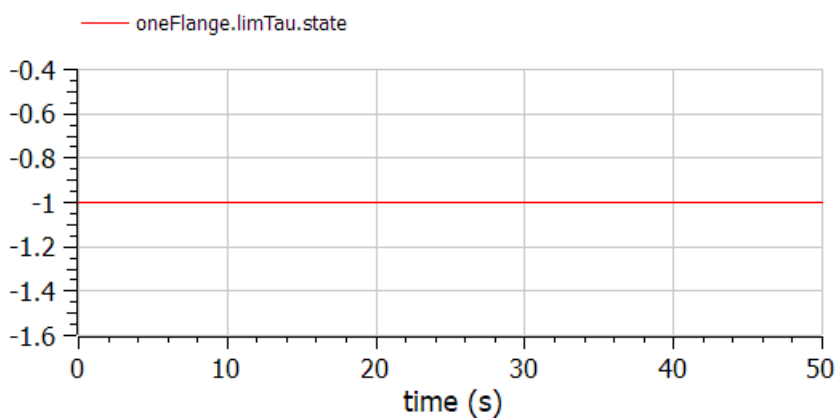
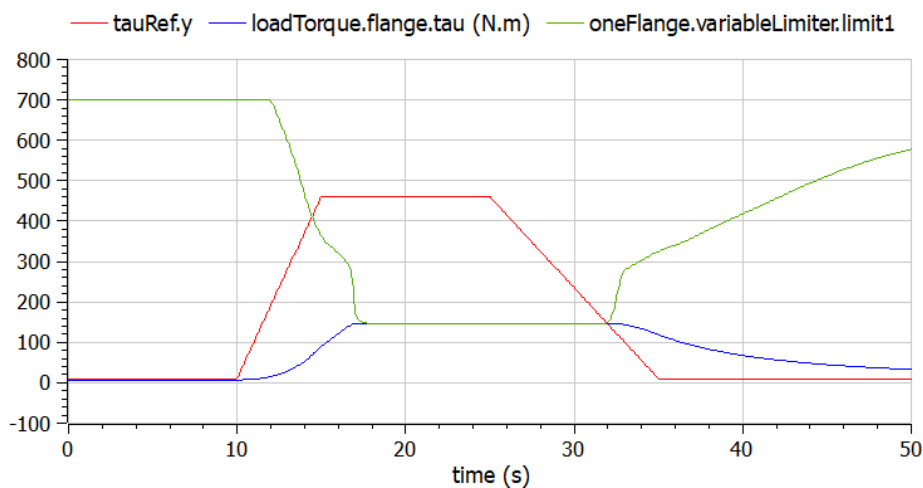
11.2.2 TestOneFlange2

It uses:

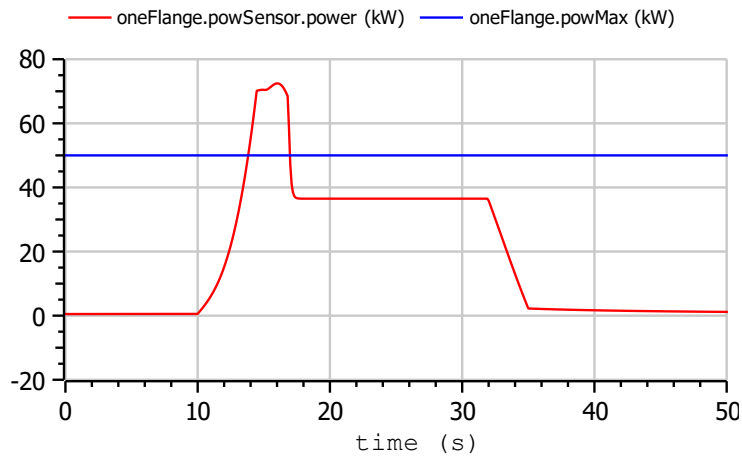
- - Torque limits from file (input in rpm, Nm)
- - Loss formula

Some plots follow, as proposed and the model info of TestOneFlange1.

First suggested group of plots:



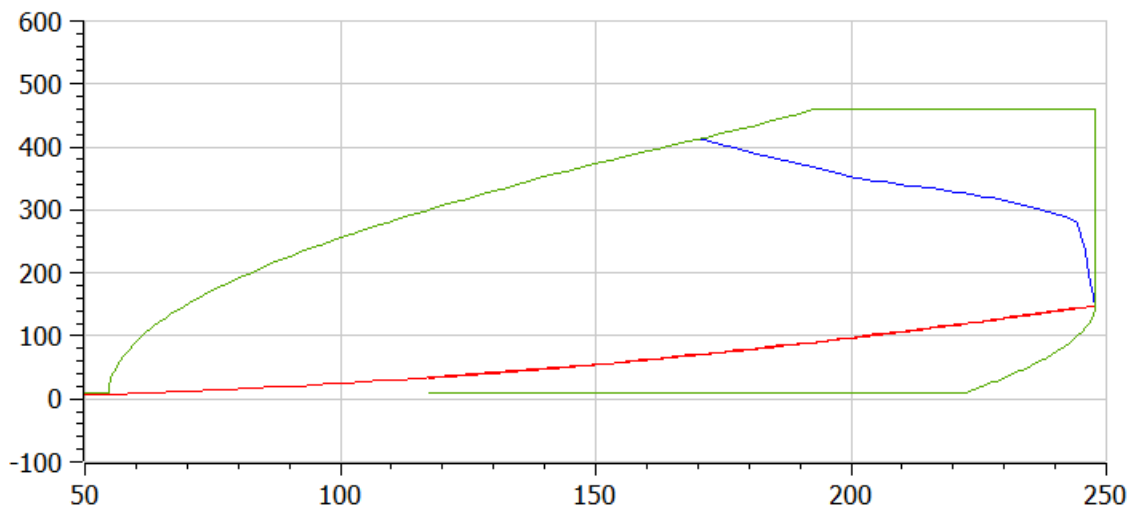
Second suggested group of plots (here we just reduced powMax from 70 to 50 kW to shoe that this value is effectless, since limits are from a file):



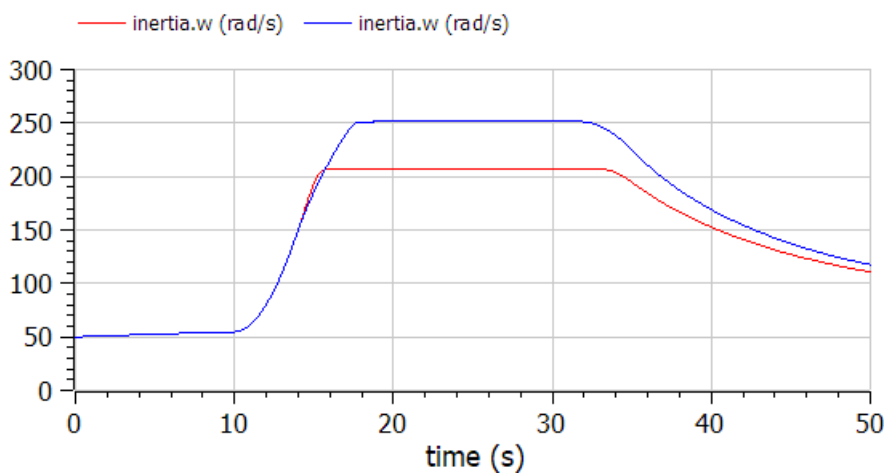
loadTorque.flange.tau (N.m) vs inertia.w (rad/s)

oneFlange.torque.tau (N.m) vs inertia.w (rad/s)

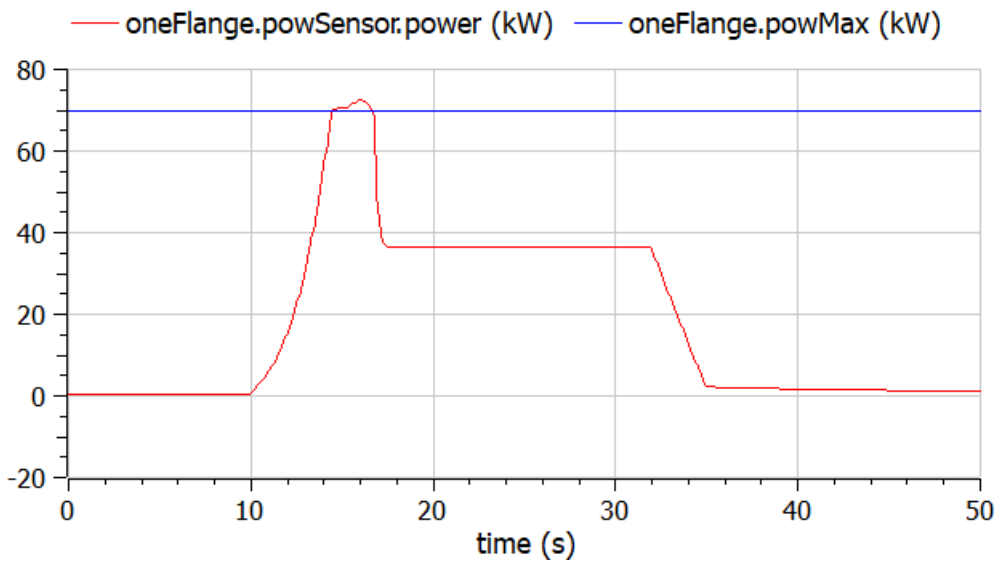
tauRef.y vs inertia.w (rad/s)



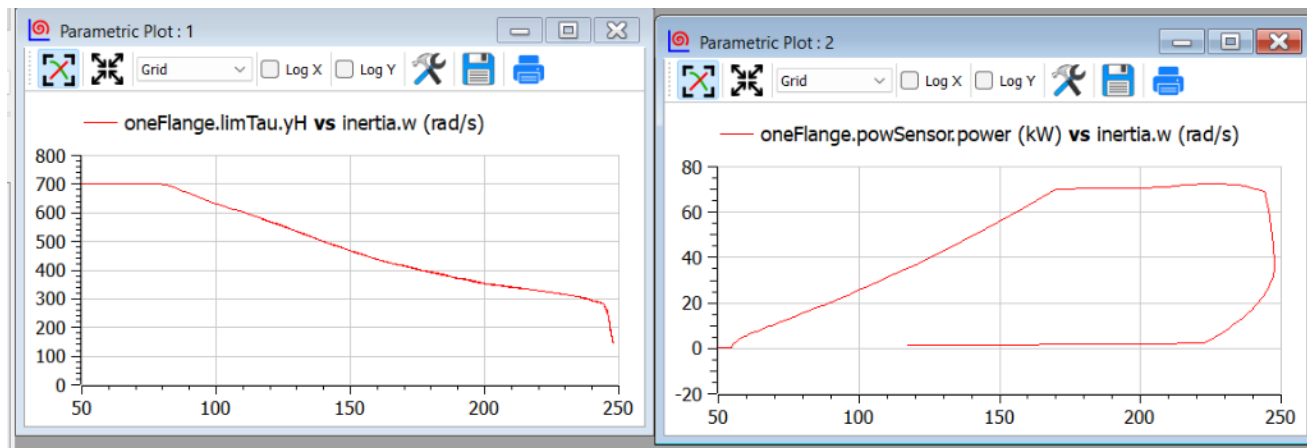
Here we compare speed between TestOneFlange1 (red) and TestOneFlange2 (blue):



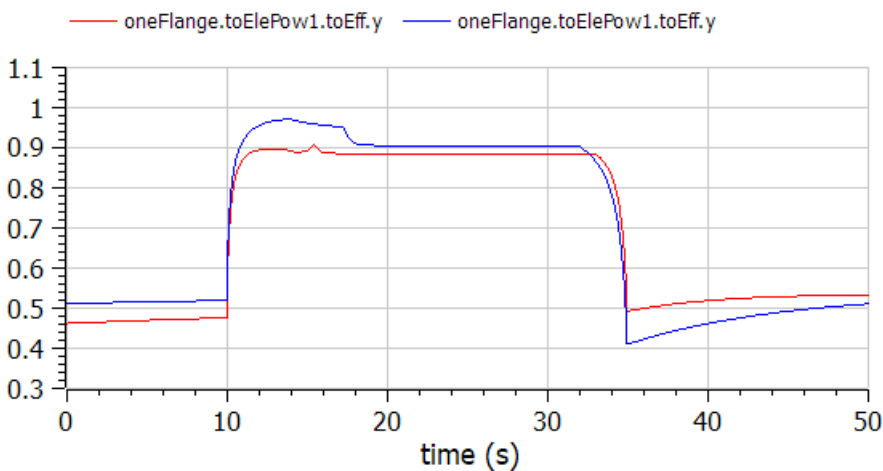
Further Suggested Plots:



Here powMax is not used for simulation; instead, torque and power limits are from a file, and give the following results:



Here we compare efficiency over time between TestOneFlange1 (red) and TestOneFlange2 (blue):



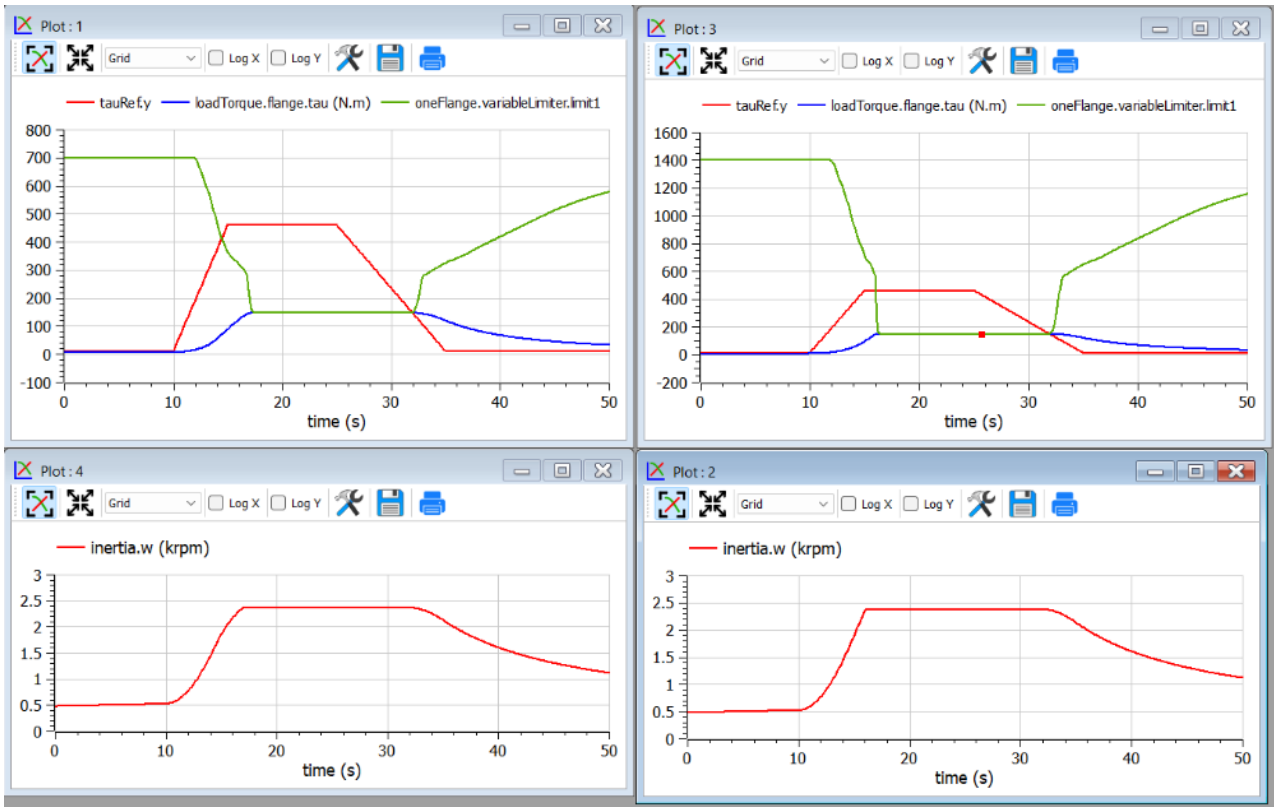
11.2.3 TestOneFlange3

It uses:

- Torque limits from file (input in rpm, Nm)

- Loss formula

Below a comparison between some of TestOneFlange2 (left) and TestOneFlange3 (right) is shown. It is apparent the doubling of limit1, when it is larger than the actual torque. In the central part of the plot the limited torque does not show significant limit change, since in that region the inputted torque limit has a nearly vertical fall, and therefore speed remains slightly above 2335 rpm, and torque is the one corresponding to this value from the load torque curve.

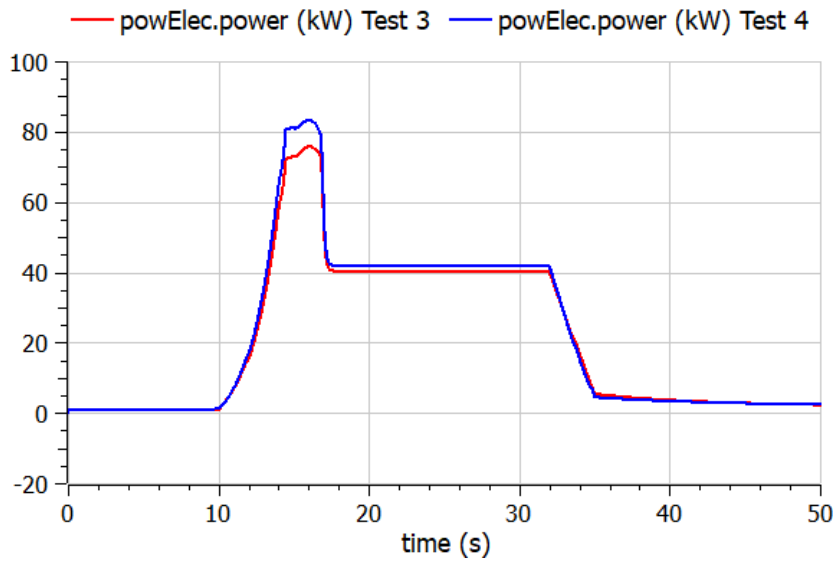


11.2.4 TestOneFlange4

It uses:

- around 70 kW, Max 700 Nm
- Torque limits from file
- Efficiency from file

Plots of mechanical quantities are identical to the one in TestOneFlange2. DC power is different, instead, since here we substitute a loss formula with a map from a file:



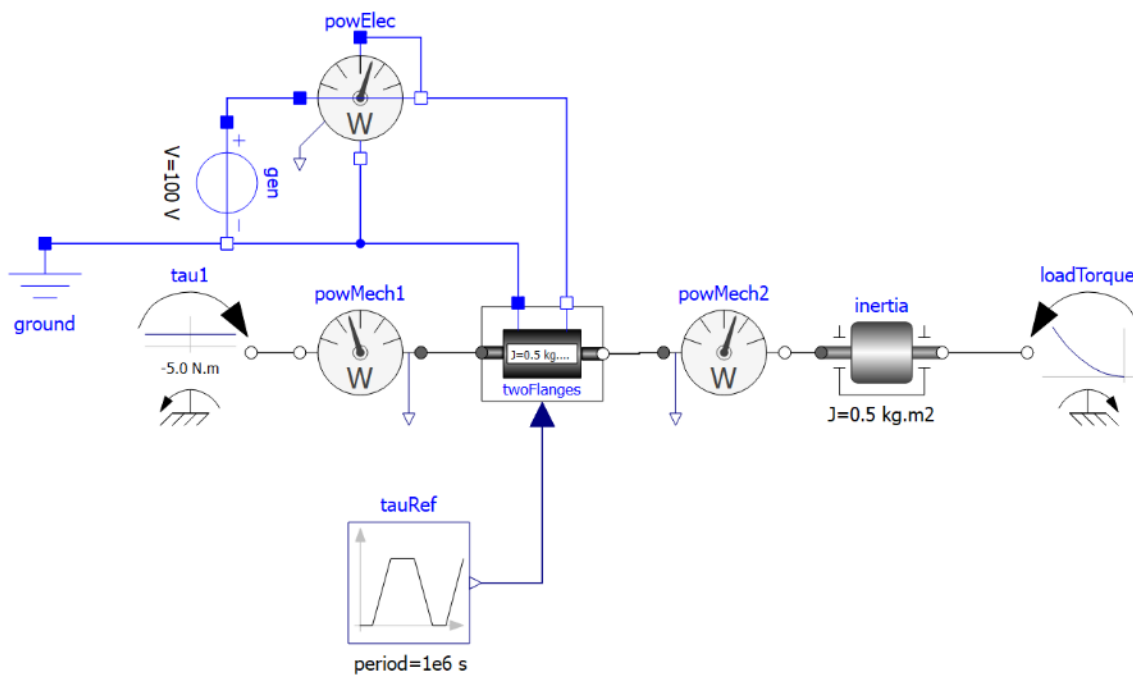
11.2.5 TestOneFlange1Conn

Identical to TestOneFlange2, but the usage of input connector to set the requested torque.

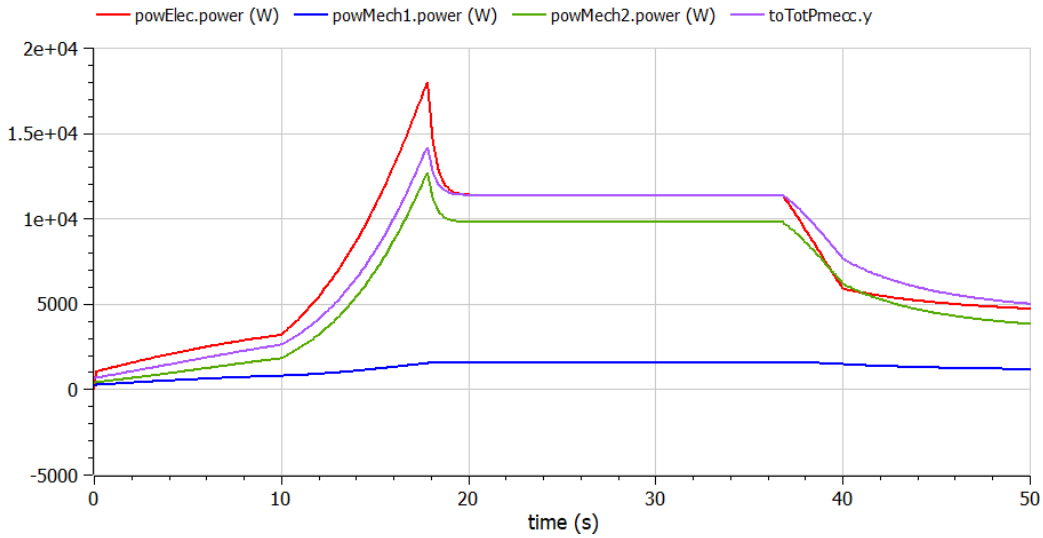
11.3 TwoFlange

11.3.1 TestTwoFlange1 and TestTwoFlange2:

Diagram for both:

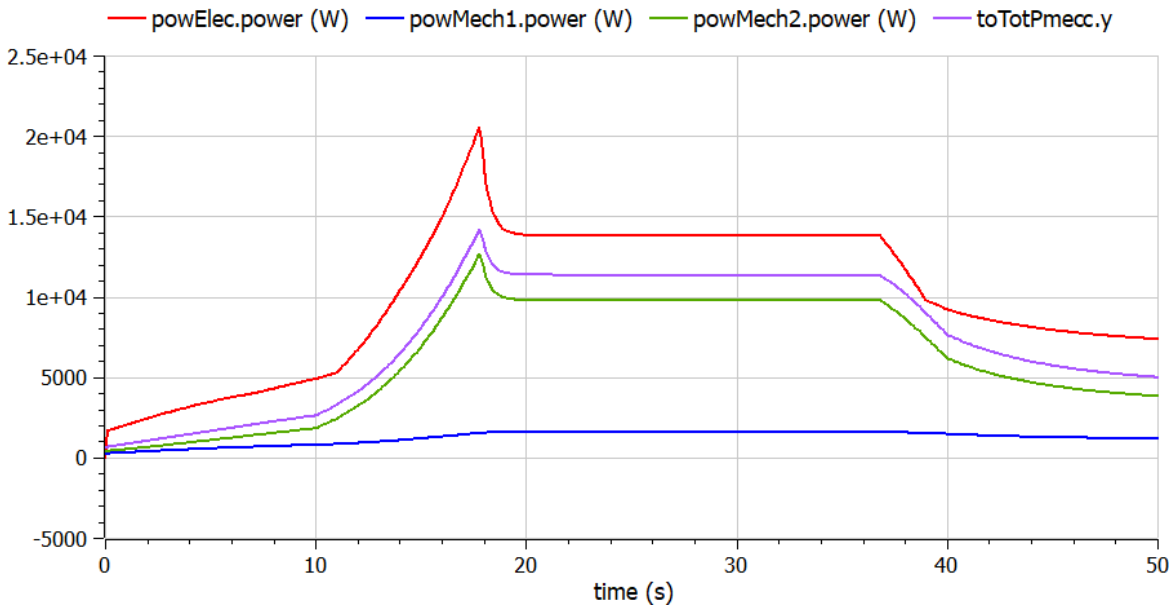


Plots from TestOneFlange1 (here twoFlanges has unit efficiency):



Here we see that the electric power, in steady state, is equal to the total mechanical power delivered by the two flanges. During transients, it is larger when accelerating and smaller when braking because of internal kinetic energy accumulation.

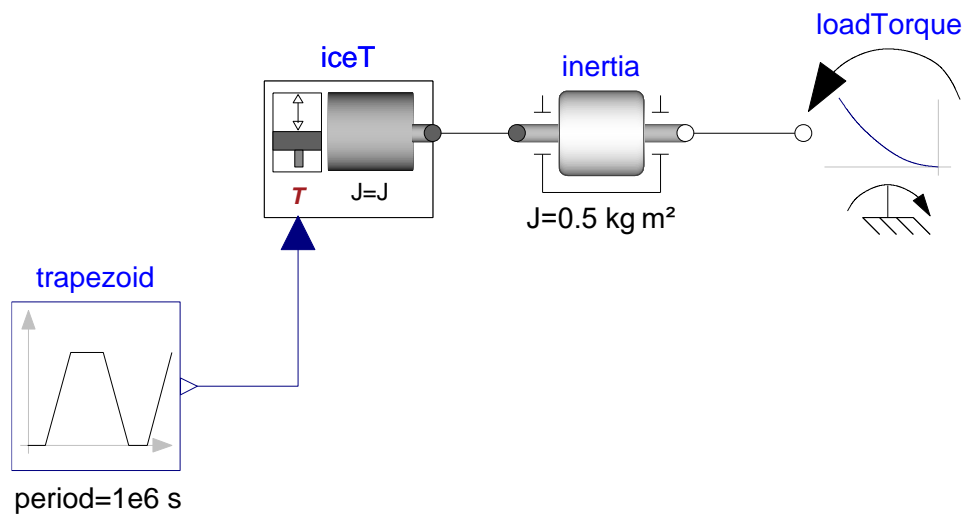
In TestTwoFlange2 we add some internal losses, through table from file “effTable”. Since we re-use an existing table thought for a different machine, we adapt it to the current use through a non unity `eSpeedFactor` (here `eSpeedFactor=0.6*60/(2*pi)`, i.e. 0.6, except for the correction rad/s to rpm). Now plots are similar to the previous ones, except of the fact that DC power is larger than total mechanical power in steadystate:



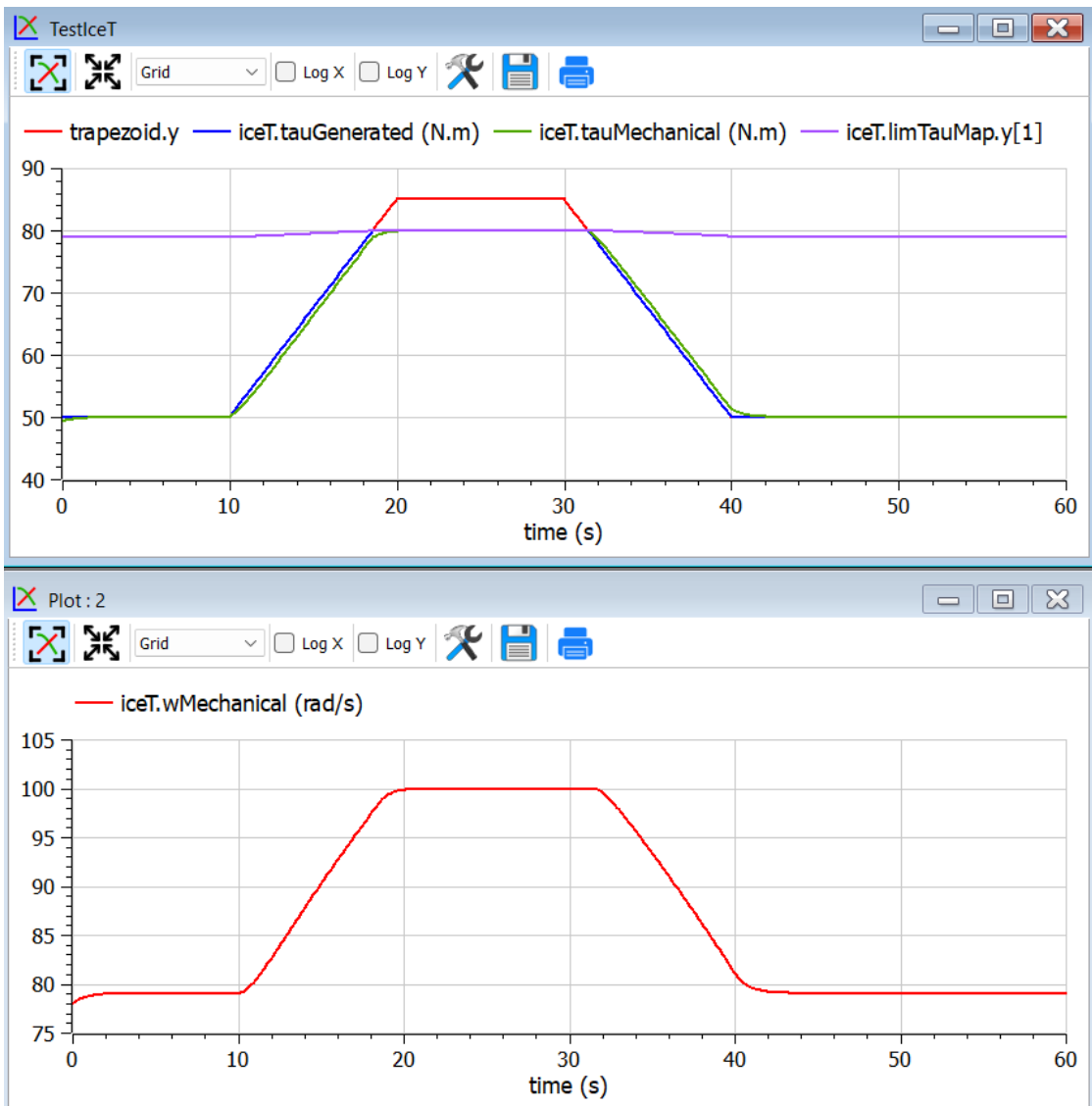
11.4 ICE

11.4.1 TestIceT

Diagram:



Some outputs:



In the first 2-3 seconds there is a slight transient to move the rotational speed to the one that corresponds to the equilibrium between the requested and load torque. The maximum deliverable

torque is much larger than the one actually requested. The maximum torque changes a little with ice rotational speed, given the maxIceTau matrix that is used in this example. When the requested torque overcomes the maximum deliverable, actual delivered torque is set equal to the maximum allowed.

maxIceTau matrix:

```
# First column: speed (rad/s) 2nd column: torque
double maxIceTau (5 2)
100 80
200 85
300 92
350 98
400 98
```

Specific consumption can be computed using manually inputted data or taking them from a file. The provided on-line data and file parameters correspond to the same specific consumption matrix:

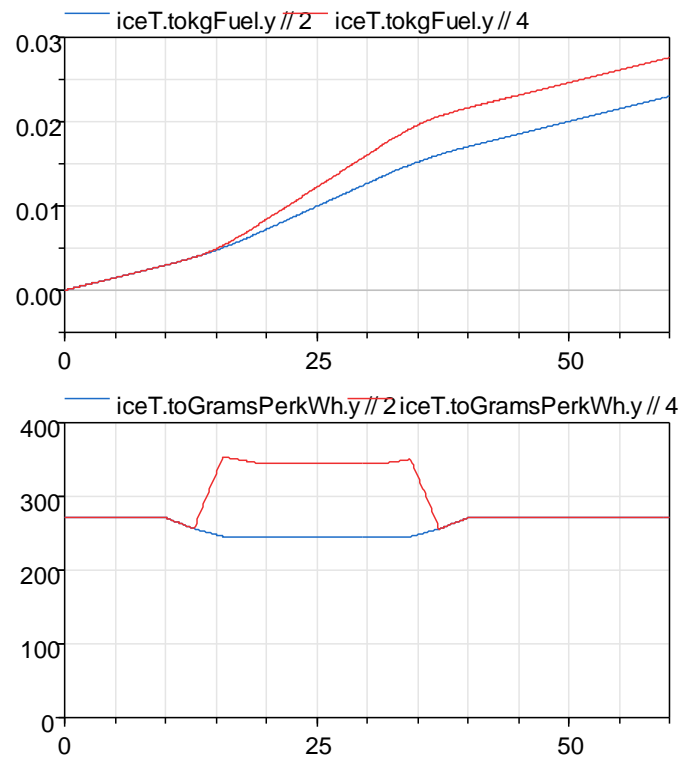
```
# First row: (from column 2) speed
# First column (from row 2): torque
# body: spec. consumption.
[0.0, 100, 200, 300, 400, 500;
 10, 630, 580, 550, 580, 630;
 20, 430, 420, 400, 400, 450;
 30, 320, 325, 330, 340, 350;
 40, 285, 285, 288, 290, 300;
 50, 270, 265, 265, 270, 275;
 60, 255, 248, 250, 255, 258;
 70, 245, 237, 238, 243, 246;
 80, 245, 230, 233, 237, 240;
 90, 235, 230, 228, 233, 235]
```

So specific consumption either from file or on-line data is the same. The following plots compare consumption using the above matrix and the following modified matrix (note a bolded numbers unrealistically modified):

0.0	100	200	300	400	500
10	630	580	550	580	630
20	430	420	400	400	450
30	320	325	330	340	350
40	285	285	288	290	300
50	270	265	265	270	275
60	255	248	250	255	258
70	345	237	238	243	246
80	345	230	233	237	240
90	335	230	228	233	235

We expect, with this modification to have much larger consumption only at high torques- low speeds.

We obtain with the original and modified matrix: (blue and red curves, respectively):

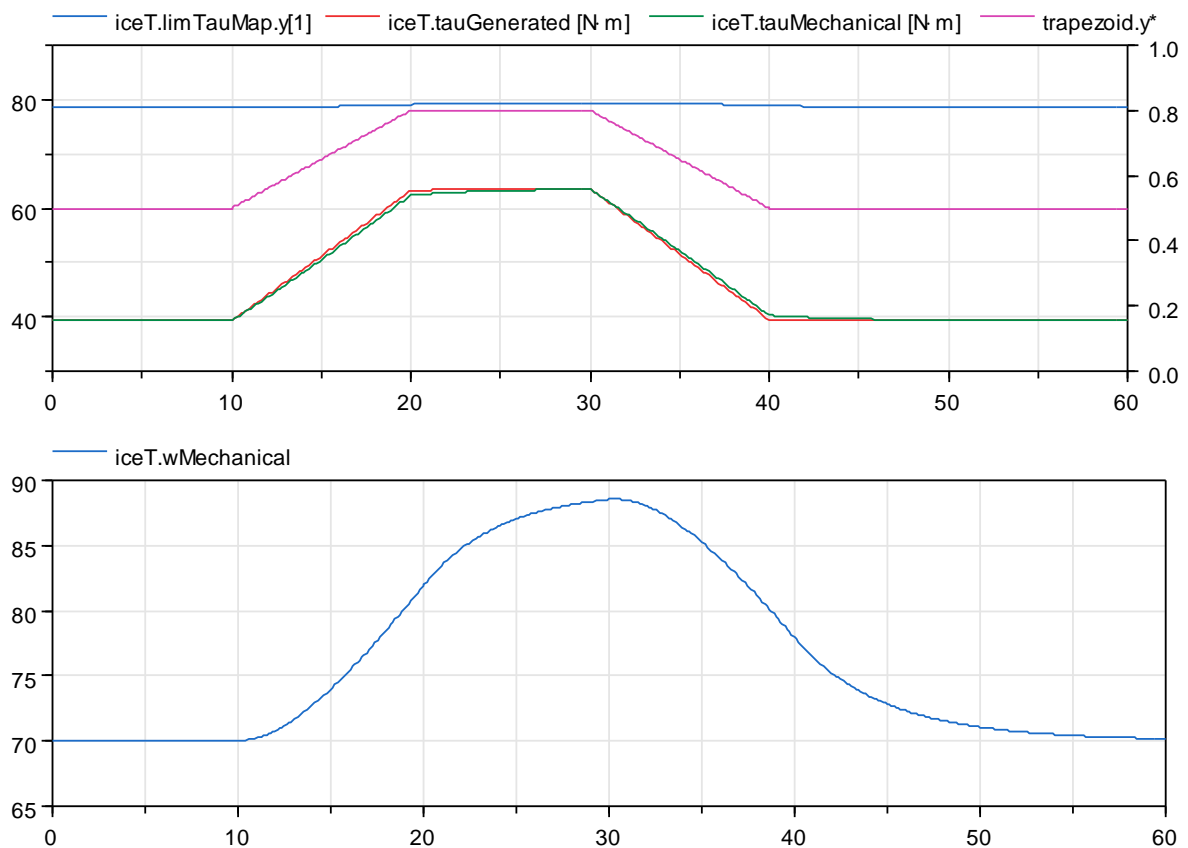


The results are as expected.

11.4.2 TestIceT01

Here I've enlarged the external inertia significantly (now it is 5 kg m²).

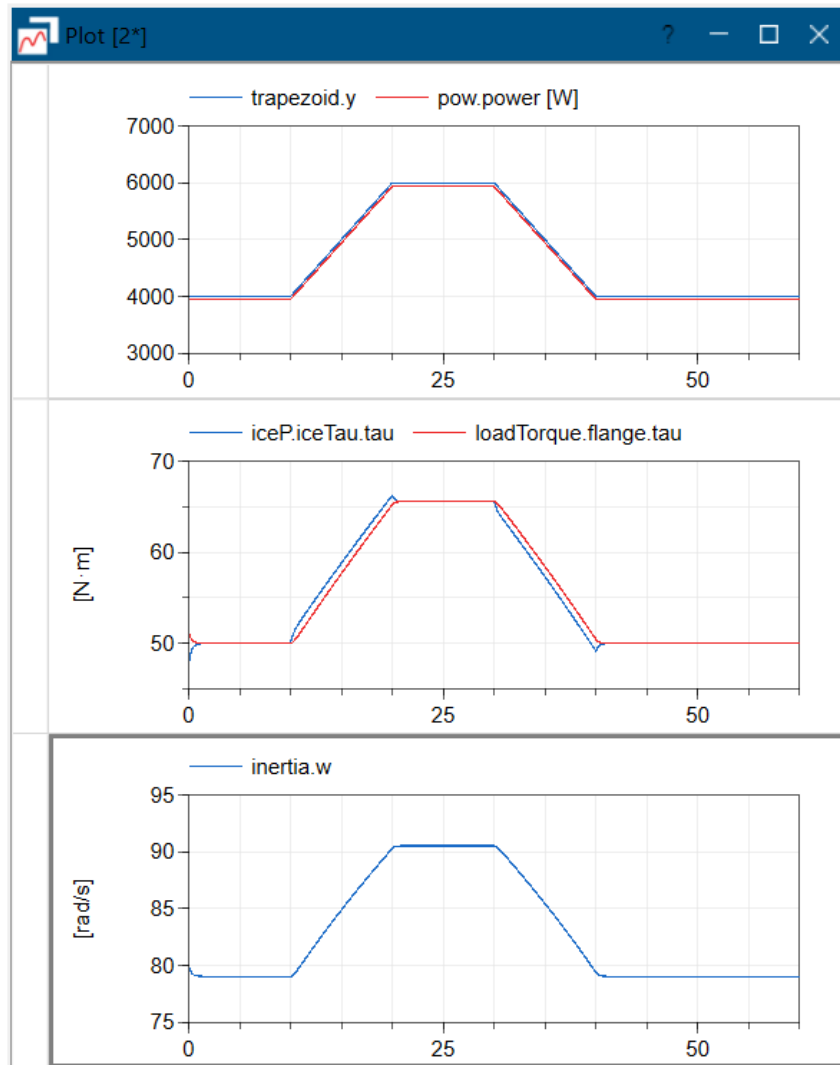
Some outputs:



The requested torque is obviously always lower than the allowable limit, since the normalised torque input is lower than one. However, the limit is important since the actual torque is the normalised input times this limit.

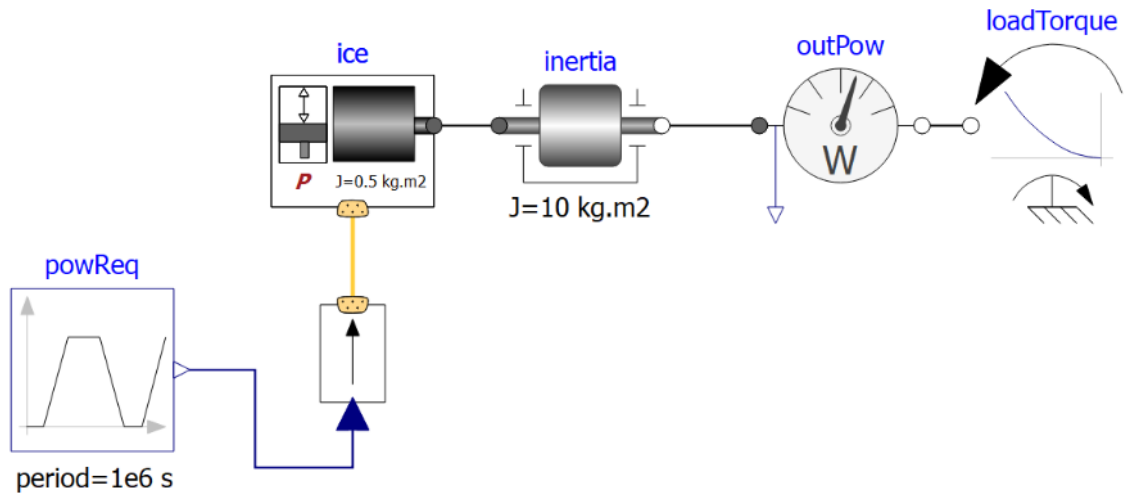
11.4.3 TestIceP

The behaviour is rather straightforward, and is easily explained by the following plots:

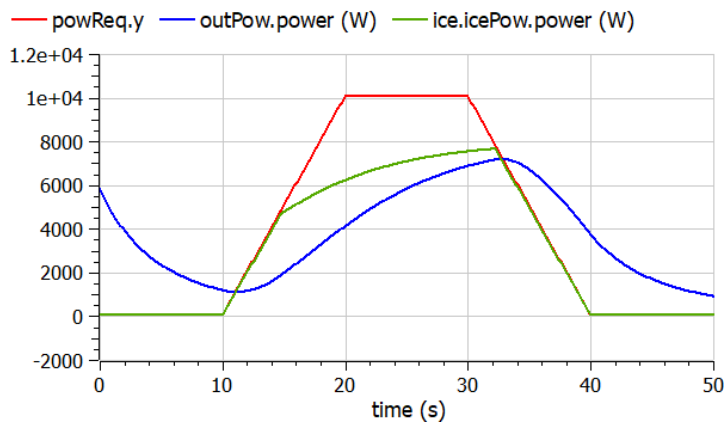


11.4.4 TestIceConn

Diagram:

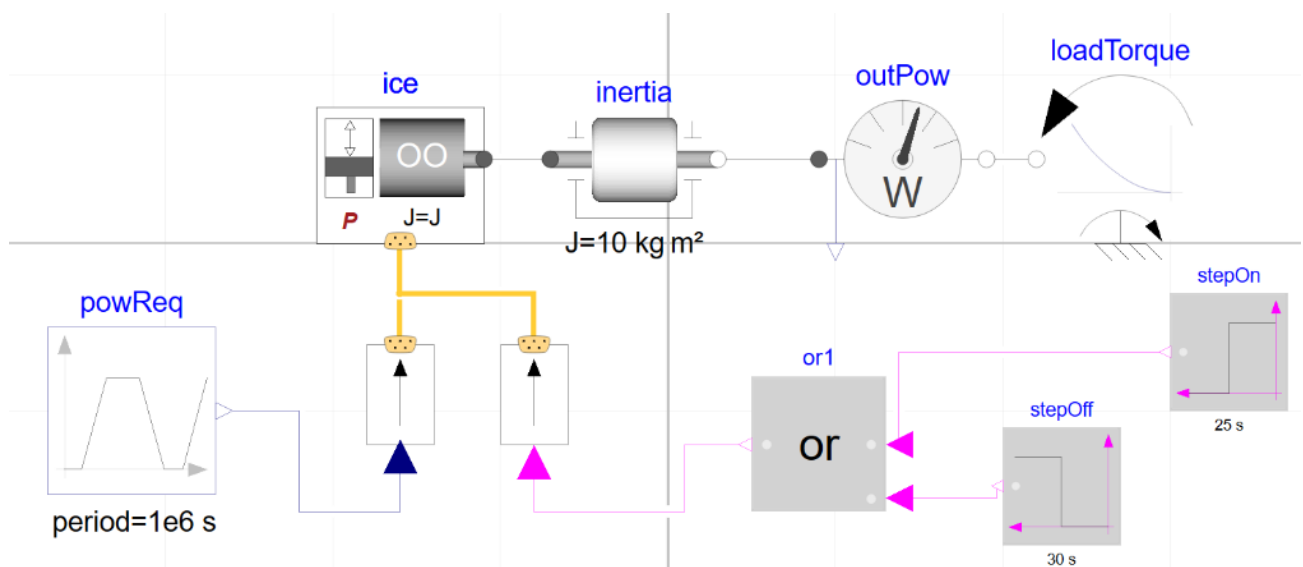


Some results:



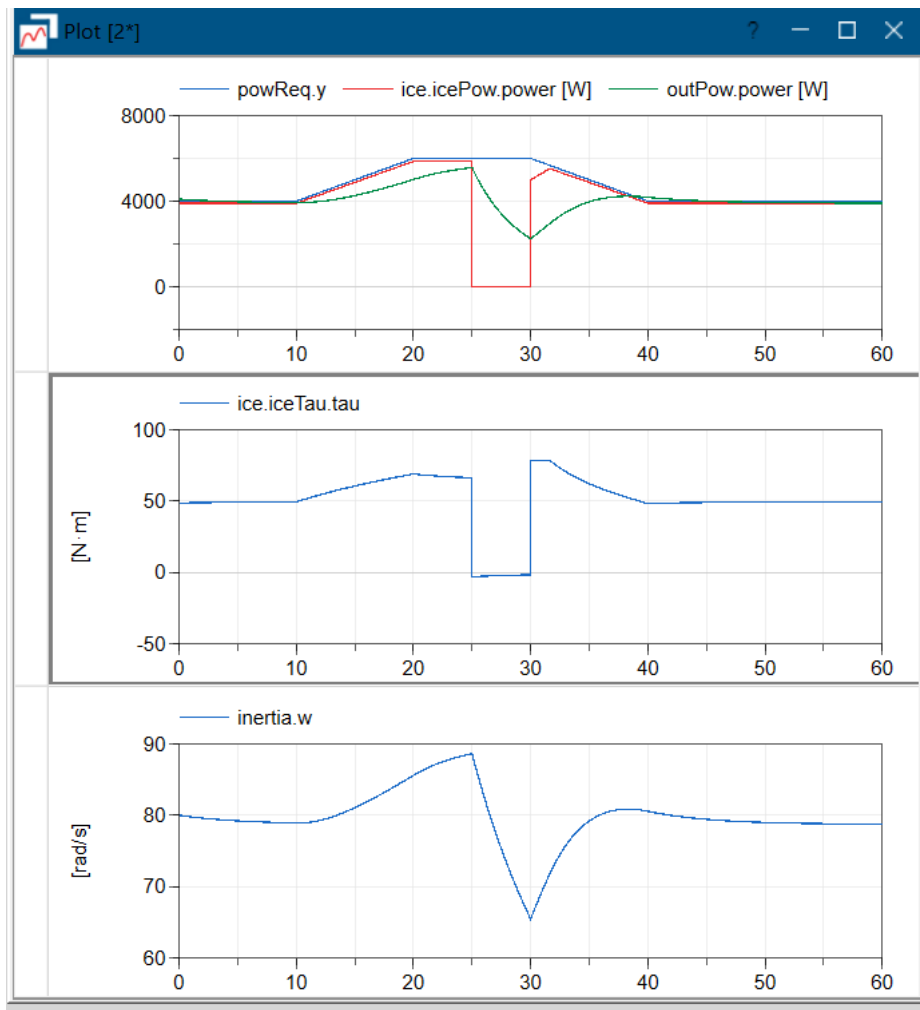
11.4.5 TestIceConnOO

Diagram:



In addition to following the power request, the model is switched off between 25 and 30s.

The following plots are rather self-explanatory:

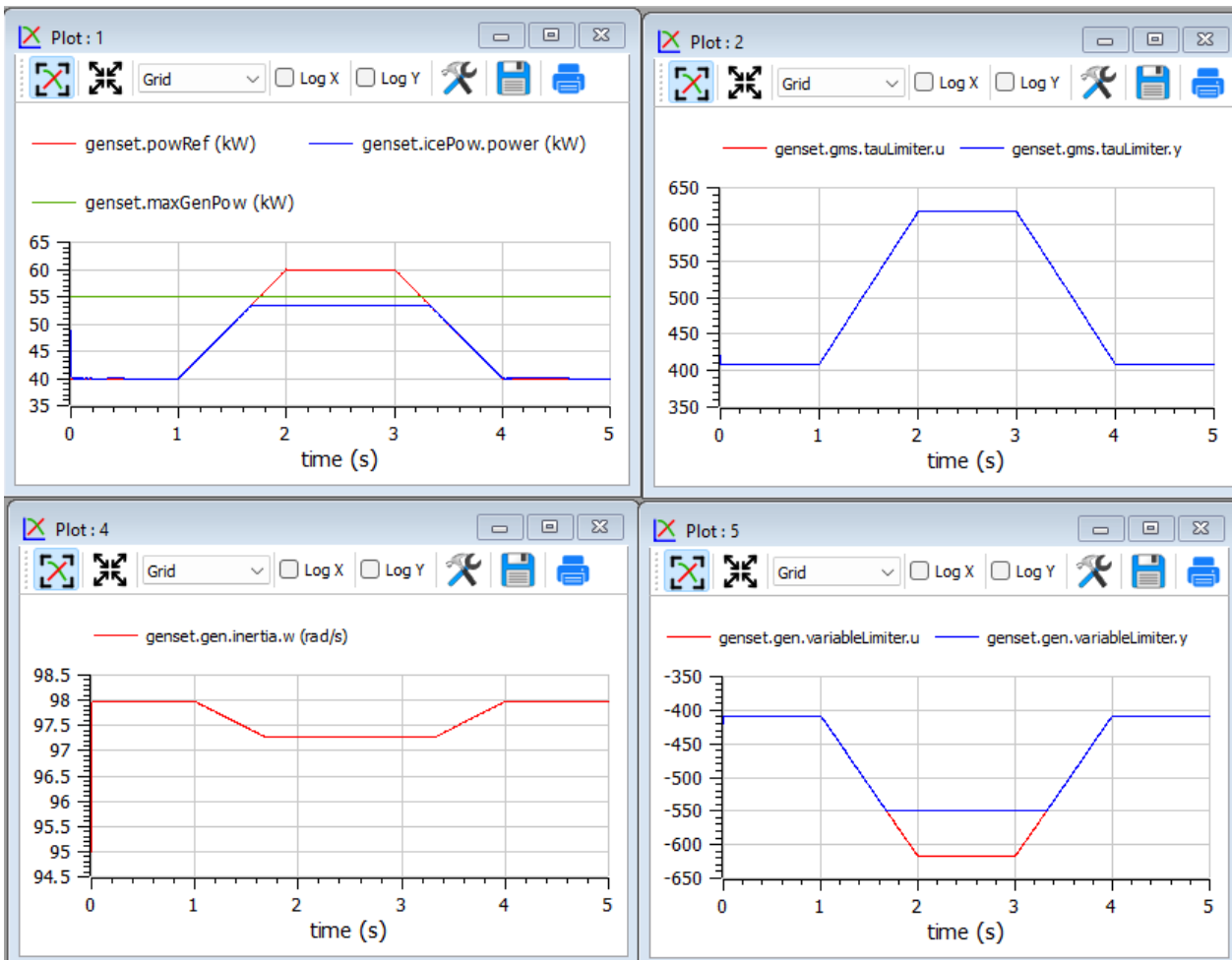


11.5 Genset

11.5.1 TestGenset1

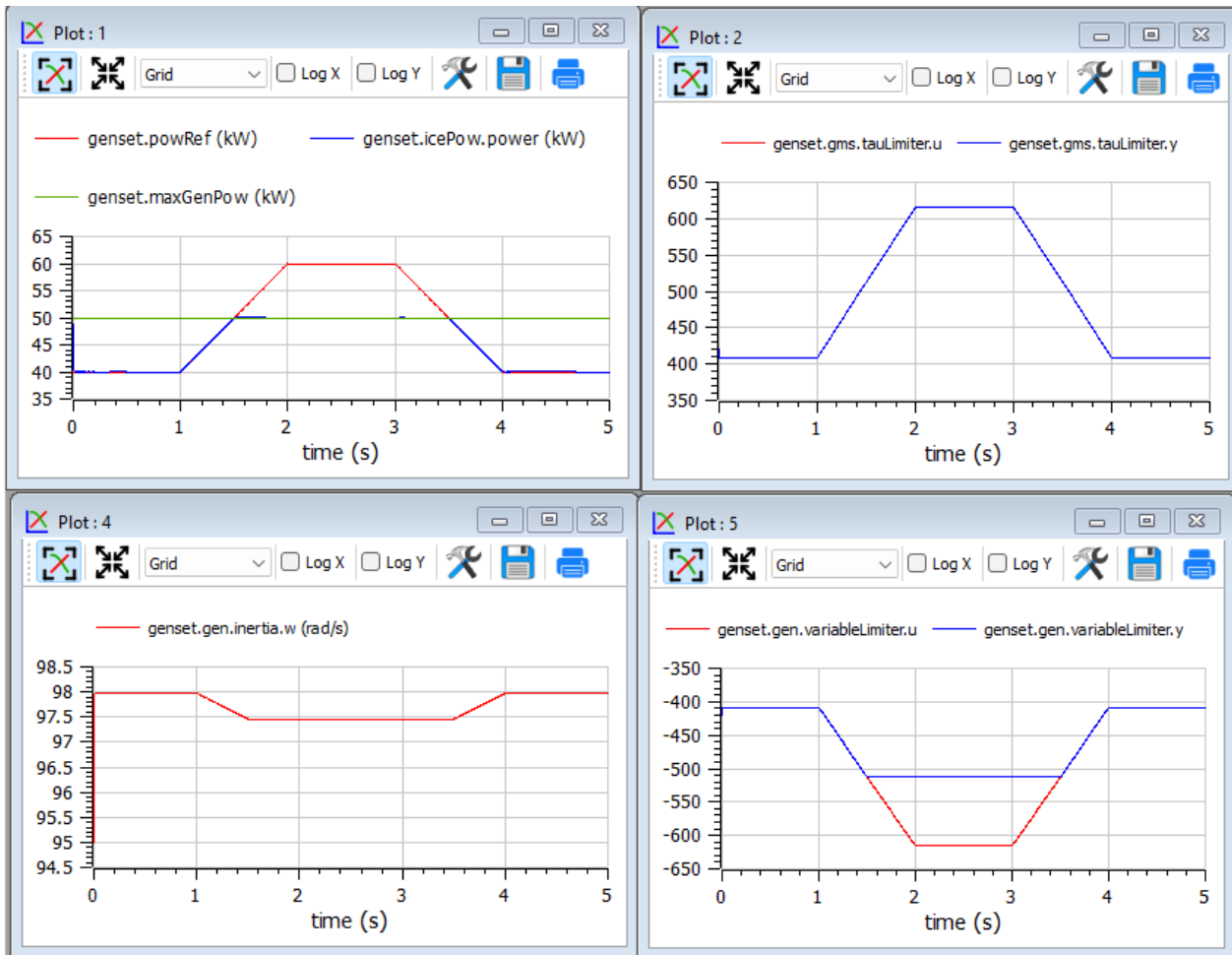
Here, we do not use maps on file, and introduce fixed limits in generator (thus genset) torque and power, and fixed ICE fuel consumption and optimal speed.

Some results with the default (built-in parameters):



We see that torque limitation to 550 Nm occurs in the central zone.

If we reduce `genset.maxGenPow` to 50 kW the results become as follows:



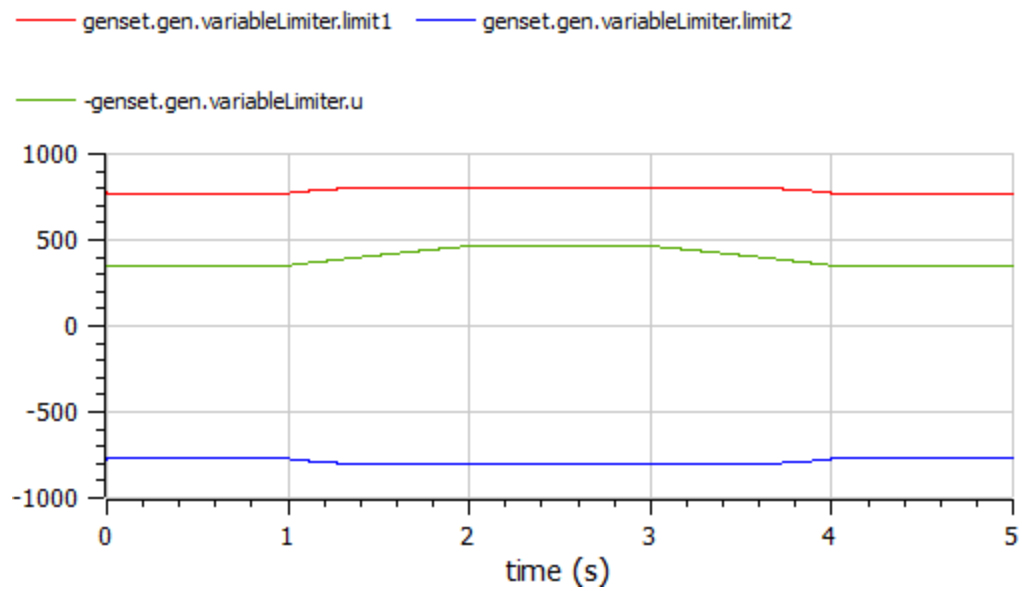
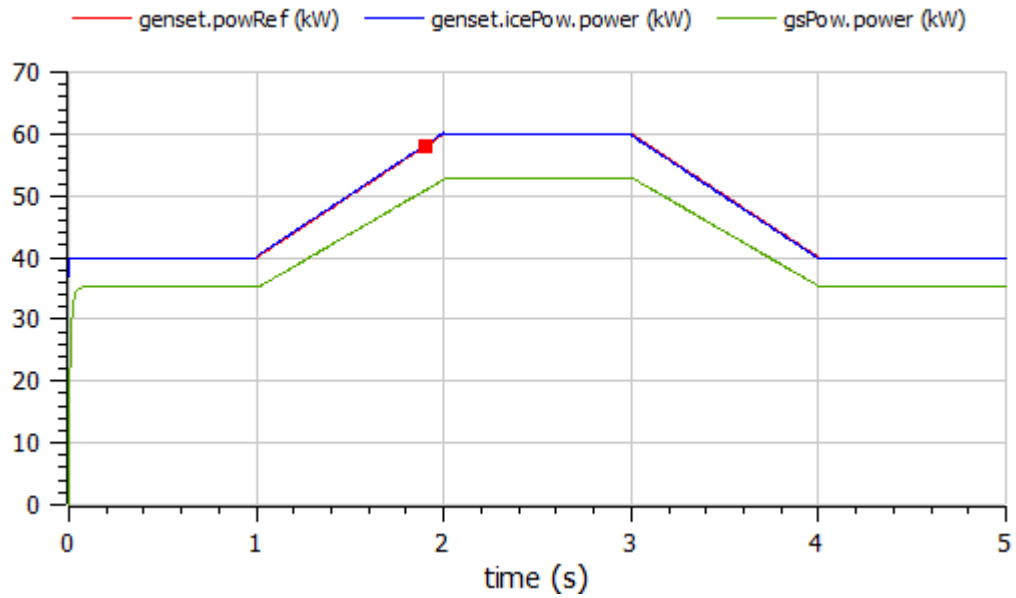
Now in the central zone, power limitation prevails. Genset rotational speed remains in the vicinity of the setpoint, which in this example is 100 rad/s.

This is a too basic genset, since it does not have <an optimal speed to follow, and related fuel map. To implement these, we must resort to maps on file, which is done with the same Genset component, used in a more advanced mode, as depicted by the following example TestGenset2-

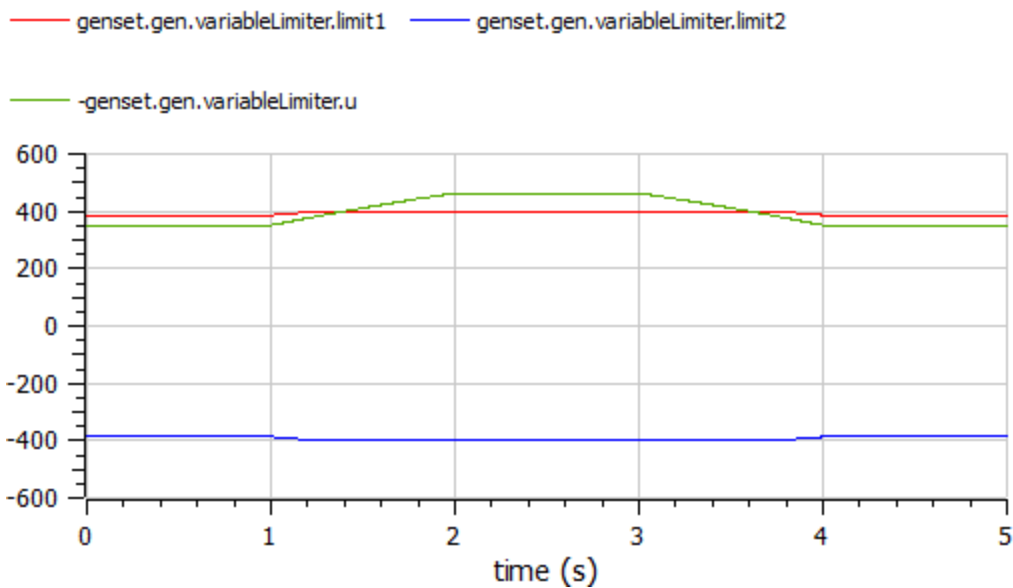
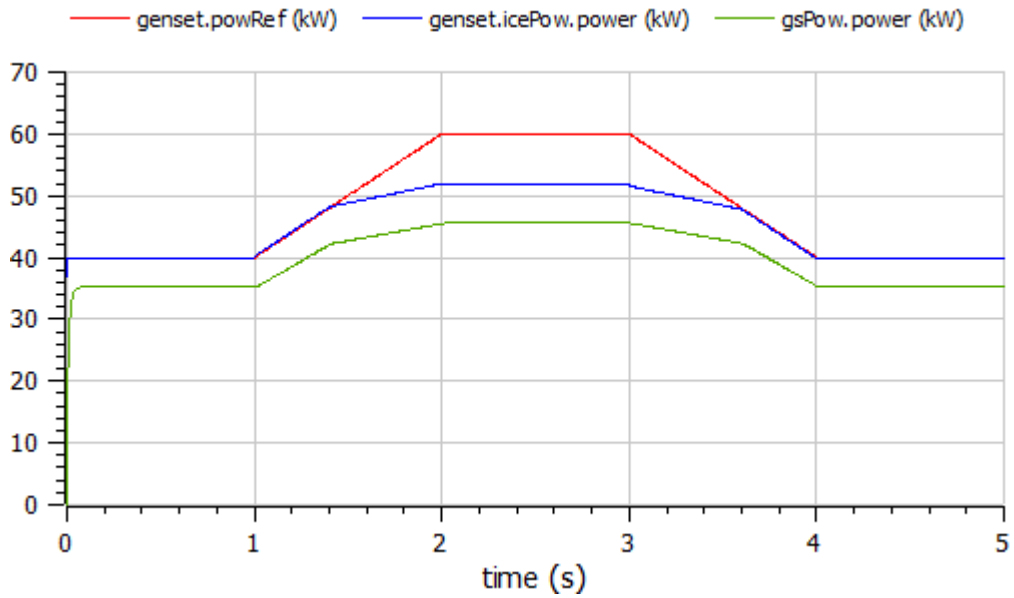
11.5.2 TestGenset2

Some plots follow (explanation in the model info).

Plots with default data:

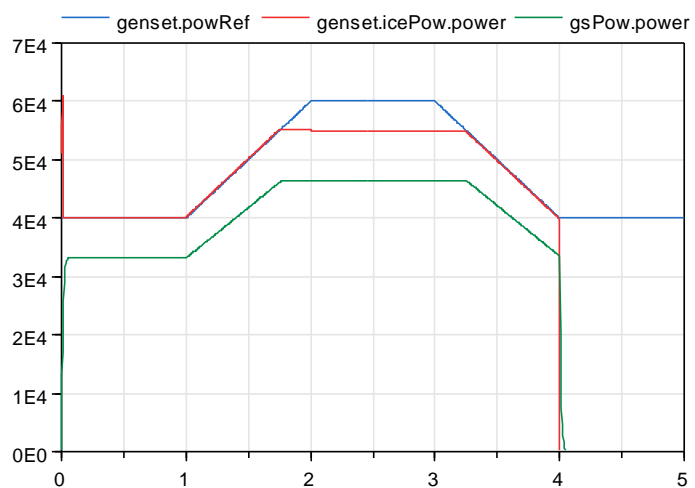


Plots reducing `genset.tlGenTorqueFactor` from 1 to 0.5:



11.5.3 TestGensetOO

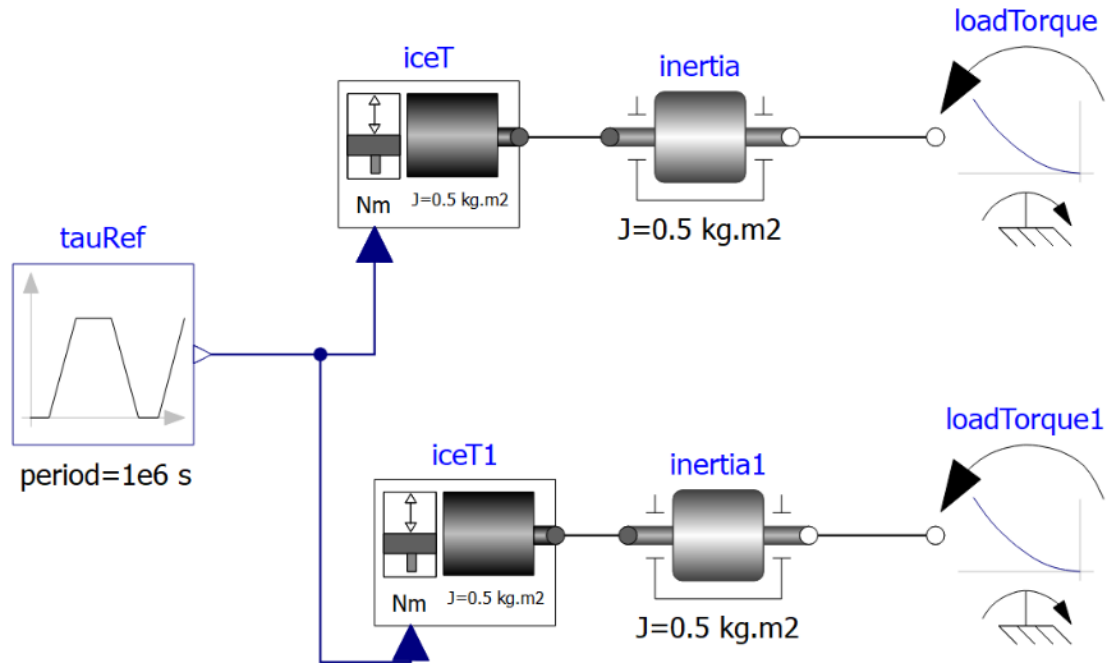
Some plots (explanation in the model info):



11.5.4 TestIceTmultipliers

Multipliers make very flexible the use of tables read from a txt file. This test just shoes the effect of changing max torque and specific consumption through multipliers.

Diagram:

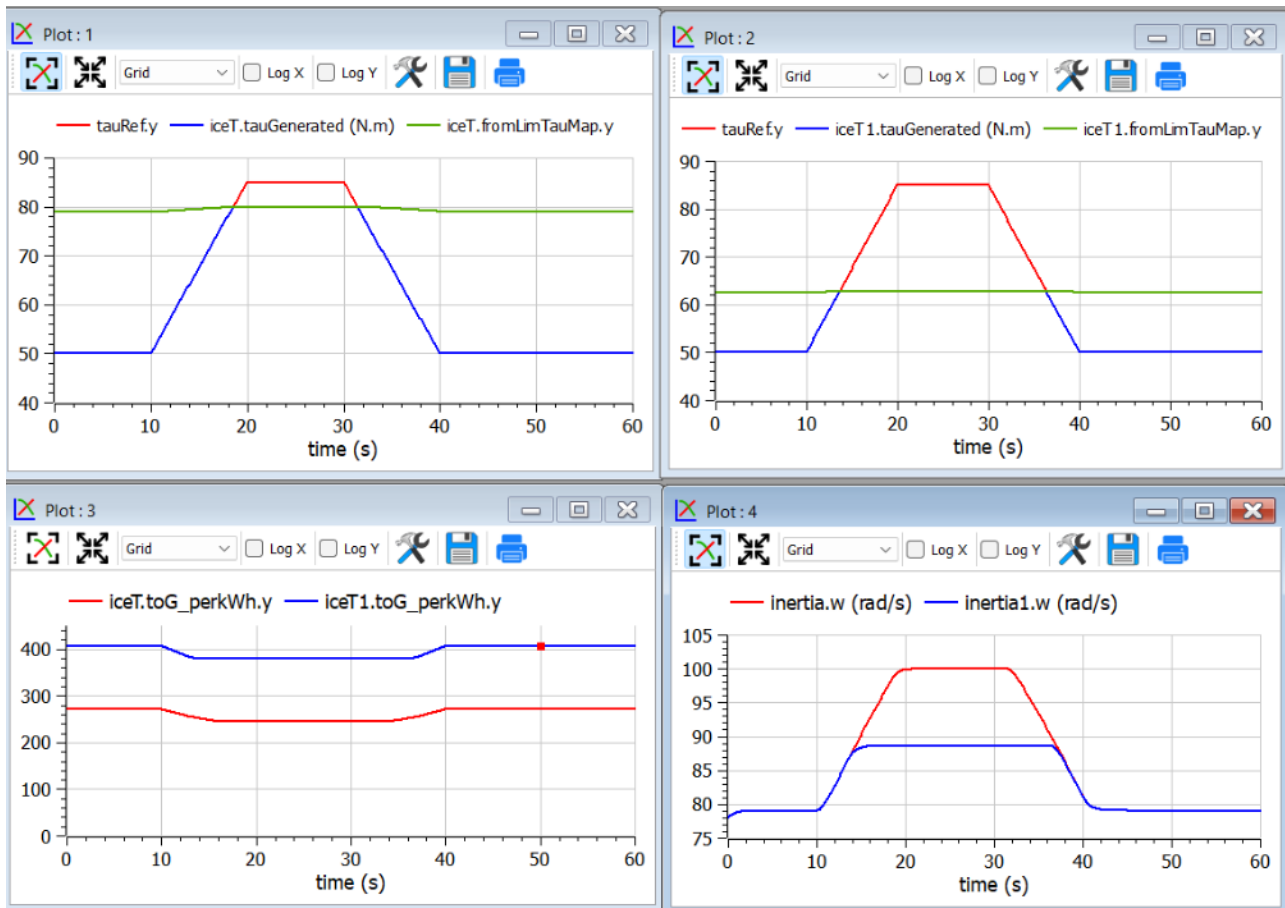


The below powerline is the same as the above, but has the following factors, instead of 1:

`scConsFactor=1.5, tlTorqueFactor=0.8`

A few plots below. In Plot1 we have curves from the above power line, in Plot2 from the below one.

In Plot3 and Plot 4 the red curves refer to the above line, the blue one to the below.



12 Appendix 1: Content of EVmaps.txt

To simulate some of the provided examples, maps are read from a file, which is provided in gitHub. These are reported were as well for completeness-

#1

```
# efficiency - rows:speeds pu, columns:torques pu
#
```

```
double effTable (6,6)
    0.00  0.00  0.25  0.50  0.75  1.00
    0.00  0.75  0.80  0.81  0.82  0.83
    0.25  0.76  0.81  0.82  0.83  0.84
    0.50  0.77  0.82  0.83  0.84  0.85
    0.75  0.78  0.83  0.84  0.85  0.87
    1.00  0.80  0.84  0.85  0.86  0.88
```

```
# first column: rpms; second Nm
```

```
double minTorque(9,2)
    0.0    -1122
    5839   -1122
    7007   -1016
    8175   -884
    9343   -777
    11095  -655
    15182  -473
    18686  -381
    23357  -296
```

```
# first column: rpms; second Nm
```

```
double maxTorque(9,2)
    0.0    1122
    5839   1122
```

7007	1016
8175	884
9343	777
11095	655
15182	473
18686	381
23357	296

13 Appendix 2: Efficiency.m

In section 9.3 an efficiency map is shown to illustrate graphically the effect of using the Modelica bloc AddLossesWT.

It is not very easy to obtain manually these maps starting from the used loss formula.

If Mathworks's Matlab^R is available, the following function can be used. If Matlab is not available, the readers can use Scilab which has a function language and set of functions very similar to Matlab, and create a similar function with it.

Here's the matlab function:

```
function [eff] = efficiency(A, bT, bS, bP, tq, sp)
%Plots efficiency contour
% pLoss = A+bT*Tq^2 +bS*Sp^2+ cP*(Tq*Sp).^2
% tq=[tqMin tqMax] (p.u. of torque max)
% sp=[spMin spMax] (p.u. of speed max)

steps=100;
% Generate equally-spaced torques between the two passed extrema:
TQ=tq(1):(tq(2)-tq(1))/steps:tq(2);
% Generate equally-spaced speeds between the two passed extrema:
SP=sp(1):(sp(2)-sp(1))/steps:sp(2);
% create torque and speed matrixes needed by contour to find the maps.
% To do this, we replicate speeds for all torques and torques for all
% speeds. for details, look at them in the workspace
[TQ SP]=meshgrid(TQ,SP);
% Calculate losses and efficiencies:
pLoss = A + bT*TQ.^2 + bS.*SP.^2 + bP*(TQ.*SP).^2;
eff=TQ.*SP./(TQ.*SP+pLoss);
figure(1);
% Draw contour lines for efficiencies:
[c, h]=contour(SP,TQ,eff, 0.7:0.01:1.00);
colorbar,title('efficiency');
clabel(c,h, [0.8 0.85 0.9 0.95 0.96 0.97]);
hold on;
plot([0 1],[1 1],1:0.1:1,1./(1:0.1:1));
hold off
figure(2);
% Draw contour lines for the losses:
[c h]=contour(SP,TQ,pLoss,0.01:0.01:0.15);
colorbar,title('losses');
clabel(c,h,[0.01 0.05 0.1 0.15]);
end
```

14 Appendix 3: Some info on EHTPlib architecture.

This library has been developed over the years without having an initial well-defined goal, but adding models every time I was in need of them.

Nevertheless, from time-to-time I make the library architecture more rational and coherent.

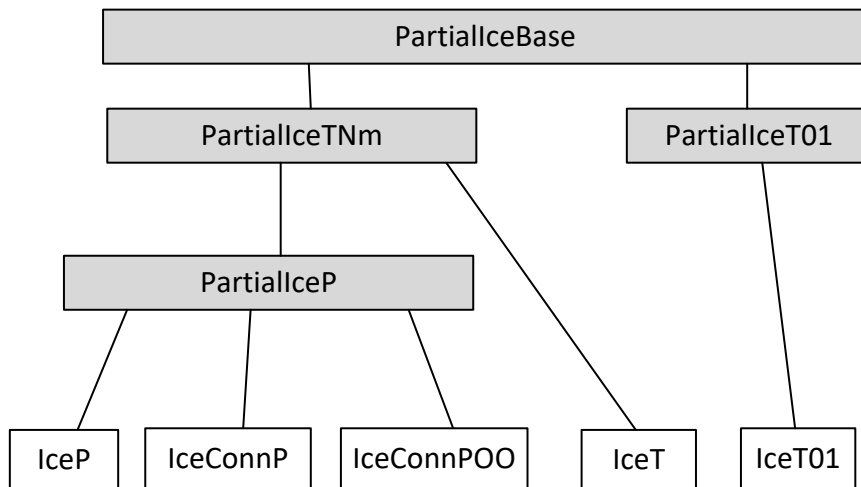
Here I present just some technicalities about inheritance.

14.1 Inheritance of ICE models

EHTPlib contains several Internal Combustion engine (ICE) models. For instance:

- They could be written so that the engine tries to follow a torque request; torque can be set as newton-metres or in per unit of the max deliverable torque
- They could be written so that the engine tries to follow a power request
- They can be able to switch ON and OFF or not
- Data about maximum available torque and specific consumption can be manually set or taken from files.

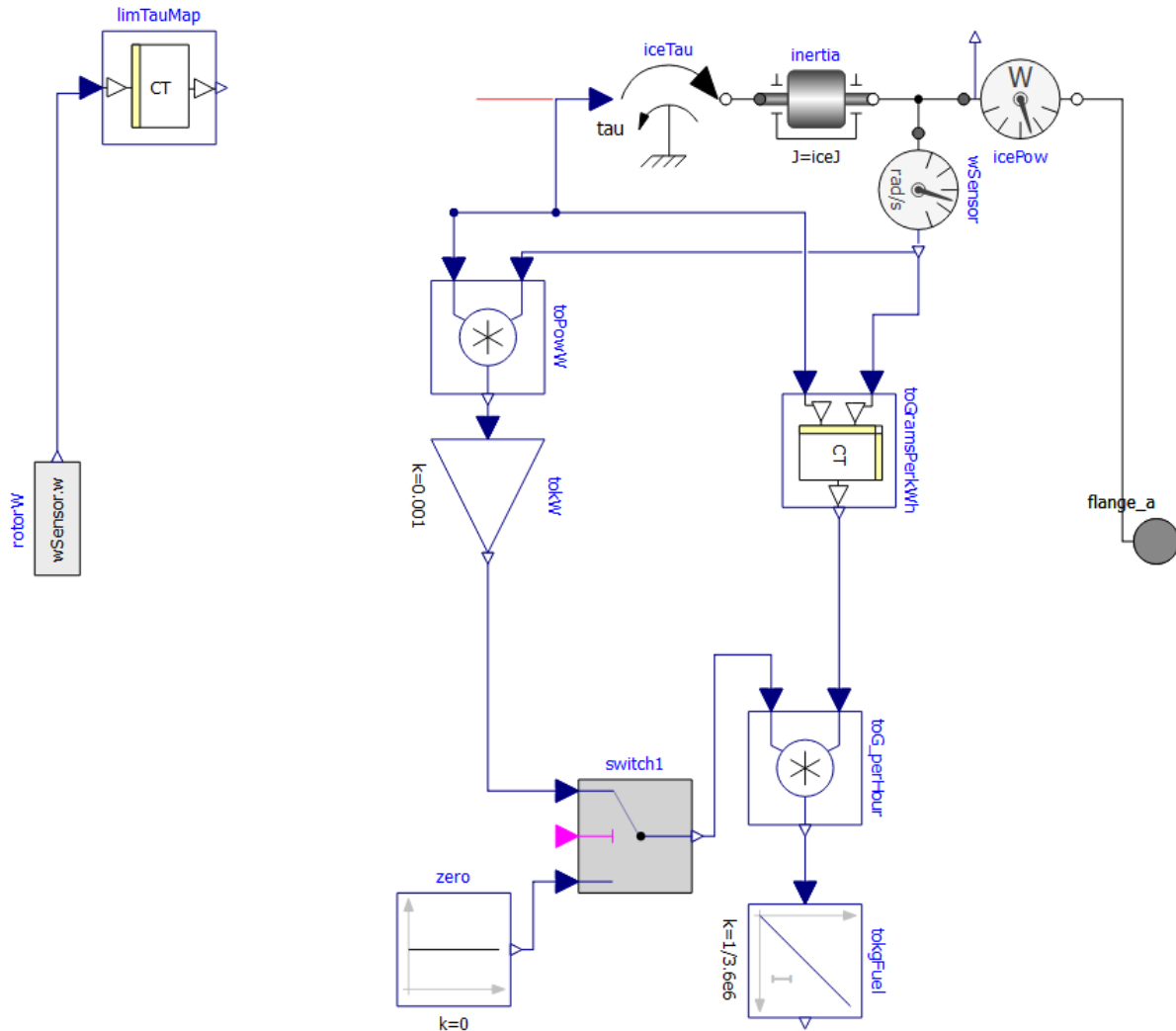
To minimise development and maintenance effort, the following taxonomy has been implemented:



Partial models are greyed

14.1.1 *PartialIceBase*.

Its diagram is as follows:



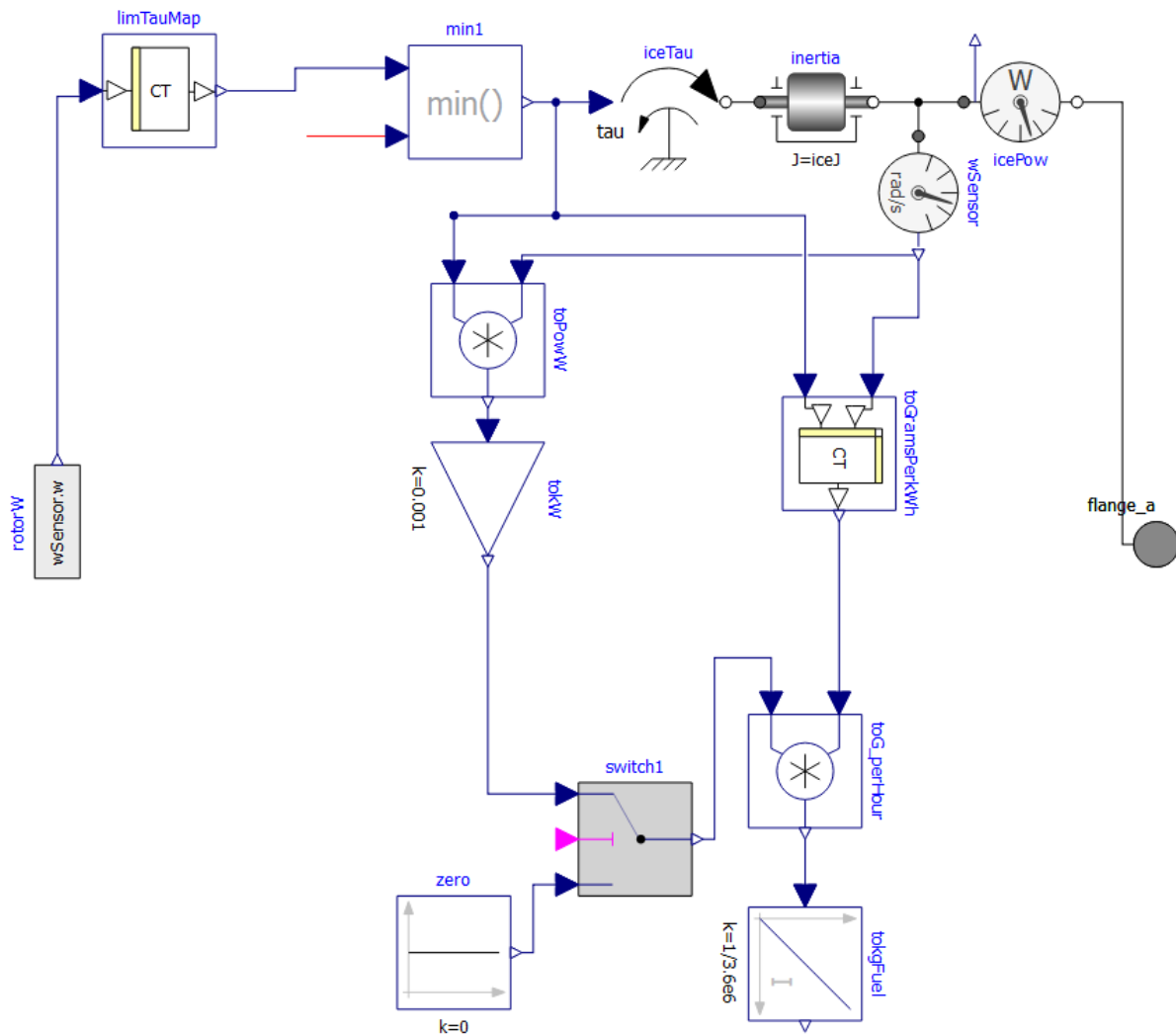
We see in the upper part the generation of torque, its application to an inertia, and the evaluation of consumption based on a map that can be selected manually or loaded from a file.

Children models will have to define the torque to be inputted to $iceTau$. i.e. will connect to the red line shown in the upper-left part of the diagram.

The **switch1** block needs a Boolean input. For children models not having the ON/OFF capability, this will be put to a constantly true value (engine ON); otherwise, it will operate according to an ON/OFF signal coming from the outside.

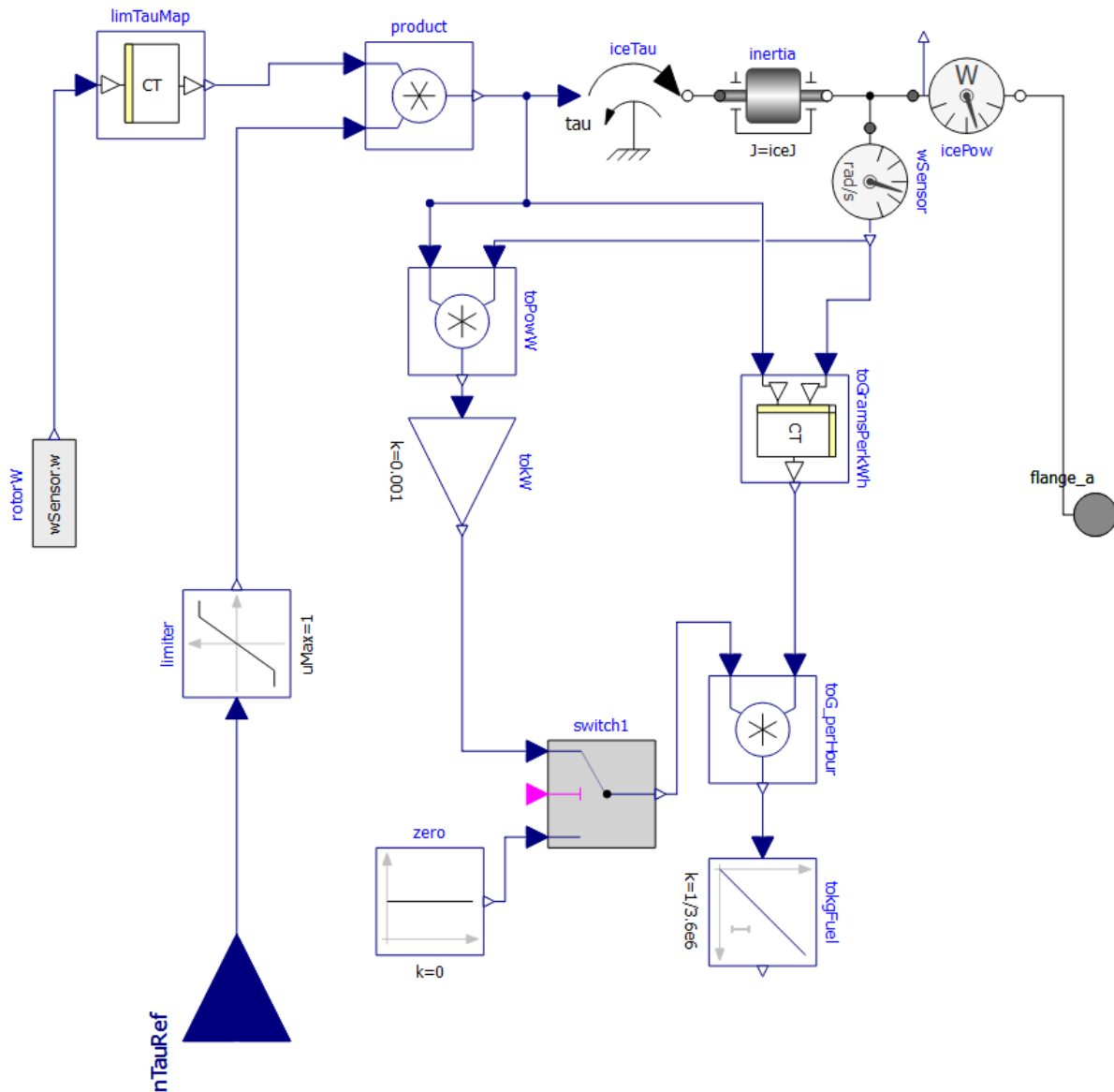
14.1.2 PartialIceTNm

This model adds the choice to input torque as Nm, and application of a maximum Torque table, that can be manually inputted or loaded from a file, as clear from the diagram, which is as follows:



Here there is the **min()** block, which implements possibility to limit torque to a maximum value.

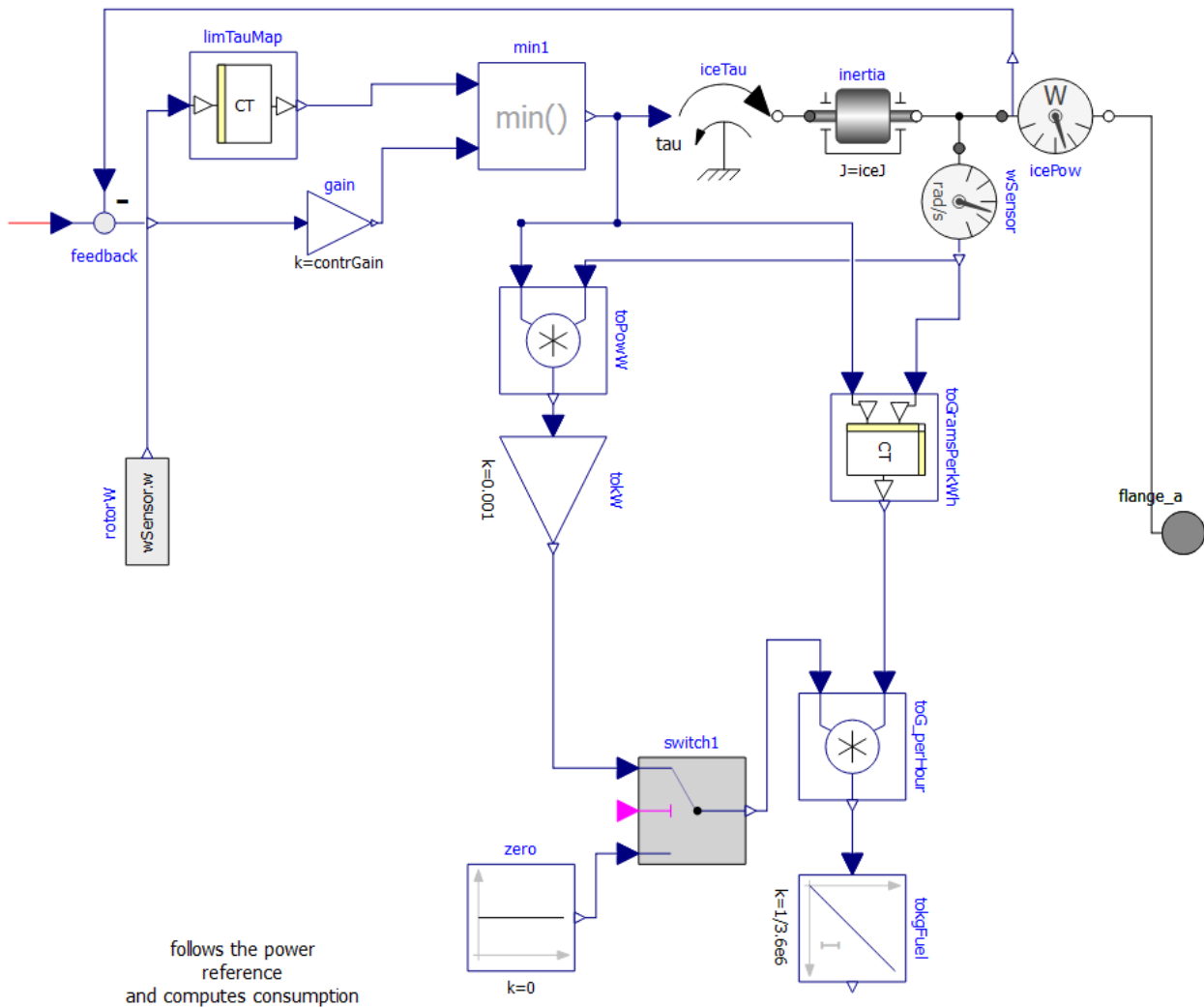
14.1.3 PartialIceT01



Here, instead of **min()** we have the **product** block, which lets this model's input torque to be interpreted as a fraction of the maximum allowable torque at each rotational speed. So, input torque must be a number between 0 and 1.

14.1.4 PartialIceP

Diagram:



14.1.5 IceP, IceConnP, IceConnPOO, IceT, IceT01

These models do not require special explanations here.

Here, we just notice that in IceConnPOO, the boolean input of **switch1**, is connected to an input from the exterior of the model, while in all the other cases it is connected to a constant equal to true.

14.2 Inheritance of OneFlange models

One flange models differ for:

- the different choice on limitation of torque (fixed values or data from a table). In case they are from a table, the table can have normalised limits (between 0 and 1) or dimensioned (torques in Nm, speeds in rad/s)
- the different choice on efficiency: efficiency and electric power can be computed through a loss formula or a combi table as well;
- the way the input torque command is received: through a Real input or an expandable connector.

After some thought it have been decided that a very simple inheritance scheme was the most suitable, the one shown in figure 65.

Selection on the way the model operates are through flags (boolean parameters):

- `limitsOnTable` => true when they are from a table, false if they are fixed values Two subcases are possible, depending on boolean `normalisedTorqueInput`:
 - `normalisedTorqueInput` => true if the input table has torques and speeds normalised (will be multiplied by `wMax` and `tauMax`)
- `efficiencyOnTable` => true when efficiency is taken from an input table, false otherwise (a loss formula is then used)

In figure 65, Tref input and Conn input boxes are just to show that these input have been added to create the two children of `PartialOneFlange`. They are not partial classes, since they are too simple to deserve this role.

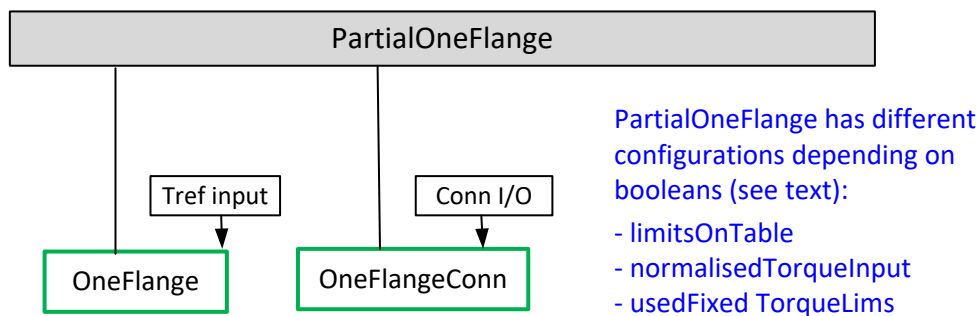


Figure 65: The two oneFlange models.

14.3 Inheritance of Gensets

I have defined two gensets:

- A basic version which defines receives and processes the input power requested from the GMS (`Genset`)
- An enhanced version which is also able to switch ON/OFF the engine (`Genset00`)

Both inherit from `MapBased.Partial.PartialGenset`.

14.4 Inheritance of ECUs

The three ECUs provided in `EHTPlib` are somewhat progressive: ECU1 has basic functionality, ECU2 adds compensation to keep battery SOC under control, ECU3 adds ON/OFF operation.

To ease maintenance, and also to favour visual comparison of the three ECUs, thy all inherit from `PartialECU`.

15 References

- [1] M. Ceraolo: "New Dynamical Models of Lead-Acid Batteries", *IEEE Transactions on Power Systems*, November 2000, Vol. 15, N. 4, pp. 1184-1190.
- [2] M. Ceraolo, A. Caleo, P. Capozzella, M. Marcacci, L. Carmignani, A. Pallottini: "A Parallel-Hybrid Drive-Train for Propulsion of a Small Scooter", *IEEE Transactions on Power Electronics*, ISSN: 0885-8993, Vol. 21, N. 3, May 2006, pp768-778.
- [3] M. Ehsani, Y. Gao, A. Emadi: "Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design", CRC Press, 2009, ISBN 9781420053982

- [4] M. Ceraolo and D. Poli: "Fundamentals of Electric Power Engineering", a full book (532 pages) published by IEEE-Wiley, 2014, ISBN 978-1-118-67969-2.
- [5] Toyota documentation <http://www.evworld.com/library/toyotahs2.pdf>, May 2003, file available for download on June 2017
- [6] M. Ceraolo, T. Huria, G. Lutzenberger: "Experimentally determined models for high-power lithium batteries", Book *Advanced battery technology*, ISBN: 978-0-7680-4749-3 doi:10.4271/2011-01-1365. Also presented at the *SAE 2011 World Congress*. Cobo Center Detroit, Michigan (USA), 12-14/4/2011.
- [7] OpenModelica Consortium www.OpenModelica.org
- [8] Dassault Systèmes Smart Electric Drives Library documentation: <http://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/dymola-smart-electric-drives-library.pdf>; File available for download on April 2015
- [9] Modelica Association: <https://www.modelica.org/libraries>, Vehicle Interfaces library Link available on April 2015.
- [10] EERE Information Center, https://www1.eere.energy.gov/vehiclesandfuels/pdfs/success/advisor_simulation_tool.pdf file available for download on June 2017.
- [11] Information on Toyota unit sold from Toyota <https://www.treehugger.com/cars/toyota-prius-hybrid-reaches-3-million-units-sold-worldwide.html>
- [12] Toyota Prius item on Wikipedia: https://it.wikipedia.org/wiki/Toyota_Prius