Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Design of the Modelica Library VehProLib with Non-ideal Gas Models in Engines

Examensarbete utfört i Teknisk Fysik
vid Tekniska högskolan vid Linköpings universitet
av

Conny Andersson

LiTH-ISY-EX–15/4888–SE

Linköping 2015

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Design of the Modelica Library VehProLib with Non-ideal Gas Models in Engines

Examensarbete utfört i Teknisk Fysik
vid Tekniska högskolan vid Linköpings universitet
av

Conny Andersson

LiTH-ISY-EX–15/4888–SE

Handledare:	**Doktorand Vaheed Nezhadali**
	ISY, Linköpings universitet
	Ingenjör Annelie Mossberg
	Wolfram MathCore
Examinator:	**Professor Lars Eriksson**
	ISY, Linköpings universitet

Linköping, 10 augusti 2015

Avdelning, Institution Division, Department	Vehicular Systems, Department of Electrical Engineering, SE-581 83 Linköping	**Datum** Date: 2015-08-10

Språk Language	**Rapporttyp** Report category	**ISBN** —
☐ Svenska/Swedish	☐ Licentiatavhandling	**ISRN** LiTH-ISY-EX–15/4888–SE
☒ Engelska/English	☒ Examensarbete	**Serietitel och serienummer** Title of series, numbering
	☐ C-uppsats	**ISSN** —
	☐ D-uppsats	
☐ ________	☐ Övrig rapport	
	☐ ________	

URL för elektronisk version

http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-XXXXX

Titel Title: Designval av Modelica Biblioteket VehProLib med Icke-Ideal Gasmodell i Förbränningsmotorer

Design of the Modelica Library VehProLib with Non-ideal Gas Models in Engines

Författare Author: Conny Andersson

Sammanfattning
Abstract

This thesis covers the reconstruction and the redesign of the modeling library VehProLib, which is constructed in the modeling language Modelica with help of the modeling tool Wolfram SystemModeler. The design choices are discussed and implemented. This thesis also includes the implementation of a turbocharger package and an initial study of the justification of the ideal gas law in vehicle modeling. The study is made with help of Van der Waals equation of states as a reference of non-ideal gas model. It will be shown that for the mean-value-engine-model, the usage of ideal gas law is justified.

Nyckelord
Keywords: Modelica, Wolfram SystemModeler, library design, Van der Waals equation of states, MVEM, mean-value-engine-model, vehicle propulsion, thermodynamics, turbo, turbocharger

Abstract

This thesis covers the reconstruction and the redesign of the modeling library VehProLib, which is constructed in the modeling language Modelica with help of the modeling tool Wolfram SystemModeler. The design choices are discussed and implemented. This thesis also includes the implementation of a turbocharger package and an initial study of the justification of the ideal gas law in vehicle modeling. The study is made with help of Van der Waals equation of states as a reference of non-ideal gas model. It will be shown that for the mean-value-engine-model, the usage of ideal gas law is justified.

Acknowledgments

Thanks to Professor Lars Eriksson at ISY and Jan Brugård at Wolfram Mathcore for providing a thesis that has been both challenging and interesting. Thanks to Ph.D. Student Vaheed Nezhadali for supervising my thesis work, always available and his great understanding in both vehicle systems and modeling in Simulink has been of an excellent help. Thanks to Anneli Mossberg for supervising my thesis work and always being helpful when I encounter problem associated with Wolfram SystemModeler. Thanks all the people at Wolfram MathCore for all help and support during my thesis and a familiar atmosphere at the office. Finally I also want to thank Johan Andersson, Oscar Wyckman and Junting Qui for great discussions and a good atmosphere in the project room at Vehicular Systems.

Linköping, August 2015
Conny Andersson

Contents

Notation

Abbreviations

Abbreviation	Meaning
CI	Compression-Ignition
BSR	Blade Speed Ratio
EM	Exhaust Manifold
IM	Intake Manifold
HEV	Hybrid Electric Vehicle
MSL	Modelica Standard Library
MVEM	Mean Value Engine Modeling
NEDC	New European Driving Cycle
TFP	Turbine Flow Parameter
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
RPM	Revolution Per Minute
RPS	Revolution Per Second
SI	Spark-Ignition
VehProLib	Vehicle Propulsion Library
VehProTestLib	Vehicle Propulsion Test Library
WSM	Wolfram SystemModeler

1 Introduction

1.1 Purpose

In today's auto-mobile industry the demands for more efficient and environmental friendly vehicles are steady increasing. With the help of modeling tools, developments can be made using less resources in time, material and manpower to achieve such goals. Today's vehicle area does not only consist of vehicles using combustion engines, but also hybrid vehicles. To be able to try and evaluate different vehicle compositions using simulations, industry can fast forward the development speed to achieve the goals of the market while keeping the environmental and security restrictions.

The purposes of the thesis are to have a functional VehProLib in WSM with a more user friendly design complemented by a test library. To include a turbocharger package which consist of a foundation for all turbocharged component, i.e. the hardware specific features can be declared on top of a template model (called partial model in Modelica). The thesis is intended to feature a study of the justification of the ideal gas law in modeling of turbocharged engines.

1.2 Outline

This thesis covers the reconstruction and expansion of VehProLib, the expansion covers both inclusion of new components as well as design choices for the library. Modelica code throughout the report is presented using a `Truetype` font, Modelica classes are presented using a **`Bold Truetype`** font and Mathematica code is presented using a Helvetica font. All code included in this report passes through WSM validation process (a fast process for finding errors), for readability reasons all code involving graphical changes has been removed.

The thesis outlined is as follows:

Chapter 1.2, Background This chapter covers a brief background of the VehProLib and some fundamentals behind vehicle modeling, together with a short introduction to the simulation and mathematical resources used in this thesis.

Chapter 2, Methodology The Methodology chapter covers the methods used during the library restoration, design investigation and inclusion of new components.

Chapter 3, Result This chapter covers results achieved based on the three parts mentioned in the previous chapter.

Chapter 4, Conclusions The conclusions chapter covers the conclusions and the recommendations for further work and improvements.

Appendix A, Derivation of Van der Waals Constants The complete derivation of Van der Waals constants, recommended for the interested reader.

Appendix B, Library Restoration This chapter covers the more detailed investigation and choices made during the restoration of the library.

Appendix C, Library Design More details about the design investigation and the changes that have been made can be found in this chapter.

Appendix E, New Components The source code for the implemented components is presented in this chapter.

1.3 VehProLib - Vehicle Propulsion Library

For studying vehicle behaviour, it has been of interest to make models of powertrains and its components (such as a turbocharger, engine, wheels, gearbox and chassis). At the Vehicular System department [Öberg, 2015] at Linköping University, Lars Eriksson has developed a modeling library for this kind of system. The library is called VehProLib [Eriksson, 2003] and stands for Vehicle Propulsion Library.

Due to unfortunate circumstances the library was not runnable due to a merging conflict during the last project using the library. This will be explained in further detail in section 2.1.

1.4 Scientific Foundation

The sections below covers some necessary technical areas.

1.4.1 Internal Combustion Engines

Engines are used in a lot of applications, for this report the focus is on engines used in the automotive industry, mainly four cylinder engines. By consuming air and fuel the engine can produce work for vehicle propulsion, there is also waste heat and exhaust gases produced in the process [Eriksson and Nielsen, 2014] and [Uwe and Nielsen, 2005]. The engine consists of several sub-systems to make this process possible, such as throttle, cylinders, manifolds and fuel injection.
To build efficient engines with as little environmental impact as possible a good way is to build simulation models. This makes it possible to test different compositions under many different circumstances which would be hard, time consuming and a lot more expensive without simulations. There are mainly two ways to simulate engines, they are presented below.

In-Cylinder Engine Modeling

In-cylinder engine models are used for understanding and analysing the complex phenomena happens inside the cylinder under one cycle, chapter 5 in [Eriksson and Nielsen, 2014], [Merker et al., 2006] and [Öberg, 2009]. Typically the resolution is in the order of one crank angle degree. This is very time consuming due to the small simulation time step (the engine shaft rotates between 800 and 6000 revolutions per minute, this means between 288000 and 2160000 time steps per minute or between 4800 and 36000 time steps per second).

Mean Value Engine Modeling

Instead of describing variations as detailed as in the case of in-cylinder, Mean Value Engine Modeling (MVEM) describes variations over one to several cycles meaning that the signals, parameters and variables are considered averages over one to several cycles [Heywood, 1988], chapter 7 in [Eriksson and Nielsen, 2014] and [Andersson, 2005]. The consequence of this is the increase in simulation speed and detail loss, the simulations are well-suited in larger simulation environments (both in time and complexity).

1.4.2 Turbocharger

The work produced by the engine is based on the amount of fuel that can be efficiently burned inside the cylinder chamber [Eriksson and Nielsen, 2014]. The burning is limited by the amount of air which can be induced. A turbocharger compresses the intake air to higher densities and therefore increase the efficiency of which the fuel is burned inside the cylinder chamber [Dixon and Hall, 2013]. This allows for downsizing since smaller engines have a lower fuel consumption and the turbocharger allows for more work produced due to the higher intake density [Spring et al., 2007]. Two properties that are highly desirable in today's automotive industry.

1.5 Resources

Throughout the thesis work several tools and other resources have been used and are presented in the sections below.

1.5.1 Modelica

Modelica [Association, 2015] is a multi-domain modeling language, which is declarative and object oriented. It can model complex systems that consist of components from various domains, such as electrical, electronic, mechanical, thermal and hydraulic.

1.5.2 Wolfram SystemModeler

Wolfram SystemModeler (WSM) [Research, 2015c] is a modeling tool for engineering as well as life science. It is developed by Wolfram MathCore and is based on the modeling language Modelica. WSM features an interactive graphical interface, customizable components and possibilities to create new components. WSM is used for all the simulations and creation of models throughout the thesis and is provided by Wolfram MathCore.

1.5.3 Mathematica

Mathematica [Research, 2015b] is a mathematical (based on symbolic mathematics) tool developed by Wolfram Research in 1988 (conceived by Stephen Wolfram the CEO of Wolfram Research). Mathematica is mainly used as a tool in scientific and engineering fields, and uses both numerical and symbolical mathematics (no limitation for mixing the two kinds). Mathematica is used as a tool for visualisation and fitting models to real data throughout the thesis. Mathematica is provided by Linköping University (Wolfram MathCore would provide this if this was not already provided).

1.5.4 Simulink

Simulink [MathWorks, 2015b] is a graphical programming language for simulation and modeling developed by MathWorks [MathWorks, 2015a]. It has a block diagram environment which can make multi-domain simulations. A major difference between Modelica and Simulink is that Simulink mainly uses a causal modeling environment. Simulink provides the reference models used for validation.

2 Methodology

2.1 Library Restoration

Original VehProLib (presented in [Eriksson, 2003]) was created by mainly two different tools, Dymola [Systèmes, 2015] and OpenModelica [OpenModelica, 2015]. It is not certain WSM supports the same functionality and structure as these two tools and some of the syntax is obsolete. In three previous projects, it has been intended to further develop the library (for example the addition of the hybrid electrical vehicle package) and restructure the layout. At the end of the projects there were merge conflicts. Due to lack of time at the end of projects, the merge conflicts were not taken care of. All things together resulted in a library which is not runnable in WSM. The following sections describes the changes and the problems related to the restoration of a runnable VehProLib in WSM. Information about the existing components were mainly gathered from [Eriksson and Nielsen, 2014], [Pettersson, 2000] and [Montell, 2004].

More detailed information about the investigation can be found in appendix section B.

2.1.1 Systematic Approach

The first step in the restoration process is to ensure all the cross references are correct, this means that commands such as *extends* and the inclusion of external models in a model should be correct. In WSM it is possible to validate a model, this gives a fast feedback about the status of the model and external models which are included as components. Validation does not ensure runnable simulations, but runnable simulation gives a successful validation. The order which the models were validated, was from the lowest orders of dependence on other models to the highest. This was followed where possible.

Sometimes validation called for missing models, which do not exists due to updates in MSL. These links where replaced by the corresponding components in this version of MSL, or with components which provides the required task. Two examples are the **`In/OutPort`** and **`GearEfficiency`** see appendix section B.1.1 and B.2.2 respectively.
When all models were validated successfully, test cases were implemented and existing ones were reused, which are now located in the new test library Vehicle Propulsion Test Library (VehProTestLib, more about the test library in section 2.3). Basic models such as **`IdealDifferential`** were tested without reference data. But more complex models such as **`MvemEngine`** were tested against the corresponding model in Simulink.

2.1.2 Simulink as Reference Data

Simulink has been and is still used as the main modeling tool at Vehicular Systems. The models are well fitted to real data and are used for education and scientific purposes at the university. That said, the model used as a reference is not always as advanced as the one in VehProLib. For instance, the models with complete drive cycles have no dynamic temperature behaviour. The general behaviour for other quantities such as torque and pressure are to be considered valid.

2.2 Library Layout

Some concepts used throughout the library design process are stated below:

Package A classification of packages and/or components/examples.

Main package The top level of classification in the library, contains sub-packages and/or components/examples.

Sub-package The lower level (two or higher) of classification in the library, contains sub-packages of lower level than itself and/or components/examples.

Template An incomplete component, containing the basic models which are in common for all components of this type. Called partial model in Modelica.

The VehProLib had a structure which can be seen in figure C.1, where examples were located in several places such as Example, Test and Engine.Example.

The design of VehProLib was investigated in two ways. One was by doing a survey involving people using WSM on a daily basis and the responsible person for VehProLib at Vehicular Systems. The other one was studying three existing libraries, two internally developed libraries by Wolfram Mathcore and the Modelica Standard Library (MSL). The two internally developed libraries were:

Hydaulic A commercial library for modeling hydraulic systems [MathCore, 2015a].

ModelPlug A free to use library for connecting the simulation with the real-world through an Arduino [MathCore, 2015b].

The three libraries are discussed in further detail in sections C.1, C.2 and C.3

The detailed information about the investigation can be found in section C.

2.2.1 Connection to Modeling and Control of Engines and Drivelines

VehProLib is intended to be connected to *Modeling and Control of Engines and Drivelines* by Lars Eriksson and Lars Nielsen [Eriksson and Nielsen, 2014]. The two medium would complement each other, when reading the book it would be possible to simulate examples and/or construct models from the book in the library, and the models existing in the library would be referred to and explained in the book.
Therefore all models in the new and the modified components are taken from the book. The reference data from Simulink (see section 2.1.2) uses models from the book as well.

2.3 VehProTestLib - Vehicle Propulsion Test Library

A test library is used for validating the actual library. When changes are made the tests can give information of what has changed and what is the same. Depending on the results, the new implementation could modify something which would change the simulations. This means that the new code or the old code has some errors. This makes it important to have good test cases and code coverage.

The investigation of the design on the VehProTestLib is based on the investigation of VehProLib, with some specific questions in the survey regarding test models and validation data. There was no access to existing test libraries.

2.4 Gas Model

VehProLib uses a replaceable gas model which functions as a register of the individual gas data, such as molar mass, specific volume, specific heat capacity at constant volume etcetera. It keeps track of changes in gas pressure, temperature, mass fraction among others and uses this for the calculation of h, $R_{specific}$, γ and ρ as follows:

$$h = u + pv \tag{2.1}$$

$$R_{specific} = \frac{R}{M} \tag{2.2}$$

$$\gamma = \frac{c_p}{c_v} \tag{2.3}$$

$$\rho v = 1 \tag{2.4}$$

where h is the mass specific enthalpy, u is the mass specific internal energy, p is the gas pressure, v is the specific volume, $R_{specific}$ is the specific gas constant, R is the gas constant, M is the molar mass, γ is the ratio of specific heats, c_p is the specific heat capacity at constant pressure, c_v is the specific heat capacity at constant volume and ρ is the gas density.

When the relations do not include any assumption of ideal gas, c_v and c_p are calculated from (2.1) as follows:

$$\left(\frac{\partial h}{\partial T}\right)_v = \left(\frac{\partial u}{\partial T}\right)_v = c_v \tag{2.5}$$

$$\left(\frac{\partial h}{\partial T}\right)_p = \left(\frac{\partial u}{\partial T}\right)_p = c_p \tag{2.6}$$

2.4.1 Connectors

The gas-related information inside the engine are transferred with help of connectors using a zero-dimensional (no spatial dimension dependence) environment. The connector is called **`FlowCut`** and is developed within VehProLib. A connector connects three variables:

- T - Temperature
- p - Pressure
- $x[n]$ - Mass fraction, vector of n components

and three flow variables:

- H - Enthalpy flow
- W - Total mass flow rate
- $W_x[n]$ - Mass flow rate, vector of n components

The connector has no dependency on the gas model initially, by creating pins which include both the gas model and connectors, a gas dependency can be created. This dependency is based on coupling the dimension (n) of the connectors to the dimension of the gas (number of gas components).

2.4.2 Gas Relations

The gas model does not describe interactions between connection points in the zero-dimensional environment, meaning that energy conservation, mass balance and interaction between pressure and temperature must be declared in the models using the gas model. This give rise to a problem when it would be of interest to change the equation of states. If one would like to switch from the ideal gas model to another equation of states such as Van der Waals Equation (both models further explained in section 2.5). This change would mean that new components for the control volume and engine flow would need to be introduced and possibly other components as well.

2.5 Non-ideal Gas Model

When modeling the physics inside the combustion chamber, the ideal gas law is commonly used (see equation 2.7). When a turbocharger is used, the pressure and the temperature increases inside the chamber leading to questions regarding molecular interaction and the impact magnitude coming from the size of the molecules. The ideal gas law assumes zero molecular interaction forces and the molecules can be seen as perfect hard spheres, where the collisions are perfectly elastic.
Ideal gas law:

$$pV = nRT \tag{2.7}$$

<u>Note:</u> The ideal gas law is a good approximation for the behaviour of many gases in a variety of environments.

2.5.1 Van der Waals Equation

The Van der Waals equation (2.8) is also an approximation of gas behaviour, but compared to the ideal gas law (2.7) the Van der Waals equation takes molecular interaction and the size of the molecules into account.
Van der Waals equation:

$$\left(p + a\left(\frac{n}{V}\right)^2\right)(V - nb) = nRT \tag{2.8}$$

Where p is the pressure, a is the attraction between particles, n is the number of moles (the number of particles is $N = N_A n$, where N_A is the number of particles per mole), V is the volume, b is the volume excluded by a mole of particles, R is the gas constant and T is the temperature.
a and b are often taken from experiments but can be derived using the critical point (the point p_c and V_c where liquid and vapour can coexist) for which $\left(\frac{\partial p}{\partial V}\right)_T = 0$ and $\left(\frac{\partial^2 p}{\partial V^2}\right)_T = 0$ (see section A in appendix) the results are from equation (A.9)

and (A.10) respectively:

$$a = \frac{27T_c^2R^2}{64p_c} \tag{2.9}$$

$$b = \frac{T_cR}{8p_c} \tag{2.10}$$

Where T_c is the critical temperature, p_c is the critical pressure and R is the gas constant.

It is worth noting that the Van der Waals equation (2.8) becomes the ideal gas law (2.7) when $a, b \to 0$. If this was not the case, then the Van der Waals equation would not be of importance.

2.6 Template Models

The library uses a syntax allowing one model to extend another. This reduces duplicates of basic equations and interfaces, making consistent development easier. There is no restriction to extend only one model. By declaring a model *partial* means that it cannot be used on its own and must be extended by another model to be allowed to simulate.

Note: A partial model extending a model still have the same properties as a partial model.

Almost all components in the turbocharger package use a template (partial model) which is a partial model containing all the common features for the general component.

2.7 Turbocharger

The turbocharger is a sub-package to the engine package. It contains all the components related to a turbocharger, such as turbine, compressor and turbo shaft.

All the turbocharger components are using a template structure which is a partial model (incomplete on its own, but containing all the basic equations) that a final component can be constructed upon by defining the missing equations. These equations are fitted according to measurements from a real turbocharger with help of Mathematica (see section 2.9.1).

Both the compressor and the turbine can be seen as a restrictions, this means that the mass flow rate (`W`) and the enthalpy flow (`H`) is the same in both **`FlowCut`** connectors (the engine components are nodes in the zero-dimensional environment). The following Modelica code describes the relation of the flows where the mass flow (`m_dot`) is decided by the compressor/turbine model:

```
i.W  = -o.W;
i.H  = -o.H;
i.Wx = -o.Wx;
i.Wx = m_dot * g.x;
i.W  = sum(i.Wx);
i.H  = i.W * g.h;
```

where `i` is the in connector and `o` is the out connector. `g` is the gas inside the component and `x` means mass fraction.

2.7.1 Compressor

A compressor uses power $\left(\dot{W}_c\right)$ from the turbo shaft to compress the incoming fluid to a higher temperature and pressure as can bee seen in figure 2.1.

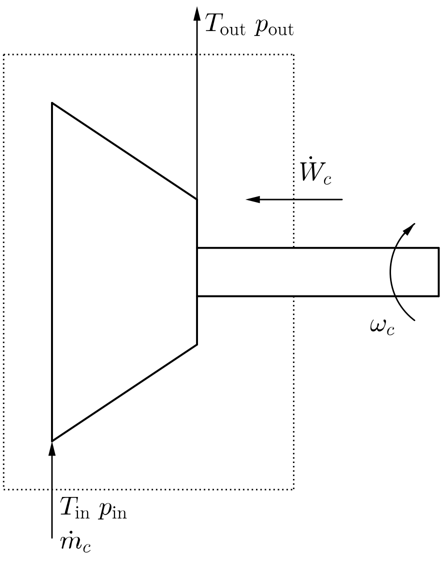

$T_{\text{out}}\ p_{\text{out}}$
$\dot{W}_c$
ω_c
$T_{\text{in}}\ p_{\text{in}}$
$\dot{m}_c$

Figure 2.1: *A schematic figure of a compressor, inspired by figure 8.4 in [Eriksson and Nielsen, 2014], where T is the temperature, p is the pressure, $\dot{m}_c$ is the mass flow, $\dot{W}_c$ is the power coming from the shaft and ω_c is the angular velocity of the shaft.*

The model which is used for the compressor is called the Generic Turbine Model (Model 8.1 in [Eriksson and Nielsen, 2014]) which is suited for the MVEM framework.

$$\dot{m}_c = f_{\dot{m},c}(p_{in}, p_{out}, T_{in}, \omega_c) \tag{2.11}$$

$$\eta_c = f_{\eta,c}(p_{in}, p_{out}, T_{in}, \omega_c) \tag{2.12}$$

$$T_c = T_{in} + \frac{T_{in}}{\eta_c}\left(\left(\frac{p_{out}}{p_{in}}\right)^{\frac{\gamma-1}{\gamma}} - 1\right) \tag{2.13}$$

$$\dot{W}_c = \dot{m}_c c_p (T_c - T_{in}) \tag{2.14}$$

where $\dot{m}_c$ is the mass flow, η_c is the compressor efficiency, $f_{\dot{m},c}$ and $f_{\eta,c}$ are functions that estimate the behaviour of a compressor map (a chart based on a test rig or simulation result), p is the pressure, T is the temperature, ω_c is the angular velocity of the shaft, γ is the ratio of specific heats, $\dot{W}_c$ is the power coming from the shaft and c_p is the specific heat capacity at constant pressure.

The functions $f_{\dot{m},c}$ and $f_{\eta,c}$ are chosen to be the same as in the Simulink reference model for comparing purposes, more information can be found in section 2.2.1. The efficiency model for the complete compressor model is chosen as the Quadratic form Compressor Efficiency with mass flow rate modeled as the Ellipse Extended to Restriction Region (Model 8.5 and 8.9 in [Eriksson and Nielsen, 2014]).

$$\Pi = \frac{p_{out}}{p_{in}} \tag{2.15}$$

$$\Pi_{max} = \left(\frac{N^2 D_c^2 * \Psi_{max}}{2 c_p T_{in}} + 1 \right)^{\frac{\gamma}{\gamma-1}} \tag{2.16}$$

$$\dot{m}_{corr} = \dot{m}_{corr,max} \sqrt{1 - \left(\frac{\Pi}{\Pi_{max}} \right)^2} \tag{2.17}$$

$$\chi = \begin{pmatrix} \dot{m}_{corr} - \dot{m}_{corr\ at\ \eta\ max} \\ \sqrt{\Pi - 1} - \sqrt{\Pi_{at\ \eta\ max} - 1} \end{pmatrix} \tag{2.18}$$

$$f_{\dot{m},c}(p_{in}, p_{out}, T_{in}, \omega_c) = \dot{m}_{corr} \frac{p_{in}/p_{ref}}{\sqrt{T_{in}/T_{ref}}} \tag{2.19}$$

$$f_{\eta,c}(p_{in}, p_{out}, T_{in}, \omega_c) = \max(\eta_{max} - \chi^t Q_\eta \chi, \eta_{min}) \tag{2.20}$$

where Π is the pressure ratio, N is the turbo shaft speed in RPM, D_c is the diameter of the compressor, Π_{max} is the maximum pressure ratio for a specific speed and Ψ_{max}, $\dot{m}_{corr,max}$, $\dot{m}_{corr\ at\ \eta\ max}$, $\Pi_{at\ \eta\ max}$, η_{max}, Q_η, η_{min} are parameters which should be fitted to measure data.

2.7.2 Turbine

A compressor creates power $\left(\dot{W}_t\right)$ to the turbo shaft by expanding the incoming fluid to a lower temperature and pressure as can bee seen in figure 2.2.

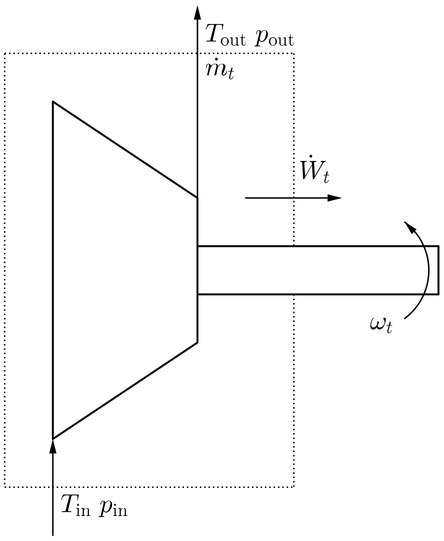

T_{out} p_{out}
$\dot{m}_t$
$\dot{W}_t$
ω_t
T_{in} p_{in}

Figure 2.2: *A schematic figure of a turbine, inspired by figure 8.4 in [Eriksson and Nielsen, 2014], where T is the temperature, p is the pressure, $\dot{m}_t$ is the mass flow, $\dot{W}_t$ is the power going to the shaft and ω_t is the angular velocity of the shaft.*

The model which is used for the turbine is called the Generic Turbine Model (Model 8.2 in [Eriksson and Nielsen, 2014]) which is suited for the MVEM framework.

$$\dot{m}_t = f_{\dot{m},t}(p_{in}, p_{out}, T_{in}, \omega_t) \tag{2.21}$$

$$\eta_t = f_{\eta,t}(p_{in}, p_{out}, T_{in}, \omega_t) \tag{2.22}$$

$$T_t = T_{in} - \eta_t T_{in} \left(1 - \left(\frac{p_{out}}{p_{in}} \right)^{\frac{\gamma-1}{\gamma}} \right) \tag{2.23}$$

$$\dot{W}_t = \dot{m}_t c_p (T_{in} - T_t) \tag{2.24}$$

where $\dot{m}_t$ is the mass flow, η_t is the compressor efficiency, $f_{\dot{m},t}$ and $f_{\eta,t}$ are functions that estimate the behaviour of a compressor map (a chart based of a test rig or simulation result), p is the pressure, T is the temperature, ω_t is the angular velocity of the shaft, γ is the ratio of specific heats, $\dot{W}_t$ is the power coming from the shaft and c_p is the specific heat capacity at constant pressure.

The functions $f_{\dot{m},t}$ and $f_{\eta,t}$ are chosen to be the same as in the Simulink reference model for comparing purposes, more information can be found in section 2.2.1. The efficiency model for the complete turbine model is chosen as the Turbine Efficiency with Blade Speed Ratio (BSR) with mass flow rate modeled as the Modified Square Root Turbine Flow (Model 8.16 and 8.14 in [Eriksson and Nielsen, 2014]). The mass flow rate is modeled with help of Turbine Flow Parameter (TFP) [Systems, 2014].

$$\Pi = \frac{p_{out}}{p_{in}} \tag{2.25}$$

$$BSR = \frac{r_t \omega_t}{\sqrt{2 c_p T_{in} \left(1 - \Pi^{\frac{\gamma-1}{\gamma}}\right)}} \tag{2.26}$$

$$TFP = TFP_{max} \sqrt{1 - \Pi^{TFP_{exp}}} \tag{2.27}$$

$$f_{\dot{m},t}(p_{in}, p_{out}, T_{in}, \omega_t) = \frac{1000\ TFP}{\sqrt{T_{out}} p_{out}} \tag{2.28}$$

$$f_{\eta,t}(p_{in}, p_{out}, T_{in}, \omega_t) = \eta_{max} \left(1 - \left(\frac{BSR - BSR_{max}}{BSR_{max}}\right)^2\right) \tag{2.29}$$

where Π is the pressure ratio, BSR is the Blade Speed Ratio, r_t is the radius of the turbine, TFP is the Turbine Flow Parameter and TFP_{max}, TFP_{exp}, η_{max}, BSR_{max} are parameters which should be fitted to measure data.

2.7.3 Turbo Shaft

The turbo shaft is the connection between the turbine and the compressor as can be seen in figure 2.3.

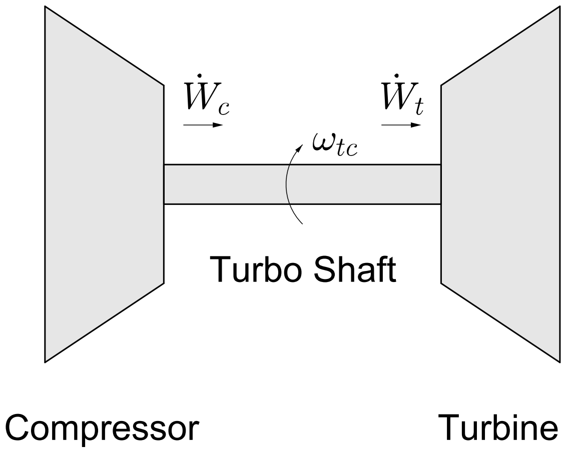


Figure 2.3: *A schematic figure of the turbo shaft, where $\dot{W}_t$ is the power going to the shaft, $\dot{W}_c$ is the power coming from the shaft and ω_{tc} is the angular velocity of the shaft.*

The turbo shaft is modeled as a rotational inertia with a mechanical loss using Model 8.2 in [Eriksson and Nielsen, 2014].

$$J_{tc} * \dot{\omega}_{tc} = \tau_t - \tau_c - \tau_{fric}(\omega_{tc}) \tag{2.30}$$

$$\dot{\omega}_{tc} = \frac{d}{dt}(\omega_{tc}) \tag{2.31}$$

$$\tau_i = \frac{\dot{W}_i}{\omega_{tc}}, \; i = t, c \tag{2.32}$$

where J_{tc} is the inertia, α_{tc} is the angular acceleration, τ is the torque, ω_{tc} is the angular velocity and $\dot{W}$ is the power.
The friction torque τ_{fric} is often modeled as a function of linear or quadratic angular velocity. Simulink uses a linear dependency of angular velocity for the friction torque and therefore is the model used in VehProLib (for futher deatails see section 2.2.1) as well.

$$\tau_{fric} = c_{fric}\,\omega \tag{2.33}$$

where c_{fric} is the friction coifficient for the friction torque.

2.7.4 Wastegate

The wastegate is modeled as a standard compressible restriction with variable effective area. Area changes are modeled with a delay:

$$\frac{d}{dt}(A) = \frac{1}{T}\left(A_{ref} - A\right); \tag{2.34}$$

where A_{ref} is the signal into the wastegate and T is the time delay.
The component called standard compressible (turbulent) restriction with variable effective area is an existing component in VehProLib which is modeled using Model 7.4 in [Eriksson and Nielsen, 2014]:

$$\Pi\left(\frac{p_{ds}}{p_{us}}\right) = \max\left(\frac{p_{ds}}{p_{us}}, \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}\right) \tag{2.35}$$

$$\Psi_0(\Pi) = \sqrt{\frac{2\gamma}{\gamma-1}\left(\Pi^{\frac{2}{\gamma}} - \Pi^{\frac{\gamma+1}{\gamma}}\right)} \tag{2.36}$$

$$\Psi(\Pi) = \begin{cases} \Psi_0(\Pi) & \Pi \le \Pi_{linear} \\ \Psi_0(\Pi_{linear})\frac{1-\Pi}{1-\Pi_{linear}} & otherwise \end{cases} \tag{2.37}$$

$$\dot{m}(\alpha, p_{us}, p_{ds}, t_{us}) = \frac{p_{us}}{\sqrt{R_{specific}T_{us}}} A C_D \Psi(\Pi) \tag{2.38}$$

where Π is the pressure ratio with lower limit, p is the pressure, T is the temperature, us is the upstream quantity which is coming from the component before (this can be viewed as a chain of components), ds is the downstream quantity which is the coming from the component after, γ is the ratio of specific heats, Π_{linear} is the Π value for which Ψ should be seen as linear, $R_{specific}$ is the specific gas constant, A is the cross-section area of the restriction and C_D is the discharge coefficient.

Note: The notation of up/downstream is the special case of the quantities in each connector when looking at causal signals. Modelica does not need to model physical systems causally.

2.7.5 Intercooler

An intercooler is used in turbocharged Spark-Ignition (SI) and in Compression-Ignition (CI) engines to cool the compressed air after the compressor. This results in higher engine efficiency and higher power output due to the increase in the air density.

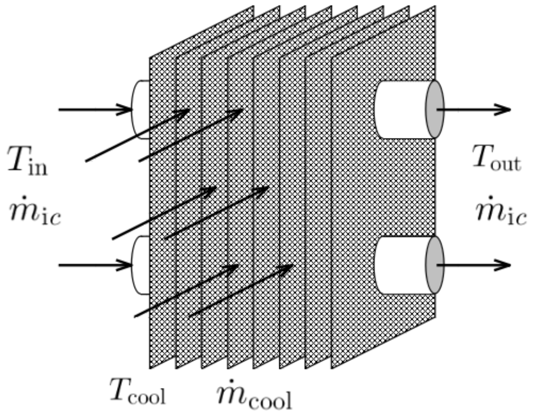


Figure 2.4: *A schematic figure of an intercooler, Inspired by figure 7.43 in [Eriksson and Nielsen, 2014]. Where T is the temperature and $\dot{m}$ is the mass flow.*

Equations for the intercooler (equation (7.69) in [Eriksson and Nielsen, 2014]):

$$T_{out} = T_{in} - \epsilon\left(\dot{m}_{cool}, \dot{m}_{ic}\right)\left(T_{in} - T_{cool}\right) \tag{2.39}$$

where T is the temperature, $\dot{m}$ is the mass flow and ϵ is the intercooler efficiency.

It is important to understand that a temperature change is coupled to enthalpy, meaning that the enthalpy flow should be changed accordantly. By letting each connector with a non-zero flow contain a gas and setting the temperature of each gas, the specific enthalpy of the gas changes accordantly. The enthalpy flow difference is then defined by the specific enthalpy of the gases and the mass flow:

$$H_{gas,in} = \frac{\dot{H}_{in}}{\dot{m}_{ic}} \tag{2.40}$$

$$H_{gas,out} = c_p \ T_{gas,out} \tag{2.41}$$

$$\dot{H}_{in} - \dot{H}_{out} = \dot{m}_{ic}\left(H_{gas,in} - H_{gas,out}\right) \tag{2.42}$$

where H is the specific enthalpy, $\dot{H}$ is the enthalpy flow (in the connector), $\dot{m}$ is the mass flow rate, c_p is the specific heat capacity at constant pressure and T is the temperature.

2.7.6 Sensors

A sensor is measuring one or more quantities from a given connector. It is important to set the flow to zero for the sensors with one connector, because otherwise the sensor influences the flow. The flows in a connector point [Mod, 2014] behave in the following way:

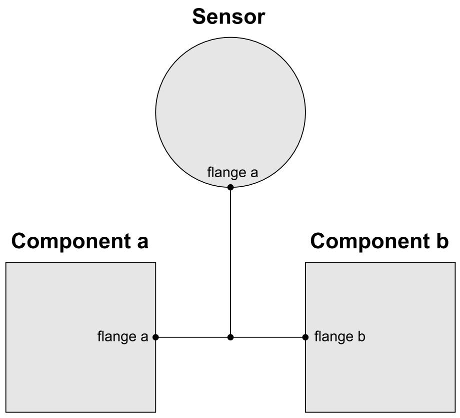


Figure 2.5: A simple example using MSL mechanical rotational connectors (angle as variable and torque as flow variable) flange a and flange b.

```
  connect(component_a.flange_a, component_b.flange_b)
  connect(component_a.flange_a, sensor.flange_a)
```

It yields the following equations:

```
equation
  component_a.flange_a.phi = component_b.flange_b.phi;
  component_a.flange_a.phi = sensor.flange_.phi;
  0 = component_a.flange_a.tau + component_b.flange_b.tau
    + sensor.flange_a.tau;
```

Sensors that measure quantities from the **`FlowCut`** connectors (see section 2.4.1) use the same interface as all gas related components in the library, meaning that gas inside the sensor must be given a temperature, pressure and mass fraction.

2.7.7 Incompressible Restriction

When modeling restrictions such as air filter, exhaust system and intercooler, an incompressible turbulent restriction is often used see section 7.2 in [Eriksson and

Nielsen, 2014].

$$\Delta_p = |p_{in} - p_{out}| \tag{2.43}$$

$$\dot{m}(p_{in}, T_{in}, p_{out}) = \begin{cases} \sqrt{\frac{p_{in}}{HT_{in}}}\sqrt{\Delta_p} & \Delta_p > p_{lin} \\ \sqrt{\frac{p_{in}}{HT_{in}}}\frac{\Delta_p}{\sqrt{p_{lin}}} & otherwise \end{cases} \tag{2.44}$$

where p_{lin} is the is the pressure limit for which the mass flow is considered linear, H is the flow resistance.

2.8 Boost Controller

A **`BoostController`** are controlling the movements of the throttle and the wastegate plates. This is done by measuring the gas pedal position, the engine speed, the intercooler pressure and the intake manifold pressure. The set up consists of two PI controllers, one interpretation block for the gas pedal position and one forward model of the throttle.This can be seen in figure 2.6.

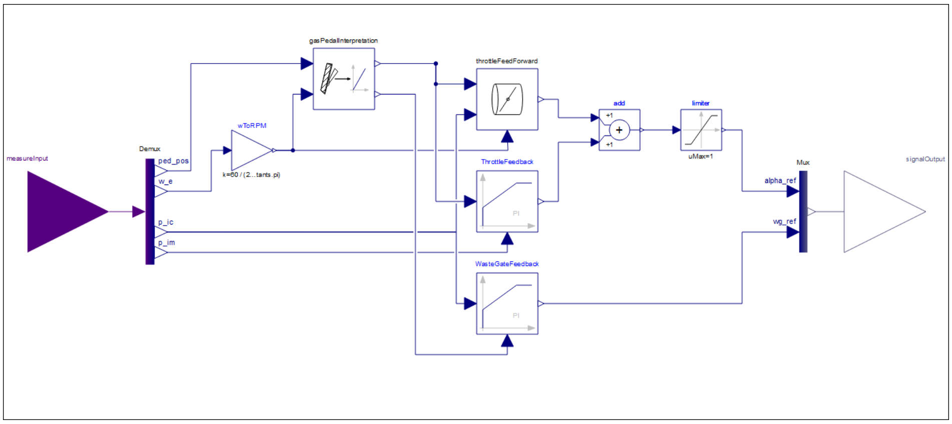


Figure 2.6: *The boost controller model seen in the diagram view in WSM.*

The equations and the interpretation of the **`BoostController`** is stated in project compendium used in TSFS09 (Modelling and Control of Engines and Drivelines) [Systems, 2014].

2.8.1 Gas Pedal Interpretation

The **`GasPedalInterpretation`** block estimates the intercooler and intake manifold pressured ($p_{im,ref}$ and $p_{ic,ref}$) based on the gas pedal position (f_{pedal}) and the engine speed (N).

The engine torque (τ_{ref}) is estimated using an interpolation table based of stationary values for different operation points (engine speed N_{rpm})

$$\tau_{ref} = \begin{cases} \tau_{max}(N_{rpm})\frac{f_{pedal}-0.1}{1-0.1}, & f_{pedal} > 0.1 \\ \\ \tau_{min}(N_{rpm})\frac{0.1-f_{pedal}}{0.1-0}, & otherwise \end{cases} \tag{2.45}$$

where f_{pedal} is the pedal position $\in [0, 1]$ and $\tau_{max/min}(N_{rpm})$ is the engine torque for a specific engine speed N_{rpm}.
The Breaking Mean Effective Pressure is estimated as equation (7.54) in [Eriksson and Nielsen, 2014] where the engine torque is estimated as τ_{ref}.

$$BMEP_{ref} = \frac{2\pi\ n_r\ \tau_{ref}}{V_D} \tag{2.46}$$

where n_r is the number of revolutions per stroke and V_D is the total engine displacement volume.
The Breaking Mean Effective Pressure is used to estimate the pressure in the intake manifold ($p_{im,ref}$), the pressure difference between the intercooler ($p_{ic,ref}$) and the intake manifold is set to a constant (Δp_{ref}).

$$p_{im,ref} = \frac{BMEP_{ref} + c_0}{c_1} \tag{2.47}$$

$$p_{ic,ref} = p_{im,ref} + \Delta p_{ref} \tag{2.48}$$

where c_0 and c_1 are fitting parameters.

2.8.2 Throttle Feed Forward

The `ThrottleFeedForward` block estimates the throttle plate position based of the engine speed, actual intercooler pressure and the estimated intake manifold pressure.
The volumetric efficiency (η_v) is estimated using a first order polynomial function:

$$\eta_v = c_{\eta,0} + c_{\eta,1}\sqrt{N_{rps}} + c_{\eta,2}\sqrt{p_{im,ref}} \tag{2.49}$$

where $c_{\eta,i},\ i = 1, 2, 3$ are fitting parameters, N_{rps} is the engine speed in revolutions per second and $p_{im,ref}$ is the estimated intake manifold pressure.
The mass flow is estimated using equation (7.13) in [Eriksson and Nielsen, 2014]:

$$\dot{m}_{ref} = \frac{V_D\ N_{rps}\ \eta_v\ p_{im,ref}}{R\ T_{im}\ n_r} \tag{2.50}$$

where V_D is the total engine displacement volume, R is the specific gas constant and T_{im} is the intake manifold temperature.

The pressure ratio is estimated using the estimated intake manifold pressure. Otherwise are estimated pressure ratio (Π_{ref}), Ψ_{ref} and estimated throttle area (A_{ref}) calculated as described in section 2.7.4.

$$\Pi_{ref} = \max\left(\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}, \frac{p_{im,ref}}{\max(p_{ic}, p_{im,ref})}\right) \tag{2.51}$$

$$\Psi_{ref} = \sqrt{\frac{2\gamma}{\gamma-1}\left(\Pi_{ref}^{\frac{2}{\gamma}} - \Pi_{ref}^{\frac{\gamma+1}{\gamma}}\right)} \tag{2.52}$$

$$A_{ref} = \frac{\dot{m}_{ref}}{C_d\, \Psi_{ref}\, p_{ic}}\sqrt{R\, T_{im}} \tag{2.53}$$

where γ is the ratio of specific heats, p_{ic} is the intercooler pressure and C_d is the discharge coefficient.
The estimated throttle angle (α_{ref}) is calculated by inverting the polynomial expression for the throttle area: $A = a_0 + a_1\alpha + a_2\alpha^2$ and choosing the positive root.

$$\alpha_{ref} = -\frac{1}{2a_2} + \sqrt{\left(\frac{a_1}{2a_2}\right)^2 + \frac{A_{ref} - a_0}{a_2}} \tag{2.54}$$

where $a_i,\ i = 0, 1, 2$ are fitting parameters.

2.9 Mathematical Modeling in Mathematica

Mathematica is a good tool for mathematical modeling of components and fitting parameters to real data. Using the WSMLink created by Wolfram MathCore, one can use predefined functions for interaction (such as simulate, plot, fitting model parameters etcetera) with WSM. This means that it is possible to mathematically model a component in Mathematica to fit model parameters to real data and then transfer this information to WSM.

2.9.1 Parameter Optimization

There are several functions for fitting parameters to data. The function used in this thesis is called NMinimize which is used on a table of normalized differences between the model and the real data.
Small example, fit a logarithmic model to the first ten primes:

```
x = Range[10];
data = Table[Prime[x], {x, 10}];
model = a x Log[b + c x];
solution = NMinimize[Norm[data - model], {a, b, c}];
```

The reason NMinimize was chosen instead of for example FindFit is that FindFit finds complex solutions and get stuck there. This could be due to too few data points or invalid domain or something else, the documentation is not detailed in this respect.

2.9.2 Handling Models in Mathematica

Mathematica has support for creating and modifying models as well as extracting data and equations from models in WSM.
When setting parameters in WSM the the function WSMSetValues can be used and similar when extracting data from a model in WSM, WSMModelData can be used.
A model can be created using the function WSMCreateModel which converts the mathematical model to a Modelica model. The model creation supports template (partial model) structure (as discussed in section 2.6). WSMCreateModel is not used in this thesis due to the limitations of names of variables in Mathematica. Mathematica does not allow for variable names containing "_" because this is a special symbol called Blank[] used for expressions. At a later instance a solution to this problem was found, another symbol called \[UnderBracket] can be used instead.

2.10 Industry Example

An industry example is a promotion for a feature, library or solution through an example posted on Wolfram SystemModeler homepage under the section Industry Example [Research, 2015a].

The industry example for VehProLib is chosen as example 8.9 (turbocharged SI engine) in [Eriksson and Nielsen, 2014]. To promote the usability and the physical significance of the library, two examples are shown and compared. The only difference between the two examples is the engine, one larger engine (2.3 litre) without turbocharger and one smaller engine (1.2 litre) with turbocharger. This is to show how a turbocharger can be used to downsize an engine. To put everything into prospective, two engines are placed in a vehicle following the New European Drive Cycle (NEDC). NEDC is a table consisting of demanded speed, selected gear and clutch position and is designed to estimate a typical car usage in Europe. Fuel consumption and the deviation of speed from the drive cycle is compared for the two examples.

In addition to the engine comparison, there is a small section covering how the models (example is the compressor) can be fitted to real data using the mathematical tool Mathematica (further discussed in section 3.7), and how the to transfer these parameters to WSM.

3 Result

3.1 Library Restoration

The library is for the most part restored to its original functionality with a few exceptions, which is presented in section 3.2. The functionality is demonstrated by an example consisting of complete vehicle from chassis to engine.

3.1.1 Validation Warnings

When most of the engine components are validated there is a warning calling for missing corresponding `inner` component, this is intentional. The (replaceable) gas model is declared in the engine model as `inner`, which makes the warning disappear. The gas model contains equations which relates specific quantities of the gas model to each other (see section 2.4). When a component without an `inner` component is validated these equations is called missed.

3.2 Library Restoration - Remaining Work

Many of the problems have been sorted out, there are some left which is presented in the sections below.

3.2.1 In-Cylinder Model

The in-cylinder model is working, but missing a controller for lambda (lambda is set to constant 1). Otto Montell's master thesis work [Montell, 2004] give a good overview of the concepts of the in-cylinder model. This means that the number of equations are not consistent for the current solution. The engine template (partial model) which is used for all types of engines have a controller connector. And

that controller connector demands a signal from the connector which the cylinder model is supposed to have, but is not yet implemented in the in-cylinder model. The conclusion is that for one cylinder it is possible to fake that connector signal, but when there is another number of cylinders the engine model got the wrong number of equations because it is not possible to fake multiple connections lines by instantiation.
The best solution to this problem is to implement the controller for the cylinder model, this will not be done during the timespan of this thesis.

3.2.2 Unbalanced Connectors

The connector **`FlowCut_liquid`** does not follow the size restriction, it has two variables and one flow variable. 9.3.1 in Modelica Specification [Mod, 2014] says that for each non-partial connector class the number of flow variables and variables shall be equal. This problem could be sorted out by following the size restriction and either remove AFs (Stoichiometric air to fuel ratio) or q_LHF (lower heating value) and place it in a more suitable place (the values are parameters which do not change during the simulation).

The Geometry connector has the same problem, it has six variables and zero flow variables. This could be solved by using `Records`, presented in section 3.2.3.

3.2.3 Signal Bus

The current solution for handling the signal bus for lambda and the idle controller are standard acausal connectors, which are redeclared in several instances. The specification [Mod, 2014] claims acausal connectors to be balanced (see section 3.2.2), this is not the case for the signal bus which is using only variables.
Three different solutions are presented for replacing the current solution for the signal bus, standard connectors, expandable connectors and records. All the three solutions have disadvantages and advantages.

Connectors

`Connectors` can be declared as inputs and/or outputs, which gives the possibility of making an I/O connector:

```
connector A
  input Real x;
  output Real y;
end A;

connector B
  output Real x;
  input Real y;
end B;
```

The connector `A` and `B` compress and decompress the two different interfaces (real in/output) to one single interface (the bus) which can be seen in figure 3.1.

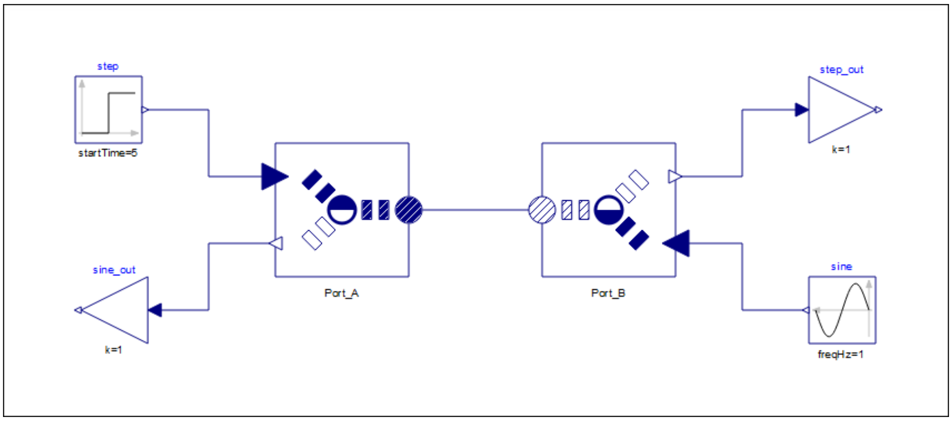


Figure 3.1: *Port_A connects the left input and output to the bus consisting of* `connector A` *and* `B` *and Port_B connects the right side in a similar way. Illustration made in* WSM.

The advantage of this solution is the clean structure with the possibility of only one connection line (multiple instances of inputs and outputs are possible). Disadvantages are the visual feedback of the direction of the causal signals and the appearance similarity to standard connectors (acausal). It is good for the understanding to see the flow of information in a control loop, which demand a minimum of two connection lines with a clear information direction.

Expandable Connectors

`Expandable connectors` are a special connector type which is intended to be used for signal buses. It is possible to connect signals not declared in the connector, then a component is created with the used name and corresponding type. There can exist components inside the connector which are not used and expandable connectors can be used as components (9.1.3 in [Mod, 2014]).

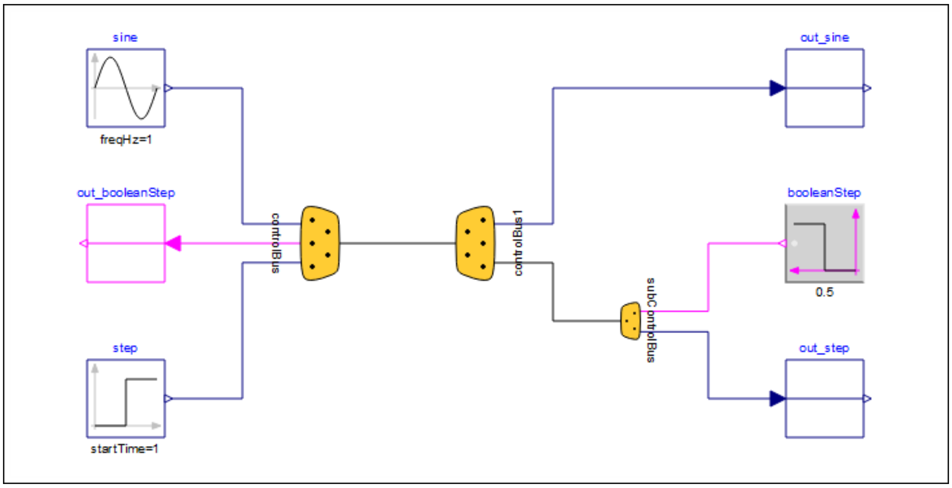


Figure 3.2: *An simple signal bus which takes three different signal (two inputs and one output) through the bus, splitting them into an output and a sub-bus. The sub-bus is then split into one input and one output. Illustration made in WSM.*

The advantages of this solution are the easy declaration of connections and the inclusion of new connections. The minimum amount of connection lines for this type of connector is one.
The disadvantages are the low insight of the connection pattern and the causality of the internal connection (all connections are causal, both input and output mixed) is not clear.

Note: To create a bus as illustrated in figure 3.2 it is necessary to connect the sub-bus with the signals from controlBus1 (otherwise the sub-bus connection would continue on the other side), the connections are not obvious and cannot be seen in the figure.

Records

`Records` are a specialized class in Modelica, meaning that it has restrictions apart from a general class. A record can only contain public sections (no equation, algorithm, protected etc.) and the same applies to any of its components (4.6 in [Mod, 2014]).

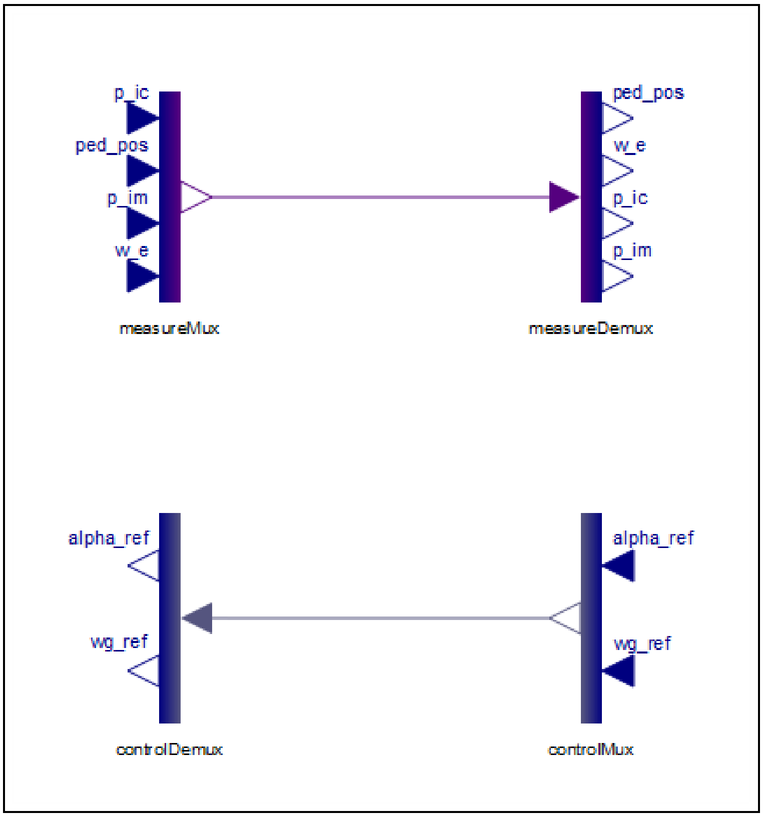


Figure 3.3: *An example of two sets of signals (measurements from sensors and control signals) which are transformed to an record interface with help of multiplexers and demultiplexers. Illustration made in* WSM.

The advantages of records are the clear interface with a clear causality direction. It is easy to keep track of the information.
The disadvantages are the minimum of two connections (one for input and one for output) and the need of multiplexers and demultiplexers.

Recommended Solution

Using records for transporting information in a control loop is the recommended solution, it enables a good flow of information and a clear structure. The multiplexers and demultiplexers keep track of connections which makes it easier for the user.

3.3 Library Design

The next two sections covers the results of the design choices for VehProLib and the new test library VehProTestLib.

3.3.1 VehProLib

The VehProLib has been allowed to keep most of its original packages, the partial packages have been removed and the partial model has instead been placed in the interfaces package. To make the library as user friendly as possible each package has received an illustrative icon in a color (RGB[B]: (85, 85, 85)) which is consistent for the package icons throughout the library, an example can be seen in figure 3.4.

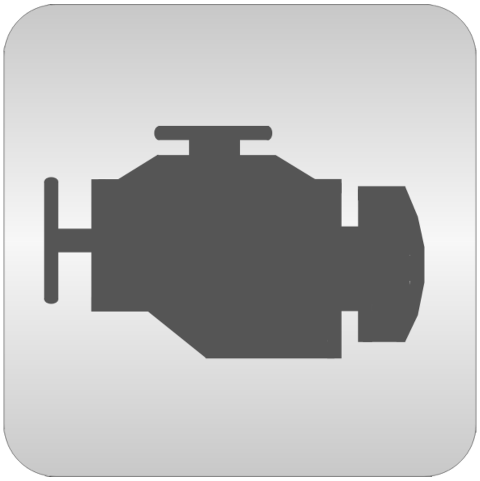


Figure 3.4: *The icon for the engine package.*

The chosen design consists of a three level structure inside the library to keep down the complexity, where the second level can consist of sub-packages and/or components/examples and the third level can only consist of components/examples as illustrated figure 3.5:

ExampleHierarchy

- Main package
  - Sub-package
    - Component
  - Component

Figure 3.5: *A schematic library structure, containing all three levels in the library structure. Each level may contain multiple instances, for illustration a minimalistic structure is shown.*

The main packages are decided not to be sorted in alphabetical order. The importance and the occurrence of components are factors that are valued more from a user standpoint. For instance the examples package is placed at the top, where it is easy to find. Library users tend to use examples frequently for both understanding and inspiration. The sub-packages and the components are sorted in alphabetical order independently. When there are both sub-packages and components the sub-packages are placed at the top for easy access and to avoid mixing of sub-packages and components.

There are some new packages added to the library:

Blocks Containing block components

Functions Containing functions

Icons Contains common icons

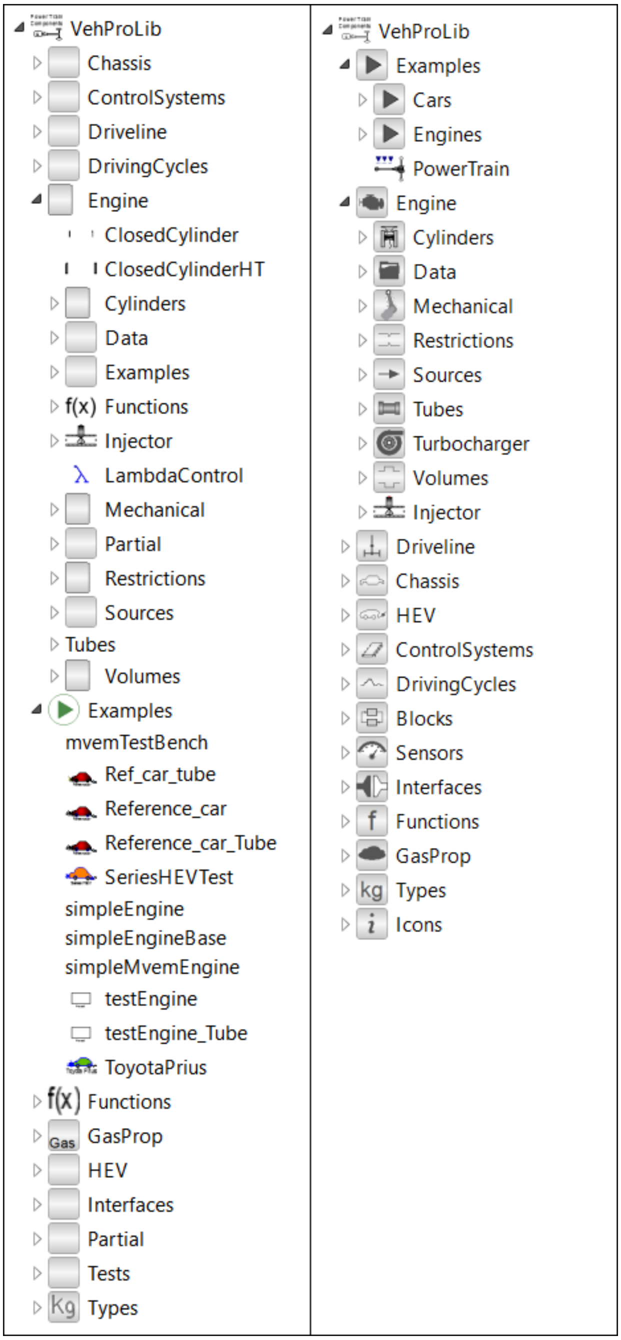
Interfaces Containing all interfaces and partial models

Sensors Containing sensors

These packages were before part of other packages or non-existent. By keeping them at a top level the availability is increased and the structure is clearer.

The recommendations for the development of Modelica libraries can be found in appendix at page 112.

The final design of the VehProLib compared to the original design can be seen in figure 3.6.



- VehProLib
  - Chassis
  - ControlSystems
  - Driveline
  - DrivingCycles
  - Engine
    - ClosedCylinder
    - ClosedCylinderHT
    - Cylinders
    - Data
    - Examples
    - f(x) Functions
    - Injector
    - LambdaControl
    - Mechanical
    - Partial
    - Restrictions
    - Sources
    - Tubes
    - Volumes
  - Examples
    - mvemTestBench
    - Ref_car_tube
    - Reference_car
    - Reference_car_Tube
    - SeriesHEVTest
    - simpleEngine
    - simpleEngineBase
    - simpleMvemEngine
    - testEngine
    - testEngine_Tube
    - ToyotaPrius
  - f(x) Functions
  - Gas GasProp
  - HEV
  - Interfaces
  - Partial
  - Tests
  - Kg Types

- VehProLib
  - Examples
    - Cars
    - Engines
    - PowerTrain
  - Engine
    - Cylinders
    - Data
    - Mechanical
    - Restrictions
    - Sources
    - Tubes
    - Turbocharger
    - Volumes
    - Injector
  - Driveline
  - Chassis
  - HEV
  - ControlSystems
  - DrivingCycles
  - Blocks
  - Sensors
  - Interfaces
  - f Functions
  - GasProp
  - kg Types
  - i Icons

Figure 3.6: *The old (left) and the current (right) version of VehProLib seen in the class browser in* WSM *with the examples and engine packages expanded.*

3.3.2 VehProTestLib

VehProTestLib is a test library used to validate the standard models in VehProLib and make sure they work as intended. At the moment there is reference files (.mat) stored in the same folder as the simulated model with simulation results. This prepares for possible future automated test suites and creates no platform or version dependency. The library has been declared dependent on VehProLib, meaning that you get an warning if you open the library without VehProLib already loaded. The structure of VehProTestLib is chosen to very similar to VehProLib for obvious reasons, what is different is the addition of the main package Cars and the inclusion of a magnifying lens on top of the package icons. The inclusion of the lens was made to make it easier to distinguish between the two libraries.

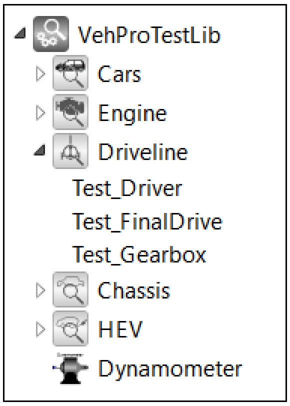


Figure 3.7: *VehProTestLib seen in the class browser in WSM with the driveline package expanded.*

3.4 Selection of Equation of States

The equation of states is the relation between state variables, one commonly used is the ideal gas law (see section 2.5). There are several different equations of states (Van der Waals equation is one of them) used in thermodynamics dependent on the approximation, environment the gas exists in etcetera. As explained in section 2.4.2 there is no easy way to change the equation of states for the gas due to how the relations are implemented. There are some different alternatives to solve this, due to prioritization and the limit amount of time the recommended solution is not implemented.

3.4.1 Equation of States as a Function

The first approach is to define the equation of states as a function inside the gas model. This allows for a convenient way of switching between different types equation of states. The component could use this functionality in this way:

```
model ControlVolume
  ...
  parameter SI.Volume V;
  SI.Pressure T;
  SI.Pressure p;
  SI.mass m;
equation
  p = g.EquationOfStates(m, g.R, T, V);
  ...
end ControlVolume;
```

There are some drawbacks to this solution, namely that if you want to use the equation of state in another form, for instance the derivative or another quantity such as temperature you need to have multiple functions, one for each case. The equation stated in the component is lost and the user get less insight into the underlying equations.
The advantages are that the equations are gathered in one place and changes can be made without involving other components. This also applies to adding new equations of states.

To this solution it is possible to make the function replaceable (the concept of replaceable was discussed by Otte Montell in section 4.1 in [Montell, 2004]), meaning that the same gas can be declared with different equations of states. To avoid implementation of the same gas model for different equations of states.

3.4.2 Selection of Equation of States

The second option is to define multiple equations inside of components and define a selector inside the gas model:

```
model ControlVolume
  ...
  parameter SI.Volume V;
  SI.Pressure T;
  SI.Pressure p;
  SI.mass m;
equation
  if g.equation_of_states == "Van der Waal" then
    (p + a * (n / V)^2) * (V - n * b) = n * R * T;
  else
    p = m * R * T / V;
  end if;
  ...
end ControlVolume;
```

There is a big drawback to this solution, there is one implementation of each equation in each involved component. Any changes or new implementation influences all involved components. The advantages is the clear understanding of the underlying equations.

3.4.3 Inner/Outer Function

In the same way as the gas model is declared it is possible to declare a function as outer inside the involved components. And then at top level declare the inner function, preferably as replaceable:

```
model ControlVolume
 ...
  parameter SI.Volume V;
  SI.Pressure T;
  SI.Pressure p;
  SI.mass m;
  outer function EquationOfStates;
equation
  p = EquationOfStates(m, g.R, T, V);
  ...
end ControlVolume;
```

This solution is very similar to the first discussed solution (see section 3.4.2) and has the same advantages and disadvantages, with the benefit of separating the gas model and the equation of states.

3.4.4 Recommended Solution

The recommended solutions is the first one. It has the advantages of interchangeability and a convenient way of changing the active equation of states. The disadvantage of not clearly see the underlying equations is insignificant compared to the interchangeability the solution provides.

3.5 Van der Waals Equation

The Van der Waals constants are calculated from equation 2.9 and 2.10. In table 3.1 below are Van der Waals constants presented for some elements (which are useful in engine modeling).

Table 3.1: *Table containing some common elements which are relevant for gas simulation in an engine. Values for temperature and pressure are taken from Physics Handbook [Nordling and Österman, 1980,1997]*

Name	**Critical temperature** K	**Critical pressure** 10^6 Pa	$\boldsymbol{a}$ 10^{-3} N m^4 mol^{-2}	$\boldsymbol{b}$ 10^{-6} m^3 mol^{-1}
Air	130	3.8	129.71	35.56
Carbon dioxide	304.19	7.38	365.67	42.84
Nitrogen	126.3	3.4	136.83	38.61
Nitrogen monoxide	366	7.26	538.12	52.40
Nitrogen oxide	312	10	283.90	32.43
Oxygen	154.77	5.08	137.52	31.66

In Physics Handbook there also exists some tabulated values for *a* and *b* which is presented in table 3.2.

Table 3.2: *Van der Waals constans tabulated in Physics Handbook [Nordling and Österman, 1980,1997] compared to the constants calculated using equation (2.9) and (2.10).*

Name	a	a_{ref}	$\left\|\frac{a-a_{ref}}{a_{ref}}\right\|$	b	b_{ref}	$\left\|\frac{b-b_{ref}}{b_{ref}}\right\|$
Carbon dioxide	365.67	362.8	0.79%	42.84	42.67	0.40%
Nitrogen	136.83	140.4	2.54%	38.61	39.13	1.34%
Oxygen	137.52	137.4	0.09%	31.66	31.83	0.52%

As can be seen in table 3.2 the values for the constants are within 5% from the reference values. This gives a good indication of that similar elements should have good accuracy when using equation 2.9 and 2.10 for calculating Van der Waals constants.

3.5.1 Comparing Van der Waals Equation and the Ideal Gas Law

Due to the lack of time the in-cylinder model is not working properly (see section 3.2.1) and therefore cannot the comparison be done for an in-cylinder simulation. Comparison has been made for MVEM simulation which is presented below. The only difference between the two models are the equation of states for the cylinder block. In future studies it would be of interest to use Van der Waals equation of states in all components of interest (see section 3.4).

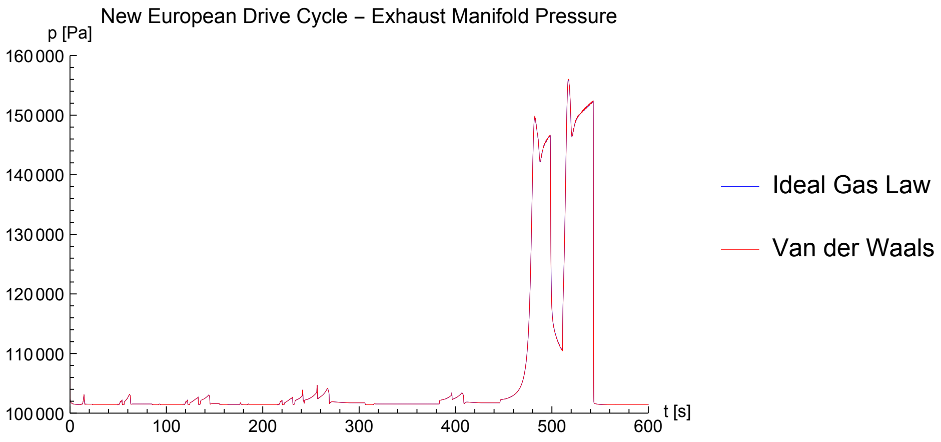


Figure 3.8: *Comparison of the exhaust manifold pressure between the ideal gas law and Van der Waals equation of states.*

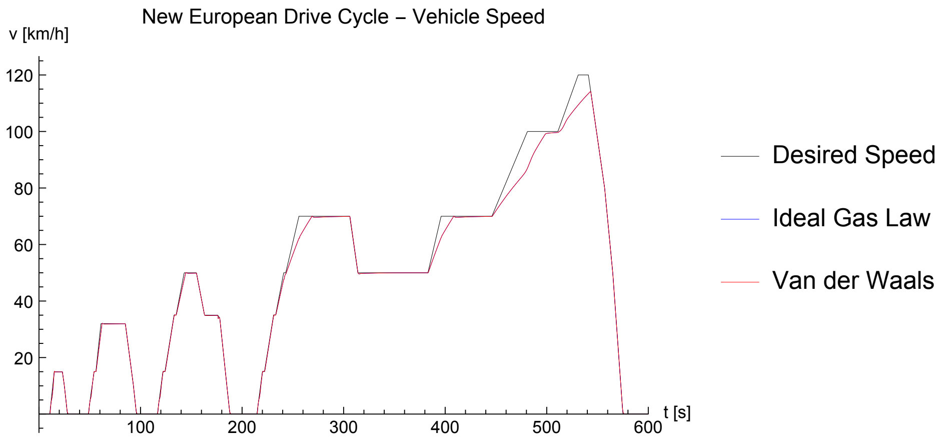


Figure 3.9: *Comparison of the speed between the ideal gas law and Van der Waals equation of states. The black graph is the demanded speed.*

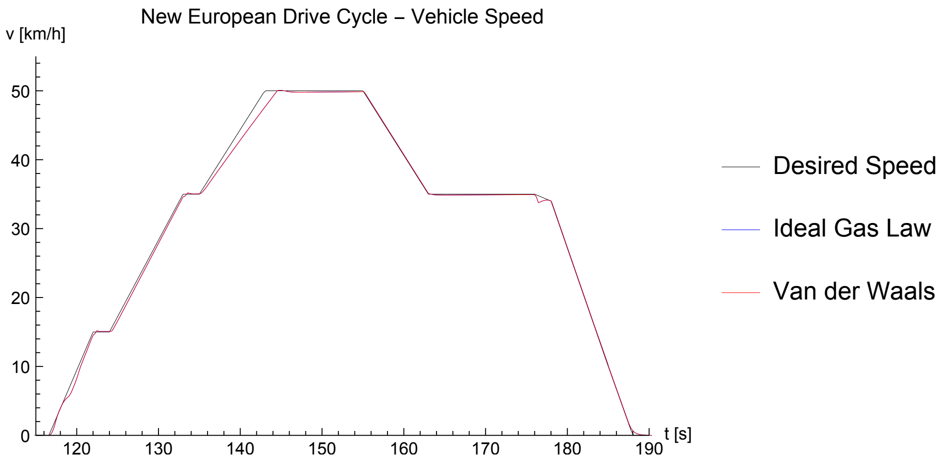


Figure 3.10: *Comparison of the speed between the ideal gas law and Van der Waals equation of states in the time span between 115 and 190 seconds*

As can be seen in all the graphs above, the deviation between the two models is very small. This indicates that the ideal gas law approximation is justified during vehicle engine simulations.

3.6 Template Models

Example of a turbo shaft model created from the turbo shaft template:

```
  model TurboShaft
  extends VehProLib.Interfaces.TurbochargerTemplates.
    TurboShaft_Template;
  parameter Real c_fric = 1e-6 "Friction coefficient
    [J s/rad^2]";
equation
  M_fric = c_fric * w;
end TurboShaft;
```

3.7 Model Fitting in Mathematica

Mathematica has been used to determine the model parameters for the turbine model and the compressor model.

3.7.1 Compressor

With help of equation (2.16) to (2.20) and NMinimize (section 2.9.1) the compressor model stated in section 2.7.1 was fitted to real data.
The achieved constants were:

$$\begin{aligned}
\eta_{min} &= 0.2960 \\
\eta_{max} &= 0.8206 \\
\dot{m}_{corr,max} &= 0.1804 \\
\dot{m}_{corr\ at\ \eta\ max} &= 0.0842 \\
\Psi_{max} &= 0.0027 \\
\Pi_{at\ \eta\ max} &= 1.9561 \\
Q_\eta &= \begin{pmatrix} 90.5045 & -6.7729 \\ -6.0504 & 0.8223 \end{pmatrix}
\end{aligned}$$

To validate the parameters the pressure ratio and the efficiency were plotted against the mass flow and the result was compared to the real data. This can be seen in figure 3.11 and figure 3.12 respectively.

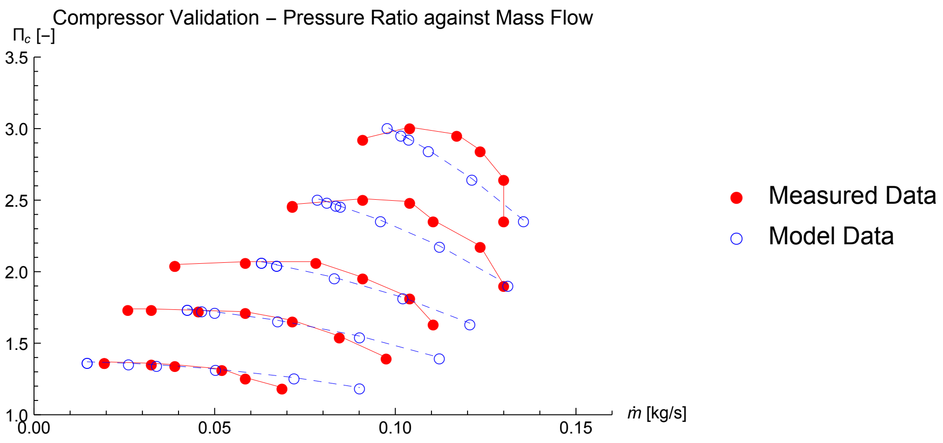


Figure 3.11: *Comparing the mass flow of the model against the real data.*

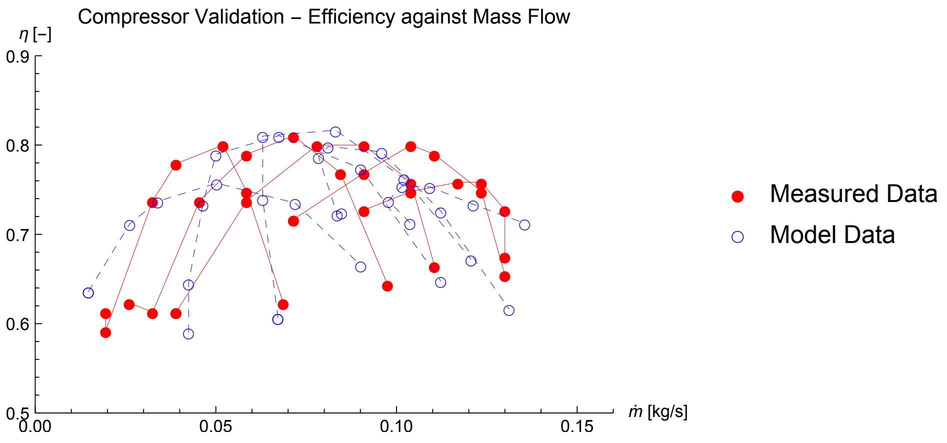


Figure 3.12: *Comparing the efficiency of the model against the real data.*

As can bee seen in figure 3.11 the mass flow model follows the real mass flow quite well. Where the efficiency model in figure 3.12 is not as accurate, it follows the real efficiency acceptably.

The Mathematica script can be found at page 87

3.7.2 Turbine

With help of equation (2.26) to (2.29) and NMinimize (section 2.9.1) the turbine model stated in section 2.7.2 was fitted to real data.
The constants achieved was:

$$
\begin{aligned}
TFP_{max} &= 0.0054 \\
TFP_{exp} &= 1.4503 \\
\eta_{min} &= 0.3891 \\
\eta_{max} &= 0.8072 \\
BSR_{max} &= 0.6715
\end{aligned}
$$

To validate the parameters the TFP was plotted against pressure ratio and the efficiency was plotted against the BSR and the result was compared to the real data. This can be seen in figure 3.13 and figure 3.14 respectively.

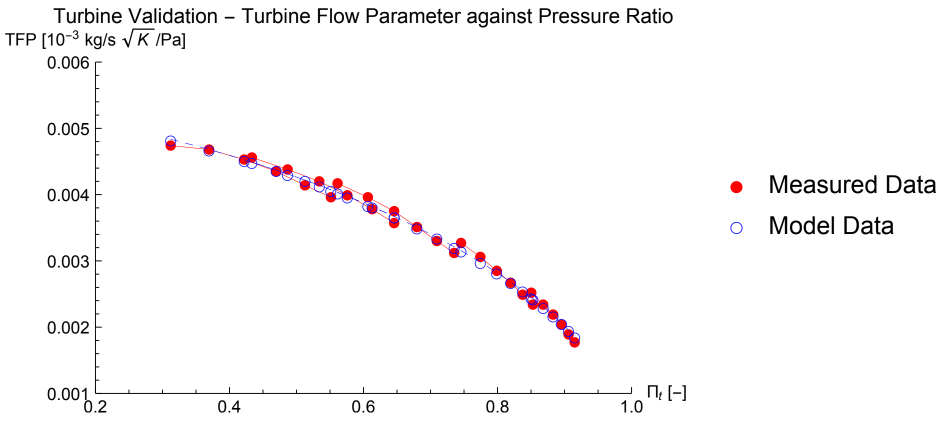


Figure 3.13: *Comparing the* TFP *of the model against the real data.*

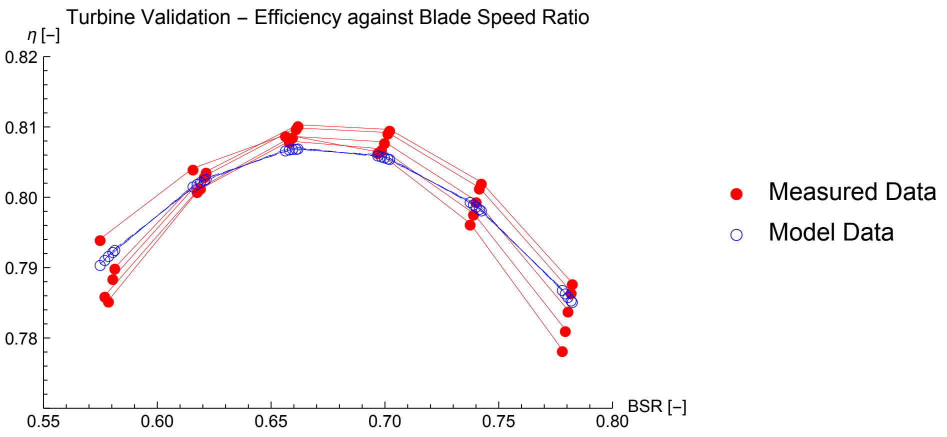


Figure 3.14: *Comparing the efficiency of the model against the real data.*

As can bee seen in figure 3.13 and figure 3.14 the turbine model follows the real data very well.

The Mathematica script can be found at page 93

3.8 Driver

The original driver consisted of a PI regulator which regulated the demanded speed against the actual speed of the vehicle. A positive error results in a gas pedal adjustment and a negative error in a brake pedal adjustment, the two actions are completely separated, meaning that it is not possible to brake and increase the gas at the same time. After the reconstruction of the MVEM cylinder model (see section 3.10.1) the PI regulator was adjusted to minimize the error, but this resulted in a ringing effect, see figure 3.15.

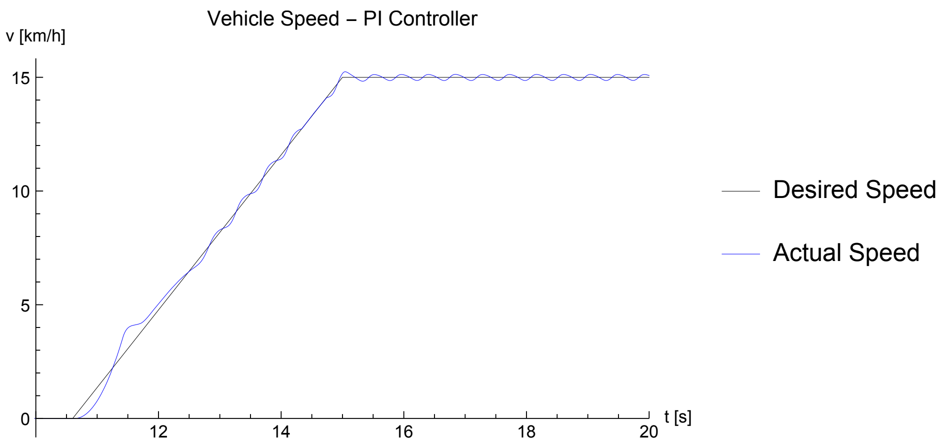


Figure 3.15: *The vehicle speed compared to the demanded speed in the interval between 10 and 20 seconds in the NEDC.*

The ringing is handled in section 3.8.1. When the Driver model were investigated the abstraction of the code and graphics was increased which is covered in section B.2.2.

3.8.1 PID

When ringing is introduced in a system (as in figure 3.15) a derivative part can be added to the regulator to suppress the ringing. The constants for the PID were decided by Ziegler-Nichols (5.4.1 in [Enqvist et al., 2010]) method and the result can be seen in figure 3.16

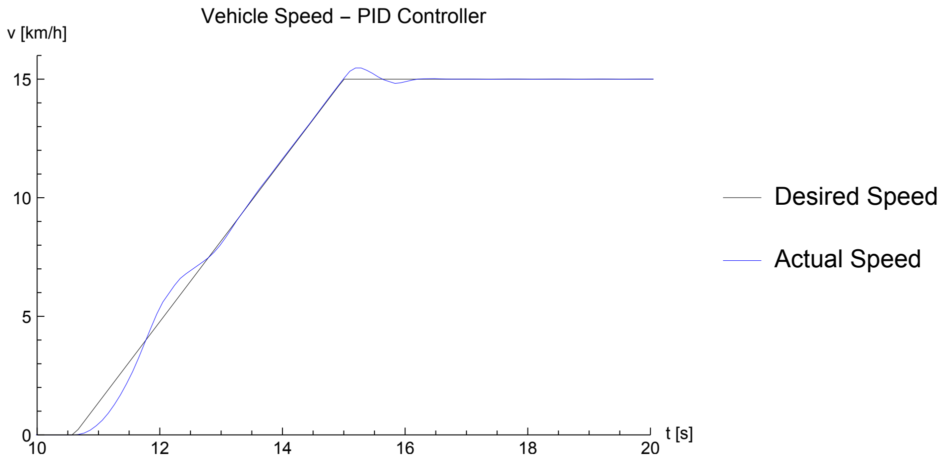


Figure 3.16: *The vehicle speed compared to the demanded speed in the interval between 10 and 20 seconds in the NEDC.*

The PID used as the driver includes an anti-windup feedback loop to prevent integral windup. Integral windup is when the integral term in the regulator grows out of the physical boundary due to a large error, or a long enough persistent error. For instance could a maximum value of the output be 100%, but due to the integral windup the output value could increase over 100 %. A saturation will limit the output but not the actual value which will yield a delay.

3.9 Model Validation

To determine the consistency of the engine models, the models have been compared to its corresponding Simulink models (which are validated against literature). The compared models have the same input and parameter values.

3.9.1 MVEM - Cylinder

The cylinder model receive a step as input for the incoming pressure (and the incoming fuel mass flow) from 20000 [Pa] to 40000 [Pa], the output torque for the models can be seen in figure 3.17. The outputs are almost identical.

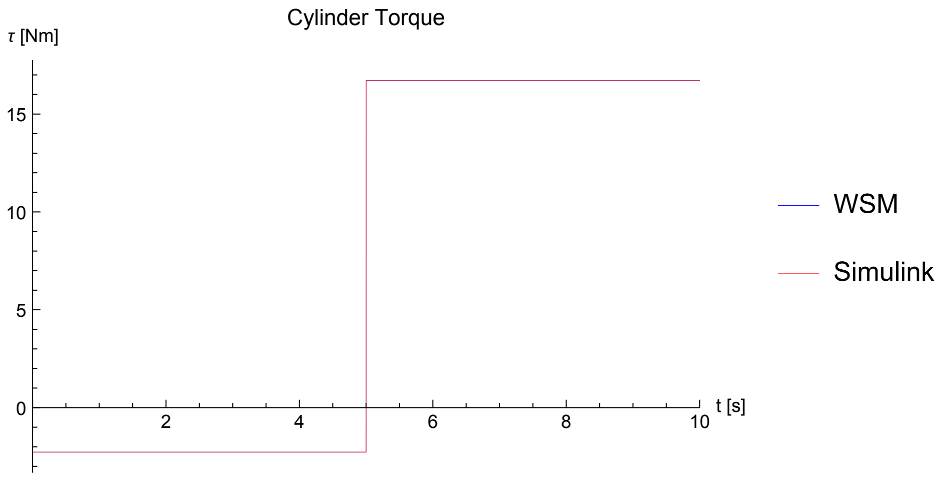


Figure 3.17: *Comparison of the cylinder torque between the* WSM *and the Simulink model.*

3.9.2 Throttle

The **`Throttle`** models receive a ramp input that increases from 0.2 to 1, that starts at 5 seconds and end at 13 seconds. As can be seen in figure 3.18 the outputs in mass flow are almost identical.

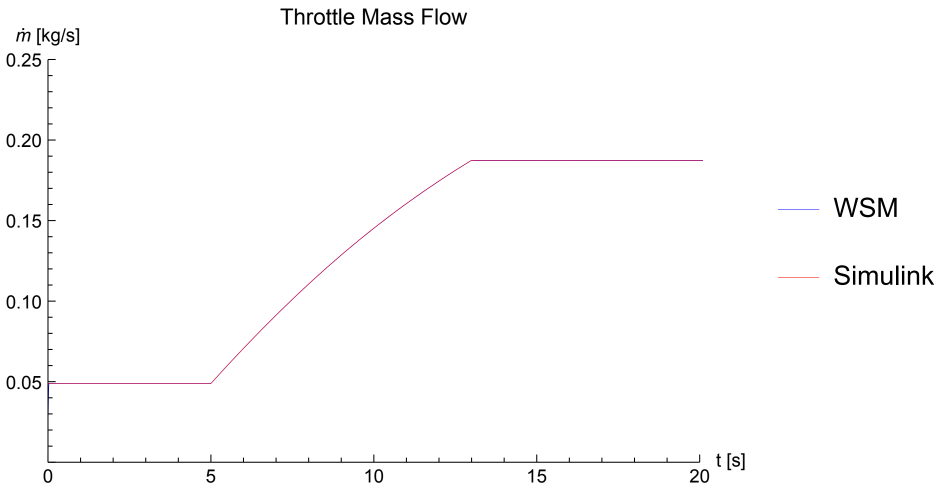


Figure 3.18: *Comparison of the throttle mass flow between the* WSM *and the Simulink model.*

3.9.3 Control Volume

The mass flow was varied as a step from lower mass flow to higher on the right hand side of the **`ControlVolume`**. The output pressures can be seen in figure 3.19, there is a small deviation in the pressure. This is probably due to the enthalpy balance occurring in the WSM model which is not included in the SImulink model. The gas interface in VehProLib needs an enthalpy flow, discussed in section 2.4.

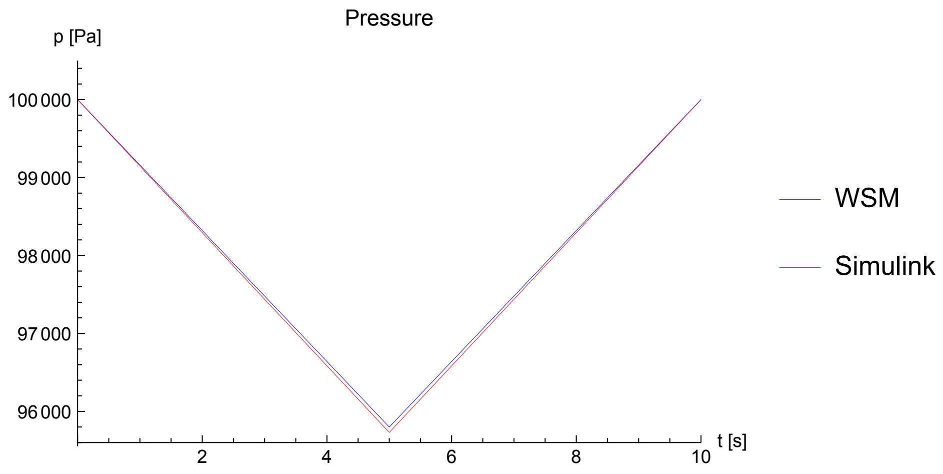


Figure 3.19: *Comparison of the control volume pressure between the WSM and the Simulink model.*

3.9.4 Incompressible Restriction

The **`IncompressibleRestriction`** received a ramp as input increased the pressure from 200 [Pa] to 80200 [Pa] starting at 2 seconds and ended at 8 seconds. The linear region is at 500 [Pa], as cen be seen in figure 3.20 the output mass flow are identical.

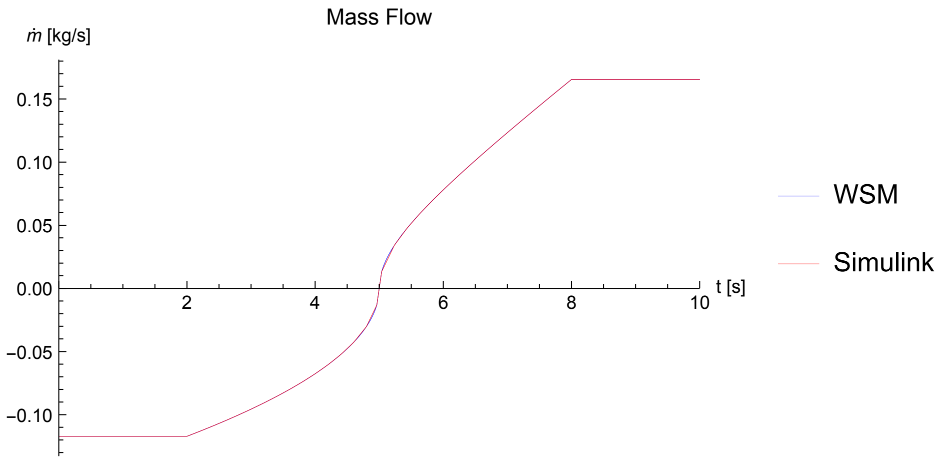


Figure 3.20: *Comparison of the mass flow between the WSM and the Simulink model.*

3.9.5 Intercooler

The **`Intercooler`** model is not compared to the corresponding Simulink model in figure 3.21, instead is the temperature drop studied. As stated in equation 2.39 the temperature drop is equal to the difference between the incoming and outgoing temperature times an efficiency. In this case: T_{in} = 393 [K], T_{out} = 293 [K], ϵ = 81%, this should give an intercooler temperature of 312 [K] which can be seen in figure 3.21.

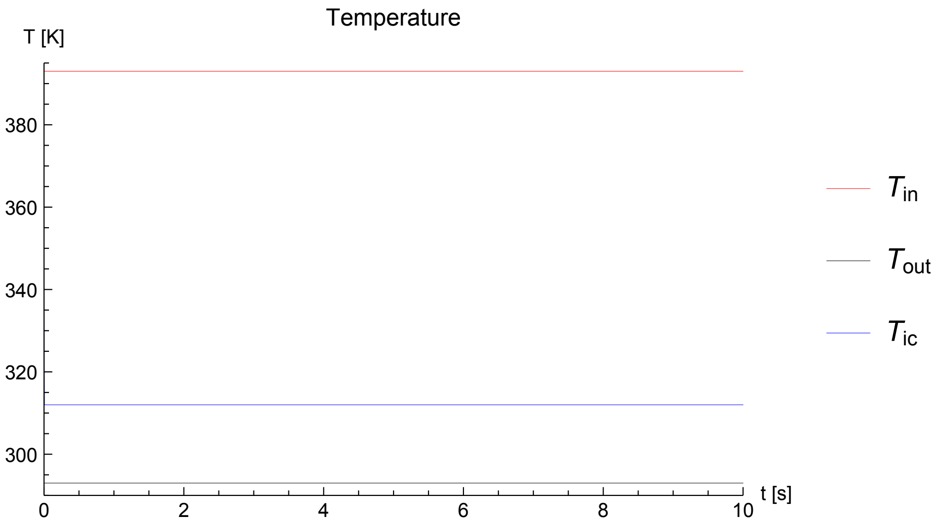


Figure 3.21: *The intercooler temperature compared to the incoming and the outgoing temperature.*

3.9.6 Boost Controller

The `BoostController` is an example of a component which due to the different model implementation yields different results. As can be seen in figures 3.22 and 3.23 the outputs are identical given the same inputs. Due to implementation problems the PI-regulator are not the same as the Simulink model. This means that the PI parameters needs to refitted to real data to yield the same output. This holds true for all PI regulators in the library. The difference in the output can be seen in figure 3.24 and 3.25.

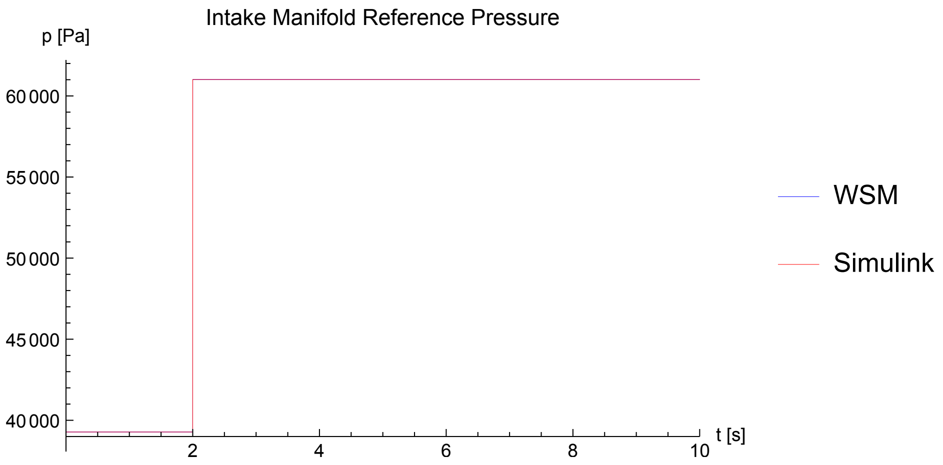


Figure 3.22: *Comparison of the intake manifold reference pressure between the WSM and the Simulink model.*

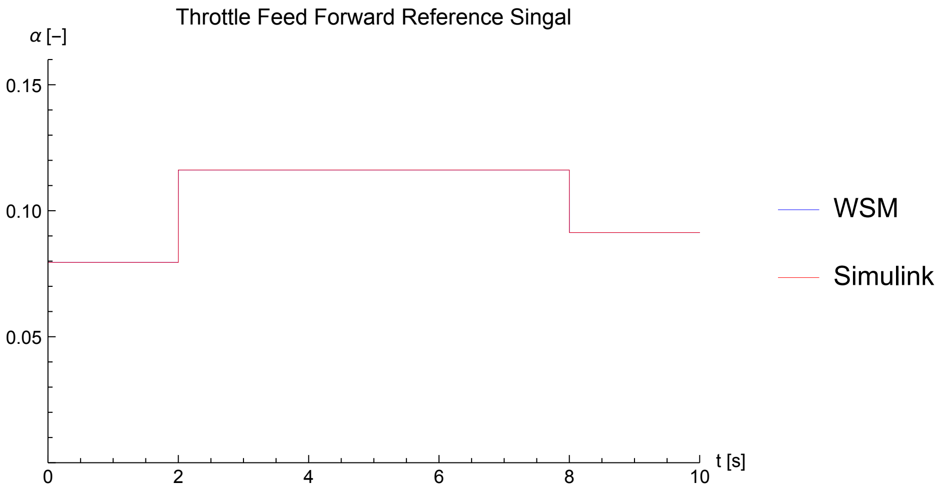


Figure 3.23: *Comparison of the throttle feed forward signal between the WSM and the Simulink model.*

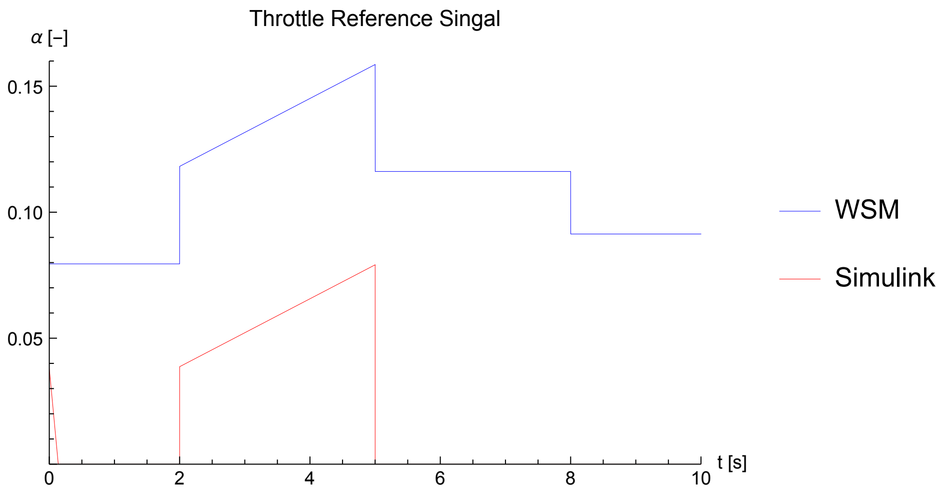


Figure 3.24: *Comparison of the throttle reference signal between the WSM and the Simulink model.*

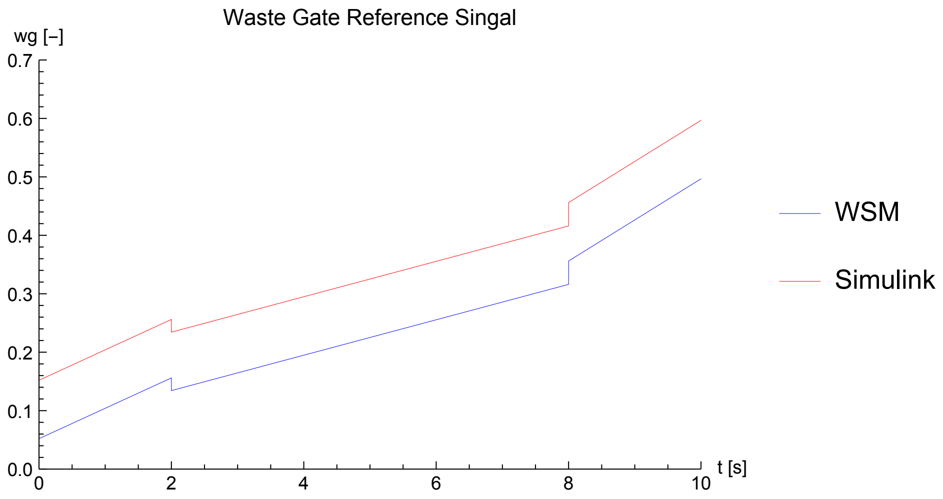


Figure 3.25: *Comparison of the waste gate reference signal between the WSM and the Simulink model.*

3.10 Vehicle Model Example

The following sections covers the interaction between the components validated in the previous section, to show that the library is linked together.
By simulating the a car following the NEDC and study the engine behaviour compared to a similar Simulink model, the behaviour of the Modelica model can be determined. The assumption is that the Simulink model has a correct behaviour.
The following sections covers the MVEM engine model first without turbocharger then with turbocharger.

3.10.1 Simulation of New European Driving Cycle - MVEM

Example of a MVEM engine without a turbocharger. This can be seen in figure 3.26 to 3.33.

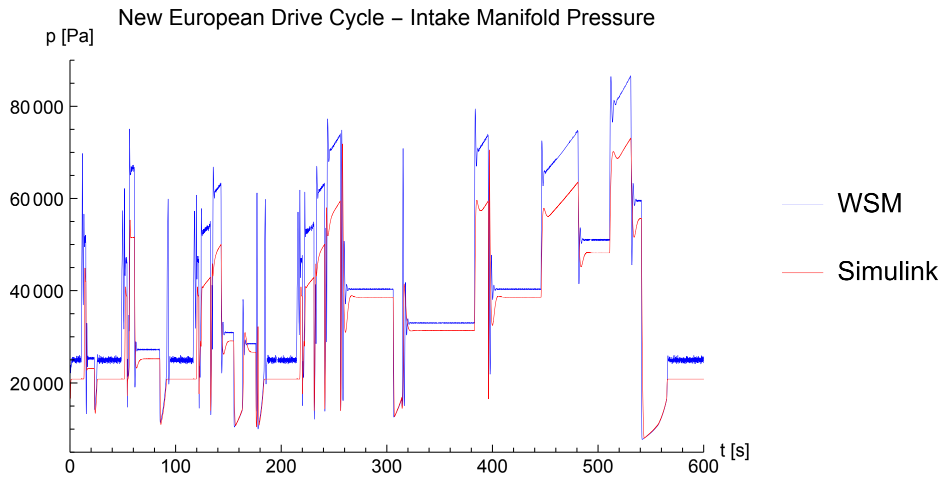


Figure 3.26: Comparison of the intake manifold pressure between the WSM and the Simulink model.

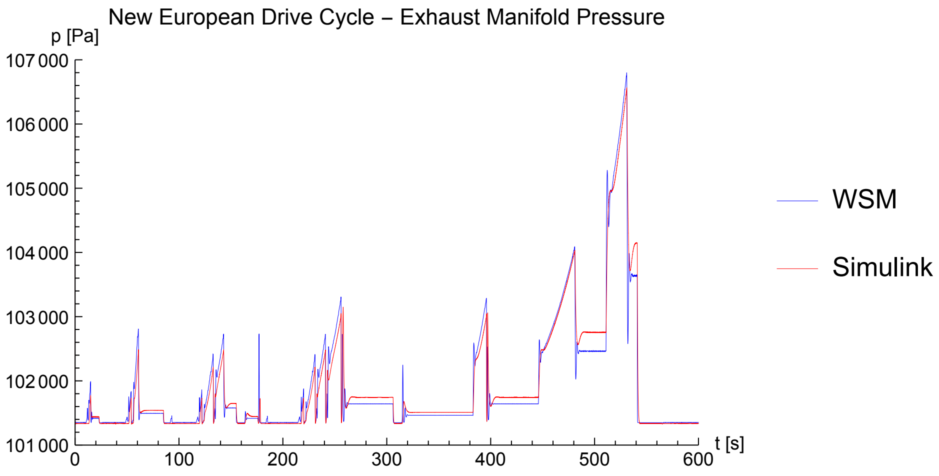


Figure 3.27: *Comparison of the exhaust manifold pressure between the* WSM *and the Simulink model.*

As can be seen in figure 3.26 and 3.27 the pressure characteristics in the IM and EM are very similar to the Simulink model, especially the EM pressure. The small deviation is discussed in section 3.10.1.

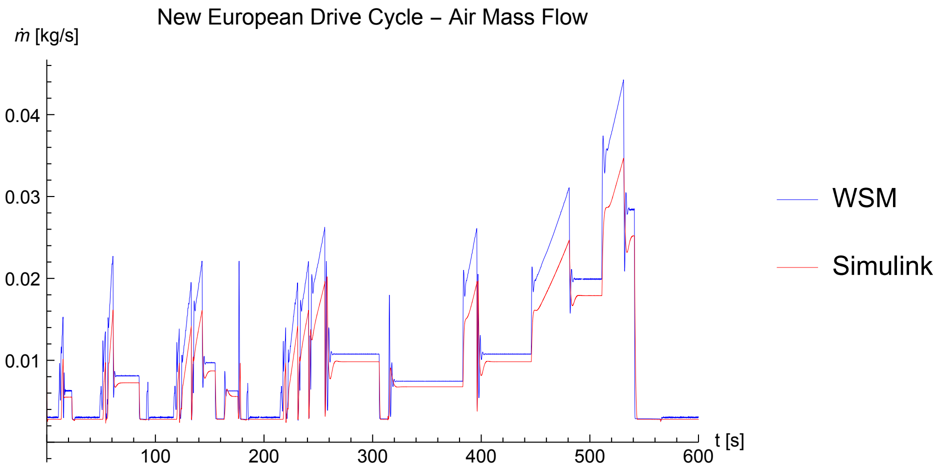


Figure 3.28: *Comparison of the air mass flow into the engine between the* WSM *and the Simulink model.*

As can be seen in figure 3.28 the air mass flow characteristics is very similar to the Simulink model. The deviation is discussed in section 3.10.1.

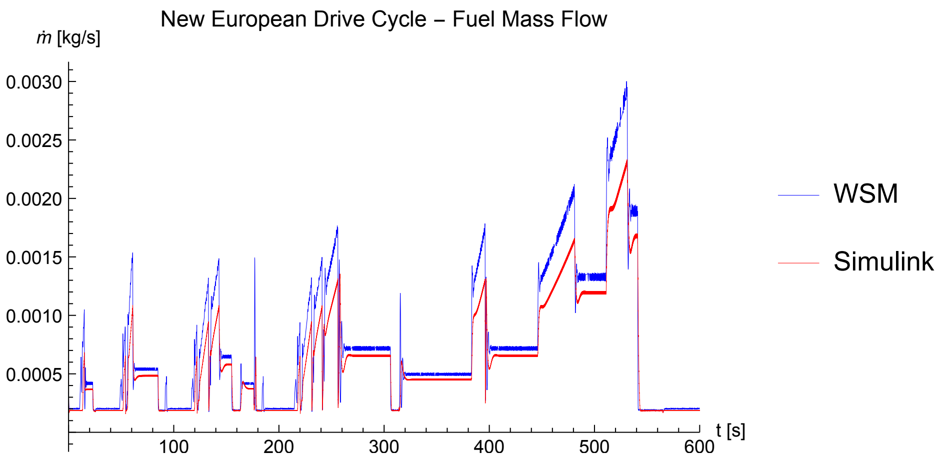


Figure 3.29: *Comparison of the fuel mass flow into the engine between the WSM and the Simulink model.*

The mass flow for the fuel follows the same characteristics as the air mass flow due to the fact that lambda is close to one, see figure 3.31. The Modelica line oscillates more compared to Simulink because the variations in lambda is larger. The offset in fuel mass flow give rise to a higher fuel consumption, the average fuel consumption is 0.73 [litre per 10 km] compared to 0.62 [litre per 10 km] for the Simulink model.

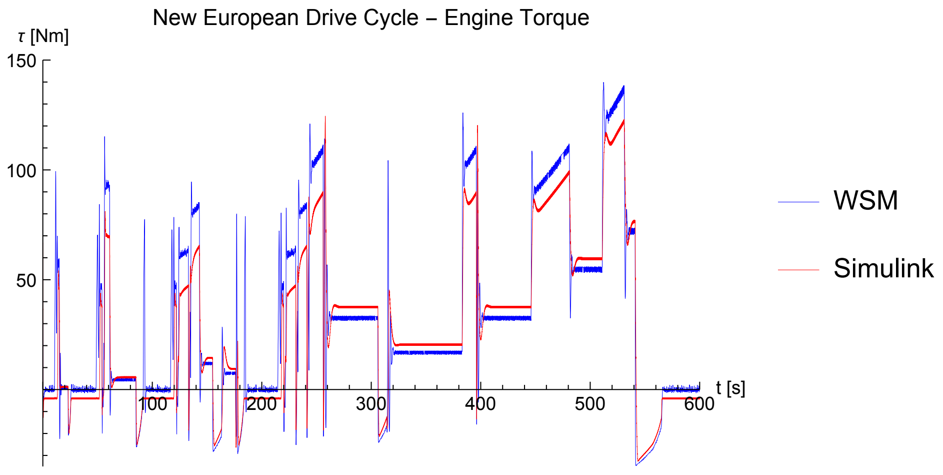


Figure 3.30: *Comparison of the engine torque between the WSM and the Simulink model.*

The torque is strongly connected to the fuel and air mass flow, the same effects are present here.

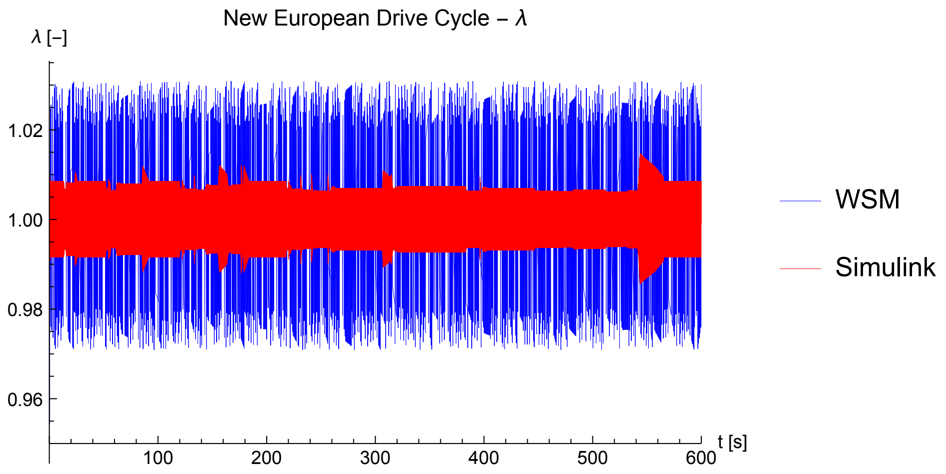


Figure 3.31: *Comparison of the lambda (proportion of oxygen) between the WSM and the Simulink model.*

The lambda value should vary around 1 with a variation around 0.03, this is the case and can be seen in figure 3.31. The larger variations are most likely due to the lambda controller, which has not been compared with the Simulink lambda controller.

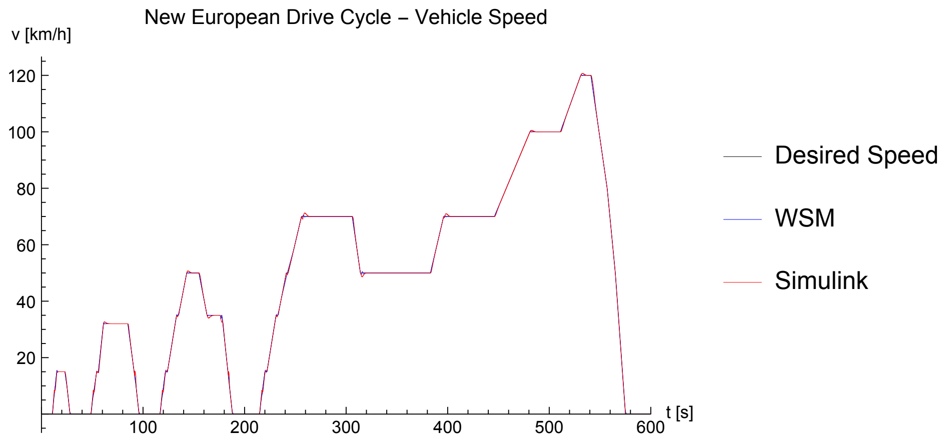


Figure 3.32: *Comparison of the speed between the WSM and the Simulink model. The black graph is the demanded speed.*

As can be seen in figure 3.32 the Modelica model follows the drive cycle with a pleasant behaviour. In some parts it follows the driveline even better than the Simulink model (it can easier be seen in figure 3.33), this should mainly be because of the different driver models (see section 3.8).

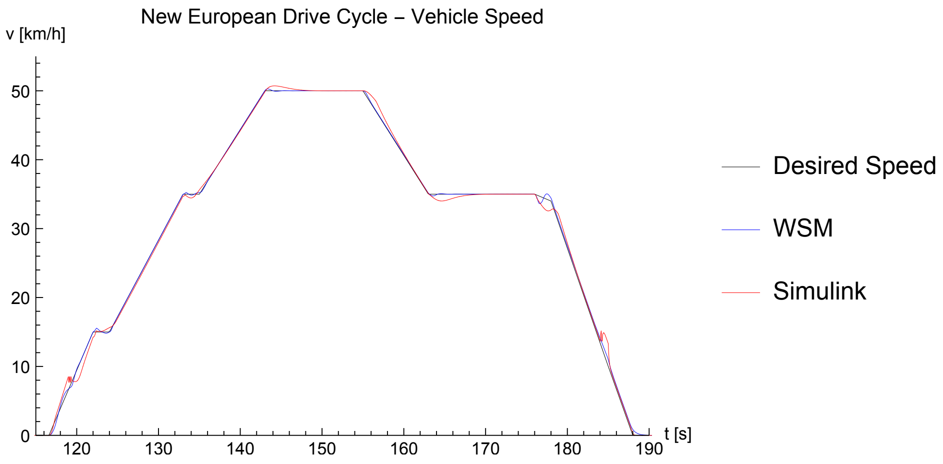


Figure 3.33: *Comparison of the speed between the WSM and the Simulink model in the time span between 115 and 190 seconds*

Deviation from the Simulink Model

The small deviation from the Simulink model can be due to several reasons. The mass flow into the engine and engine torque deviates from the reference data with an offset. The wheel model is different from the Simulink model which is likely the cause to the offset. There exist some cases where the fuel mass flow is higher for the WSM model than the Simulink model (for instance at time 500 seconds) where the engine torque for the WSM model is lower than that of the Simulink model. This indicates that the engine characteristics are not exactly the same.
As can be seen in figure 3.32 the vehicle speed for the Modelica model compared to the Simulink model is slightly different, this is most likely due to another driver model and different PID parameters. The wheel model in Modelica is including wheel slip, which the Simulink model does not. It effects the engine speed and torque, by studying the engine speed this effects are small.

3.10.2 Turbocharged Engine Model

Example of a MVEM engine with a turbocharger. This can be seen in figure 3.34 to 3.42.

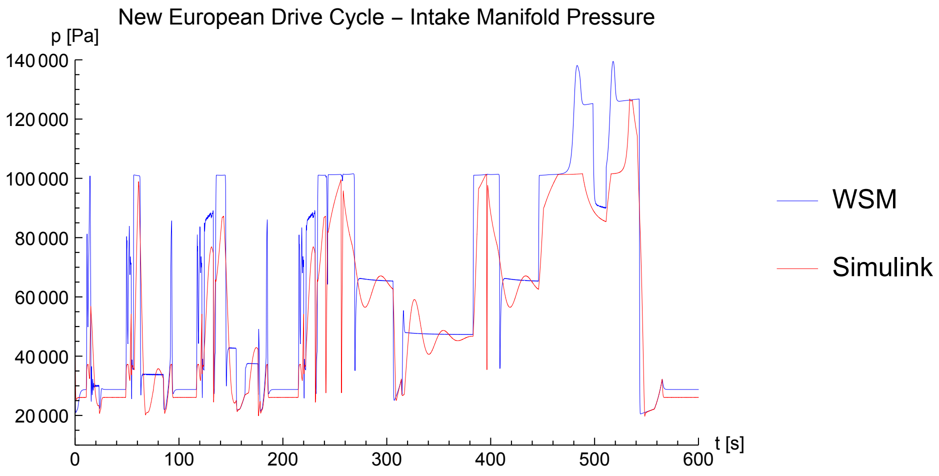


Figure 3.34: *Comparison of the intake manifold pressure between the* WSM *and the Simulink model.*

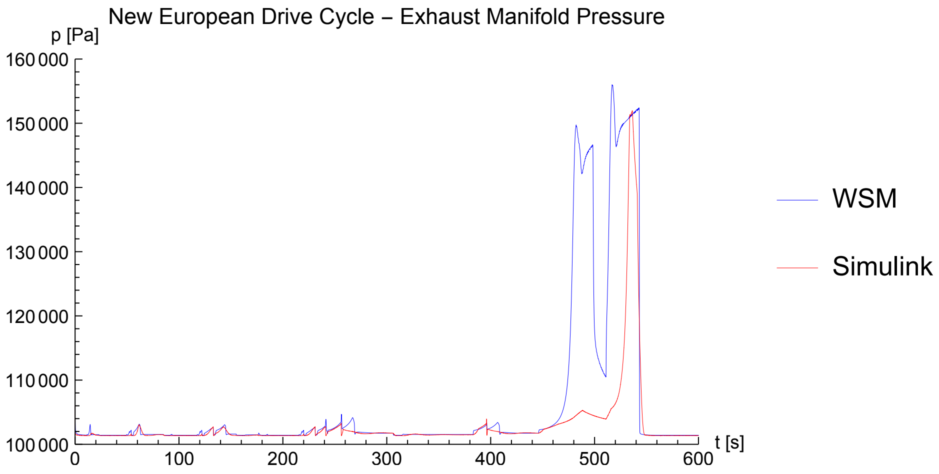


Figure 3.35: *Comparison of the exhaust manifold pressure between the* WSM *and the Simulink model.*

As can be seen in figure 3.34 and 3.35 the pressure characteristics in the IM and EM are very similar to the Simulink model. The small deviation and the additional top at the end are discussed in section 3.10.2.

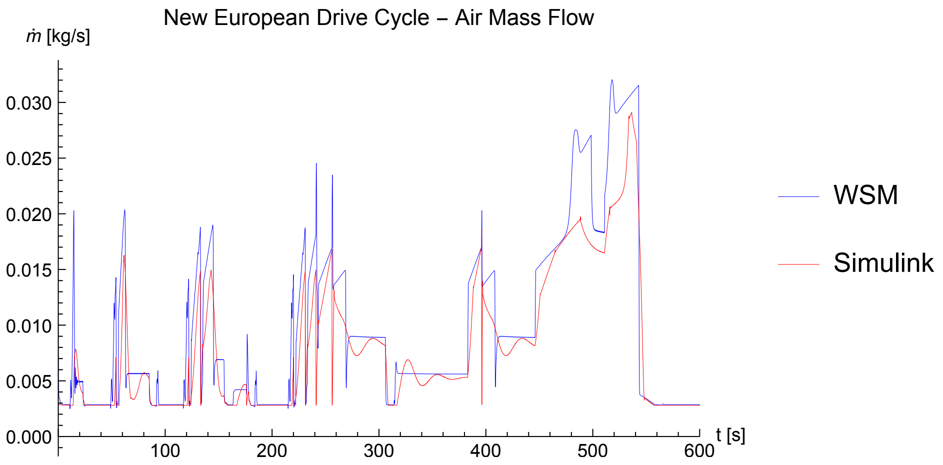


Figure 3.36: *Comparison of the air mass flow into the engine between the WSM and the Simulink model.*

As can be seen in figure 3.28 the air mass flow characteristics is very similar to the Simulink model. The deviation is discussed in section 3.10.2.

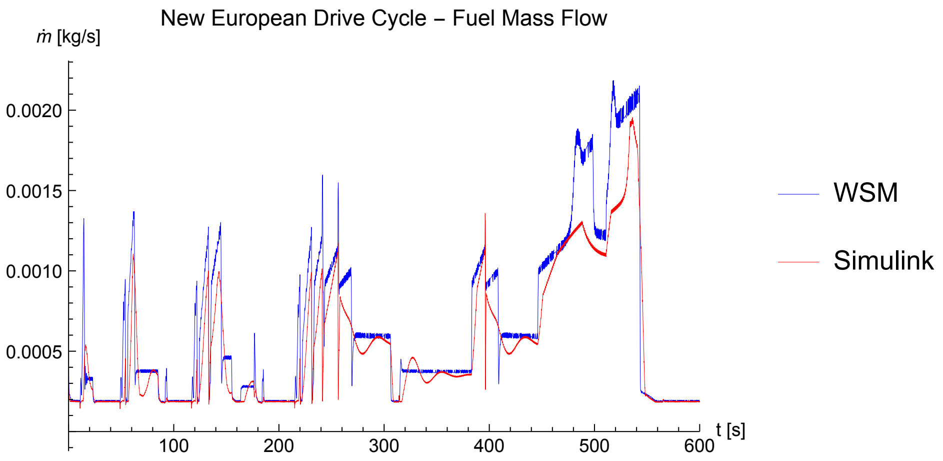


Figure 3.37: *Comparison of the fuel mass flow into the engine between the WSM and the Simulink model.*

The mass flow for the fuel follows the same characteristics as the air mass flow due to the fact that lambda is close to one, see figure 3.40. The Modelica line oscillates more compared to Simulink because the variations in lambda is larger. The offset in fuel mass flow give rise to a higher fuel consumption, the average fuel consumption is 0.66 [litre per 10 km] compared to 0.55 [litre per 10 km] for the Simulink model.
The WSM model decreased the average fuel consumption by 9.6 % compared to

11.2 % for the Simulink model.

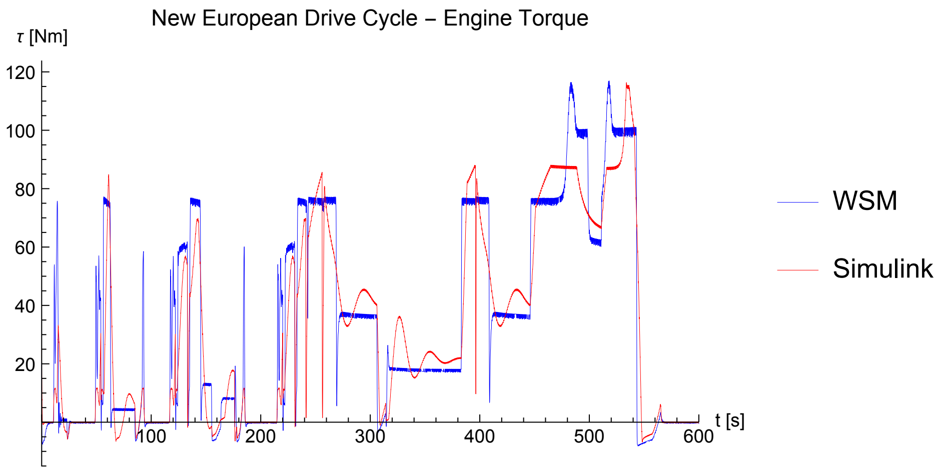


Figure 3.38: *Comparison of the engine torque between the* WSM *and the Simulink model.*

The torque is strongly connected to the fuel and air mass flow, the same effects are present here.

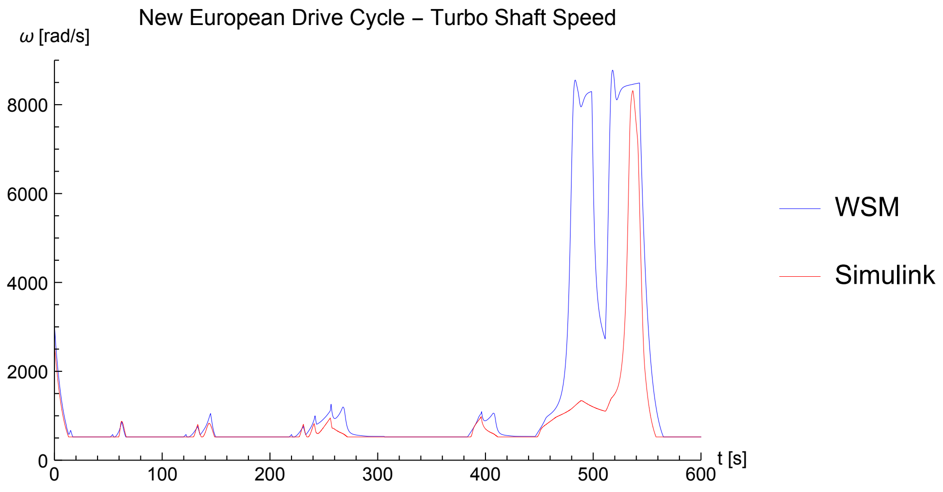


Figure 3.39: *Comparison of the angular velocity of the turbocharger between the* WSM *and the Simulink model.*

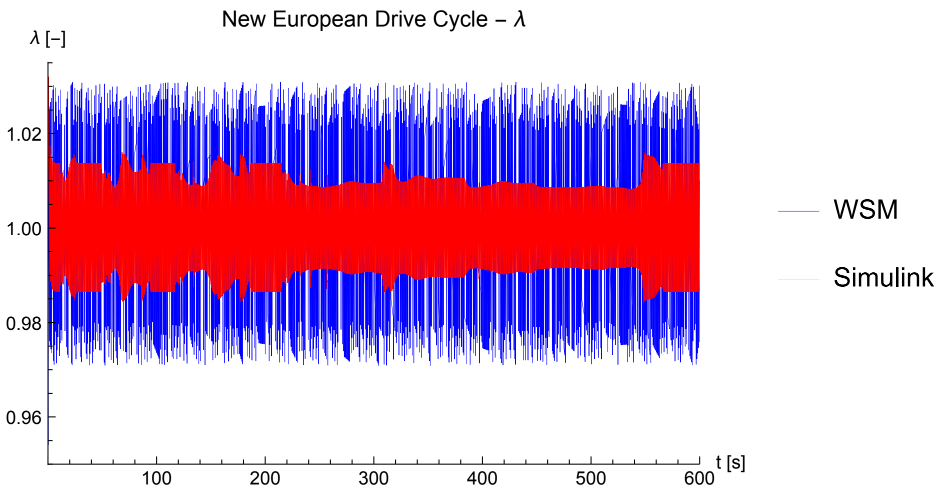


Figure 3.40: *Comparison of the lambda (proportion of oxygen) between the WSM and the Simulink model.*

The lambda value should vary around 1 with a variation around 0.03, this is the case and can be seen in figure 3.31. The larger variations are most likely due to the lambda controller, which has not been compared with the Simulink lambda controller.

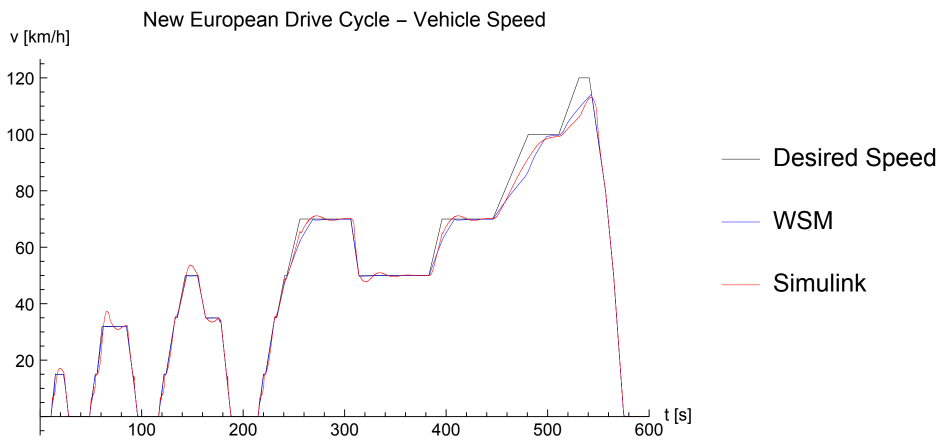


Figure 3.41: *Comparison of the speed between the WSM and the Simulink model. The black graph is the demanded speed.*

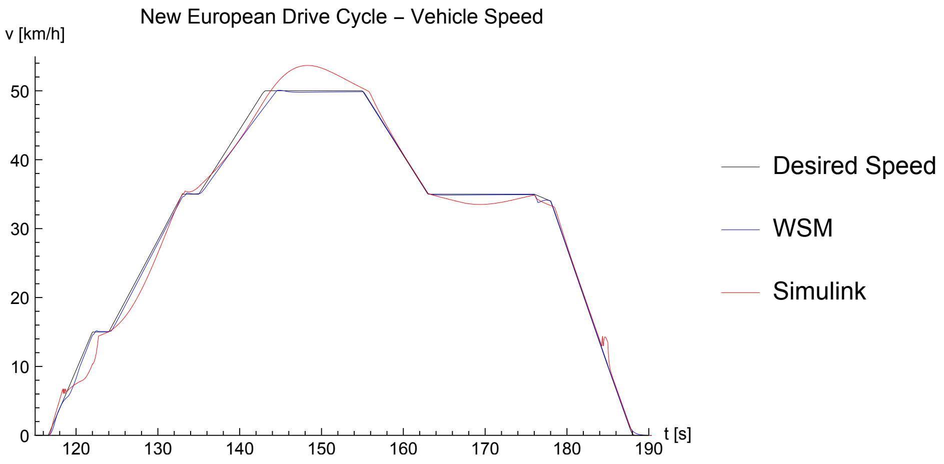


Figure 3.42: *Comparison of the speed between the WSM and the Simulink model in the time span between 115 and 190 seconds*

Deviation from the Simulink Model

Most of the deviations discussed for the model without a turbocharger are present here (see section 3.10.1). The increase in engine torque at the end of the simulation (which is not included in the Simulink simulation) is due to the different driver model. The driver is much more aggressive and suppresses oscillations, which can be seen in figure 3.42.
The WSM model has a higher fuel consumption due to problem with the fuel mass flow offset, but the same argument for downsizing can still be stated.

3.11 Industry Example

The provisional Industry Example is attached at the end of the thesis (see page 123). The layout of the example is presented as a Notebook (in *Mathematica*) for an overview, the final product will be integrated in the Wolfram web environment.

4 Conclusions

This chapter will summarize the conclusions drawn in the previous chapter and present suggestions for future work.

4.1 Summary

VehProLib is reconstructed and is functional to a acceptable degree, the parts which remains is stated in section 4.2. The individual components are shown to agree well with literature, which has been shown for the **`ControlVolume`**, **`Compressor`**, **`Turbine`**, **`BoostController`**, **`MVEMCylinder`**, **`Throttle`**, **`IncompressibleRestriction`** and the **`Intercooler`**.
The MVEM models with and without a turbocharger shows similar behaviour as the reference models showing that the components work together. Deviations are due to different solvers, step size and difference in some of the models.
It is important to understand that for the vehicle model to be as good as the Simulink model, there need so be done fine tuning against real data. When this is done the vehicle model can be used to study components in an environment close to the real world. This is the strongest argument for choosing Modelica as a simulation language compared to Simulink, because of the clean well structured interface between components, it is easy to change or replace component.

The new turbocharger package has support for a template structure where the fundamental structure (all the common equation and parameters) exists in a partial model which can be extended to create a complete component. All the parameters which are hardware dependent are defined on top of the partial model, this parameters can be fitted (and set in the WSM model) to real data with help of a Mathematica script.

Simulation results from the comparison between ideal gas law and Van der Waals equation of states indicates that the ideal gas law approximation is justifies during vehicle engine simulations.

The new design of the library is more user friendly and has a more compact and consistent design where the each package has received an self-explanatory icon.

The test library VehProTestLib includes a reference files (.mat) to all the working simulations and is prepared for automated tests.

4.2 Future Work

- Implement a complete engine controller for the in-cylinder model.
- Improve the graphical design of all the icons.
- Handle the gas components of higher dimensions in the right fashion.
- Extend the wheel and chassis model to move in the xy-plane.
- Change the signal bus to a proper bus using records.
- Move the equation of states into the gas model.
- Reintroduce the bearing friction into the models.
- Extend the VehProTestLib to cover all classes in VehProLib.
- Introduce support for Exhaust Gas Recirculation.
- Extend the Icon package to include more common icons.
- Create a library icon for VehProLib.
- Extend the documentation of components.

Appendix

A
Derivation of Van der Waals Constants

Let us rewrite Van der Waals equation (2.8) with only the pressure at the left hand side:

$$p = \frac{nRT}{V - nb} - a\left(\frac{n}{V}\right)^2 \tag{A.1}$$

To derive a and b we use the fact that at the critical point (the point p_c and V_c where liquid and vapour can coexist) is a inflection point in the PV-diagram, this yield the following equations:

$$\left(\frac{\partial p}{\partial V}\right)_T = 0 \tag{A.2}$$

$$\left(\frac{\partial^2 p}{\partial V^2}\right)_T = 0 \tag{A.3}$$

Use equation (A.2) and (A.3) on (A.1) to get the following relations:

$$\frac{2an}{V^3} = \frac{RT}{(V - nb)^2} \tag{A.4}$$

$$\frac{3an}{V^4} = \frac{RT}{(V - nb)^3} \tag{A.5}$$

Lets solve for V_c by divide (A.4) by (A.5):

$$\begin{aligned} \frac{\frac{2an}{V_c^3}}{\frac{3an}{V_c^4}} &= \frac{\frac{RT}{(V_c - nb)^2}}{\frac{RT}{(V_c - nb)^3}} \Rightarrow \\ \frac{2}{3}V_c &= V_c - nb \Leftrightarrow \\ V_c &= 3nb \end{aligned} \tag{A.6}$$

Next step is to solve for T_c by substituting (A.6) into (A.2):

$$\begin{aligned}
\frac{2an}{(3nb)^3} &= \frac{RT_c}{(3nb-nb)^2} \Rightarrow \\
RT_c &= \frac{2an(2nb)^2}{(3nb)^3} \Rightarrow \\
T_c &= \frac{8a}{27Rb} \qquad \text{(A.7)}
\end{aligned}$$

Finally we solve for p_c by substituting (A.6) and (A.7) into (A.1):

$$\begin{aligned}
p_c &= \frac{nR}{3nb-nb}\frac{8an}{27Rb} - a\left(\frac{n}{3nb}\right)^2 \Rightarrow \\
p_c &= \frac{4a}{27b^2} - \frac{a}{9b^2} \Rightarrow \\
p_c &= \frac{a}{27b^2} \qquad \text{(A.8)}
\end{aligned}$$

Now there is enough equations to express a and b in terms of p_c, p_c and R. Start with a by substituting (A.7) into (A.8):

$$\begin{aligned}
p_c &= \frac{a}{27}\left(\frac{27RT_c}{8a}\right)^2 \Rightarrow \\
a &= \frac{27T_c^2R^2}{64p_c} \qquad \text{(A.9)}
\end{aligned}$$

b by substituting (A.9) into (A.7):

$$\begin{aligned}
T_c &= \frac{8}{27Rb}\frac{27T_c^2R^2}{64p_c} \Rightarrow \\
b &= \frac{T_cR}{8p_c} \qquad \text{(A.10)}
\end{aligned}$$

Conclusion:

$$\begin{aligned}
a &= \frac{27T_c^2R^2}{64p_c} \\
b &= \frac{T_cR}{8p_c}
\end{aligned}$$

where p_c is the critical pressure, T_c is the critical temperature and R is the gas constant.

B Library Restoration

The following sections describes the changes which has been made to restore the library to a working state.

B.1 Overall Changes

The changes which were applied throughout the whole library is presented in the sections below.

B.1.1 InPort and OutPort

The concept of **`InPort`** and **`OutPort`** is the old method of **`RealInput`** and **`RealOutput`**, were handling a single signal is associated in the following way:

```
...Interfaces.InPort InPort(final n = 1); →
        ...Interfaces.RealInput InPort;

     InPort.inPort.signal[1] → InPort.u
```

Multiple signals is associated in the following way:

```
...Interfaces.InPort InPort(final n = 2); →
      ...Interfaces.RealInput InPort[2];

         InPort.inPort → InPort.u
```

or

```
   InPort.inPort.signal[2] → InPort.u[2]
```

B.1.2 Library2

Most of the packages used **`Icons.Library2`**, which is an old icon package which does not exist any more. **`Icons.Library`** is used instead. This is used for cosmetic reasons.

B.1.3 Package

Some components uses `package SI = Modelica.SIunits`, which does not work in WSM, `import` is used instead of `package`.

B.2 Specific Package Changes

SynchroController

The **`SynchroController`** is a block component which takes one in–argument and returns a vector with out–arguments (the number of out arguments is defined when the component is used). The in–argument selects which place in the out–vector that should be set to one, and put all the remaining places to zero.

Two small examples (5 out–arguments):

In = 2:

$$2 \rightarrow \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

In = 4:

$$4 \rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

B.2.1 Chassis

In the component Wheel the component **`ModelicaAdditions.Tables.CombiTable1D`** was replaced with **`Modelica.Blocks.Tables.CombiTable1D`** from the standard library. This involved some changes in notations such as, icol is now named columns.

B.2.2 Driveline

In the components **`FinalDrive`** and **`Gearbox`** were the old component **`GearEfficiency`** used. This component is obsolete and **`LossyGear`** should be used instead. **`LossyGear`** is a much more advanced component, and a small knowledge base about the component should be known before usage. **`LossyGear`** includes mesh efficiency from either flanges depending on which one is the driving force.

Driver

The original **`Driver`** model had several components which were not connected by connection lines, instead declared in the code. Some components which could be used from the standard library were reintroduced. The remade Driver can be seen in figure B.1, where all the problems stated above are taken care of.

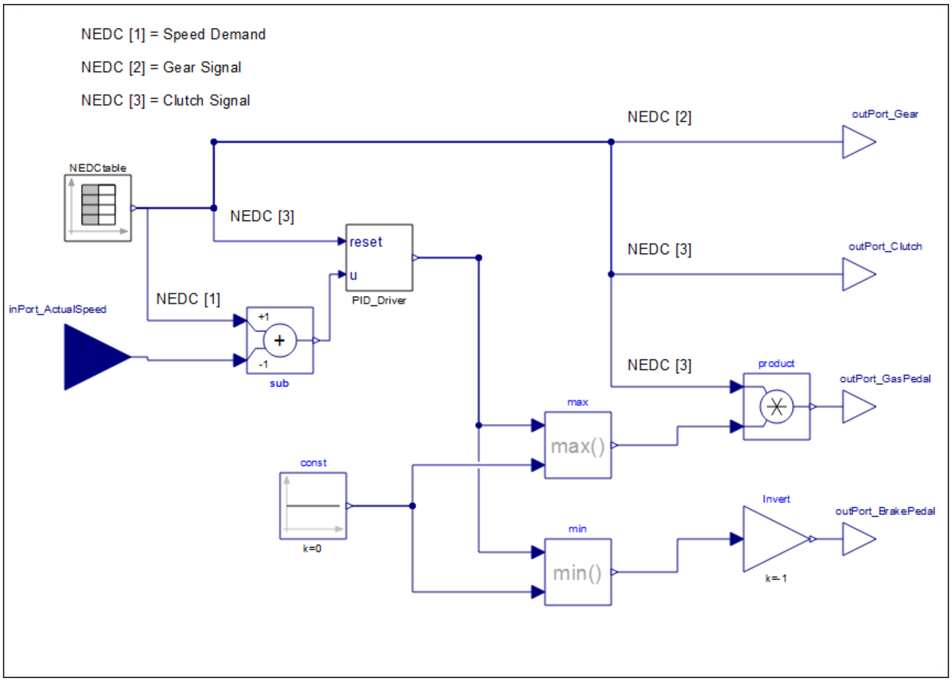


Figure B.1: The WSM illustration of the Driver model.

Gearbox

In figure B.2 the structure of the Gearbox can be seen. A new component (see section B.2) **`SynchroController`** has been added to improve the understanding of how the Gearbox works. In the previous model the connections for the synchros were hidden in the code. Due to simulation crashes and significant drops in simulation speed, the component **BearingFriction** has been excluded, until the problems have been solved.

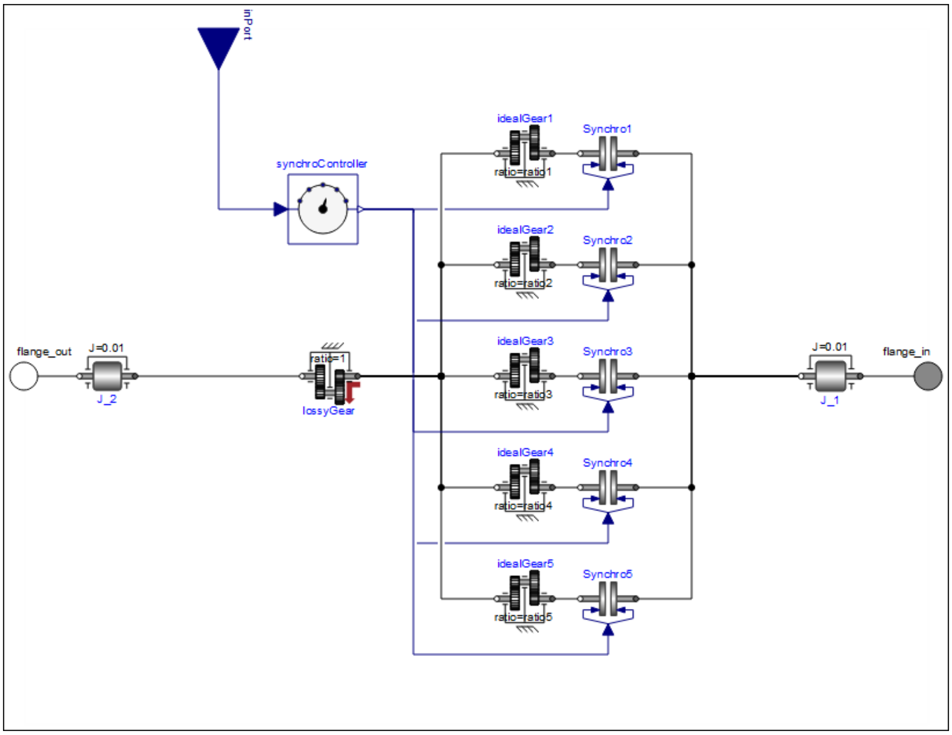


Figure B.2: The WSM illustration of the gearbox model.

B.2.3 DrivingCycles

The **DrivingCycles** were using the **ModelicaAdditions.Tables.CombiTableTime**, which is now replaced with **Modelica.Blocks.Sources.CombiTimeTable** from the standard library.

B.2.4 Engine

Cylinders

The engine model had problems with unit consistency (RPM and rad/s) and due to different formulations of equations compared to the Simulink reference model there were problems finding the errors. It turned out that the sensitivity on fitting parameters (to real data) is high, making it very important to identify that the parameters are the same as in the reference model. The torque generated by the cylinder is calculated by the following equations (stated in [Eriksson and Nielsen, 2014]):

The pump work (7.9.2), the difference in incoming and outgoing pressure:

$$W_p = V_D\left(p_{out} - p_{in}\right) \tag{B.1}$$

where V_D is the displacement volume for the whole engine.

The engine friction work can be modeled as a polynomial function of the engine speed (7.9.3):

$$W_{fr} = V_D\left(C_{fr0} + C_{fr1} * N + C_{fr2} * N^2\right) \tag{B.2}$$

where N is the engine speed in RPS and C_{fri} are fitting parameters.

Gross indicated work is produced from the combustion in the cylinder, in this process there are some efficiency losses (7.57) due to heat losses:

$$\eta_{g,i} = \left(1 - \frac{1}{r_c^{\gamma-1}}\right)\min(1,\lambda)\eta_{ign} \tag{B.3}$$

where r_c is the compression ratio, $\left(1 - \frac{1}{r_c^{\gamma-1}}\right)$ is the efficiency for an ideal Otto cycle, λ is the relative air to fuel ratio and η_{ign} is the combustion phasing losses due to non-optimal injection timing.

The gross indicated work (7.56) (work produced by burning fuel):

$$W_{g,i} = m_f q_{LHV} \eta_{g,i} \tag{B.4}$$

where m_f is the fuel mass and q_{LHV} is the lowering heating value for the incoming fuel.

The total torque produced by the engine can be described in terms of the work stated in equation B.1, B.2 and B.4:

$$M_e = \frac{W_{g,i} - W_p - W_{fr}}{n_r 2\pi} \tag{B.5}$$

where n_r is the number of revolution per stroke.

The enthalpy change in the engine can be defined as heat radiation losses (well explained by Per Öberg in figure 2.4 in [Öberg, 2009]):

$$\dot{H}_{in} - \dot{H}_{out} = m_f q_{LHV}(1 - \eta_{g,i}) \tag{B.6}$$

The engine friction work B.2, can also be modeled as the ETH model (7.9.3):

$$W_{fr} = V_D \xi_{aux}((0.464 + 0.0072\ S_p^{1.8})\Pi_{bl}10^5 + 0.0215\ \text{BMEP}))(0.075/B)^{1/2} \tag{B.7}$$

where S_p is the mean piston speed, ξ_{aux} is the load from auxiliary devices, Π_{bi} is the boost layout, B is the boar diameter and $BMEP$ is the Break Mean Effective Pressure defined as:

$$BMEP = \frac{W_{g,i} - W_p - W_{fr}}{V_D} \tag{B.8}$$

This model is also included in the MVEM cylinder model.

Restrictions

The component **`StandardRestrictionWArea`** uses the function *redeclare type* to set the dimension of a real signal to area. This is not possible in WSM, and has been solved by declaring the unit as area (unit = "Area").

Tubes

The old notation for the natural logarithm was ln, it is replaced with log.

Volume

In the component **`OpenCylinder`** there is an old notation for Q_flow (heat flow rate in Kelvin), Q_dot. A warning occur for the public variable PI, function variables that are not input/output must be protected.

B.2.5 Examples

The engine examples crashed due to the lack of flexibility between the dynamo and the engine. This was solved by adding a clutch.

Note: A damped spring adds flexibility to the system.

A few more MVEM engine examples(engine with turbocharger) are added to the engine example package.

B.2.6 GasProp

Two warnings occur when the **`IdealGasBase`** in the partial package is validated. Unknown dimension for a_low and a_high. This problem occurs when the gas has yet to be decided, meaning that the warning does not occur for complete gas models.

B.2.7 HEV - Hybrid Electric Vehicle

Components

In the component **`VariableResistor`** uses the function *redeclare type* to set the dimension of a real signal to resistance. This is not possible for WSM, and has been solved by declaring the unit as resistance (unit = "Resistance").

Battery

`ModelicaAdditions.Tables.CombiTable1D` was included in the Battery component and is handled in the same way as section B.2.1.

PM_motor

The component `Modelica.Electrical.Analog.Ideal.IdealSwitch` does not exist anymore, so the component `Modelica.Electrical.Analog.Ideal.IdealOpeningSwitch` is used instead. Their functionality is the same.

B.2.8 Tests

The tests package has been removed an the test models have been placed in a new test library called VehProTestLib, which is described in section 3.3.2.

B.3 Unclear Models

Some of the models are not used and have no well defined purpose, they are stated in the sections below.

B.3.1 IdealTransmission

A transmission has multiple definitions depending on which language base is used. In British English transmission refers to the whole drive-line [2015, 2015] and in American English it refers to the gearbox (this is the case in [Eriksson and Nielsen, 2014]). The current ***IdealTransmission*** model consists of a ideal gear and a ideal differential, which do not belong in any of the definitions. In the drive-line package it exists individual parts for constructing a drive-line of variable complexity, therefore considered the ***IdealTransmission*** to be redundant and is removed from the library.

B.3.2 FlexibleCrankPlus4C

The ***FlexibleCrankPlus4C*** model appeared to be in an early state of development, at the moment it has no connection to the surrounding environment. A model exists which has the same composition of a flexible crank called ***FlexibleCrank4C***. Therefore considered the ***FlexibleCrankPlus4C*** to be redundant and is removed from the library.

Note: The library contains a rigid crank shaft which is the simple model of the crank shaft.

B.3.3 SimpleTorque

This component appears to be intended as a simple model of the cylinder torque. The applied torque is based on a simple cylinder pressure model. This model is incomplete in several senses, it misses the connection which provides the information about incoming pressure and temperature and the outgoing pressure. It has no code which provides the torque relation to the pressure. If this should be used as a simple torque model, following semi-complete model could be used:

```
model SimpleTorque "A simple torque model based on the
    simple cylinder pressure model"
  extends VehProLib.Interfaces.GasPins.TwoPinStatic;
  import SI = Modelica.SIunits;
  import Modelica.SIunits.Conversions.*;
  parameter SI.Angle crank_offset = 0 "Crank angle offset";
  parameter SI.Angle ign_angle = from_deg(15) "Ignition
    angle which tells the start of combustion";
  parameter SI.Angle delta_theta = from_deg(15) "The
    approxiamte duration of the combustion";
  parameter SI.Area A = 0.05 "Cylinder area [m^2]";
  parameter SI.Radius r = 0.05 "crank radius";
  Modelica.Mechanics.Rotational.Interfaces.flange_b;
  SI.Angle theta;
  SI.Pressure p_cyl;
equation
  // Gas properties
  g.T = if i.p > o.p then i.T else o.T;
  g.p = if i.p > o.p then i.p else o.p;
  g.x = if i.p > o.p then i.x else o.x;
  // Flow relations
  i.Wx = g.x * VehProLib.Functions.
    SimpleMassFlow(theta, i.p, o.p, i.T);
  i.H = g.h * i.W;
  i.W = sum(i.Wx);
  // Torque calculation
  theta = flange_b.phi - crank_offset;
  p_cyl = VehProLib.Functions.
    SimplePressure(theta, ign_angle, delta_theta);
  // Torque = F*r, F = p*A, where r is the distanance from
    the axis point to the point where the force is applied,
    A the cylinder area.
  flange_b.tau = p_cyl * A * r;
end SimpleTorque;
```

The function ***SimpleMassFlow*** is not included in VehProLib and needs to be implemented for this model to be useful. ***SimpleMassFlow*** is the mass flow through the cylinder and should be based on the pressure difference, temperature and crank angle.

B.3.4 Tube1D_partial

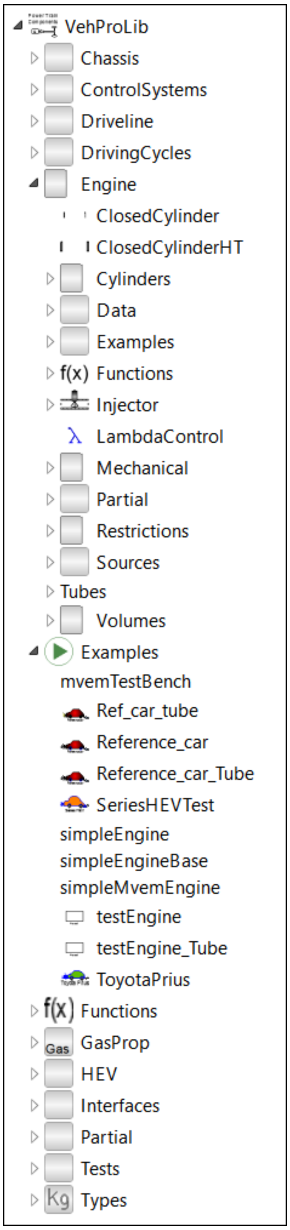
This model is not used except in one test model. There are several components missing for both the test model and the actual model, in this state it is not possible to simulate the models. To make them work again it is necessary to implement gas sources and interfaces which do not contain a gas model. Due to the fact that it exists components which handle tube systems including a gas model, this model appears redundant. Due to lack of information regarding the purpose and development of this model it is kept in the library (in the interfaces package).

B.3.5 MultiPort

The connector ***MultiPort*** is a component intended to be used as a bus. It relies on the obsolete connector ***RealPort*** (last used used in MSL version 1.6, current version is 3.2.1) which is a combination of ***RealInput*** and ***RealOutput***, meaning that it works both as an input and as a output. This component (includes ***Bus*** and ***RestBus***) is not used and cannot be simulated due to the invalid connectors. A working bus exists (with its own problems) discussed in section 3.2.3 and therefore considered the ***MultiPort*** components to be redundant and removed.

C

Library Design



```
VehProLib
  Chassis
  ControlSystems
  Driveline
  DrivingCycles
  Engine
    ClosedCylinder
    ClosedCylinderHT
    Cylinders
    Data
    Examples
    Functions
    Injector
    LambdaControl
    Mechanical
    Partial
    Restrictions
    Sources
    Tubes
    Volumes
  Examples
    mvemTestBench
    Ref_car_tube
    Reference_car
    Reference_car_Tube
    SeriesHEVTest
    simpleEngine
    simpleEngineBase
    simpleMvemEngine
    testEngine
    testEngine_Tube
    ToyotaPrius
  Functions
  GasProp
  HEV
  Interfaces
  Partial
  Tests
  Types
```

Figure C.1: *VehProLib seen in the class browser in WSM with the examples and engine packages expanded.*

It exists a variety of different libraries today, with different design choices. In this chapter three libraries are presented and their strengths and weaknesses are discussed.

C.1 Modelica Standard Library

The Modelica Standard Library (MSL) is q free to use library distributed by the Modelica Association [Association, 2015].

C.1.1 Library Hierarchy

The library hierarchy is divided into main packages (such as blocks) which then consist of different sub-packages of the same type and additional packages (such as types and interfaces), the structure can be seen in figure C.2.

Note: MSL is a large library and can rightfully be seen as a collection of smaller libraries. The connections between those libraries are in the most cases of the same magnitude as the connection between an externally developed library and MSL.

Examples are located as a sub-package at level two inside the main packages.

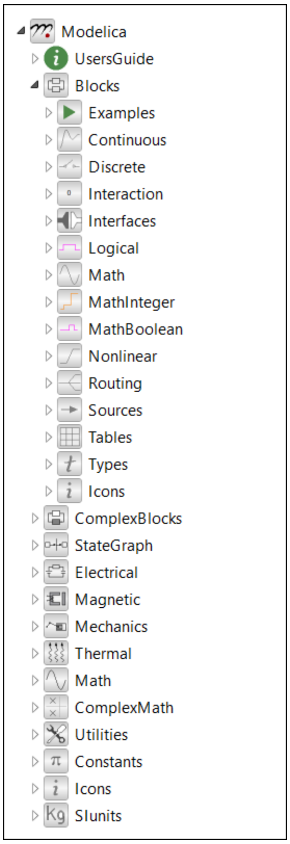


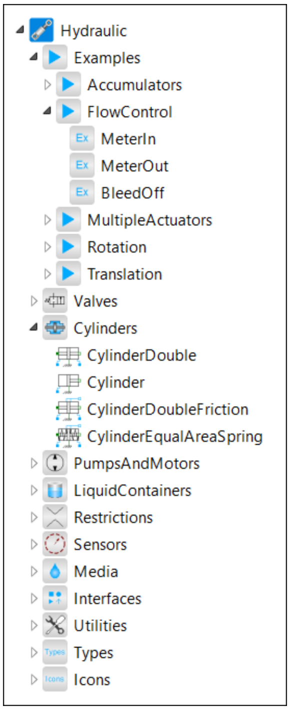
Figure C.2: *The library hierarchy with the blocks package expanded.*

C.2 Hydraulic Library

The Hydraulic library is a commercial library developed by Wolfram MathCore. The library contains components used for modeling hydraulic systems and the components are directly compatible with the components of the MSL (such as translational components) [MathCore, 2015a].

C.2.1 Library Hierarchy

The Hydraulic Library is a smaller library in comparison to MSL discussed in section C.1, it has comparable structure as one of the main packages in MSL. All the components and packages in the Hydraulic library are sorted to be as appealing to the user as possible. Meaning it is not sorted by alphabetical order, but instead by importance and occurrence. If there exists a packages alongside components, the packages are placed at the top. The examples package is following a more detailed structure where examples of the same type are placed in the same sub-package (example sub-packages are sorted in alphabetical order), this can be seen in figure C.3.



```
Hydraulic
  Examples
    Accumulators
    FlowControl
      MeterIn
      MeterOut
      BleedOff
    MultipleActuators
    Rotation
    Translation
  Valves
  Cylinders
    CylinderDouble
    Cylinder
    CylinderDoubleFriction
    CylinderEqualAreaSpring
  PumpsAndMotors
  LiquidContainers
  Restrictions
  Sensors
  Media
  Interfaces
  Utilities
  Types
  Icons
```

Figure C.3: *The library hierarchy with the cylinders and the flow control (sub-package in examples) packages expanded.*

C.3 ModelPlug Library

The ModelPlug library is a free to use library developed by Wolfram MathCore. The library is used to connect the real world to simulations in an easy way through an Arduino board or similar (open-source electronics platform). For instance, this allows simulations to measure real-time sensor data and process it. The processed data can be used to generate an output to the Arduino board to interact with the environment [MathCore, 2015b].

C.3.1 Library Hierarchy

ModelPlug is an even smaller library compared to the Hydraulic Library (see secrion C.2) and is structurally similar to one of the main packages of MSL (see section C.2) except that the additional packages (such as types and interfaces) are placed in a package called Internal, the structure can be seen in figure C.4. The Example package is placed second from the end instead of at the top.

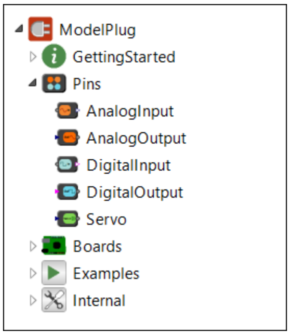


Figure C.4: *The library hierarchy with the pins package expanded.*

C.4 User Feedback

The following sections covers opinions and wishes about the library design from people connected to VehProLib and people which have worked with libraries and library development.

C.4.1 Vehicular Systems

Vehicular Systems wants the examples in VehProLib to be in a separate package, at the top level alongside the other main packages. The library should be clean with as little irrelevant models as possible. There exists a test bench for engines in the library, it should exist alongside the test models, and other test benches could be included (for instance for the drive-train).

C.4.2 Wolfram MathCore

In the library structure it is possible to sort the packages (they are sorted by alphabetical order by default) in different ways to make the user as comfortable when using the library as possible. Wolfram MathCore suggests that any kind of

user information or guide should be placed at the top followed by the examples. This would introduce the user to the library in a good way and give easy access to the examples when expanding the library. The rest of the library is suggested to be sorted after how common they are in models.
Sensor and block component is suggested to be placed at a top level for easy access. Partial models should be placed in the Interfaces package due to library standards, but there can exist a sub-package structure inside the Interfaces package.
The examples package is strongly suggested to be placed in a package at the top level, there can be sub-packages if they contain more than two examples. Wolfram MathCore prefers to have the tests and validation of models outside the library in a separate library, this also favours automated test suits (which may be of interests in the future).

C.5 Graphical Changes

Some of the icons in the library looked compressed in the class browser (this can be seen in figure C.1), the people at Wolfram MathCore explained this as a known Dymola bug. Many connectors were not symmetrical (left and right connector was not placed in the middle, causing the connection lines to be jagged), this could be caused by the same Dymola bug.

C.5.1 Signal Bus

The original signal bus had a custom made icon, for both the complete signal bus and the signal sub-bus which was not self explanatory. The MSL contains a standardized icons for both buses and sub-buses, which was chosen to become the new icon, the changes are illustrated in figure C.5 and C.6.

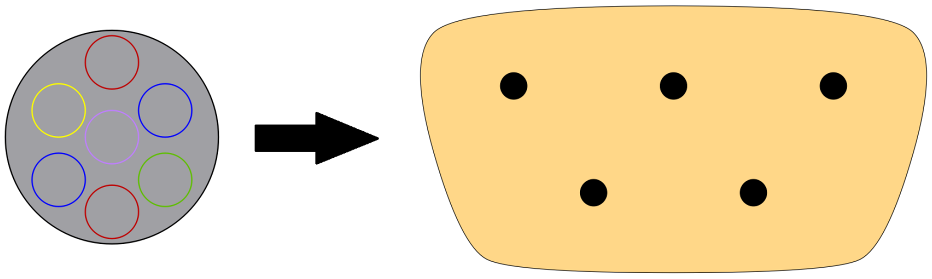


Figure C.5: *The icon for the signal bus, containing the illustration of the complete bus. The old icon is to the left, new is to the right.*

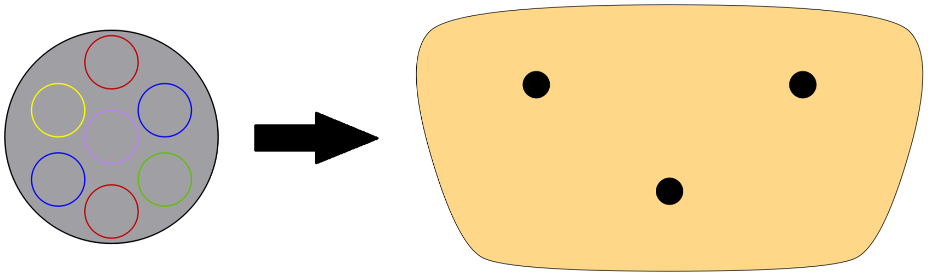


Figure C.6: The icon for the signal sub-bus, used for any type of sub-bus. The old icon is to the left, new is to the right.

C.5.2 Driver, Chassis and Engine Model

Purely for aesthetic reasons the driver, the car and the engine model icons have been changed to be more presentable. These models among some other is shown in the industry example (see section 2.10) which is used to promote the library. The old icons where self-explanatory but gave an impression which were considered non professional. The changes can be seen in figures C.7, C.8 and C.9.

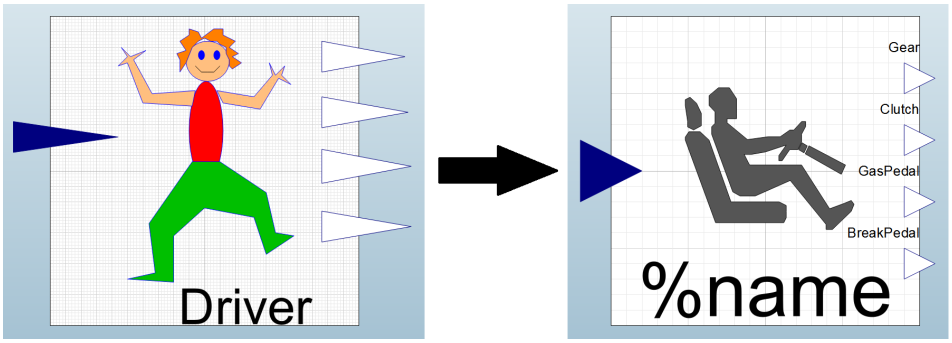


Figure C.7: The icon for the driver model, left is the original version, right is the current version.

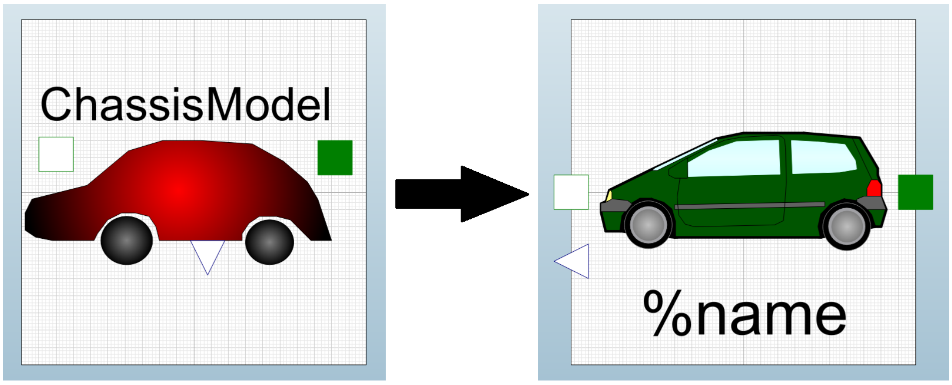


Figure C.8: *The icon for the chassis model, left is the original version, right is the current version.*

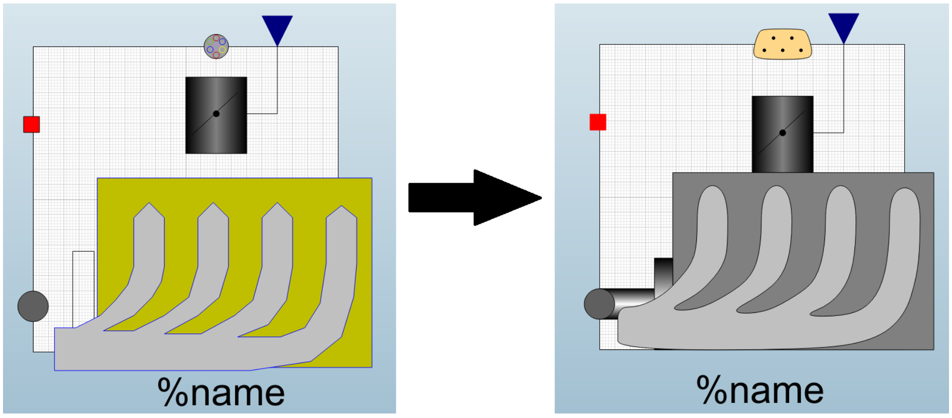


Figure C.9: *The icon for the engine model, left is the original version, right is the current version.*

C.5.3 Package Icons

All packages received icons that corresponding to the package content. This makes it easier for the user to distinguish between the packages and find the component he/she is looking for. The package names meet the requirements on its own, but a icon enhances the experience for the user. Consistency enhances the appearance and the usability of the library, and therefore are icons created for all the packages.

C.5.4 Function Icons

There are several functions included in VehProLib and originally they had a package icon not following the rest of the library, and the functions themselves had their names as icons. The idea of having all functions collected in a main package is kept, but each function receives a function icon (package and function icon can be seen in figure C.10) similar to how MSL and the Hydraulic library handle functions. This makes the functions easily accessible and recognizable.

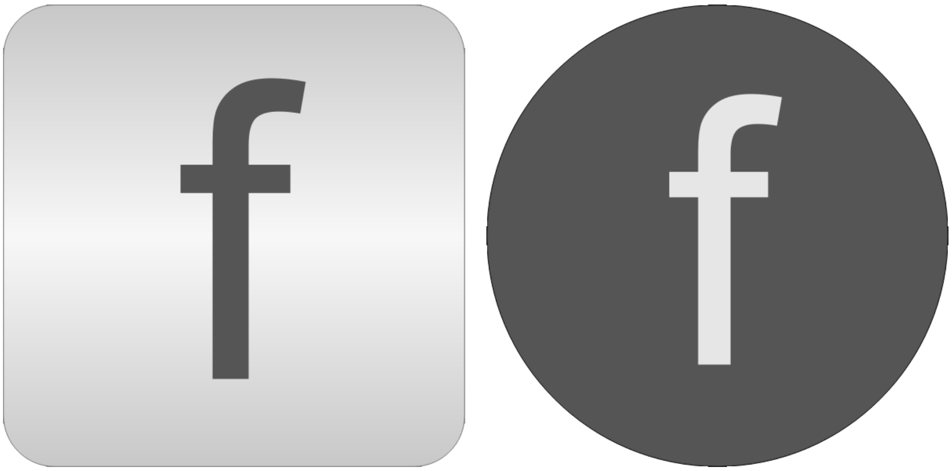


Figure C.10: *The icons for the functions, left is the package icon, right is the function icon.*

D
Mathematica Scripts

Compressor

```
Needs["WSMLink`"];
Needs["Notation`"];
Symbolize[ __ ];

ηmap =
  Flatten[Import[NotebookDirectory[] <> "Map\\etaC.mat", "MAT"]];
Nmap =
  Flatten[Import[NotebookDirectory[] <> "Map\\NcCorr.mat", "MAT"]];
Πc =
  Flatten[Import[NotebookDirectory[] <> "Map\\PiC.mat", "MAT"]];
ṁcorr,map = Flatten[Import[NotebookDirectory[] <> "Map\\WcCorr.mat", "MAT"]];
p01 =
  First[
   Flatten[Import[NotebookDirectory[] <> "Map\\p01.mat", "MAT"]]];
pref = First[Flatten[Import[NotebookDirectory[] <> "Map\\pCref.mat", "MAT"]]];
T01 =
  First[
   Flatten[Import[NotebookDirectory[] <> "Map\\T01.mat", "MAT"]]];
Tref = First[Flatten[Import[NotebookDirectory[] <> "Map\\TCref.mat", "MAT"]]];

lineThickness = 0.0005;
plotPoints = 100;

ηmea = ConstantArray[0, {5, 7, 2}];
ηmod = ConstantArray[0, {5, 7, 2}];
Πmea = ConstantArray[0, {5, 7, 2}];
Πmod = ConstantArray[0, {5, 7, 2}];

ηmea[[All, All, 1]] = ArrayReshape[ṁcorr,map, {5, 7}];
ηmea[[All, All, 2]] = ArrayReshape[ηmap, {5, 7}];
Πmea[[All, All, 1]] = ArrayReshape[ṁcorr,map, {5, 7}];
Πmea[[All, All, 2]] = ArrayReshape[Πc, {5, 7}];

ηmea[[Dimensions[ηmea][[1]], Dimensions[ηmea][[2]]] ] = {Null, Null};
Πmea[[Dimensions[ηmea][[1]], Dimensions[ηmea][[2]]] ] = {Null, Null};

Dc = 56 * 10^(-3);
rc = Dc / 2;
γ = 1.3139;
cp = 1200;

Nc = Nmap * Sqrt[T01 / Tref];
ωc = ṁcorr,map * Sqrt[p01 / pref] / Sqrt[T01 / Tref];
U2 = rc * Nc * Pi / 30;
```

```
Clear[ṁ_{corr}, ṁ_{corr,max}, Ψ_{max}];
Π_{max} = Map[Function[y, y^(γ/(γ - 1))],
    (Map[Function[x, x^2], N_c] * D_c^2 * Ψ_{max} / (2 * c_p * T_{01}) + 1)]; (* 8.54a *)
ṁ_{corr} = ṁ_{corr,max} * Sqrt[1 - (Π_c / Π_{max})^2];
ṁ_{corr,sol} =
  NMinimize[
   {Norm[ṁ_{corr,map} - ṁ_{corr}], (ṁ_{corr,max} ≥ 0.18) && (Ψ_{max} ≥ 0.002)}, {ṁ_{corr,max}, Ψ_{max}}
  ];
{ṁ_{corr,max}, Ψ_{max}} = {ṁ_{corr,max}, Ψ_{max}} /. Last[ṁ_{corr,sol}]
ṁ_{corr} = ṁ_{corr,max} * Sqrt[1 - (Π_c / Π_{max})^2];
```

```
{0.18, 0.00276847}
```

```
Clear[Q_{11}, Q_{12}, Q_{21}, Q_{22}, ṁ_{corr,η,max}, Π_{η,max}, η_{max}];
Q_η = {{Q_{11}, Q_{12}}, {Q_{21}, Q_{22}}};
χ = Thread[{ṁ_{corr,map} - ṁ_{corr,η,max},
     Map[Function[x, Sqrt[x - 1]], Π_c] - Sqrt[Π_{η,max} - 1]}]; (* 8.44 *)
η = η_{max} - Map[Function[x, x.Q_η.x], χ]; (* 8.42 *)
η_{sol} =
  NMinimize[
   {Norm[η_{map} - η], Π_{η,max} ≥ 1 && η_{max} ≥ 0 && η_{max} ≤ 1},
   {Q_{11}, Q_{12}, Q_{21}, Q_{22}, ṁ_{corr,η,max}, Π_{η,max}, η_{max}}
  ];
{Q_{11}, Q_{12}, Q_{21}, Q_{22}, ṁ_{corr,η,max}, Π_{η,max}, η_{max}} =
 {Q_{11}, Q_{12}, Q_{21}, Q_{22}, ṁ_{corr,η,max}, Π_{η,max}, η_{max}} /. Last[η_{sol}]
η = η_{max} - Map[Function[x, x.Q_η.x], χ];
```

```
{90.5045, -6.7728, -6.0504, 0.822306, 0.0842128, 1.9561, 0.82064}
```

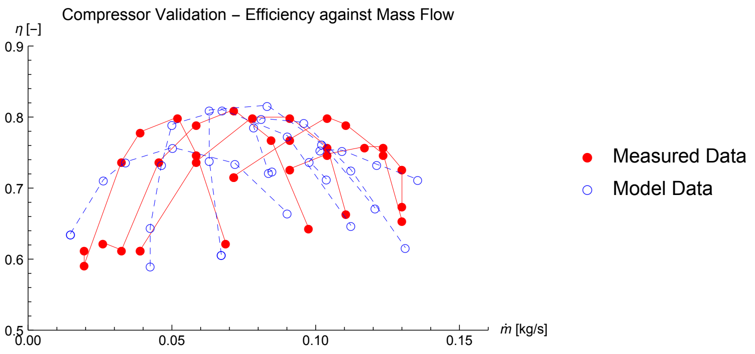
```
Clear[η_{min}]
η_{min} = Min[η_{map} / 2]
η_2 = Map[Function[x, Max[x, η_{min}]], η];
```

```
0.296075
```

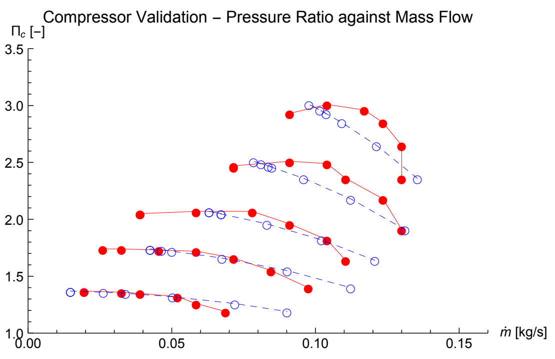
```
η_{mod}[[All, All, 1]] = ArrayReshape[ṁ_{corr}, {5, 7}];
η_{mod}[[All, All, 2]] = ArrayReshape[η_2, {5, 7}];
Π_{mod}[[All, All, 1]] = ArrayReshape[ṁ_{corr}, {5, 7}];
Π_{mod}[[All, All, 2]] = ArrayReshape[Π_c, {5, 7}];

η_{mod}[[Dimensions[η_{mod}][[1]], Dimensions[η_{mod}][[2]]]] = {Null, Null};
Π_{mod}[[Dimensions[Π_{mod}][[1]], Dimensions[Π_{mod}][[2]]]] = {Null, Null};
```

```
range = {{0, 0.16}, {0.5, 0.9}};
etaDPlot0 = ListPlot[{{Null}, {Null}},
    PlotLabel → "Compressor Validation - Efficiency against Mass Flow",
    AxesLabel → {"ṁ [kg/s]", "η [-]"},
    PlotLegends →  {"Measured Data", "Model Data"},
    PlotStyle →
     {{Red, Thickness[lineThickness]}, {Blue, Thickness[lineThickness]}},
    PlotMarkers → {●, ○},
    AxesOrigin → {0, 0.5},
    PlotRange → range
   ];
etaDPlots = {etaDPlot0};
AppendTo[etaDPlots, ListLinePlot[ηmea,
    PlotStyle → {
      {Red, Thickness[lineThickness]},
      {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]},
      {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]}}
   ]];
AppendTo[etaDPlots, ListPlot[ηmea,
    PlotStyle → {Red},
    PlotMarkers → {●}
   ]];
AppendTo[etaDPlots, ListLinePlot[ηmod,
    PlotStyle → {
      {Blue, Thickness[lineThickness], Dashed},
      {Blue, Thickness[lineThickness], Dashed},
      {Blue, Thickness[lineThickness], Dashed}, {Blue, Thickness[lineThickness],
       Dashed}, {Blue, Thickness[lineThickness], Dashed}}
   ]];
AppendTo[etaDPlots, ListPlot[ηmod,
    PlotStyle → {Blue},
    PlotMarkers → {○}
   ]];
etaDPlot = Show[etaDPlots]
```



```
range = {{0, 0.16}, {1, 3.5}};
PIDPlot0 = ListPlot[{{Null}, {Null}},
    PlotLabel → "Compressor Validation - Pressure Ratio against Mass Flow",
    AxesLabel → {"ṁ [kg/s]", "Πc [-]"},
    PlotLegends → {"Measured Data", "Model Data"},
    PlotStyle →
     {{Red, Thickness[lineThickness]}, {Blue, Thickness[lineThickness]}},
    PlotMarkers → {●, ○},
    AxesOrigin → {0, 1},
    PlotRange → range
   ];
PIDPlots = {PIDPlot0};
AppendTo[PIDPlots, ListLinePlot[Πmea,
    PlotStyle → {
      {Red, Thickness[lineThickness]},
      {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]},
      {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]}}
   ]];
AppendTo[PIDPlots, ListPlot[Πmea,
    PlotStyle → {Red},
    PlotMarkers → {●}
   ]];
AppendTo[PIDPlots, ListLinePlot[Πmod,
    PlotStyle → {
      {Blue, Thickness[lineThickness], Dashed},
      {Blue, Thickness[lineThickness], Dashed},
      {Blue, Thickness[lineThickness], Dashed}, {Blue, Thickness[lineThickness],
       Dashed}, {Blue, Thickness[lineThickness], Dashed}}
   ]];
AppendTo[PIDPlots, ListPlot[Πmod,
    PlotStyle → {Blue},
    PlotMarkers → {○}
   ]];
PIDPlot = Show[PIDPlots]
```



```
locationP = "Plots" <> "" <> "/";
If[DirectoryQ[locationP],
 ,
 dirP = CreateDirectory[];
 CopyFile[dirP, NotebookDirectory[] <> locationP, "OverwriteTarget" → True];
]

Export[NotebookDirectory[] <>
   locationP <> "Build_Compressor_Verify_PI.pdf", PIDPlot];
Export[NotebookDirectory[] <> locationP <> "Build_Compressor_Verify_eta.pdf",
  etaDPlot];

(* Set parameters decided from the mathematical models *)

WSMSetValues["VehProLib.Engine.Turbocharger.CompressorWResRegionExt",
  {
   "D_init" → Floor[D_c, 0.0001],
   "m_corr_max_init" → Floor[ṁ_corr,max, 0.0001],
   "Psi_max_init" → Floor[Ψ_max, 0.0001],
   "Q_eta_init" → Floor[{{Q11, Q12}, {Q21, Q22}}, 0.0001],
   "m_corr_at_eta_max" → Floor[ṁ_corr,η,max, 0.0001],
   "PI_at_eta_max_init" → Floor[Π_η,max, 0.0001],
   "eta_max_init" → Floor[η_max, 0.0001],
   "eta_min_init" → Floor[η_min, 0.0001],
   "p_ref_init" → Floor[p_ref, 0.0001],
   "T_ref_init" → Floor[T_ref, 0.0001]
  }
 ];
```

Turbine

```
Needs["WSMLink`"];
Needs["Notation`"];
Symbolize[ __ ];

η_map = Flatten[Import[NotebookDirectory[] <> "Map\\etaT.mat", "MAT"]];
Π_t = Flatten[Import[NotebookDirectory[] <> "Map\\PiT.mat", "MAT"]];
TFP_map = Flatten[Import[NotebookDirectory[] <> "Map\\TFP.mat", "MAT"]];
TSP_map = Flatten[Import[NotebookDirectory[] <> "Map\\TSP.mat", "MAT"]];
p_04 = First[Flatten[Import[NotebookDirectory[] <> "Map\\p04.mat", "MAT"]]];
T_03 = First[Flatten[Import[NotebookDirectory[] <> "Map\\T03.mat", "MAT"]]];

lineThickness = 0.0005;
plotPoints = 100;

η_mea = ConstantArray[0, {5, 6, 2}];
η_mod = ConstantArray[0, {5, 6, 2}];
Π_mea = ConstantArray[0, {5, 6, 2}];
Π_mod = ConstantArray[0, {5, 6, 2}];


(* Π_t = p_i/p_o*)
Π_t = 1/Π_t;
η_mea[[All, All, 2]] = ArrayReshape[η_map, {5, 6}];
Π_mea[[All, All, 1]] = ArrayReshape[Π_t, {5, 6}];
Π_mea[[All, All, 2]] = ArrayReshape[TFP_map, {5, 6}];

D_t = 52.2 * 10^(-3);
γ = 1.3139;
c_p = 1200;

r_t = D_t/2;
(*Π_t=p_03/p_04;*)
W_t = TFP_map * 10^(-3) * Π_t / Sqrt[T_03];
N_t = TSP_map * Sqrt[T_03];
ω_t = N_t * 2 * π/60;

BSR_map = ω_t * r_t * Map[Function[x, 1/Sqrt[2 * c_p * T_03 (1 - x^((γ - 1)/γ))]], Π_t];

η_mea[[All, All, 1]] = ArrayReshape[BSR_map, {5, 6}];
```

```
Clear[TFP_mod, TFP_sol, TFP_max, TFP_exp];
TFP_mod = TFP_max * Map[Function[x, Sqrt[1 - x^TFP_exp]], Π_t];
 (* 8.54a *)
TFP_sol = NMinimize[Norm[TFP_map - TFP_mod], {TFP_max, TFP_exp}];
{TFP_max, TFP_exp} = {TFP_max, TFP_exp} /. Last[TFP_sol]
TFP_mod = TFP_max * Map[Function[x, Sqrt[1 - x^TFP_exp]], Π_t];
```

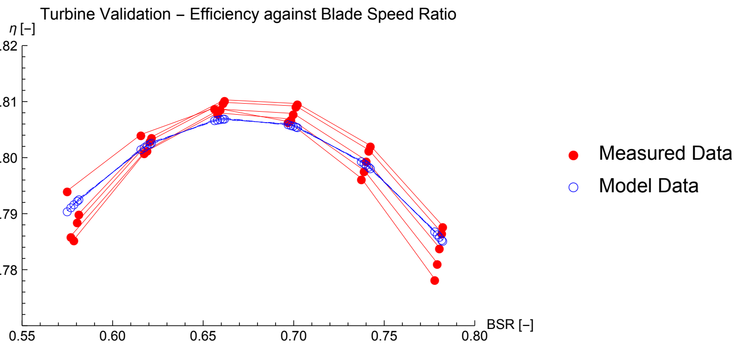
```
{0.00535703, 1.45034}
```

```
Clear[η, η_max, BSR_max];
η = η_max * Map[Function[x, 1 - ((x - BSR_max) / BSR_max)^2], BSR_map];(* 8.42 *)
η_sol = NMinimize[Norm[η_map - η], {η_max, BSR_max}];
{η_max, BSR_max} = {η_max, BSR_max} /. Last[η_sol]
η = η_max * Map[Function[x, 1 - ((x - BSR_max) / BSR_max)^2], BSR_map];
```

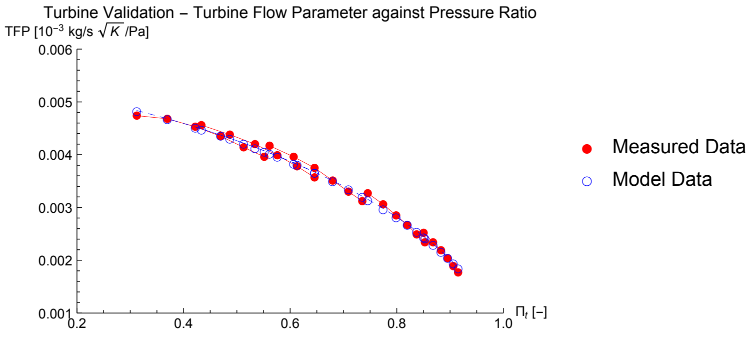
```
{0.807244, 0.671524}
```

```
η_mod[[All, All, 1]] = ArrayReshape[BSR_map, {5, 6}];
η_mod[[All, All, 2]] = ArrayReshape[η, {5, 6}];
Π_mod[[All, All, 1]] = ArrayReshape[ Π_t, {5, 6}];
Π_mod[[All, All, 2]] = ArrayReshape[TFP_mod, {5, 6}];
```

```
range = {{0.55, 0.8}, {0.77, 0.82}};
etaDPlot0 = ListPlot[{{0}, {0}},
   PlotLabel → "Turbine Validation - Efficiency against Blade Speed Ratio",
   AxesLabel → {"BSR [-]", "η [-]"},
   PlotLegends → {"Measured Data", "Model Data"},
   PlotStyle →
    {{Red, Thickness[lineThickness]}, {Blue, Thickness[lineThickness]}},
   PlotMarkers → {●, ○},
   AxesOrigin → {0.55, 0.77},
   PlotRange → range
  ];
Clear[etaDPlots];
etaDPlots = {etaDPlot0};
AppendTo[etaDPlots, ListLinePlot[η_mea,
   PlotStyle → {
     {Red, Thickness[lineThickness]},
     {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]},
     {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]}}
  ]];
AppendTo[etaDPlots, ListPlot[η_mea,
   PlotStyle → {Red},
   PlotMarkers → {●}
  ]];
AppendTo[etaDPlots, ListLinePlot[η_mod,
   PlotStyle → {
     {Blue, Thickness[lineThickness], Dashed},
     {Blue, Thickness[lineThickness], Dashed},
     {Blue, Thickness[lineThickness], Dashed}, {Blue, Thickness[lineThickness],
      Dashed}, {Blue, Thickness[lineThickness], Dashed}}
  ]];
AppendTo[etaDPlots, ListPlot[η_mod,
   PlotStyle → {Blue},
   PlotMarkers → {○}
  ]];
etaDPlot = Show[etaDPlots]
```


Turbine Validation – Efficiency against Blade Speed Ratio
η [–]
0.82
0.81
0.80
0.79
0.78
BSR [–]
0.55
0.60
0.65
0.70
0.75
0.80
● Measured Data
○ Model Data

```
range = {{0.2, 1.0}, 10^(-3) {1, 6}};
PIDPlot0 = ListPlot[{{0}, {0}},
   PlotLabel →
    "Turbine Validation - Turbine Flow Parameter against Pressure Ratio",
   AxesLabel → {"Πt [-]", "TFP [10^-3 kg/s √K /Pa]"},
   PlotLegends → {"Measured Data", "Model Data"},
   PlotStyle →
    {{Red, Thickness[lineThickness]}, {Blue, Thickness[lineThickness]}},
   PlotMarkers → {●, ○},
   AxesOrigin → {0.2, 10^(-3)},
   PlotRange → range
  ];
PIDPlots = {PIDPlot0};
AppendTo[PIDPlots, ListLinePlot[Πmea,
   PlotStyle → {
     {Red, Thickness[lineThickness]},
     {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]},
     {Red, Thickness[lineThickness]}, {Red, Thickness[lineThickness]}}
  ]];
AppendTo[PIDPlots, ListPlot[Πmea,
   PlotStyle → {Red}
  ]];
AppendTo[PIDPlots, ListLinePlot[Πmod,
   PlotStyle → {
     {Blue, Thickness[lineThickness], Dashed},
     {Blue, Thickness[lineThickness], Dashed},
     {Blue, Thickness[lineThickness], Dashed}, {Blue, Thickness[lineThickness],
      Dashed}, {Blue, Thickness[lineThickness], Dashed}}
  ]];
AppendTo[PIDPlots, ListPlot[Πmod,
   PlotStyle → {Blue},
   PlotMarkers → {○}
  ]];
PIDPlot = Show[PIDPlots]
```



```
locationP = "Plots" <> "" <> "/";
If[DirectoryQ[locationP],
  ,
 dirP = CreateDirectory[];
 CopyFile[dirP, NotebookDirectory[] <> locationP, "OverwriteTarget" → True];
]

Export[
  NotebookDirectory[] <> locationP <> "Build_Turbine_Verify_PI.pdf", PIDPlot];
Export[NotebookDirectory[] <> locationP <>
   "Build_Turbine_Verify_eta.pdf", etaDPlot];

(* Set parameters decided from the mathematical models *)

WSMSetValues["VehProLib.Engine.Turbocharger.Turbine",
  {
   "D_init" → Floor[D_t, 0.0001],
   "eta_min_init" → Floor[Min[η_map / 2], 0.0001],
   "eta_max_init" → Floor[η_max, 0.0001],
   "BSR_max_init" → Floor[BSR_max, 0.0001],
   "TFP_max_init" → Floor[TFP_max, 0.0001],
   "TFP_exp_init" → Floor[TFP_exp, 0.0001]
  }
 ];
```

E New Components

E.1 Turbocharger

E.1.1 Compressor Template

```
partial model Compressor_Template "Model of a
    compressor based of the Generic Compressor Model"
  extends VehProLib.Interfaces.GasPins.TwoPinStatic
    (i(n = g.n), o(n = g.n));
  import SI = Modelica.SIunits;
  parameter SI.Length D = 0.1 "Compressor Diameter";
  Modelica.Mechanics.Rotational.Interfaces.Flange_b
    flange_b;
  //
  SI.Temperature T_out "Outgoing Temperature";
  SI.AngularVelocity w "Incomming angular velocity";
  SI.MassFlowRate m_dot " The massflowrate into the
    compressor";
  SI.Power W_dot "Power from the shaft";
  Real PI "Pressure ratio";
  Real eta "Compressor efficiency";
protected
  SI.MassFlowRate f_m "Mass flow function that estimate
    the information extracted from the compressor map";
  Real f_eta "Efficiency function that estimate the
    information extracted from the compressor map";
  Real N "Compressor shaft speed in revolution per
    second";
  SI.SpecificHeatCapacityAtConstantPressure c_p = g.c_p
```

```
    "Specific heat capacity at constant pressure";
  Real gamma = g.gamma "Ratio of specific heats";
  Real R = g.R "Gas constant";
equation
  // Compressor flow
  g.T = i.T;
  g.p = i.p;
  g.x = i.x;
  //
  // Regular flow
  i.Wx = m_dot * g.x;
  i.W = sum(i.Wx);
  i.H = i.W * g.h;
  //
  // Pressure ratio
  PI = o.p / i.p;
  //
  // Shaft
  if abs(der(flange_b.phi)) < Modelica.Constants.small
    then
    // To avoid division by 0
      (several places is divided by w_c)
    w = sqrt(Modelica.Constants.small);
  else
    w = der(flange_b.phi);
  end if;
  //
  N = w * 60 / (2 * Modelica.Constants.pi);
  flange_b.tau = W_dot / w;
  //
  // Below is the Generic Compressor Model
  // (8.5a):
  m_dot = f_m;
  // (8.5b):
  eta = f_eta;
  // (8.5c):
  T_out = g.T +
    g.T / eta * (max(PI, 0) ^ ((gamma - 1) / gamma) - 1);
  // (8.5d):
  W_dot = m_dot * c_p * max(T_out - g.T, 0);
end Compressor_Template;
```

E.1.2 Complete Compressor

```
model CompressorWResRegionExt "Complete compressor with
    Ellipse Extended to Restriction Region Model 8.9 and
    Quadratic form Compressor Efficiency Model 8.5"
  extends VehProLib.Interfaces.TurbochargerTemplates.
    Compressor_Template(D = D_init);
  import SI = Modelica.SIunits;
  //
  constant SI.Length D_init = 0.056 "Compressor Diameter";
  constant Modelica.SIunits.Efficiency eta_min_init = 0.2960
    "Minimum efficency";
  constant Modelica.SIunits.Efficiency eta_max_init = 0.8206
    "Maximum efficency";
  constant Modelica.SIunits.MassFlowRate m_corr_max_init =
    0.1804 "Tuning parameter";
  constant Modelica.SIunits.MassFlowRate
    m_corr_at_eta_max_init = 0.0842 "Tuning parameter";
  constant Real Psi_max_init = 0.0027 "Tuning parameter";
  constant Real PI_at_eta_max_init = 1.9561 "Tuning
    parameter";
  constant Real Q_eta_init[2, 2] = {{90.5045, -6.7729},
    {-6.0504, 0.8223}} "Tuning parameters";
  constant Modelica.SIunits.Temperature
    T_ref_init(displayUnit = "K") = 293.1499;
  constant Modelica.SIunits.Pressure
    p_ref_init(displayUnit = "Pa") = 101300.;
  //
  parameter Modelica.SIunits.Efficiency eta_min =
    eta_min_init "Minimum efficency";
  parameter Modelica.SIunits.Efficiency eta_max =
    eta_max_init "Maximum efficency";
  parameter Modelica.SIunits.MassFlowRate m_corr_max =
    m_corr_max_init "Tuning parameter";
  parameter Modelica.SIunits.MassFlowRate m_corr_at_eta_max
    = 0.0842 "Tuning parameter";
  parameter Real Psi_max = Psi_max_init "Tuning parameter";
  parameter Real PI_at_eta_max = PI_at_eta_max_init "Tuning
    parameter";
  parameter Real Q_eta[2, 2] = Q_eta_init "Tuning
    parameters";
  parameter Modelica.SIunits.Temperature
    T_ref(displayUnit = "K") = T_ref_init;
  parameter Modelica.SIunits.Pressure
    p_ref(displayUnit = "Pa") = p_ref_init;
  //
  Real chi[2];
  Real m_corr;
  Real PI_max;
equation
```

```
  // (8.54a):
  PI_max = (N^2 * D^2 * Psi_max / (2 * c_p * i.T) + 1)
    ^(gamma / (gamma - 1));
  // (8.54b):
  m_corr = m_corr_max * sqrt(max(1 - (PI / PI_max) ^ 2, 0));
  // (8.41):
  chi = {m_corr - m_corr_at_eta_max,
    sqrt(max(PI - 1, 0)) - sqrt(PI_at_eta_max - 1)};
  // (8.19):
  f_m = m_corr * (i.p / p_ref) / sqrt(i.T / T_ref);
  // (8.42):
  f_eta = max(eta_max - chi * Q_eta * chi, eta_min);
end CompressorWResRegionExt;
```

E.1.3 Turbine Template

```
partial model Turbine_Template "Model of a turbine based of
   the Generic Turbine Model"
  extends VehProLib.Interfaces.GasPins.TwoPinStatic
    (i(n = g.n), o(n = g.n));
  import SI = Modelica.SIunits;
  parameter SI.Length D = 0.05 "Turbine diameter";
  Modelica.Mechanics.Rotational.Interfaces.Flange_a
    flange_a;
  //
  SI.Temperature T_out "Outgoing Temperature";
  SI.AngularVelocity w "Outgoing angular velocity";
  SI.MassFlowRate m_dot " The massflowrate out from the
    turbine";
  SI.Power W_dot "Power to the shaft";
  Real PI "Pressure ratio";
  Real eta "Compressor efficiency";
protected
  SI.MassFlowRate f_m "Mass flow function that estimate the
    information extracted from the turbine map";
  Real f_eta "Efficiency function that estimate the
    information extracted from the turbine map";
  Real N "Turbine shaft speed in revolution per second";
  SI.SpecificHeatCapacityAtConstantPressure c_p = g.c_p
    "Specific heat capacity at constant pressure";
  Real gamma = g.gamma "Ratio of specific heats";
  Real R = g.R "Gas constant";
equation
  // Turbine flow
  g.T = i.T;
  g.p = i.p;
  g.x = i.x;
  //
  // Regular flow
  i.Wx = m_dot * g.x;
  i.W = sum(i.Wx);
  //
  i.H = i.W * g.h;
  //
  // Pressure Ratio
  PI = min(o.p / i.p, 1);
  //
  // Shaft
  if abs(der(flange_a.phi)) < Modelica.Constants.small then
    // To avoid division by 0
      (several places is divided by w_t)
    w = sqrt(Modelica.Constants.small);
  else
    w = der(flange_a.phi);
  end if;
  N = w / (2 * Modelica.Constants.pi);
  flange_a.tau = W_dot / w;
  //
  // Below is the Generic Turbine Model
  // (8.9a):
  m_dot = f_m;
  // (8.9b):
  eta = f_eta;
  // (8.9c):
  T_out = g.T - eta * g.T * (1 - PI ^ ((gamma - 1) /
    gamma));
  // (8.9d):
  W_dot = -m_dot * c_p * max(g.T - T_out, 0);
end Turbine_Template;
```

E.1.4 Complete Turbine

```
model Turbine "Complete turbine with Modified Square Root
    Turbine Flow Model 8.14 and Turbine Efficiency with BSR"
  extends VehProLib.Interfaces.TurbochargerTemplates.
    Turbine_Template(D = D_init);
  import SI = Modelica.SIunits;
  constant SI.Length D_init = 52.2e-3 "Turbine diameter";
  constant Real TFP_max_init = 0.0053 "Tuning parameter";
  constant Real TFP_exp_init = 1.4503 "Tuning parameter";
  constant SI.Efficiency eta_min_init = 0.3891
    "Minimum efficency";
  constant SI.Efficiency eta_max_init = 0.8072
    "Maximum efficency";
  constant Real BSR_max_init = 0.0909
    "Maximum efficiency blade speed ratio";
  //
  parameter Real TFP_max = TFP_max_init "Tuning parameter";
  parameter Real TFP_exp = TFP_exp_init "Tuning parameter";
  parameter SI.Efficiency eta_min = eta_min_init
    "Minimum efficency";
  parameter SI.Efficiency eta_max = eta_max_init
    "Maximum efficency";
  parameter Real BSR_max = BSR_max_init
    "Maximum efficiency blade speed ratio";
  //
  Real BSR "Blade Speed Ratio";
  Real TFP "Turbine Flow Parameter";
equation
  // section 8.7.2:
  BSR = D / 2 * w / sqrt(max(2 * c_p * g.T *
    (1 - PI ^ ((gamma - 1) / gamma)), 0.001));
  // Square Root Turbine Flow (8.60):
  TFP = TFP_max * sqrt(max(1 - PI ^ TFP_exp, 0));
  //
  f_m = TFP / sqrt(T_out) * o.p / 1000;
  // Turbine Efficiency with BSR (8.62):
  f_eta = min(max(eta_max * (1 -
    ((BSR - BSR_max) / BSR_max) ^ 2), eta_min), eta_max);
end Turbine;
```

E.1.5 Turbo Shaft Template

```
partial model TurboShaft_Template "Model of a turbine shaft
    based of Model 8.3 Turbo Shaft Dynamics"
  extends Modelica.Mechanics.Rotational.Interfaces.
    PartialTwoFlanges;
  import SI = Modelica.SIunits;
  parameter SI.Inertia J(min = 0) = 3e-5
    "Moment of inertia";
  parameter StateSelect stateSelect = StateSelect.default
    "Priority to use phi and w as states";
  parameter SI.AngularVelocity outMin = 5e3 / 30 *
    Modelica.Constants.pi "Lower limit of output";
  parameter SI.AngularVelocity w_start = 5e3 / 30 *
    Modelica.Constants.pi "Initial or guess value of output
    (must be in the limits outMin ..)";
  SI.Angle phi(stateSelect = stateSelect)
```

```
    "Absolute rotation angle of component";
  SI.AngularVelocity w(stateSelect = stateSelect,
    start = w_start) "Absolute angular velocity of component
    (= der(phi))";
  SI.AngularAcceleration a "Absolute angular acceleration of
     component (= der(w))";
  SI.Torque M_fric "Frictional torque, M_fric(w)";
equation
  phi = flange_a.phi;
  phi = flange_b.phi;
  w = der(phi);
  der(w) = if w < outMin and a < 0 then 0 else a;
  //
  J * a = flange_a.tau + flange_b.tau - M_fric;
end TurboShaft_Template;
```

E.1.6 Complete Turbo Shaft

```
model TurboShaft "Complete turbo shaft, friction torque
    modeled as a linear function in rotational speed"
  extends VehProLib.Interfaces.TurbochargerTemplates.
    TurboShaft_Template;
  parameter Real c_fric = 1e-6 "Friction coefficient
    [J s/rad^2]";
equation
  M_fric = c_fric * w;
end TurboShaft;
```

E.1.7 Two Pin with External Source

```
partial model TwoPinWithExternalSource
  import SI = Modelica.SIunits;
  VehProLib.Interfaces.FlowCuts.FlowCut_i i;
  VehProLib.Interfaces.FlowCuts.FlowCut_o o;
  VehProLib.Interfaces.FlowCuts.FlowCut_i ext;
  outer model Gasmedium =
    VehProLib.Interfaces.GasBase.GasPropBase;
end TwoPinWithExternalSource;
```

E.1.8 Intercooler

```
model Intercooler "Model of an intercooler"
  import SI = Modelica.SIunits;
  extends VehProLib.Interfaces.GasPins.
    TwoPinWithExternalSource (i(n = g_i.n), o(n = g_o.n),
    ext(n = g_i.n));
  parameter Real epsilon = 0.81 "Intercooler effectness";
  // Gas properties
  Gasmedium g_i(T(start = 300), p(start = 100000.0))
    "The incomming gas, which is considered hot";
  Gasmedium g_o(T(start = 300), p(start = 100000.0))
    "The outgoing gas, which is cooled by the intercooler";
  //
  SI.Temperature T_o(displayUnit = "K") "Temperature out";
protected
  SI.Temperature T_cool "Temperature for the cooling (air)
    flow";
  SI.Temperature T_i "Temperature in";
equation
  // Flow equations
  i.p = o.p;
  i.T = o.T;
  i.x = o.x;
  i.W = -o.W;
  i.Wx = -o.Wx;
  i.H = (-o.H) + i.W * (g_i.h - g_o.h);
  // Set flows from Ambiant to zero
  ext.W = 0;
  ext.Wx = ext.x * 0;
  ext.H = 0;
  // Gas properties, same upstream and downstream
  g_i.x * i.W = i.Wx;
  g_i.p = i.p;
  g_i.h * i.W = i.H;
  g_o.x = g_i.x;
  g_o.p = g_i.p;
  g_o.T = T_o;
  // Temperature calculations
  T_cool = ext.T;
  T_i = g_i.T;
  // Temperature of flowing gas is modeled as a temperature
    drop/raise.
  T_o = T_i - epsilon * (T_i - T_cool);
  // (7.69)
end Intercooler;
```

E.1.9 Wastegate

```
 model Wastegate "Complete Wastegate, Area modeles as a
    percentage of the maximum area"
  extends VehProLib.Engine.Restrictions.StandardRestriction;
  import SI = Modelica.SIunits;
  parameter SI.Area WG_Amax = 3.5e-4 "Maximum Wastegate Area";
  parameter SI.Time tau = 0.2 "Time constant for
    wategateplate movement";
  Modelica.Blocks.Interfaces.RealInput f_normalized;
  //
  Real pos(start = 0.06, fixed = true);
equation
  der(pos) = 1 / tau * (f_normalized - pos);
  A = WG_Amax * min(max(pos, 0), 1);
end Wastegate;
```

E.2 Other Components

E.2.1 Incompressible Turbulent Restriction with Linear Region

```
model IncompressibleRestriction "Incompressible Turbulent
    restriction with Linear Region - Model 7.3"
  extends VehProLib.Interfaces.GasPins.TwoPinStatic;
  import SI = Modelica.SIunits;
  parameter Real H = 1e7 "Flow resistance (C_tu / sqrt(R))";
  parameter SI.Pressure p_lin(displayUnit = "Pa") = 100
    "Limit for linear region";
  SI.Pressure delta_p;
equation
  // (7.7):
  g.T = if i.p > o.p then i.T else o.T;
  g.p = if i.p > o.p then i.p else o.p;
  g.x = if i.p > o.p then i.x else o.x;
  delta_p = abs(i.p - o.p);
  // (7.6)
  i.Wx = (if delta_p > p_lin then sqrt(delta_p)
    else delta_p / sqrt(p_lin)) *
    sign(i.p - o.p) * sqrt(g.p / (H * g.T)) * g.x;
  i.H = g.h * i.W;
  i.W = sum(i.Wx);
end IncompressibleRestriction;
```

E.2.2 Sensors

Pressure Sensor

```
model PressureSensor "Pressure Sensor"
  extends VehProLib.Interfaces.GasPins.OnePin(c(n = g.n));
  import SI = Modelica.SIunits;
  Modelica.Blocks.Interfaces.RealOutput p(unit = "Pascal")
   "Absolute pressure in Pascal as output signal";
```

```
equation
  g.p = c.p;
  g.T = c.T;
  g.x = c.x;
  p = c.p;
  c.H = 0;
  c.W = 0;
  c.Wx = 0 * g.x;
end PressureSensor;
```

Pressure Sensor

```
model TemperatureSensor "Temperature Sensor"
  extends VehProLib.Interfaces.GasPins.OnePin(c(n = g.n));
  import SI = Modelica.SIunits;
  Modelica.Blocks.Interfaces.RealOutput T(unit = "K")
    "Absolute temperature in Kelvin as output signal";
equation
  g.p = c.p;
  g.T = c.T;
  g.x = c.x;
  T = c.T;
  c.H = 0;
  c.W = 0;
  c.Wx = 0 * g.x;
end TemperatureSensor;
```

To Library Developers

Recommendations for the Development of Modelica Libraries

General

- Please avoid any Dymola specific code, such as Dymola functions or annotations. Avoiding this contributes to a more tool independent library.
- Use experiment annotations since different tools have different default settings.
- Use Modelica URLs for image paths.
- Follow the Modelica specification as much as possible.

Modelica Code Conventions

We prefer that any library to be implemented in Wolfram *SystemModeler* follow the code conventions based on the Modelica Standard Library:

1. Naming

1.1. Class and instance names are written in upper and lower case letters, e.g "ElectricCurrent". An underscore is only used at the end of a name to characterize a lower or upper index, e.g "pin_a".

1.2. Class names always start with an upper case letter.

1.3. Instance names, i.e names of component instances and of variables (with the exception of constants) usually start with a lower case letter. The only exception to this is when the instance or variable starts with an upper case letter by tradition, such as “T” for a temperature variable.

1.4. Constant names, i.e names of variables declared with the “constant” prefix, follow the usual naming conventions (= upper and lower case letters) and usually start with an upper case letter, e.g UniformGravity, SteadyState.

1.5. Two connectors of a domain that have identical declarations and different icons are usually distinguished by “_a”, “_b” or “_p”, “_n”, e.g., Flange_a/Flange_b, HeatPort_a, HeatPort_b.

1.6. The instance name of a component is always displayed in its icon (= text string “%name”). A connector class has the instance name definition in the diagram layer and not in the icon layer. Parameter values, e.g resistance, mass, gear ratio etc are displayed in the icon layer. The text should be black and the font size smaller than the instance name.

Library Structure

A main package usually has the following subpackages:

1. A UsersGuide containing an overall description of the library and how to use it.

2. Examples containing models that demonstrate the usage of the library.
3. Interfaces containing connectors and partial models.
4. Types containing type, enumeration and choice definitions.

Documentation

General

- All classes should be documented.
- All parameters and variables, including protected ones, should have a one-line comment. Comments should also be added to each class (package, model, function etc). The first character should be upper case. For one-line comments of parameters, variables and classes, no period should be used at the end of the comment.
- Spelling and grammar must be correct.
- In the HTML documentation of any Modelica library, the headings <h1>, <h2> and <h3> should not be used since they are utilized by the automatically generated documentation.
- The utilized heading format starts with <h4> and terminates with </h4>, e.g.,
  <h4>Description</h4>
  The <h4> and <h5> headings must not be terminated by a colon (:).
- For additional structuring <h5> and </h5> may be used.

Structure

The following parts should be added to the documentation of each component:

1. General information, without additional subsections, that explains how the class works
2. Syntax (for functions only). Show the syntax of a function call with minimum and full input parameters.
3. Optional:
   - 3.1. Implementation: explains how the implementation is made.
   - 3.2. Limitations: explains the limitations of the component.
   - 3.3. Notes
   - 3.4. Examples: show basic usage of the components.
   - 3.5. References.
   - 3.6. Acknowledgments
   - 3.7. Revision history: if required/intended for a package/model, the revision history should be placed in annotation(Documentation(revisions=”...”));

These sections should appear in the listed order above.

Getting Started Guide

Preferably, there should be a getting started guide available such as a video screen cast, Modelica UserGuide or a tutorial.

Availability

- It must be clear which features of the library are supported on which platforms (Mac, Windows, Linux).
- It must be clear which versions (if any) of the Modelica Standard Library are supported/needed.
- All library dependencies must be declared.

Additional software

- If additional software is needed, such as DLLs or compilers, these should be clearly stated.

Appearance

- All packages and classes should have an icon.

Other

- Where applicable, all variables must have units.
- A class that has not been fully tested or validated should be marked with *Beta* at the end of the name.
- The example models should preferably cover all main components available in the library. Each example should also have documentation describing the relevance of the example and what is illustrated in it (in addition to the more common documentation that instructs the user how long to simulate and what to plot).

Appendix

Examples of Dymola code recurrent in Modelica libraries

- Experiment annotations specifying Dymola solvers, such as Esdirk45.
- Tables such as dymTableInit and dymTableIpol
- Graphical annotations such as IMG SRC=”../Pics/SingleLever.png”, in which “../” means go to root in Dymola, but in WSM (as is expected for relative paths) it means go up one level in the directory.

Vehicle Modeling

Introtext

In todays automobile industry, efficient engines with as little environmental impact as possible are a demand. Optimal engine and vehicle compositions for meeting those demands can be made with help of simulations. The VehProLib library allows you to build multiple kinds of vehicles for propulsion. We will present below, how downsizing a 2.3 litres engine to a 1.2 liters turbocharged engine can produce similar performance for a vehicle following a drive cycle.

Example

Heading

A Realistic Engine

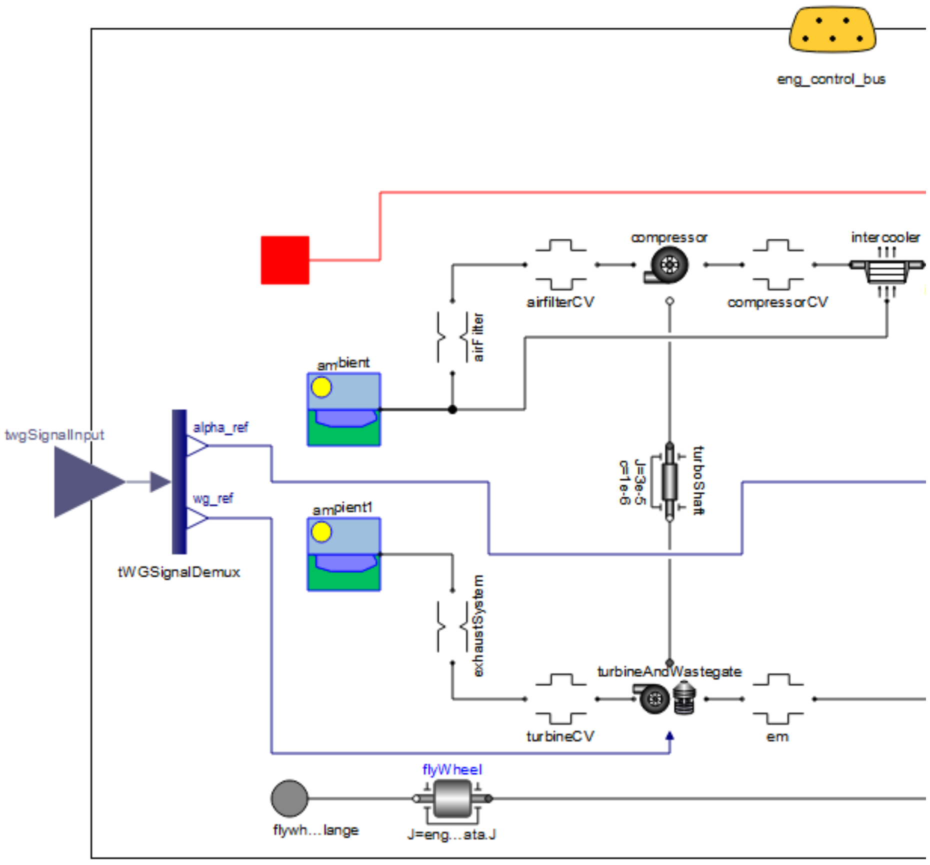
Paragraph

By taking advantage of the built-in gas interface in VehProLib, quantities like pressure, temperature and mass flow can be studied under different phases of the driving cycle. This is important for understanding the engine behaivior and how changes will influence the performance.

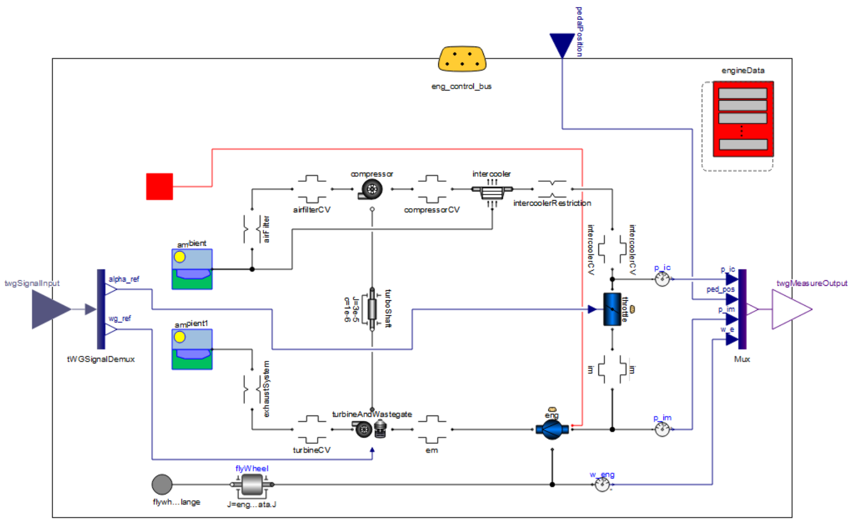
SimpleLayout

Image

FullSize



Thumbnail



Caption

The engine model seen in the diagram view in Model Center.

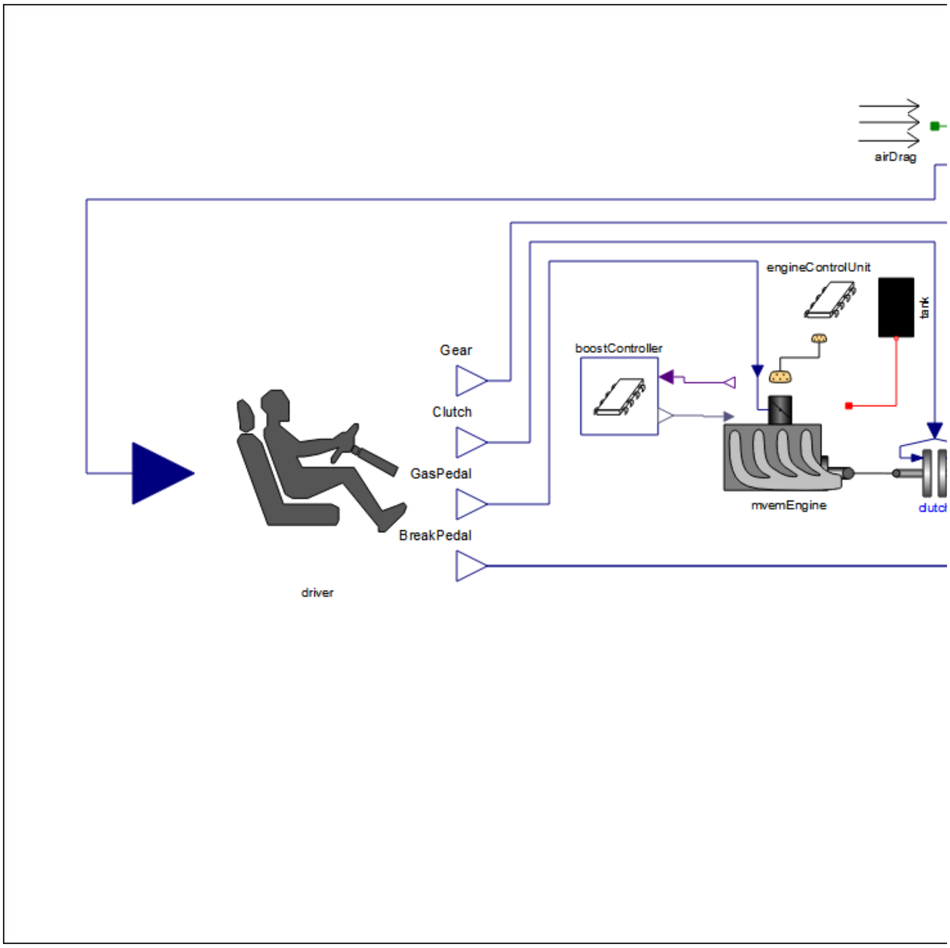
Paragraph

The engine consists of incompressible and compressible restrictions such as air filter and throttle respectively. The engine also consists of several control volumes which store energy and mass, for dynamics in the system. There are several other components included, the most important is the cylinder which generates a torque based on the work produced by the cylinder.

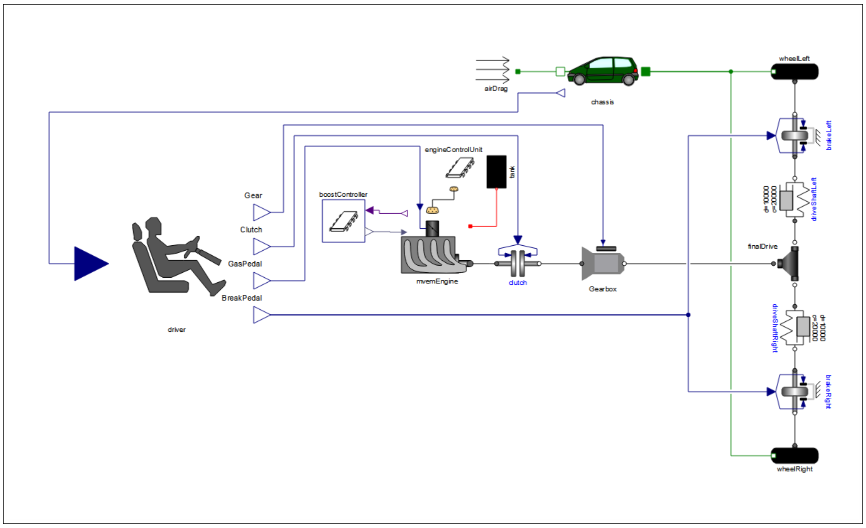
SimpleLayout

Image

FullSize



Thumbnail



Caption

The vehicle model seen in the diagram view in Model Center, where the engine from the previous figure is named *mvemEngine*.

Paragraph

The vehicle model shown in the figure above is driving on a one-dimensional road with models for air drag, rolling resistance and wheel slip. The gearbox model has five gears and an efficiency loss.

Heading

Engine Downsizing

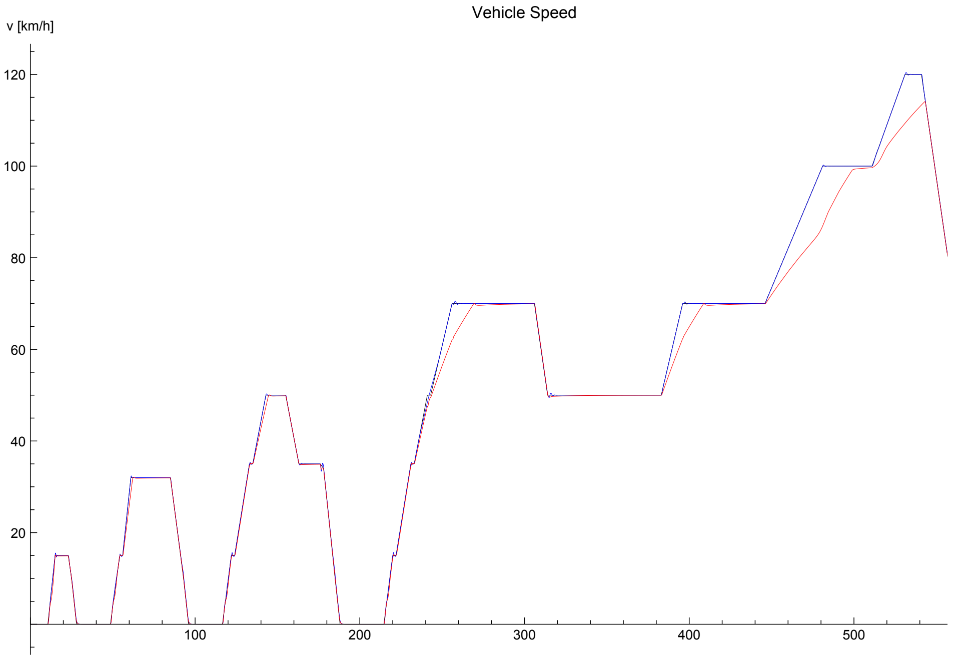
Paragraph

Engine downsizing is the usage of a smaller engine to produce similar power of a larger engine. The amount of fuel that can be efficently burned inside the cylinder chamber will decide how much work the engine can produce. A turbocharger compresses the intake air allowing a higher amount of fuel inside the chamber.

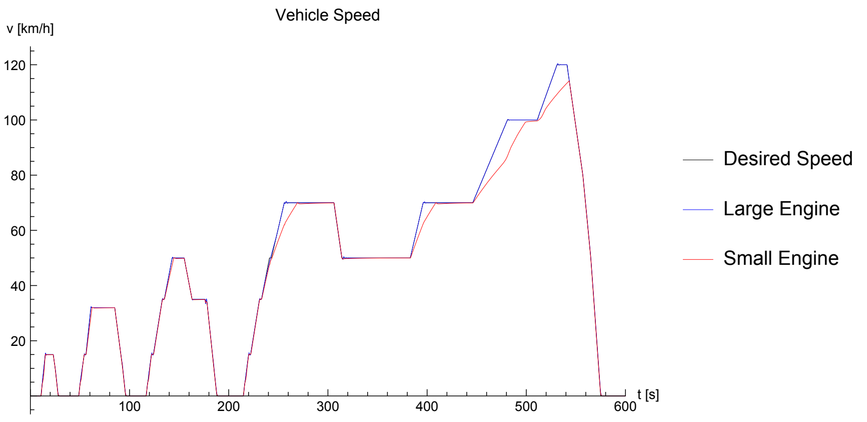
SimpleLayout

Image

FullSize



Thumbnail



Caption

In the figure above the vehicle speeds are shown for the vehicle with a larger engine and the vehicle with the smaller turbocharged engine.

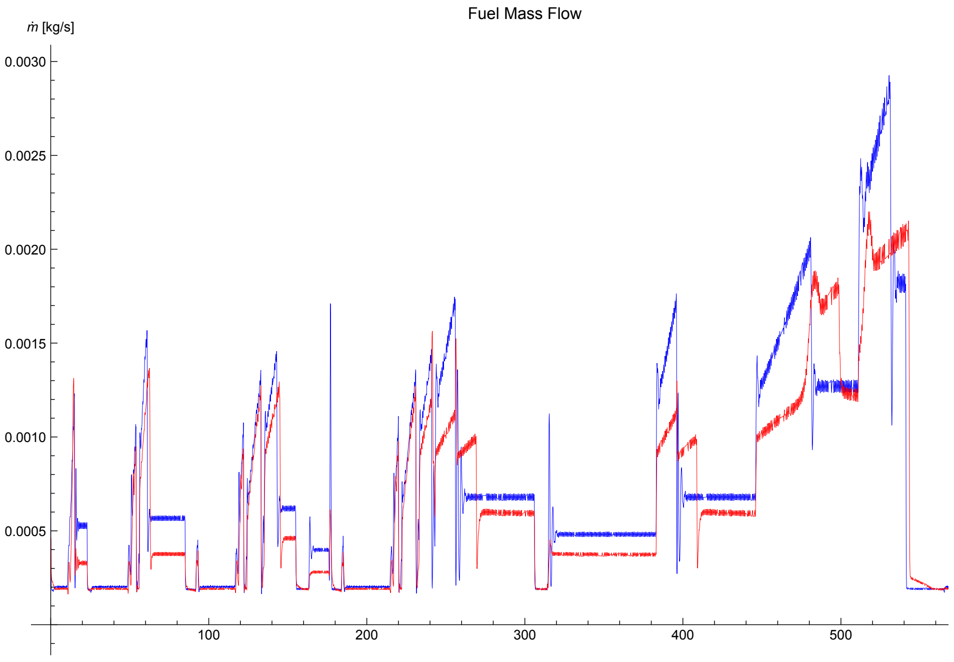
Paragraph

The fuel consumption is an important factor from both an environmental standpoint and an economical standpoint. As can be seen in previous figure the speed behaviour is similar between the two engines, the average fuel consumtion for the larger engine is 0.71 litre per 10 km compared to 0.66 litre per 10 km for the smaller engine. The fuel consumption profile can be seen in the figure below.

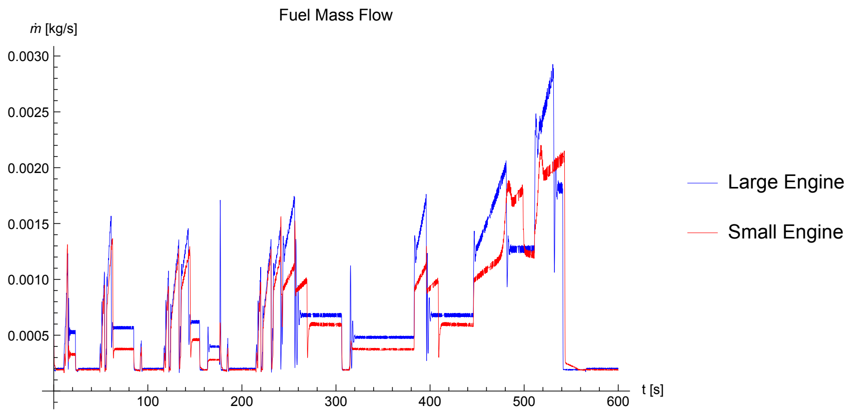
SimpleLayout

Image

FullSize



Thumbnail



Caption

The figure above shows the comparison of fuel mass flow between the two engines.

Heading

Model Calibration

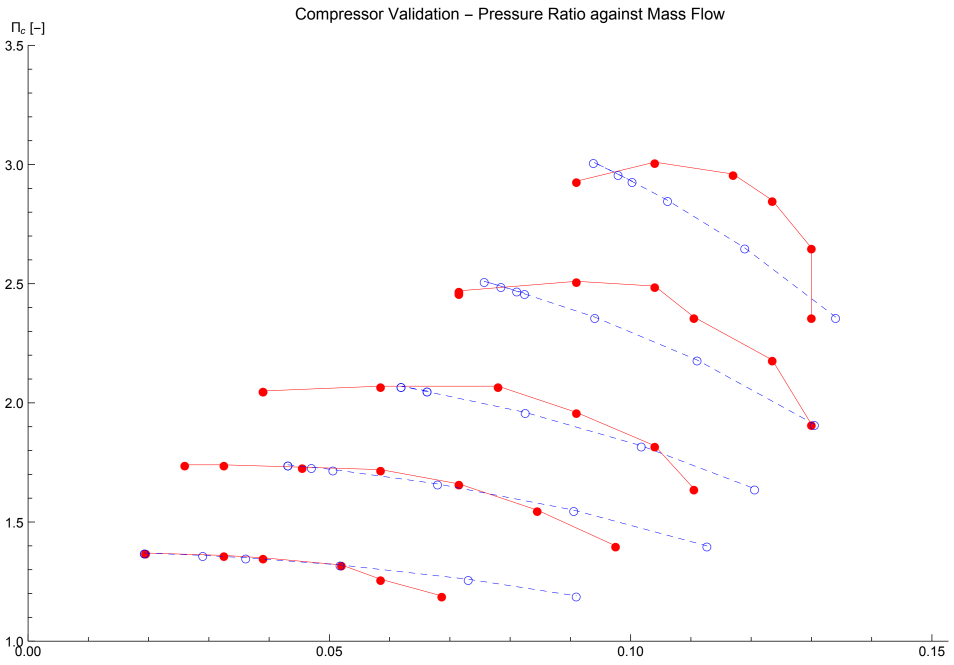
Paragraph

Most of the components in VehProLib are or are created from templates where the individual properties of the real component is specified on top of the template. The individual properties can be calibrated to suit (NMinimize) real data and be set (WSMSetValues) in *SystemModeler* using *Mathematica*.

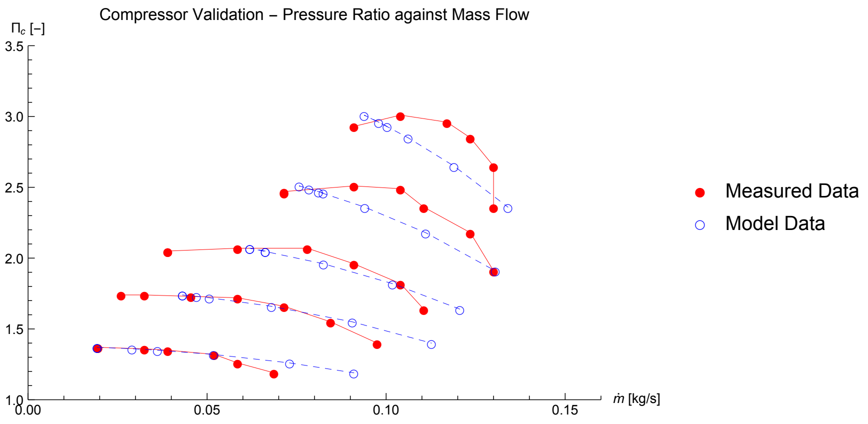
SimpleLayout

Image

FullSize



Thumbnail



Caption

Real data plotted against model data for the compressor model using *Mathematica*.

Callout

CallOutHeader

Model vehicle systems with VehProLib

CallOutCaption

The VehProLib can be used to model cars with SI-engines (Spark-Ignition) or HEV (Hybrid Electric Vehicle). The library supports different kinds of components using a template structure. With the use of *Mathematica* one can adjust the library components to fit the real components smoothly.

Related WSM Examples

- Driveline: Analyze a Drive Cycle
- Wheel with Dry Friction
- Battery: Model a Chemical Electrical System

Related Wolfram Blogs

- Announcing Wolfram SystemModeler
- Announcing Wolfram SystemModeler 4

Related WSM Features

- Multidomain Modeling
- Model Calibration
- Hierarchical modeling

Related WSM Libraries

- VehProLib
- PlanarMechanics
- Modelica Standard Library

Keywords

- Compressor
- Engine
- Engine dynamics
- Engine flow
- HEV
- Hybrid Electric Vehicle
- MVEM

- Mean Value Engine Modeling
- Parameter estimation
- Parameter optimization
- Spark Ignition
- SI-engine
- Template models
- Throttle
- Turbine
- Turbo
- Turbocharger
- Vehicle modeling
- VehProLib

Meta Description

Example of a vehicle made in VehProLib following a driving cycle.

Bibliography

Cambridge University Press 2015. Cambridge dictionaries online, June 2015. URL `http://dictionary.cambridge.org/dictionary/british/`. Cited on page 73.

Per Andersson. *Air charge estimation in turbocharged spark ignition engines*. Department of Electrical Engineering, Linköping University, 2005. Cited on page 3.

Modelica Association. Modelica homepage, March 2015. URL `http://www.modelica.org/`. Cited on pages 4 and 79.

Per Öberg. A dae formulation for multi-zone thermodynamic models and its application to cvcp engines. 2009. Cited on pages 3 and 71.

Per Öberg. Vehicular systems homepage, March 2015. URL `http://www.vehicular.isy.liu.se/`. Cited on page 2.

S Larry Dixon and Cesare Hall. *Fluid mechanics and thermodynamics of turbomachinery*. Butterworth-Heinemann, 2013. Cited on page 3.

Martin Enqvist, Torkel Glad, Svante Gunnarsson, Peter Lindskog, Lennart Ljung, Johan Löfberg, Tomas McKelvey, Anders Stenman, and Jan-Erik Strömberg. Industriell reglerteknik kurskompendium. *Automatic control, ISY Linköpings Institute of Technology*, 2010. Cited on page 43.

Lars Eriksson. Vehprolib - vehicle propulsion library. library development. In *3rd International Modelica Conference*, pages 249–256, 2003. Cited on pages 2 and 5.

Lars Eriksson and Lars Nielsen. *Modeling and Control of Engines and Drivelines*. John Wiley and Sons, 2014. Cited on pages 3, 5, 7, 11, 12, 14, 15, 16, 17, 18, 19, 21, 23, 70, and 73.

John B Heywood. *Internal Combustion Engine Fundamentals*, volume 930. Mcgraw-hill New York, 1988. Cited on page 3.

Wolfram MathCore. Hydraulic documentation, June 2015a. URL `http://reference.wolfram.com/system-modeler/libraries/Hydraulic/Hydraulic.html`. Cited on pages 6 and 81.

Wolfram MathCore. Modelplug documentation, June 2015b. URL `http://reference.wolfram.com/system-modeler/libraries/ModelPlug/ModelPlug.html`. Cited on pages 7 and 82.

MathWorks. Mathworks homepage, May 2015a. URL `http://se.mathworks.com/`. Cited on page 4.

MathWorks. Simulink homepage, May 2015b. URL `http://se.mathworks.com/products/simulink/`. Cited on page 4.

Günter P Merker, Christian Schwarz, Gunnar Stiesch, and Frank Otto. *Simulation combustion, simulation of combustion and pollution formation for engine-development*. Germany, Spinger-Verlag Berlin Heiddelberg, 2006. Cited on page 3.

Modelica® - A Unified Object-Oriented Language for Systems Modeling. Modelica Association, 3.3 revision 1 edition, July 2014. Language Specification. Cited on pages 19, 26, and 28.

Otto Montell. Advanced concepts in modelica and their implementation in vehprolib. Master's thesis, Linköping University, 2004. LiTH-ISY-EX-3511. Cited on pages 5, 25, and 34.

Carl Nordling and Jonny Österman. *Physics handbook*. Studentlitteratur, fifth edition edition, 1980,1997. Cited on page 36.

OpenModelica. Openmodelica homepage, March 2015. URL `https://openmodelica.org/`. Cited on page 5.

Fredrik Pettersson. Simulation of a turbo charged spark ignited engine. Master's thesis, Linköping University, 2000. LiTH-ISY-EX-3010. Cited on page 5.

Wolfram Research. Wolfram systemmodeler industry examples, March 2015a. URL `http://www.wolfram.com/system-modeler/industry-examples/`. Cited on page 23.

Wolfram Research. Mathematica homepage, May 2015b. URL `http://www.wolfram.com/mathematica/`. Cited on page 4.

Wolfram Research. Wolfram systemmodeler homepage, March 2015c. URL `http://www.wolfram.com/system-modeler/`. Cited on page 4.

P Spring, CH Onder, and L Guzzella. Egr control of pressure-wave supercharged ic engines. *Control Engineering Practice*, 15(12):1520–1532, 2007. Cited on page 3.

Vehicular Systems. Project compendium modelling and control of engines and drivelines. Project Compendium TSFS09, 2014. URL `http://www.vehicular.isy.liu.se/Edu/Courses/TSFS09/ProjektPM/ProjektPM.pdf`. Cited on pages 15 and 20.

Dassault Systèmes. Dassault systèmes homepage - dymola, June 2015. URL `http://www.3ds.com/products-services/catia/products/dymola/`. Cited on page 5.

Kiencke Uwe and L Nielsen. *Automotive control system*. Springer Berlin, 2005. Cited on page 3.