Ben Smulter

Building a GT-Power-in-the-Loop System comprising a Fast-Running Model as an Extended Digital Twin for a Single-Cylinder Research Engine

School of Technology and Innovations
Degree Programme in Energy and Information Technology
Automation and Computer Science

Vaasa 7.5.2024

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Ben Smulter
Title of the Thesis: Building a GT-Power-in-the-Loop System comprising a Fast-Running Model as an Extended Digital Twin for a Single-Cylinder Research Engine
Degree: Energy and Information Technology
Programme: Automation and Computer Science
Supervisor: Mohammed Elmusrati
Year: 2024 **Number of Pages:** 96

ABSTRACT:

Due to the distinct characteristics of single-cylinder engines (SCE) regarding flow dynamics, platform structure and used auxiliaries, conventional multi-cylinder engine (MCE) turbochargers cannot be installed for utilization of the exhaust gas kinetic energy to compress the intake air. The purpose of this thesis was to compensate for this lack of a turbocharger and the disadvantages of the currently used Matlab turbocharger model, by creating an extended digital twin in the form of a virtual turbocharged MCE, whose calculated exhaust pressure can be sent to and utilized by a research SCE through real-time co-simulation. In this thesis, this extended digital twin went by the name GT-Power-in-the-Loop (GTPiL).

A detailed, crank-angle resolved, one-dimensional model of a medium-speed and -bore engine is reduced to a fast-running model (FRM), used as a black box in a via Matlab initialized Simulink model. This model is communicating with the SCE's control system using the Modbus protocol during the measurement of an engine test point. The graphical user interface of the Simulink model also provides the operator with manual control of the FRM input if desired, and real-time monitoring of the FRM output. When the test point measurement is done the GTPiL system triggers a so-called "Start-Stop" (SS) simulation, that is an offline simulation initialized by an average of the measurement point's last 300 cycles for the FRM input parameters and run until steady-state, with the purpose of saving the results of the FRM. An Excel macro is also triggered at this stage, to format and visually present the FRM outcome in an Excel file, saved together with the other GTPiL system related output files in a test point specific folder.

A comparison of the FRM with the detailed model shows negligible differences in performance after the FRM conversion. A validation using measured engine data however reveals that the FRM needs a minor recalibration to better match engine air flow and especially the mean firing pressure during combustion. The calculated exhaust pressure is still within a 5% error margin of the measured engine output.

To reproduce the exhaust system dynamics of an MCE when running a research SCE without a turbocharger, these flow dynamics must be modelled somehow. By using the GTPiL system instead of the currently used Matlab turbocharger model relying on injected water for its calculations, the operator receives a more robust way of modelling the exhaust system phenomena, and the flexibility of e.g. virtually testing different turbochargers and engines with various cylinder amount.

KEYWORDS: Digital twin, Fast-running model, GT-Power-in-the-Loop, Matlab, Simulink, Single-cylinder engine, Turbocharging

VASA UNIVERSITET
Skolan för Teknologi och Innovationer
Författare: Ben Smulter
Avhandlingens titel: Skapande av ett GT-Power-in-the-Loop System bestående av en Snabb Simuleringsmodell som en Utvidgad Digital Tvilling för en Singel-Cylinder Forskningsmotor
Examen: Energi- och Informationsteknologi
Program: Automations- och Datateknik
Handledare: Mohammed Elmusrati
År: 2024 **Sidantal:** 96

ABSTRAKT:

På grund av de distinkta karaktärsdragen hos singel-cylinder motorer (SCE) gällande flödesdynamik, plattformsstruktur och använda supportsystem, kan en multi-cylinder motors (MCE) konventionella turboladdare inte installeras för tillvaratagande av avgasens kinetiska energi för att komprimera insugsluften. Syftet med denna avhandling var att kompensera för denna brist på turboladdare och nackdelarna med den för tillfället använda Matlab turboladdarmodellen, genom att skapa en utvidgad digital tvilling i form av en virtuell turboladdad MCE, vars beräknade avgastryck kan skickas till och användas av en forsknings-SCE genom parallell simulering i realtid. I denna avhandling gick denna utvidgade digitala tvilling under namnet GT-Power-in-the-Loop (GTPiL).

En detaljerad, vevaxelupplöst en-dimensionell modell av en mediumvarvig och -stor motor reduceras till en snabb simuleringsmodell (FRM), använd som en svart box i en via Matlab initialiserad Simulinkmodell, som kommunicerar med SCE kontrollsystemet genom att använda Modbusprotokollet. Det grafiska användargränssnittet i Simulinkmodellen ger också operatören manuell kontroll över input till FRM om så önskas, och realtidsmonitorering av dess resultat. När mätningen av en testpunkt har gjorts triggar GTPiL systemet en såkallad ”Start-Stop” simulering, vilket är en offline simulering initialiserad av ett medeltal av mätpunktens sista 300 cyklar för FRMs input parametrar och som körs tills den nått ett stabilt läge, med syfte att spara FRMs resultat. En Excel makro triggas också i detta skede, för att formatera och visuellt presentera FRMs resultat i en Excel fil, som sparas tillsammans med andra resultatfiler genererade av GTPiL systemet i en testpunktsrelaterad mapp.

En jämförelse mellan FRM och den detaljerade modellen visar försumbara skillnader i prestanda efter FRM konverteringen. En validering med riktig experimentdata visar dock att FRM är i behov av en småskalig omkalibrering för att bättre matcha luftflödet genom motorn samt speciellt det genomsnittliga maximala cylindertrycket under förbränning. Det beräknade avgastrycket hålls ändå inom en felmarginal på 5% jämfört med detta uppmätta experimentdata.

För att avbilda dynamiken i avgassystemet hos en MCE när man kör en forsknings-SCE utan turboladdare, så måste denna flödesdynamik modelleras på något sätt. Genom att använda sig av GTPiL systemet istället för den för tillfället använda Matlab turboladdarmodellen som förlitar sig på injicerat vatten i sin kalkyl, kan operatören få ett mera pålitligt sätt för att modellera fenomenen i avgassystemet, och flexibiliteten för att exempelvis virtuellt testa olika turboladdare och motorer med varierande cylinderantal.

NYCKELORD: Digital tvilling, snabb simuleringsmodell, GT-Power-in-the-Loop, Matlab, Simulink, Singel-cylinder motor, Turboladdning

Contents

Figures

Abbreviations

°CA	*crank angle degrees*
aBDC	*after bottom dead center*
aTDC	*after top dead center*
ADC	*analog-to-digital converter*
BMEP	*brake mean effective pressure*
BR	*burn rate*
BSEC	*brake specific energy consumption*
BSFC	*brake specific fuel consumption*
CA5	*crank angle at which 5% of the heat has been released*
DAQ	*data acquisition*
DF	*dual fuel*
DOS	*disk operating system*
DSL	*diesel*
ECU	*engine control unit*

EMS	*engine management system*
EWG	*exhaust waste gate*
FAR	*fuel-to-air ratio*
FRM	*fast-running model*
GT	*gamma technologies*
GTPiL	*gt-power-in-the-loop*
GUI	*graphical user interface*
HiL	*hardware-in-the-loop*
HP	*high pressure*
HRR	*heat release rate*
IMEP	*indicated mean effective pressure*
IV	*intake valve*
IVC	*intake valve closing*
LP	*low pressure*
MCE	*multi-cylinder engine*
PI	*proportional-integral*
PID	*proportional-integral-derivative*
RPM	*rotations per minute*
RT	*real-time*
RT&D	*research, technology and development*
SCE	*single-cylinder engine*
SOC	*start of combustion*
SS	*start-stop*
TC	*turbocharger*
TDC	*top dead center*
TDCf	*top dead center firing*
TS	*two-stage*
W25	*Wärtsilä 25*

Symbols

λ	*lambda*	*[-]*
η_{TC}	*turbocharger efficiency*	*[-]*
p_3	*intake boost pressure*	*[bar]*
p_5	*exhaust back pressure*	*[bar]*
T_3	*intake boost temperature*	*[°C]*
T_5	*exhaust back temperature*	*[°C]*

1 Introduction

1.1 Background

In the design of future powertrains there are several conflicting requirements to pursue simultaneously due to ever more stringent emissions regulations, like minimizing exhaust emissions, increasing fuel efficiency, and increasing engine power output. To aid in this complex design, Hardware-in-the-Loop (HiL) simulation environment has emerged as a very valuable tool, in which various control strategies can be tested. By using HiL systems, realistic tests can be performed repeatedly in a cost-effective and efficient way. A HiL system usually comprises a virtual engine, i.e. an engine simulation model representing the real engine, and some physical components like e.g. an engine management system (EMS) and other physical elements such as sensors and actuators. The physical effects from the actuator signals, in this example case generated by the EMS, are imposed onto the virtual engine, and the output from these engine simulations are then fed back to the EMS in real-time (Andric, Schimmel, Sediako, Sjoblom, & Faghani, 2018).

Capturing complex non-linear behaviour of powertrain systems can be difficult, but well enough calibrated models in the HiL system can achieve this. These models need to be reliable and accurate, while being compliant with real-time constraints to be capable of capturing transient engine dynamics. This is another conflicting requirement; achieving adequate model fidelity while using large enough simulation time steps for being real-time capable or faster. With present limitations of hardware performance and computational power, engine system-level simulations are usually constituted of transient 1D crank-angle resolved models, to analyze an engine's performance in detail. However, these models are usually slow and can have a real-time factor of 20-30, and hence need to be simplified to meet the real-time constraints of a HiL system (Andric, Schimmel, Sediako, Sjoblom, & Faghani, 2018). Gamma Technologies (GT) develops and licenses the computer-aided engineering system simulation software GT-Suite, which includes a com-

plete library of physics-based modelling templates covering i.a. fluid flow, thermal, mechanical and controls (Gamma Technologies, 2022). By using the Fast Running Model (FRM) tool in the GT-Power library, a detailed and slow engine model can be simplified, balancing the calculation speed and accuracy trade-off, to meet the real-time requirements for partaking in a HiL system.

1.2 Research gap, problem and objectives

Multi-cylinder engines (MCE) at Wärtsilä all have some kind of a turbocharger (TC), but the research single-cylinder engines (SCE) do not have a conventional TC system, meaning that the boundary conditions in the form of signals like intake boost pressure (p_3), intake boost temperature (T_3), exhaust back pressure (p_5) and exhaust back temperature (T_5), need to be set and adjusted in the SCE to correspond to realistic values from an MCE for every case tested. **Figure 1** illustrates the standardized signal labelling used at Wärtsilä for an MCE, where also the previously mentioned signals can be seen. When running the SCE to test different cases however, only target parameters on the intake side are set by the operator, while the exhaust conditions are currently being calculated using a Matlab turbo model that is part of the SCE control system. More information about the SCE and the charge air and exhaust gas systems can be found in chapter 2.1 Wärtsilä small bore single-cylinder engine. This turbo model has however not always been very accurate, especially low load conditions has been causing problems. The reason for this is that the model utilizes injected water in its calculations, used for cooling of the exhaust gas, and at low load water cooling might not even be needed and water is hence not injected, causing inaccuracies in the calculations. It has also been difficult to control the amount and temperature of the water accurately enough, with nozzle leakage and injector failure sometimes occurring as well. By instead utilizing a system where a virtual turbocharged MCE GT-Power model runs in parallel with the SCE, the p_5 calculated in this MCE model could be fed back to the SCE control system, which could improve the testing accuracy and reliability. The virtual MCE in this thesis is represented by a W8L25TS-DF engine, i.e. a dual-fuel (DF) Wärtsilä 25 (W25) engine with eight cylinders and a 2-stage

(TS) turbo system with a low pressure (LP) and a high pressure (HP) turbo, which has been converted to an FRM, while the research engine is a W25 SCE. To ensure that the FRM is indeed real-time capable during runtime, Gamma Technologies offers a solution in the form of an advanced license called *GT-Power-xRT* that uses the same governing equations as GT-Power, but that provides with an unparalleled execution speed. This xRT-license was however not something Wärtsilä had at its disposal at the time of creating the FRM, meaning that the model had to be simplified enough to comply with the real-time demands using the normal GT-Power license only. During the work of the project however one GT-Power-xRT license was acquired from Gamma Technologies, to be used by this extended digital twin system.

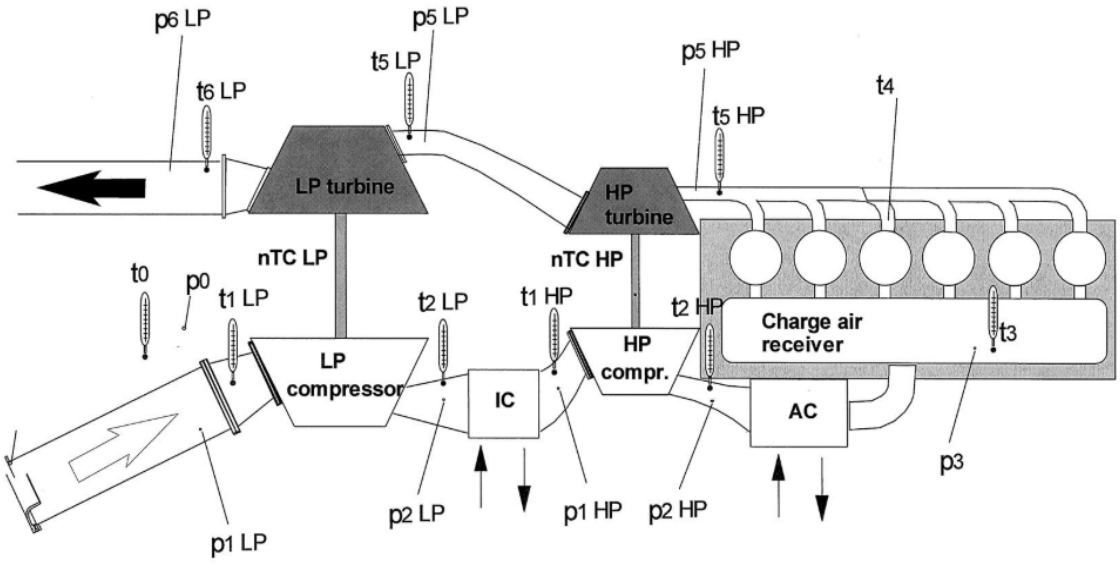

Figure 1. Standardized signal labelling used at Wärtsilä.

This thesis is the result of a project in the Research, Technology and Development (RT&D) department at Wärtsilä, and the research problem can be formulated as "*How to accurately model realistic, multi-cylinder engine exhaust system pressure, that can be used by the research engine's control system during each measurement instance?"*. The aim for solving this research problem is to develop an "extended digital twin" of a research SCE in the form of a virtual turbocharged MCE GT-Power model, where necessary parameter values are imposed from the engine onto the model that in return sends the p_5

calculated by the TC model back to the engine's control system, to be used by the operator for adjusting and fine tuning of the engine's back pressure. The objectives of the thesis are:

1. Build a real-time capable GT-Power FRM from an already existing, detailed GT-Power model representing a W8L25TS-DF engine, with minor deviation in model accuracy compared to the detailed model.
2. Create a system based on Matlab/Simulink that contains the GT-Power FRM as a black box component, working as a platform for the communication between the physical SCE and the virtual MCE.

This extended digital twin system is illustrated in **Figure 2**, which clarifies that also other signals can be sent to the physical SCE from the virtual MCE, like i.a. friction and in-cylinder conditions. The 1st objective is hence to build the MCE model on the right in **Figure 2**, while the 2nd objective is to build a communication platform for realizing the signal arrows between the physical and the virtual engine.

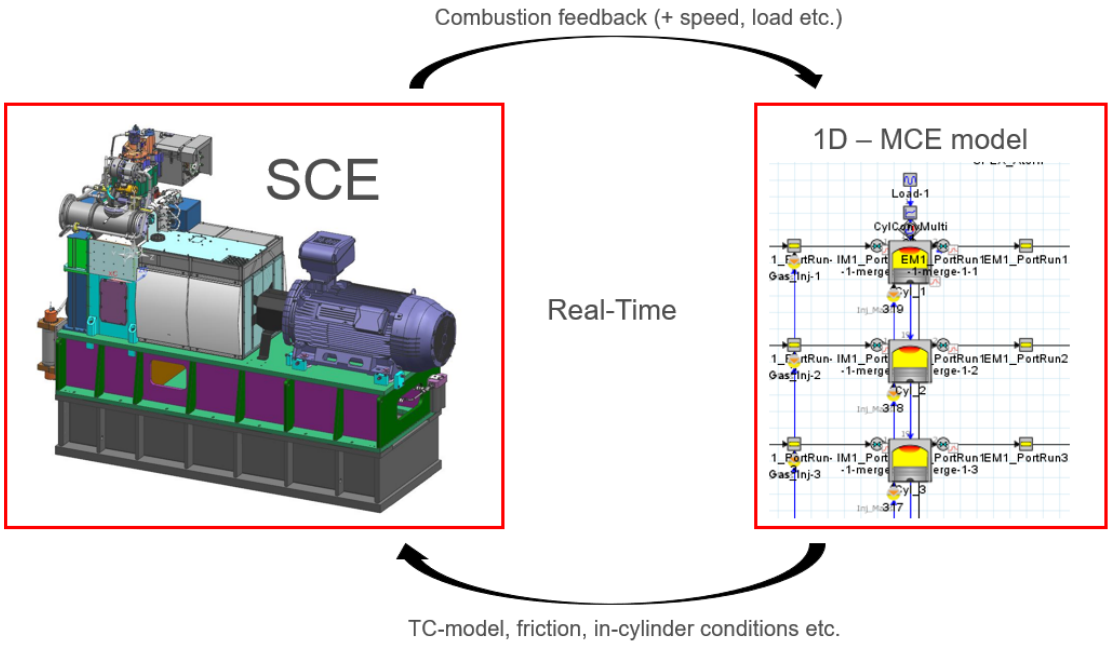

Figure 2. Extended digital twin system overview.

1.3 Definitions and limitations

As mentioned in the Background chapter, when working with powertrains, it is not unusual to develop a HiL system composed of a virtual engine and some physical components, like an EMS. In this system however, the physical component is a whole research SCE at the Wärtsilä Vaskiluoto Engine Laboratory (VEL), while the virtual engine is an MCE model of a W8L25TS-DF engine. The physical SCE with auxiliary systems, as well as a detailed GT-Power model, were already existing when starting this work. This thesis instead focuses on the software side by first building an FRM of the detailed model, and then creating a system in Matlab/Simulink for communicating with the research engine. The word *program* has also been used throughout the thesis, to describe this system consisting of Matlab code, a Simulink model, a GT-Power model, and some external macros developed in Excel VBA. Since in this project the SCE is an already functioning unit with its own control system, and the virtual MCE has got a supportive role by calculating the p_5 value at every time step and returning it to the SCE, it is debatable whether this system/program should be defined as Hardware-in-the-Loop (HiL), or if it is indeed more of a Model-in-the-Loop (MiL) system. A new term, “GT-Power-in-the-Loop” (GTPiL), was coined during the project; this term alongside “program” will be the primary names referring to the developed concept in this thesis.

To have the FRM run in parallel with the SCE, some of the measured parameter signal values need to be sent from the SCE and imposed onto the FRM during every time step, so that the FRM behaves in a similar manner to the SCE. Some of these parameters are i.a. engine speed, engine load, ambient temperature and pressure, valve timings etc. Similarly, the back pressure p_5 calculated on the virtual MCE side should be sent back to the SCE. These parameters are sent using the Modbus communication protocol. To also make the combustion similar to what occurs in the SCE, the combustion behaviour should be imposed by either sending a burn rate (BR) or a heat release rate (HRR) vector from the SCE to the FRM during every engine cycle. This was however not in the scope of this thesis, and instead measured tabulated HRRs already embedded into the FRM were used. The FRM then interpolates between these HRRs depending on the engine

load and fuel mode, i.e. if the engine runs in DF, diesel or pure diesel mode. The difference between *diesel* and *pure diesel* in this system is that when running *diesel*, the same exact engine settings as when running DF are used, while when running the model in *pure diesel* mode the compression ratio is different, to better simulate running a normal diesel engine instead of a dual fuel engine, while also the tabulated HRRs are slightly different.

Even though accuracy in results are important, the major focus of this thesis is still on the platform aspect of the system, i.e. the ability to control the simulation via Matlab/Simulink and the communication with the actual engine. When looking at some of the works in the Literature review chapter, it can be noted the effort taken to create as accurate models as possible. In this thesis however, since the primary focus was on building of the platform itself, not much time and effort was put into calibrating the FRM after it had been built. Instead only some very minor calibration was done during most steps of the FRM creation process, after which the FRM was compared with the detailed model and validated against measured data. A recalibration of the FRM is hence a necessary task for the future.

It should also be mentioned that while developing this GTPiL system, many versions were created, each one building on the previous version. This was done to follow a good coding practice, since previous versions could be referenced in case bugs would be accidentally introduced somewhere in the new version. Since the writing process of this thesis has been ongoing simultaneously as these new versions have been developed and taken into use, some text and pictures of the GTPiL system layout might be referring to an older version than what is currently existing. This will however not affect the reader's understanding of the GTPiL system, since the overall working principle is the same no matter the version.

1.4 Structure of the thesis

The Literature review chapter starts by presenting more details about what an SCE is, and how the current Matlab TC model works. It also clarifies the implication of the word *real-time*. Other work in the realm of "in-the-Loop" systems, where GT-Power and Matlab/Simulink has been used, are also presented.

The chapter Software involved in GT-Power-in-the-Loop presents the different software constituting the system. The FRM is created in GT-Power. Functions and scripts are written in Matlab, forming a runnable program that binds all the involved software together. The program opens and starts a Simulink model, which has got the FRM embedded as a black box. The Simulink model communicates with the SCE over Modbus. Dewesoft, Excel and VBA are required for creating the needed simulation output.

In the GT-Power-in-the-Loop chapter the GTPiL system itself is presented, starting by an illustration of the program working principle and structure. A walk-through of the steps taken when running the model is also shown. The FRM was also validated against experimental data, and the results are analyzed. Some of the challenges with GT-Power simulations are also discussed.

2 Literature review

2.1 Wärtsilä small bore single-cylinder engine

There are two SCE test platforms at Wärtsilä VEL: A small bore SCE with a nominal power of 200 kW and a bore ranging from 130-250 mm, and a medium bore SCE with a nominal power of 800 kW and a bore ranging from 260-400 mm (Strang, 2018). The work presented in this thesis has been developed for the small bore SCE, but can in the future also be installed on the medium bore SCE. The small bore SCE can run in Diesel mode with LFO (Light Fuel Oil) or HFO (Heavy Fuel Oil), but also in DF (Dual Fuel) Gas mode using LNG (Liquefied Natural Gas) as its main fuel (Strang, 2018).

By utilizing SCEs, more advanced and detailed research can be performed compared to an MCE. The flexible SCE test platform allows for faster and easier investigation of e.g. bore and stroke, to gain an understanding of how basic internal combustion engine parameters affect i.a. efficiency, combustion, friction, emissions etc. As its name reveals, an SCE has only got one set of a cylinder, piston and cylinder head. A large flywheel is needed for balancing of the rotational speed variations, while mass balancing shafts in the engine compensate for the first and second order inertia forces. Besides the engine itself, the SCE testing concept also includes a generator that is controlled by a frequency converter and coupled to the engine via a shaft, forging a combination called a *generating set* that is shown in **Figure 3**. The mechanical power from the SCE can run the generator that supplies electrical power to the grid, but the generator can also act as an electrical motor to rotate the SCE at a desired operating speed (Strang, 2018).

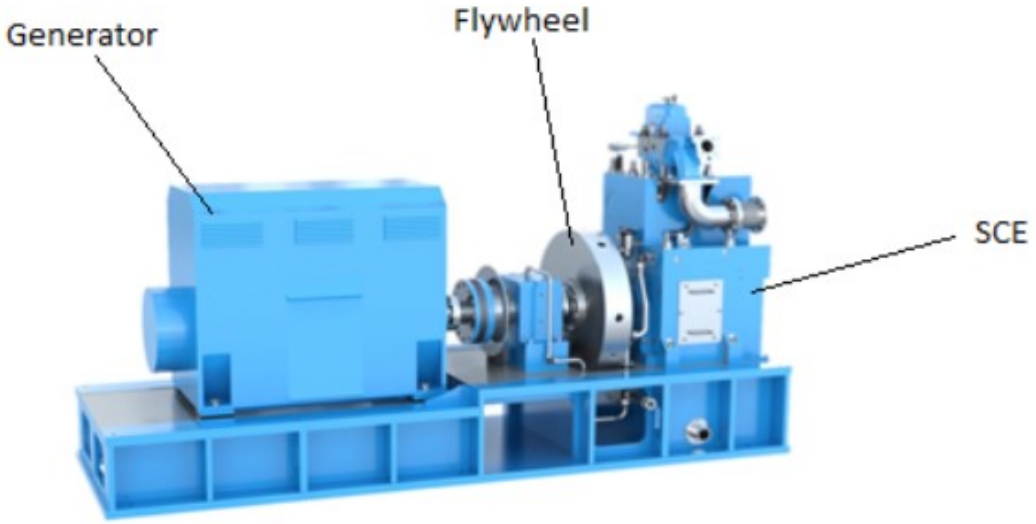

Figure 3. SCE generating set (Strang, 2018).

The need for the GT-Power-in-the-loop system in this thesis can be recognized after investigating the auxiliaries of an SCE which are very different from the auxiliaries of an MCE, especially considering the charge air and exhaust gas systems due to the lack of a conventional TC for the SCE. In the charge air system the TC compressor is replaced by a large-scale compressor and several bulky buffer tanks, to even out the flow pulsations, that pressurizes the intake air before it enters the engine (Strang, 2018). Likewise in the exhaust gas system there are lots of buffer tanks, but also controllable back pressure valves (Strang, 2018), using which the operator can select the desired back pressure against which the piston will be pushing out the exhaust gases during the exhaust stroke.

2.2 Matlab turbocharger model

A flowchart representing the current turbocharger model for the SCE is shown in **Figure 4**. The operator defines either lambda (λ) or p_3 (depending on if running in diesel or gas mode), turbocharger efficiency (η_{TC}) and if the exhaust waste gate (EWG) is open or closed. Lambda is the ratio of actual air-fuel ratio to stoichiometric air-fuel ratio for a given air-fuel mixture. Engine constants are also predefined in the model, and are used

for tuning the model. The brake specific fuel consumption (BSFC), brake specific energy consumption (BSEC) and several other parameters are measured from the SCE during the engine run. The temperature in the exhaust manifold before the location of the imaginary HP turbine, T_5, is needed for calculating the p_5. To calculate T_5, i.a. the flow and temperature of the injected cooling water is needed, which are among the measured parameters. All these parameters are fed into the model, whereby p_3 (if lambda was defined) and p_5 are calculated (Palmkvist, 2011).

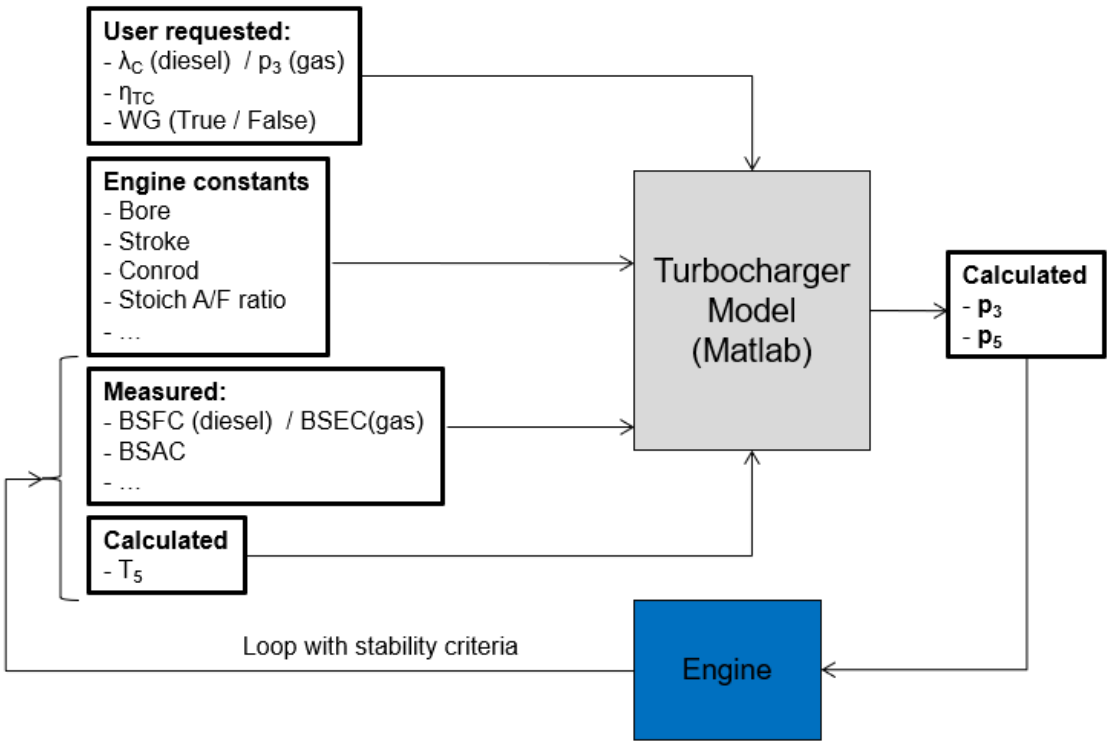

Figure 4. Turbocharger model in Matlab (Palmkvist, 2011).

As mentioned in chapter 1.2, one of the TC model's drawbacks is its need for injected water into the exhaust for the calculations in the model to be accurate. At low load, water injection might not even be needed, and the water injection itself is difficult to control with enough precision. BSFC is another needed parameter for the calculations, which is continuously measured in gas mode, but that has got a delay of three minutes before it can be measured in diesel mode. Another issue is the inaccuracies of calculating

the TC efficiency, and especially the T5 in the SCE, due to the lesser amount of hot exhaust gas pulses compared to on an MCE with more cylinders, and hence a constant must be added to the SCE's calculated T5 to correct for this. **Figure 5** shows the relative difference between the TC efficiency and the T5 that have been calculated for the SCE, and the same parameters calculated and measured for the MCE. The figure shows large relative differences at lower loads, but better values at higher loads, meaning that the TC model accuracy is also better in the high load range. It should also be pointed out that the Matlab TC model only calculates a couple of quantities in the exhaust system, while the GTPiL system calculates quantities from a whole multi-cylinder engine during the simulation (Palmkvist, 2011).

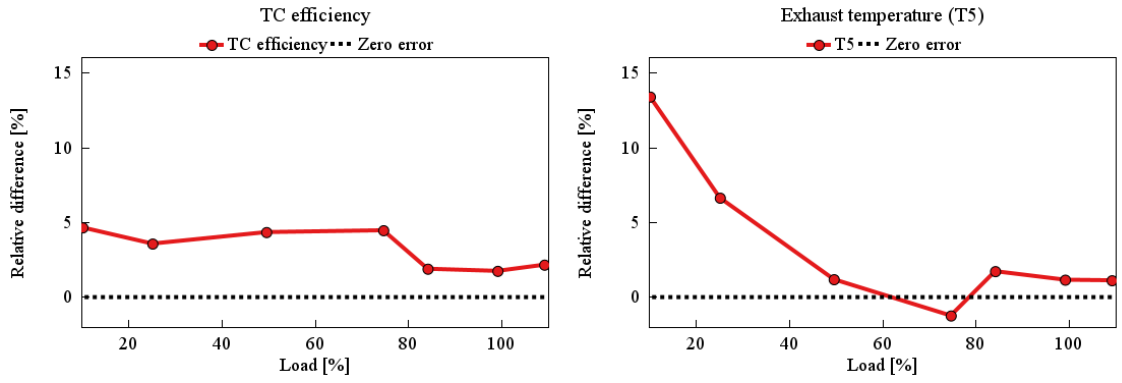

Figure 5. Relative difference between SCE and MCE regarding TC efficiency and T5.

2.3 Real-time simulation

Simulators has played a critical role in the successful development of many applications during several decades already. Because of the rapid evolution of computer technologies where they have decreased in cost and increased in performance, simulation tools have become able to solve very complex problems in a shorter time, even in real-time. For the types of digital simulation discussed in the paper written by Bélanger et al. (2008), along with the simulations presented in this thesis, a discrete-time and constant step duration is performed. When using discrete-time, time moves forward in steps of equal duration, commonly known as *fixed time step simulation*. Other solving techniques which use

variable time steps also exist, these are however not suitable for real-time simulations (Bélanger, Venne, & Paquin, 2008).

When solving mathematical functions and equations at a certain time step, every variable or system state is solved successively as a function of the ones at the end of the previous time step. During discrete-time simulations, the amount of real-time that is required to calculate all equations and functions of a system during a given time step may be shorter or longer than the duration of the simulated time step. **Figure 6** shows a comparison between the *Computation* (simulation) time step and the *Sim. Clock* (real/CPU) time step. In **Figure 6** a), the simulation time is shorter than a fixed time step (also referred to as accelerated simulation) and hence faster than real-time, while in b) the simulation time is slower than real-time. According to the authors Bélanger et al. (2008), these two simulations are referred to as o*ffline simulation*, since the moment at which a result becomes available is irrelevant. The objective when performing offline simulations is to obtain results as fast as possible. Conversely, during real-time simulations (**Figure 6** c)), the accuracy of the calculations not only depends upon the correct dynamic representation of the system, but also on the time it takes to create results. The real-time simulator must accurately produce the system calculation results in the same time as a physical counterpart (the wall-clock) would to be valid. In practice though, the real-time simulator has to be a little faster than this, since it also needs time for other operations like driving inputs and outputs to and from externally connected devices. For a given time step in a real-time simulator any idle-time preceding or following simulator operations is lost, as opposed to in accelerated simulations where idle-time is used for starting the computations meant for the next time step, to maximize its speed. The real-time simulator instead waits until the wall-clock ticks to the next time step, before proceeding with the calculations intended for it. If indeed the real-time simulator would not be able to perform all operations within the required fixed time step, the real-time simulation is considered erroneous. This phenomena is known as an *overrun* (Bélanger, Venne, & Paquin, 2008).

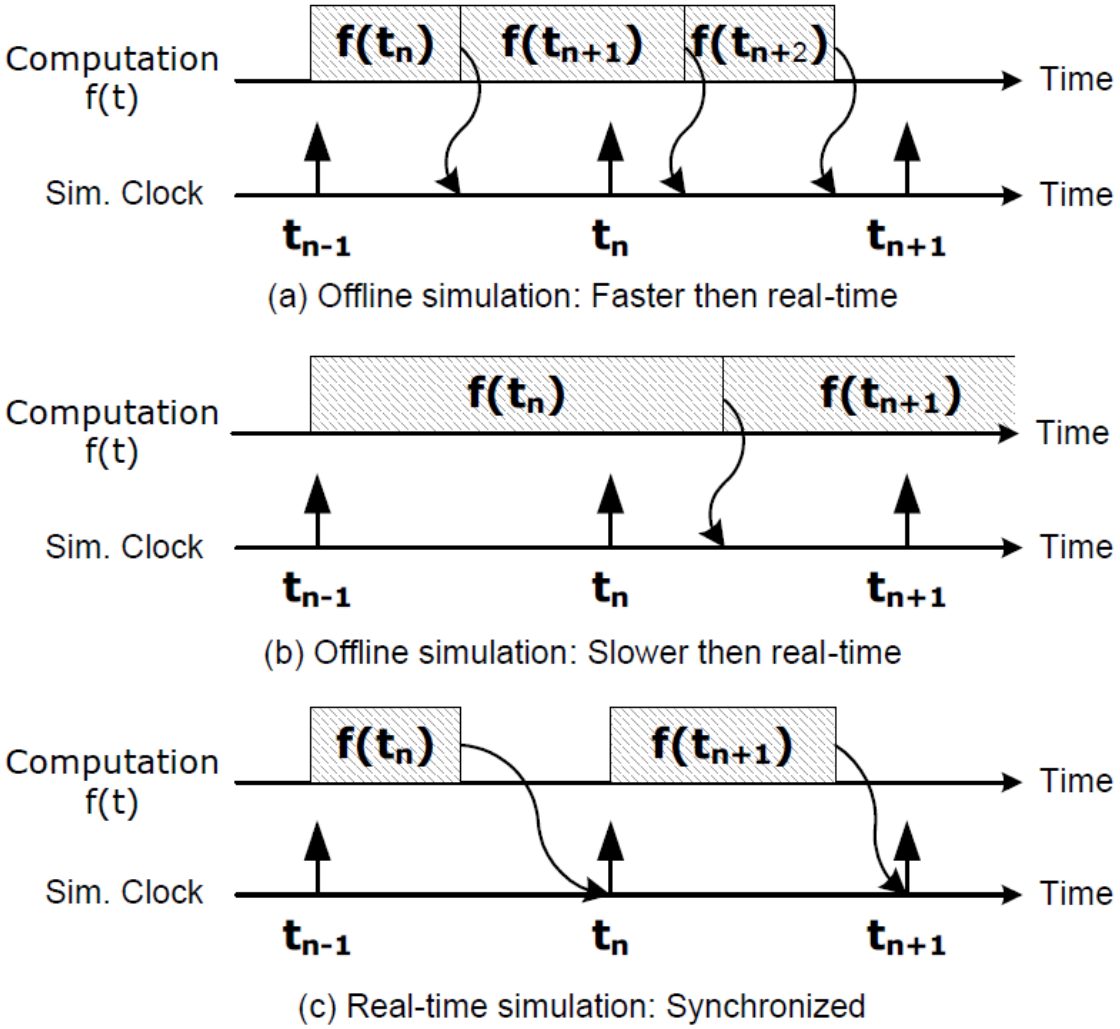

Figure 6. Real-time simulation requisites and other simulation techniques (Bélanger et al., 2008).

The simulation program in the GTPiL system of this thesis does not require real-time simulation according to the definition of the authors Bélanger, Venne, & Paquin (2008). Instead an accelerated simulation, where the simulator maximizes the calculation speed with no idle-time, is accepted and used in the GTPiL system. The calculation speed should however be either close to, equal to or faster than real-time.

2.4 Fast-running models and Hardware-in-the-Loop systems

To include GT-Power into a GTPiL system, it is most often required to use the FRM tool to create a fast running model. For this thesis the traditional approach was used, meaning that a base detailed model was created based on physical engine geometry, and then by calibration using experimental data a detailed model was obtained. This detailed model was then sent to the author, by whom the detailed model was converted into an FRM. In the article *Development and Calibration of One Dimensional Engine Model for Hardware-In-The-Loop Applications* by Andric et al. (2018), a novel approach was used where a base FRM was created directly from base data and geometry, and after that calibrated to obtain an FRM. Both the FRM creation approach used by the author of this thesis, and the one used by the researchers in the aforementioned article, are illustrated in **Figure 7**. The FRM created by these authors was based on a D13-700 MC Volvo Penta engine, which is an in-line 6-cylinder 12.8-liter, direct injected turbocharged diesel engine. The model can be seen in **Figure 8**, where the small amount of components characterizes an FRM. The combustion model employed for this FRM was the DI Pulse combustion model, which is a predictive phenomenological combustion model for direct injection diesel engines, developed by GT. As will be presented in chapter 4 GT-Power-in-the-Loop, the GTPiL system in this thesis does not utilize a predictive combustion model, instead a tabulated approach is used where the model interpolates between several combustion profiles depending on fuel mode and engine load.

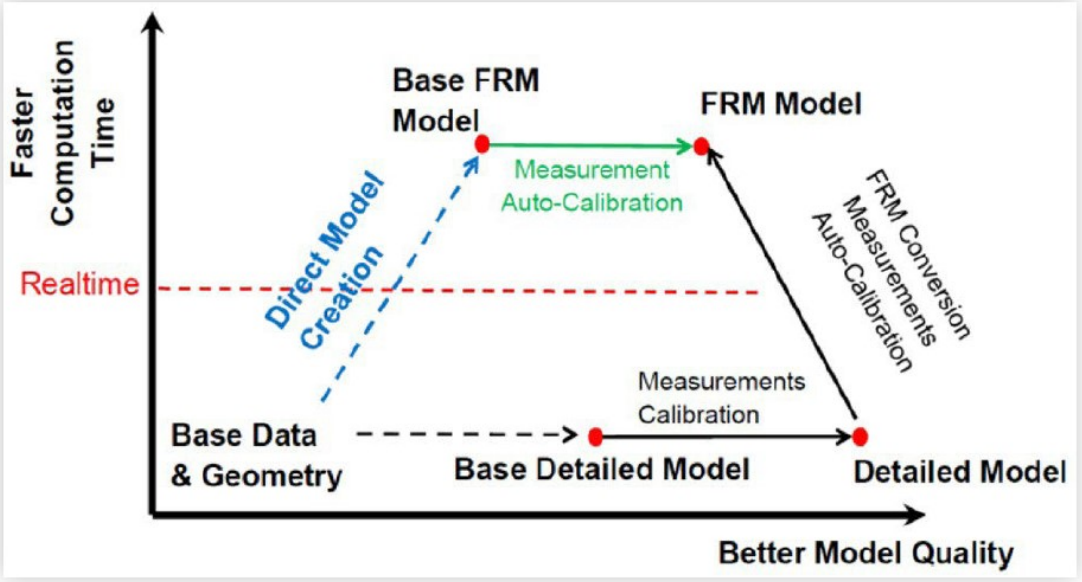

Figure 7. A schematic of FRM development process: traditional versus novel approach (Andric et al., 2018).

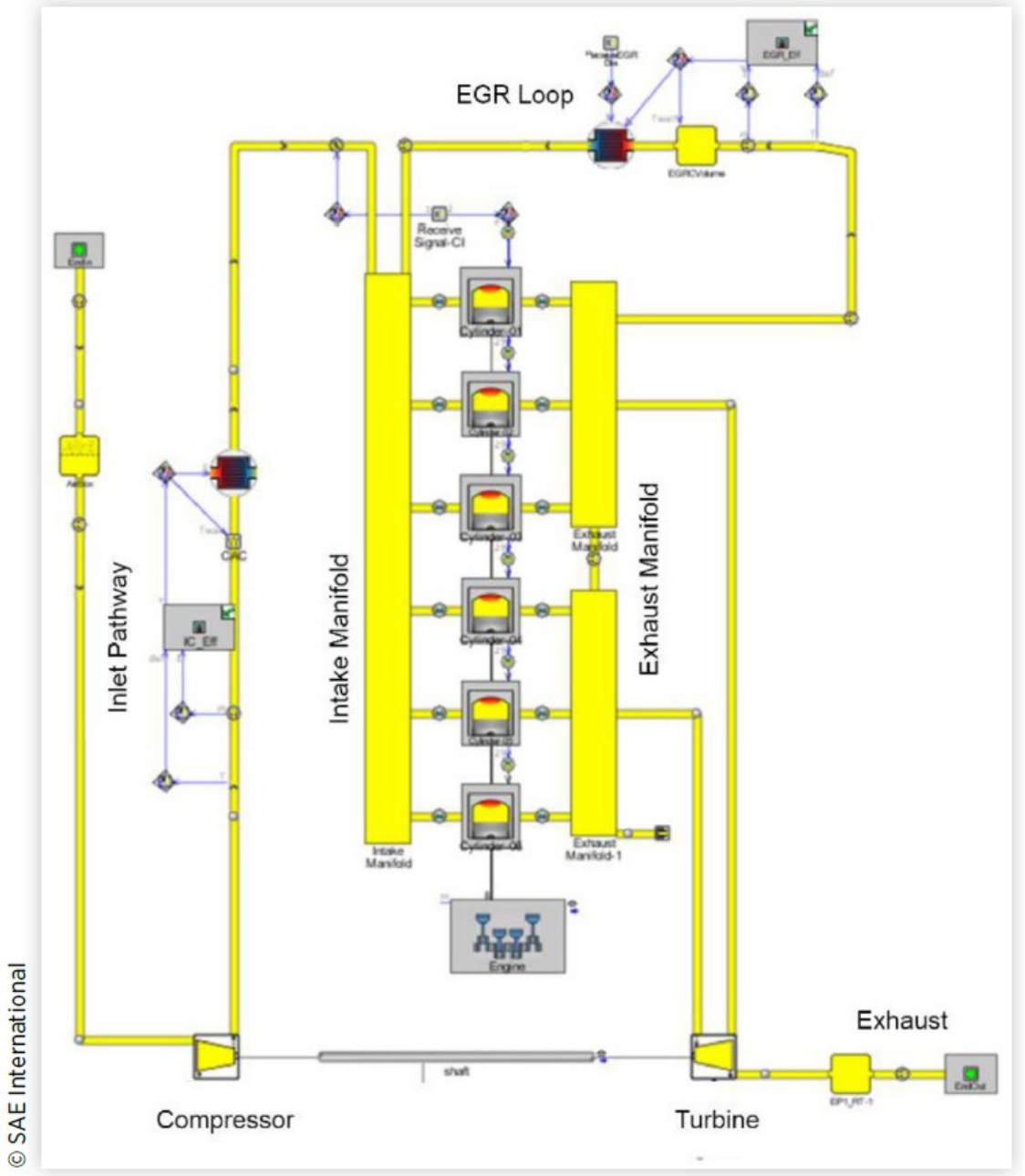

Figure 8. A schematic of GT-Suite 1D FRM representing a 6-cylinder twin scroll diesel engine (Andric et al., 2018).

The article by Andric et al. (2018) presents how to create an FRM directly from base data and geometry and how to calibrate it, but no HiL application was included in their work. Wu & Li (2016) conducted a bench test where a GT-Power FRM was embedded in a Simulink model connected to a dSPACE Simulator, forming a HiL system. They explain very thoroughly how they converted a detailed engine model into an FRM, and what to

consider in the process regarding the conversion and calibration. Their FRM can be seen in **Figure 9**, whose factor of real-time was substantially improved from about 30 to 0.8 as a result of the conversion, while well preserving engine parameter values like volumetric efficiency, brake mean effective pressure (BMEP) and exhaust manifold pressure. The top-most component named *SimulinkHarness-1* is an interface component in GT, used for exchanging data with Simulink (Wu & Li, 2016).

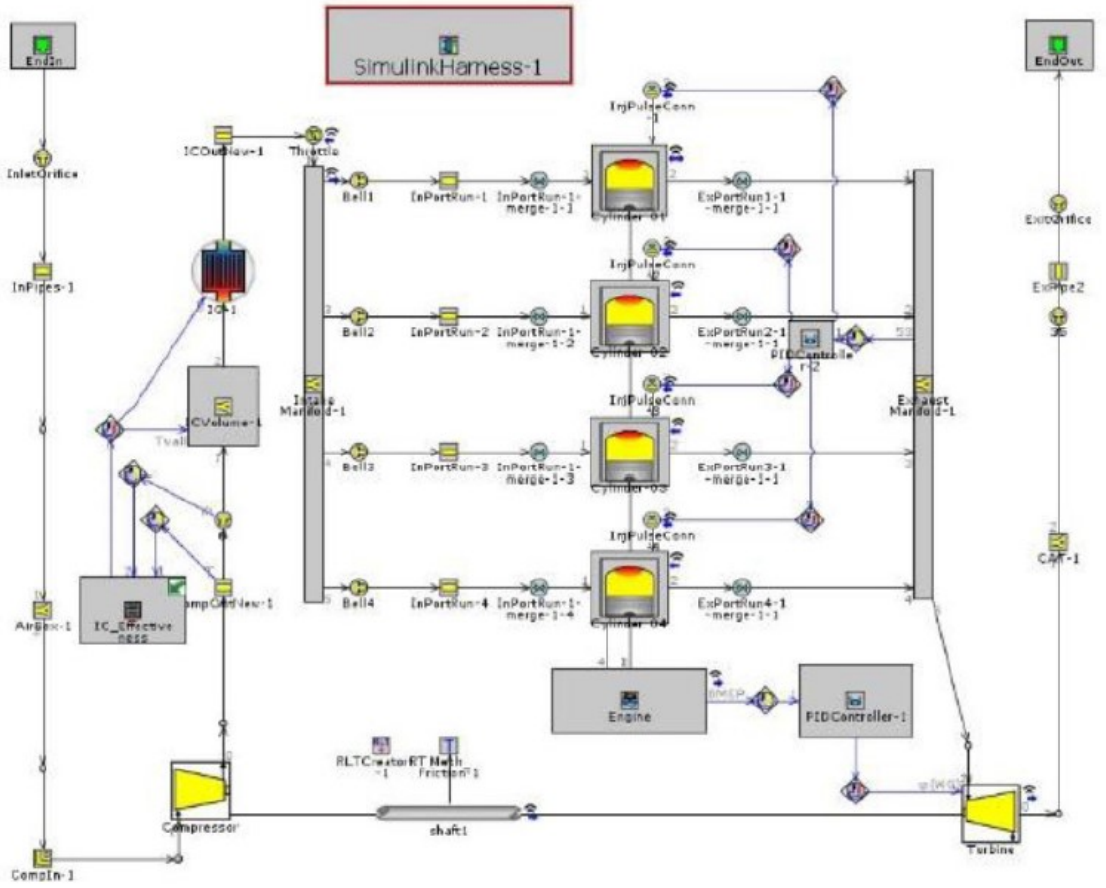

Figure 9. Final simplified FRM model layout with GT fuelling and turbocharger controllers (Wu & Li, 2016).

After the FRM conversion they created a co-simulation between the GT-Power FRM and Simulink (**Figure 10**). They had one Fuel-to-Air ratio (FAR) controller with the purpose of adjusting fuel injection pulse width with a PI-controller to target a specific FAR value, and a controller that adjusted the turbine EWG position with a PID-controller to target either a specific BMEP or intake manifold pressure value. Both controllers, shown in green and

red colors in **Figure 10**, were originally inside of the GT-Power model but were later moved out to the Simulink side of the co-simulation. Finally, they created an interface between this co-simulation model and the dSPACE Simulator (also called RCP, or Rapid Control Prototyping), moving the controllers again from Simulink to the RCP side (Wu & Li, 2016).

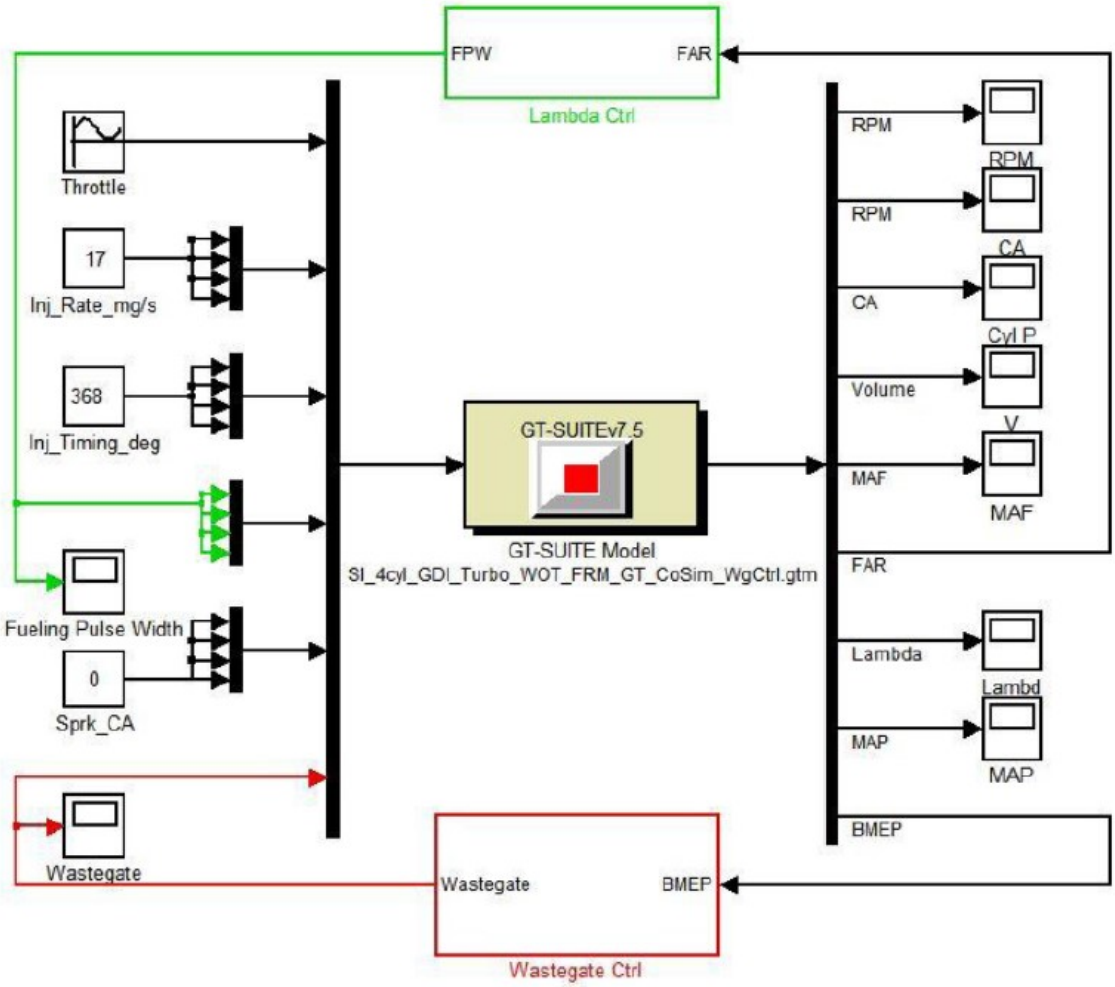

Figure 10. The Simulink part of co-simulation with fuelling controller and PID turbocharger controller (Wu & Li, 2016).

When the system was ready, a series of tests were conducted to validate the real-time capacity and consistency in results. One of these tests was a BMEP manipulation, seen in **Figure 11**. In this test BMEP was controlled to follow a step change from 14 to 18 bar (black line) under wide open throttle conditions, and the GT-Power detailed model (red

dashed line) was compared to the newly built HiL system (blue solid line). It can be noted that both the PID controller in the RCP and the turbocharger (TC) controller in the GT-Power detailed model reach the same steady state without noticeable errors in BMEP, waste gate position, intake manifold pressure, air mass flow, and torque. There is a good agreement for both steady and transient state when comparing the HiL system to the GT-Power detailed model, while being capable of running in real-time (Wu & Li, 2016).

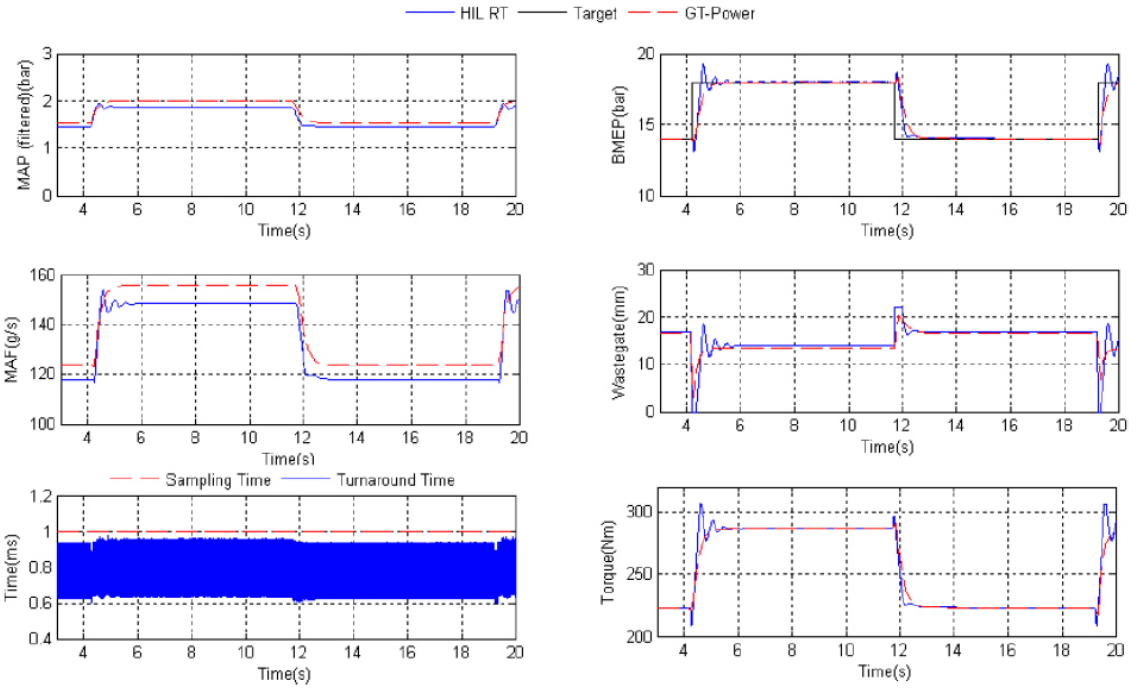

Figure 11. The test results of step response to BMEP in HiL/RCP comparing to simulation results with GT-Power detail model (Wu & Li, 2016).

In another study by Dorscheidt, Düzgün, Claßen, Krysmon, & Pischinger (2021), a novel HiL based co-simulation approach for virtual engine control unit (ECU) calibration of a vehicle gasoline engine was performed. This closed-loop HiL platform had a GT-Power FRM engine model and GT-Suite based real-time engine-out emission and exhaust after treatment system models, which were coupled to a hardware ECU with a throttle valve, variable valve lift actuators, a waste gate and fuel injectors as connected actuators. The co-simulation platform used was xMOD, and as HiL-simulator dSPACE was used with the necessary I/O boards. A setup flowchart of the real-time co-simulation platform is illustrated by **Figure 12**, where the GT-Power/GT-Suite models are integrated into xMOD (top

right in the figure). **Figure 13** depicts the powertrain models more in detail, with the engine model as the central part, through which most of the signal exchange between hardware (the ECU) and model environment occurs. The driver, transmission and chassis models are developed in Simulink, and run on the dSPACE platform. **Figure 14** shows the engine model and the corresponding signal exchange with the ECU, where parameters like i.a. engine throttle, fuel injection, engine speed and waste gate position are imposed onto the model from the ECU, while i.a. engine torque, lambda, air mass flow, boost pressure and boost temperature are sent back to the ECU. The in-cylinder combustion is here determined by a heat release rate simulated using the Wiebe function, whose parameters are determined by two artificial neural networks (Dorscheidt, Düzgün, Claßen, Krysmon, & Pischinger, 2021).

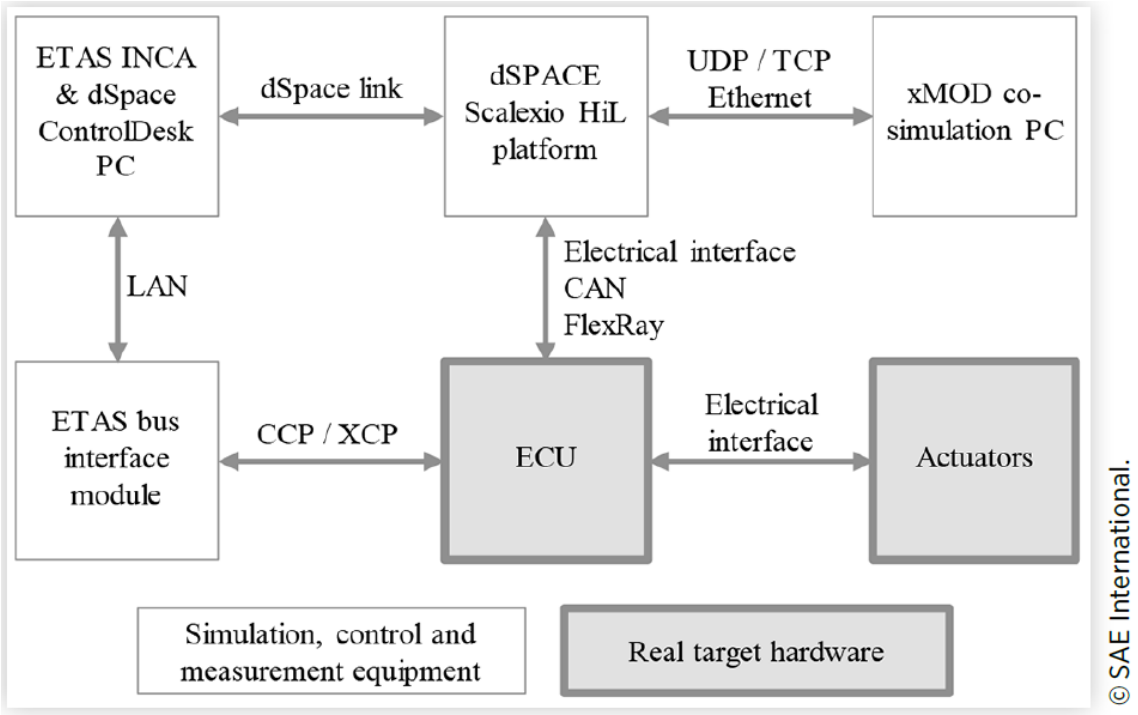

Figure 12. Setup of the real-time co-simulation platform (Dorscheidt et al., 2021).

After having built and calibrated all the models required it was time to confirm that the whole co-simulation platform worked as intended, by validating the performance of all real-time models in compound and during closed-loop operation with the ECU during vehicle transient cycle operation. The transient cycle chosen for this validation was NEDC,

the New European Driving Cycle. **Figure 15** shows a comparison in operating point related parameters between a virtual vehicle under warm engine conditions and measurements from a vehicle chassis dynamometer during this driving cycle. The graphs show a good agreement between vehicle and HiL measurements. The PI-controller of the driver model is also able to closely mimic a real human driver in how the accelerator pedal is commonly used (Dorscheidt, Düzgün, Claßen, Krysmon, & Pischinger, 2021).

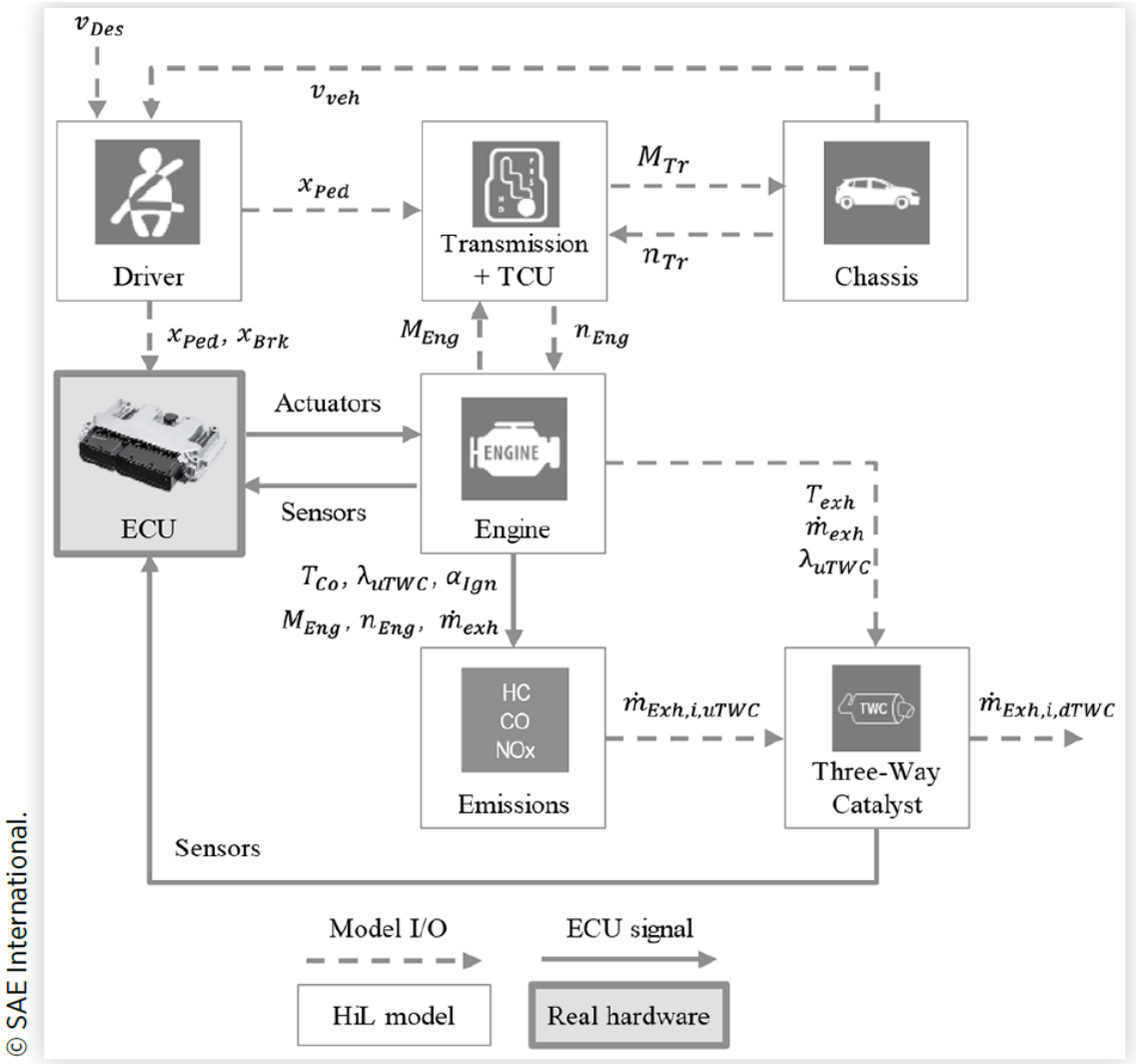

Figure 13. Schematic overview of closed-loop simulation environment (Dorscheidt et al., 2021).

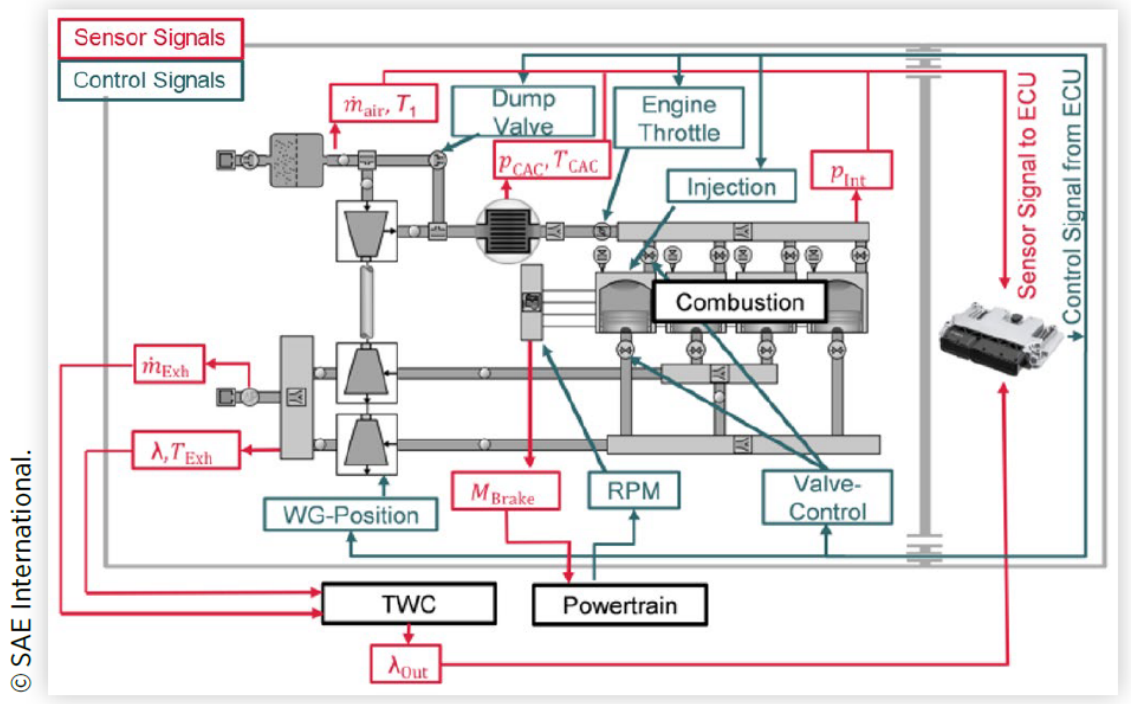

Figure 14. Schematic overview of the engine model and the corresponding signal exchange with the ECU (Dorscheidt et al., 2021).

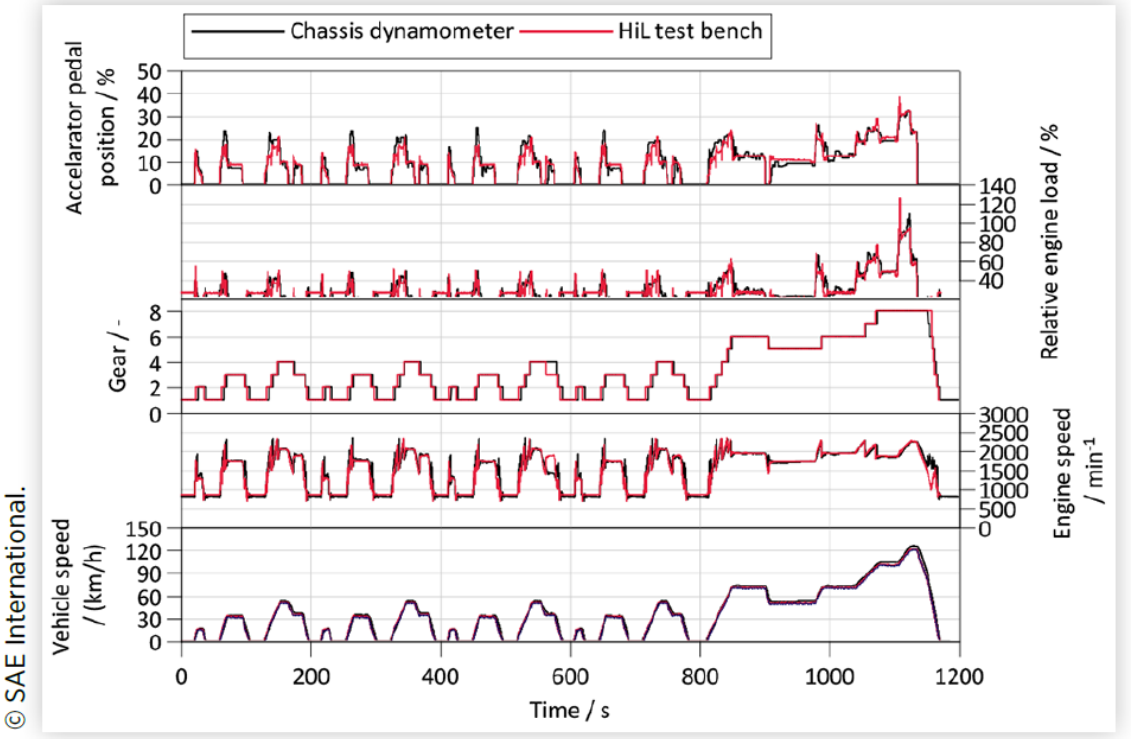

Figure 15. Comparison of NEDC HiL and chassis dynamometer measurement – Operating point related parameters during cycle operation (Dorscheidt et al., 2021).

Instead of using only the ECU and actuators on the hardware side, the authors Pirker, G., Mayr, P., Kranawetter, K., Bauer, R. et al. (2021) who wrote the article *Application of the HiL Method to Develop Transient Operating Strategies for Highly Flexible Power Generation in Gas Engine Power Plants* created a HiL system consisting of a virtual MCE and a research SCE test bed. The goal of this research was to transfer the transient behaviour of an MCE to an SCE test bed so that transient MCE combustion and control strategies could be developed directly on the SCE test bed. Proper transient operation of gas engines will be very important in future smart power generation that integrates lots of fluctuating renewable energy, to meet the high transient requirements while at the same time generating significantly less emissions than diesel engines and other fossil generation facilities (Pirker, et al., 2021).

The authors developed and tested two different HiL setups: one for grid parallel engine operation and one for island engine operation. In grid parallel operation the genset (engine and electric generator) feeds into a large power grid, and the influence on the grid frequency is hence negligible. The engine therefore does not receive any feedback of the fed-in power to the engine's speed, so the speed remains constant. In island mode on the other hand, also called *off-grid*, power is fed to a small power grid, meaning that if there would be an imbalance between generator and consumer power, the engine speed changes and with it the grid frequency. The genset must therefore adjust the power so that the engine speed and grid frequency is kept at a constant level. A speed-independent power controller is thus required for grid parallel operation, while a speed-dependent power controller is required for island operation (Pirker, et al., 2021).

The powertrain HiL interactions in island operation as well as in grid parallel operation can be seen in **Figure 16**. In the island operation HiL system, the power demand is set by the operator and passed to the voltage knee model that describes the generator behaviour. Due to the frequency deviation the generator reduces its voltage and therefore also the generator power to the powertrain model of the MCE. The indicated power measured at the SCE is the other input to this model. The calculated MCE speed is fed to the

SCE control unit, forcing the SCE to run at that same speed. To counteract the deviation between the rated and actual speed, a speed controller adjusts the amount of fuel gas injected into the SCE by changing the port injection duration. In the grid parallel operation HiL system, the electrical grid instead determines the engine speed and hence the speed control used in island mode is replaced by a power control. To minimize the deviation between target and current power, the power control also changes the port injection duration to adjust the SCE fuel gas amount. Skip firing is also used in this mode to reduce fuel consumption at part load.

The MCE is a 20-cylinder gas engine, from which the calculated charge air pressure (p_3) and exhaust back pressure (p_5) are transferred from the gas exchange model to the SCE via the SCE control unit. The p_3, provided by a screw compressor, and the p_5 are adjusted with flaps on the test bed according to how the 2-stage TC in the gas exchange model behaves, which is inside of the MCE powertrain model. The gas exchange model takes the measured indicated mean effective pressure (IMEP), the cylinder pressure and the current engine speed as inputs from the SCE. The compressors and turbines in the 2-stage TC use map-based models, while the TC shafts are modelled based on inertia. The combustion is not modelled however, instead the SCE's cylinder pressure is imposed in the gas exchange model. All the MCE models are implemented in Matlab/Simulink.

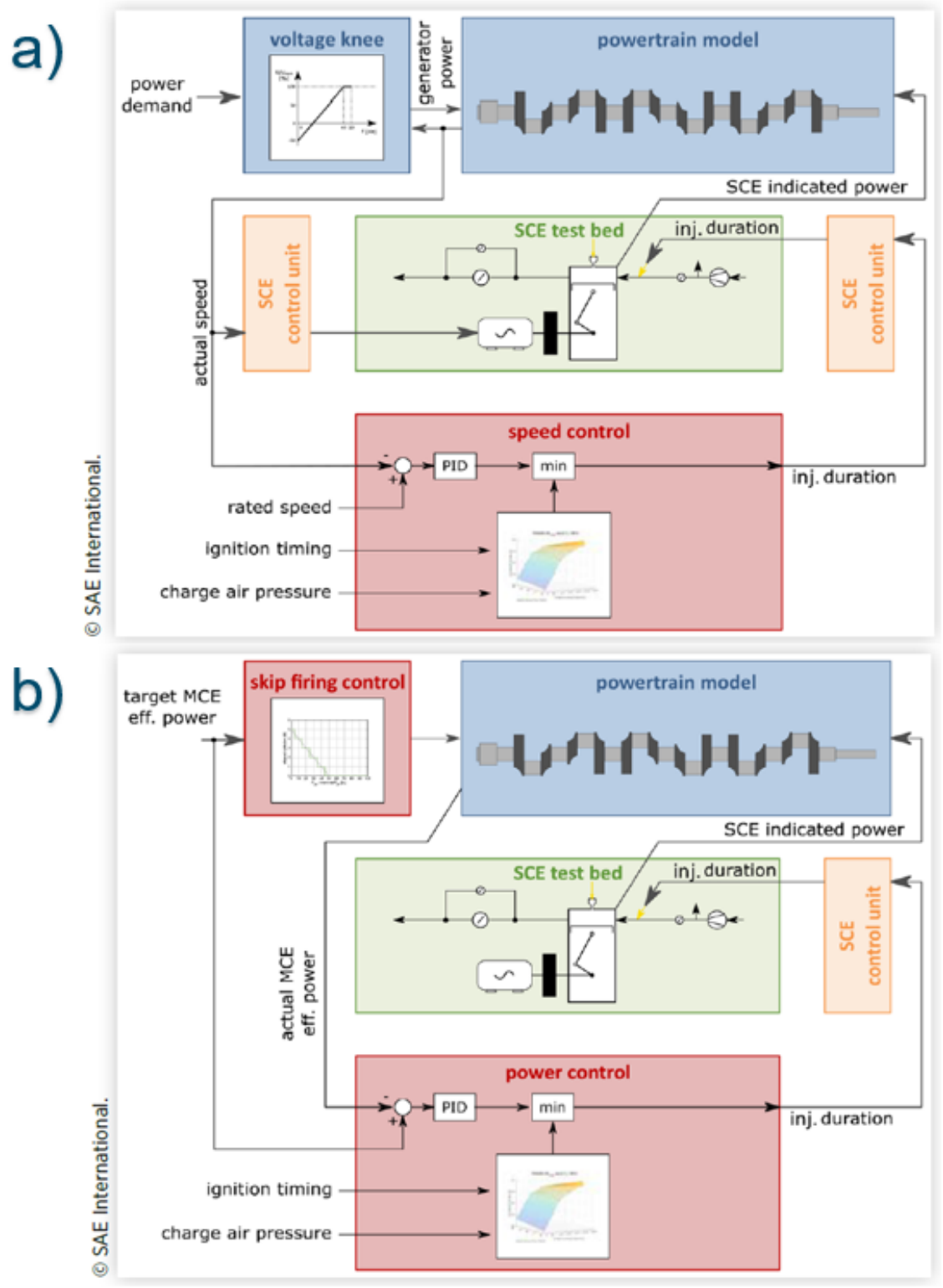

Figure 16. a) Powertrain HiL interactions in island operation, and b) Powertrain HiL interactions in grid parallel operation (Pirker, et al., 2021).

The results of a load step on the SCE from 47% to 80% of nominal power in island mode, and a load ramp with a power gradient of 9.47% of nominal power per second in grid parallel mode, are seen in **Figure 17** a) and b), respectively. In both a) and b), the original figures were cut to only show effective power, engine speed deviation, charge air pressure and exhaust back pressure. In the island mode load step in a), a drop in engine speed of 117 rpm occurs after the load step is taken. With the help of the speed controller, the TC and the air blow-off flap, the engine power and speed soon recovers. In the grid parallel mode load ramp, the engine speed is constant during the loading. The effective power deviates from the target during the first 60 seconds of the ramp due to the simulated turbo lag, caused by the TC inertia. The test bed is well able to follow the transient charge air and exhaust back pressure traces imposed on it from the MCE.

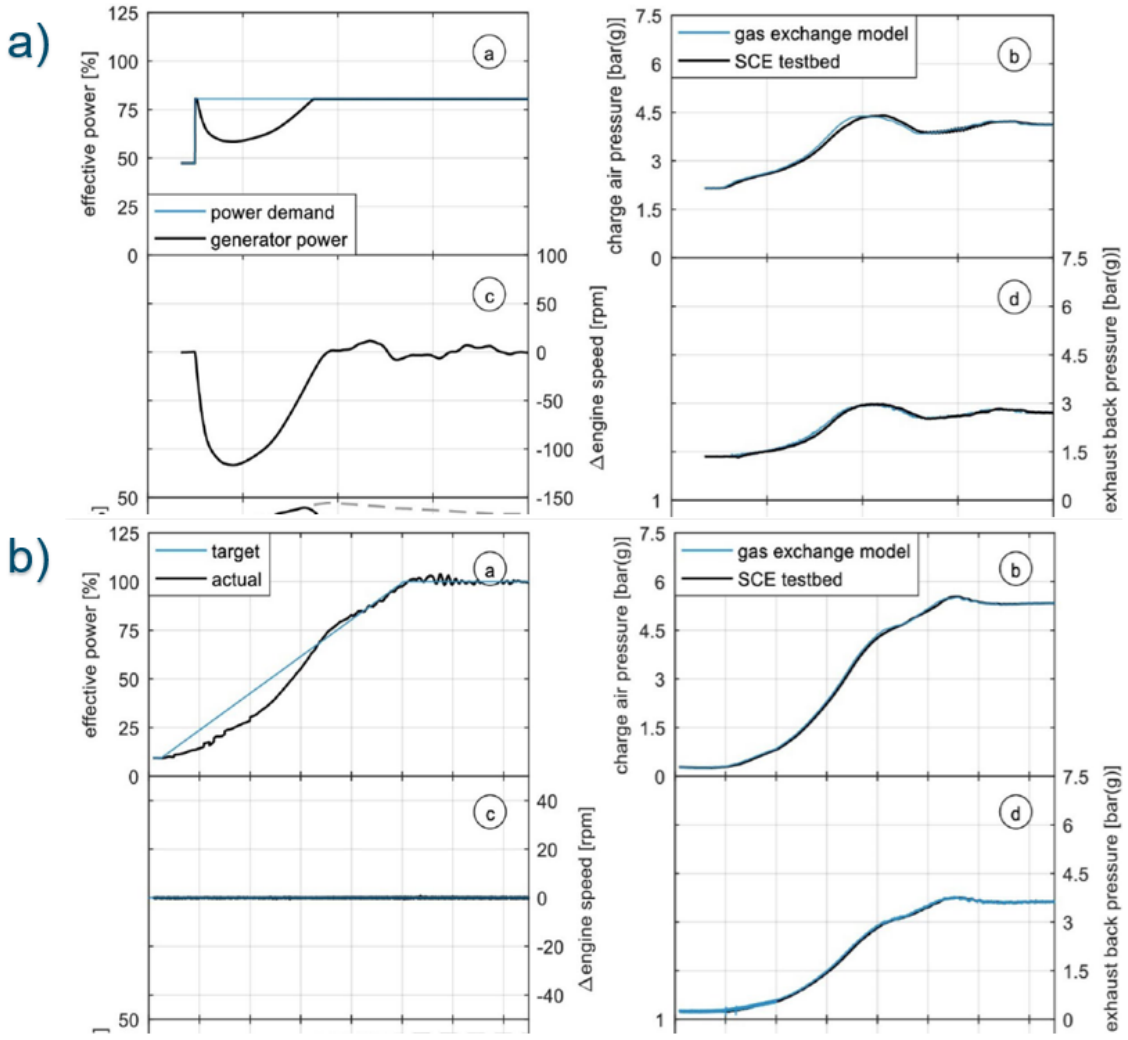

Figure 17. a) A load step from 47% to 80% of nominal power in island operation, and b) A load ramp with a power gradient of 9.47% of nominal power per second in grid parallel operation (Pirker, et al., 2021).

Wärtsilä has had extensive collaboration together with Vaasa university during the recent years, where researchers Hautala et al. in 2022 developed a control-oriented digital twin of a Wärtsilä 4L20 marine engine in their article *Toward a digital twin of a mid-speed marine engine: From detailed 1D engine model to real-time implementation on a target platform* (Hautala, Mikulski, Söderäng, Storm, & Niemi, 2022). They started from a detailed 1D GT-Power model and used reduction strategies to obtain an FRM, balancing the calculation speed and accuracy trade-off. Using experimental data from the 4L20 platform in the VEBIC Engine Laboratory they calibrated and validated the model at four representative operating points, and ended up with a model three times faster than real-

time and an accuracy loss within 5% tolerance levels for the governing outputs, including crank angle (°CA) resolved in-cylinder pressure. The idea was to implement this FRM onto a target machine that would be running in real-time; this target PC was a Dell Optiplex 760 PC (model year 2008) running the Simulink Real-Time kernel and using a Modbus card to communicate with a Speedgoat Performance Machine, which runs the engine control system in the laboratory and enables communication between the model and the real engine in a complete digital twin implementation. Similar to the GTPiL system developed in this thesis, a tabulated approach describing the combustion was used in this model.

To be able to compile the model for use on a real-time machine, a GT-Suite-RT license had to be used instead of the standard GT-Power license, and since this GT-Suite-RT license is particularly optimized for speed it reduced the calculation time over 70% compared to the standalone FRM, illustrated by the yellow line in **Figure 18**. The graphs show that the acceleration of the model speed did not bring substantial change in model accuracy, while it also stabilized the simulation speed over the cases, making the speed case independent compared to with the standard license. When actually embedding the model in Simulink and testing it on the target platform, the Simulink RT simulation had to be run with an imposed integration step size of 1 s to still meet the real-time requirements with limited computational performance, since this target platform had considerably lower central processing unit than the desktop PC used for prior simulations, seen in **Figure 19**. This was insufficient to capture the detailed dynamics of e.g. in-cylinder processes, but the graphs still demonstrate a stable, real-time implementation of the FRM on the target platform. This real-time capable model was later used in a follow-up project at Vaasa university, where the researchers Söderäng et al. used this digital twin in a real-time co-simulation with an electrical power plant model including battery storage, developed in Simulink (Söderäng, Hautala, Mikulski, Storm, & Niemi, 2022).

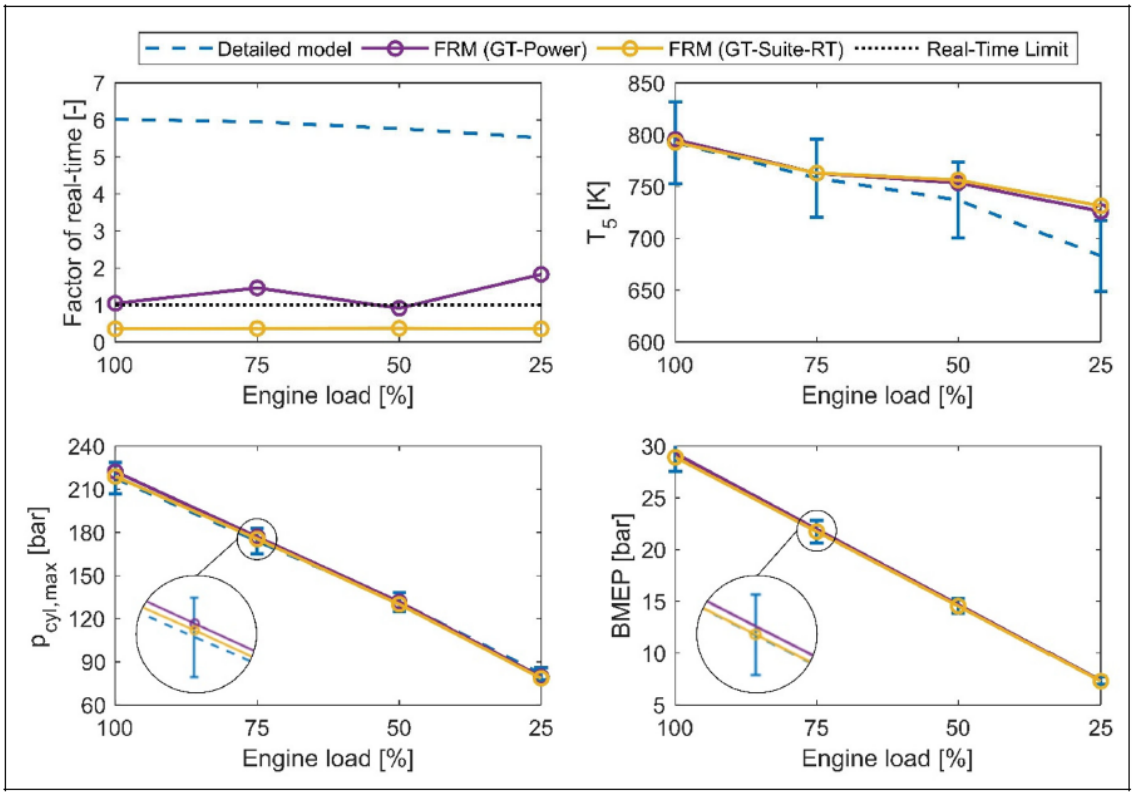

Figure 18. Real-time factors and simulated engine performance parameters of FRM in GT-Power and FRM in GT-Suite-RT after required modifications for Simulink coupling (Hautala et al., 2022).

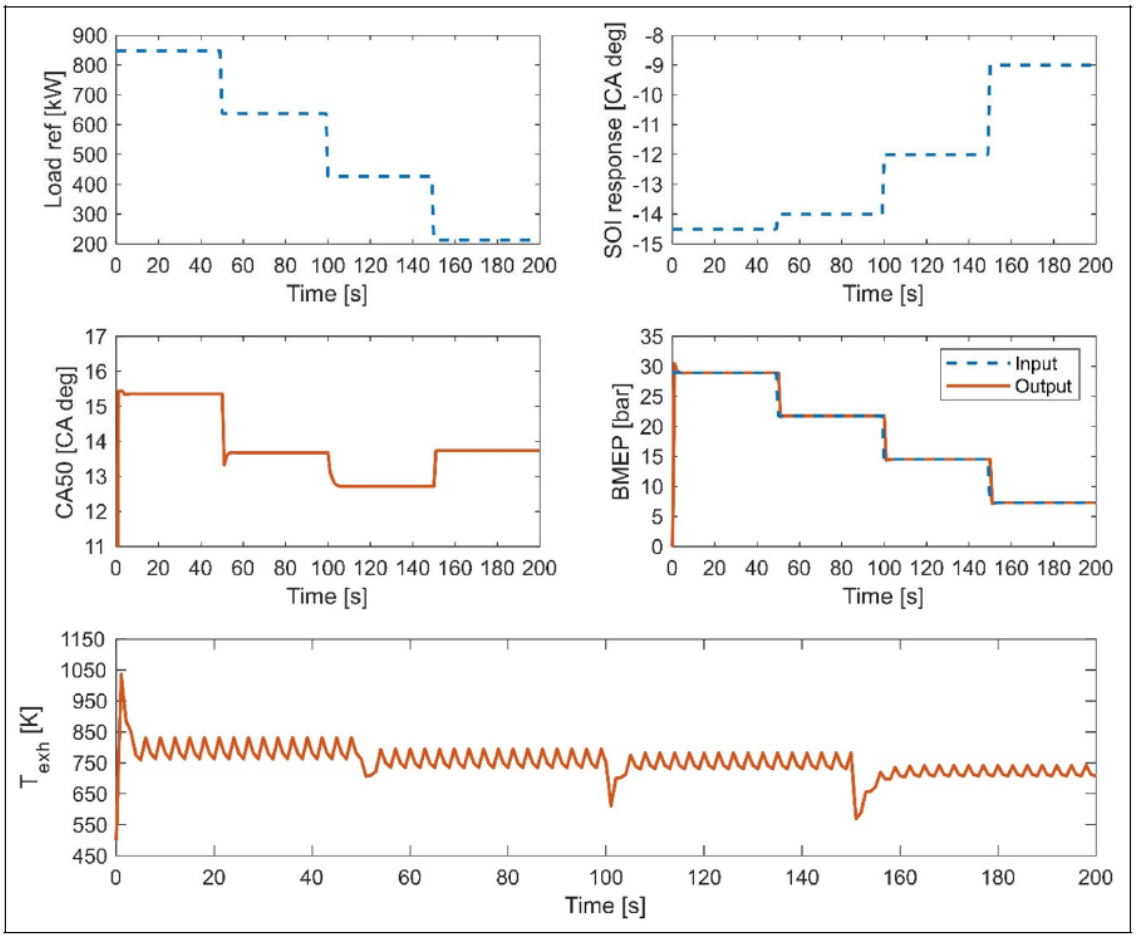

Figure 19. Simulated engine inputs and outputs in Simulink RT when time step of 1 s was used (Hautala et al., 2022).

3 Software involved in GT-Power-in-the-Loop

3.1 GT-Power

Gamma Technologies develops and licenses GT-Suite, a simulation software that includes a complete library of physics based modelling templates, of which one library is GT-Power (Gamma Technologies, 2022). The flow model in the GT applications involves the solution of the Navier-Stokes equations, i.e. the conversion of continuity, momentum and energy equations. These equations are solved in one dimension, all quantities are hence averages across the flow direction. There are two choices of time integration methods; an explicit method and an implicit method, where the difference between the methods lies in the solution variables and limits on time steps. The explicit method is used here, and is usually used for engine performance simulations in general, since the small time steps required by this method make it desirable when unsteady flow, pressure pulsation and high frequency wave dynamics are of interest. The primary solution variables in the explicit method are mass flow, density and internal energy (Gamma Technologies, 2022).

The whole system is discretized into many volumes, where each flowsplit component is represented by a single volume, and every pipe is divided into one or more volumes, connected to each other by boundaries. The discretization can be viewed in **Figure 20**. The scalar variables are assumed to be uniform over each volume, while the vector variables are calculated for each boundary. At each time step, the solver calculates new values for the primary solution variables using the following conservation equations (Gamma Technologies, 2022):

$$\frac{dm}{dt} = \sum_{boundaries} \dot{m}\,, \tag{1}$$

$$\frac{dme}{dt} = -\rho\frac{dV}{dt} + \sum_{boundaries}(\dot{m}H) - hA_s\left(T_{fluid} - T_{wall}\right), \tag{2}$$

$$\frac{d\dot{m}}{dt} = \frac{dpA + \sum_{boundaries}(\dot{m}u) - 4C_f\frac{\rho u|u|dxA}{2\quad D} - K_p\left(\frac{1}{2}\rho u|u|\right)A}{dx}. \tag{3}$$

In the above equations, $\dot{m}$ is boundary mass flux into the volume, m is mass of the volume, V is volume, p is pressure, ρ is density, A is cross-sectional flow area, A_s is heat transfer surface area, e is total specific internal energy (internal energy plus kinetic energy per unit mass), H is total specific enthalpy, h is heat transfer coefficient, T_{fluid} is fluid temperature, T_{wall} is wall temperature, u is velocity at the boundary, C_f is Fanning friction factor, K_p is pressure loss coefficient (commonly due to bend, taper or restriction), D is equivalent diameter, dx is length of mass element in the flow direction (discretization length), and dp is pressure differential acting across dx (Gamma Technologies, 2022).

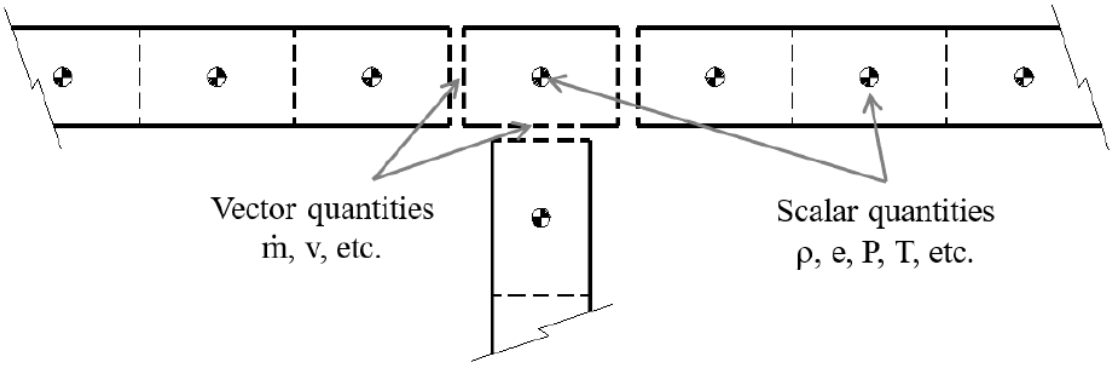

Figure 20. Schematic of staggered grid approach: scalars calculated at centroid, vector quantities at boundaries (Gamma Technologies, 2022).

In the explicit method, the right-hand side of the equations is calculated using values from the previous time step of the subvolume in question, and its neighboring subvolumes, to calculate the solution at the new time. This means that the time step must be small enough so that the relation between the time step and the discretization length satisfies the Courant condition (Gamma Technologies, 2022):

$$\Delta t = \frac{0.8 * M * \Delta x}{|u| + c}. \tag{4}$$

In the equation, Δt is the time step (s), Δx is the minimum discretized element length (m), u is the fluid velocity (m/s), c is the speed of sound (m/s), and M is a time step multiplier. Hence, to increase the speed of a simulation and be able to take larger time steps, one must also use larger discretization lengths for the subvolumes in the model. By using larger discretization lengths there are less subvolumes in the system requiring calculation of pressure, temperature etc. This is sometimes at the expense of accuracy, though. As an example, if increasing a given discretization length (dx = 25) by a factor of 2 (dx = 50), this means that with the explicit method, the computational time will decrease by a factor of ~4. The reason for this is that the time step will increase by a factor of 2 due to the Courant condition, and there will also now be only half the amount of subvolumes requiring calculation (Gamma Technologies, 2022).

3.2 Matlab and Simulink

Matlab stands for *Matrix Laboratory*, and is a registered trademark of The MathWorks, Inc. Matlab is a high-level language software used by millions of engineers and scientists worldwide to analyze systems and products. It is used for machine learning, signal processing, image processing, computer vision, communications, computational finance, control design, robotics, and more. Matlab is matrix-based, and hence convenient to use when expressing computational mathematics. Built-in graphics make it easy to visualize and gain insights from data. Matlab has got a vast library of pre-built toolboxes, making it easier to find needed algorithms for the task at hand. Matlab code can also be integrated with other languages, due to interfaces to C/C++, Java, .NET, Python, SQL, Hadoop, and Microsoft Excel (MathWorks, 2021).

All Matlab variables are multidimensional arrays, no matter what type of data. While other programming languages most often work with one number at a time, Matlab is designed to operate primarily on whole arrays and matrices. Accessing selected elements of an array or matrix is done by indexing. The workspace contains variables that are either created within or imported into Matlab from data files or other programs.

Matlab provides many functions that perform computational tasks, equivalent to subroutines or methods in other programming languages. The simplest type of Matlab program is called a script, which is a file that contains multiple sequential lines of commands and function calls (MathWorks, 2021).

Another registered trademark of The MathWorks, Inc. is Simulink, which is a block diagram for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. It provides a graphical editor, customizable block libraries, and solvers for modelling and simulating dynamic systems. Since Simulink is integrated with Matlab, one can incorporate Matlab algorithms into Simulink models and export simulation results back to Matlab for further analysis. Simulink has also got many libraries of predefined blocks for modelling continuous-time and discrete-time systems. The simulation engine has got ordinary differential equations (ODE) solvers capable of both fixed- and variable-step. Using the Legacy Code Tool, also C and C++ code can be imported into Simulink models. An example Simulink model is illustrated in **Figure 21** (MathWorks, 2021).

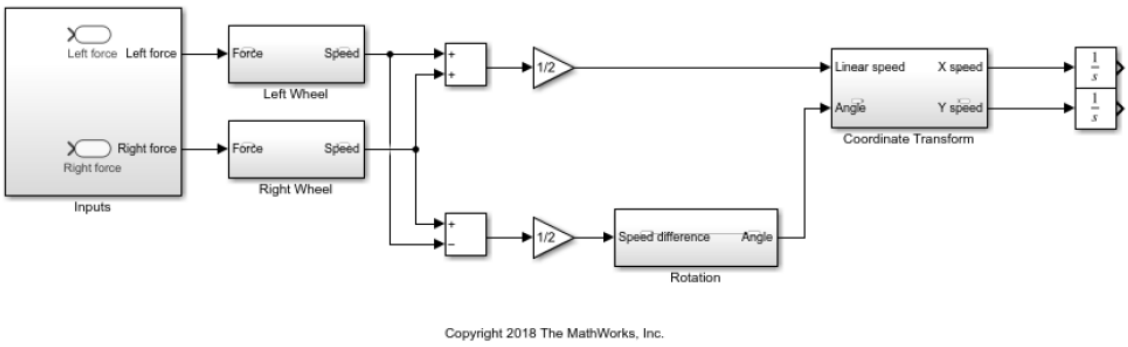

Figure 21. A Simulink model (MathWorks, 2021).

3.3 Modbus

Modbus is an application-layer messaging protocol that provides client/server communication between devices connected on different types of buses or networks. One device is a client, which initiates a request to another device, the server, which then executes the required actions and responds to the client. Modbus can communicate complicated requests and information; a Modbus level transmitter might be able to communicate i.a. top level, interface level, temperature and alarm data, all on a 3-wire RS-485 network, while a traditional level transmitter may communicate only a single process variable to another device on a 4-20 mA output. Modbus is a very flexible communication protocol since the standard does not define the physical layer, the mechanical wiring and electrical specifications used between devices. This means that Modbus can be used over common serial communications methods such as RS-232, RS-485, USB connections etc. Converters are readily available from one physical layer standard to another, making the wiring of a Modbus network even easier (Precision Digital, 2022).

The advantages of using Modbus are several, of which two common ones are the ability to easily use multivariable transmitters and the increased accuracy from a digital measurement and display system. Multivariable level transmitters can measure multiple process variables. Using an example application of a tank with oil and water, the multivariable level transmitter can sense the top level, interface level of the two liquids, and the average temperature in the tank requiring only three wires for RS-485, while traditional analog outputs and digital panel meters would require three isolated 4-20 mA signal loops with three isolated power supplies. The reading of a digital value using Modbus is more accurate since there is no analog signal inaccuracy or temperature drift in a Modbus system; the reading on the display is exactly what the transmitter is detecting (Precision Digital, 2022).

The user must set common serial communication protocol parameters for each Modbus device in the system. A unique device address/slave ID must be determined for each device. A baud rate, i.e. the communication speed in bits per second, needs to be set

identical for all devices. The data format, as well as the type of parity, should also match on all devices. Other possible parameters may also be set, like byte-to-byte timeouts, transmit delays and other settings (Precision Digital, 2022).

3.4 Excel and VBA

Excel, known by most engineers, is a software program created by Microsoft that uses spreadsheets to organize numbers and data with formulas and functions. Excel is typically used for data entry and management, financial analysis, accounting, charting and graphing, programming and organizing. Excel's functionality is enhanced by its various functions, formulas and shortcuts (Corporate Finance Institute, 2022).

Excel VBA on the other hand is Microsoft's programming language for Excel and all the other Microsoft Office programs, like Word and Powerpoint, and stands for Visual Basic for Applications. By using VBA, users can create programs/functions called *macros*, and this can be done in two ways. The first method is by activating the Macro Recorder, after which Excel will record all the steps a user makes and save it as a macro. When the recording is stopped, this macro is saved and can then be assigned to a button that can run the exact same process again when clicking on it. This method is however not very customizable, and due to using absolute referencing instead of relative, it is difficult to use these macros with variables and more automatized processes (Corporate Finance Institute, 2022).

The second method to create a macro is to code one using VBA. To access the VBA window, "ALT + F11" is pressed within any Office program. This opens a window with a file structure tree, properties, a debug pane, and the coding section that takes up most of the screen in the middle, where most of the coding occurs (**Figure 22**). When the macro code is finished it can be attached to certain triggers in e.g. Excel, where the macro can be activated at e.g. the push of a specific button on the worksheet, or when certain cells are modified (Corporate Finance Institute, 2022).

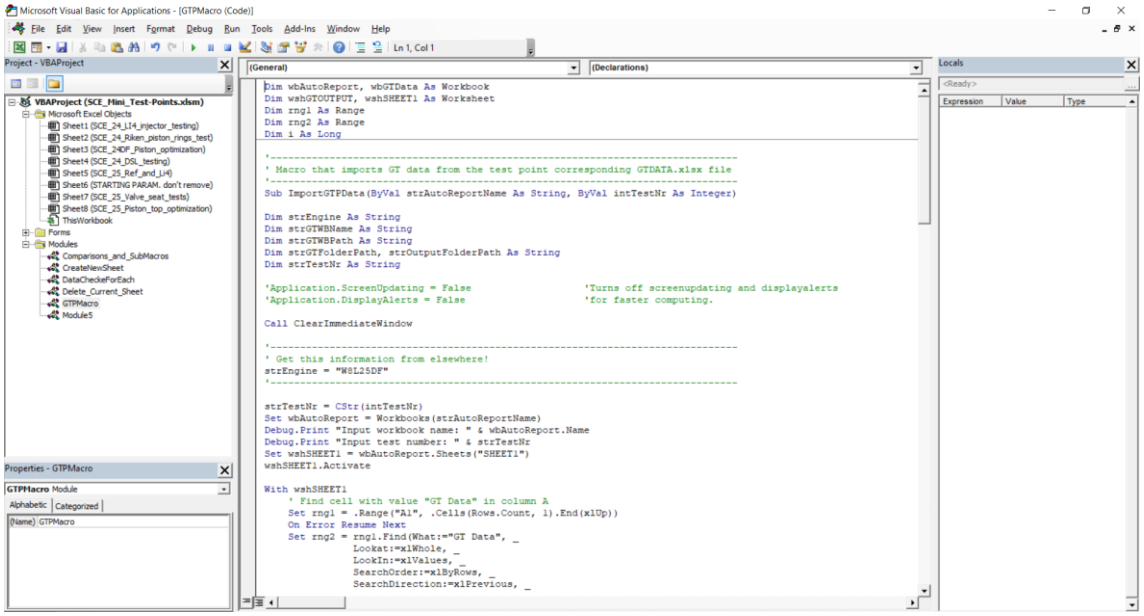

Figure 22. Snapshot of VBA code used in the GTPiL system.

3.5 Dewesoft

Dewesoft is a data acquisition (DAQ) software, where DAQ is the process of sampling signals that measure real-world physical phenomena and converting them into a digital form that can be used by a computer. A modern digital DAQ system consists of four vital components that form the entire measurement chain of physics phenomena, these are *sensors*, *signal conditioning*, an *analog-to-digital converter* (ADC), and a *computer with DAQ software for data logging and analysis*. These components are illustrated in **Figure 23** (DEWESoft, 2022).

Figure 23. Data acquisition system block diagram and the elements of the modern digital data acquisition system (DEWESoft, 2022).

DAQ systems measure physical phenomena such as temperature, voltage, current, strain and pressure, shock and vibration, distance and displacement, rotations per minute (RPM), angle and discrete events, weight etc. The primary purpose of a DAQ system is to acquire and store data, but they are also used to provide real-time and post-recording visualization and analysis of data. A sensor, also called a transducer, measures a physical phenomenon and converts it into a measurable electrical signal, which can be a voltage, current, resistance or some other electrical attribute that varies over time. The output of this analog sensor is connected to the input of a signal conditioner, that prepares the signal for digital sampling. This is done by isolating the signal from other sources of electrical potentials, filtering the signal from electrical interference or noise, and amplifying the signal from a low voltage up to a nominal level for digitizing. After the signal conditioning, the signal is converted from the analog domain to the digital domain using an ADC. The ADC output in the form of high-speed digital values are needed by the DAQ system for displaying and storing the signal data. On the X-axis of the digital values is the sample rate, i.e. the rate at which the signals are converted. Applications like temperature measurements do not require a high sample rate since the measurands do not change very rapidly, while other applications like AC voltages and currents require sample rates in the tens or hundreds of thousands of samples per second. On the Y-axis is the actual data, which today is mostly digitized with a resolution of either 16-bit or 24-

bit. By using a 24-bit resolution, an incoming one-volt signal can be divided into more than 16 million steps on the Y-axis. When the signal has been converted to digital, it can be processed by the computer by displaying and storing the data. **Figure 24** illustrates the digitizing steps from analog to digital, while **Figure 25** shows an example snapshot of the Dewesoft software (DEWESoft, 2022).

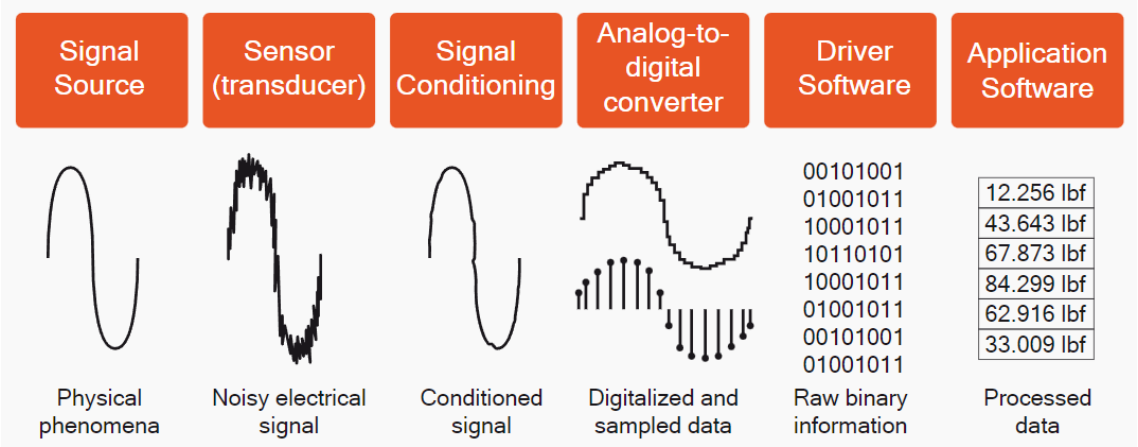

Figure 24. From analog signal sources to digitalized data ready for processing by computer and software (DEWESoft, 2022).

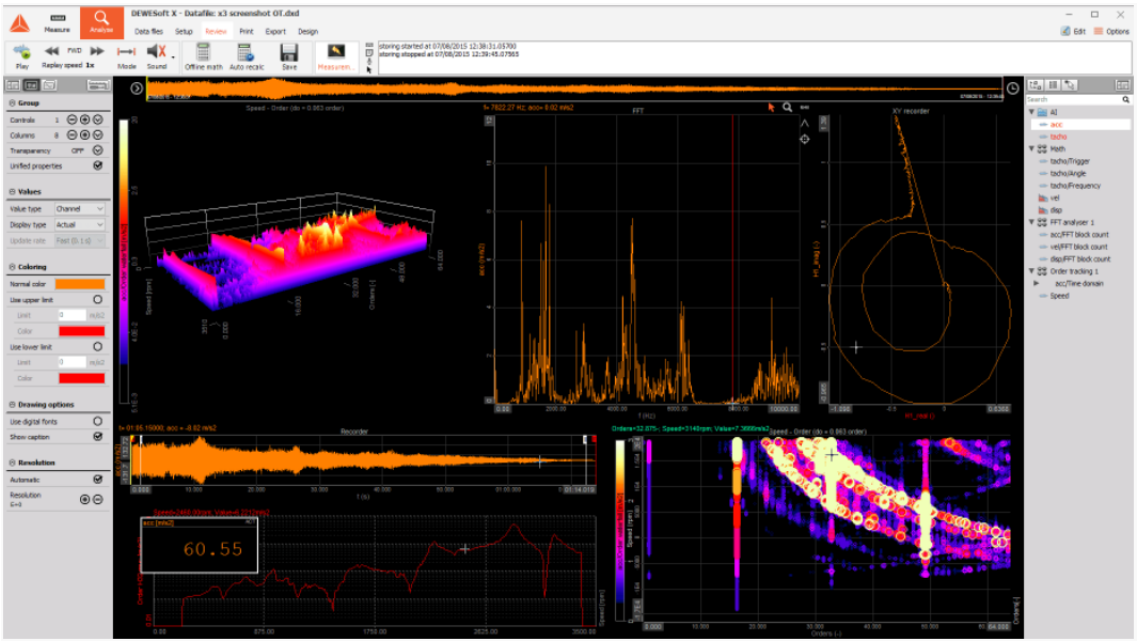

Figure 25. Dewesoft data acquisition systems provide a wide array of data analysis features inside Dewesoft X data acquisition software (DEWESoft, 2022).

4 GT-Power-in-the-Loop

4.1 Working principle

The overall working principle of the program is to start the GT-Power model and run it in parallel with the SCE by imposing signals from the SCE onto the model, and simultaneously sending the p_5 in the other direction, i.e. from the model back to the SCE. This working principle is illustrated in **Figure 26**.

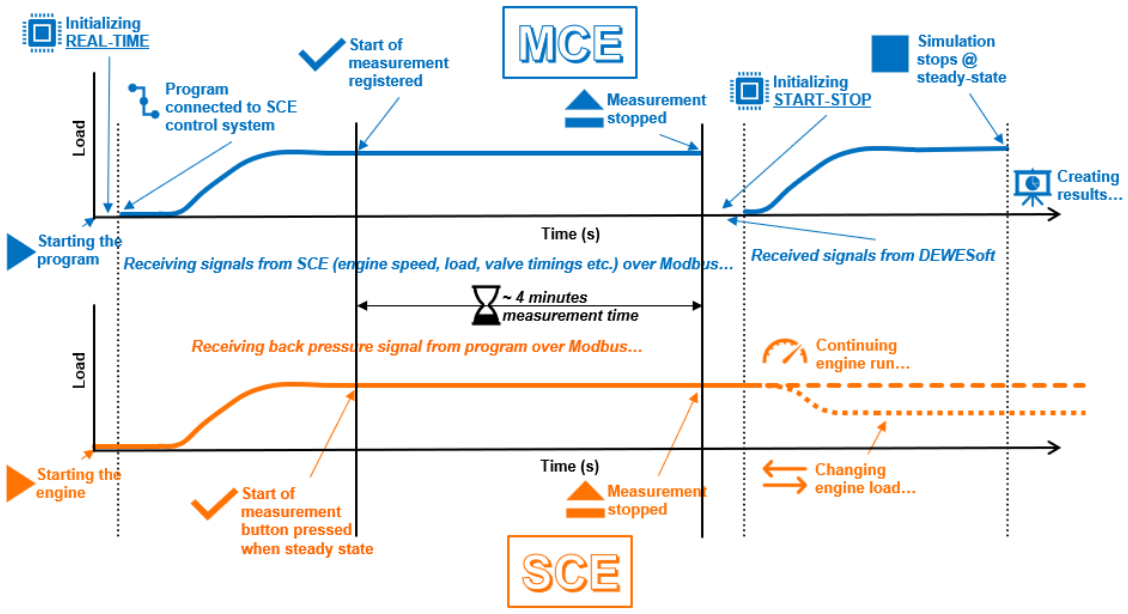

Figure 26. Program working principle when GTPiL system is real-time capable.

In **Figure 26** a fictitious process of performing a measurement for an engine test case is shown, with the SCE load curve in orange colour below and the virtual MCE load curve in blue colour above. The SCE is started into an idle mode with no engine load, after which the engine load is slowly increased to a load corresponding to what should be tested in the next engine test case. The program can be started at any time, but should be started before the start of measurement, to exchange signals with the SCE during the whole measurement process. In this example, the program is started at the same time as the SCE is started, but the GT-Power model in the program starts running in conjunction with the SCE after initializing the REAL-TIME (RT) script. This RT script is run during

the measurement process when exchanging signals, while the so called START-STOP (SS) script is run afterwards to create simulation output for that same test case. One of the differences between the RT and the SS script is that in the RT script the program can only run the GT-Power model via Simulink, since the signals can only be exchanged in real-time when utilizing a software like Simulink, while in the SS script the input to the model is only loaded and initialized before starting it, meaning that then it is also possible to omit Simulink and start the simulation directly from Matlab.

When the program has been initialized with the RT script, the program immediately connects to the SCE through Modbus via Simulink, to start the transfer of signals. The GT-Power model in the program is then fed with needed signals from the SCE in real-time, like engine speed, load, valve timings etc., making the model run with the same settings as the actual SCE during every time step. Hence, when the SCE in this case ramps up the load to a higher level, the in parallel running model follows. When the SCE is at steady-state and a reliable and stable measurement can be performed, the operator presses the *Start of measurement*-button in the SCE control system. The program senses this through Modbus and confirms that the measurement has started. The p_5 value calculated by the model and sent at every time step to the SCE can now be utilized by the SCE controls when adjusting the SCE's actual p_5. The measurement itself takes about four minutes, after which it stops automatically. The program again senses that the measurement has stopped, after which it stops the RT script and instead starts the SS script. By using polling, the program checks the status of when the SCE control system has created a file called *Autoreport*, from which the program will fetch the averaged variable values from the last 300 cycles which will be imposed onto the model for recreating the test case, i.a. engine speed, load, valve timings, λ target etc. An HRR curve averaged over 300 cycles will also be fetched and imposed onto the model, but how this is done depends on the operator. The program has been coded in such a way that depending on the operator's selection either Dewesoft or a software called Indicom can be used to fetch this HRR curve. If Dewesoft was selected, it can be further selected whether to fetch the by Dewesoft already calculated HRR curve, or to instead fetch the cylinder pressure and calculate the

HRR using a script in Matlab. If instead Indicom was selected, the only option is the latter. The in-house aspiration is however to replace Indicom with Dewesoft over the long term, so in this example the operator has selected Dewesoft for this purpose. The program again uses the method of polling to sense when this Dewesoft file has been created, and when it finds this file the HRR is then either fetched directly or calculated from the cylinder pressure, whereby it is imposed as a combustion object in the GT-Power model. The model now starts again and runs until reaching steady-state, after which it is stopped. Meanwhile if preferred, the operator can already start preparing the engine for the next test case by changing the engine parameters, since the program has already gotten all the input it needs for the SS simulation through Dewesoft. When the model stops, the program creates a set of output files which are sent to a common network folder. After this, the program automatically starts the RT script again, and the whole process can be repeated.

So far in this Working principle chapter, the true intention of the program has been presented, i.e. running RT during the measurement and having the SS script run afterwards to create output from the simulation. However, the program can also be run using SS only, omitting the RT part. This functionality was built in the early phase of the project in parallel with creating the FRM when it was yet unsure whether it would be possible to create a real-time capable FRM from the detailed model or not, and if it would be possible to exchange signals through Modbus via Simulink during runtime. If the real-time exchange of signals would not have been possible, a working principle corresponding to the one illustrated in **Figure 27** could have been used. In this convention, the model would be initialized with parameter values from Modbus via Matlab before starting the GT-Power model. The model would then run until steady-state after which it would stop, and would then send the calculated steady-state p_5 to the SCE. This process would repeat itself until the operator halts the program execution, or switches to the RT process instead. The calculated p_5 would hence be sent as discrete points to the SCE every time the model has reached steady-state. In this current version of the program, the model is not able to sense when a measurement is ongoing using SS only, which

means that the operator would manually have to pause the program and modify the *saveMeas* variable in the script, for the program to create output from the measurement point.

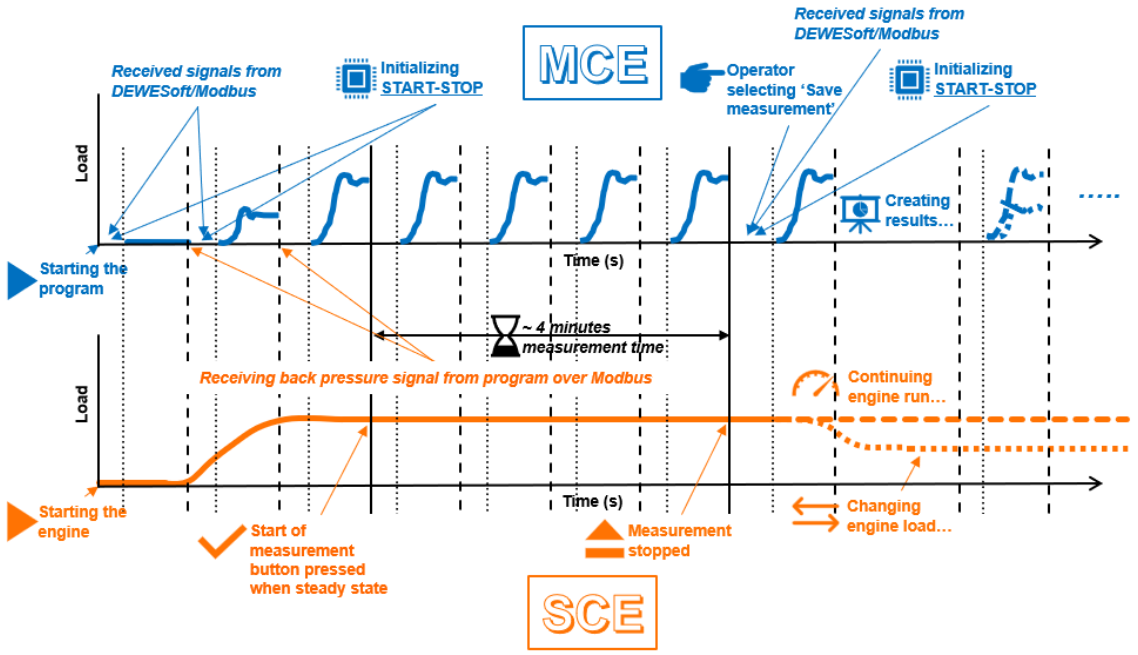

Figure 27. Program working principle when GTPiL system is not real-time capable.

As can be seen from the flowchart in **Figure 28**, the program runs on a stand-alone computer, which is connected to the software *D2T Morphee 2* belonging to the SCE control system via Modbus. Morphee is a test bed automation software that i.a. provides a graphical user interface between engine, test cell and operator, complete manual control of all engine and test cell parameters, modifiable safety and test point quality alarms and shutdown actions etc. Morphee is in turn connected to Matlab/Simulink, which is used for two main purposes in the SCE test environment: To perform different calculations (i.a. the Matlab turbo model is embedded here), and to work as an interface between Morphee and the actual automation system. Dewesoft is also connected between the engine, Morphee and the program.

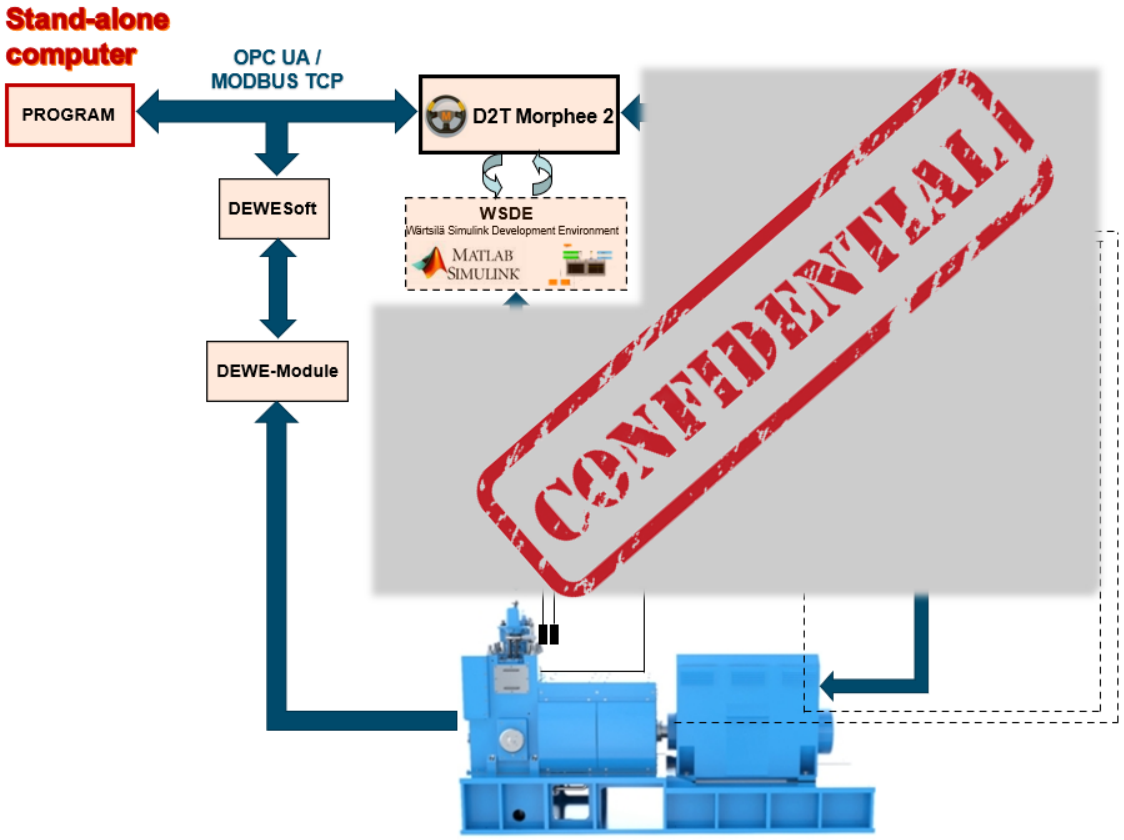

Figure 28. Flowchart of SCE control system.

4.2 Program structure

4.2.1 GT-Power model

The FRM used in the program was created from a detailed GT-Power model, which was already calibrated against engine test data. As can be seen from **Figure 29**, the detailed model is comprehensive with lots of components, while the FRM in **Figure 30** is much more simplified. To perform the FRM conversion, the FRM Converter tool in GT-Power was used to first tag different subsystems, and then simplify them one at a time. To make sure that similar dynamics as in the detailed model were maintained, it was sometimes necessary to calibrate certain parameters after the simplification of a specific subsystem. The discretization length was also increased for every pipeline part to speed up the model. When all the tagged subsystems had been converted using the automatic tool,

also manual merging of flow volumes was executed. Unnecessary components used for performing calculations not needed for this project were also deleted. The large gas pipe to the right in the detailed model was removed and replaced with only one injector per intake port runner in the FRM, where a Load PID was determining the fuel mass to be injected through each injector depending on the load demand. The turbochargers in the FRM are inside of the *Turbo* subassembly in the middle of **Figure 30**; the FRM is modular in that the operator can select whether to run 1- or 2-stage turbocharging, and depending on the choice, either the 1- or 2-stage subassembly is imported and merged into the main model when the simulation is started. The operator can also select from a large group of compressor and turbine maps which one to use in the simulation.

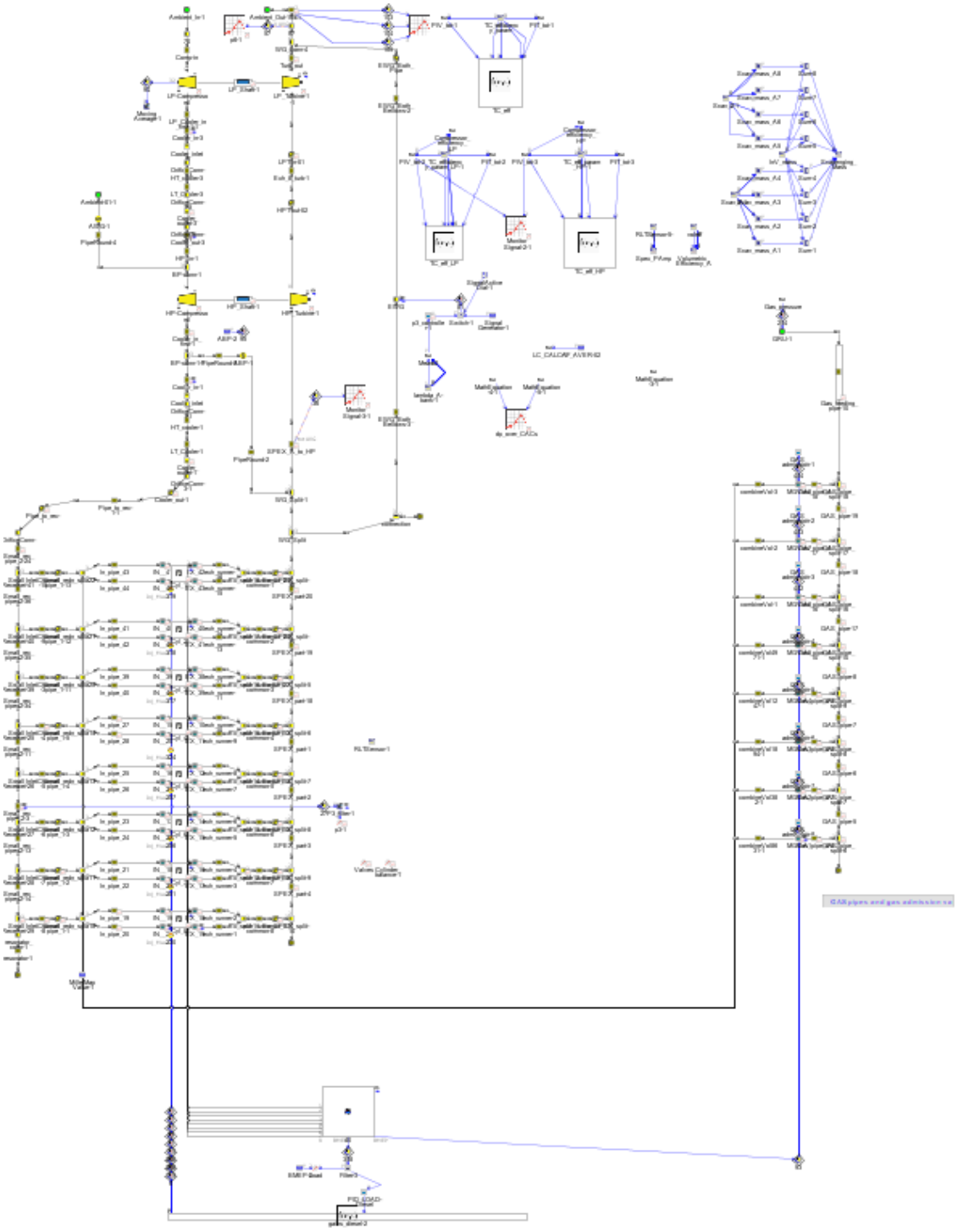

Figure 29. Detailed GT-Power model.

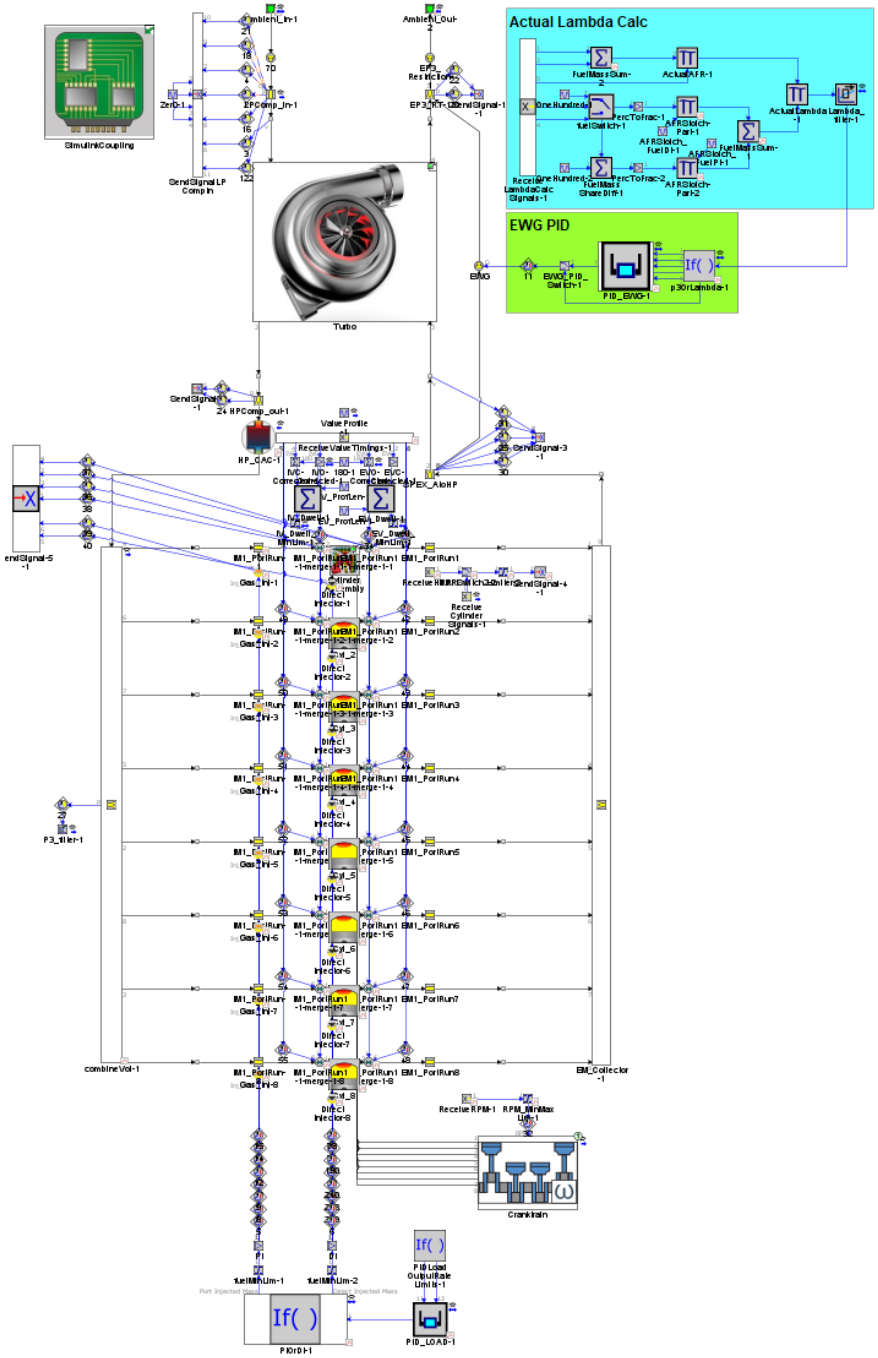

Figure 30. Fast running GT-Power model.

Figure 31 and **Figure 32** show the outcome of the FRM conversion, by comparing the FRM with the detailed model. The Y-axis has been removed due to confidentiality. In **Figure 31** the FRM in a blue solid line and the detailed model in a red solid line have been compared in regard to several cycle-averaged parameters. The top left graph shows the BMEP, that naturally is matching well due to the PID controller in both models adjusting

the fuel mass to target the BMEP. The BSFC graph to the right reveals the efficiency in the form of brake specific fuel consumption, i.e. the amount of fuel in both cases needed to achieve the BMEP levels in the BMEP graph. The BSFC level is highly dependent on i.a. the friction mean effective pressure (FMEP), determined by the friction model located in the EngineCrankTrain template. The relation between BMEP, IMEP and FMEP is shown in the following equation:

$$BMEP = IMEP - FMEP \, . \tag{5}$$

Hence, the larger the friction, the smaller the BMEP, and the more fuel must be injected every cycle to reach a certain BMEP level. There are several terms that can be tuned in the friction model, like i.a. a constant pressure term, cylinder pressure dependence term, piston speed dependence term etc., but in the simulations run using both the FRM and the detailed model, only the *constant pressure term* was used, while the others had been given a value of 0. However, a case dependent value for the constant pressure term had been used for the detailed model, with a value of 3.1 bar at high loads and only 2.7 bar at the lowest load, while a case independent constant value of 3.1 bar was used for the FRM for all loads. This explains the difference between the models regarding BSFC. To make the FRM more predictive it is desired to find and use case independent values for as many tuning parameters in the model as possible, including the ones for the friction model. The friction model is hence in need of calibration in the future improvements of the GTPiL system. The EWG PID controlled the EWG valve opening angle to target the boost pressure p_3 which is why this curve matches well, but still the air flow through the engine is slightly underpredicted for the FRM at higher loads, meaning that some minor tuning of parameters affecting the flow in either the intake valve or port is needed to give a better match. This lower air flow at higher load is most probably also the reason for the slight decrease in the exhaust pressure p_5 for the FRM at higher loads. The middle right graph shows the 1 / Factor of Real-Time graph, where a value of 1 at the grey dotted line is exactly real-time, while values above this means that the simulation is faster than real-time. One step on the Y-axis in this graph equals 0.5, so the FRM is marginally faster

than 2 times real-time. It will however not be possible to maintain this speed when using the FRM as part of the GTPiL system, since Simulink will also bring some overhead and contribute to slowing the simulation speed down a bit.

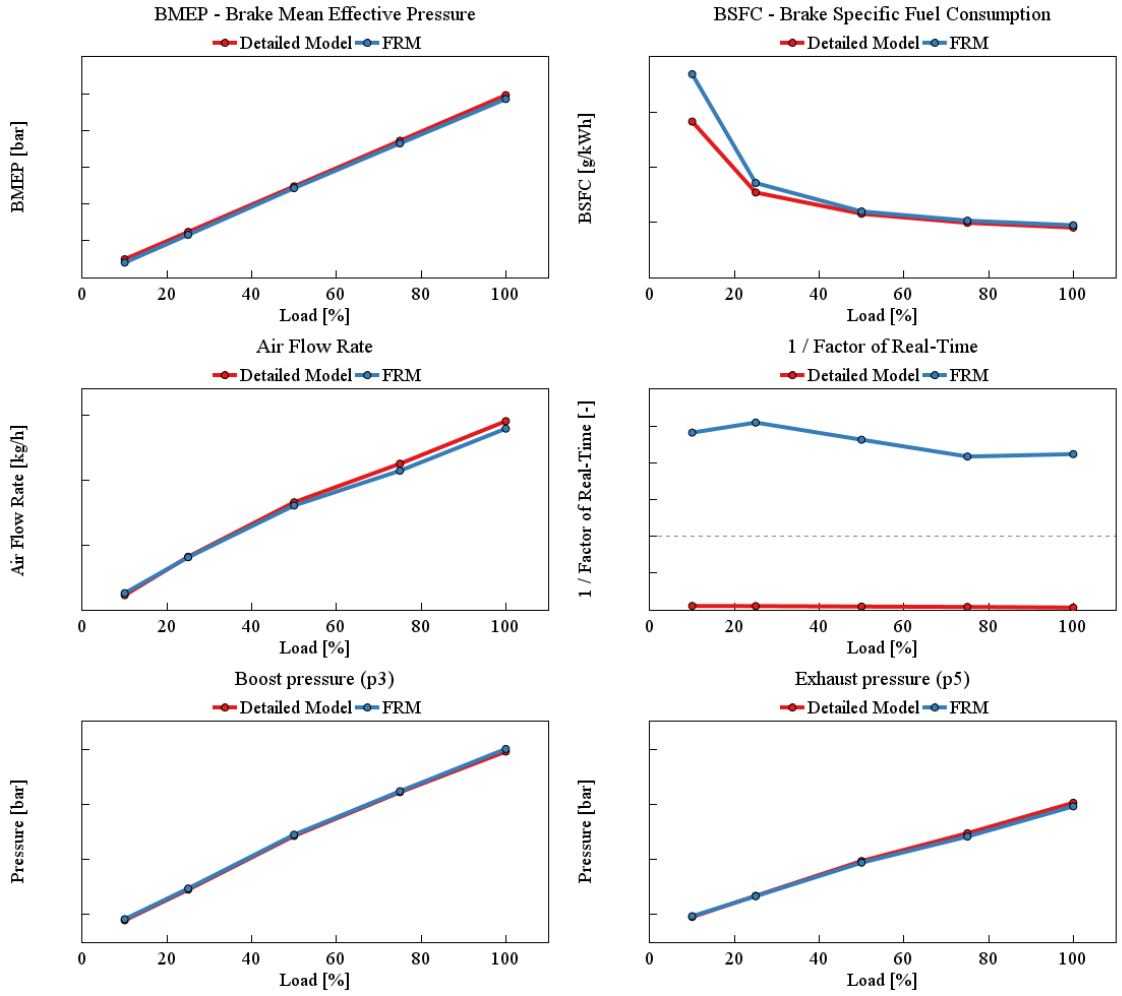

Figure 31. Cycle-averaged parameters comparison between FRM and detailed model.

Figure 32 instead shows the in-cylinder pressure of the detailed model and the FRM, with the imposed HRRs for both models in orange. As can be seen from the highest load point in the top left graph, the FRM conversion had a minor effect on the peak cylinder pressure due to the minimal decrease in the peak value of the blue curve, but the difference in the end was only ~3 bar. Looking at the other loads, there is a very good match in in-cylinder pressure. The Y-axis scale is the same among all graphs.

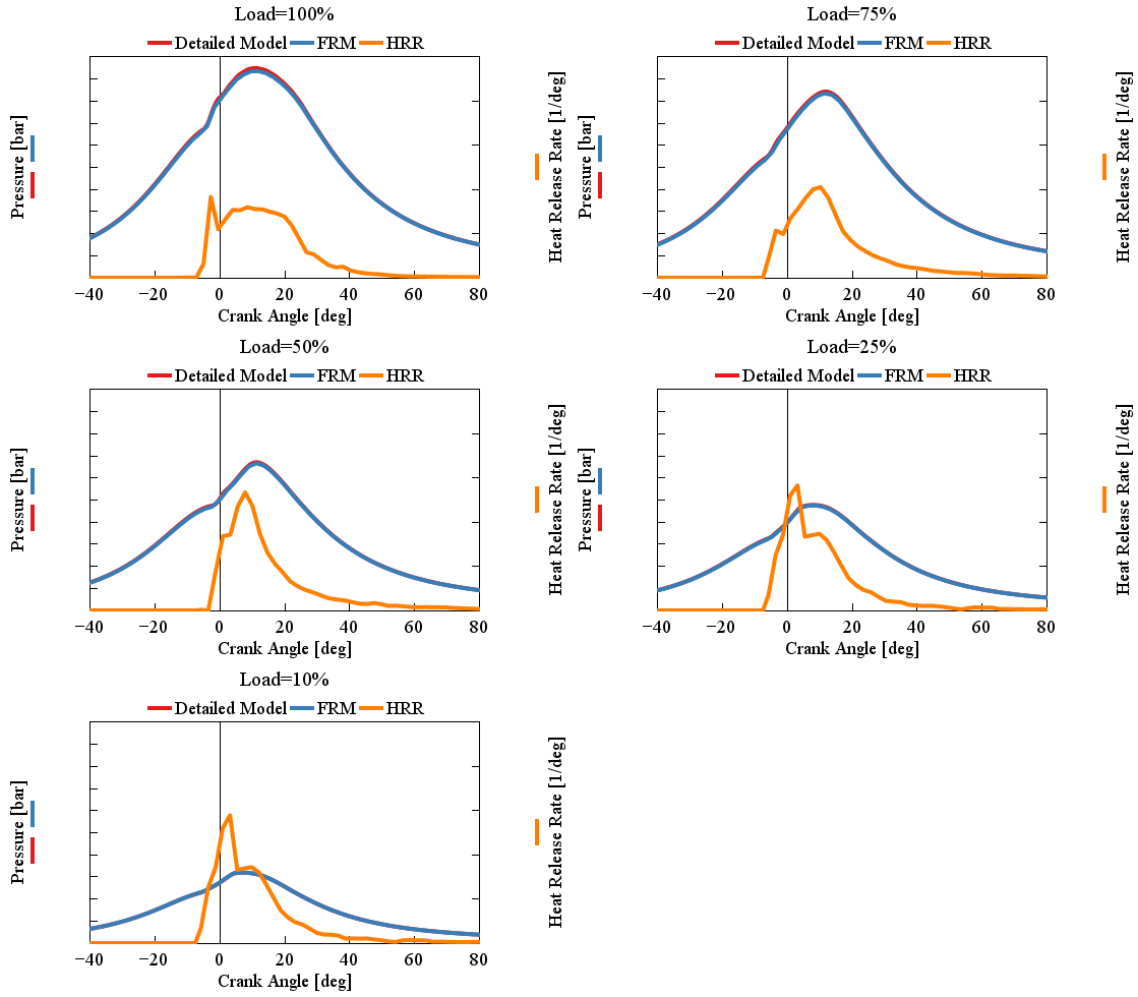

Figure 32. In-cylinder pressure comparison between FRM and detailed model.

4.2.2 Matlab script

The Matlab part is the main element of the program, from which the program is initialized and started by the operator. The Matlab code is divided into several scripts and functions, of which the *MAIN.m* script is the main script from which the program is started. In the beginning of the main script there is a selection menu for the operator; the selection menu from the program version 42 is presented in Appendix 1, but the most important contents of this selection menu will be explained here.

With the *engineType*, *stroke* and *connectingRod* variables the user can give mechanical information about the engine. The *runAlt* variable determines how the simulation will be run; the user can either select to run SS in a continuous mode either via the GT-Suite GUI or Disk Operating System (DOS) mode in Matlab, or then to run RT using either the normal GT-Suite license or the faster xRT license. The *fetchHRR* variable determines how to fetch the crank-angle based arrays; whether to fetch them from Dewesoft or Indicom and whether to calculate the HRR in Matlab or use the already calculated one if Dewesoft is selected. With *VSS_BR* it can be selected if the HRR should be interpolated among tabulated HRRs already inside of the GT-Power model, or whether the actual SCE HRR should be fetched every cycle from the SpeedGoat control hardware and also be imposed every cycle onto the GT-Power model. Fetching of the HRR via SpeedGoat is a Simulink module that has been developed in a separate Master's thesis, that shortly will be presented in chapter 4.5 Modelling challenges. The *pathsSrcDest* is used for selecting the folders from which to take the input data when running the SS script, so that the script can be more easily tested. The *mainFolderPath* variable sets the folder path to where all scripts and models are located. The *IndicomFileFolderReal*, *AutoreportFileFolderReal*, *DeWeFileFolderReal* and *outputFolderPathReal* determine where the script can find necessary input data for starting the SS simulation and also where to store the output files it generate when the *pathsSrcDest* variable is set to 2, while the *IndicomFileFolderLocal*, *AutoreportFileFolderLocal*, *DeWeFileFolderLocal*, and *outputFolderPathLocal* determine the same when the *pathsSrcDest* variable is set to either 1 or 0.

The second page of the selection menu appendix starts with the *GTversion* variable, that is needed by the script command for starting the SS simulation, and must correspond to the year of the GT Power model's version. The *SimulinkModelName* variable determines the name of the Simulink model and is needed by the script command for starting the Simulink model in case it is not already open, while *GTModel1Name* and *GTModel2Name* are the names of the GT Power models with the GT-Suite license and the xRT license, respectively. In this current version (v42) of the program there must be a separate model depending on which license to run, since it is currently not possible to

programmatically from Matlab change the license of a particular model. The *GTModel1SimTS* and *GTModel2SimTS* variables determine the maximum time step the model is allowed to take for when running with the GT-Suite license and the xRT license, respectively. The variables starting with "SubAss_" store the names of the GT-Power sub-assembly models that are used depending on which license is used and how the HRR should be determined, while *strImposedHRR* and *strSpeedGoatHRR* store names of the EngCylCombProfile templates to be used in the GT-Power model depending on the selected value of variable *VSS_BR*. With the *selectedTurbo* variable the operator can select both whether to use a 1-stage or a 2-stage TC, and at the same time which compressor and turbine maps the TC should have. With the *fuelMode* variable the user can select whether to run in DF or diesel mode when running a DF engine, or whether to run in diesel mode when running a pure diesel engine. In the current program version the user can namely select if to run a DF or a diesel engine, which is done by the code varying the compression ratio between the values stored in variables *CR_DF* and *CR_PureDSL* and using diesel specific HRR combustion profiles. The user can also select among five different fuels to be either port or directly injected using the *selectedPortFuel* and *selectedDirectFuel* variables; which injectors that in the end are actually injecting any fuel depends on the *fuelMode* variable but also on the *DIFuelMassPerc* variable, which determines the mass percentage of directly injected fuel when running in DF mode. Using the *IV_FlowAreaMult* and *EV_FlowAreaMult* variables the operator can scale the geometrical area of the intake and exhaust valves to e.g. correct for an incorrect air mass flow through the engine, while IMEP_Max and BMEP_Max tell the engine's IMEP and BMEP values when running at 100% load, used in some functions inside the GT-Power model.

On the last page of the selection menu is a variable called *EWGControl*, with which the operator can determine whether the PID controller of the EWG valve should target boost pressure or lambda, or if the EWG should be closed completely. The *p3_SteadyStateTol_Perc* is used for determining when the model has converged and hence is in a steady state when running the SS function, so that the script can stop the model already before reaching the maximum simulation cycles amount set in the *maxCycles* variable (but after

the minimum simulation cycles amount in the *minCycles* variable). The *timeUntilMeasActive* is a variable used for sensing the time in seconds that the measurement has been continuously active; this variable had to be introduced since the bit that the program reads over Modbus from the SCE control system was wrongly switching between 1 and 0 at occasions when it should have been only set to 0, and therefore the program must now sense a bit value of 1 for at least 10 seconds time to decide that the measurement of a test case has started. The *timeUntilMeasDone* tells how long the measurement point is in seconds, and since a measurement is decided to be 4 minutes, a value of 240 seconds is set for this variable. The *modbusParameterList* is a cell array containing the signals to be imposed onto the GT-Power model from the SCE over Modbus. Each row consists of the corresponding variable name in GT-Power, the signal's Modbus address, a scaling factor and an initial value for the parameter. The scaling factor is used for converting the signal value into the correct unit used in the GT-Power model. The variables *connectionType*, *deviceAddress* and *port* defines the variables needed for connecting to Modbus. The *exportFileName* points to a file needed by the script for fetching instantaneous crank-angle resolved data from the GT-Power results file to Matlab, while *instVarFileName* is the name of a text file to where this data is saved. The *ExcelMacro* variable stores the name of the Excel macro, version and function. This macro is used after the SS simulation to clean up the Excel output file, create headings and graphs etc. Finally, the *paramFileName* is a variable that refers to the name of a text file to where the GT-Power Case Setup parameter names and values are written, which are then imposed onto the model before it starts.

Many of the settings from the selection menu, like the variables to be imposed onto the GT-Power model from the *modbusParameterList* and several of the GT-Power specific variables, are more relevant for the SS than for the RT simulation. The reason for this is that when running the RT script, many of the parameter values are either fetched over Modbus at every time step, or then manually set by the operator in the control section of the Simulink model graphical user interface (GUI). When running SS on the other

hand, no parameters can be changed anymore once the model has started, which is why correct parameter initialization is crucial.

After the selection menu in the main script, the code uses the input from the selection menu when initializing the program. Error checking is done which will stop the program in case of an error. The code also checks the availability of Simulink: If Simulink is not open already, the program opens it. Finally, the program goes into a while-loop where, depending on the interactive selections of the operator, either RT or SS simulation is run. This is illustrated in the high level flowchart of the Matlab code logic in **Figure 33**.

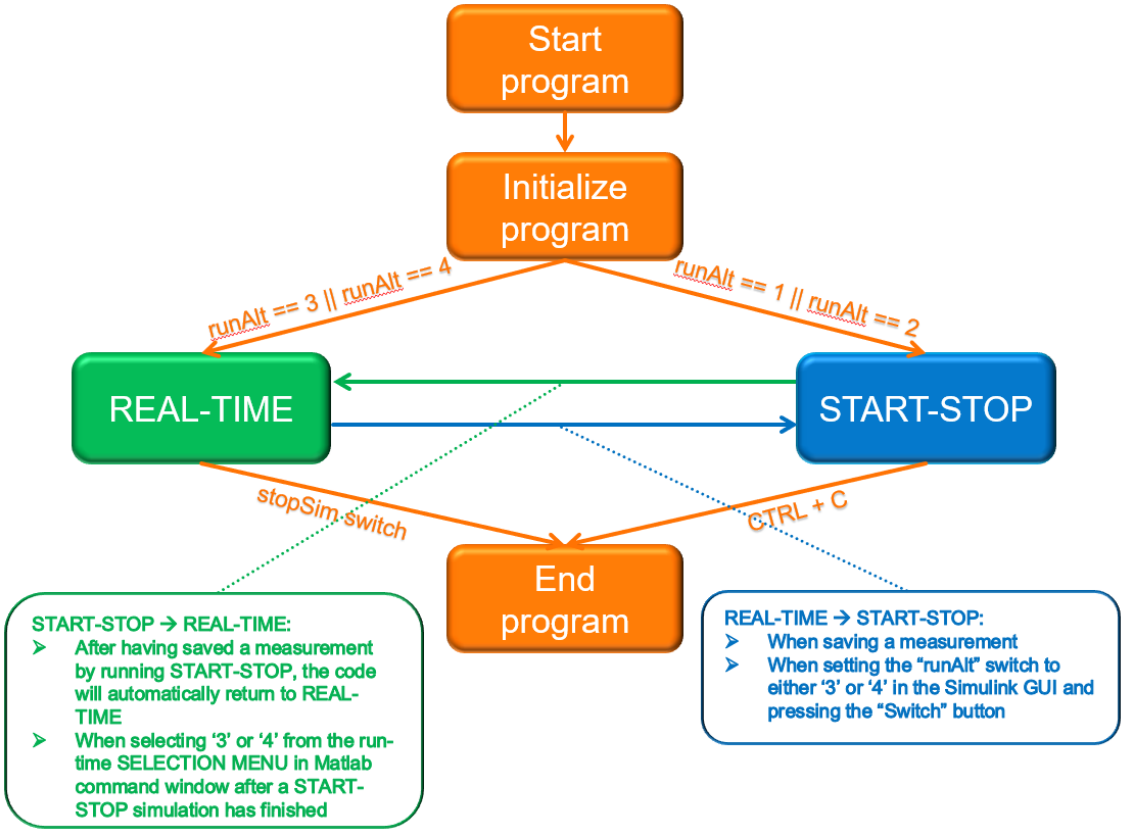

Figure 33. Flowchart of Matlab code logic.

4.2.3 Simulink model

The Simulink model used in the program is shown in **Figure 34**. In the top left is a purple coloured subsystem named *INPUT*, where the input signals are fetched over Modbus and sent to the GT-Power model. This block also contains other logic related to the controls. The GT-Power model itself is embedded as a “black box” inside of the *GTMODEL* block. In the cyan coloured subsystem *OUTPUT*, output is sent from the GT-Power model back to the Simulink model, to be used for visualization or calculation. In the *GAUGES* field are several gauges displaying the engine status, like engine speed, load, ambient conditions etc. The fuel icon changes colour depending on the fuel used.

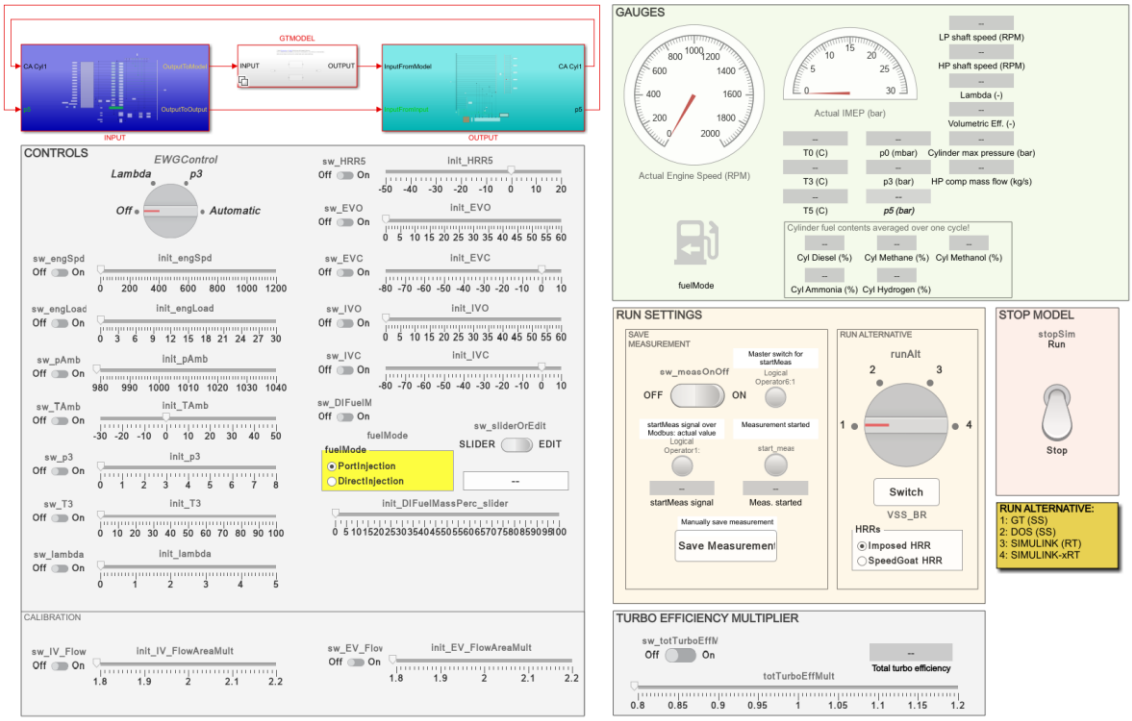

Figure 34. Simulink model.

The remaining fields are related to controlling and modifying settings for either the GT-Power model or the program in general. Using the buttons and sliders in the *CONTROLS* field on the left-hand side, the parameters imposed onto the model from Morphee can also be separately adjusted manually by the operator. The target for the EWG PID and the fuel mode can also be changed here. The TC efficiency multiplier can be adjusted in

the *TURBO EFFICIENCY MULTIPLIER* field, in case the operator wants to run a TC efficiency swing for the SCE where the TC efficiency is varied. In the *RUN SETTINGS*, the operator can choose whether the program should react to starting and ending measurements, or if the model should run continuously. In this area it can also be chosen to run in an SS mode instead of the RT mode by adjusting the *runAlt* rotary switch, where a value of '1' means to run an SS simulation via the GT-Suite GUI, '2' means to run the SS simulation via Matlab without any monitors displaying, '3' means to run an RT simulation using the GT-Suite license, and '4' means to run an RT simulation using the xRT license. To end the program, the switch in the *STOP MODEL* field is set to "Stop".

4.3 Using GT-Power-in-the-Loop

4.3.1 Real-Time simulation

When starting the RT simulation through the *MAIN.m* script, the following output is printed to the Matlab command window (depending on the chosen initialization values):

```
************** AUTOMATED RUN SCRIPT STARTED **************

******************REAL-TIME SIMULATION*******************
          Real-Time simulation script started.
*********************************************************

Warning: Could not create modbus object!
> In REALTIME (line 60)
In MAIN (line 349)

* MATLAB IMPOSED PARAMETERS *
FLUIDSS=OFF
MonitorSelection=<W8L25DF_FRM_Combined_xRT_NoMonitors-
V2023.gtsub>
HRRSelection=<W8L25DF_FRM_Combined_xRT_ImposedHRR-
V2023.gtsub>
TurboSelection=<W25_FRM_Turbo2s-V2023.gtsub>
LPCompMap=<LPR5_CF18A_C27-F18r01.cmp>
LPTurbMap=<LPR5_TD222_TD34-146r01.trb>
HPCompMap=<HSR4_CB14A_C01-B01s01.cmp>
HPTurbMap=<HSR4_TB190_TB28-101r01.trb>
FixedTimeStepSize=0.2
selectedPortFuel=diesel-1fo
```

```
selectedDirectFuel=diesel-lfo
AFRStoich_FuelPI=14.5
AFRStoich_FuelDI=14.5
DIFuelMassPerc=5
p3_SteadyStateTol_Perc=0.05
minCycles=10
maxCycles=200
scriptVersion=2
saveMeas=0
runAlt=4
fuelMode=1
EWG_PID_Type=2
CR=11.7
IV_FlowAreaMult=2
EV_FlowAreaMult=2
RPM=1000
IMEP=21
Ambient_p0=1014
Ambient_t0=-5
target_p3=4
target_T3=60
target_lambda=1.5
EVO_TIMING=36
EVC_TIMING=-5
IVO_TIMING=12
IVC_TIMING=-45
HRR5=0
Warning: Cannot connect to Modbus! Starting real-time simu-
lation with user imposed values.
> In REALTIME (line 82)
In MAIN (line 349)

* STARTING SIMULATION! *
INFO: Successfully loaded C:\Program Files
(x86)\GTI\v2023\simulink\GTPOWERxRT\libgtpowerxrt_x64.dll.
INFO: Copied C:\Program Files (x86)\GTI\v2023\sim-
ulink\GTPOWERxRT\libgtpowerxrt_x64.dll to C:\Us-
ers\BSM007\AppData\Local\Temp\GTxRT\U5vHid.dll.
INFO: Successfully loaded C:\Users\BSM007\AppData\Lo-
cal\Temp\GTxRT\U5vHid.dll.
INFO: Current directory C:\PROJECTS_CDRIVE\PROJECT-GTP_As-
sisted_Controls_and_Evaluation\SCE25\GTP-Simulink_Cou-
pling_Combined\v42_MSc.
```

In the particular version of the program from which this Matlab command window output was copied, some of the variables and their values under '* MATLAB IMPOSED PARAMETERS *' were removed due to confidentiality. The warning about failing to connect to Modbus is a user coded warning that appears in case not running the script on the standalone computer connected to the SCE control system, but instead if running the

script on a computer without connection. The program will then run the RT simulation using the initialized parameter values until manually adjusted by the user, since it is not connected to the SCE control system.

After this the Simulink model starts running, and therefore also the GT-Power model (**Figure 35**). In older versions of the program monitors from GT-Power were also showing during runtime like in this figure, but in newer versions Scope monitor components in Simulink were taken into use to display different parameters. By using the sliders in the *CONTROLS* field, the parameter values can also be adjusted manually by the operator. In **Figure 36** a), the program still uses the same values as it was initialized with, but in **Figure 36** b) however, the intake valve closing (IVC) slider was changed from -70 to -30 °CA after bottom dead center (aBDC) so that the intake valve (IV) represented by the red curve closes at a later time, and the program updates this changed IVC value in the GT-Power model in real-time. It should be noted that the output printed to Matlab presented here are displayed when running version 42 of the program, while **Figure 35** and **Figure 36** are taken from when running version 27. The reason for this is that, as mentioned in the older versions of the program, it was possible to also show monitors from GT-Power while running the model from Simulink, which means that i.a. the compressor maps could be shown in **Figure 35**, and how the valve timings in the GT-Power model react to user input via Simulink could be illustrated in **Figure 36**. In this case it was hence more illustrative to look at the runtime monitors from version 27, to better explain the functionality.

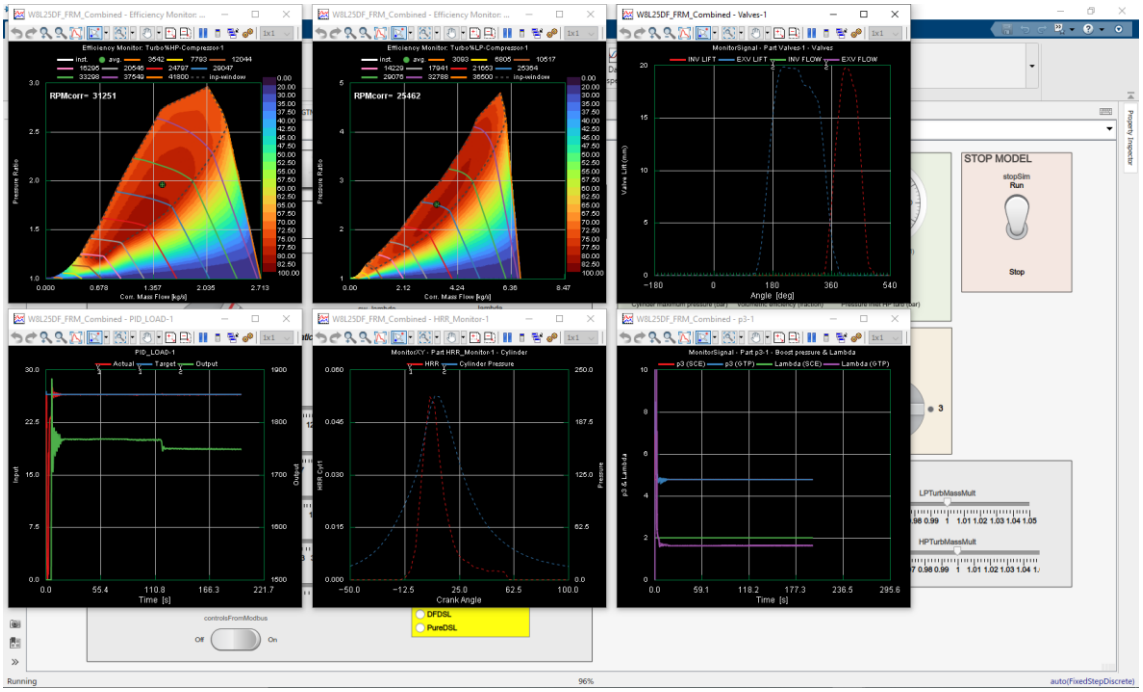

Figure 35. Running Real-Time simulation. Monitors from GT-Power visible.

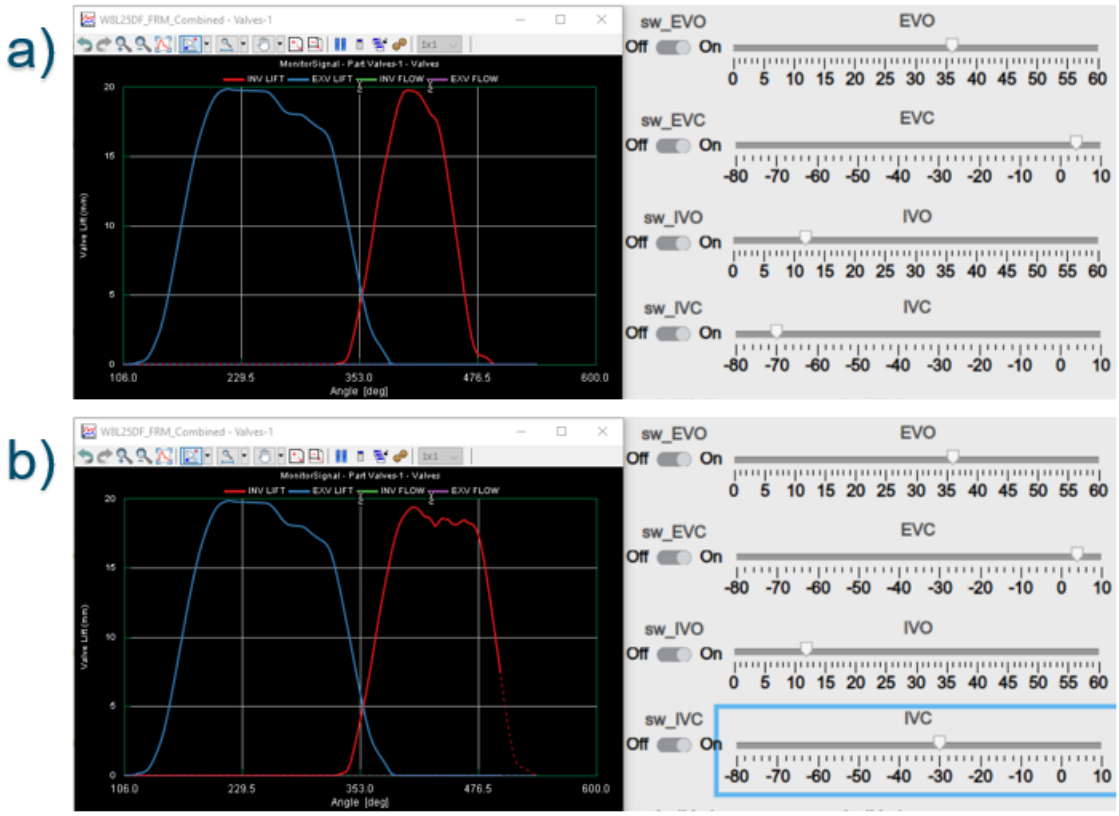

Figure 36. Adjusting the intake valve closing time while running the program.

Figure 37 continues to show how the model responds to inputs from the operator, this time through changes in the *EWGControl* rotary switch. In the figure, the red and blue lines above are p_3 from the SCE and from the MCE, while the green and purple lines below are λ from the SCE and from the MCE, respectively. The signals from the SCE are targets, which the virtual MCE tries to follow at every time step. For illustration purposes, the SCE target signals are being manually adjusted in the figure. In the beginning of the graph the *EWGControl* rotary switch is set to "p3", meaning that the EWG PID controller in the MCE model is set to regulate p_3 by adjusting the EWG valve diameter. At time instance "1." the operator turned the *EWGControl* rotary switch to "Lambda", so that the EWG PID controller instead regulates λ by adjusting the EWG valve diameter. The purple and green lines now show that the MCE λ follows the target SCE λ. At time instance "2." the operator turned the *EWGControl* rotary switch back to "p3", and again the model starts following the target p_3 from the SCE. At time instance "3." the target p_3 is dropped to a new level, and the MCE follows closely, depending on how well tuned the EWG PID controller is.

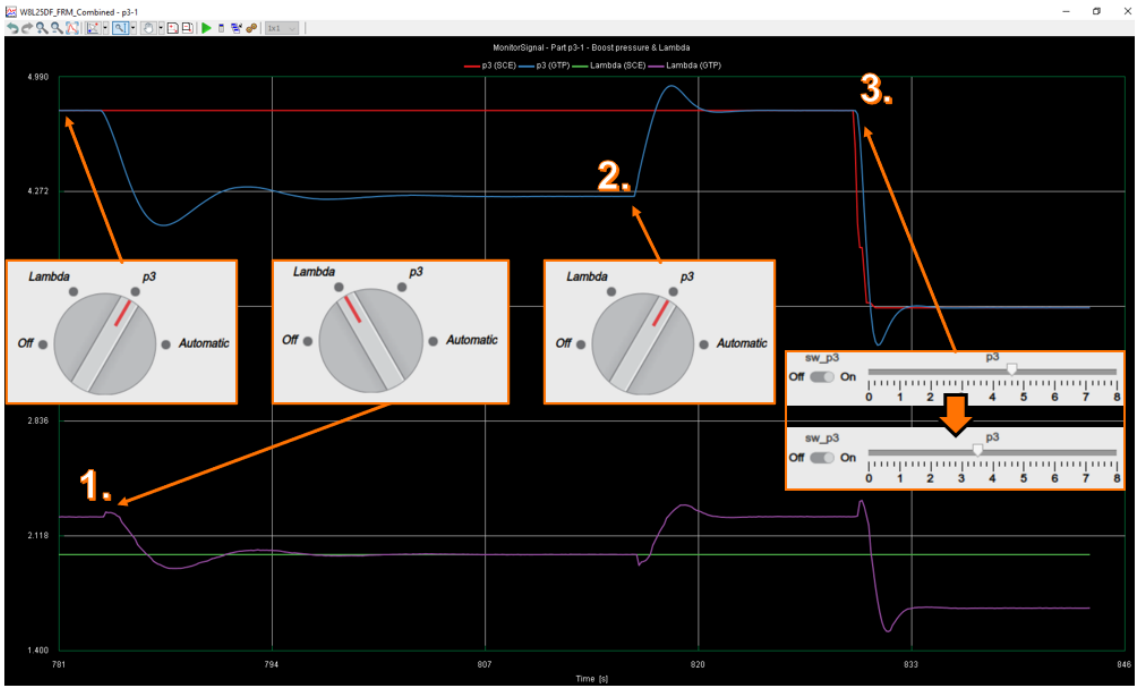

Figure 37. Showcase of running the virtual and real engines in parallel.

Figure 38 illustrates the procedure for taking a measurement point. If the *sw_measOnOff* switch is set to "OFF" like in **Figure 38** a), it should be set to "ON" like in **Figure 38** b), whereby a blue light beside the switch lights up, indicating that the program now will sense through Modbus when the Start of measurement-button is pressed in the SCE control system. **Figure 38** c) shows the appearance of the *RUN SETTINGS* field when the Start of measurement-bit has had an active value of '1' for 18.8 seconds time, which is longer than the *timeUntilMeasActive* time that in this case was 10 s, and a yellow light shines above the time field. As can be seen, a value of 9.3 s is shown in the *Meas. Started* field on the right side with a green light shining above it, meaning that the measurement has been active during a time of 9.3 s. Due to bad resolution when the snapshot was taken the *Meas. Started* field shows a value of 9.3 s, although it should show a value of 8.8 s (*timeUntilMeasActive* – 10). When this value has ticked up to 240 s, the RT simulation stops and an SS "save measurement" simulation starts. The button *Save Measurement* can also be pressed, which will override this procedure and instantly start the SS simulation.

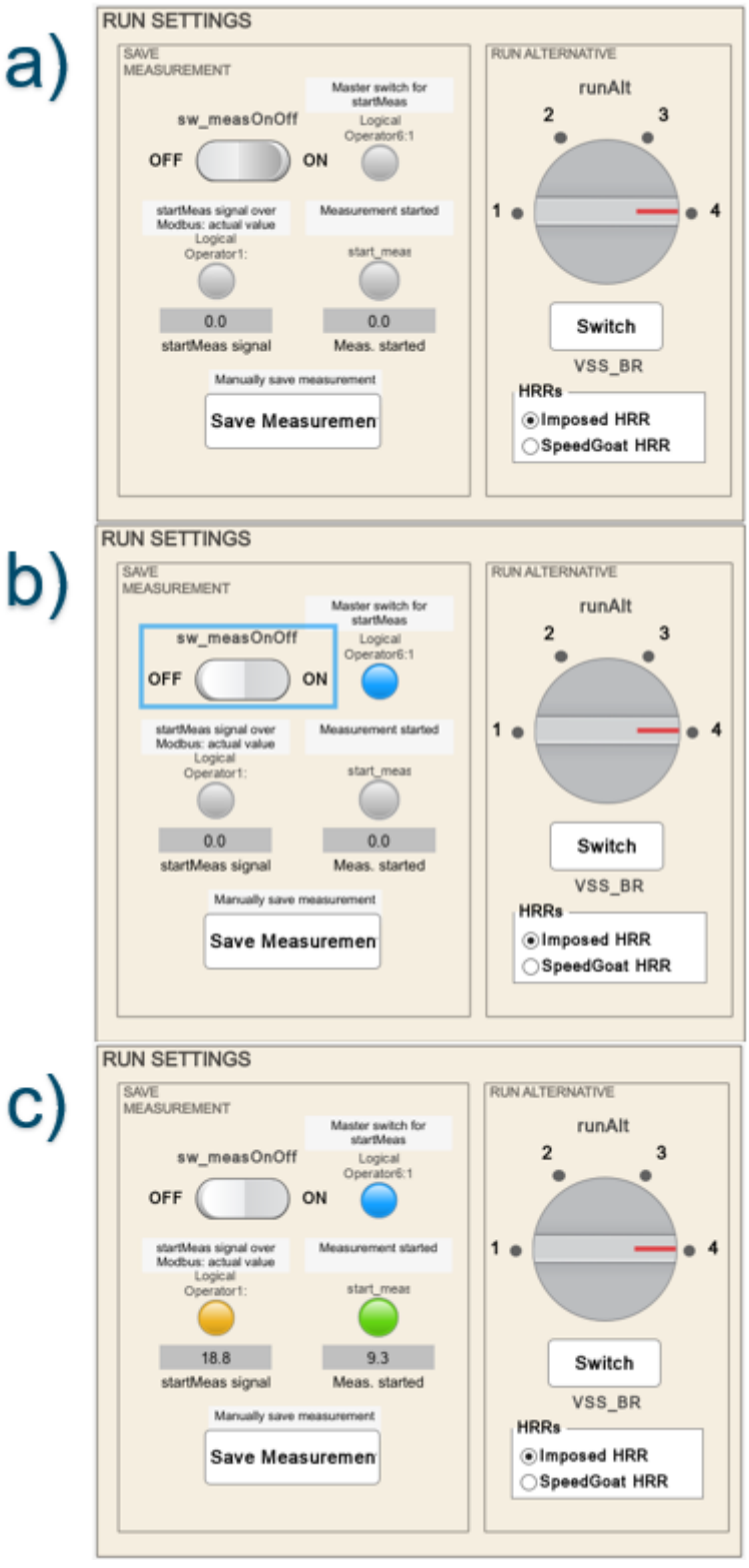

Figure 38. Initiating the ”Save measurement” procedure.

In the INPUT subsystem of **Figure 34** there is a Matlab Function block that reads variable values from the SCE control system over Modbus, which after this are imposed onto the GT-Power model. This block also handles the sending of the simulated p_5 back to the SCE, done by the following rows of code:

```
%% Send back pressure to Morphee
digits = 2;
trb_pia_modbus = fix(round(trb_pia * 10^digits));
% Write to Morphee over Modbus
write(m,'holdingregs',1200,trb_pia_modbus,'int16');
```

This code first rounds the simulated p_5 to a suitable amount of digits, after which it is sent to the SCE using the *write()* function in Matlab.

4.3.2 Start-Stop simulation

When the SS simulation begins, the following output is printed to the Matlab command window (depending on the chosen initialization values):

```
*****************START-STOP SIMULATION*********************
      Start-Stop simulation script started.
      Script version selected: Start-Stop
      Run alternative selected: GT
      Run type selected: Save measurement
***********************************************************

Waiting for Autoreport file 1_Small_SCE_14044_Performance-
File.xlsm to be created in folder
C:\PROJECTS_CDRIVE\PROJECT-GTP_Assisted_Controls_and_Evalu-
ation\SCE25\GTP-Simulink_Coupling_Combined\v42_MSc\AUTORE-
PORT\...
Waiting for Dewesoft file 13780.dxd to be created in folder
C:\PROJECTS_CDRIVE\PROJECT-GTP_Assisted_Controls_and_Evalu-
ation\SCE25\GTP-Simulink_Coupling_Com-
bined\v42_MSc\DEWESOFT\...
Press Enter to pause the script and select whether to fetch
the data from Indicom or Dewesoft.
Newest Dewesoft file found!
Newest Dewesoft file loaded!

* MATLAB IMPOSED PARAMETERS *
FLUIDSS=ON
```

```
MonitorSelection=<W8L25DF_FRM_Combined_Monitors-
V2023.gtsub>
HRRSelection=<W8L25DF_FRM_Combined_ImposedHRR-V2023.gtsub>
TurboSelection=<W25_FRM_Turbo2s-V2023.gtsub>
LPCompMap=<LPR5_CF18A_C27-F18r01.cmp>
LPTurbMap=<LPR5_TD222_TD34-146r01.trb>
HPCompMap=<HSR4_CB14A_C01-B01s01.cmp>
HPTurbMap=<HSR4_TB190_TB28-101r01.trb>
FixedTimeStepSize=ign
selectedPortFuel=diesel-lfo
selectedDirectFuel=diesel-lfo
AFRStoich_FuelPI=14.5
AFRStoich_FuelDI=14.5
DIFuelMassPerc=5
p3_SteadyStateTol_Perc=0.05
minCycles=10
maxCycles=200
scriptVersion=1
saveMeas=1
runAlt=1
fuelMode=1
EWG_PID_Type=2
CR=11.7
IV_FlowAreaMult=2
EV_FlowAreaMult=2
RPM=900.21
IMEP=20.96
Ambient_p0=1013.4
Ambient_t0=70.67
target_p3=2.5
target_T3=65.04
target_lambda=1.53
EVO_TIMING=33.53
EVC_TIMING=-2.32
IVO_TIMING=10.13
IVC_TIMING=-50.68

* STARTING SIMULATION! *

GTcom: connected to RabbitMQ server
GTCom::notifyServiceReady for SolverUI
DEBUG - Request::solverStateChanged() state: RUNNING
```

From the code section above it can be noted that Dewesoft provides the simulation with the input values (“Newest Dewesoft file loaded!”), the run type is set to “Save measurement”, and the run alternative is “GT”, meaning that the SS simulation will be run via GT-Post where also the GT monitors are visible. Before starting the simulation, the program will first search for the 300 cycle averaged variable values to be fetched from the Autoreport file, which has been created for the just measured test point by another script.

When the Autoreport file is found, the program searches for the Dewesoft file, from which it will fetch the 300 cycle averaged HRR curve, also to be imposed onto the model. **Figure 39** illustrates the “Save measurement” SS simulation with the purpose of running until steady-state, after which the simulation stops and the output creation stage starts.

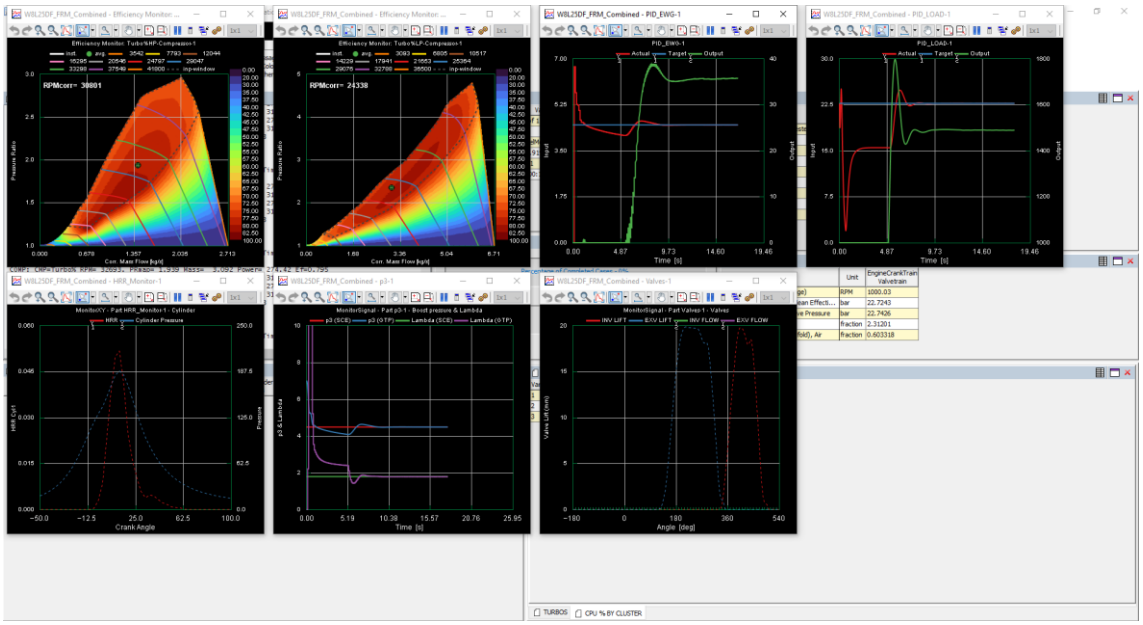

Figure 39. Start-Stop simulation at steady-state.

The value of the *runAlt* rotary switch when the “Save measurement” action has started determines the SS simulation’s run alternative. In the example in **Figure 38**, the switch was set to ‘4’ when this action started, while as mentioned earlier a value of ‘1’ or ‘2’ should in theory be selected when desiring to run an SS simulation. The program is coded in such a way that the run alternative “GT” is default, meaning that in case a rotary switch value other than ‘2’ (run the SS simulation via Matlab without any monitors displaying) is selected, the program will default to running via GT-Post. If the rotary switch on the other hand would have been set to a value of ‘2’ at the end of the RT simulation, the SS simulation would instead have been run through Matlab. An example of running through Matlab can be seen in **Figure 40**. The benefit of this run alternative is its speed, while the drawback is the lack of monitors, making it more difficult to notice possible faults in the simulation.

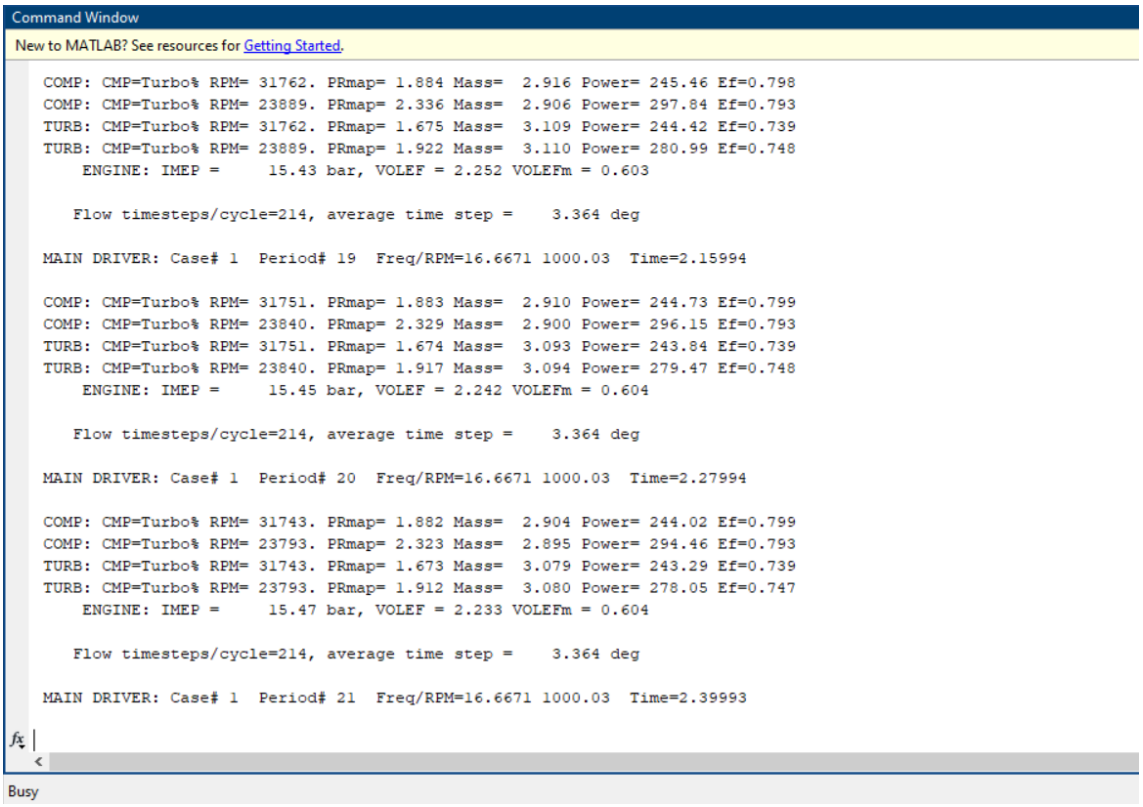

Figure 40. Running Start-Stop through Matlab.

4.3.3 Output

When the SS simulation has reached steady-state and stopped, the following is printed in the Matlab command window:

```
* FORMATTING OUTPUT DATA... *
CompileCommand: exclude java/swing/plaf/basic/BasicSplit-
PaneUI$BasicHorizontalLayoutManager.layoutContainer
INFO: Acquired 1 GTGUIB license in 1000 ms [Mon Apr 08
08:14:41 EEST 2024]
MSG     :Loading Steering File from [C:\PRO-
JECTS_CDRIVE\PROJECT-GTP_Assisted_Controls_and_Evalua-
tion\SCE25\GTP-Simulink_Coupling_Combined\v42_MSc\ex-
portdata.exp]
MSG     :Writing file [C:\PROJECTS_CDRIVE\PROJECT-GTP_As-
sisted_Controls_and_Evaluation\SCE25\GTP-Simulink_Cou-
pling_Combined\v42_MSc\GT_InstVar.txt]
```

```
WARNING: Process 12628 with title OpenJDK Platform binary
has an open handle to
C:\Temp\gtidata\dbdir\BSM007\FIL40294W\v2024\glx_d256ffa56c
ac42ad91130caec5152360\W8L25DF_FRM_Combined-V2023.db, will
try [Mon Apr 08 08:14:43 EEST 2024]
SEVERE: Mon Apr 08 08:14:43 EEST 2024: Exception class
[class com.gtisoft.db.system.g] caught
NextGen DB Engine: Database
'C:\Temp\gtidata\dbdir\BSM007\FIL40294W\v2024\glx_d256ffa56
cac42ad91130caec5152360\W8L25DF_FRM_Combined-V2023.db'
could not be deleted.    Trace (10 deep)
          [0] - com.gtisoft.db.e.c.f(c.java:437)
          [1] - com.gtisoft.result-
datadriver.b.e.k(e.java:684)
          [2] - com.gtisoft.resultdatadriver.aE.lambda$de-
leteDatabaseBruteForce$3(aE.java:418)
          [3] - java.base/java.lang.Thread.run(Unknown
Source) [Mon Apr 08 08:14:43 EEST 2024]
To create GTM Service: :0.231
MSG     :   File [C:\PROJECTS_CDRIVE\PROJECT-GTP_As-
sisted_Controls_and_Evaluation\SCE25\GTP-Simulink_Cou-
pling_Combined\v42_MSc\GT_InstVar.txt] successfully writ-
ten! (3.865) s
Export Process Complete!  (4.498) s

* CREATING EXCEL OUTPUT FILE... *

* RUNNING EXCEL OUTPUT MACRO... *

* MOVING OUTPUT DATA... *

Warning: Test point folder with test number "14045" already
exists, appending suffix to name.
> In moveOutputData (line 14)
In STARTSTOP (line 114)
In MAIN (line 346)
.dat FILE COPIED TO OUTPUT FOLDER
.glx FILE COPIED TO OUTPUT FOLDER
.out FILE COPIED TO OUTPUT FOLDER
.rlt FILE COPIED TO OUTPUT FOLDER
.xlsx FILE COPIED TO OUTPUT FOLDER

******************START-STOP SIMULATION******************
      Start-Stop simulation script ended.
      Start-Stop execution time: 79.2663 s.
      The MATLAB script will be paused for 5 s.
      Press Enter to show simulation loop selections.
      If no input is given, MATLAB will perform a new
      simulation loop, based on previous run settings:
      Script version selected: Real-Time
      Run alternative selected: Simulink (xRT)
      Press CTRL + C to stop the script.
*********************************************************
```

The program starts by loading a steering file with extension *.exp*, which tells what type of vector data from the last simulation cycle to fetch from the FRM (the model); this can be data like i.a. cylinder pressure or valve mass flows. This data is fetched from the FRM and inserted into a text file, here named *GT_InstVar.txt*. The program also fetches cycle-averaged data from a file with extension *.rlt*. After this, the program creates an Excel file called *GTDATA.xlsx*, to which it copies this data. The program then starts an external Excel macro, whose purpose is to format the GTDATA file by i.a. changing fonts and colors of text/data, inserting headers, creating graphs etc. **Figure 41** shows one of the graphs from the GTDATA file of intake and exhaust valve lifts and mass flows, automatically created by the Excel macro called by the program.

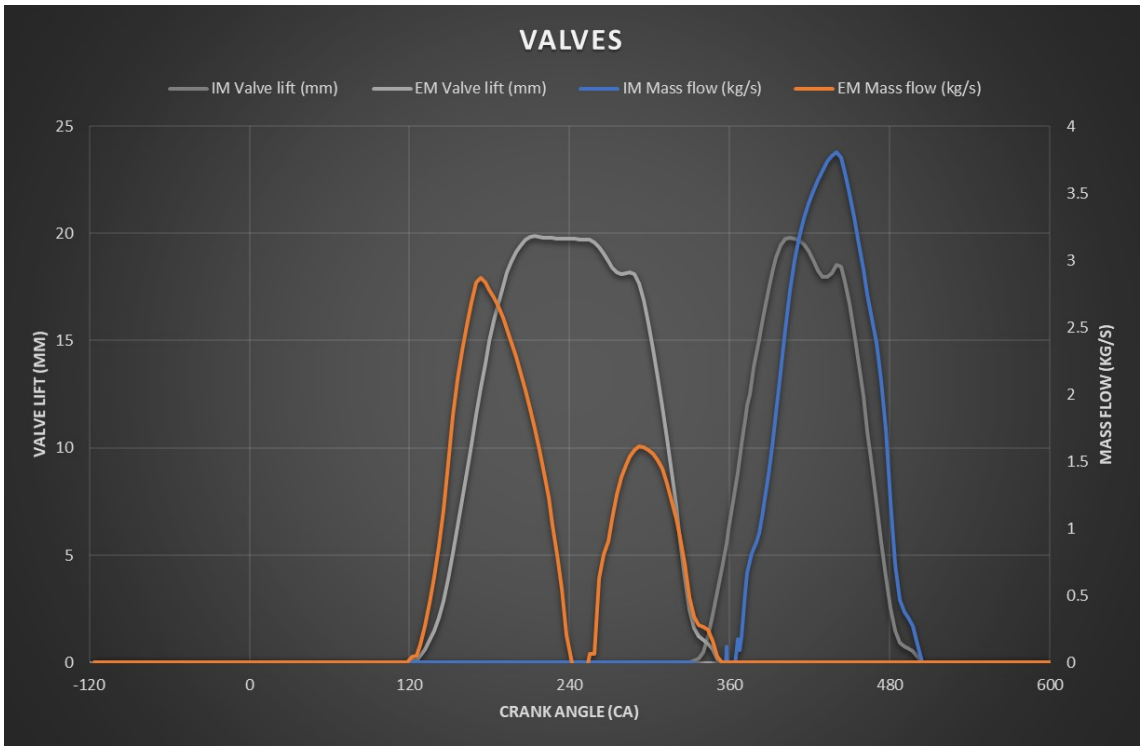

Figure 41. Intake and exhaust valve lifts and mass flows.

This output Excel file, as well as other output files automatically generated by GT-Power, are then moved to a folder with a test/measurement point specific name. The program searches a network folder, to which measurement data is moved after completing a test point, for the ID of the most recent measurement test point. This ID is then copied and

added to the name of the folder, to which the simulation output files are moved. In this code output example, the program gives a warning that the test point folder with ID ‘14045’ already exists, and therefore it appends a suffix to the name of the folder, e.g. “_2”. **Figure 42** shows the content of a simulation output folder from a measurement. The *.dat* file is the GT-Power model in text format, which can be used for re-running the model. The *.glx* file is an output file created by GT-Power. The *.out* file shows the proceedings of the simulation in the same format as in **Figure 40**. The *.rlt* file consists of cycle-averaged results from the simulation. Finally, there is also the GTDATA file.

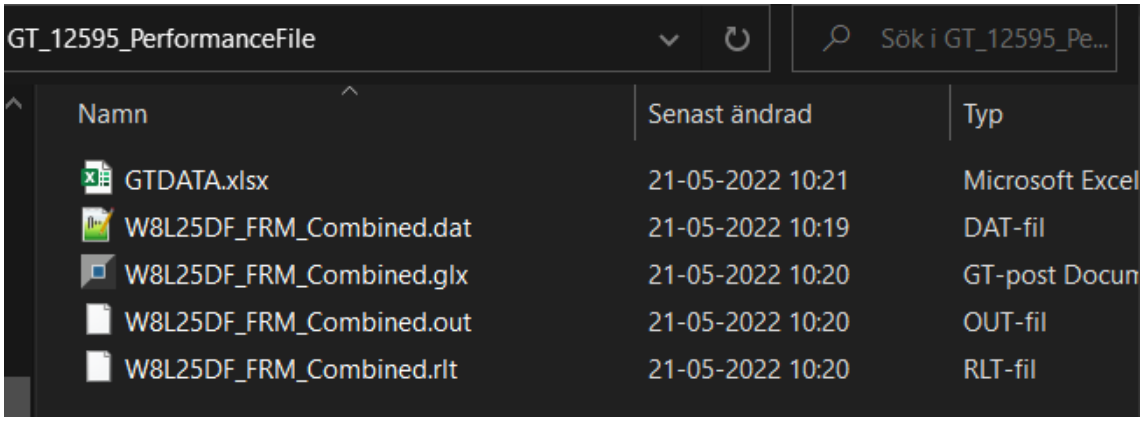

Figure 42. Program output folder content.

When the simulation output files have been moved to the test point specific folder, the program informs the operator that the SS simulation has ended, and how long the execution time was. By pressing the Enter button inside a time interval of a couple of seconds, another menu appears from where the user can select to continue running in SS mode, and in that case, which run alternative to use. Otherwise, the program defaults to restarting a new RT simulation which, as always, is run through Simulink. By pressing “CTRL + C”, the program can be halted.

4.4 Fast-running model validation

In contrast to the HiL setup developed by Pirker, G., Mayr, P., Kranawetter, K., Bauer, R. et al. (2021), where real-time capable, physical 0D models were developed and coupled with Simulink and where several signals is fed back to the SCE, only the exhaust back pressure (p_5) is currently fed back in this setup. A comparative study, where the p_5 calculated in this setup would have been compared both with the p_5 calculated by the SCE's TC model and with the experimental p_5 of an actual MCE, would have been interesting to undertake to see which model is the most accurate. A study like this could however not be done due to a couple of reasons. To accomplish this, one would firstly need e.g. a load swing that has been run with the same settings on both a laboratory W8L25 MCE and the SCE, so that these same settings also could have been imposed on the virtual MCE model, and then the exhaust pressure could have been compared for the MCE, SCE and the virtual MCE. It was stated that such data also existed, but in an attempt to search for these, only the SCE data could be found. Into this SCE data file an expert at Wärtsilä had copied the measured T5 and the calculated TC efficiency, but this was the only data at the author's disposal from the MCE load swing. It was however assumed that the load swings had been run with exactly the same settings and therefore an attempt to run the simulations was made, but since the information about what kind of TC that had been used in the MCE test run also was missing, a 2-stage TC model with what was thought to be somewhat correct compressor and turbine maps was used. This TC was however unable to build up the simulated p_3 to the target, meaning that either the selected TC was wrong, or then some different settings between the SCE and MCE had been used after all. Secondly, it became known that the tuning parameters for the SCE's TC model had not been calibrated during this run; instead the operators had inserted a target p_5 value directly and then the SCE controls adjusted a back pressure valve to match this target. Since the TC model had not been calibrated, a fair comparison could not however have been made. What could have been done though is to have run the GTPiL system in parallel with the SCE as it is supposed to do, and then the p_5 values would have been compared live, but this would only have given a relative comparison between the TC model and the GTPiL system. It was instead deemed more interesting to compare the GTPiL

system against measured MCE data directly, not only regarding p_5 but also other quantities. Another MCE load swing was found including information about the used TC, this data was imposed onto the GTPiL system and the resulting comparison is illustrated by **Figure 43**. This figure shows the same graphs as **Figure 31**, with the difference being that **Figure 43** shows the mean peak firing pressure during combustion in the middle right graph, while **Figure 31** shows the models' simulation speed. The X-axis range is slightly different since the lowest case in the measured load swing was at 25% load and the highest at 110% load. The minimum and maximum value for the Y-axis of the top left graph have both been increased by a value of 5 bar, since in this figure IMEP is compared instead of BMEP, but otherwise the Y-axis is the same. The measured data is illustrated by the black curve, while the FRM is again shown in blue color, but as can be noted there are now two curves representing the FRM: One solid curve named "FRM_RT", and a dashed curve named "FRM_SS". As the names imply, the FRM_RT represents the performance of the model when it is run in an RT mode with the semi-predictive tabulated combustion, while the FRM_SS shows the performance of the model when it is run in an SS mode with the 300 cycle averaged measured HRR curve as imposed.

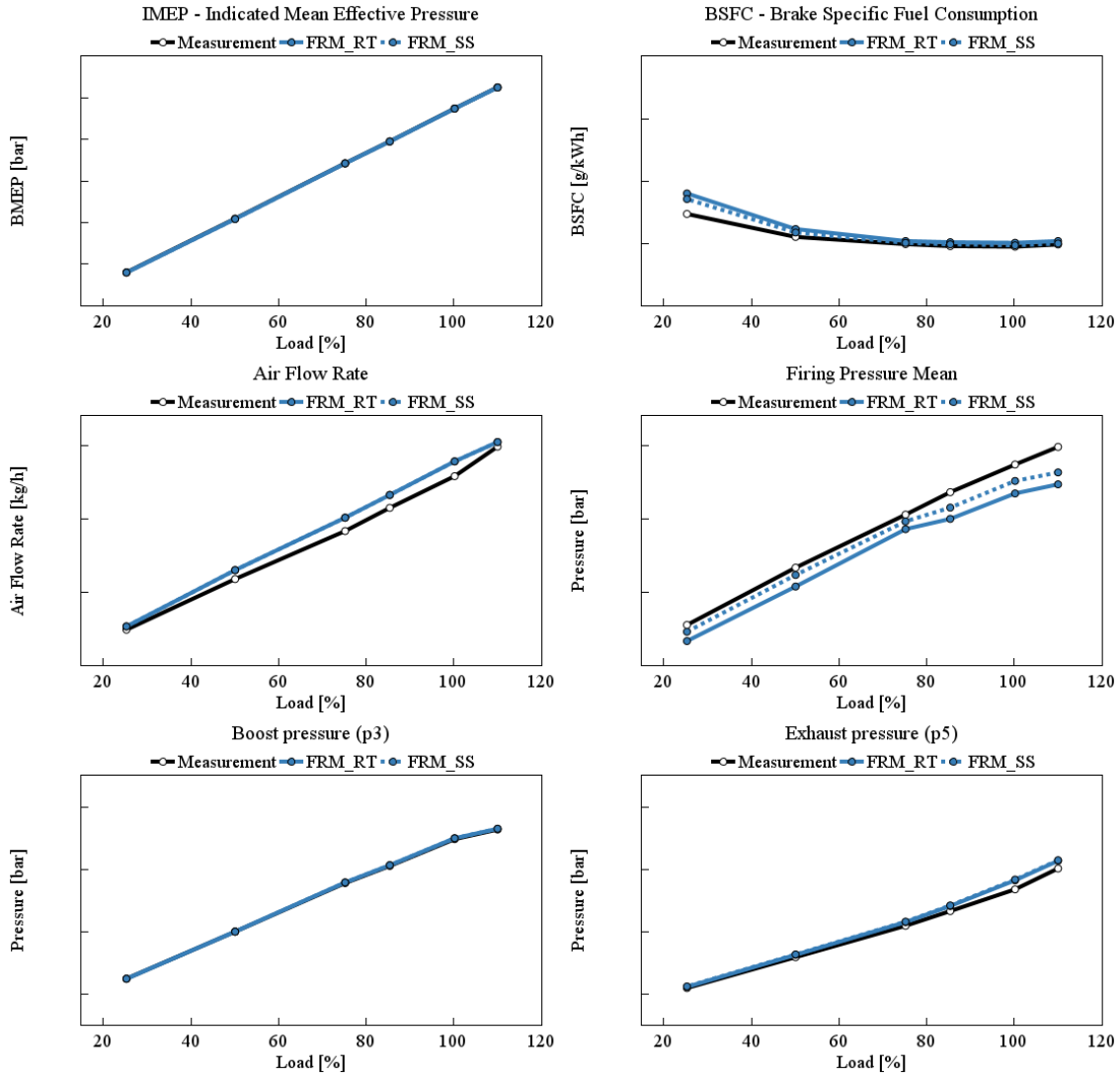

Figure 43. Cycle-averaged parameters comparison between FRM and measurement.

The simulated IMEP in the top left graph matches the measured IMEP well due to the PID controller in the model adjusting the fuel mass to target the measured IMEP. Similarly, the EWG PID controller adjusted the EWG valve opening angle to target the boost pressure p_3. The higher BSFC in the simulations can again partly be attributed to an inaccurate friction model, but also i.a. to the lower mean peak firing pressure during combustion and to the slightly higher exhaust pressure p_5. Even though the simulated air flow rate is larger than the measured air flow rate, the simulated mean firing pressure is lower than in the measurement, about 25 bar lower in the FRM_RT case. If the flow parameters in the model would be tuned to get a decrease in air flow through the engine to better

match the experiment, the simulated mean firing pressure would decrease further. There are several reasons why the simulated mean firing pressure is underpredicted compared to measurement. Firstly, since a heat release rate (HRR) is imposed as combustion object in the model instead of a BR, the simulated combustion is slightly lagging the actual combustion (more on this in chapter 4.5 Modelling challenges). The shape of the BR could also differ slightly from the HRR. Secondly, a minor offset added to the compression ratio in the simulation might prove necessary. Thirdly, explaining the difference between the FRM_RT and the FRM_SS data series, the semi-predictive tabulated combustion in the FRM_RT might be calculating the wrong start of ignition (SOI) timing for the combustion, and the shape of the imposed HRR could be considerably different from the actual combustion. The FRM_SS on the other hand that directly imposes the measured HRR curve, and because of this also obtains the correct SOI timing, predicts a higher mean firing pressure that is closer to the measurement throughout the load range. To know the real reason behind the difference in cylinder pressure, a three-pressure analysis (TPA) should be performed in GT-Power, to match the simulated and measured cylinder pressures and thereby be able to compare the subsequent simulated BR with the imposed heat release rate(s). This TPA task is however out of the scope of this thesis, since according to the first objective it was stated that the FRM should have a minor deviation in model accuracy compared to its preceding detailed model, and not to an arbitrary experimental load swing at this stage. This TPA is therefore something to be carried out in the future. The p_5 value is seemingly not sensitive to the differences among FRM_SS and FRM_RT, and is ~0.28 bar higher at the highest load point in the simulations compared to measurement, which is inside of a 5% error margin.

4.5 Modelling challenges

There are a couple of challenges with the GTPiL system way of modelling the p_5 that will be mentioned in this chapter. Firstly, the Matlab TC model uses one Matlab license during runtime, while the GTPiL system on the other hand uses one Matlab license, one Simulink license and one GT-Power license. Individual Matlab and Simulink licenses were

however bought for the stand-alone computer, so that these software are always accessible for running this program. The GT-Power license on the other hand needs to be reserved from a common pool of licenses, and might therefore not be available when needed. The GTPiL system is therefore a more expensive approach.

Another challenge relates to using the HRR profile for defining the combustion instead of the BR. According to Gamma Technologies (2022), the BR can be described as the instantaneous rate of fuel consumption within the cylinder combustion process, i.e. the rate at which fuel and air molecules are transferred from the unburned zone to the burned zone, where they begin to participate in the chemical reactions. Contrarily, the HRR is the instantaneous rate at which the energy stored in the fuel molecules is released as thermal energy in the cylinder. The difference between BR and HRR is that some of the energy in the fuel will not be released until a later stage if any intermediate products of combustion are formed before the complete combustion takes place, since the fuel-air mixture entering the equilibrium equations does not break down immediately into its final products of combustion. The consequence of this event is that the HRR will lag the BR. The energy released is also dependent on equivalence ratio and temperature which will vary inside of the cylinder if the mixture is not completely homogeneous, causing a local difference between HRR and BR (Gamma Technologies, 2022). Since only the HRR is calculated and available as outcome from the experiments at Wärtsilä, HRR is used instead of BR for defining the combustion in the SS simulations, which introduces an error into the process. HRR profiles are also used in the semi-predictive tabulated combustion model used in the RT simulations, although a TPA could be run to replace these with generic BRs instead.

The semi-predictive tabulated combustion model used in the RT simulations is map-based, i.e. derived from a pre-defined lookup table. The lookup table is three-dimensional, where the shape of the HRR depends on which combustion modality (type of fuel and injection) is used, as well as the current load. The dispute with this approach is that

the combustion is non-physical, i.e. not a reflection of the actual ongoing physical phenomena in the cylinder. Having a tabulated representation of the combustion specify the actual in-cylinder combustion in a realistic manner would require many more dimensions with better precision than what is currently used, since the combustion depends on many different factors not accounted for here. The combustion phasing, i.e. the °CA at which 50% of the heat has been released, depends to a large extent of the start of combustion (SOC). An optimized combustion phasing is crucial for peak cylinder pressure, engine efficiency, emissions and other engine quantities. In GT-Power the SOC can be set to any value, at which the combustion will start and advance according to the defined combustion profile array, which in this case is an HRR. The combustion profile must however be manipulated before being used in the model, both by the program when imposing the measured 300 cycle averaged HRR during the initialization phase of the SS simulation, and by the user when creating the semi-predictive tabulated combustion model to be used in the RT simulations. An arbitrary filtered but unmodified measured HRR profile can be seen in **Figure 44**, with a SOC at about -10 °CA after top dead center (aTDC). This array must be cut so that it starts no earlier than -180 °CA, and negative values must also instead be set to 0 since they are not allowed in the combustion profile array template. After these modifications to the array however, it can be imposed as combustion profile for the SS simulation.

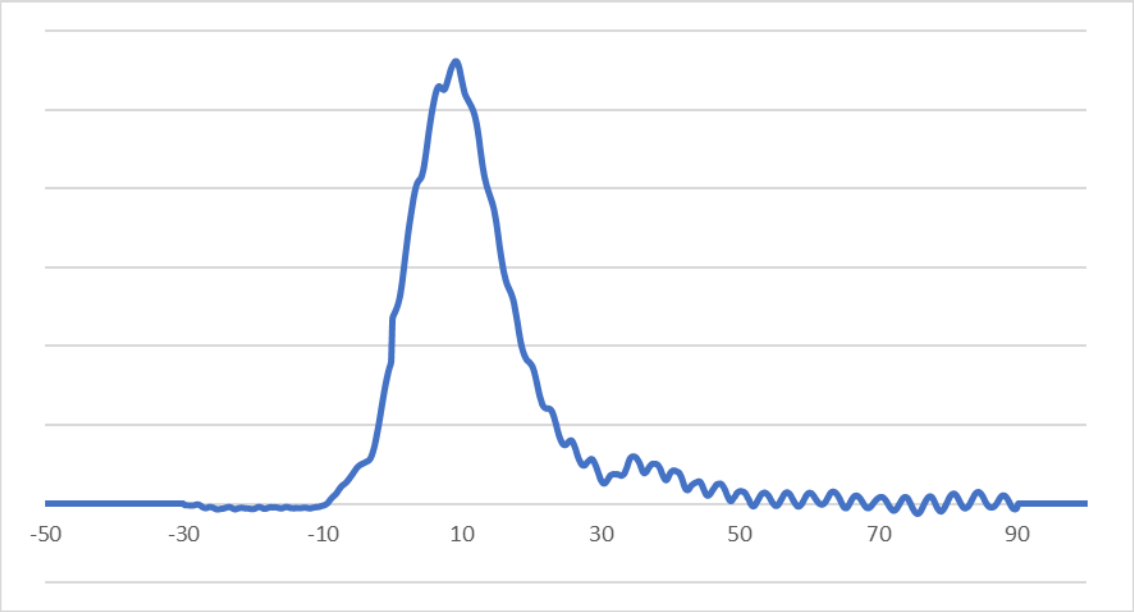

Figure 44. A filtered but unmodified measured HRR profile.

The creation of a tabulated combustion model to be used in the RT simulations requires more work however, because if the combustion profile array in **Figure 44** would be used here the model would not function as intended. This example profile namely starts at about -50 °CA aTDC, but actual combustion does not occur until about -10 °CA aTDC, which is 40 °CA later. This means that if the SOC value, or in practice the °CA at which 5% of the heat has been released (CA5, more about this in the next paragraph) that is either being sent over Modbus or manually adjusted by the operator during the simulation, would be set to e.g. -15, the actual combustion would not start until 25 °CA aTDC, which is unrealistically late and not intended. The used combustion profiles hence need to be cut at the timing when the combustion actually starts in the array, before being inserted as part of the tabulated combustion model. **Figure 45** illustrates the outcome of this procedure with a cut version of the original profile as represented by the solid line, while the dashed lines represent the same profile but with 10 °CA earlier or later SOC. When the operator now sets the SOC value to e.g. -15 using these profiles, the combustion will start advancing according to the defined combustion profile at this exact timing.

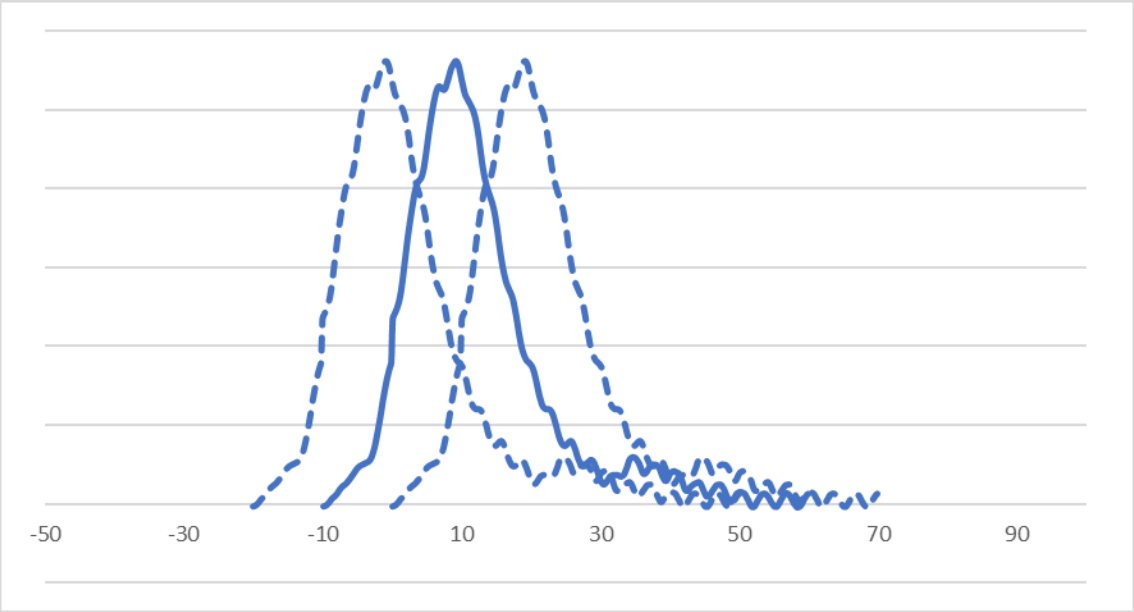

Figure 45. Filtered and modified measured HRR profiles at different timings.

When running regular simulations in GT-Power, SOC can be determined directly by the user, but determining SOC based on measured signals from an actual engine is more complicated, and especially when this value should be used for determining an accurate SOC for the tabulated combustion model. SOC is namely not measured nor calculated by the SCE control system; the closest signal possible to fetch is the CA5. If the CA5 value would be used as SOC in the model, the modelled combustion would always lag the real combustion with the amount of °CA between the real SOC and the CA5. To account for this difference in °CA, a simple linear SOC offset equation was created for each fuel used as seen in **Figure 46**. Looking at measured HRR profiles at different engine loads for a particular fuel, the °CA at the first rational positive HRR value from the left for each load was saved, to get the °CA at which the actual combustion starts (rational here meaning that arbitrary positive values far from the large positive rate of change in heat released were neglected). This °CA at the first positive HRR value was then subtracted from the actual measured CA5 for that specific HRR profile, to get a SOC offset. By doing this for each measured HRR profile in a load sweep for each combustion modality, graphs similar to that in **Figure 46** could be obtained. With load on the X-axis and SOC offset on the Y-axis, a linear trend was made from the SOC offset points describing what value should

be subtracted from the measured CA5 parameter value to model the SOC with more accuracy. For this particular combustion modality the SOC offset was rather constant across the load sweep at a value about 3.6 ± 0.15. However, since this calculated SOC offset is only dependent on two parameters while in reality tens of parameters would have to be considered to get a realistic representation of SOC offset, this is only a simple attempt at timing the SOC in a better way.

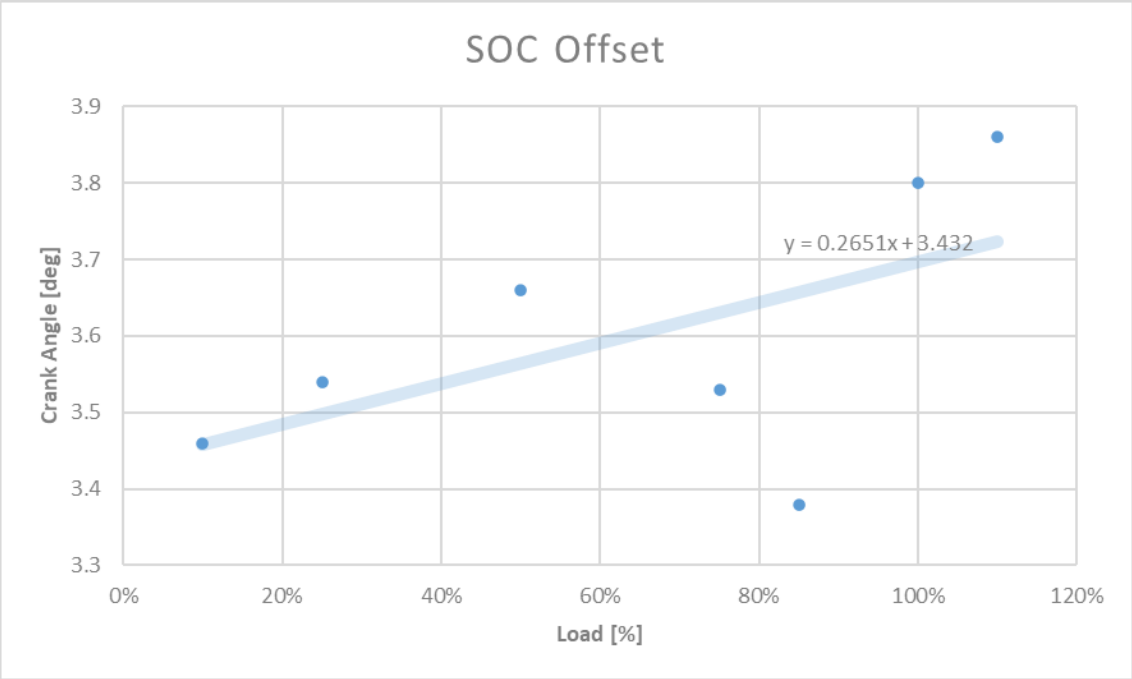

Figure 46. Linear trend for start of combustion offset.

Ideally in this setup, to model the in-cylinder combustion most accurately, either a measured cylinder pressure or BR should be directly imposed into the GT-Power model from the actual SCE in real-time. The cylinder pressure cannot however be directly imposed into the GT-Power model in the forward run combustion calculation, leaving BR or HRR as the only options. Also, the only way to directly impose the combustion in GT-Power, in real-time for every cycle, seems to be imposing the cumulative burned fraction one data point at a time. The difference between an instantaneous HRR and a cumulative HRR is illustrated in **Figure 47**; the cumulative HRR (orange dotted line) is obtained by integrating the instantaneous HRR (blue dotted line). To impose the cumulative burned

fraction one data point at a time there are two obstacles to overcome. Firstly, a module/subassembly in Simulink needs to be developed that fetches the cylinder pressure in real-time, calculates the HRR and sends this array point by point to the GT-Power model. Secondly, a GT-Power-xRT license is needed due to this very computationally expensive process. As mentioned in the Introduction chapter however, a GT-Power-xRT license was bought to Wärtsilä during the writing of this thesis. Also, another Master's thesis was written in parallel with this one, with the goal of creating a Simulink module for this purpose (Ahlskog, 2023). By imposing one data point at a time from this cumulative HRR profile as the engine cycle progresses, one can eliminate both the uncertainty regarding combustion profile shape and SOC timing, and hence get more accurate and realistic in-cylinder behaviour. The only uncertainty left then is the short lag between a BR and an HRR as mentioned previously.

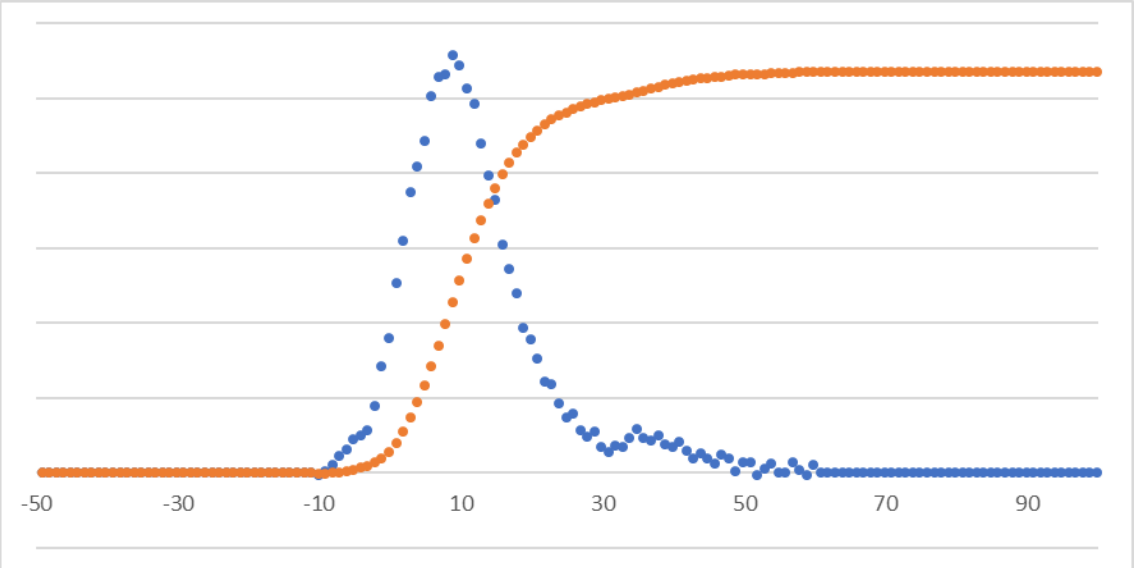

Figure 47. Instantaneous HRR (blue dotted line) versus cumulative HRR (orange dotted line).

5 Conclusion

To compensate for the lacking TC system in Wärtsilä's laboratory SCEs, the boundary conditions in especially the exhaust system must be calculated and imposed in the SCE control system. Given the limitations of the current Matlab TC model, this research aimed at developing a "GT-Power-in-the-Loop" (GTPiL) configuration where a virtual MCE acts as an extended digital twin to a laboratory SCE, where data is imposed from this engine onto its extended digital twin that then returns its calculated exhaust pressure back to the SCE. The first objective to be reached in the creation of this extended digital twin was to generate a real-time capable GT-Power FRM from an already existing, detailed GT-Power model representing a W8L25TS-DF engine, with minor deviation in model accuracy. The second objective was to develop the full GTPiL system based on Matlab/Simulink that contains the GT-Power FRM as a black box component.

The Literature review chapter examined previous work in the realm of real-time modelling and "in-the-Loop" utilizations in the internal combustion engine industry. The chapter affirmed that these techniques are broadly used in the industry to develop and test i.a. engines and control systems, and for further model-to-model coupling in more extensive fully virtual testing and validation. The software GT-Suite developed by Gamma Technologies has most often been the relied on "go-to" simulation software used in these applications, but in some cases engine models have also been fully described by e.g. Matlab/Simulink code.

As shown in the Software involved in GT-Power-in-the-Loop chapter, various software encompass the GTPiL system. The virtual MCE model has been developed in GT-Power, a template library of GT-Suite. The platform used for initializing and starting the simulation, and for communication between all involved systems, is Matlab/Simulink. The client/server communication between the GTPiL and SCE control systems during the Real-Time (RT) simulation is handled by the application-layer messaging protocol Modbus. When initializing the model for the Start-Stop (SS) simulation, the necessary crank-angle based data is fetched from Dewesoft. The Matlab script also triggers a macro developed

in Excel VBA during the post-processing of the SS simulation, that formats and visually presents the simulation results.

The results of the first objective were presented in chapter 4.2.1 GT-Power model, and although a small difference in the friction model parameters between the models resulting in BSFC discrepancies at lower loads, the FRM conversion process had not affected the FRM's performance in any significant shape or form. The rest of chapter 4 GT-Power-in-the-Loop presented the results of the second objective, where the working principle of the GTPiL system is basically to first aid the SCE with calculated TC quantities during the RT simulation, after which GTPiL can receive the necessary input from the SCE to run the SS simulation and save the results of the extended digital twin. Some challenges with this modelling approach also exist, uncovered partly by the comparison between the FRM and experiments, revealing the need for a minor recalibration of the FRM and for fine tuning of the semi-predictive tabulated combustion model. By implementing the Simulink module developed in a parallel thesis written by Ahlskog (2023), the in-cylinder phenomena during combustion and ultimately the performance of the whole model improves. With a well calibrated engine FRM, the GTPiL system developed in this thesis can be utilized for modelling exhaust pressure true to that of multi-cylinder engines, that can be used by the single-cylinder engine to counteract the effects of a missing turbocharger.

References

Ahlskog, A. (2023). *Operating a Single-Cylinder Engine with a Multi-Cylinder Engine 1D-Model in the Loop with Real-Time Combustion Feedback.* Vaasa: Yrkeshögskolan Novia.

Andric, J., Schimmel, D., Sediako, A., Sjoblom, J., & Faghani, E. (2018). Development and Calibration of One Dimensional Engine Model for Hardware-In-The-Loop Applications. *SAE Technical Paper 2018-01-0874, 2018*, doi:10.4271/2018-01-0874.

Bélanger, J., Venne, P., & Paquin, J.-N. (2008). The What, Where and Why of Real-Time Simulation. *Electronics, ISIE, 2008*, 2231-2235.

Corporate Finance Institute. (2022, May 11). *Excel Definition*. Retrieved from corporatefinanceinstitute.com: https://corporatefinanceinstitute.com/resources/excel/study/excel-definition-overview/

Corporate Finance Institute. (2022, May 11). *Excel VBA*. Retrieved from corporatefinanceinstitute.com: https://corporatefinanceinstitute.com/resources/excel/study/excel-vba/

DEWESoft. (2022, May 11). *What Is Data Acquisition (DAQ or DAS)? The Ultimate Guide*. Retrieved from dewesoft.com: https://dewesoft.com/daq/what-is-data-acquisition

Dorscheidt, F., Düzgün, M., Claßen, J., Krysmon, S., & Pischinger, S. (2021). Hardware-in-the-Loop Based Virtual Emission Calibration for a Gasoline Engine. *SAE Technical Paper 2021-01-0417, 2021*, doi:10.4271/2021-01-0417.

Gamma Technologies. (2022, April 28). *About Gamma Technologies*. Retrieved from gtisoft.com: https://www.gtisoft.com/about_gamma-technologies/

Gamma Technologies. (2022). *Engine Performance Application Manual.* Oakmont Lane, Westmont, USA: Gamma Technologies.

Gamma Technologies. (2022). *Flow Theory Manual.* Oakmont Lane, Westmont, USA: Gamma Technologies.

Hautala, S., Mikulski, M., Söderäng, E., Storm, X., & Niemi, S. (2022). Toward a digital twin of a mid-speed marine engine: From detailed 1D engine model to real-time implementation on a target platform. *International Journal of Engine Research, 24(12), 4553–4571*, 19.

MathWorks. (2021). *Matlab Primer.* Natick, Massachusetts, USA: MathWorks.

MathWorks. (2021). *Simulink Getting Started Guide.* Natick, Massachusetts, USA: MathWorks.

Palmkvist, A. (2011). *SCE - p3 - p5 calculation MATLAB.* Vaasa: Wärtsilä.

Pirker, G., Mayr, P., Kranawetter, K., Bauer, R., Seeber, R., & Wimmer, A. (2021). Application of the HiL Method to Develop Transient Operating Strategies for Highly Flexible Power Generation in Gas Engine Power Plants. *SAE Technical Paper 2021-01-0421, 2021*, doi:10.4271/2021-01-0421.

Precision Digital. (2022, May 11). *Introduction to Modbus Serial Communication*. Retrieved from predig.com: https://www.predig.com/whitepaper/introduction-to-modbus-serial-communication

Söderäng, E., Hautala, S., Mikulski, M., Storm, X., & Niemi, S. (2022). Development of a digital twin for real-time simulation of a combustion engine-based power plant with battery storage and grid coupling. *Energy Conversion and Management, 266, 115793*, 16.

Strang, A.-S. (2018). *Dimensioning of Auxiliaries for a Single-Cylinder Research Engine.* Vaasa: University of Vaasa.

Wu, H., & Li, M.-F. (2016). A Hardware-in-the-Loop (HIL) Bench Test of a GT-Power Fast Running Model for Rapid Control Prototyping (RCP) Verification. *SAE Technical Paper 2016-01-0549, 2016*, doi:10.4271/2016-01-0549.

Appendix

```
%% INITIALIZING MODEL
% SELECTIONS BY USER
%*************************************************************************
**************************************************************************
%*************************************************************************
**************************************************************************
% Give type of engine:
%       W25
%       W31
engineType = 'W25';
% Give stroke of the engine (in cm):
stroke = X;
% Give connecting rod length of the engine (in cm):
connectingRod = X;
% Select GT Model run alternative:
%       1) Start-Stop: GT (monitors ON)
%       2) Start-Stop: DOS (monitors OFF)
%       3) Real-Time: Simulink, Normal license
%       4) Real-Time: Simulink, xRT license
runAlt = 1;
% For Start-Stop (SS); select how to get the crank-angle based arrays (HRR,
cyl p, p3, p5):
%       1) Fetch everything from Dewesoft
%       2) Fetch cyl p, p3 and p5 from Dewesoft and calculate HRR in
%       Matlab.
%       3) Fetch cyl p, p3 and p5 from Indicom and calculate HRR in
%       Matlab.
fetchHRR = 1;
% For Real-Time (RT); select how to get the HRR:
%       1) Imposed HRR
%       2) SpeedGoat HRR
VSS_BR = 1;
% Select "local" or "real" run (i.e. from which folders to take input data):
%       0) Run local (when testing, runnable without modifications of the
code since a specific Autoreport file is taken directly)
%       1) Run local (when testing the script)
%       2) Run real (when running the engine)
pathsSrcDest = 2;
% Give main folder path
mainFolderPath = [cd '\'];
% Give REAL data folder paths
IndicomFileFolderReal = '\\accdom.for.int\LabData\Lab_doc\SCE Testing\G7\In-
dicom_results\'; % Indicom file data location
AutoreportFileFolderReal = '\\accdom.for.int\LabData\Lab_doc\SCE Test-
ing\G7\Export\'; % Autoreport export folder
DeWeFileFolderReal = '\\accdom.for.int\LabData\Doc_raw\SCE_Test-
ing\G7\Dewesoft_data\'; % Dewesoft file data location
outputFolderPathReal = '\\accdom.for.int\LabData\Lab_doc\SCE Test-
ing\G7\GTP_Results\';
% Give LOCAL data folder paths
IndicomFileFolderLocal = [cd '\INDICOM\'];
AutoreportFileFolderLocal = [cd '\AUTOREPORT\'];
DeWeFileFolderLocal = [cd '\DEWESOFT\'];
```

```
outputFolderPathLocal = [mainFolderPath 'OUTPUT\'];
% Give GT model version
GTversion = '2023';
% Give Simulink model name (without extension .slx)
SimulinkModelName = 'W8L2XDF_FRM_Simulink';
% Give GT model name (without extension .dat) as well as the time step to
% use for that model in Simulink
GTModel1Name = 'W8L25DF_FRM_Combined-V2023';
GTModel1SimTS = 0.0002;
GTModel2Name = 'W8L25DF_FRM_Combined_xRT-V2023';
GTModel2SimTS = 0.0002;
% Give GT subassembly model name (without extension .dat) for "Monitor", "No
% Monitor", "ImposedHRR", "ImposedHRR_xRT" and "SpeedGoat_HRR"
SubAss_Monitors = 'W8L25DF_FRM_Combined_Monitors-V2023';
SubAss_NoMonitors = 'W8L25DF_FRM_Combined_xRT_NoMonitors-V2023';
SubAss_ImposedHRR = 'W8L25DF_FRM_Combined_ImposedHRR-V2023';
SubAss_ImposedHRR_xRT = 'W8L25DF_FRM_Combined_xRT_ImposedHRR-V2023';
SubAss_SpeedGoatHRR = 'W8L25DF_FRM_Combined_xRT_SpeedGoatHRR-V2023';
% Select turbo according to the list in X.pdf.
selectedTurbo = 8;
% Select Fuel Mode:
%       0) DF - Port injection
%       1) DF - Direct injection
%       2) Pure Diesel engine (LFO)
fuelMode = 1;
% Select port-injected fuel (for use when fuelMode above is DF - Port injec-
tion):
%       0) Diesel (LFO)
%       1) Methane
%       2) Methanol
%       3) Ammonia
%       4) Hydrogen
selectedPortFuel = 0;
% Select direct-injected fuel:
%       0) Diesel (LFO)
%       1) Methane
%       2) Methanol
%       3) Ammonia
%       4) Hydrogen
selectedDirectFuel = 0;
% Give mass-% of the direct injected fuel (for use when fuelMode above is DF
- Port injection):
DIFuelMassPerc = 5;
% Give Compression Ratio:
CR_DF = X;
% Give Compression Ratio for Pure diesel engine:
CR_PureDSL = X;
% Give Flow Area Multiplier for IV
IV_FlowAreaMult = 2.0;
% Give Flow Area Multiplier for EV
EV_FlowAreaMult = 2.0;
% Give max IMEP for this engine
IMEP_Max = X;
% Give max BMEP for this engine
BMEP_Max = X;
% Select EWG control:
%       0) EWG closed
```

```
%       1) EWG controls lambda
%       2) EWG controls p3
%       3) EWG is automatic (controls lambda when gas, and p3 when diesel)
EWGControl = 2;
% Give Steady State Tolerance (in %) for p3 convergence:
p3_SteadyStateTol_Perc = 0.05;
% Give minimum amount of cycles to run:
minCycles = 10;
% Give maximum amount of cycles to run:
maxCycles = 200;
% Save measurement settings when running RT
% Give the amount of time in seconds that the startMeas signal over Modbus
must continuously have a value of 1,
% to activate the test point measurement timer.
timeUntilMeasActive = 10;
% Give the test point measurement time in seconds (normally 4 minutes)
timeUntilMeasDone = 30;
% Define GT parameter name, address, scaling factor and initial value
% for parameters to be sent over Modbus from Morphee.
modbusParameterList = ...
    { ...
    'RPM', 1038, 10, 1000; ...           % Divide by: 10
    'IMEP', 0184, 100, 21; ...                % Divide by: 100
    'Ambient_p0', 1049, 10, 1014; ...       % Divide by: 10 (mbar)
    'Ambient_t0', 1058, 100, -5; ...        % Divide by: 100 (C)
    'target_p3', 1162, 1000, 4; ...       % Divide by: 1000 (bar)
    'target_T3', 1167, 100, 60; ...         % Divide by: 100 (C)
    'target_lambda', 1000, 1000, 1.5; ...     % Divide by: 1000
    'EVO_TIMING', 4887, 100, 36; ...          % Divide by: 100
    'EVC_TIMING', 4888, 100, -5; ...          % Divide by: 100
    'IVO_TIMING', 4883, 100, 12; ...          % Divide by: 100
    'IVC_TIMING', 4885, 100, -45; ...         % Divide by: 100
    'HRR5', 0185, 100, 0; ...                % Divide by: 100
    };
% Define variables for Modbus connection
connectionType = 'tcpip';
deviceAddress = '10.193.27.10';
port = 502;
% Give names for files related to the creation of instantaneous variables
% from GT (without extensions .exp and .txt)
exportFileName = 'exportdata';
instVarFileName = 'GT_InstVar';
% Give names for Excel macro
ExcelMacro = ...
    { ...
    'Macro_adjustOutput.xlsm'; ...
    'v6'; ...
    'adjustOutput'; ...
    };
% Give parameter file name (without extension .txt)
paramFileName = 'paramfile';
%**************************************************************************
**********************************************************************
%**************************************************************************
**********************************************************************
```