Dynamic Model Based Diagnosis for Combustion Engines in RODON

Master Thesis
performed at **Vehicular Systems**
and **Sörman Information & Media AB**

by

Joella Lundkvist
Stina Wahnström

Reg nr: LiTH-ISY-EX -- 07/4003 -- SE

August 31, 2007

Dynamic Model Based Diagnosis for Combustion Engines in RODON

Master Thesis

performed at **Vehicular Systems**
at **Linköpings University**

and at **Sörman Information & Media AB**
by

Joella Lundkvist & Stina Wahnström

Reg nr: LiTH-ISY-EX -- 07/4003 -- SE

Supervisor: **Peter Bunus, Ph.D**
Sörman Information & Media AB
Mattias Krysander, Ph.D
Linköping University

Examiner: **Associate Professor Lars Eriksson**
Linköping University

Linköping, August 31, 2007

Avdelning, Institution
Division, Department

Division of Vehicular Systems
Department of Electrical Engineering
Linköpings universitet
SE-58183 Linköping, Sweden

Datum
Date

2007-08-31

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
—

ISRN
LiTH-ISY-EX -- 07/4003 -- SE

Serietitel och serienummer
Title of series, numbering

ISSN
—

URL för elektronisk version
http://www.vehicular.isy.liu.se
http://ep.liu.se

Titel
Title

Dynamisk modellbaserad diagnos av förbränningsmotorer med RODON

Dynamic Model Based Diagnosis for Combustion Engines in RODON

Författare
Author

Joella Lundkvist & Stina Wahnström

Sammanfattning
Abstract

Diagnosis is the task of finding faults or malfunctioning components in a technical system, e.g a car. When doing diagnosis on cars with combustion engines, a computer program can be used. The computer program, also called diagnosis system, needs information about the car. This information could be data sheets of all the electronic components in the car. It could also be a description of how the engine behaves in a nominal and a non-nominal case. This information is contained in a model of the engine. RODON, a diagnostic tool developed by Sörman Information and Media AB, uses models of systems for conflict detection diagnosis. RODON needs fault models of the components to do diagnosis. The diagnosis system is then used in workshops, factories, or other places where cars need to be surveyed.

In this thesis, a Simulink model of the nominal behaviour of a combustion engine is given. The problem is how to make use of the model as well as the diagnostic tool RODON for combustion engine diagnosis. To solve this, the Simulink model is translated into a RODON model. Translating a Simulink model into a RODON model requires a new library in RODON. The library developed in this thesis is called AdvancedBlocks library.

The Simulink model describes the nominal behaviour of a combustion engine but for diagnosis with RODON, fault models are needed as well. Several types of faults that can occur in an engine have been studied and fault models have been implemented in RODON. The conclusion is that diagnosis in RODON with a translated engine model is possible.

Nyckelord
Keywords

Model Based Diagnosis, Conflict Detection Diagnosis, RODON, Simulink to RODON Translator

Abstract

Diagnosis is the task of finding faults or malfunctioning components in a technical system, e.g a car. When doing diagnosis on cars with combustion engines, a computer program can be used. The computer program, also called diagnosis system, needs information about the car. This information could be data sheets of all the electronic components in the car. It could also be a description of how the engine behaves in a nominal and a non-nominal case. This information is contained in a model of the engine. RODON, a diagnostic tool developed by Sörman Information and Media AB, uses models of systems for conflict detection diagnosis. RODON needs fault models of the components to do diagnosis. The diagnosis system is then used in workshops, factories, or other places where cars need to be surveyed.

In this thesis, a Simulink model of the nominal behaviour of a combustion engine is given. The problem is how to make use of the model as well as the diagnostic tool RODON for combustion engine diagnosis. To solve this, the Simulink model is translated into a RODON model. Translating a Simulink model into a RODON model requires a new library in RODON. The library developed in this thesis is called AdvancedBlocks library.

The Simulink model describes the nominal behaviour of a combustion engine but for diagnosis with RODON, fault models are needed as well. Several types of faults that can occur in an engine have been studied and fault models have been implemented in RODON. The conclusion is that diagnosis in RODON with a translated engine model is possible.

Keywords: Model Based Diagnosis, Conflict Detection Diagnosis, RODON, Simulink to RODON Translator

Preface

This master thesis was performed at Sörman Information & Media AB (Sörman) in Linköping and at the Department of Electrical Engineering, division of Vehicular Systems, at Linköping University. Sörman is a leading technology provider of Product Lifecycle Management (PLM) information solutions and services for advanced products and systems.

Acknowledgment

A wise little girl with cool glasses told us that this might be the only time we could write exactly what we want in a technical report, so this is an opportunity we are going to take. During this project with our master thesis, we have come across a lot of people with a lot of knowledge in most varying fields. For example that it is stupid to stand under the chimney you are trying to tear down, and that a one dimensional array is not the same as scalar value. With great joy and big enthusiasm we have absorbed all this information and some of it ended up here in this report.

First we have our colleagues at Sörman. Peter Bunus, you have guided us along this project with a big heart and faith in us. Then we have Henrik Johansson and Kaj Nyström that can answer to all kinds of questions concerning programming, turtles and how to swing the control when playing Wii. Our German colleague Jürgen Zoller and all the others in Heidenheim, have always helped us in tricky modelling situations but have also with big hospitality shown us the German culture. Beate, Martin, Jürgen, Frank, Burkhard and Gerd, thank you!

To continue, we have our thesis colleague Sebastian Schygge that is a good friend and knows a lot about carpentry. We would like to thank our boss at Sörman, Johan Gunnarsson, who gave us the opportunity to make this thesis. We would also like to thank Bourhane for interesting discussions over a cup of coffee.

This thesis could not have been done unless our examiner, Lars Eriksson at the department of Vehicular Systems at Linköping University, would have wanted us to. You also assigned us the best supervisor, Mattias Krysander. Mattias, you have always been there for us, pushing us in the right directions in the jungle of writing a report and shown us how to overcome technical problems in diagnosis.

We are here today thanks to our parents who have raised us well to become curious young women with a lot of love and happiness to give to our environment. Lena, Kjell, Anna and Lasse, thank you! We would also like to thank the rest our families, for always supporting and believing in us.

Last but not least, we have our beloved boyfriends that has not always been happy about our whining and constantly talking about the project, but still stood by our side and laughed with us in happy times and consoled us when needed.

We have probably left out many important persons, but do not feel offended. Come join us in our dance of life, which has just changed from a well structured foxtrot to a mad lindy hop.

Joella Lundkvist and Stina Wahnström
Linköping, August, 2007

Contents

Abstract v

Preface and Acknowledgment vi

I Introduction and Theory xi

1 Introduction 1
- 1.1 Problem Description . . . 1
- 1.2 Objectives . . . 1
- 1.3 Limitations . . . 2
- 1.4 Existing Work . . . 2
- 1.5 Contributions . . . 2
- 1.6 Target Group . . . 2
- 1.7 Overwiew . . . 3

2 Diagnosis 5
- 2.1 General Concept . . . 5
- 2.2 Model Based Diagnosis . . . 6
- 2.3 Conflict Detection Diagnosis . . . 7
- 2.4 Fault . . . 8

3 RODON 9
- 3.1 Introduction to RODON . . . 9
- 3.2 Simulation and Diagnosis . . . 11
- 3.3 Working with RODON . . . 16
- 3.4 More Functionality of RODON . . . 16

II Design 17

4 Choice of Diagnosis Method 18
- 4.1 Introduction . . . 18
- 4.2 Modelling in Rodelica . . . 19

4.3 MATLAB Diagnostic Toolbox 19
4.4 Translator 20
4.5 Conclusions 20

5 Translation from Simulink to RODON **22**
5.1 Introduction 22
5.2 Levels of Translation 25
5.3 Functionality of the Translator 26
5.4 AdvancedBlocks Library 27
5.5 Case Study: The Engine Model 37

6 Fault modelling in RODON **44**
6.1 Faults in the Air Intake System 44
6.2 Faults Impact on the Engine System 45
6.3 Different Implementation Methods 46
6.4 Implementation 46
6.5 Discussion 59

III Model Based Diagnosis **60**

7 RODONDiagnosis on the Air Intake System **61**
7.1 Data Generation 61
7.2 Time Simulation in RODON 64
7.3 Fault Free Case 64
7.4 Sensor Fault 65
7.5 Clogging 67
7.6 Leakage 69
7.7 Multiple Faults 70
7.8 Discussion 72

8 Discussion **73**

9 Conclusions **75**

10 Future Work **76**
10.1 AdvancedBlocks library 76
10.2 Diagnosis of the Engine 76

References **78**

Notation **80**

A Translator **81**

B Data Generation **86**

Part I

Introduction and Theory

Chapter 1

Introduction

1.1 Problem Description

In a car many faults can occur, for example a flat tire or dimmed front lights. These faults can be discovered by looking at the car. There are other faults that may be harder to discover. For example, when the engine stops the cause can often not be determined by looking at it. The driver might then take the car to a workshop to find out the problem. Given the complexity in a car the mechanic often uses a computer based program to determine the fault, i.e to diagnose the engine. The computer program, also called diagnosis system, needs information about the car to be helpful for the mechanic. This information could be data sheets of all the electronic components in the car. It could also be a description of how the engine behaves in a nominal and a non-nominal case. Such a description is for example a model of the engine.

The process of constructing a diagnosis system usually begins in the design department. The design department produces the model that describes the physical system, e.g. a car engine. When the model is ready the diagnosis department continues the work. Their work consists of identifying, describing, and implementing the possible faults the system can have. The diagnosis system is then used in workshops, factories, or other places where systems need to be surveyed.

The aim of this thesis work is to investigate how a method for the diagnosis process can be done.

1.2 Objectives

The objective of this thesis is to present a diagnosis proposal for a combustion engine. RODON, the diagnostic tool from Sörman Information & Media AB, is used to present possible faults that may occur in the engine. The existing model of the engine is a Simulink [8] model. Hence the translation of the

model from Simulink to Rodelica, the programming language in RODON, is an important stage for the diagnosis process.

An emphasis is put on how the diagnosis is performed. The meaning of diagnosis in this thesis is to identify the possible faults and to implement them. For doing diagnosis, faults must be identified and knowledge about how the faults affect the system must be acquired. The goal is to create a process that uses the information from the design departments in the workshops to find the real fault, thus to do diagnosis on an engine.

1.3 Limitations

The objective of this thesis is to investigate a method for performing diagnosis on a combustion engine using RODON. Hence, several types of engine faults are implemented. The air intake system of an engine is studied. The reason is that it is possible to find the common faults desired to diagnose in this part of the engine, for example sensor faults and leakages. The intake system in this thesis work consists of the air filter, the following pipe, the compressor, and its following pipe.

1.4 Existing Work

A third party translator is used for translation of the Simulink model into a Rodelica model. The Simulink model of a turbo charged engine [2] is given and seen in Figure 1.1. The model describes the components in the engine. Moreover it describes the effect of the air and fuel that flows through the engine.

1.5 Contributions

A library in RODON, called AdvancedBlocks, has been developed. The library is used by a Simulink to Rodelica translator. Accordingly, Simulink models can now be used in RODON for diagnosis. Several types of faults that can occur in an engine have been studied and implemented in RODON and the translated engine model can be diagnosed in RODON.

1.6 Target Group

The target group for this thesis is undergraduate and graduate engineering students with a background from electrical engineering with an interest of learning more about this area of model based diagnosis.

1.7 Overwiew

Part I This part gives an introduction to the thesis. The background of the project is described as well as the important parts of diagnosis needed in this thesis. Also a description of the diagnosis tool RODON is given in this part.

Part II This is the part where the design and structures of the diagnosis method are presented. All the contributing work is presented and explained. This part is divided into two sections. The first one concerning the translator, the issues in transforming a Simulink model into a Rodelica model. The diagnosis design is presented in the second section.

Part III This is the final concluding part which presents the outcome of the diagnosis and conveys the results and the analysis of the work. The faults are diagnosed as single faults as well as multiple faults. This part assembles the thesis and concludes it with some thoughts for the future.

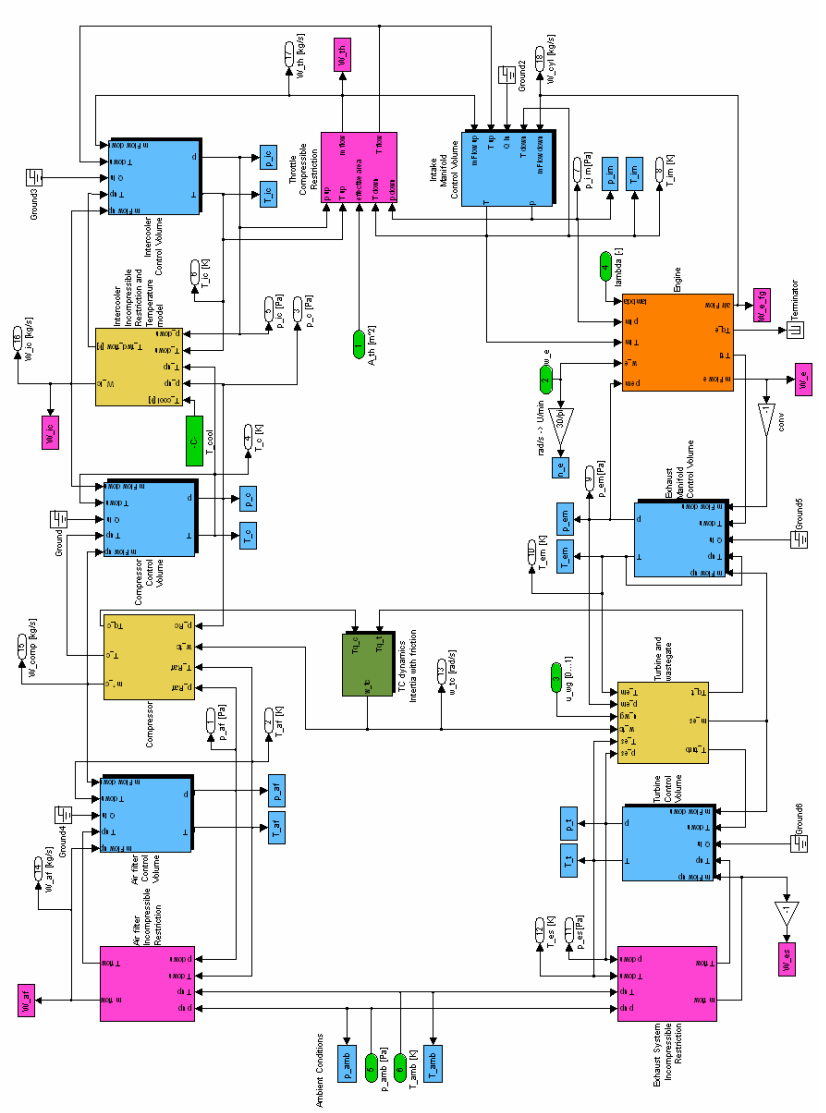


Figure 1.1: Model of the engine in Simulink.

Chapter 2

Diagnosis

This chapter gives an introduction to diagnosis with emphasise on diagnosis with RODON.

2.1 General Concept

Diagnosis, from the Greek words *dia* meaning *by* and *gnosis* meaning *knowledge*, is the process of identifying a disease by its signs and symptoms, [16]. When a car engine is under consideration the disease is a fault in the engine which affects the car so that it is not drivable. This can be an issue concerning the comfort of driving the car, e.g. the engine sounds too much. It can also be an environmental matter, for example that the levels of emissions are too high. The aim of the diagnosis is to find the malfunctioning of the system. There are several methods to find the faults in the system.

One method to do diagnosis is to compare sensor values to a predefined threshold value. If the sensor signal exceeds the limit, an alarm is sent out notifying that the sensor value is too high. Another technique is to use redundancy of components, i.e. to have several sensors measuring the same value. This is most common in safety critical systems where it is crucial to have a functioning system.

A method of diagnosis, that has a close connection to automatic control, is model based diagnosis. Model based diagnosis is based on the First Principle in Engineering which means that the relationship between function, behaviour, and structure can be described by using physical laws of nature. This means that with a mathematical model of a system and by using the same input signal as for the physical system, it is possible to find faulty behaviour or faulty components.

As an introduction to the diagnosis concept, an example of a system is introduced. A circuit which consists of a battery, a light bulb, and a switch

connected by electrical wires can be seen in Figure 2.1. Each component can work correctly or incorrectly. Incorrectly means that the component is not working the way it is designed for. For example the lamp can work just fine, it can be dimmed or it might not shine at all. These different modes of the lamp and the other components are called behaviour modes. A behaviour mode is a state that the component can be in. If a component is in a behaviour mode and does not work correctly, then this behaviour mode is called a failure mode.

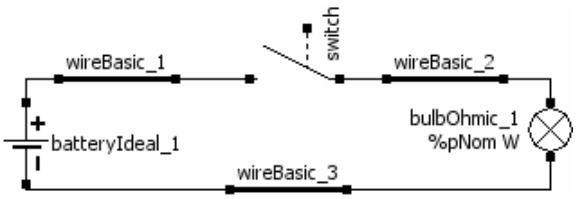


Figure 2.1: Small electrical circuit.

Several observations can be done by observing a system. In this example, one can see if the lamp is lit or not, or if the switch is open or closed. It is also possible to measure the voltage somewhere in the circuit. By using these observations one can find out which behaviour mode the system is in. Thus, it is possible to make a diagnosis by using knowledge of the system and the observations.

2.2 Model Based Diagnosis

In model based diagnosis the models can be of different types, for example logic based models or models based on differential equations. In Section 2.1 the idea of hardware redundancy was explained, i.e. having several sensors measuring the same value and comparing them. In model based diagnosis the sensor value is instead compared to the output from a model, this is called analytical redundancy.

In [5] analytical redundancy is defined as:

Definition [Analytical redundancy] *There exists analytical redundancy if there exists two or more, but not necessarily identical ways to determine a variable where at least one uses a mathematical process model in analytical form.*

Figure 2.2 explains the idea of analytical redundancy. The system output is compared to the model output. In a nominal case these two should be the same, given a correct model. Compared to traditional limit checking and hardware redundancy, model based diagnosis has several advantages, for example:

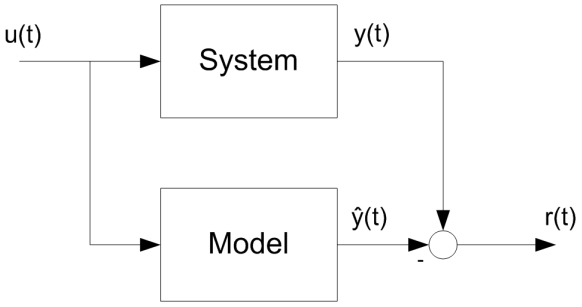


Figure 2.2: Simple fault detection system.

- It can provide higher diagnosis performance.
- It can be performed passively.
- Isolation of different faults becomes possible.
- No extra hardware is needed

There are also disadvantages concerning model based diagnosis, e.g. the need of a reliable model [12]. One reason for having a reliable model is that a diagnosis system operates in open loop, which means that the system does not get any feed-back. Driving a car illustrates the difference between closed loop and open loop. The person driving the car is the control system and makes a model of the road and how to control the car. The model is constantly being updated with new information from the driver who sees how the road changes. If the driver closes the eyes the model is not getting any feed-back on how the control system is to behave next, i.e. the driver operates in open-loop. Therefore the driver might come to a bad decision on how to manoeuvre the car.

2.3 Conflict Detection Diagnosis

Diagnosis of technical systems has evolved from two origins, automatic control and computer science. The approach for the two has been a bit different [14].

Conflict detection diagnosis is one way to do model based diagnosis from a computer science point of view and the idea of conflict detection diagnosis is shown in Figure 2.3. The model describes the behaviour modes of the system. The physical plant can be for example a car or a nuclear reactor. Measurements and sensors give a set of observations. The task for the diagnosis system is to find a set of behaviour modes which describes the observations correctly. A conflict is detected if the set of observations is not consistent with

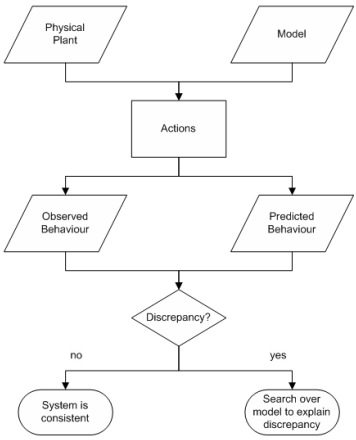


Figure 2.3: Basic idea of conflict detection diagnosis.

the model. Given a conflict, the failure modes which accurately describes the model and the observations is assessed. Those are the candidates of the diagnosis. Consequently, the candidates are the outcome of the diagnosis. The conflict detection diagnosis will be treated further in Chapter 3 where RODON is demonstrated.

2.4 Fault

A fault in a system or a component is a failure to perform its normal and characteristic behaviour. In [15] a fault is defined as: ”Unpermitted deviation of at least one characteristic property or variable of the system from acceptable/unusual/standard/nominal behaviour.”

Chapter 3

RODON

This chapter describes the diagnosis tool RODON.

3.1 Introduction to RODON

RODON is a tool used for model based analysis of systems. The origin of the RODON modelling language, Rodelica, is the object oriented language Modelica [10]. Rodelica is designed for multi-domain modelling involving for example applications in the mechanical, electrical and hydraulically world. In Rodelica, it is possible to do high level modelling by composition as well as detailed library component modelling by equations [17]. Models of standard components are typically available in model libraries, see for an example Figure 3.1. The figure shows the models in the Rodelica bulb library. The difference between the three models is that the two last bulbs have a fixed power consumption of $10W$ and $55W$ while for the first bulb the power consumption depends on the voltage running through the bulb.

Figure 3.1: Rodelica bulb library.

In Figure 2.1, an example of an electrical circuit was given. The same example will be studied here, but now to exemplify the functionality of RODON. The textual representation of the model is:

```
1:   model electricalCircuit
2:     BulbOhmic bulbOhmic_1;
3:     WireBasic wireBasic_1;
4:     WireBasic wireBasic_2;
5:     WireBasic wireBasic_3;
6:     BatteryIdeal batteryIdeal_1;
7:     SwitchOnOff switch;
8:   behavior
9:     connect(wireBasic_1.p2, batteryIdeal_1.p1);
10:    connect(wireBasic_1.p1, switchOnOff_1.p1);
11:    connect(wireBasic_2.p2, switchOnOff_1.p0);
12:    connect(wireBasic_2.p1, bulbOhmic_1.p1);
13:    connect(wireBasic_3.p1, bulbOhmic_1.p2);
14:    connect(batteryIdeal_1.p2, wireBasic_3.p2);
15:  end electricalCircuit;
```

The statement `BulbOhmic bulbOhmic1` on line 1 declares a component *bulbOhmic1* of model class *BulbOhmic*. The other components are defined in the same way in the lines 3 to 7. The connections between components are defined in the behaviour section, line 8 to 14. The `p0`, `p1` and `p2` in the connect statements are electrical pins of each component that render the possibility to connect components to each other.

The model above describes the general system. The physical behaviour is though modelled on a lower level, i.e. in each component. The general system inherits the physical behaviour of the components in the model. The wire model will be investigated for a better understanding.

```
1:   model WireBasic
2:      Pin p1;
3:      Pin p2;
4:      FailureMode fm (max = 1,
5:      mapping = "ok, disconnected");
6:   behavior
7:      if(fm = = 0){
8:        // Current balance:
9:        Kirchhoff(p1.i, p2.i);
10:        // No voltage drop between pin 1 and 2:
11:       p1.u = p2.u;
12:      }
13:     if(fm = = 1){
```

```
14:        // Current balance:
15:        Kirchhoff(p1.i, p2.i);
16:        // No current through pin 1:
17:        p1.i = 0;
18:       }
19:   end WireBasic;
```

In the behaviour section it is possible to describe the physical behaviour of the component, for the wireBasic Kirchhoff's law is used. In RODON, Kirchhoff's law is a predefined standard constraint and can therefore be used without the need of typing its equation. One feature in RODON is the possibility of specifying failure modes, hence simulations of faulty systems in RODON are achievable [11]. A definition of a failure mode is seen on line 4 where `fm` is the failure mode variable. The failure mode represent the *wire* that can be *ok* or *disconnected*. The failure mode is a discrete variable. The nominal minimal value is 0 and the maximum value is here defined to be 1. This implies two discrete states. The first one represents the 0 and the *ok* case. The second state corresponds to the 1 and the *disconnected* case. The constraints that are valid for each failure mode are represented in the behaviour section.

3.2 Simulation and Diagnosis

RODON diagnosis is based on the conflict detection diagnosis described in Section 2.3. To explain how RODON performs the diagnosis the same example as above is studied. The physical behaviour of each component is described by constraints in the behaviour section of the model. Together these constraints create an equation system. An example of an equation system could be,

$$p1.i + p2.i = 0$$

defined by Kirchhoff's law. For a nominal behaviour of the circuit this constraint must be valid. In each component it is possible to define failure modes. The failure modes describe the physical behaviour of the component in case of a fault. For a wire, a faulty behaviour could be that Kirchhoff's law does not hold. Imagine now that the current is measured somewhere in the circuit. This measurement can thus provide the system with data, for example:

$$p1.i = 0.1.$$

Given the equation system and additional data, RODON can try to solve the system of equations. First it will be checked if the nominal behaviour equations can be solved. If so, the conclusion of the diagnosis is that the system is ok, i.e. the fault free equation system is consistent together with the observations. If not, RODON will check if any of the failure modes could

contribute to a consistent equation system. In that case, the components of the contributing failure modes are the candidates of the diagnosis. In the example above, `fm` would be set to 1 and the diagnosis would be a disconnected wire.

This is how RODON performs a static simulation. It is also possible in RODON to do a kind of dynamic simulation where a time step is inserted into the models. In each time step, an equation system is solved like for the static case. However, when doing this kind of simulation, it will only be executed until a conflict is detected. The simulation then stops and diagnosis can be performed. This means that it is not possible to get candidates over time, i.e. to have different candidates at different times.

Simulation and Diagnosis Example in RODON

In this section, the example in Figure 2.1 will be studied again, this time to show how a simulation and a diagnosis can be performed in RODON. The small electrical circuit is loaded into the Analyser mode in RODON where it is possible to do simulation and diagnosis. The Analyser mode is described further in Section 3.3. The longer list in the bottom of Figure 3.2 shows the variables for the battery, the switch and the bulb. By double clicking on the variables it is possible to set their values. In the figure, the position for the switch is set to on. This means that the circuit is closed and that current can flow through it. In Figure 3.3 the result of the simulation is shown. The switch position in the bottom of the list is grey because it is a preset value. One can for example see that the failure modes `fm` are ok for all the components and that the bulb is shining bright, and that the simulation succeeded.

In Figure 3.4, the model has been reloaded and the original values are back. The switch is set to on as before and additionally the bulb is set to off. The result of the simulation can be found in Figure 3.5. This time the preset values do not match and conflicts are detected. The reason is that the bulb is not shining even though there is current flowing through the circuit according to the switch position. Logically, this implies that something is wrong in the circuit. The diagnosis outcome in Figure 3.6 tells us that the switch and the bulb are both candidates of the diagnosis. Either the bulb is disconnected, otherwise the switch. That the bulb is disconnected means that it has no current running trough it. The cause can be that it is broken or that the current flow has been interrupted in another way. Concisely, RODON managed to do the simulation and diagnosis candidates were given.

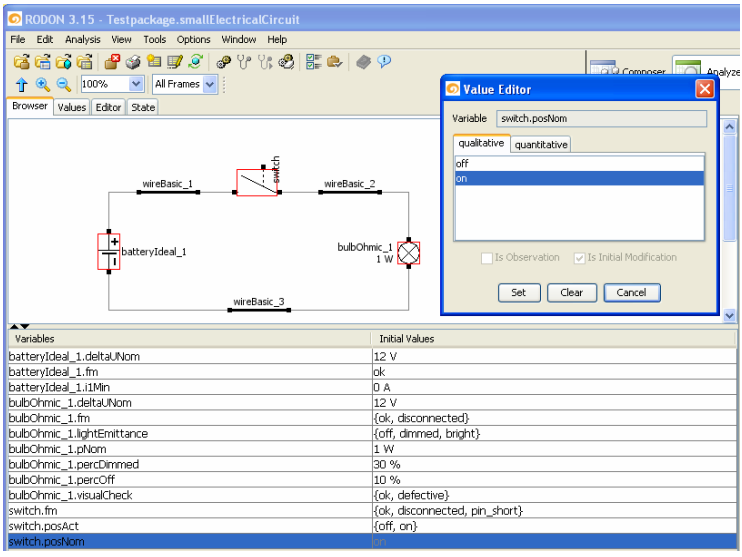


Figure 3.2: Value setting in a small electrical circuit.

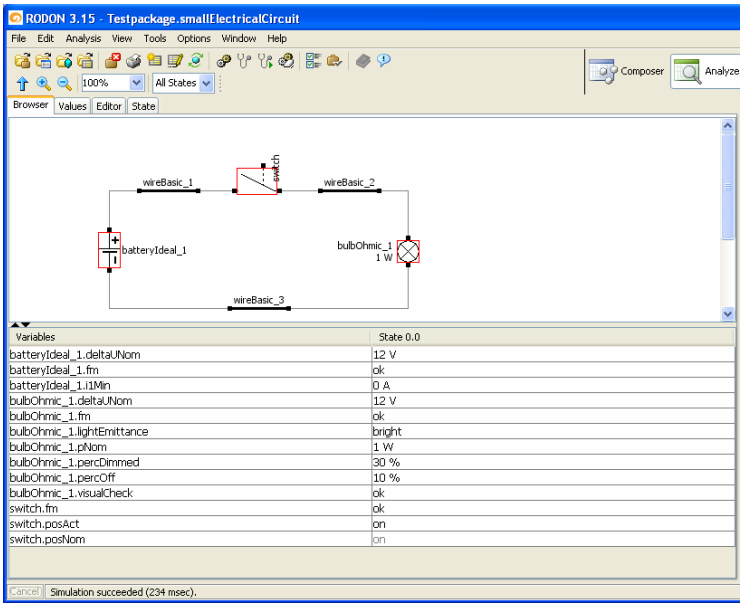


Figure 3.3: Simulation result of small electrical circuit.

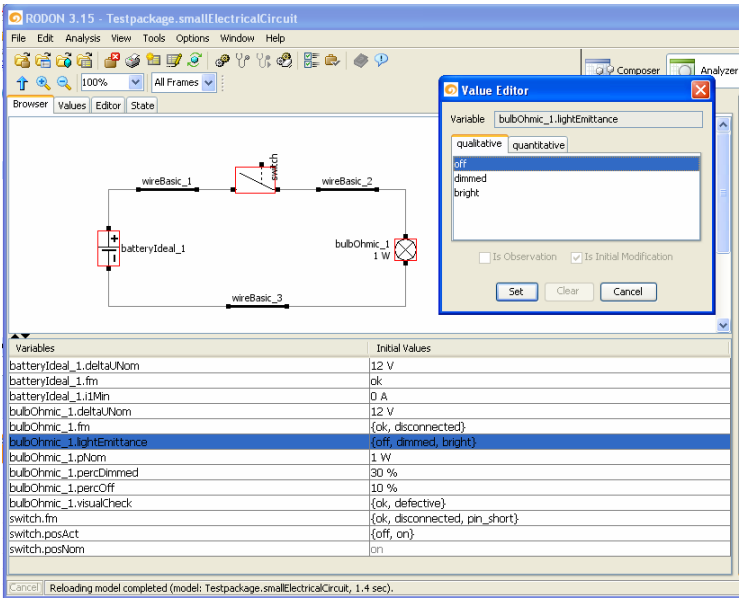


Figure 3.4: Setting conflict values in the circuit.

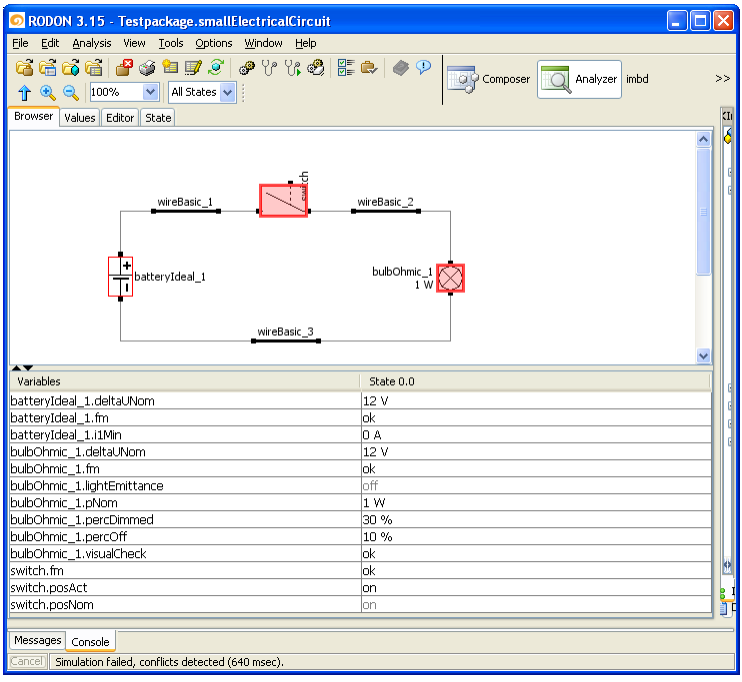


Figure 3.5: Simulation result.

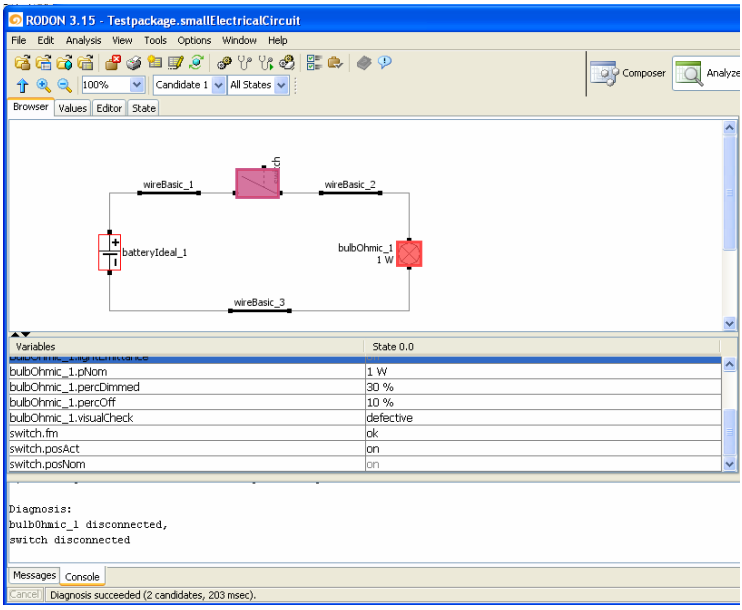


Figure 3.6: Diagnosis outcome.

3.3 Working with RODON

RODON has two different modes which were used in this thesis, Composer mode and Analyzer mode. The Analyser mode has already been showed in the previous section. In the Composer mode it is possible to do the modelling, change names and replace components. The faults are modelled in this mode and all properties to the nominal and faulty behaviour are set. In the Analyzer mode, the evaluation of the model is done and the variables are instantiated and receive their proper value. It is possible to simulate and to diagnose. To do simulation and diagnosis, the model sometimes needs data files, these can also be generated or created in this mode.

Modelling in RODON can be performed in two ways. One way is to drag and drop the components or models into the graphical representation of the model, for example a bulb from the bulb library in Figure 3.1. Another way is to use textual programming, an example of this can be seen in Section 3.1. The two alternatives are not isolated from each other, it is possible to drag and drop a component and afterwards manually edit its behaviour or add for example a failure mode.

3.4 More Functionality of RODON

RODON has more functionalities than the one that was showed in Section 3.2. For example AutoSim simulation which is a simulation testing of all possible cases, both faulty and nominal ones, in a model. The simulation generates a lot of data stored in a database. The data can be used to generate decision trees and FMEA. Decision trees [18] are used for off board diagnosis. The concept is to propagate through predefined trees to find the malfunctioning component. FMEA [4], Failure Mode Effective Analysis, is an analysis done for calculation of consequences of faulty.

Part II

Design

Chapter 4

Choice of Diagnosis Method

In this chapter, three different diagnosis methods are evaluated. The diagnosis methods are intended to be a bridge between the Simulink model presented in Section 1.4 and a diagnosis solution that can be used in the workshops for diagnosing combustion engines.

4.1 Introduction

As presented in Section 1.4, a Simulink model of a combustion engine already exists. The model describes the nominal behaviour of the air and fuel flow in the engine and was built for engine control. Consequently, the model contains no models of faults that can occur in the engine. Another prerequisite is the diagnostic tool RODON which can diagnose RODON models. The objective of this thesis is to find a diagnosis method which uses the Simulink model to present a diagnosis result, either by using RODON as diagnostic tool or by finding another solution.

RODON is a tool used for diagnosing static system and the diagnosis is based on the conflict detection diagnosis described in Section 2.3. In MATLAB and Simulink, diagnosis is done by using precompiled test which means for example that hypothesis tests and residuals are constructed [15]. A problem is that the desired conflict detection diagnosis on combustion engines can not be done in MATLAB without changes. In addition, RODON is used for static diagnosis and the combustion engine is a dynamic system. Thus, the purpose of this thesis is to find a method that solves the problem of using information in the Simulink model for doing model based diagnosis. The possible methods to consider are:

- Modelling in Rodelica
- Develop a diagnosis solution for another tool than RODON
- Using a translator and develop RODON to handle dynamic systems

The benefits and drawbacks for each method are studied. In the end of the chapter a conclusion on which design to use is presented. Hence, only one method will be implemented, the others are evaluated in this chapter to reach a conclusion on which method to use.

4.2 Modelling in Rodelica

One way to carry out diagnosis is to use the tool RODON. For doing this it is necessary to have a model in Rodelica. This model can be created in different ways, for example an engineer can build the model. Another option is to import the topology of a model from e.g. a CAD drawing and to model the behaviour in RODON. An advantage when modelling in RODON is that the tool for diagnosis, i.e. RODON, is implemented and ready to be used. Once a model is introduced in RODON, diagnosis is possible to do. Considering faults and fault models, they can be introduced into the RODON model in a natural way when doing the modelling. A disadvantage is though, that the modelling work is time demanding and moreover will be duplicated since a Simulink model already has been modelled. As described in the introduction section, RODON is not a dynamic diagnostic engine which is a drawback when it comes to modelling an engine. Since it is possible to time discretize derivatives it should be possible to overcome this problem.

4.3 MATLAB Diagnostic Toolbox

The diagnosis task is not necessarily connected to RODON. Many design departments in companies use MATLAB and Simulink for modelling. Thus, a MATLAB Diagnostic Toolbox is an alternative when the diagnosis is the goal and not necessarily by using RODON. MATLAB and Simulink are already used for diagnosis in many on-board and off-board applications where the residual concept is applied [15]. An advantage with a toolbox is that the model, which already exists, can be used directly and does not need to be converted into another modelling language. On the other hand, the disadvantage is that Simulink is based on signal flow modelling. The signals are defined as the output of a block in a Simulink diagram where the blocks might contain dynamic systems [9]. The lines represent the variables and the relationship between the variables is represented by the blocks. This means that the causality is fixed for the system. When modelling faults in a system, causality can change and that means that the Simulink model is no longer valid. A Simulink model generally describes the nominal behaviour and when introducing the faults the causality might get affected. This causes trouble, MATLAB can not handle this changed causality. This means that a new way of thinking needs to be introduced into MATLAB. To be able to do this a great amount of information is needed. To sum up, MATLAB is a really good

and powerful tool for simulation, but for diagnosis it is not optimal. Some research needs to be done on how to overcome the fixed causality.

4.4 Translator

As described in Section 1.1, the combustion engine is modelled in Simulink. It is desirable to use this model, or other Simulink models, in Rodelica but without the time demanding task of rebuilding them. If a translator is used the model will be ready to use in RODON right away. The advantage of using a translator is the reusability; a translator can be used many times for different systems. A disadvantage with the translator is that it translates the nominal behaviour of the Simulink model. This means that fault models need to be introduced after the translation. This might be a problem since the concept of an automatic translation is to remove the human factor. By making a framework for how to model the faults the factor can be reduced. A big advantage by using RODON as a diagnosis tool is that the diagnosis takes place automatically. As for the first option, Modelling in Rodelica, this option would need a development of RODON to also handle dynamic system. However, by discretizing derivates it is a problem that can be handled right now.

4.5 Conclusions

Given is a Simulink model and the problem of doing diagnosis. There are three methods which have been presented. The three methods are all associated with advantages and disadvantages as described above.

First, a MATLAB Diagnostic Toolbox. However, Simulink and MATLAB are created for simulation and not for diagnosis and a faulty behaviour is not easily implemented in Simulink. A more thorough investigation is needed to be able to do a toolbox. The conclusion is that the MATLAB Diagnostic Toolbox will not be implemented.

The remaining alternatives both use translated models of the system. The question is how to translate the Simulink model into RODON; either by translating the model by hand or by constructing a translator. One advantage when using the translator is the reusability. Instead of duplicating the modelling work the model can easily be introduced into RODON. A lot of time can be saved by using a translator instead of doing the modelling by hand. A disadvantage for the translator is that the procedure can not be made completely automatic. The fault models need to be modelled in the translated model. Though, if a framework on how to initiate fault modes is introduced, the human factor is decreased. The idea is to make a simple method for diagnosis by using the knowledge in the design department all the way out in the workshops where the diagnosis take place.

The translator is a flexible and fast way to have the models in RODON from the original Simulink models and is the method to be used in this thesis.

Chapter 5

Translation from Simulink to RODON

This chapter describes the translation process from Simulink to RODON.

5.1 Introduction

In Simulink, the models are based on signal flow modelling described in Section 4.3. This implies that diagnosis becomes complicated since a fault, e.g. a leakage, could change the causality of the model. Because of this, the Simulink model needs to be remodelled. In order to overcome this problem, it is desirable to translate the Simulink models into RODON models to be able to carry out diagnosis.

A third party translator that translates Simulink models to Modelica models already exists. When translating, the translator maps component in the Simulink library to corresponding components in a Modelica library. The Simulink library is seen in Figure 5.1. A corresponding library in Modelica also exists and is shown in Figure 5.2. The Modelica library is called AdvancedBlocks and has been developed by the third party. To be able to use the translator for translation of Simulink models into RODON models, an AdvancedBlocks library needs to be developed in RODON. Some modifications in the translator are necessary since the functionality of RODON and the functionality of Modelica are similar but not the same. The modifications are done by the third party and they consist of changes in the how the translator map components in Simulink to RODON components, i.e. how the component calls are written by the translator in the Rodelica code.

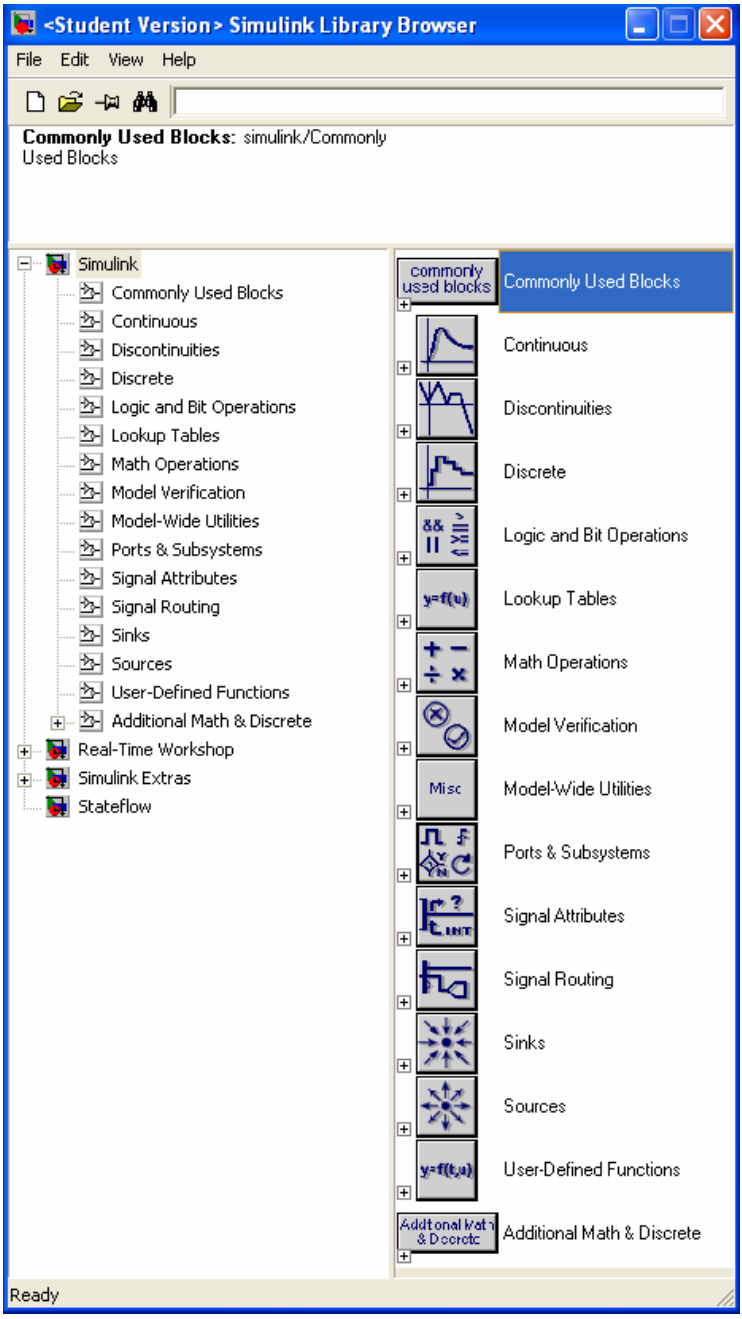


Figure 5.1: Structure of the Simulink library.

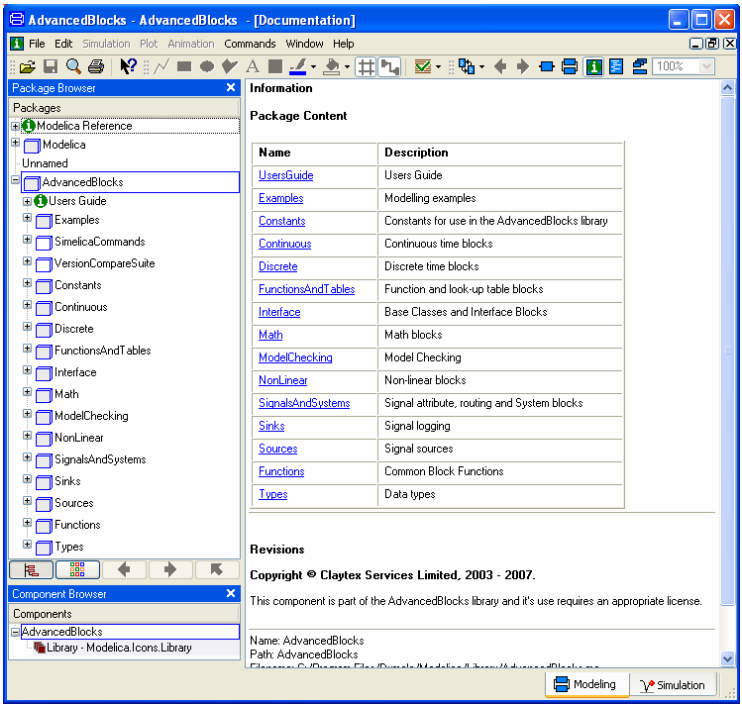


Figure 5.2: Structure of the AdvancedBlocks library in Modelica.

5.2 Levels of Translation

When translating the model, the translator could be working on several levels. In this thesis three options are considered, they are shown in Figure 5.3:

1. Map components in Simulink to corresponding components in RODON, for example map a compressor to a compressor.
2. Extract flat equations from the Simulink model.
3. Map logic components in the Simulink model to the corresponding ones in RODON.

Figure 5.3 shows the functionality of the three translator levels in the list. The second option, extracting flat equations from Simulink needs an explanation. From every Simulink model, it is possible to create an equation system from the blocks and the lines in the model. The equation system is the flat equations and it contains no information of the components. In the figure, one can see that the Simulink model has a fixed causality which is the normal behaviour when using Simulink. In RODON, the causality is normally not fixed. This is a great advantage when doing diagnosis. But, when translating the Simulink model, the fixed causality is also translated, and unfortunately the RODON potential is not completely used.

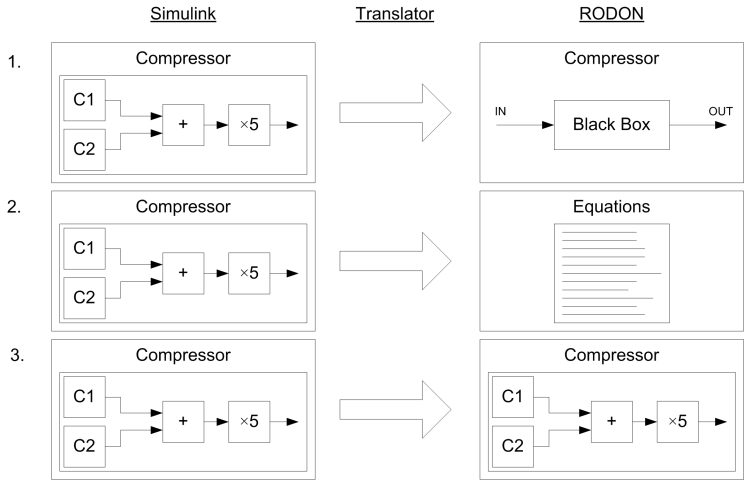


Figure 5.3: Different levels of translation.

Choice of Translation Level

The different options described in the previous section all have their benefits and drawbacks. The first one translates the physical behaviour on the top

level, for example the behaviour of a compressor. The physical behaviour consists of constraints from the law of physics. In Simulink, the blocks and lines in models create an equation system, for example like:

$$x_3 = f(x_1, x_2)$$

A question with a translator on this level is thus, how the equation system should be interpreted into a physical behaviour in components, for example a compressor. On the other hand, once the interpretation is done, such a translator should be appropriate since a RODON model normally describes the physical behaviour of systems. An example of this is shown in Section 3.1 where an electrical circuit is modelled. It is however a challenging task to create a translator that can interpret the physical behaviour of signal flow models such as Simulink models.

The second option might be the most straight-forward translator to create. Nevertheless, a question is how to use the equations for diagnosis. If for example the compressor is malfunctioning, how is that related to the equations and what equations are affected. This issue might be more complicated to solve than for the two other cases since equations are not related to a specific component. Since this thesis is looking for a method that works well for the whole design chain, this is not the best solution to choose.

As mentioned before, the third option affects the causality in RODON. Normally, the causality is not fixed in RODON, which is the base of how RODON performs its diagnosis. Consequently, this is an disadvantage with the third option. A great advantage though with the third option is that the translator must not interpret the behaviour of the system, but only transform the model of one programming language into another one. Another advantage is that there exists a translator from Simulink to Modelica (see Section 1.4). This translator is thus chosen to be used in this thesis.

5.3 Functionality of the Translator

The original AdvancedBlocks library in Modelica has a similar structure as the Simulink library. The procedure of the translation is that the translator first locates what kind of blocks there are in the Simulink model and then translates them into Modelica. This is done by mapping the blocks found in the Simulink model to the AdvancedBlocks library block. The components in the library are arranged in the same structure as in the Simulink library. The translator writes the Rodelica code which is valid for the models in the library. The translator writes code for every component to translate and if it does not exists in the RODON library it will appear as a faulty compoment in RODON.

The translator needs information on how the AdvancedBlocks library is constructed and the naming standard for the translation. The translator from Simulink to RODON works in the same manner as the translator from

Simulink to Modelica. It is an updated version of the Simulink to Modelica translator which is compatible with the developed AdvancedBlocks library in RODON. These updates were made by a third party that got information on how the translator should work with the library.

5.4 AdvancedBlocks Library

In this master thesis project we developed a library that is utilized by the translator when mapping components in Simulink to the corresponding ones in RODON. This library is called AdvancedBlocks and the naming standard is the same in RODON as in Modelica. The origin AdvancedBlocks library in Modelica has a similar structure as the one in Simulink to facilitate the understanding and development of the translator. In this section, some parts of the developed library are demonstrated. First, the structure of the AdvancedBlocks library in RODON is explained. Then, the matrix multiplication functionality and the integrator are shown, since they are both two important blocks in the engine model. The integrator is used to model the pressure and temperature changes in the engine. Last an icon functionality that facilitates the comprehension of the model is shown.

Structure

In this section, the structure of the AdvancedBlocks library is explained to give an understanding of how the modelling is implemented. First, an explanation to RODON and how its libraries are normally constructed, see Figure 5.4. A package is used to group models and is the base for the library structure. Packages are found on the top level in the libraries. Inside a package, it is possible to have other packages but also models, functions, blocks, connectors etc. Since Rodelica is an object oriented programming language, it is a good idea to let the library structure profit from it. Therefore, the models usually inherit properties from other models and can then be extended by adding further properties.

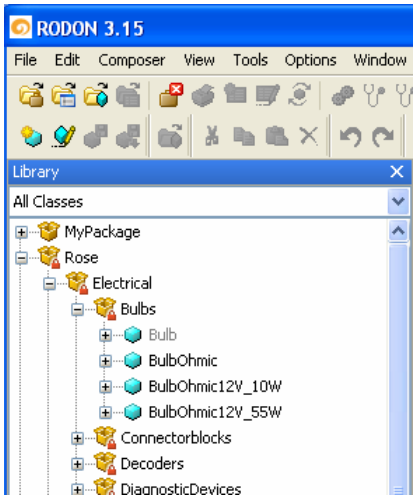


Figure 5.4: Rodelica bulb library.

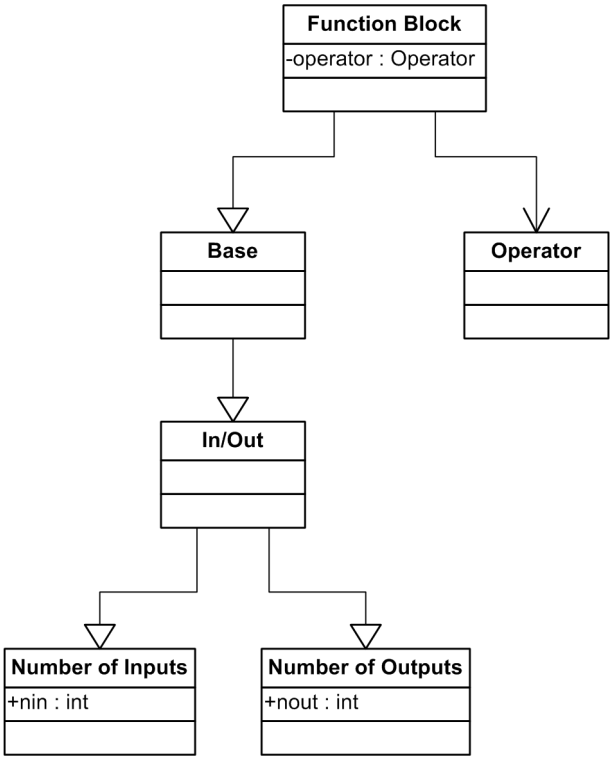


Figure 5.5: AdvancedBlocks library structure.

In Figure 5.5, the structure of the AdvancedBlocks library is shown. In the figure, UML standard is used. This means that the closed arrow symbolizes an inheritance and the open arrow symbolizes an execution of an operation. The inheritance goes from the bottom to the top, i.e. the *Base* block inherits properties and variables from the *In/Out block*. In the bottom of the tree, the number of inputs and outputs are defined. The function block at the top, e.g. an addition block, then inherits the number of inputs and outputs and other properties from the predefined In/Out block. To actually execute an operation, operator blocks are called. These operator blocks are possible to redeclare, meaning that the translator can overwrite the default operator declared in the function blocks. For example, the *Addition2* function block is desired to execute addition and subtraction operations between two values. The function call looks like following, where `Sum` is the name of the *Addition2* block:

```
AdvancedBlocks.Arithmetic.Addition.Addition2
Sum(redeclare model
Action=AdvancedBlocks.Arithmetic.Addition.Options.pm
,nin=Array[:,:]{{1,1},{1,1}})
```

The default operation is set to be `pp` which corresponds to a normal addition, whilst the `pm` operation will take a positive sign for the first input signal and a negative sign for the second one.

A problem with this structure is that the functionality of Simulink is hard to recreate. In Simulink, the number of inputs or outputs is easily changed by using a graphical user interface, according to Figure 5.6. For example in an adder block, one can choose between 1 to a large number of inputs. Whilst with the Rodelica structure, the number of inputs and outputs to use is fixed for a certain function, i.e. there is an *Addition2* block, *Addition3* block etc.

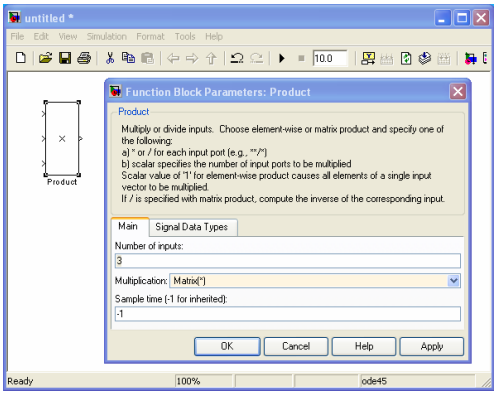


Figure 5.6: Matlab input functionality.

Matrices

The engine model contains a multiplication of three matrices. Just like in Simulink, the matrix multiplication is included in the *Product* block. The size of the input signals decides which multiplication to use, element-wise for one-dimensional input signals or matrix multiplication for multi-dimensional signals. In Simulink it is also possible to do element-wise multiplication for matrices, but this is not implemented in RODON yet.

For the multiplication of three matrices, a function which multiplies two matrices was implemented. This function, *MatrixMult*, can then be called several times. For three matrixes the function call in Rodelica looks like:

```
out1:= MatrixMult((MatrixMult(in1,in2,numberin)),
in3,numberin);
```

where `in1`, `in2` and `in3` are the input matrices and `numberin` describes the size of the matrices, i.e. the number of rows and columns. The multiplication is performed for the two first in matrices. The outcome of this call is then multiplied with the third input matrix, and a graphical illustration of this model is seen in Figure 5.7.

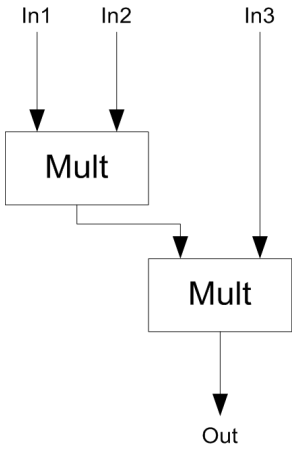


Figure 5.7: Functionality of the matrix multiplication block.

The function *MatrixMult* contains an algorithm for the multiplication of two matrixes. The algorithm is general for matrix multiplication and can be found in [1]. A and B are the matrices to multiply and their sizes are defined by n, m and p:

$$A_{n\times m}, B_{m\times p}$$

The algorithm has the following appearance:

$$M(i,j) = \sum_{i,j,k}(A_{i,k} \times B_{k,j}) \qquad i = 1:n, j = 1:p, k = 1:m \tag{5.1}$$

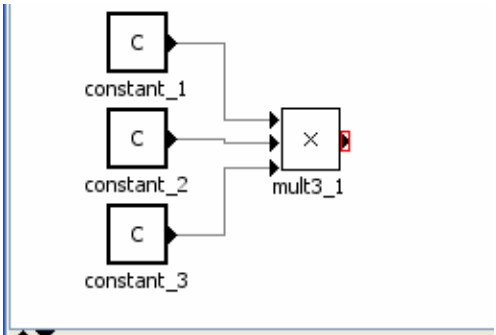
To verify that the matrix multiplication is performed correctly, a test model was created and can be seen in Figure 5.8. It is the result from test case number 5 in Table 5.1 that is displayed. The model contains three constants and a product block. The constants work as matrices since it is possible to change their output sizes, just like in Matlab, to vectors and matrices. In the figure, the result of the fifth multiplication is shown. *out1.m* and *out1.n* in the table defines the size of the output matrix where *out1* is the name of the output matrix. The *out1.signal* values correspond to the output matrix and are the result of the multiplication.

In the following table the result of the matrix multiplication is shown:

Table 5.1: Validation for Matrix multiplication

Input Matrices	Result	Status
2*3*4	24	OK
$2*\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}3\\4\end{bmatrix}$	22	OK
$2*\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}14 & 20\end{bmatrix}$	OK
$\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}3\\4\end{bmatrix}$	61	OK
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}37 & 54\\81 & 118\end{bmatrix}$	OK
$\begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}4\\5\\6\end{bmatrix} * 2$	64	OK

As seen in Table 5.1, the multiplication is executed correctly.



Variables	State 0.0
mult3_1.out1.m	2
mult3_1.out1.n	2
mult3_1.out1.signal[1, 1]	37
mult3_1.out1.signal[1, 2]	54
mult3_1.out1.signal[2, 1]	81
mult3_1.out1.signal[2, 2]	118

Figure 5.8: Matrix multiplication model.

Integrator

RODON does not handle dynamic systems. This means that there are no integrators implemented in RODON. To solve this, a discretization and hence approximation of an integrator needs to be done to simulate dynamic systems. In the engine model in Figure 1.1, integrators are used for modelling pressure and temperature. The integrators are defined as

$$\frac{1}{s}$$

which is the Laplace transform of the normal Riemann integral with an initial condition (y_0). The Riemann integral is defined as

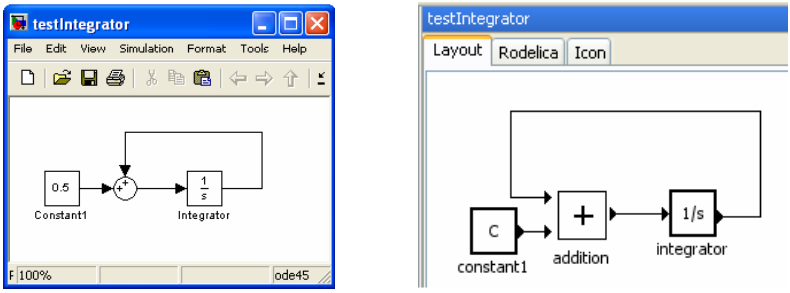
$$y(t) = y_0 + \int_0^a f(t)dt. \tag{5.2}$$

When the initial condition is set to zero the integral can be rewritten as a derivate

$$\dot{y(t)} = f(t) \tag{5.3}$$

where the derivate can be approximated using forward Euler method

$$\dot{y(t)} \approx \frac{y(t_{n+1}) - y(t_n)}{\Delta t} \tag{5.4}$$



(a) Model in Simulink.

(b) Model in RODON.

Figure 5.9: Models in MATLAB and RODON.

where Δt is the step size. The forward Euler approximation is not optimal. Because compared to other methods of approximation with the same step size, forward Euler is not so stable, nor very accurate. Especially stiff models need very small steps in order for forward Euler to be stable. The engine model is stiff with a lot of different time constants and therefore needs a small time step. Nevertheless, this method is used. The reason is that the only methods which can be implemented in RODON are explicit methods, as far as we know. The most basic explicit method is the forward Euler and that is the reason for us using it. There are some other methods such as Adams-Bashforth [6] but this method needs more known values and is more complicated to implement. Yet, the forward Euler method is implemented and the numerical error for an Euler forward approximation is [3]

$$d_i = \frac{\Delta t}{2}\ddot{y}(t_i) + O(\Delta t^2). \tag{5.5}$$

This means that the approximation is good when both the second derivate and the step size are small.

To start with, the step size is set to $10^{-4}s$. A small model with an integrator with an initial condition equal to 3, an addition block and a constant set to 0.5, was modelled in Simulink, see Figure 5.9(a), then translated into RODON, see Figure 5.9(b). The system was then simulated to verify whether the approximation is satisfying or not. Thus, Simulink is used for validation and is assumed to do a correct integration.

The simulation in Simulink was done with the ODE45 solver, which can be seen in the lower right corner of Figure 5.9(a). The result from the simulations is shown in Figure 5.10. It shows that the calculated integration in Rodelica is close to the one in Simulink and more or less follows an exponential function. In Figure 5.11, the difference between the two integrals is calculated and plotted.

According to this plot, the fault is increasing with time. This is comprehensible since the total error is the sum of all the small errors in Equation 5.5.

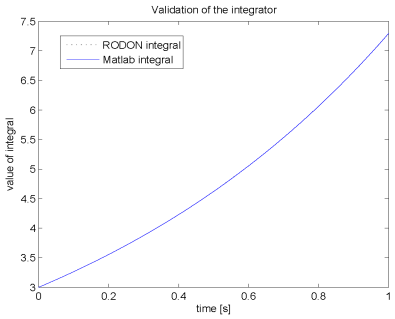


Figure 5.10: Simulation result of the integrator in RODON and Simulink.

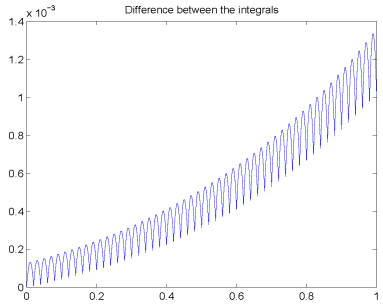


Figure 5.11: Difference between the RODON and the Simulink integrator.

The fluctuations in the figure originate most probably from the fact that the Simulink values have been interpolated with a linear interpolation. The interpolation is needed since it is desired to have the same amount of values as for the RODON simulation. This creates an error in the evaluation data from Simulink. With great probability, the local minimum points in the figure are the true values. To verify this, the analytical value of the integral is also calculated. In Equation 5.6, the differential equation for the system is written. The solution for the equation is found in Equation 5.7

$$\dot{y(t)} - y(t) - \frac{1}{2} = 0 \tag{5.6}$$

$$y(t) = -\frac{1}{2} + \frac{7}{2}e^t = 0. \tag{5.7}$$

The difference between the analytical value of the integral and the result from RODON is displayed in Figure 5.12. As can be seen, the values correspond to the minimum points above. The absolute error is 10^{-3} in magnitude.

This results in an relative error around 10^{-4} which is sufficient. If higher accuracy is required, a reduction of the step size is needed. Though, a smaller step size increases the simulation time. Therefore, an adjustment between time and accuracy is needed for the integrator. When validating the translated model it is necessary to consider whether a modification of the step size is needed or not.

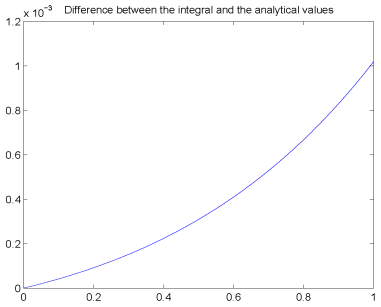


Figure 5.12: Analytical error of the integrator in RODON.

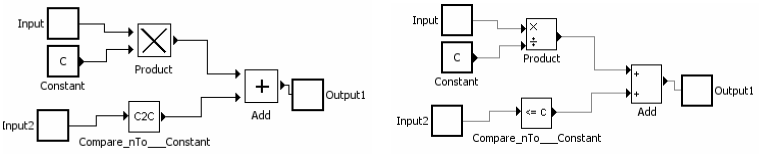
Icons

One important requirement regarding the translator is that the result should be visible. When a model is translated one should be able to see what kind of blocks it contains as well as their functionality. One problem that came up during the development process concerned this requirement. As described in Section 5.4, the models performing for example an addition can be redeclared meaning that the performed operation is changed from perhaps an addition to a subtraction. When having a model with e.g. 10 adders, it is desirable to be able to see what kind of operation that is executed in the different blocks. Thus, the icons should change their appearance according to this operation. When doing the translation the icon was fixed and had the same look even though the operation was changed, this was of course a big issue.

To solve this, a feature in RODON called Animated Icons was used. This feature was first supposed to be used when diagnosing for example electrical systems, where it should be possible to see e.g. if the light is lit or not. Depending on the status of the component the icon will change appearance.

To profit from this animated feature, an additional variable was added to the components in the AdvancedBlocks library. In every operation block belonging to a certain model, e.g. Addition2, this variable *IconNr* was given a different number. So when this operation block is called, the variable will be set to a number corresponding to a certain icon. Though, this feature is only seen in Analyzer mode and not in the Composer mode, see Section 3.3. The reason for this is that it is first in the Analyzer mode that all the variables

are instantiated and receive their proper value. To overcome this, a default icon is chosen for the Composer mode for every model. The default icon is supposed to be general for every model and the choice of the icon is therefore based on this fact. This is shown in figure 5.13(a) where the icons are general. For example, the C2C icon is general for a logic compare to constant and it is not possible to tell whether the comparison will compare the constant with greater or smaller values. When going into the Analyzer mode the correct icon will appear. In Figure 5.13(b), the model is now in Analyzer mode and the correct icons appear.



(a) Composer mode. (b) Analyser mode.

Figure 5.13: Model in Composer mode and in Analyser mode.

Discussion

The developed AdvancedBlocks library has its benefits and drawbacks. An advantage is that the translation from Simulink is now possible. After a translation, the blocks implemented in RODON can be used for simulation and diagnosis, and according to the validation the desired behaviour is achieved. Though, there is an issue concerning the update of the library. When having models in Simulink where the corresponding blocks in RODON are implemented, the translation is not a problem. However, for Simulink models where not all the blocks are implemented, the usability after the translation is affected. The translator manages to write the Rodelica code for unimplemented blocks, but RODON cannot handle them. In other words, the code written by the translator is correct, but since there are no such blocks in RODON an error will occur. This seems natural since the block does not exist in RODON, nor does the behaviour of the block. Regrettably, this puts demands on Simulink models for translation. The blocks must all be implemented in RODON; otherwise the model will fail. A solution for this is to have an empty default model which is called by RODON when no other block is found. In that way, the model can be simulated and diagnosed even though not all blocks are implemented, but the correct behaviour for the model will not occur. Unfortunately this affects the reusability of the translator negatively which is normally a positive characteristic of the translator. On the other hand, the most common blocks are implemented in the library. Unimplemented are for example model verification blocks and user defined functions, i.e. not so common blocks. In general, most models can be translated. Models

explaining a physical behaviour using the laws of physics can be translated. However, for logical systems and signal processing systems there are blocks missing.

Another issue is that the causality of the implemented blocks is fixed. This is a disadvantage since RODON usually uses the non-fixed causality when diagnosing. The reason why the blocks were implemented in this way is that they should capture the Simulink behaviour. In Simulink, the causality is fixed and then to get the same behaviour in RODON, this implementation method was chosen. This will unfortunately affect the implementation of fault models which will be seen in Chapter 6.

As described in Section 5.4, Simulink is a flexible tool when it comes to handling the number of in and out ports of a block. The structure of the AdvancedBlocks library is not as flexible as in Simulink though. A block for every number of input or output performing the same operation needs to be implemented. This is manageable for a low number of inputs. It is however not the best solution to have for example 25 different addition blocks instead of one in which it is possible to change the number of inputs like in Simulink. A question is though how many Simulink models that have an addition block with for example 25 inputs. This is left for the future development of the AdvancedBlocks and also the development of RODON. A good thing about the translator is that the translation process is easy, and no knowledge about modelling is actually needed for a user of the translator, given an implemented Simulink model. Although, to be able to diagnose the system of the translated model, failure modes need to be implemented in either Simulink or in the RODON model. The AdvancedBlocks library is complete in the sense that it can be used with the translator to get the desired model into RODON. To sum up, the translation is possible to carry out.

5.5 Case Study: The Engine Model

The model introduced in Section 1.4 describes the entire air system of a turbocharged engine. The air system goes from the air filter to the exhaust pipe where the emissions come out after the catalyst and can be seen in Figure 5.14. It is a *mean value engine model* which is defined in [5] as:

Definition *Mean value engine models are models where the signals, parameters and variables are averaged over one or several cycles.*

The air filter is a part of the intake system and can be clogged just like the intercooler and the air mass flow sensor. The entire engine consists of pipes of different kinds and they can all get a crack or a hole. In addition, an engine has a lot of sensors and actuators which are used for control and diagnosis. Since the purpose is to investigate a method for combustion engine diagnosis in RODON, the first components of the air intake system are used. The chosen components are the air filter and the compressor with models of their control

volumes, see Figure 5.14. Doing like this, the concept of faults and the use of RODON can still be demonstrated. In the chosen system, there are four states coming from the four integrators that are used to model temperature and pressure. The integrators are found in the models of the control volumes. In each control volume, there is an integrator for the pressure and one for the temperature, which gives a total of four integrators. In the selected engine part, there are normally three sensors measuring the ambient temperature, the ambient pressure, and the air mass flow into the compressor.

The Simulink model in Figure 1.1 is assumed to be correct and it will be used for validation.

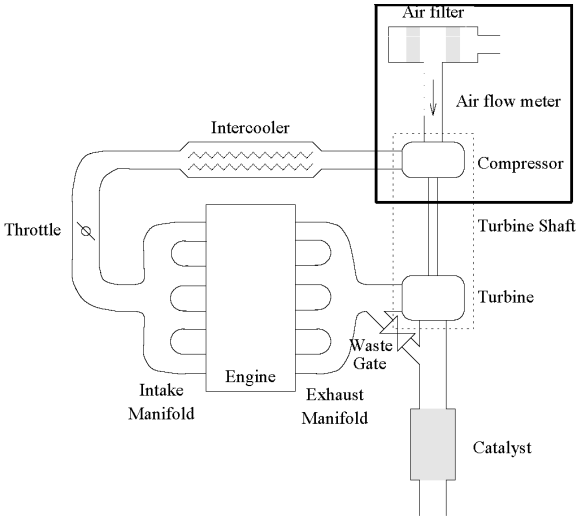


Figure 5.14: A sketch of a Turbo-charged SI engine.

Functionality of the Model

The Simulink model of the engine is constructed for the purpose to control the engine [2]. More exactly the model is built to control the air-fuel ratio in order to reduce the emissions. The air-fuel ratio depends on the amount of air in the cylinder. Since there are no sensors measuring the air going into the cylinder, a model is needed to estimate the air charge in the cylinder based on information from other sensors. The model was developed to introduce a flexible framework for cylinder air charge estimation that can easily be modified to different engines. This framework consists of an engine library from which the model is built. The model represents the nominal behaviour of the engine and in this thesis the model is used for diagnosis. This means that the model is first translated, and then fault models are introduced and after that

diagnosis is performed on the model by using RODON to find the possible faults that may have occurred in the engine.

Translation of the Model

Since the model of the engine is built by using the engine library, the model needs to be rebuilt with normal subsystems in Simulink. The reason for this is that the AdvancedBlocks library only contains the normal Simulink library seen in Figure 5.1.

The model to translate is shown in Figure 5.15. It is the first four components from the original model seen in Figure 1.1. The translated model is shown in Figure 5.16. The structure is the same and the subsystems contain the same information. In Figure 5.17 and 5.18, the model for the control volume for the compressor is shown. They look alike and that is the idea. It is expected to be effortless to understand the behaviour by looking at the RODON model.

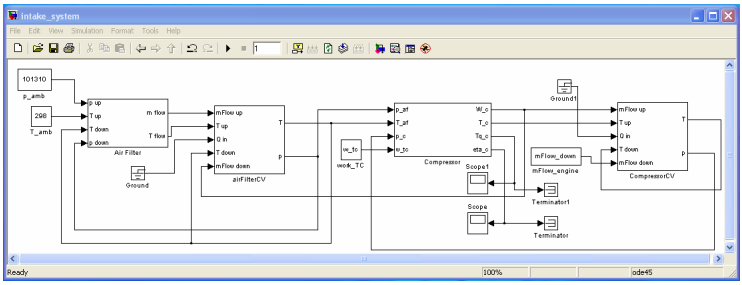


Figure 5.15: The Simulink model to translate.

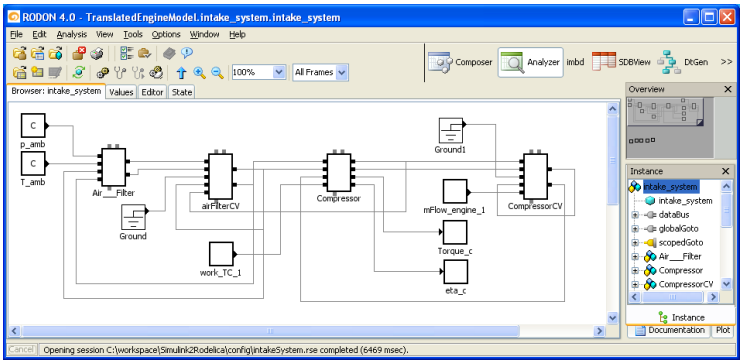


Figure 5.16: The translated RODON model.

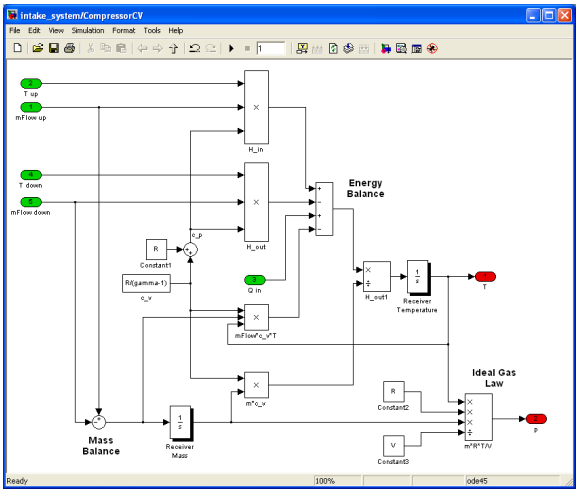


Figure 5.17: Model of compressor control volume in Simulink.

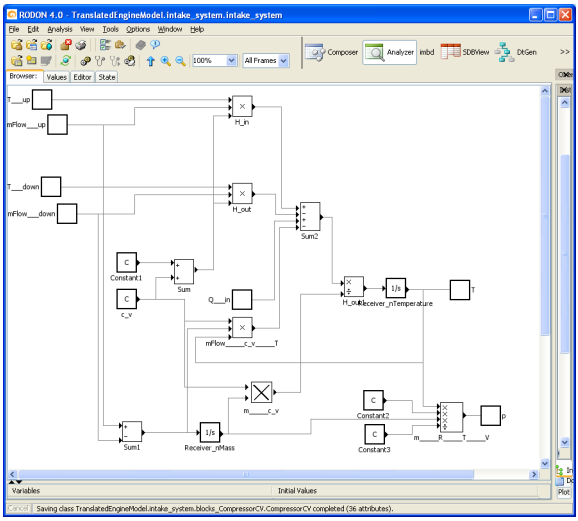


Figure 5.18: Model of compressor control volume in RODON.

This model is cut out from the complete engine model. To do simulations and diagnosis properly, it is necessary to have some of the values from the engine as input signals instead of the rest of the engine. The values in this case are the work produced by the turbine and the air mass flow computed in the intercooler model. Because of this, the two blocks *w_tc* and *mFlow_engine* are used as input signals to the model. The torque and the efficiency of the compressor, are signals going to the turbine shaft block in the complete model. In

this case, they are not interesting to study so therefore simple terminators are used. In RODON, the input signals are modelled as look up tables with time on the x-axis and value of the work or the mass flow on the y-axis. The data for the work and mass flow is generated by the original Simulink model in MATLAB.

Validation

The output signals of the integrators are used for validation. The plots below show the values from RODON and values from Simulink. The difference of the values from RODON and Simulink can not be seen, so the relative error is also plotted. The relative error is around 10^{-5} which is acceptable. When generating test data, the second was divided into 10 parts to simplify the handling of the data. This can be seen in the plots where the relative error changes a great deal every $0.1s$. To minimize the error it is possible to make smaller time steps in the integration, but it will affect the simulation time. It takes 4 hours to simulate 1 second in RODON for a step time of $10^{-4}s$ and a smaller step increases the time by a factor of ten for every power decreased. In this first version of the translator and the AdvancedBlocks library, 4 hours of simulation is ok. It is in the region of expectation according to RODON engineers. For future utilization further development is needed to decrease the simulation time, because 4 hours is quite a long time waiting in the workshop. For this model, when the error is this small it is not necessary to make smaller steps. Therefore we conclude that the translation from Simulink to RODON was carried out successfully.

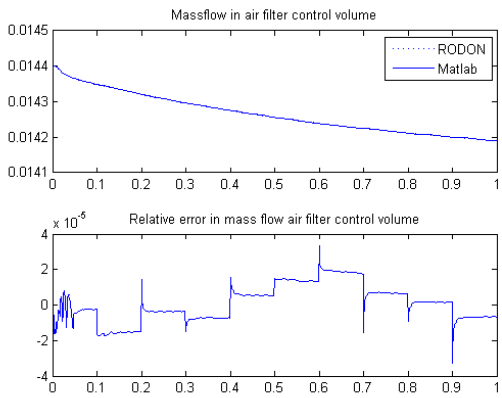


Figure 5.19: Validation of mass flow integrator in the air filter control volume.

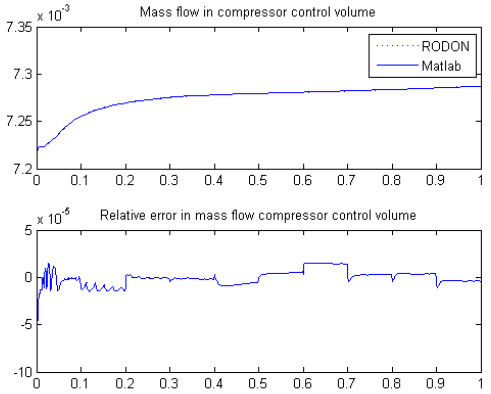


Figure 5.20: Validation of mass flow integral in the compressor control volume.

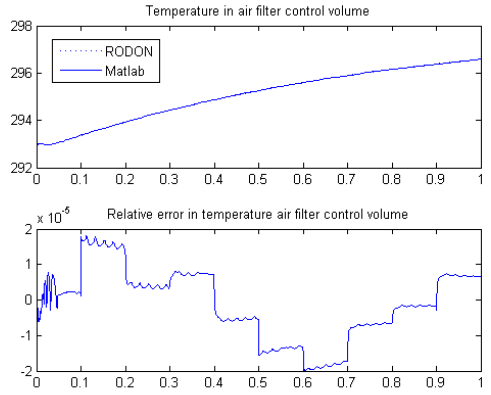


Figure 5.21: Validation of temperature in the air filter control volume.

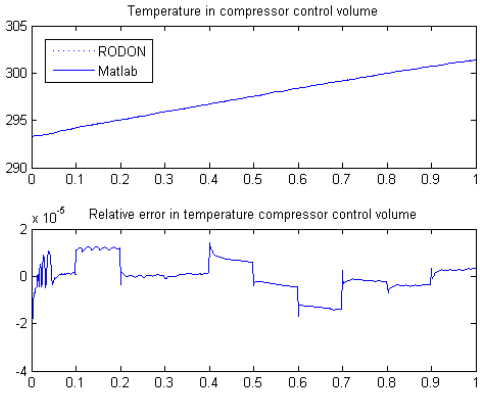


Figure 5.22: Validation of the temperature in the compressor control volume.

Chapter 6

Fault modelling in RODON

This chapter treats the diagnosis part of the thesis. In Section 5.5, a model of a intake system was translated from Simulink to RODON. In this chapter, possible faults that can occur in this part of the engine are discussed as well as the question of how to choose and implement the faults for diagnosis. Knowledge about the combustion reaction and λ measuring in an engine as well as basic knowledge about combustion engines are prerequisites of this chapter.

6.1 Faults in the Air Intake System

In the selected engine part in Figure 6.1, there are several sensors measuring temperature, pressure, and air mass flow. For our diagnosis, we are using the air mass flow sensor after the air filter which is shown with the *AS* in the figure. The pressure sensor, *PS* in the figure, measuring the pressure after the compressor is also used. In the selected engine part and the system around the engine, there is a large number of faults that can occur. Faults are defined and described in Section 2.4. Some faults may be possible for a human to discover since the car will behave in a non-normal way, others may be more difficult to discover without the help of a computer system. There are four main types of faults in the air intake system of a combustion engine; sensor faults, actuator faults, leakage, and clogging. The motive of this thesis is to examine a method that connects design departments to workshops where faults are found. The faults studied in this thesis are sensor fault, clogging, and leakage. The air flow sensor is modelled after the air filter where it is positioned in a real engine. Clogging is modelled in the air filter where it is common that clogging occurs since it filters the air from dirt. Leakage is modelled in the pipe after the compressor, as a hole in a pipe causes a leakage. The pressure after the compressor is higher than the ambient pressure during normal operation [13]. This means that if a leakage occurs here, air will flow out of the air pipe and the pressure is reduced. Therefore the pressure sensor

will be used to detect this fault.

In Section 5.2, the fixed causality problem when using the translator was discussed. The fixed causality is an issue that we needed to take into consideration when doing the fault modelling. Because if this, we decided to do simple fault models to verify whether engine diagnosis is possible in RODON. The fault modelling is done on output variables in the model and compared to sensor values which gives drawbacks that are discussed in the following sections.

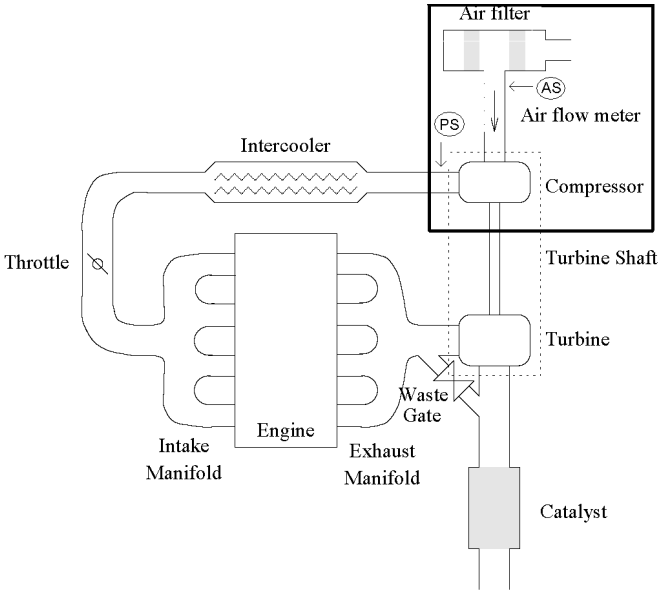


Figure 6.1: A sketch of a Turbo-charged SI engine.

6.2 Faults Impact on the Engine System

If a sensor fault occurs, the control system of the engine does not get the correct information which means it can not control properly. This might lead to high emissions or total failure of the engine. For example, if the air mass flow sensor constantly measures a too high value, then the control system injects more fuel to control the air-fuel ratio. This means that the real value of λ is below one, and there is not enough air for a complete combustion. As consequence, unburned hydrocarbon molecules and carbon monoxide exits the cylinders. After a while the catalyst can not handle any more of those and they go directly out into the environments. On the other hand, if there is a clogging or a leakage, less air could enter the cylinders than measured, not certain since the engine is a closed system with feed-back. An effect is

that NO_x is released into the catalysts and finally comes out into the air and pollutes the environment. If a complete clogging occurs, there will be no air in the cylinder and therefore no combustion can take place which means that the engine stops.

6.3 Different Implementation Methods

There are different methods for the implementation of faults. One way is to do the implementation directly in Simulink. However, this means that the model might be completely remodelled as discussed in Section 4.3. Another option which does not affect the original model is to model the faults and introducing them into the nominal Simulink model via a switch. After introducing the faults in Simulink, they are translated together with the nominal model. This is not optimal since the failure modes which contribute to the RODON diagnosis do not exist in Simulink and therefore manual modifications after a translation need to be done. Another way to implement the faults is to take the translated model and make fault models as blocks in a RODON library. The faults are modelled as parameters which have failure modes. This is advantageous since the faulty behaviour is described directly with a physical behaviour in the fault model. Although, this means that a lot of modifications needs to be done after a translation. On the other hand, a fault model can be reused, for example a sensor fault model can be used for several sensors in the engine.

6.4 Implementation

As said before, three types of faults have been studied; sensor faults, clogging, and leakage. The way of implementation that we chose, compares simulated sensor values obtained using the model to real sensor values. The simulated sensor values are computed by RODON from the engine model. Instead of modelling faults inside components, they were modelled in separate fault models which facilitate the reusability. The fault model library can be seen in Figure 6.2. The reusability means that the same sensor fault model can be reused in different places in the model to describe different sensors. It is just to drag and drop the wanted fault model in its right place and connect the input and output connectors.

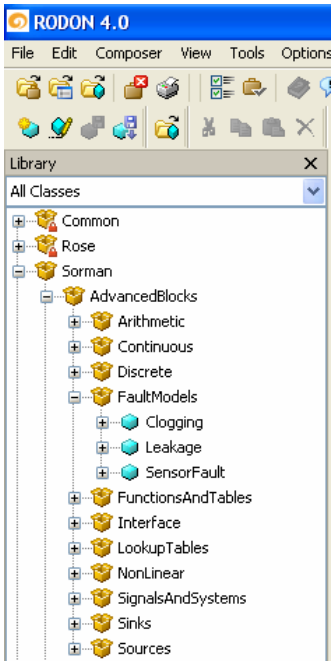


Figure 6.2: Fault models in the AdvancedBlocks library.

Sensor Fault

First, the implementation of the sensor fault is treated. The idea is that in case of no fault, the sensor value should be the same as the simulated value. When doing like this, it is assumed that the model is correct and corresponds to true values in case of no fault. For modelling this behaviour, an additive fault model is used,

$$y_s = f + y_r$$

where y_s is the real air flow sensor value, y_r the simulated sensor value in RODON, and f the fault. If a fault corresponding to f occurs, which means $f \neq 0$, it is desirable that RODON detects it, i.e. can determine that $f \neq 0$. A sensor fault in this thesis is defined as a sensor not measuring the correct value. The Rodelica code for the sensor fault model looks like following:

```
1:  model SensorFault
2:    import Rose.Interfaces.BehavioralModes.*;
3:    import Sorman.*;
4:    extends
5:    AdvancedBlocks.Interface.IOLayers.I1_O1;
6:    parameter Integer[:,:] nin=Array[:,:]{{1,1}};
7:    parameter Interval tolerance = [-0.01 0.01];
8:    public Interval fSensor; //the fault signal
9:    public Real mDotSensor;// signal from sensors
10:   // TranslatedEngineModel.TimedData mDotSensor
11:   FailureMode    fm(max=2, pm="fm",
12:   mapping = "ok=0, pos_shifted=1,
13:   neg_shifted=2,");
14:  behavior
15:
16:   out1.signal[1,1] = in1.signal[1,1];
17:   fSensor = mDotSensor - in1.signal[1,1];
18:
19:        if(fm == 0){ //OK-case
20:            fSensor = tolerance;
21:            }
22:        if(fm==1){ //positive shifted case
23:            fSensor >= tolerance;
24:            }
25:
26:        if(fm==2){ //negative shifted case
27:            fSensor <= tolerance;
28:            }
29:
30:  end SensorFault;
```

The needed variables are defined in the first lines of the code. `nin` defines the size of the input signal, `fSensor` is the unknown fault signal. When simulating in RODON, and using time as variable, `fSensor` value is computed in each time step. In this test case, the variable `mDotSensor` on line 9 is used as sensor data since it is possible to change it manually in the Analyser mode. However, when inserting this model into the engine model, the commented variable on line 10 is used instead since it is a variable which collects data from a data file, i.e. sensor values. In case of no fault, the real sensor value and the model sensor value should be the same. However, this is unlikely in a real engine and therefore the variable `tolerance` is used on line 7. If the fault signal `fSensor` is in the tolerance interval, it is defined as no fault. The input and output signals are created on line 4 and 5, where the `extends` command means that our sensor model inherits the properties of the In/Out model. `I1_O1` stands for one input and one output signal.

In the sensor fault model the output signal is equal to the input signal, seen on line 16. The reason is that the fault model should not affect the engine model, just be there to check if the sensor is ok. As seen in lines 19 to 21, a well functioning sensor means that fSensor is in the given tolerance interval. If the sensor is positive shifted, the output from the sensor is greater than it should be. This implies that `fSensor` becomes greater than the tolerance. For the negative shifted case, the opposite scenario occurs.

When the sensor fault has been implemented, it is necessary to check if the desired behaviour is achieved. For this, three test cases were created. The first one is seen in Figure 6.3. The real sensor value was set to 0.5 and the constant value to 0.505. The constant output becomes input signal in the fault model. The tolerance interval is $[-0.01 + 0.01]$. Thus, this is an ok case. As seen in the figure, the outcome of the RODON diagnosis is that the system is OK. Consequently, the fault free case works.

In the second test in Figure 6.4, the negative shifted case is tested. The sensor value is still the same, i.e. 0.5, and the tolerance interval is also the same. This time the constant is set to 0.55. Hence, the sensor value is much smaller than the model value. The figure shows the outcome of the diagnosis which is that the sensor is negatively shifted. Accordingly, also this diagnosis works.

For the third case, a positively shifted sensor, the result is seen in Figure 6.5. Also this time, the diagnosis result is as expected. The conclusion of the three test cases is that the sensor fault model works as expected.

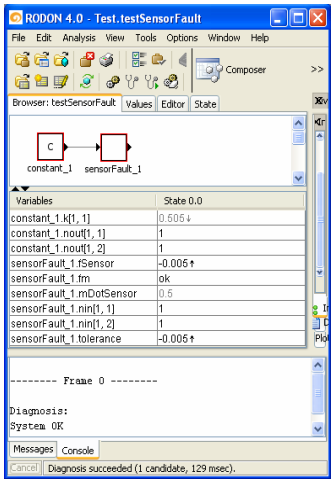


Figure 6.3: Diagnosis result when no sensor fault.

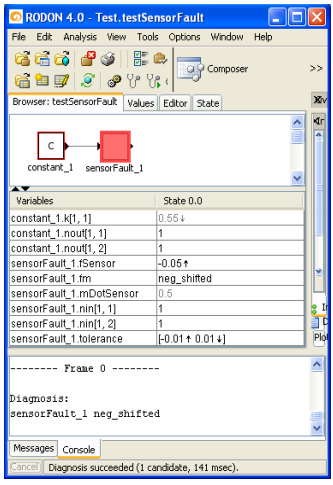


Figure 6.4: Diagnosis result for negatively shifted sensor fault.

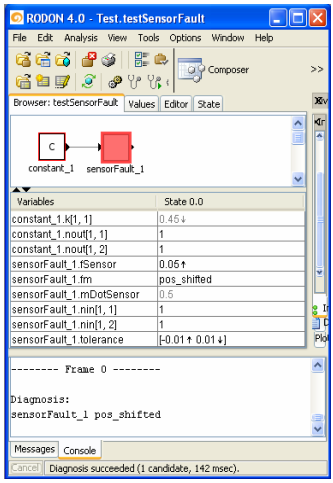


Figure 6.5: Diagnosis result for positively shifted sensor fault.

Clogging Fault

The next fault to implement is the clogging. The causalty problem is present here as well and the choice of implementation method for the clogging fault is affected by this. Therefore, for the clogging fault modelling, a sensor value will be compared to a value computed from the model. When a clogging occurs, the air flow in the engine will be limited since there is a blockage. The real air flow is thus smaller compared to the air flow value calculated from the model. The real value divided by the model value is therefore in the $[0\ 1]$ interval. Because of this, a parametric fault model was used to implement the fault,

$$y_s = f y_r$$

where y_s is the real sensor value for the air flow, y_r the simulated sensor value in RODON, and f the fault parameter. In case of no fault, y_s and y_r are equal and the fault parameter f is then 1. In case of total clogging, y_s is 0 and f is then also 0. For partly clogged pipes, f is in the $[0\ 1]$ interval. This clogging fault model is a simple model. It only considers a sensor value and does not care about other affects on the engine in case of a clogging, for example how the states in the engine changes. The Rodelica code for the clogging fault model is:

```
1:  model Clogging
2:    import Rose.Interfaces.BehavioralModes.*;
3:    import Sorman.*;
4:    extends
5:    AdvancedBlocks.Interface.IOLayers.I1_O1;
6:    public
7:    parameter Integer[:,:] nin=Array[:,:]{{1,1}};
8:    // this is the intervals in which the
9:    // fault parameter can be
10:   parameter Interval nominal = [0.95 1.101];
11:   parameter Interval small  = [0.7 1.101];
12:   parameter Interval big = [0.001 1.101];
13:   parameter Interval total = [-0.001 1.101];
14:
15:   // the fault we wish to find, in this
16:   // case a fault in the flow = clogging
17:   Interval fFlow;
18:
19:   FailureMode    fm(max=3, pm="fm", mapping =
20:   "ok=0, small clogging=1, big clogging=2,
21:   total clogging=3");
22:   Real mDotSensor;  // Sensor value
23:   // TranslatedEngineModel.TimedData mDotSensor
24:
```

```
25:  behavior
26:
27:   fFlow = mDotSensor.out1.signal[1,1]/
28:   in1.signal[1,1];
29:   out1.signal[1,1] = in1.signal[1,1];
30:
31:         if(fm == 0){ //OK-case
32:             fFlow = nominal;
33:            }
34:         if(fm==1){ //partly clogged pipe
35:             fFlow = small;
36:             }
37:         if(fm==2){ //clogged pipe
38:             fFlow = big;
39:             }
40:         if(fm==3){ // totally clogged pipe
41:             fFlow = total;
42:             }
43:  end Clogging;
```

Since sensor values normally vary even in a nominal case, the nominal interval on line 10 is around 1, ideally it would be equal to 1. Like for the sensor fault model, the out signal is equal to the in signal, seen on line 29. This is not an optimal behaviour for a clogging fault model since a clogging should affect the state of the system. As said before, the reason is the fixed causality in the air intake system model after the translation which was performed in Section 5.5.

The validation of the clogging fault model is performed in the same way as for the sensor fault. First, it is verified that the nominal case is working, this is seen in Figure 6.6. The sensor value is set to 0.5 and the model value is set to 0.505. This gives a quotient of 0.99 which is in the ok interval. The diagnosis result is that the system is ok, like expected. In Figure 6.7, the small clogging case is tested. As seen in the figure, the sensor value is still 0.5 but the model value is set to 0.55. Accordingly, the quotient is now in the small clogging case which is also seen in the figure. So far, the clogging fault models works as expected.

For the big clogging, the diagnosis result is shown in Figure 6.8. This time the quotient is 0.6667 which is in the big clogging interval. In Figure 6.9, the total clogging case is tested. The difference between the model value and the sensor value is very large and causes the quotient to end up in the total clogging interval. There were four failure modes implemented in the clogging model, and they are all functioning like expected.

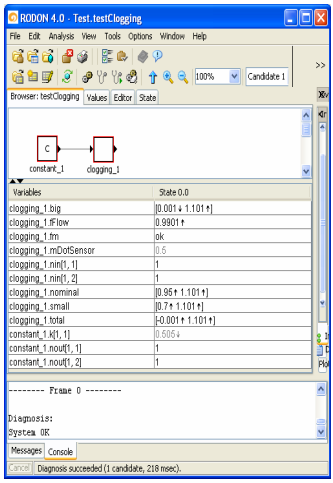


Figure 6.6: Diagnosis result when no clogging.

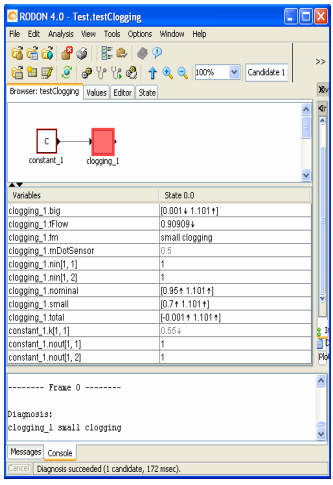


Figure 6.7: Diagnosis result when small clogging.

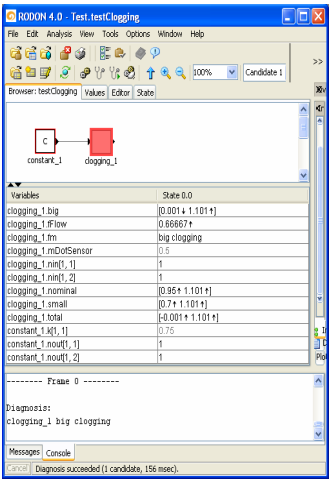


Figure 6.8: Diagnosis result when big clogging.

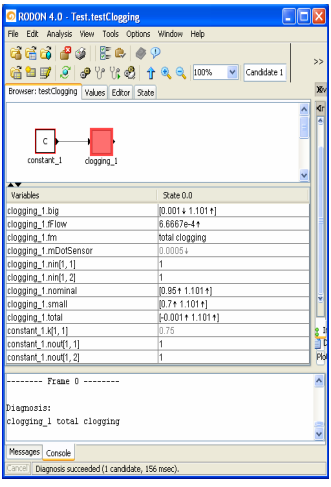


Figure 6.9: Diagnosis result when total clogging.

Leakage Fault

The third fault model to implement is the leakage. In case of a leakage, the pressure is affected since a hole lets air out. To model this, a parametric fault model is used, like for the clogging,

$$y_s = f y_r$$

where y_s is the pressure sensor value, y_r the pressure model value and f the fault parameter. In a fault free case, f is 1 and if a leakage occurs f is in the $[0\ 1]$ interval. Like for the clogging fault model, this model should affect changes in states in case of a leakage. The textual representation of the leakage fault model in Rodelica is:

```
1:  model Leakage
2:    import Rose.Interfaces.BehavioralModes.*;
3:    import Sorman.*;
4:    extends
5:    AdvancedBlocks.Interface.IOLayers.I1_O1;
6:    public
7:    parameter Integer[:,:] nin=Array[:,:]{{1,1}};
8:    // Fault parameter intervals
9:    parameter Interval nominal = [0.99 1.01];
10:   parameter Interval fault = [0.7 1.01];
11:   parameter Interval bigfault = [0 1.01];
12:
13:   // The fault parameter
14:   Interval leakage;
15:   FailureMode fm(max=2, pm="fm", mapping =
16:   "ok=0, leakage=1, big_leak =2");
17:   Real PressureSensor;  //Sensor value
18:   //TranslatedEngineModel.TimedData PressureSensor;
19:  behavior
20:
21:   leakage = PressureSensor/in1.signal[1,1];
22:   out1.signal[1,1] = in1.signal[1,1];
23:
24:   if(fm == 0){ //OK-case
25:            leakage = nominal;
26:           }
27:   if(fm == 1){ //faulty case
28:            leakage = fault;
29:           }
30:      if(fm == 2){ //faulty case
31:            leakage = bigfault;
32:           }
```

```
33:
34:  end Leakage;
```

Considering the validation, the fault free case is first tested and is seen in Figure 6.10. The quotient of the sensor value and the model is 0.999 which is in the ok interval and gives the ok diagnosis result. In Figure 6.11, a leakage case is demonstrated. Since the sensor value and the model value differ too much, the expected RODON conclusion is that a leakage has occurred and this is also the diagnosis result. A big leak causes a large pressure drop, this has been simulated in Figure 6.12. The anticipated diagnosis result is a big leakage. To conclude, the leakage fault model behaves as expected.

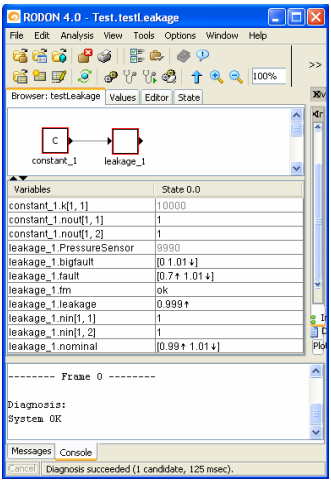


Figure 6.10: Diagnosis result when no leakage.

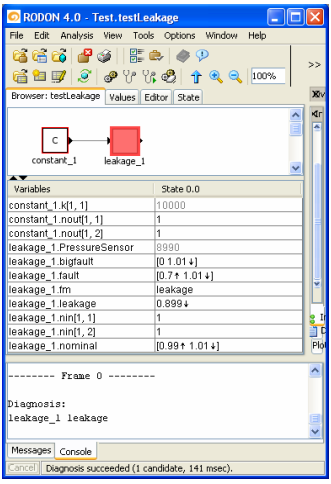


Figure 6.11: Diagnosis result for a leakage.

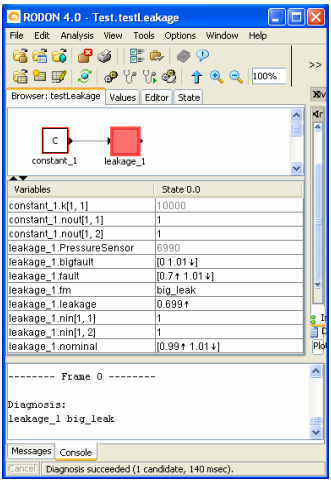


Figure 6.12: Diagnosis result for a big leakage.

6.5 Discussion

The fault models treated in this chapter are all simple models. The meaning of simple here, is that the fault models only compare values computed by the model to sensor values. In case of a fault, especially clogging and leakage, the states of the system should be affected. This is not handled by the current fault models. When choosing how to implement the faults, this problem occurred in our minds. However, the present solution was chosen because of the fixed causality in the AdvancedBlocks library. The reason is that the given Simulink model describes the nominal behaviour. To model an affect on the states would imply a change in the model, either done in Simulink or in RODON. For example the leakage fault model could be modelled like [13]:

$$m_{boostLeak} = \frac{k_b p_b}{\sqrt{T}} \Psi(\frac{p_{amb}}{p_b})$$

where k_b is proportional to the leakage area. The model for the whole air intake system should then be updated by:

$$m = m_{th} + m_{boostLeak}$$

instead of

$$m = m_{th}$$

like in the nominal model. In Simulink this would require building the $m_{boostLeak}$ constraint by blocks and connecting them. In RODON, it could be done either by using the same technique as in Simulink or by typing equations in the Rodelica code. Additionally to typing equations, changes in the model would be required. However, when rebuilding the whole model to insert the faulty behaviour, the benefit of the nominal model is decreased since it needs to be modified a great deal. The purpose of a translator is to create a process which is as automatic as possible. If the nominal modelling needs to be adjusted, this feature is limited. Instead, we chose to compare sensor values to simulated values from the model to investigate if combustion engine diagnosis is RODON is possible at all. The fault models we created do not affect the nominal behaviour as such, they just need to be inserted in the model where a sensor would be in a real engine. If this method is possible, it is left for further development to improve the fault models.

As seen in the previous sections, the implemented fault models work in RODON. They have all been tested and fulfil the requirements. However, they have been tested in separate models where the test environment has been created to prove they are well-functioning. In the following chapter, the fault models will be introduced into the intake system model for verification and diagnosis in a larger system.

Part III

Model Based Diagnosis

Chapter 7

RODON Diagnosis on the Air Intake System

In this chapter, diagnosis of the combustion engine will be treated. The translated engine model from Section 5.5 will be used together with the fault models developed in Chapter 6 to verify that diagnosis can be performed. The performance of the fault models have been seen in Chapter 6, in this chapter it will be studied if they function in the translated air intake system model in RODON.

7.1 Data Generation

In the previous chapters, different diagnosis elements have been produced, like a model describing the nominal behaviour as well as fault models. In this chapter, we will use these elements for concluding diagnosis on a combustion engine.

The box in Figure 5.14 contains the part considered in this thesis. The model describes the air filter and the compressor as well as their following pipes. By using this part of the engine, it is possible to find different types of faults, for example sensor faults, clogging and leakage, as discussed in Chapter 6. The given model is a Simulink model and has been translated into a RODON model in Section 5.5. Since the purpose is to use the model for diagnosis, fault models have been modelled in RODON and validated on their own in Chapter 6. Here, we will introduce the fault models into the nominal engine model and use RODON to do the diagnosis and detect the faults.

For diagnosis, data is required. Since the Simulink engine model in Figure 1.1 is assumed to be correct and has been used for validation, it can also be used for data generation. Therefore, the data needed for simulation and diagnosis does not come from a real engine but from the Simulink model. We have not added any noise to our data coming from the Simulink model. In

real life, the data is not as good as the data we produced, and therefore added noise would have been a step towards the real life. Since our cut out system does not have any feed-back, it is a sensible system and noise would not give a good result. Therefore noise has not been added. Concerning the sensor values that are used for diagnosis, a sampling frequency of 100 Hz is desired because with that frequency, a sample time of $0.01s$ is simulated. This is assumed to be a number that could be used in real life for an application like this, for example in workshops. Values already produced with the ode s23 solver was used and we then picked out values every $0.01s$. This resulted in samples around $0.01s$ but not exactly and sometimes there were no measurement at all. We did some interpolation trying to get a good vector of sensor values, but simulating with it in RODON it caused conflicts. The reason is that there is no feed-back in our model, our cut out part is an open system. It was not the wanted result. If we use a sampling frequency of 1000 Hz it works, but this might be a too high frequency for a real sensor. Therefore we decided to use the vectors generated from the ode s23 solver as sensor values. How the data was generated in Simulink and introduced into RODON can be seen in Appendix B. The part of the model used in this thesis has been cut out from the given Simulink model and the translated RODON model can be seen in Figure 7.1. The data needed for simulation is therefore signals coming from the cut out part of the engine, as well as sensor data. In the figure, the two boxes called *work_TC_1* and *mFlow_engine_1* contains data that before came from other parts of the engine model and has now been generated in Simulink. The box called *SF*, and *sensorFault_1*, contains the fault model for the sensor. In the same way, the leakage fault model is inside the box called *L* and *leakage_1*. Sensor data from Simulink is coupled to the fault model boxes. This is done by having data files containing the sensor values. Since the boxes have different names, the data will be transferred to the boxes by having code lines like below in the data files:

```
leakage_1.PressureSensor.values[1, 1] = 0;
leakage_1.PressureSensor.values[1, 2] = 101000;
```

The result of this is that the sensor fault model and the clogging model use the same sensor value but they will be coupled to the model as two sensors, `sensorFault_1.mDotSensor` and `Air_Filter.clogging_1.mDotSensor`. Figure 7.2 shows the data for the pressure sensor after the compressor and shows both a nominal behaviour and a faulty behaviour. It looks like the values are constant, like it would be for the ambient pressure sensor, but they vary more if we would have zoomed in more.

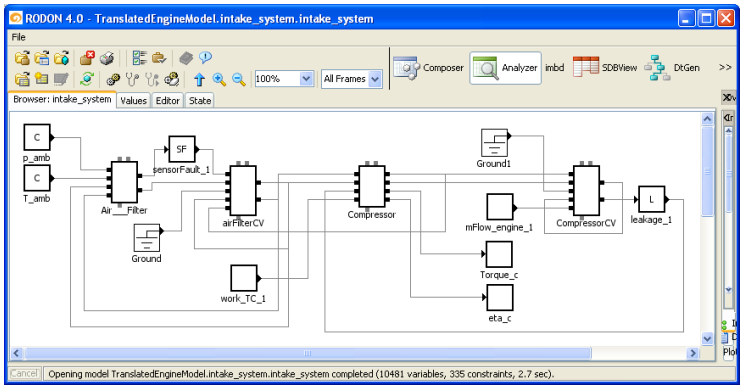


Figure 7.1: Model in RODON of the cut out intake system.

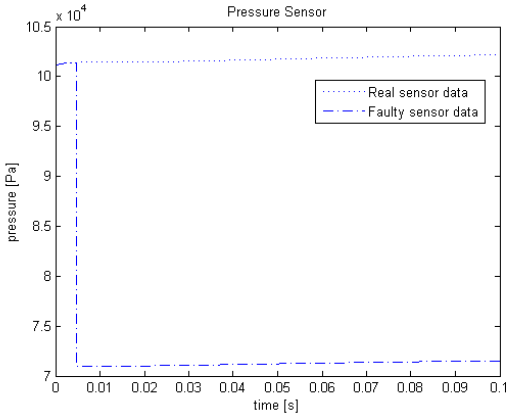


Figure 7.2: Pressure sensor data, generated from ode s23 solver.

7.2 Time Simulation in RODON

In Section 3.2, it was shown how RODON performs simulation and diagnosis by solving equation systems. With the generated data that we have and the integrators in the model, a dynamic simulation is needed where the time is increased by small time steps. Like described in Section 3.2, a dynamic simulation is only performed until a conflict is detected and then the simulation stops. This means that the diagnosis is only executed on the simulated data since there are no values computed after that time. Consequently, it is not possible to get different candidates at different times, only candidates at the time of the discovered conflict are received.

7.3 Fault Free Case

In Section 5.5, the engine model has been translated and validated through simulation. This model describes the nominal behaviour of the engine. Therefore fault models have been implemented in Chapter 6 and then introduced into the model in this chapter. In Figure 7.3, the simulation and diagnosis result of a fault free case is shown. The fault models should not affect the nominal behaviour of the model. As can be seen in the figure, the system is OK. We can therefore conclude that the model still behaves accurately.

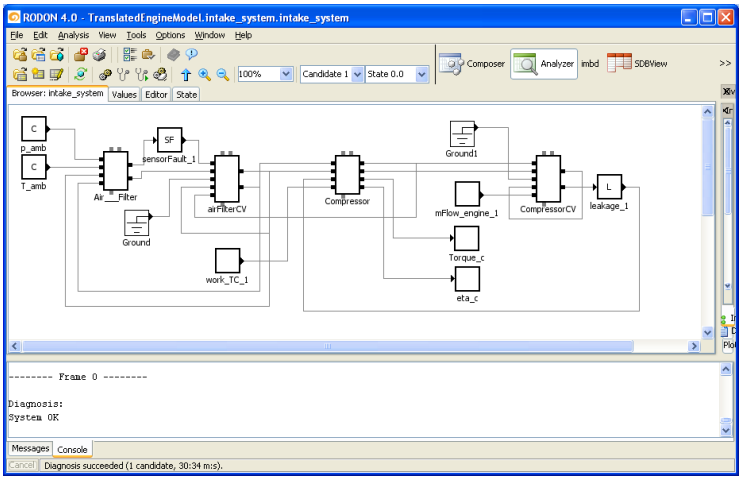


Figure 7.3: Diagnosis result for a fault free case.

7.4 Sensor Fault

In this section, the sensor fault is studied. In the fault model, a sensor value is used to find the sensor fault. When generating the sensor data in Simulink, it is possible to afterwards add for example 30% to the value to introduce a fault. The data generation was done and the data was used together with the model in RODON. A 30% fault was introduced at a time of $0.046s$. This is early but it is assumed that the initial states are known and then the system does not need time to adjust the initial values. The sensor values were decreased to demonstrate a negative shifted sensor fault, i.e. that the sensor measures a too low value. This was however not done for the sensor value for the clogging model and therefore only a sensor fault is expected to be discovered. Figure 7.4 shows the result of the simulation and diagnosis. The sensor fault model is highlighted which means that it contains a conflict. In the bottom of the figure, a candidate of the diagnosis is given. The first conflict in the figure stands for the discovered conflict during simulation. The two conflicts under `Frame 0` are the result of the diagnosis. The first one corresponds to *failure mode* 0, i.e. no sensor fault, since that failure mode is first checked by RODON. The different failure modes are explained in Section 6.4. Also *failure mode* 1, positively shifted sensor, causes a conflict seen as the conflict in the bottom. *Failure mode* 2 causes no conflict and is thus the result of the diagnosis since the data is then consistent with the model. As wished, the conclusion is that the sensor is negatively shifted, i.e. measuring a too low value.

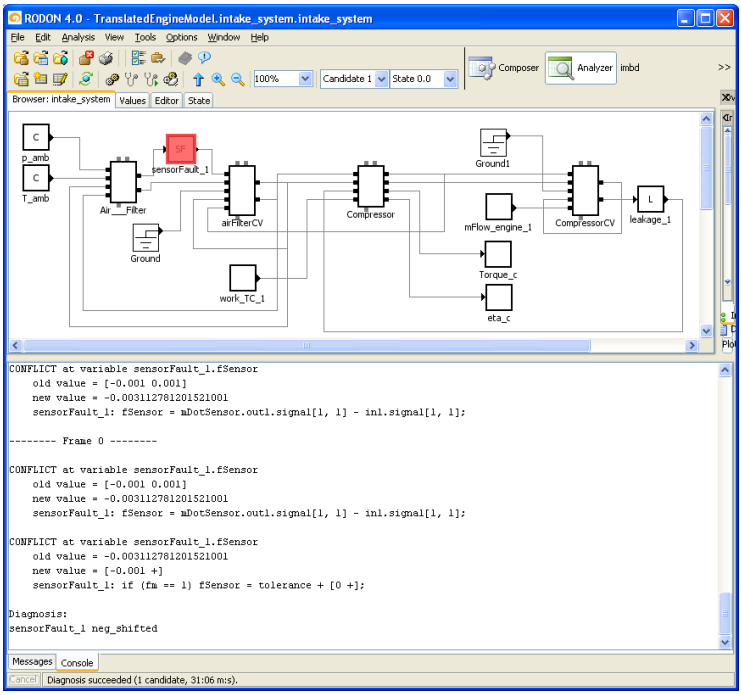


Figure 7.4: Diagnosis result for a sensor fault.

7.5 Clogging

The next fault to study is the clogging. If clogging occurs, the result is that the air flow is limited. In the fault model used for clogging there are several levels of clogging, from normal flow to total clogging when no air flows in the pipes. The clogging fault model compares a sensor value to a value computed by the model, like for the sensor fault model. In Figure 7.5, the clogging fault model is introduced in the air filter model as a component just before the output on the right side. Since it is desirable to compare the air flow from the model to a sensor value, it is a good placement for the fault model. Once again, a fault has been inserted in the data file by using MATLAB. Also this time, a 30% fault was introduced at a time of $0.046s$ and this time for the clogging fault model sensor data.

In Figure 7.6, the result of the simulation and diagnosis is shown. The air filter model is highlighted which means it contains a conflict. In Figure 7.7, it is seen that the clogging fault model causes the conflict since it is the highlighted component. In the text below the models, the diagnosis result is seen. The first conflict is a result of the simulation, because of this conflict RODON detects that the system is not ok. The second conflict shows that *failure mode* 0 causes a conflict, the first one seen in the text below `Frame 0`. Since *failure mode* 0, i.e. nominal behaviour and no fault, caused a conflict, *failure mode* 1 is then checked which represents a small clogging. Also *failure mode* 1 causes a conflict which is seen as the third conflict in the figure. For *failure mode* 2, no conflict is found. Thus, the diagnosis result is a big clogging, which corresponds to *failure mode* 2. Consequently, it was possible to find also the clogging fault.

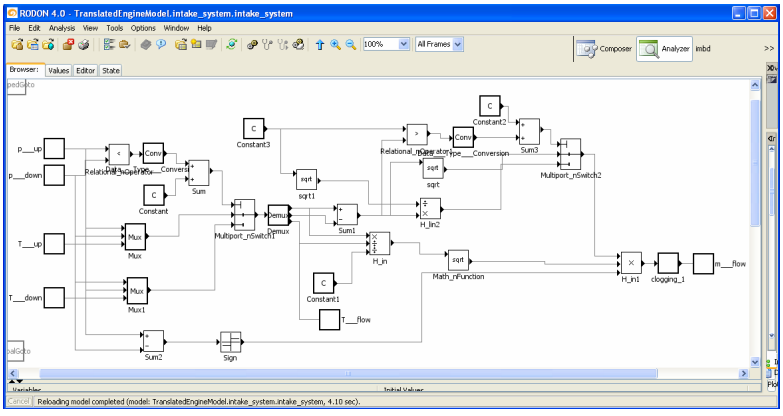


Figure 7.5: Model of the air filter with clogging fault model.

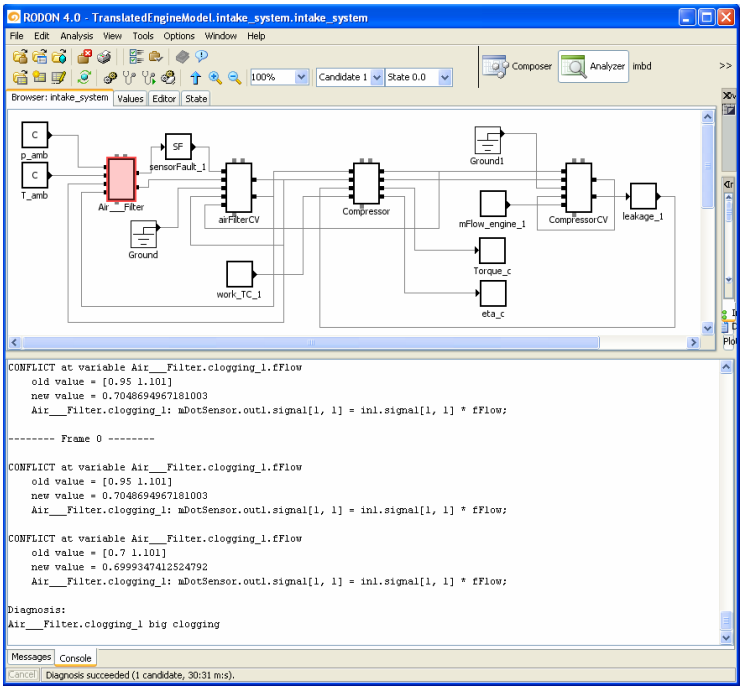


Figure 7.6: Diagnosis result for a clogging from top view.

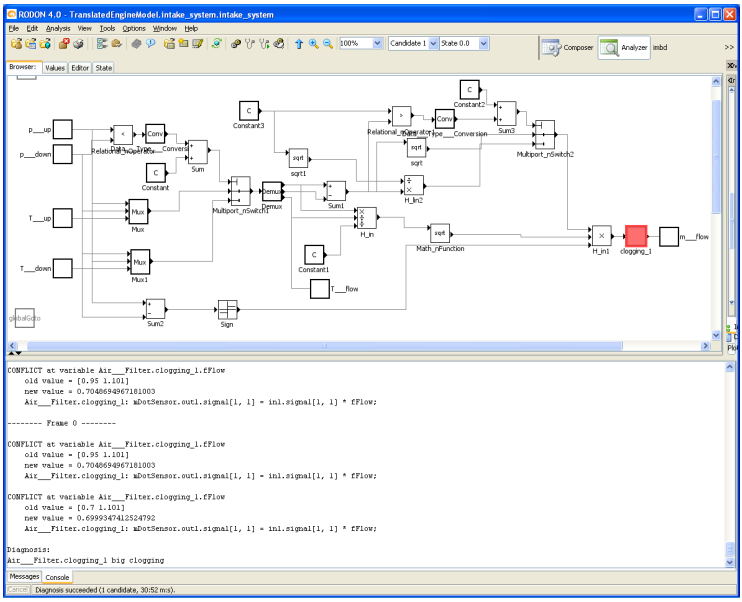


Figure 7.7: Diagnosis result for a clogging.

7.6 Leakage

The third implemented fault model is a leakage. When leakage occurs, pressure drops since there is a hole letting air out. Because of this, the fault model compares a pressure sensor value to a pressure value computed by the model. If they differ too much, a fault is assumed, i.e. a leakage. In our model we let the leakage fault model verify the pipe after the compressor. A leakage could also occur in other places in the model, like the pipe after the air filter. Since the purpose is to find a functioning diagnosis method only one leakage is needed. Like for the two other fault types, a fault has been inserted in the data file by using MATLAB. A 30% fault was introduced at a time of $0.046s$. In Figure 7.8, the diagnosis result is shown. The two first failure modes can not explain the data. This means that *failure modes* 0 and 1, i.e. no leak and a small leak, contribute to conflicts which is seen in the figure. The third failure mode, big leak, does not give a conflict and is therefore the diagnosis result. Accordingly, also the leakage can be found in the intake system model.

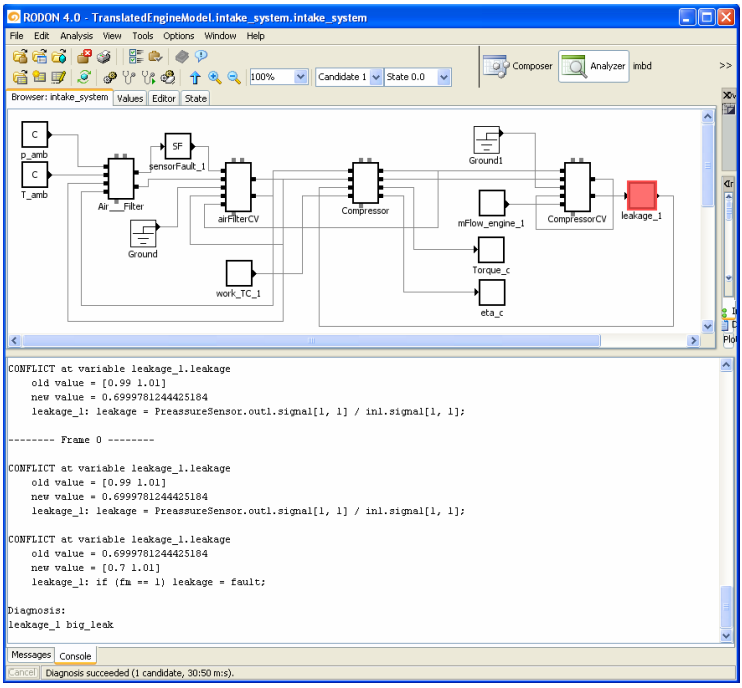


Figure 7.8: Diagnosis result for a leakage.

7.7 Multiple Faults

The three fault models have now been tested separately in the intake system model. A question is though if our solution works for several types of faults at the same time? To test this, data from the clogging fault and the sensor fault tests were merged together into larger data files now containing data for two faults. Figure 7.9 shows the first test case where a sensor fault and a clogging are introduced. For the first two conflicts in the figure, RODON was not able to find a single fault that describes the system and data consistently. The three ... in the middle of the figure is RODON's way of saying that a single fault could not explain the data, but that further diagnosis can be done by searching for double faults. When continuing the diagnosis process, the two conflicts in the bottom come up. This time, the diagnosis result is now a double fault. The & between the two candidates marks a double fault, i.e. that the two faults together explain the data. Accordingly, also double faults are possible to discover by using our fault models and RODON.

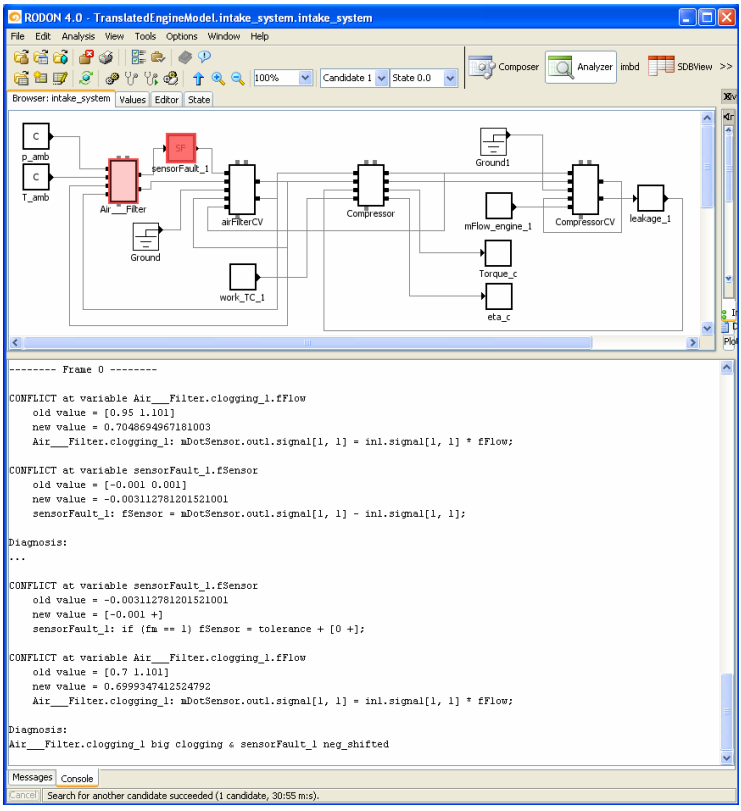


Figure 7.9: Diagnosis result for a clogging and a sensor fault.

The second test case contains both a clogging and a leakage and is seen in Figure 7.10. No single fault is consistent with the model and the data. However, a double fault is consistent, and that was expected. The diagnosis result is the double fault containing a clogging and a leakage.

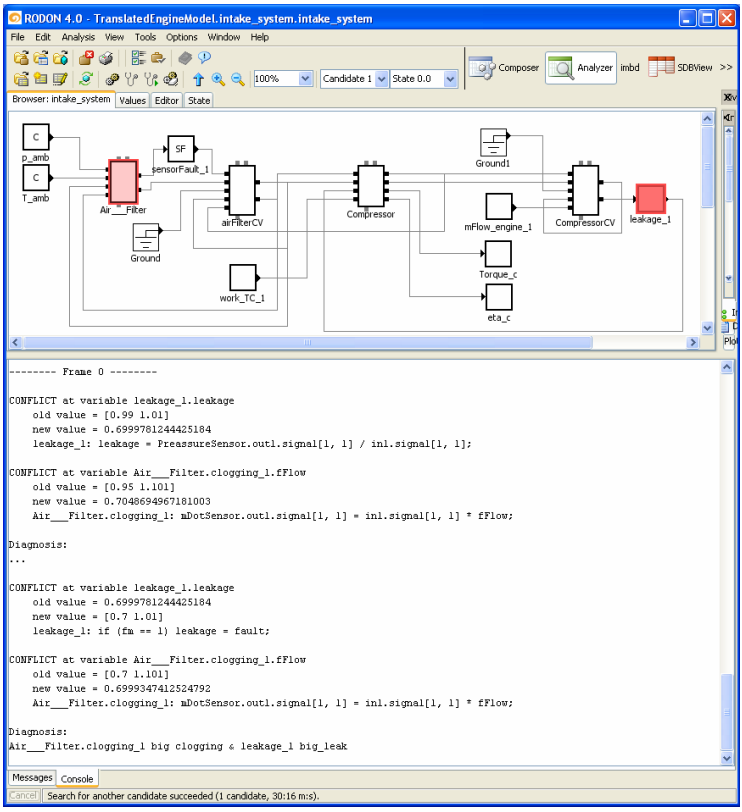


Figure 7.10: Diagnosis result for a clogging and a leakage.

7.8 Discussion

The previous sections in this chapter contain many test cases. These were created to prove that diagnosis can be performed on a combustion engine by using RODON given a Simulink model. The simulation of the cases for 0.1 seconds goes rather quickly, around 20 seconds. On the other hand, it takes about 30 minutes to diagnose the system for $0.1s$ and this is very time consuming. The faults have been inserted in the data files at an early stage and that is why the simulation is quick since the simulation is stopped when a conflict is discovered for dynamic simulations in RODON. When a conflict is discovered, the simulation stops and it is then possible to do diagnosis. For the diagnosis part, the system is checked in all time steps up to the time of the conflict and this takes a lot of time. The amount of time it takes to do a proper diagnosis with simulation is way over a reasonable time to wait in the workshop for the diagnosis outcome. For now this is acceptable since this thesis is of the investigating kind.

This diagnosis solution has some issues that we are now going to discuss. The first one concerns our fault modelling. This has already been discussed in Chapter 6 where the fault models are implemented. Since the Advanced-Blocks library has fixed causality, translated models will have that as well. This affects the fault models. Otherwise in RODON, the diagnostic engine facilitates for a physical modelling of fault, for example a smaller effective area of the pipe in case of a clogging. This would also affect the state of the system in case of fault which the current solution does not do.

The next issue concerns RODON's handling of dynamic simulation. In the present way, the simulation is stopped in case of a conflict. However, if the simulation could continue also in case of a conflict, other conflicts could be found as well and the behaviour in case of a conflict could be observed. That would be a help for the isolation of faults. The current solution cannot discern a sensor fault from a clogging fault which is desirable. If the simulation could continue, one possibility to distinguish faults from each could be take advantage from the extra information this simulation could give. For example, that a clogging could bring a constant decrease of the air mass flow, while the same procedure is not likely in case of a sensor fault.

The current diagnosis solution in RODON brings some drawbacks that have been discussed in this chapter. However, for the current implementation and solution the diagnosis is performed according to the expected results, even though the diagnosis was time consuming. We can therefore come to the conclusion that the diagnosis works.

Chapter 8

Discussion

As seen in the previous chapters, the result is that we are able to translate a Simulink model into a RODON model. It is also possible to perform diagnosis on a model of the air intake system of a combustion engine. The faults that were chosen for thesis, sensor fault, clogging, and leakage were possible to detect by using RODON. There are however some issues we would like to discuss in this chapter.

Recursive Definition of Addition

The structure of the RODON AdvancedBlocks library that we have produced, is sometimes a bit too complicated. The reason is that the translator that was used to convert Simulink models into RODON models was already constructed for a Modelica AdvancedBlocks library. When trying to keep the same structure as in Modelica, the library became complex in RODON for certain models. An example is the Addition class which is described in Section 5.4. Instead of having one Addition model, the library contains an Addition2 with two inputs, an Addition3 with 3 inputs etc. To overcome this issue, a recursive definition of the addition could be used. This means that there is a function performing the addition for two inputs. This function can then be called several times to extend the addition to several inputs, like it was done with the MatrixMult function in Section 5.4.

Causality in the AdvancedBlocks Library

The developed AdvancedBlocks library was implemented to be able to translate Simulink models into RODON models. In Simulink the causality is fixed. As our library components should function like in Simulink they also have fixed causality. This results in some issues that have been discussed throughout the thesis. The engine in a car is a dynamic system and therefore this fixed causality is problematic. However, we think that for a static system the

translation and diagnosis process in this thesis should be more appropriate and possible to improve to make use of it in for example workshops.

Data Generation

When checking the diagnosis system, data produced from the nominal Simulink model was used. Faults were then introduced by using MATLAB and changing the values by 30%. This is perhaps not the best solution when you want to prove that the diagnosis can work in a real system. It is desirable that the data used for simulation and diagnosis comes from a real engine. However, with the data produced from Simulink and MATLAB, it was possible to show that the desired behaviour was obtained. The purpose of this thesis was to indicate that the diagnosis method works, and therefore the data produced in MATLAB is ok. For further development, the data should come from a real engine to verify that the system can find faults accurately. Another option, if real data is not available, is to add noise to the produced data in Simulink.

Fault Models

The fault models we have created are simple models. For the sensor fault model such a simple model should be ok, since a fault causes a deviation in the sensor value. For the leakage and the clogging, a fault may have several effects on the system that are not modelled in the current fault models. A leakage and a clogging should have an influence on the state of the system that is not treated in the current solution. Before putting this diagnosis system into a car, this should be investigated more. The purpose was to show that RODON is a possible tool for combustion engine diagnosis and that has been done in the thesis.

Symptoms

In order to improve the diagnosis performance, an idea could be to use symptoms to help the diagnosis tool RODON. What we call symptoms are observations that humans can make, e.g. a customer or a workshop employee. Examples of these symptoms can be white smoke or a metallic sound coming from the engine etc. When doing diagnosis, the symptoms may be helpful for generating candidates since they can increase the information about the system. Although, a problem arises, that is how to implement these observations in the model. Another thing to consider is how to connect a symptom to a fault, i.e. what does a symptom mean for the engine behaviour, and what could be the possible candidate according to the symptom.

Chapter 9

Conclusions

In this thesis, a diagnosis process for combustion engines has been investigated. The process connects information in design departments to workshops where the diagnosis is performed. A Simulink model describing the nominal behaviour of a combustion engine was given. The diagnostic tool RODON was also available as well as a Rodelica to Modelica translator. A process that could join these parts and make it possible to do diagnosis on an engine was missing. The main parts and conclusions of the investigated process are described below.

A library in RODON AdvancedBlocks library was developed. The library can be used for translation of Simulink models into RODON models. In the thesis, this has been done successfully in Section 5.5 for a part of the combustion engine which is seen in Figure 5.14. The given translator can now be used as a Simulink to Rodelica translator.

Fault models for sensor faults, clogging, and leakage have been implemented and validated in RODON in Chapter 6. They have been implemented in separate fault models to make it possible to reuse them, for example the sensor fault model can be utilized in several places in the engine.

For the translated intake system model, sensor faults, clogging, and leakage were possible to detect with RODON. The diagnose was performed successfully in RODON as shown in Chapter 7.

Chapter 10

Future Work

10.1 AdvancedBlocks library

The developed AdvancedBlocks library can be used for translation of Simulink models into RODON models. However, there are still some items that can be improved:

- The option of choosing the number of input and output is not flexible. To develop a solution for the inputs and outputs to have a functionality more similar to the one in Simulink is desirable. In other words, to have for example one addition block where the inputs can be set to a number between 1 and 30 instead of having an *Addition2* block with two inputs, an *Addition3* block with 3 inputs etc.

- Implement more models in the library to increase the flexibility for modelling of Simulink models since unimplemented models can be translated from Simulink to RODON but they can not be used in RODON after the translation. The reason for this is that the translator in the translation process writes the code for the RODON model, but RODON does not know how to use it since it has no models that fits the descriptions in the library.

10.2 Diagnosis of the Engine

We managed to prove that the engine model can be used for diagnosis in RODON. The implemented faults, sensor fault, clogging, and leakage, can be detected by RODON. Although for arriving in a more complete combustion engine diagnosis, further development is needed:

- Translate the entire Simulink model and do diagnosis on it.

- Introduce more faults, fault models, and improve the present fault models.
- Use data for simulation and diagnosis from a real engine.
- Improve the dynamic diagnostic engine in RODON in order to decrease the time needed for simulation and diagnosis.
- Improve the dynamic diagnostic engine in RODON to be able to contine the diagnosis in case of a conflict.
- Include symptoms; human observables, to improve the diagnosis process.

References

[1] K.G. Andersson. *Lineär algebra*. Number 91-44-01608-5. Studentlitteratur, Lund, 2000.

[2] P. Andersson. *Air Charge Estimation in Turbocharged Spark Ignition Engines*. Ph.d. thesis 989, Department of Electrical Engineering, University of Linköping, Linköping, Sweden, November 2005.

[3] U.M. Ascher. *Computer methods for ordinary differential equations and differential-algebraic equations*. Number 0-89871-412-5. Society for Industrial and Applied Mathematics, cop., Philadelphia, USA, 1998.

[4] Å. Lönnqvist C. Britsman and S.O. Ottosson. *Handbok i FMEA: failure mode and effect analysis*. Number 91-7548-317-3. Förlags AB Industrilitteratur, 1993.

[5] L. Eriksson and L. Nielsen. *Vehicular Systems*. Bokakademin, Linköping, Sweden, 2006.

[6] T. Glad and L. Ljung. *Modellbygge och Simulering*. Number 91-44-02443-6. Studentlitteratur, Lund, 2004.

[7] MathWorks. *MATLAB The Language of Thecnical Computing*, 2002.

[8] MathWorks. www.mathworks.com. Internet, 2007.

[9] MathWorks. www.mathworks.com/access/helpdesk/help/toolbox/simulink/. Internet, 2007. MathWork- Support- Documentation- Simulink- Working with signals- SignalBasic.

[10] Sörman Information & Media. *Getting Started with RODON*, 2006.

[11] Sörman Information & Media. *RODON Tutorial*, 2006.

[12] M. Nyberg. *Model based Fault Diagnosis- Methods, Theory, and Automotive Engine Applications*. Ph.d. thesis 591, Department of Electrical Engineering, University of Linköping, Linköping, Sweden, May 1999.

[13] M. Nyberg. *Model Based fault Diagnosis- Methods, Theory, and Automotive Engine Applications*. Number 91-7219-521-5. Linus & Linnea, Linköping, 1999.

[14] M. Nyberg and E. Frisk. *Model Based Diagnosis of Technical Processes*. LiUTryck, Linköping, Sweden, 3.16 edition, 2006. Chapter 1.12.

[15] M. Nyberg and E. Frisk. *Model Based Diagnosis of Technical Processes*. LiUTryck, Linköping, Sweden, 3.16 edition, 2006.

[16] Encyclopædia Britannica Online. Search word: Diagnosis. Internet, June 2007. http://search.eb.com/eb/article-9106175.

[17] M. Otter and H.Elmqvist. *Modelica- Language, Libraries, Tools, Workshop and EU-Project RealSim*, June 2001.

[18] J.B. Fussel R.E. Barlow and N.D. Singpurwalla. *Theoretical and Applied Aspects of System Reliability and Safety Assessment*. Number 0-89871-033-2. Society for Industrial And Applied Mathematics, 1975.

Notation

Definitions

Candidate	The possible contributors to the faults discovered by RDT.
Symptoms	Observations from users, e.g. ”white smoke”, ”metallic sound”.
RODON	Tool for diagnostics and generating of Decision Trees based on a given system, e.g. electric, mechanical etc.
MBD	Model Based Diagnosis.

Appendix A

Translator

In this appendix, a case study is chosen to demonstrate the functionality of the translator used in this thesis. In Figure A.1 and Figure A.2, a Simulink model is found. The model is a part of the engine model used in the thesis, this time only the compressor control volume is demonstrated. The first figure shows the top view of the model. Since this model is cut out from the rest of the engine, constants and scopes have been added to be able to simulate the model. The second figure shows the subsystem that is the control volume model of the compressor and which represents the pipe after the compressor in a real engine.

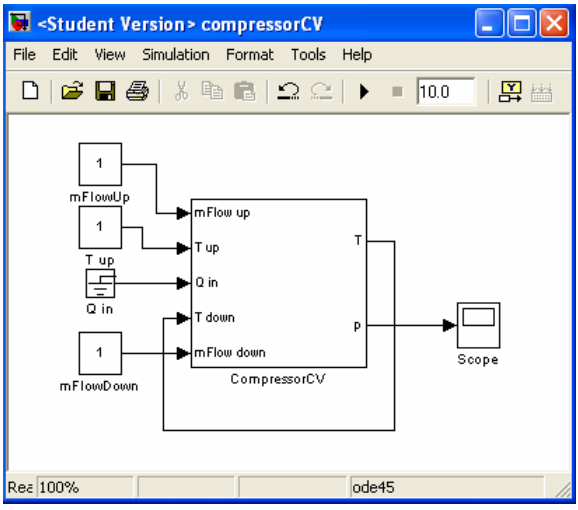


Figure A.1: Top view of Simulink model of the compressor control volume.

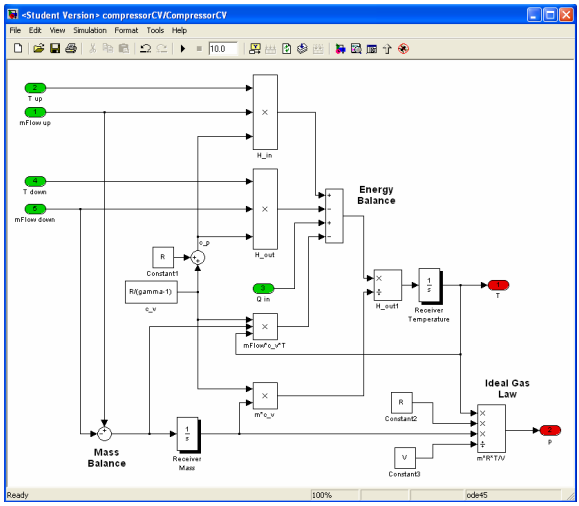


Figure A.2: Simulink model of the compressor control volume.

In Figure A.3, the translator has been opened. The name of the translator is still *Simulink to Modelica Translator* which shows that it has been used for Modelica before. The name should be changed to *Simulink to* RODON *Translator* in the future. The translator consists of an interface in which it is possible to open and close models; this can be seen in the figure. After pressing the *Open model* button, it is possible to choose which model to translate. In Figure A.4 the compressor control volume Simulink model has been opened. All the blocks are represented in the tree. The plus sign in the top, demonstrates that there is a subsystem in the model. To actually translate the model into a RODON model, the *Write Rodelica* button must be pressed, this is seen in Figure A.5. It is also possible to decide where the translated model should end up. This is done by choosing the *Set output directory* button.

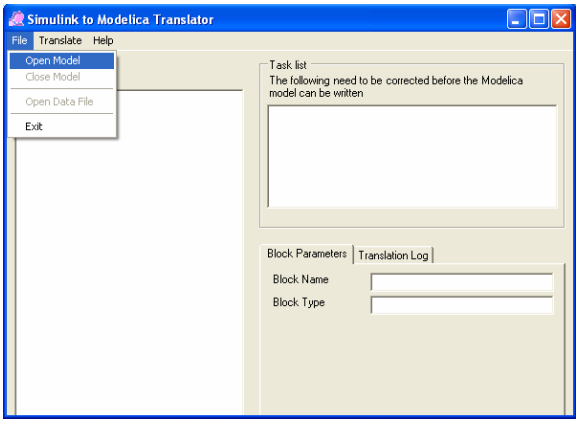


Figure A.3: Simulink to Rodelica translator.

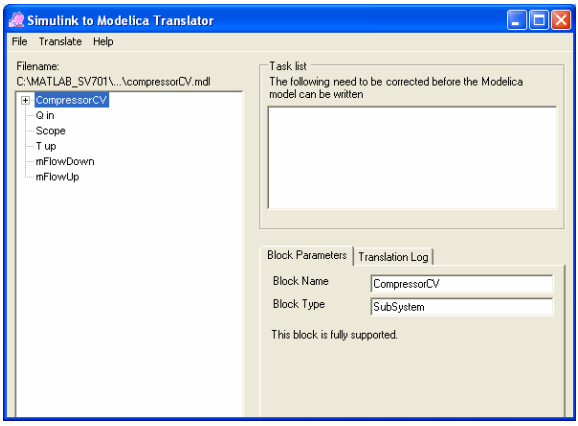


Figure A.4: Simulink to Rodelica translator.

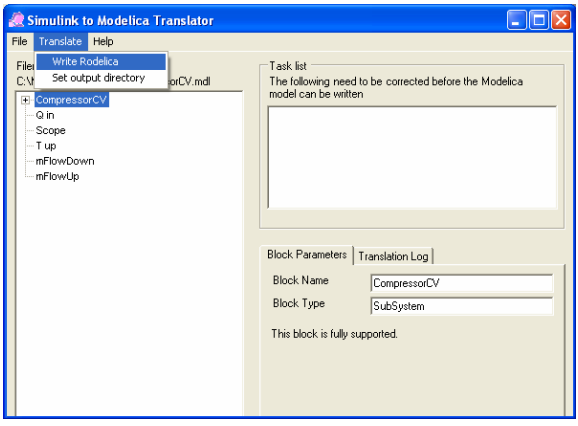


Figure A.5: Simulink to Rodelica translator.

When the model has been translated, it can be found in RODON in the selected output directory. Figure A.6 shows the top view of the model, but this time in RODON. In Figure A.7 the control volume model can be seen. As one can see, it contains the same blocks as in Simulink and according to the library tree on the left side, the hierarchical structure remains the same. To conclude, we have now shown how the translator is used to translate a Simulink model into a RODON model.

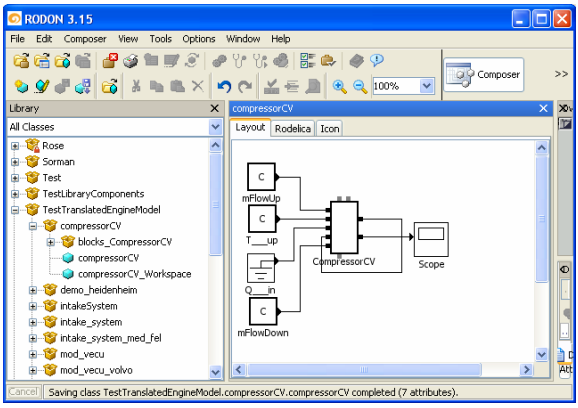


Figure A.6: Top view of RODON model of the compressor control volume.

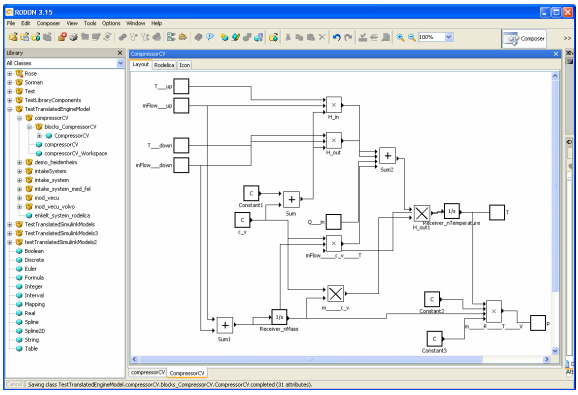


Figure A.7: RODON model of the compressor control volume.

Appendix B

Data Generation

This appendix treats the data generation. Knowledge about Matlab and Simulink simulation and how data is stored when simulating and writing to the Matlab workspace is required.

For simulation and diagnosis, input data is needed. This data is placed in look up tables with time on the x-axis and values on the y-axis. In Figure B.1 an example on how the look up table is initialized is shown.

```
mFlow_engine_1.values_m[32, 1] = 0.000554829;
mFlow_engine_1.values_m[32, 2] = 0.00457375;
mFlow_engine_1.values_m[33, 1] = 0.000554831;
mFlow_engine_1.values_m[33, 2] = 0.00457376;
mFlow_engine_1.values_m[34, 1] = 0.000554831;
mFlow_engine_1.values_m[34, 2] = 0.00457376;
mFlow_engine_1.values_m[35, 1] = 0.000554833;
mFlow_engine_1.values_m[35, 2] = 0.00457377;
mFlow_engine_1.values_m[36, 1] = 0.000554835;
mFlow_engine_1.values_m[36, 2] = 0.00457378;
```

Figure B.1: Segment of .dat file, look up table initialization.

In this case, it is the mass flow from the cut off parts of the engine which is shown in the figure above. The name of the variable, its index and its values are set with a .dat file. Each element in the look up table has two values. The first is the time and the second is the value. In this example element 34 sets the time to 0.000554831 and the mass flow to 0.00457376. The numbers has been taken from Matlab. By doing a simulation in Simulink and saving the values to the workspace it is possible to make .dat files by using a .m script. Here is an example on how a file is generated.

```
1:  %% Start for generating .dat file for diagnosis with no faults
2:  start = 1;
3:  finish = size(w_tc);

4:  fid = fopen('intake_system_input_data_for_diagnosis.dat', 'w');
5:  fprintf(fid, '// autogenerated file for simulation and diagnosis\n\n');
6:  fprintf(fid, '// this file is generated for dt =1.0e-4 and 1000 states per frame.\n\n');
```

```
7:  fprintf(fid,'<initial-modifications>\n\n\n');

8:  fprintf(fid,'\n\n\n//------Engine data.------\n\n\n');

9:  for i = start:finish(1)
10:    x = t(i);
11:    y = mFlow_e(i);
12:    fprintf(fid, 'mFlow_engine_1.values_m[%g, 1] = %g;\n',i,x);
13:    fprintf(fid, 'mFlow_engine_1.values_m[%g, 2] = %g;\n',i,y);
14: end

15: fprintf(fid,'\n\n');

16: for i = start:finish(1)
17:    x = t(i);
18:    y = w_tc(i);
19:    fprintf(fid, 'work_TC_1.values_w[%g,1] = %g;\n' ,i,x);
20:    fprintf(fid, 'work_TC_1.values_w[%g,2] = %g;\n' ,i,y);
21: end

22: fprintf(fid,'\n\n\n//-------Sensor data.------\n\n\n');

23: for i = start:finish(1)
24:    x = t(i);
25:    y = mDotSensor(i);
26:    fprintf(fid, 'Air___Filter.clogging_1.mDotSensor.values[%g, 1] = %g;\n',i,x);
27:    fprintf(fid, 'Air___Filter.clogging_1.mDotSensor.values[%g, 2] = %g;\n',i,y);
28: end

29: fprintf(fid,'\n\n');

30: for i = start:finish(1)
31:    x = t(i);
32:    y = mDotSensor(i);
33:    fprintf(fid, 'sensorFault_1.mDotSensor.values[%g, 1] = %g;\n',i,x);
34:    fprintf(fid, 'sensorFault_1.mDotSensor.values[%g, 2] = %g;\n',i,y);
35: end

36: fprintf(fid,'\n\n');

37: for i = start:finish(1)
38:    x = t(i);
39:    y = preasureSensor(i);
40:    fprintf(fid, 'leakage_1.PreassureSensor.values[%g, 1] = %g;\n',i,x);
41:    fprintf(fid, 'leakage_1.PreassureSensor.values[%g, 2] = %g;\n',i,y);
42: end

43: fprintf(fid,'\n\n\n</initial-modifications>\n\n');

44: status = fclose(fid);
```

On row 2 and 3, the start and finish values of a counter is set. The counter is used to iterate through the variables to create a look up table, this can be seen in lines 9-14 and Figure B.1. Row 4 creates (open) a writable file called *intake_system_input_data_for_diagnosis.dat*. On row 5 - 6 some initial comments in the .dat file is generated. As can be seen the // represents a comment in RODON. The data generated from Matlab is not going to be changed in RODON during simulation. Therefore it is set as *initial-modifications* on line 7. In lines 8-21 the engine data is written into the file and in lines 22-42 the sensor data is set. The data is taken from the i:th element of the vector created in Simulink. The for-loop sets $i = 1$ the first time and ends when the whole vector is iterated. The file is ended and closed on line 43 and 44. The concept of reading and writing on files via .m-scrpit can be found in chapter 6 in [7].

When generating faulty sensor value the only difference is that the sensor value is changed by a multiplication of a number. For example, if we would like to change the pressure sensor in order to make it look like a leakage we just enter

```
preasureSensor(279:558) = preasureSensorTrue(279:558)*0.7;
```

between line 39 and 40. This means that the pressure sensor measures

a 30% lower pressure than it is supposed to. The other sensor values are modified in the same way.

Copyright

Svenska

English