FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Real-time distributed testing for electrical vehicle

Ricardo Maria de Portugal Bettencourt



Mestrado Integrado em Engenharia Mecânica- Especialização em Automação

Supervisor (FEUP): Prof. Dr. Joaquim Gabriel Magalhães Mendes

Co-Supervisor (SISW-NV): Dr. Bernhard Sputh

October, 2020

Real-time distributed testing for electrical vehicle

Ricardo Maria de Portugal Bettencourt

Mestrado Integrado em Engenharia Mecânica

Especialização em Automação

October, 2020

Abstract

Real-time distributed testing is the field of study that allows a cooperative testing scenario with test benches running in real-time and geographically distributed. This type of distributed scenarios allows to decentralize the development and testing, reducing its costs and enhancing the process. The focus of this thesis lies on the coupling of the test benches, particularly on the development of the communication procedure and the implementation of correction methods for coupling errors between the test benches. These coupling errors exist because of the communication impairments and the stability and initialization problems occurring in distributed scenarios.

The development of the distributed scenarios follows three stages, according to the model-based development (MBD) approach used. First, there is the model-in-the-loop stage where the models for the distributed virtual testing are created, with particular focus on their real-time capability. Secondly, there is the software-in-the-loop stage where the coupling element programs are developed. In this thesis, this includes the implementation of the correction methods and the creation of communication impairments simulators. In the last stage, hardware-in-the-loop, occurs the integration of physical elements in the distributed scenario. In this case, two real-time computers where connected in a local area network with each computer having the models and coupling methods previously created.

The accuracy of the simulation in regards to a non-distributed scenario was continuously evaluated through-out the process. For low communication impairments, the ones of a local area network, distributed testing was achieved with minor accuracy losses. However, the capacity of the methods implemented was not guaranteed for the desired distributed testing scenario, with further work being required.

The outcomes of this thesis suggest that distributed testing is possible, but more robust coupling elements should be built. This can be achieved by integrating other suggested correction methods and stability and initialization procedures. Furthermore, this work reinforces the need for proper methods of building models for distributed virtual testing.

Keywords: distributed testing, co-simulation, network impairments, coupling methods, AMESim

Resumo

Teste distribuído em tempo real para veículos elétricos

O teste distribuído em tempo real é o campo de estudo que permite um cenário de teste co-operativo com bancadas de teste a funcionar em tempo real e distribuídas geograficamente. Este tipo de cenários distribuídos permite descentralizar as fases de desenvolvimento e teste, reduzindo custos e potencializando o processo. O foco desta tese reside no acoplamento das bancadas de teste, em particular no desenvolvimento de procedimentos de comunicação e na implementação de métodos para correção de erros de interface entre as bancadas de teste. Estes erros ocorrem devido às deficiências de comunicação e aos problemas de estabilidade e inicialização de cenários distribuídos.

O desenvolvimento dos cenários distribuídos segue três fases, de acordo com a abordagem de desenvolvimento utilizada (*model-based development* (MBD)). Em primeiro lugar, existe a fase *model-in-the-loop*, onde os modelos para o teste virtual distribuído são criados, com foco particular na sua capacidade de execução em tempo real. Em segundo lugar, existe a fase *software-in-the-loop*, onde os programas dos elementos de acoplamento são desenvolvidos. Nesta tese, isso inclui a implementação dos métodos de correção e a criação de simuladores de deficiências de comunicação. Na última etapa, *hardware-in-the-loop*, ocorre a integração dos elementos físicos no cenário distribuído. Neste caso, dois computadores de tempo real foram ligados numa rede local, com os modelos e métodos de acoplamento criados nas fases anteriores.

A precisão da simulação em relação a um cenário não distribuído foi avaliada continuamente ao longo do processo. Para deficiências de comunicação baixas, existentes numa rede local, o teste distribuído foi alcançado com pequenas perdas de precisão. No entanto, a capacidade dos métodos implementados não foi garantida para o cenário de teste distribuído desejado, sendo necessária continuidade deste trabalho.

Os resultados desta tese sugerem que o teste distribuído é possível, mas que elementos de acoplamento mais robustos devem ser construídos. Isso pode ser alcançado por integração de outros métodos de correção sugeridos e procedimentos de estabilidade e inicialização. Além disso, este trabalho reforça a necessidade de métodos adequados de construção de modelos para testes virtuais distribuídos.

Keywords: teste distribuído, co-simulação, deficiências de comunicação, métodos de acoplamento, AMESim

Acknowledgements

The academic work is never made solo. Many are the people who, directly or indirectly, helped me in the writing of this thesis and in the development of the work that sustains it.

In this sense, the first and greatest acknowledgment goes to Bernhard. The time he invested and the concerns he took went beyond his responsibilities as a supervisor. Much of this work and my joyful first professional experience is credited to him. Also, thanks to all the other colleagues at SISW-NV that somehow contributed to this work, namely, Mohamed, Laurane, Roland and Clement.

A big acknowledgement is also due to SISW-NV and to the ERASMUS+ programme for the opportunity and financial support that allowed this thesis.

To my supervisor at FEUP, professor Joaquim Gabriel, for looking after this work, and to all the FEUP community for helping me understand all the challenges in engineering and how much there is still to be done.

To Simão, Patrício, André, Bernardo and Bruno, who survived the Belgian quarantine with me, and for all the discussions and beers we shared, I thank you for keeping my mental sanity in these troubled times.

To all my friends, in Porto and Aveiro, with whom I lived many great moments and to whom I credit the life I had in these last 5 years. You are too many to be named, but know that I cherish you the most.

And to my family, who gave me all the opportunities I had in life and to whom I always go for advise. My appreciation goes way beyond this thesis.

Ricardo Bettencourt

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 3
- 1.3 Objectives . . . 5
- 1.4 Structure . . . 5

2 Theoretical background **7**
- 2.1 Co-simulation . . . 7
  - 2.1.1 Terminology . . . 8
  - 2.1.2 Continuous-time-based co-simulation . . . 10
  - 2.1.3 Simulation fidelity . . . 13
  - 2.1.4 Standards for co-simulation . . . 14
- 2.2 Communication network . . . 17
  - 2.2.1 Communication impairments . . . 17
  - 2.2.2 Communication protocols . . . 19
  - 2.2.3 Network Emulation . . . 22
- 2.3 Challenges for real-time distributed testing . . . 23
  - 2.3.1 Real-time requirements . . . 23
  - 2.3.2 Time synchronization . . . 23
  - 2.3.3 Real-time coupling . . . 26

3 Literature Review **29**
- 3.1 Signal-based coupling . . . 29
  - 3.1.1 Zero-order hold (ZOH) . . . 29
  - 3.1.2 First-order hold (FOH) . . . 30
  - 3.1.3 Second-order hold (SOH) . . . 31
  - 3.1.4 Linear curve fitting (LCF) . . . 32
  - 3.1.5 Waveform properties . . . 34
- 3.2 Model-based coupling . . . 35
- 3.3 Energy-conservation-based coupling . . . 36

4 Model-in-the-loop stage **39**
- 4.1 Bicycle car modelling . . . 39
  - 4.1.1 Quarter car . . . 40
  - 4.1.2 Single body bicycle car . . . 42
- 4.2 Non-distributed co-simulation . . . 45
  - 4.2.1 Split body bicycle car . . . 45
  - 4.2.2 Interface definition . . . 47
  - 4.2.3 Local AMESim co-simulation . . . 52

5 Software-in-the-loop stage **57**
5.1 Experimental setup . . . 57
5.2 Netwok impairments simulation . . . 58
5.3 Mitigation techniques . . . 60
5.3.1 Implementation . . . 60
5.3.2 Performance . . . 63

6 Hardware-in-the-loop stage **75**
6.1 Network quality evaluation . . . 75
6.1.1 Server . . . 76
6.1.2 Client . . . 77
6.2 Real time platform . . . 79
6.2.1 AMESim models . . . 79
6.2.2 Coupling elements . . . 84
6.2.3 Distributed co-simulation . . . 87

7 Conclusion **89**
7.1 Further Work . . . 90

References **93**

List of Figures

1 Introduction

1.1 MBST development and validation stages. Adapted from Van der Auweraer, Sarrazin & Santos (2017). . . . 3
1.2 Geographically distributed architecture for XiLforEV. . . . 4

2 Theoretical background

2.1 Linear relations of simulated and wall-clock times. . . . 8
2.2 Schematic representation of types of scheduling. . . . 11
2.3 Multi-body system coupled by a massless link. Adapted from Gomes et al. (2018). 12
2.4 FMI design principles. Adapted from Modelica Association Project "FMI" (2013). 15
2.5 FMI Co-simulation Slave State Machine. Extracted from Modelica Association Project "FMI" (2013). . . . 16
2.6 DCP Slave State Machine. Adapted from Modelica Association Project DCP (2019). 18
2.7 Transmission rate temporal adaptation. Extracted from Vogel et al. (2019). . . . 21
2.8 Network Time Protocol (NTP). . . . 25
2.9 Precision Time Protocol (PTP). . . . 27

3 Literature Review

3.1 Polynomial extrapolation techniques. Extracted from Stettinger et al. (2013). . . . 30
3.2 Zero-order hold signal approximation. . . . 30
3.3 First-order hold signal approximation. . . . 31
3.4 Second-order hold signal approximation. . . . 32
3.5 Linear regression. . . . 33
3.6 Extrapolation, interpolation and interpolated extrapolation. Extracted from Dronka & Rauh (n.d.). . . . 34
3.7 Model-based coupling element. Extracted from Stettinger et al. (2017). . . . 35
3.8 Residual power in co-simulation. Extracted from Sadjina et al. (2019). . . . 37

4 Model-in-the-loop stage

4.1 Bicycle car schematics. Extracted from Jazar (2008). . . . 40
4.2 Quarter car models. . . . 41
4.3 Block used for excitation of the mechanical system. . . . 42
4.4 Quarter car iterative and non-iterative integrators comparison. . . . 43
4.5 Bicycle car model in AMESim. . . . 44

4.6 Bicycle car absolute error between integrators. 46
4.7 Bicycle car body splitting. 46
4.8 Rear and front body absolute error between split and unsplit model. 48
4.9 Wheels displacement comparison between split and unsplit model. 49
4.10 Variables exchanged at the co-simulation coupling point. 50
4.11 Interface variables alternatives. 50
4.12 Absolute error between acceleration input source and single body bicycle car models. 51
4.13 Joint relative rotation variation with the first-order lag time constant. 52
4.14 Absolute error between position input source and single body bicycle car models. 53
4.15 AMESim co-simulation models. 54
4.16 AMESim co-simulation results. 55

5 Software-in-the-loop stage

5.1 AMESim model for implementation of executables 58
5.2 Executable blocks interfaces. 58
5.3 Impairment implementation validation for macro-step size of 0.1 s and micro-step size of 0.001 s. 61
5.4 Buffer state evolution example 62
5.5 Mitigation implementation validation for macro-step size of 0.1 s, micro-step size of 0.001s and a constant delay (avgDelay = 0.05 s) without packet loss 64
5.6 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0. 65
5.7 Mitigation for impairment of avgDelay = 0.5s, stdDelay = 0s and packetLossRatio = 0. 66
5.8 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0.01s and packetLossRatio = 0. 68
5.9 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0.1s and packetLossRatio = 0. 69
5.10 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.05. 70
5.11 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.5. 71
5.12 Mitigation for impairment of avgDelay = 0.093 s, stdDelay = 0.007 s and packetLossRatio = 0.05. 73
5.13 Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.5 of a square wave. 74

6 Hardware-in-the-loop stage

6.1 Communication architecture 76
6.2 Communication state machines 78
6.3 AMESim models with UserCosim interface 80
6.4 Body vertical displacement comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim 81
6.5 Angular rotation comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim 82

6.6 Wheel displacement comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim 83
6.7 Split quarter car models for co-simulation 84
6.8 Absolute error between split quarter car model and unsplit quarter car model with iterative solver, for macro step of 1 milliseconds 85
6.9 General coupler FMU architecture 85
6.10 Quarter car distributed testing results 88

List of Tables

2 Theoretical background

2.1 Performance results for TCP/IP and UDP/IP. Adapted from Wheeb (2015). . . . 20
2.2 TCP and RTP performance comparison. Adapted from Ma & Gao (2012). 21
2.3 Real-time levels. Adapted from Schreiber et al. (2018). 24

4 Model-in-the-loop stage

4.1 Quarter car parameters. 41
4.2 Bicycle car parameters. 45
4.3 Rotary junction selected parameters. 47
4.4 Influence of the first-order lag time constant on the maximum time step. 52

5 Software-in-the-loop stage

5.1 Measured impairments between Leuven and Ferrol, via IMCP. 72

Abbreviations

CT	Continuous Time
co-SU	Co-simulation Unit
DCP	Distributed Co-simulation Protocol
DOF	Degree of freedom
DE	Discrete Event
FMI	Functional Mock-Up Interface
FMU	Functional Mock-Up Unit
FOH	First-order hold
HiL	Hardware-in-the-loop
HuIL	Human-in-the-loop
IMCP	Internet Message Control Protocol
MBST	Model-based system testing
MiL	Model-in-the-loop
NTP	Network Time Protocol
LCF	Linear Curve Fitting
PTP	Precision Time Protocol
RT	Real-time
RTP	Real-time Transmission Protocol
RTCP	Real-time Control Protocol
SiL	Software-in-the-loop
SiTL	System-in-the-loop
SOH	Second-order hold
SU	Simulation Unit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XiL	X-in-the-loop
ZOH	Zero-order hold

Chapter 1

Introduction

1.1 Context

It is generally understood that engineering systems are becoming increasingly complex. First, there is an increasing number of vehicle models to satisfy every clients' needs. Allied to this, vehicles are required to have more safety mechanisms, which by themselves lead to the incorporation of more components, namely sensors and electronics (Maurer, 2013).

Schreiber et al. (2018) focus on the example of the current development of automated driving where the operation of sensors, actuators, and on-board processors needs to be tested for the whole life-cycle of the vehicle. All this while witnessing its operation on near real-life conditions through billions of kilometres of virtual test roads. The number of components and software so interdependently connected requires the development of improved and more efficient simulation and testing tools.

With this in mind, the concept of Model-Based System Testing (MBST) gains relevance. MBST is defined by Van der Auweraer, Sarrazin & Santos (2017) as:

> "[. . .] the discipline combining physical testing and simulation models with the aim to study, identify, validate and improve the behaviour of multi-physical and mechatronic systems."

This means that the combination of testing and simulation, in the same development framework, allows for improved validation of simulations while also improving enhanced information of the testing phase. Also, it allows for the simulation and testing of interconnected components of different physical domains which require different simulation tools and testing setups. At last, this framework enhances hybrid testing approaches, where both physical and virtual components are present in the testing scenario. According to Van der Auweraer, Sarrazin & Santos (2017) and dos Santos et al. (2016), the different contributions of the MBST approach can be synthesized as:

- **Testing for simulation** where the role of testing is the obtention of data for improvement and completion of models and simulations, such as model parameters or realistic input data;

- **Simulation for testing** where the simulation purpose is to improve the testing process by providing insight on the optimal test setup, such as sensor placement, ideal excitation or component configuration. Simulation also aids the testing process by means of virtual sensing, where the model is fed with experimental data, to allow the sensing of internal variables or variables of which the sensing is impossible or unfeasible;

- **Testing with simulation** where simulation and testing occur simultaneously, allowing for the combination of real and virtual systems, thus enabling to witness the effects of integration of components on a full system. According to Siemens Digital Industries Software (2019), these systems combination can be classified as:

  - Hardware-in-the-loop (HiL), if the physical component is an electronic controller;
  - System-in-the-loop (SiTL), if the physical component is a subsystem (e.g. steering system);
  - Hybrid testing, if the physical component is part of a component (more common in structural analysis);
  - Human-in-the-loop (HuiL), if human users interact with the virtual system (e.g. driving simulator).

  These systems can be generically called as X-in-the-loop (XiL), where X stands for the real system. The fact that testing and simulation happen together means that simulation needs to run in real-time.

For testing with simulation, Van der Auweraer, Sarrazin & Santos (2017) presents three distinct stages for the development and validation of XiL solutions, as can be seen in Figure 1.1. The first stage is model-in-the-loop (MiL) which includes the generation of simulation models and test cases. This allows for the creation of a reference signal that will serve as comparison for the following validation steps. The software-in-the-loop stage (SiL) occurs when the simulation model is replaced by an executable code running on a standard PC with the purpose of software validation and enabling the models for real-time computing. In this stage, real-time simulation should also occur in order to validate time-sensitive setups for XiL (Nguyen et al., 2019). Finally, the XiL stage is reached with the introduction of physical components.

Another advantage of this approach, as it allows the integration of different test benches in the same system, is to geographically distribute the test setups. With different components and test setups being developed by different partners, avoiding the logistical costs of joining these into one location can increase drastically the iterative development process of these components.

This approach, with its capability of early evaluation of performance and rupture points in the designed system, is currently getting recognition as a solution for ever more complex engineering systems.

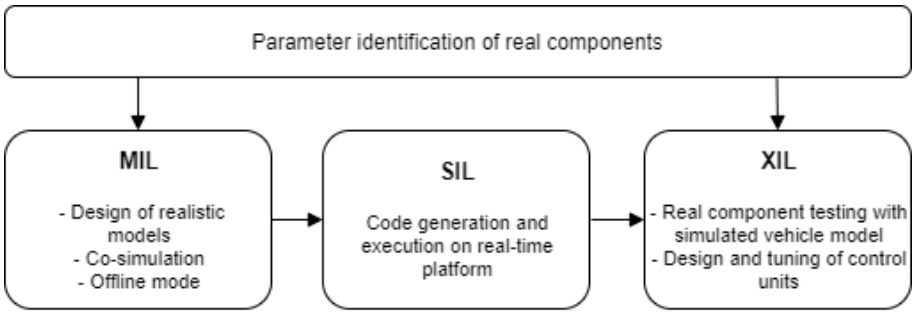


Figure 1.1: MBST development and validation stages. Adapted from Van der Auweraer, Sarrazin & Santos (2017).

1.2 Motivation

The European Commission estimated that, by 2050, CO_2 emissions in the EU would increase by 35%, congestion costs would increase to 200 billion € per year and accidents would cost over 60 billion € annually (European Commission, 2013). In fact, as Europe grows into an increasingly connected community, the transportation financial, social and environmental costs will increase proportionally. Due to this, the pursuit for smart, green and integrated transportation in the EU has become one of its priorities.

Allied to the development demands imposed by this, the fact that the transportation industry accounts for 6.3% of the EU gross domestic product intensifies the need for efficient engineering processes that keep the competitiveness and innovativeness of Europe's transportation sector (European Commission, 2013; Albers et al., 2013).

Considering this, the XiLforEV project, which stands for X-in-the-loop for Electrical Vehicles, was created. The purpose of this project is to:

> "[...] provide an experimental environment for simultaneous real-time validation and testing of electrical vehicles power-train and chassis systems using connected facilities distributed between different geographical locations." (Ivanov et al., 2019).

This project aims to use the model-based development approach for electrical vehicles, taking advantage of the capabilities of simultaneous integration of different test benches, as well as of their geographic distribution.

The system architecture under development, presented in Figure 1.2, shows the location of the different test setups. A power-train setup, a driving simulator and a suspension test rig, which correspond to the main test setups, are placed respectively in Ilmenau (Germany), Ermua and Zaragoza (Spain). Secondary test setups, that can be plugged when necessary, are included in Ljubljana (Slovenia) and Sint-Truiden (Belgium) (Ivanov et al., 2019). The distances between locations range from 216 to 1380 kilometres.

The basilar concerns and challenges of the project are the real-time capabilities of the test benches and the real-time performance and capabilities of the communication network that connects them (Schreiber et al., 2018).

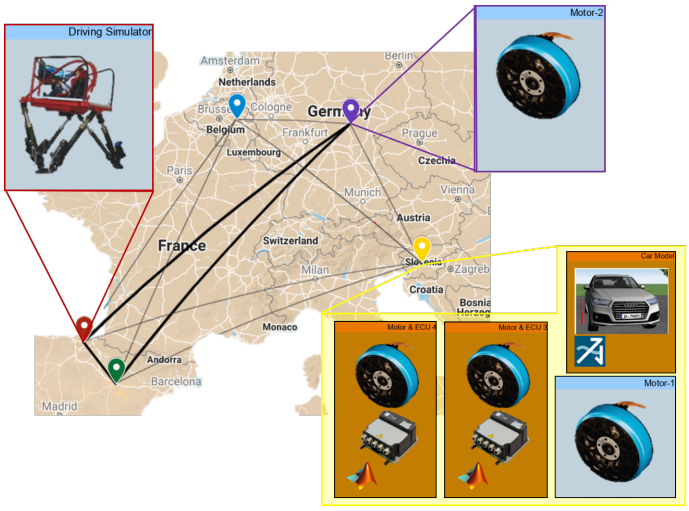


Figure 1.2: Geographically distributed architecture for XiLforEV.

The entities that partnered in the development of this project are (Bernard et al., 2019):

- **AUDI** (Germany), an original equipment manufacturer (OEM) of the automotive industry;
- **TENNECO** (Belgium), a manufacturer of vehicle chassis systems;
- **ELAPHE** (Slovenia), a manufacturer of electric power-train components;
- **SISW**, a developer of industrial and multi-domain simulation software;
- **ITAINNOVA** (Spain), a research institution with expertise in reduced order modelling, industrial electronics and manufacturing;
- **TUIL** (Germany), a research institution with expertise in automotive engineering and testing technologies.

Siemens Industry Software (SISW), in the form of SISW-NV (Belgium) and SISW-SAS (France), has long-lasting expertise in the field of system simulation and in methods for model-based development. SISW has contributed for relevant projects in the field, such as the MODELISAR and ACOSAR projects, on the development of standards for MBST methods.

This thesis was proposed and developed within an internship at SISW-NV, where the author integrated the Model-Based System Testing team.

SISW in-house-developed software for simulation and testing also provides them with modelling and testing capabilities that further aid the XiLforEV project. This tools include but are not limited to Simcenter AMEsim, a multi-domain modelling and simulation software widely used in this thesis, and Simcenter Testlab, a software for testing solutions capable of data acquisition, visualization, post-processing and reporting.

Simcenter AMESim

Given that Simcenter AMESim is widely used in this project, a brief introduction to the functioning of this simulation tool needs to be made. First, AMESim is a block modelling software

with multiple libraries, each of them referring to different physical domains. The linking of the blocks is done by assessing the coherency of the variables interchanged between them. AMESim provides also a variety of model analysis tools, of which the one most used in this work is the performance analyser. This tool evaluates the system's dynamics, finding its eigenfrequencies and defining the maximum integration step size that can be used with that model.

1.3 Objectives

The main goal of this thesis consists on the development and connection of virtual test benches for implementation of robust distributed testing solutions This can be devised as the following secondary goals:

- Develop simulation models for usage in the virtual test bench;
- Enable models for execution in real-time;
- Understand the errors created by the impairments in geographically distributed networks;
- Implement network impairments simulation models;
- Implement coupling methods for allowing the correction of these coupling errors;
- Evaluate the distributed testing results with no impairments;
- Evaluate the distributed testing results with impairments but no coupling methods;
- Evaluate the distributed testing results with impairments and coupling methods;
- Devise guidelines for robust distributed testing.

1.4 Structure

This thesis starts with a theoretical background in Chapter 2, where all the established knowledge related to this work is presented as well as its most relevant concepts and terminology. This includes the field of co-simulation, basilar to distributed testing, communication networks and a synthesis of the detected challenges concerning real-time distributed testing.

In Chapter 3 are described some of the approaches currently used when addressing these challenges. These include signal-based, model-based and energy-conservation-based methods. More attention is dedicated to the first.

The development of this thesis follows a linear approach, according to the principles of MBST. In this sense, Chapter 4 focus on the MiL stage of development, where the models are gradually built for distributed testing. The accuracy of the models in each stage of their splitting is constantly analysed.

In Chapter 5, the development of the network impairments simulation models is described. This allows for the implementation and performance evaluation of the chosen coupling methods. This corresponds to the SiL stage of development.

For the HiL stage, in Chapter 6, physical networks are included in the virtual testing scenario. This includes the creation of a procedure for assessment of the impairments over the network and the creation of a distributed virtual testing environment.

Finally, the conclusions of the work and the establishment of guidelines for future work are presented, in Chapter 7.

Chapter 2

Theoretical background

This chapter describes the theoretical concepts necessary for real-time distributed testing. First, the concept of co-simulation is presented since it is basilar for MBST. Then, the relevant aspects of communication theory, some protocols for distributed networks and the concept of network emulation are described. Lastly, there is a description of the identified challenges and issues imposed by real-time distributed testing.

2.1 Co-simulation

As explained in Chapter 1, the testing with simulation approach of MBST consists of the coupling of test setups and simulators between themselves. This is based on the concept of co-simulation, which consists of the coupling of simulators.

The concept of co-simulation has been around since the beginning of the 90's, right after the boom of simulation solutions in the late 80's. This has given enough time for this field to be studied and developed and, in 2018, Gomes et al. (2018) wrote a comprehensive revision article that brought some uniformity to the concepts and terminology of the field.

Co-simulation can be defined as the construction and simulation of systems composed of coupled simulators of subsystems, each with their respective model. The main focus of co-simulation theory is not on the models themselves but on the techniques and tools that allow the independent exchange of data across simulators. In co-simulation, simulators are treated as black-boxes, which means that the simulator can be viewed as a block capable of producing outputs based on the inputs given. The objective of this approach is to enable multidisciplinary development across different teams. This allows the use of the most suitable tools and solvers for each simulator, the distribution of the computational effort of simulation and the protection of intellectual property of the model, as it hides its internal dynamics. Furthermore, the communications between simulators needs to be addressed as the input and output variables are discrete. This raises concerns about the stability and accuracy of the co-simulation which led to the development of the techniques that assure a proper data exchange and synchronization between models (Gomes et al., 2018; Sadjina et al., 2019).

2.1.1 Terminology

The first thing to consider, given the black-box analogy presented, is that the set of trajectories followed by the state and outputs of the system can be called *behaviour trace*. This can be understood as a mapping between the time base of the simulation and the real value of the state or output. What is understood as time base can be distinguished into *wall-clock time* (τ) and *simulated time* (t). The first is related to the time the amount of time it takes to perform a given simulation, and can also be called *simulation time*. The second is the timeline for which the model is simulated, i.e. the progression of time in the simulation. These two can be defined with the relation $t = \alpha\tau$. This means that the simulated time elapsed over the interval $[0,t]$ is α units of the wall-clock time (Gomes et al., 2018). The cases of a linear relation, depicted in Figure 2.1, represent a fixed velocity of the simulation and include three distinct situations (Nguyen et al., 2019):

- slow simulation, i.e. $\alpha < 1$ and the simulation time evolves slower than the wall-clock time;
- real-time (RT) simulation, i.e. $\alpha = 1$ and the simulation time evolves at the same pace as the wall-clock time;
- fast simulation, i.e. $\alpha > 1$ and the simulation time evolves faster than the wall-clock time;

For clarity, one can look at the following example. Assuming that the simulated time corresponds to the interval $[0,1]$ seconds, $\alpha = 0.5$ implies that 2 s elapse in the real-world, meaning that the simulation is slow. On the other hand, if $\alpha = 2$, 0.5 s elapse in the real-word, meaning that the simulation is fast.

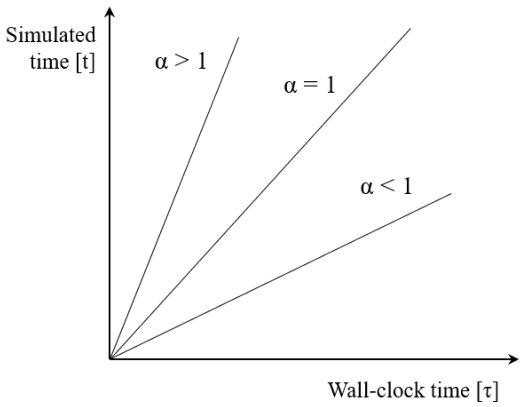


Figure 2.1: Linear relations of simulated and wall-clock times.

Another important concept is *validity*, which is a property that allows evaluation of the model. This is defined as the difference between the behaviour trace of the model and the behaviour trace of the real system (Gomes et al., 2018). This is however a simplified understanding of this concept as the validation process occurs in multiple stages of the development of models and simulations (Sargent, 2010). This topic is further explained in Section 2.1.3

To compute the behaviour trace of the model is necessary an algorithm, which is called *simulator* or *solver*. As the computer on which the simulator runs is in most cases digital, this induces a *simulation error* due to the discretization of continuous time and the impossibility of representing infinitely small values. This error can be calculated as the norm of the difference between the behaviour trace produced and the analytically calculated behaviour trace. With this, the concept of *accuracy* can be defined as the capacity of the simulator to compute the behaviour trace of the model. The correctness of the simulation depends on both accuracy and validity (Gomes et al., 2018).

The junction of a simulator with a model is called a *simulation unit* (SU) and can be described as an entity capable of producing a behaviour trace when inputs are given. As a simulator without a model can not produce a behaviour trace, the SU is the closest to the black-box analogy. Therefore, we can redefine *simulation* as the behaviour trace obtained with a SU and *co-simulation* as the behaviour trace obtained with coupled SUs. To couple SUs, an *orchestrator* is needed, which is responsible for the progression of time and for the transferring of data between SUs (Gomes et al., 2018).

For coupling purposes, each SU needs to have an interface defined, which is a set of conventions to which the other SUs need to adhere to. These can be distinguished as high-level interfaces that concern the physical level and the physical interpretation of data, or low-level which concern raw binary data and coherence between types of variables (integers, floats, char, etc.). High-level interfaces are commonly supported by low-level interfaces since ultimately most data is exchanged as binary data. The exception to this, are interfaces that exchange analogical signals. The utility of high-level interfaces is the expression of data in terms of physical variables which can correct or protect against incorrect coupling of variables (for example, coupling force with power or coupling a force expressed in N with another expressed in kN) (Sadjina et al., 2019).

Just as a SU results from the junction of a simulator and model, the composition of an orchestrator with a co-simulation is a *co-simulation unit* (co-SU). This concepts allows the hierarchical scalability of co-simulations as one can be composed of coupled co-SUs.

Regarding the type of coupling present on the co-simulation, Gomes et al. (2018) presents three main approaches which differ on the cues that trigger the data transfer between SUs. On one hand, there is discrete-event (DE)-based co-simulation where the SUs exhibit the characteristics of DE systems, i.e. the behaviour trace is changed upon the production of an event, either of the input or the states of system. The communication between SUs is also event-based, which means that at times when no events are produced there is an absence of output. On the other hand, there is continuous-time (CT)-based co-simulation where the SUs behave as continuous systems, i.e. the behaviour trace is assumed to change continuously over time. This systems can typically be described by a set of equations that relate the inputs and the behaviour trace. The communication between SUs, mainly when digital signals are exchanged, happens at defined sampling instants which means that, even-though it is not purely continuous, it is not event-based. The third approach is hybrid co-simulation which combines both CT-based and DE-based scenarios. The way in which these two distinct approaches are combined is one of the main challenges of this scenario

and there is not much consensus on it as the ideal solution depends on the application and SUs involved. However, Gomes et al. (2018) presents two methods for the joining of the DE-based and the CT-based approaches:

- **Hybrid DE** - adapt every CT SU as a DE SU and use a DE orchestrator;
- **Hybrid CT** - adapt every DE SU to become a CT SU and use a CT orchestrator.

2.1.2 Continuous-time-based co-simulation

As the components in the XiLforEV architecture are continuous systems, the focus resides now on CT-based co-simulation, as the methods and challenges imposed by this are different from the ones of DE-based co-simulation.

According to Gomes et al. (2018), CT dynamical systems can be generically described in the state space form as:

$$\dot{x} = f(x,u) \quad y = g(x,u) \quad x(0) = x_0 \tag{2.1}$$

where

x is the state vector,
$\dot{x}$ is the time derivative of x,
u is the input vector,
y is the output vector and
x_0 is the initial state.

Considering that $f(x,u)$ is differentiable, x can be approximated by a truncated Taylor series:

$$x(t+h) = x(t) + f(x(t),u(t)) \times h + \mathcal{O}(h^2) \tag{2.2}$$

where

$h \geq 0$ is the micro-step size or integration step size and
$\mathcal{O}(h^2)$ are the terms of h with order 2 or higher.

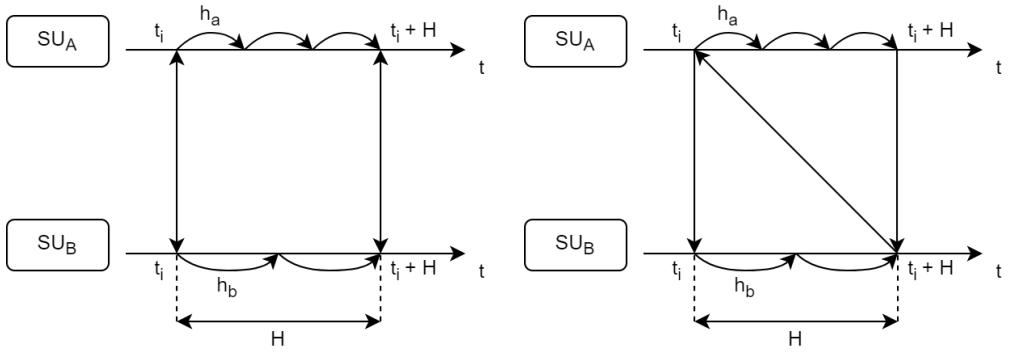
This equation is the basis of many numerical solvers for computing differential equations, which are required by CT simulations. This discretization of the continuous signal is the reason why the behaviour trace of a SU is approximated.

As each SU has its own simulator, naturally the micro-step sizes will be independent from each other. The orchestrator allows for coordination of the communication instants between SUs through a communication step size H, also known as macro-step size. This results in the solvers needing to run without updated input between macro-steps which leads to one of the challenges of CT co-simulation. In fact, the choice of the macro-step size results from a trade-off between the needed overall simulation time, increased by the transfer of data, and the achieved accuracy of the co-simulation results, which worsens with bigger macro-step sizes. In theory, for linear systems, the macro step time can be chosen from the eigenfrequencies, i.e. the natural frequencies of the system (Sadjina et al., 2019; Stettinger et al., 2013). The Nyquist-Shannon sampling theorem states that:

> "If a function *f(t)* contains no frequencies higher than W Hz, it is completely determined by giving its ordinates at a series of points spaced 1/2W seconds apart." (Shannon, 1949).

Following the sampling theorem, it is understood that the sampling frequency of a properly defined system should be at least twice its highest natural frequency. However, Stettinger et al. (2013) states the selection of the communication frequency in a different manner, saying that it should be smaller or equal to the fastest update rate of the SUs. As the update rate is the inverse of micro-step size, this appears as a more restrictive condition. Despite this, the agreement is that these are theoretical formulations and that the coupling frequency should be increased past the theoretical value until the required accuracy is met.

Considering the architecture of the co-simulation, two approaches for data exchange can be defined: *Jacobi* or *parallel scheduling* and *Gauss-Seidel* or *sequential scheduling*. A scheme for both types of scheduling is presented in Figure 2.2.



(a) Jacobi scheduling (b) Gauss-Seidel scheduling

Figure 2.2: Schematic representation of types of scheduling.

In Jacobi scheduling, the SUs exchange data at time t and independent and parallelly compute their behaviour trace for that macro-step, i.e. until time $t+H$. Each SU is responsible of extrapolating the intermediate input values based on past input data. Extrapolating data introduces a coupling error as it is based on an estimation. This error may propagate over the internal loops increasing over time. Jacobian scheduling requires smaller macro-step sizes especially if the coupling signals are sensitive to signal discontinuities caused by some extrapolation techniques (Stettinger et al., 2013; Gomes et al., 2018).

On the other hand, in Gauss-Seidel scheduling, an order of the SUs is defined so that one of them receives data at time t and independently computes its behaviour trace until time $t+H$ through extrapolation. After that, the SU feeds its output to the necessary SUs and, having the data at time t and at time $t+H$, the behaviour trace is computed in that macro-step through interpolation. This procedure is followed across the SUs with the order set by the orchestrator. The concern with this approach is selecting the SU whose behaviour trace is extrapolated as this will affect the overall solution. On the other hand, as only one extrapolation is required, the accuracy is increased due to fewer estimation errors (Stettinger et al., 2013; Gomes et al., 2018).

The coupling of SUs is still a topic of research and many techniques are proposed to diminish the coupling errors. The ones deemed relevant for the scope of this thesis are presented in Chapter 3.

As already stated, as simulators are required to compute the behaviour trace in between macro-steps, this raises some problems concerning the simulation accuracy, which were extensively addressed by Gomes et al. (2018).

Considering the system in Figure 2.3, for both the SUs (distinguished between themselves by the dashed line), the input is a force F_c and the outputs are a displacement x and a velocity v. In this scenario, the inputs and outputs can not be mapped from one SU to the other. A possible coupling strategy is stating that $F_{SU_1} = F_{SU_2}$, $v_{SU_1} = v_{SU_2}$ and $x_{SU_1} = x_{SU_2}$, but this requires the orchestrator to deduct the inputs based on the outputs, which is difficult under the black-box approach. This problem is known as *Algebraic Constraints* and is normally solved by relaxing the black-box constraints, sharing either the equations of the system or at least the sensitivity of the outputs in regards to the inputs.

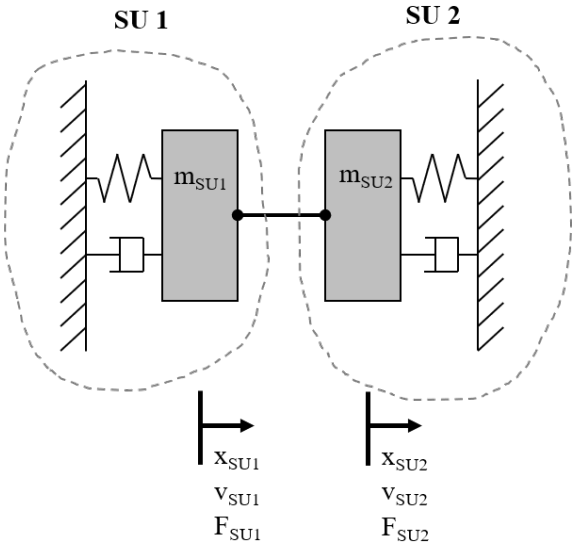


Figure 2.3: Multi-body system coupled by a massless link. Adapted from Gomes et al. (2018).

Another problem is that of *Algebraic Loops* which occurs when a variable indirectly depends on itself. This may lead to an exponential growth of the error in between macro-steps, jeopardizing the simulation results. The solution to this problem is making use of fixed-point iteration which consists of repeating the co-simulation macro-step until convergence of the SUs. This is inadvisable for real-time co-simulation due to the increased simulation time and the solution usually passes by identifying and eliminating algebraic loops.

There is also the problem of *Consistent Initialization* between simulators. This is normally solved in an initialization phase of the co-simulation where consistent values are found but the simulation time is still not evolving.

Finally there are the problems of *Compositional Convergence*, *Stability* and *Continuity*. Compositional convergence is stated as the ability of the SU to be convergent, i.e. when micro-step sizes tends to zero (0), it will mimic the analytical solution. According to some authors, when using Jacobi or Gauss-Seidel coupling, the co-simulation will be convergent if the SUs involved are convergent. For convergent SUs and co-SUs, an error estimation and correction can be applied for re-approximation. Compositional stability regards the numerical stability of the co-simulation, as the systems, the coupling approaches and the numerical solvers yield different stability properties. There are not many mitigation proposals for this besides the choice of the appropriate solutions for each scenario. Compositional continuity refers to the problem imposed by a behaviour trace of a CT system that is not continuous, which leads to a reduced accuracy as these errors tend to propagate. One of the proposed solutions is the usage of extrapolated interpolation methods.

Gomes et al. (2018) also refers to the constraints associated with real-time and delay but these will be addressed in Section 2.3.

2.1.3 Simulation fidelity

The concept of accuracy (previously introduced) is not the only way to evaluate the fidelity of a simulation. For this, the terms verification, validation, accreditation and credibility need to be explained. According to Pace (2004), verification is defined as the assessment of the design and implementation of the model in regards to its specifications, while validation concerns the ability of the model to provide reliable results, within its domain of applicability. On the other side, both accreditation and credibility concerns public and management perception of results, which are not considered relevant for the scope of this thesis.

According to Sargent (2010), verification mainly concerns the computerized implementation of the model, i.e. how the conceptual model was implemented as a computer program. This is mainly affected by the programming language used, particularly if it is a special-purpose simulation language, a general-purpose simulation language or a high level programming language (generally FORTRAN, C or C++). When a simulation language is used, the verification process focus on ensuring an error-free simulation language, on having a proper pseudo-random number generator and on a correct implementation of the model. With high level programming languages the focus is on verifying the simulation functions implementation (e.g. the time-flow functions) and that the model has been programmed correctly.

On the other hand, the validation process occurs in multiple stages of the process which are the development of the conceptual model, the operational level (i.e. validation of the accuracy of the simulation behaviour trace) and on the data level (i.e. the correctness of the data for building evaluating and testing of the model). Sargent (2010) describes some of the validation techniques used for these purposes such as:

- *Animation*, where the models operational behaviour is displayed graphically;
- *Comparison to other models*, where results are compared to the results of previously validated models;

- *Extreme conditions test*, where the plausibility of the results under extreme conditions is evaluated;
- *Face validity*, where experts on the system evaluate the model and/or its behaviour trace;
- *Multistage validation*, which consist of (1) developing the model's assumptions based on theory, (2) empirically validating the model's assumptions and (3) comparing the behaviour traces between model and real-system;
- *Operational graphics*, where different performance values are shown graphically through time
- *Sensitivity analysis*, where input or internal parameters are changed to determine their effect on the model's behaviour;
- *Traces*, where the behaviour of different entities is followed to determine if the model's logic is correct;
- *Turing Tests*, where experts are asked to distinguish between the model and real system behaviour trace.

One commonly usage of the *Comparison to other models* technique when handling co-simulation is the comparison with a single simulation of the full-model, as an ideal coupling will not introduce any errors in the simulation results (Stevic et al., 2017).

The evaluation of the models validation process usually relies on statistical methods such as confidence intervals and hypotheses test (Sargent, 2010).

2.1.4 Standards for co-simulation

With the black-box approach for the SUs included in a co-simulation, the importance of standards for the interface of the SUs becomes clear. This allows for an easier integration of SUs by avoiding the establishment of interface protocols for each co-simulation.

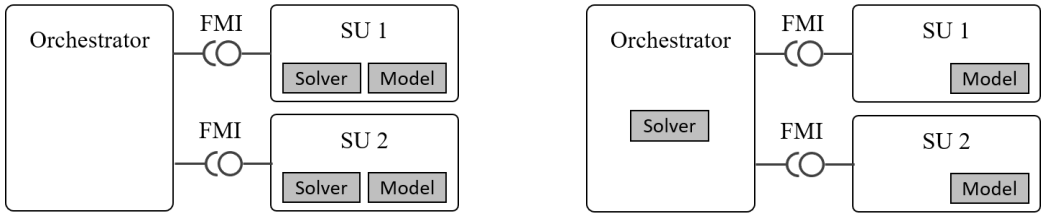
On of the most-widely accepted standards is Functional Mock-Up Interface (FMI) (Modelica Association Project FMI, n.d). Another important protocol is Distributed Co-simulation Protocol (DCP) (Modelica Association Project DCP, n.d.). The latter is based on the FMI standard but evolved to provide features regarding real-time systems integration. DCP is still in its early stages of development, therefore not having the same industrial importance of FMI. It is still presented due to its potential preponderance. Other relevant standards are High Level Architecture (HLA) (IEEE Standards Association, 2010) which was developed for large scale co-simulation, ASAM XIL (ASAM, 2020) which is an API (Application Programming Interface) standard dedicated to the communication between test benches and test automation tools and Distributed Interactive Simulation (DIS) (IEEE Standards Association, 2015), which focuses on communication protocols for both real-time and non-real-time distributed co-simulations in the military domain (Krammer et al., 2018).

Functional Mock-Up Interface (FMI)

The Functional Mock-Up Interface (FMI) was developed in a project called MODELISAR, from the Modelica Association, which focused on improving the design of systems and of embedded software for vehicles. The current version FMI 2.0 was released in 2014 and is supported by over 100 simulation and modelling tools, showing its wide acceptance (Modelica Association Project "FMI", 2013).

The idea behind FMI is that modelling tools should allow packaging of models as Functional Mock-up Units (FMUs), following the FMI specification. FMUs are archive files with standardized files (in both their format and structure) and directories. The files that carry the model information can either be the actual source files or shared object files, which protect the intellectual property of the model. Along these, metadata and documentation files are also included. FMI is a tool independent standard (since it is distributed in C and not in a specially developed language) and provides stability and compatibility between versions (Andersson, 2016; Sadjina et al., 2019; Modelica Association Project "FMI", 2013).

The standard defines two approaches: model exchange and co-simulation. With *FMI for model exchange* the interface represents a set of differential, algebraic and discrete equations relating inputs and outputs. After coupling with other models they are solved together in some external simulation environment. Therefore, the FMU for model exchange provides only the model, while the solver is supplied by the simulation software. This requires general simulation software that may be too inflexible, inaccurate or time-costly, neglecting the advantages of domain-specific simulation tools. With *FMI for co-simulation*, both the model and solver are part of the FMU and the focus is in defining an interface for data exchange between the simulators, allowing for the use of appropriate solvers for each model. This approach is based on a master/slave architecture of communication and control between the simulators, with each SU being a slave and the master being the orchestrator. FMI only specifies how the co-simulation software should interact with the models while the architecture design, time synchronization and coupling of variables are to be defined and implemented by the developer of the co-simulation. In Figure 2.4, we can see this difference synthesized as a placement of the solver (Sadjina et al., 2019).



(a) FMI for co-simulation (b) FMI for model exchange

Figure 2.4: FMI design principles. Adapted from Modelica Association Project "FMI" (2013).

Figure 2.5 illustrates the FMI slave state machine, which dictates the FMU states and transitions. It is noticeable the existence of 9 states. The first, *instantiated*, occurs with the creation of the FMU on the machine where it will be run. After this comes the *initialization mode*, where

all the variables are set to their initial values before entering the actual running phase. This phase includes 4 possible states, depending on the step execution stage: *step in progress*, *step complete*, *step failed* and *step cancelled*. From this running phase, the state machine can either go to *terminated* or *error* state, depending on whether the co-simulation has ended or an error has occurred. In any mode a fatal error can occur, leading into the *fatal* state and the subsequent FMU termination (Modelica Association Project "FMI", 2013).

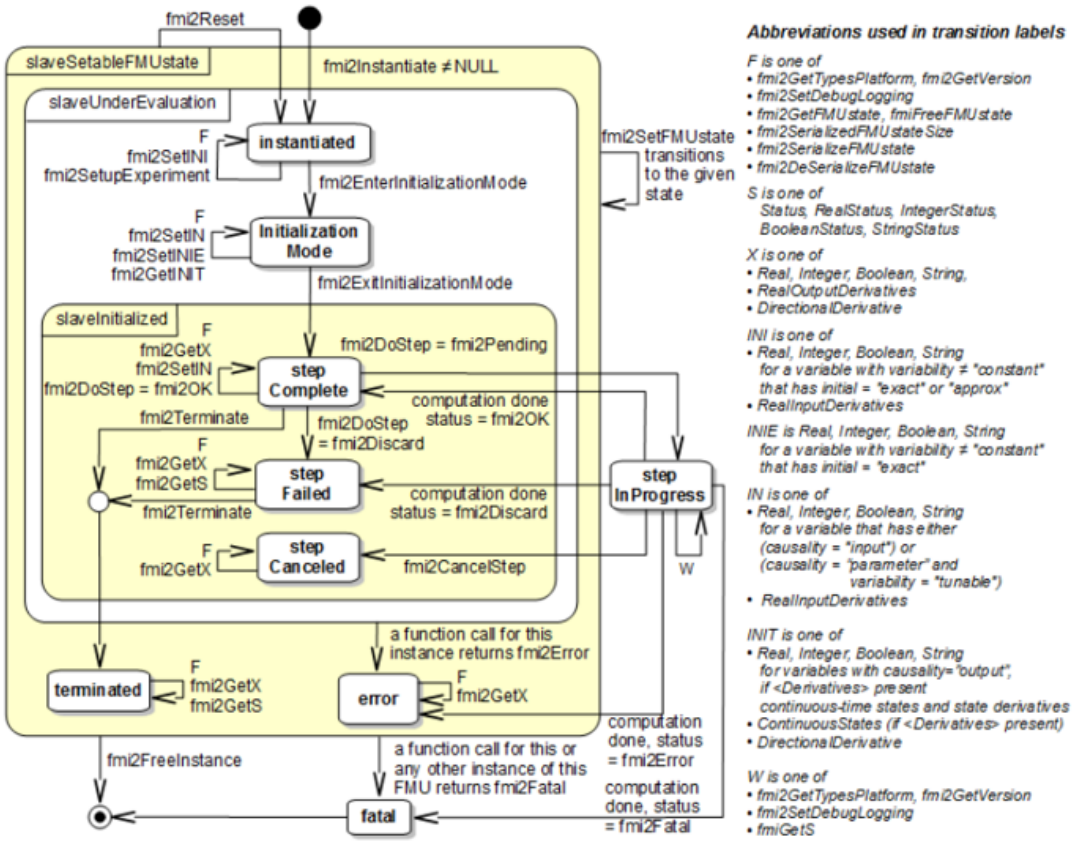


Figure 2.5: FMI Co-simulation Slave State Machine. Extracted from Modelica Association Project "FMI" (2013).

Distributed Co-Simulation Protocol (DCP)

The Distributed Co-Simulation Protocol (DCP) was developed by the ACOSAR project but, at the end of the project, a decision by the consortium led to it being integrated in the Modelica Association, which is currently the responsible and maintainer of both FMI and DCP standard. DCP is currently in version 1.0, which was released in March 2019 (Modelica Association Project DCP, 2019). Being so recent, there is barely any support for this standard at the moment, because of the effort of integration with modelling and simulation tools.

DCP focus on the coupling of systems both in real-time and non-real-time domain. Being based on the FMI standard, the goal of developing DCP was easing the integration of real-time

systems through the development of a communication layer that was not present on the FMI standard (Krammer et al., 2018). As in the FMI standard, the architecture of DCP is based on a master-slave network, with the master being responsible for the orchestration and configuration of the SUs, which are the slaves. However, this standard also defines the type of messages to exchange, called Protocol Data Units (PDUs), and defines a slave state machine and a communication protocol to handle the different type of PDUs sent and received by the slaves (Modelica Association Project DCP, 2019).

Due to this communication protocol, the interface defined by DCP can more easily integrate real-time and non-real-time systems as the communication procedure is capable of handling these two distinct situations. For this, different operating modes are defined according to the real-time requirements of the co-simulation (further explanation of the operating modes is given in Section 2.3.1). Furthermore, this protocol exists on the application level which means that it makes use of already existent transport and network protocols. This means that, even-tough DCP is currently defined only for UDP/IPv4, TCP/IPv4, RFCOMM, USB and CAN bus, it allows for integration with many other communication protocols (Modelica Association Project DCP, 2019; Krammer et al., 2018).

The state machine for the DCP slave is presented in Figure 2.6, where some similarity with the FMI state machine can be found. The *alive state* works as the instantiation phase of the FMI state machine but the initialization state is now defined into *configuration*, *preparation* and *initialization* states. In this state machine there exists also a *synchronization*, responsible for the temporal coherence across SUs. Important to notice is the distinction of real-time and non-real time operation mode. Non-real time co-simulation is processed in its own super state, while real-time co-simulation is processed in the *running* state. The termination of the DCP slave occurs either by *stoppage* or by *error*. The distinction between fatal and non-fatal errors does not exist in this state machine.

2.2 Communication network

After the explanation of co-simulation, it becomes clear that the connection between SUs is important for the quality of the results. This becomes even more relevant when the co-simulation becomes geographically distributed as the communication links introduce faults in the data exchange, i.e. communication impairments, which need to be accounted for.

2.2.1 Communication impairments

In communication theory, impairments are widely understood. However a simple explanation of the impairments and their causes is due.

First, there is *packet delay*. This can be measured as end-to-end delay (the time elapsed between a packet being sent and being received) or round-trip delay (the time elapsed between a packet being sent, retransmitted and received again at the sending side). The latter is more commonly measured given its low implementation effort. This is because, unlike with end-to-end

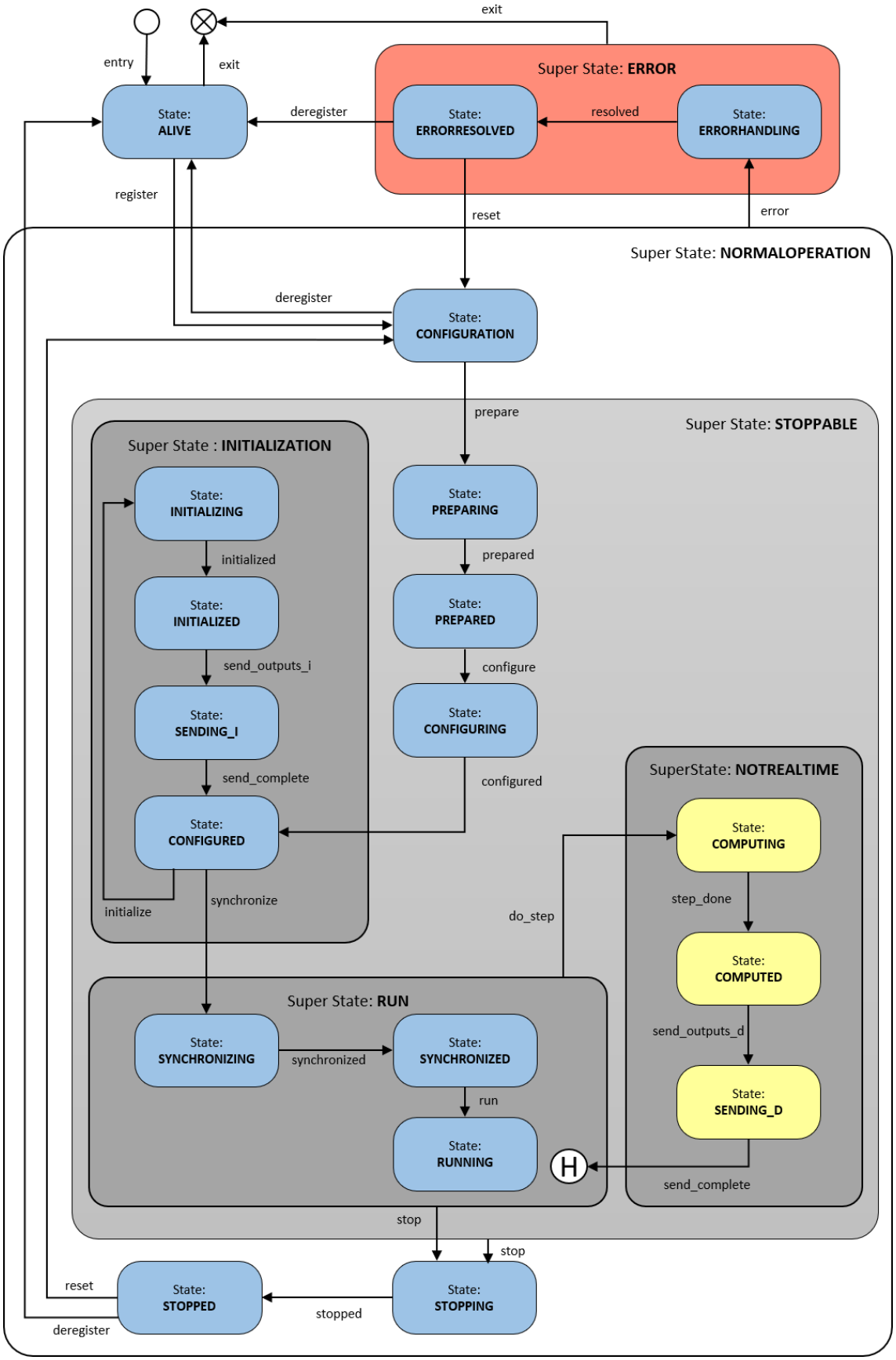


Figure 2.6: DCP Slave State Machine. Adapted from Modelica Association Project DCP (2019).

delay, there is no need for time synchronization between the network locations. There are cases however when end-to-end measurement is required due to unbalanced links, i.e. the delay is not the same in both directions. Delay can be decomposed as:

- processing delay, the time spent processing the packet at each node of the network;
- queuing delay, the time waiting in a queue for transmission at each node;
- transmission delay, the time it takes to pass the packet from node to the communication link;
- propagation delay, the time it takes for a bit to propagate over the link.

This impairment is affected by the route followed by the packet, the traffic flow at the time and eventual network failures, which makes it important to measure the delay on multiple and diverse instants, making the use of statistical analysis for evaluation of the results (Kwon, 2015; Wheeb, 2015). This leads to the concept *packet delay variation* or jitter which is the measurement of the volatility or variation of the delay. This is normally evaluated as probability density functions such as Gaussian distributions or kernel density estimators (Schreiber et al., 2018).

Another relevant measurement of the quality of the network is the *transmission rate* which is strongly related to delay. This is the maximum number of successfully delivered bits per unit of time, usually measured in bps (bits per second). The transmission rate is mostly influenced by the link bandwidth but also by the queueing of packets in the routers and switches of the network and by the congestion caused by other packets in the communication media (Wheeb, 2015; Kwon, 2015).

Also important is *packet loss* which is defined as the non-receipt of a sent packet on the receiving side. This is caused either by full queues that lead to packets being dropped, errors in the router configuration or link failures. Depending on the way that the communication is implemented, an excessive packet delay can result in a packet being lost before arriving. This impairment can be expressed as *packet delivery ratio* that is the ratio between received and transmitted packets (Wheeb, 2015; Kwon, 2015).

Other interesting phenomenons include packet duplication, due to retransmission without loss for example, packet corruption, where the packet is received but is damaged, and out of order packets, due to some being transmitter through faster routes. Another concept to bear in mind is *headroom* which is defined as the gap between a real world network and the point of failure or very poor performance.

2.2.2 Communication protocols

Important for the connection is selecting the communication protocol. Even-tough DCP supports RFCOMM, USB and CAN bus, those are not considered for the purpose of this work due their range limitation. In fact, RFCOMM has a range of 100 m at most (BluAir, n.d.), USB usually does not go further than 5 m (Payne, n.d) and CAN bus maximum distance is around 1 km (AXIOMATIC Global Electronic Solutions, 2006). So, although these protocols are used in locally distributed co-simulation, they are not feasible for wide-area networks. Considering that the

minimum distance in the distributed scenario of 216 km, as mentioned in Section 1.2, it is useless to discuss these protocols.

Furthermore, the use of dedicated mediums such as optical fibre is considered cost-prohibitive (Liu et al., 2017). For this reason, distributed co-simulation must rely on internet-based protocols given that internet infrastructures are already in place and widely available. This way, three protocols that shall be considered are User Datagram Protocol (UDP) , Transmission Control Protocol (TCP) and Real Time Transfer Protocol (RTP).

TCP is designed to provide reliable communication without overloading the network. This means that TCP incorporates services related to flow control and congestion management, by varying the transmission rate with the availability of the network. Also, by analysis of the network bottleneck links, the transmission rate can be adjusted as to not overload queues at intermediate nodes (Wheeb, 2015; Kwon, 2015).

On the other hand, UDP is designed to be a fast transmission protocol, achieving that through the lack of flow control and connection management services, unlike TCP. This leads to lower processing times and unreduced transmission rates. Even tough this introduces unreliability, as more packets get lost, it is more suitable for delay sensitive applications such as streaming services (Wheeb, 2015). An overview of the results of a performance test for both UDP and TCP are presented on Table 2.1, which corroborate the description presented for these protocols.

Table 2.1: Performance results for TCP/IP and UDP/IP. Adapted from Wheeb (2015).

	TCP	UDP
Average transmission rate [kbps]	739.45	586.59
Average packet delay [ms]	59.37	51.33
Packet delivery ratio [%]	96.92	74.13

The real-time transfer protocol (RTP) is a protocol designed for real-time multimedia streaming applications, where the the transmission rate of the signal is dynamically adjusted depending on the current congestion of the network. This contrasts with the procedure for defining a static transmission rate which is based on a set of empirical studies for determining the conditions of the network before the communication. This approach, apart from being time consuming (because the studies need to cover multiple conditions for every link), is ineffective in compensating unexpected variations of the congestion. The way this protocol implements this congestion avoidance algorithm is through definition of a dynamic transmission rate based on the Quality of Service (QoS) at that instant. This is achieved by use of Real-time Control Protocol (RTCP) in parallel with RTP. In fact, both RTCP and RTP are part of the same standard. RTCP is responsible for exchanging QoS reports that measure the network impairments at the current instant (delay, jitter and packet loss) while RTP handles the data exchange, with its transmission rate being based on the QoS report (Vogel et al., 2019; Durresi & Jain, 2005).

Vogel et al. (2019) is one of the few, if not the only one, that reports using this protocol with co-simulation having implemented its proper algorithm for defining the transmission rate. It was

implemented, in an attempt to avoid large alterations of the transmission rate, by linearly increasing the sending rate with no congestion and, in case of congestion, reducing it by a multiplicative factor, that itself decreases by half for every congestion event. This means that RTP has an initial adaptation phase, where changes in the communication occur more abruptly. The theoretical temporal evolution of the sending rate can be seen in Figure 2.7. As RTP functions on the application layer of the 5-layer network model (also known as TCP/IP network model), the underlying transport protocol used was UDP. This is common due to the fact that TCP already has its underlying compensation algorithms.

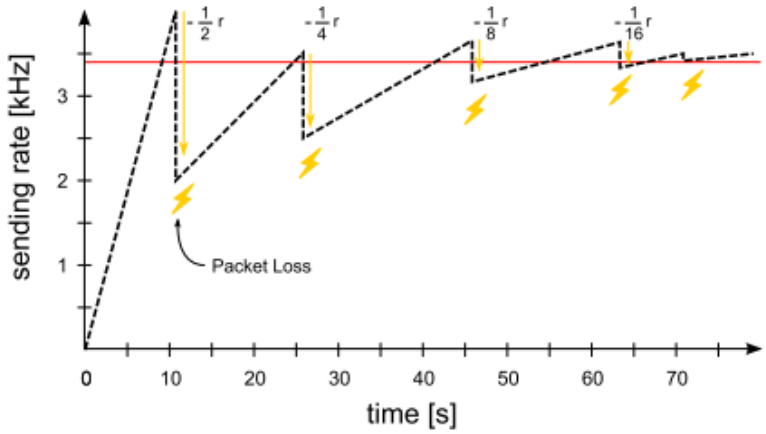


Figure 2.7: Transmission rate temporal adaptation. Extracted from Vogel et al. (2019).

A comparison of TCP/IP with RTP is provided by Ma & Gao (2012). It is important to notice that the metrics used in this study are slightly different from the ones defined in Section 2.2.1. *Data integrity ratio* is used instead of packet delivery ratio, as damaged packets are also counted. It is also evaluated the *control load* of the communication. This is because, unlike UDP, both TCP and RTP have control algorithms that induce extra congestion in the network, affecting its condition. Control load is therefore defined as the ratio between the amount of control data and the amount of total data exchanged. Due to an initial adaptation phase present in RTP, the results in Table 2.2 correspond to a period after this phase. The results show, that there is a slight decrease on the data integrity ratio compared to TCP, but that there is great improvement on the average packet delay and on the control load. This shows that RTP can maintain a quite reliable communication while providing with more real-time capable data exchange.

Table 2.2: TCP and RTP performance comparison. Adapted from Ma & Gao (2012).

	TCP	RTP
Average data integrity ratio [%]	100.00	97.92
Average packet delay [ms]	522.58	29.92
Average control load [%]	36.15	13.77

As the experimental setup is different from the one that originated Table 2.1, a direct comparison of RTP with UDP can not be made. This can be witnessed by the complete discrepancy of the results for TCP in both tests.

2.2.3 Network Emulation

When researching communication networks or the influence they have on the experimental setup, it is important to be in control of their condition, mainly for consistency across the results. For this reason, experimental procedures mostly run on local area networks (LANs). Due to it being a dedicated close-range network, it provides nearly ideal conditions which are not representative of the real network to be used, namely for geographically distributed testing (Hemminger, 2005).

One of the solutions for testing with impaired networks in LAN is through *network emulation*. This means that a network emulator is introduced in the network to alter the traffic flow for it to present the desired behaviour. This allows for creation of a diversity of network scenarios on which the procedure can be run. Another solution is network simulation but this introduces issues about the fidelity of simulation. This is not a concern in emulation as it replicates every single component of the network, either by simulation or incorporation of specific hardware components in the network (iTrinegy Ltd, 2015; Hemminger, 2005).

Like communication networks there are also different topologies of network emulation, such as (iTrinegy Ltd, 2015):

- *point to point*: two endpoints with one fixed single path
- *point to point with multiple links*: two endpoints with multiple fixed paths
- *point to point with multiple hop*: two endpoints but with two or more network hops between the endpoints.
- *interconnected networks*: multiple nodes are linked to each other through many differently impaired links.

One of the issues when selecting an emulator is the inherent impairment. By inserting a new device on the network there are inherent impairments related to emulation, such as processing, transmitting and receiving of the packets. Selecting an emulator that can counterbalance these impairments is vital for a good emulation (iTrinegy Ltd, 2015).

However, simulation may be preferred to emulation in certain cases. Mainly, it is necessary to evaluate the accuracy required for the network emulation/simulation. Having an emulator requires further investment while most modelling tools provide signal modelling capabilities that can, to a certain extent, model the communication network (Hemminger, 2005).

2.3 Challenges for real-time distributed testing

When dealing with RT co-simulation there is an extra set of requirements. As explained in Section 2.1.1, for a RT simulation the simulated time needs to match the wall-clock time and be time accurate. The difficulty of implementing this increases with co-simulation as both the SUs need to share the same timeline. Furthermore, as the RT co-simulation often includes physical components, robustness to measurement noise must be considered. Lastly, as explained in Section 2.2, the quality of the network affects the data exchange and, consequently, the quality of a co-simulation in RT (Gomes et al., 2018; Nguyen et al., 2019; Stettinger et al., 2013).

2.3.1 Real-time requirements

First, there is the need to verify the conditions that make a simulator RT compliant. A sufficient condition for this, is that the computational cycle (computation plus idle time) is equal to the macro-step size of the simulation. This way, there can be two types of systems: *online systems*, where the computation time is equal to the macro-step size and *offline systems* where the computation time is lower than the macro-step size (fast simulation) and simulation idling is activated until the next macro step. Slow simulations can not be classified as online or offline systems, which makes them non-compliant with RT. For that, the system either has to be simplified or the time step increased, with the consequent losses in the accuracy of the simulation (Nguyen et al., 2019; Stettinger et al., 2013).

Apart from this definition based on the computational costs, there are proper RT characteristics that allow for the evaluation of the RT capability of a system. First, there is *determinism* which states that a giver set of states and inputs should generate an unambiguous set of outputs and next states. This in fact, is common to most systems. *Promptness* on the other hand is a characteristic of time critical systems and relates to the capacity of the system to respond in time. On the real-time demands of a scenario, Table 2.3 makes the distinction between the systems response to unmet real-time deadlines. The most distinctive are hard RT, where the system fails on the situation, soft RT, where the system continues but with reduced quality, and non RT, where there are no demands for RT capability. Firm and near RT are intermediate levels, which often are not considered in the literature. Lastly, *reliability* is defined as the capacity of the system to support failure detection, troubleshooting and resuming normal operation after failure, ensuring safe operation even if the system is damaged. This is important for the trustworthiness of the results and for fail-safe operation, and the solution passes by equipping the system with redundancies (Schreiber et al., 2018).

2.3.2 Time synchronization

For real-time co-simulation, besides the RT compliant timeline, a synchronization of the timelines across SUs is required. The offline systems need to be synced to the online ones so that

Table 2.3: Real-time levels. Adapted from Schreiber et al. (2018).

level	**description**
hard RT	The RT system has to ensure that a task is done until a deadline. Missing the deadline causes a system failure.
firm RT	Firm RT system tolerates infrequent missing of deadlines while processing. However, exceeding the deadline criteria reduces the quality of service. The outcome after missing a deadline can not be used.
soft RT	Soft RT systems desires deadlines. For a certain threshold it is allowed to miss a deadline. The outcome after missing a deadline is usable, but reduces the quality of service.
near RT	The term near RT is not well defined in publications, literature or standards. The authors propose to define near RT as a deterministic delay of process outcomes. This delay does not influence the quality of service significantly.
non RT	Non RT processes do not require an immediate execution.

information is exchanged at the correct coupling instants. Time synchronization requires the timelines to respect the same time horizon. It is therefore, to some extent, a matter of scheduling as already discussed in Section 2.1.2. In light of the promptness demands of a real-time system, parallel scheduling seems the most suitable approach, given that parallel processing is faster than sequential processing. However, the choice is not straightforward because, depending on how fast the offline systems are, sequential scheduling may be able to cope with the real-time constraints. This would be preferential as parallel scheduling introduces bigger coupling errors (Stettinger et al., 2013; Nguyen et al., 2019).

The synchronization methods are, therefore, a set of techniques that chain the events among simulators to a desired order, i.e. it maps the events of different SUs onto the same timeline. In continuous time co-simulation, these events are related to the computational cycle of the simulators by their macro-steps. However, for proper synchronization of these events clock synchronization is usually in place.

Regarding this topic, two main concepts should be discussed. The first is clock synchronization protocols, in which the more common are Network Time Protocol (NTP) and Precision Time Protocol (PTP). The goal of these protocols is ensuring time synchronization across the network, not necessarily in terms of absolute real-world time but by the calculation of the clock offset in relation to more precise clocks. The way this offset is calculated is through the exchange of time stamps between the elements of the network. The offset calculation is similar between the two protocols but differences exist on the communication architecture and procedure.

NTP is based on a server-client architecture. In the network, there exists at least a primary server whose clock is considered as the reference time. This server provides the reference clock value to the clients and secondary services requesting it. The concept of primary and secondary servers allows to stratify the network in terms of its distance to the reference (primary server). An example of an NTP network can be found in Figure 2.8a. As NTP relies on server-client

communication, the decision to synchronize the clock comes from the client. Having understood this, one can look at Figure 2.8b for clarification on the communication procedure. It is visible that a first message is sent to the server, carrying the a timestamp of the expected sending instant (CT1). The server, after the message receipt, replies to the client sending both the timestamp of the receiving instant of the first message (ST1), as well as of the sending of this last one (ST2). Lastly, the client gets a timestamp of the receipt instant of the server reply (CT2) (Mills, 1991; Neagoe, Cristea & Banica, 2006). These timestamps allow the calculation of the client offset in regards to the server, according to Equations (2.3) to (2.5).

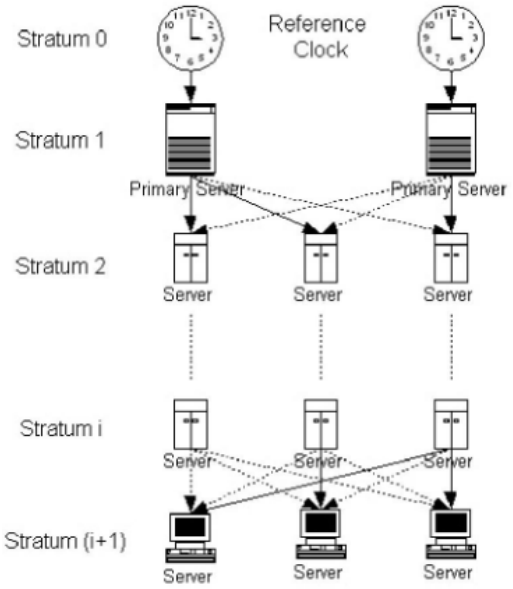
$$offset = (ST1 - CT1) - delay \tag{2.3}$$

where,

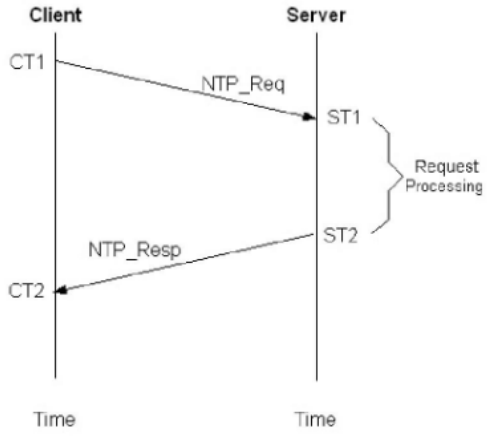
$$\text{delay} = \frac{NTP_{Reqdelay} + NTP_{Respdelay}}{2} = \frac{(ST1 - CT1) + (CT2 - ST2)}{2} \tag{2.4}$$

Combining Equations (2.3) and (2.4) results in:

$$\text{offset} = \frac{(ST1 - CT1) - (CT2 - ST2)}{2} \tag{2.5}$$



(a) Typical hierchical network. Extracted from Neagoe, Cristea & Banica (2006).



(b) Communication procedure. Extracted from Neagoe, Cristea & Banica (2006).

Figure 2.8: Network Time Protocol (NTP).

In terms of precision, Neagoe, Cristea & Banica (2006) states that a massive study over the internet concluded that NTP achieves an average precision of 29 ms and that 50 % of the NTP clocks had at least 21 ms of precision. Depending on the macro-step size of the distributed testing, this level of precision might not be enough.

The PTP on the other hand relies on a master-slave architecture. This means that, in each link, the node with the highest precision clock is called the master and demands its slaves to be synced to it. Looking at Figure 2.9a, it is visible a typical network topology of this protocol, and the subsequent hierarchy definition. First, there is the grandmaster clock which is the highest precision clock of the network. The boundary clock is responsible for starting and terminating the time distribution. These type of clocks are commonly used in network gateways. The slave clock synchronizes to a master clock but is not responsible for the synchronization of any other clocks. Lastly, there exists the transparent clock which acts as a routing devices that also compensates for the time spent on the device. Generally, transparent clocks represent devices on the network that do not require syncing but that should be PTP-aware, in the sense that they should not influence the delay. The hierarchical distribution of the network into slaves and masters is self-made, i.e. there exists a communication procedure where each clock precision is shared among themselves and the clocks are subsequently ranked. In the interaction between clocks, looking at the rank of the clocks is enough to define which is the slave and which is the master of that link. In terms of offset calculation, Equations (2.3) to (2.5) are also used. However, the communication for delay and offset measurement differs. At first a *sync* message is sent from master to slave, carrying the sending timestamp (t_1). The slave registers the receiving time of the sync message (t_2), only lacking the delay knowledge to calculate the offset (Equation (2.3)). By requesting the delay to the master, and corresponding response, the slave is able to calculate the delay by an equation similar to Equation (2.4) and, finally, it is possible to calculate the clock offset and correct it (Watt et al., 2016; Neagoe, Cristea & Banica, 2006). For clarity purposes, the offset equation can be rewritten according to the nomenclature of Figure 2.9b as can be seen in Equation (2.6).

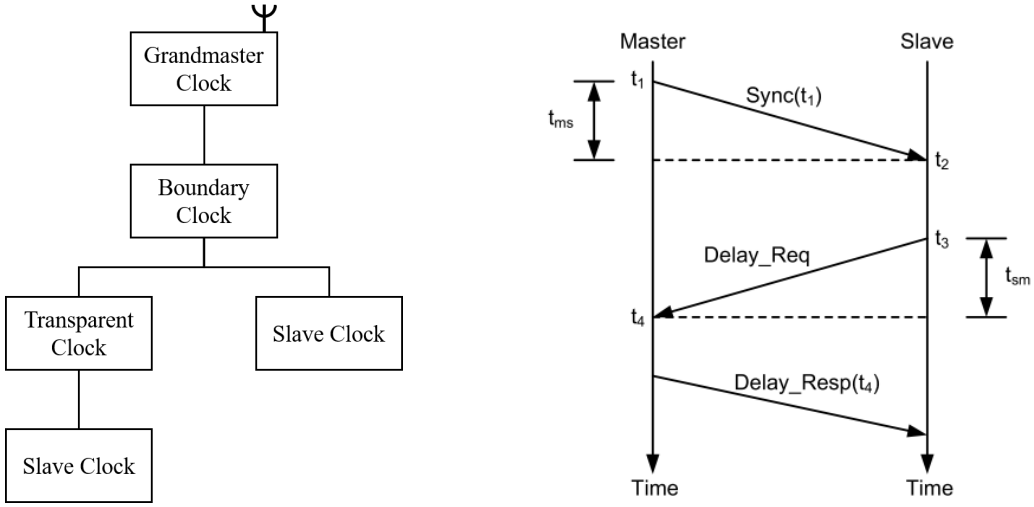
$$\text{offset} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.6}$$

The precision of PTP ranges from 100 ns to 1 μs. However, for this level of precision, hardware capability is a concern given that standard computers do not possess clocks with that resolution. In terms of software, the implementation cost is also higher than with NTP. For this reason, the choice of the protocol should only be made after an analysis of the clock precision requirements (Watt et al., 2016).

As noticed, both NTP and PTP rely on high-precision clocks for the network time synchronization. Some solutions exist in this sense, namely GPS signal clocks, atomic clocks and high precision local oscillators (Neagoe, Cristea & Banica, 2006). Again the solution depends on the precision desired for the clock signals.

2.3.3 Real-time coupling

The first thing to consider when looking at the distributed testing scenario is the effect of network impairments on the real-time capabilities of the system. The delay affects the promptness of the RT system given that required data is not available at the necessary time. The delay variation affects the determinism desired for RT as data delivery time can not be assured. In this sense, one



(a) Typical hierarchical network. Adapted from Watt et al. (2016).

(b) Communication procedure. Extracted from Watt et al. (2016).

Figure 2.9: Precision Time Protocol (PTP).

could say that the bigger the variation, the less deterministic is the distributed testing scenario. In terms of reliability, this is affected by the message loss. This is understandable given that message loss causes the SUs to diverge from one another (Schreiber et al., 2018).

This leads to the necessity of methods capable of coping with the network impairments. These techniques should, based on information of the model or the exchanged signals, be able of generating inputs to the SUs that compensate the delay in the connection and potential information losses (Stettinger et al., 2013). These techniques are discussed in Chapter 3.

Furthermore, the introduction of physical components on the system brings the problem of signals carrying noise. This noise is produced by the usage of physical sensors for the measurement of the interface variables. The same techniques discussed for the handling of network impairments can also be analysed in terms of their noise compensation capabilities (Stettinger et al., 2013).

The capability to handle network impairments and noise necessarily leads to an approximated coupling signal, which is not error-free. The fact that the signal is approximated, and not the ideal coupling signal, brings problems in terms of accuracy and stability of the distributed testing scenario. These issues were discussed in Section 2.1.2. Additionally, there needs to be fatal error recognition and safe-termination mechanisms. Because of the signal approximation, there may be cases where energy is being created in the system via the coupling mechanisms which is a major concern when dealing with physical systems (Stettinger et al., 2013).

Lastly, there is the mater of iterative or non-iterative coupling. Even tough iterative couplers solve the majority of instability issues, the computational cycle time is not assessable. If it happens to be higher than the macro-step size, hard RT conditions are not met. The safe option is using a non-iterative coupler, even with the issues it may cause. However, iterative couplers can be used when the computational cycle is so fast that it is assured it will not interfere with RT requisites (Stettinger et al., 2013).

Chapter 3

Literature Review

After having understood the concepts and challenges of distributed testing, this chapter aims to explain some of the mitigation techniques currently being used for this type of solutions. In this sense, the coupling techniques can be distinguished as:

- Signal based techniques, which make use of numerical extrapolation and past values to generate the approximated signal;
- Model based techniques, which, based on a relevant amount of data, create a numerical model that is an approximated description of the signal;
- Energy-conservation based techniques, which apply the principles of energy conservation on the interface to prevent the problems of instability on the signal approximation .

3.1 Signal-based coupling

The signal-based techniques differ in the used numerical extrapolation methods. In this sense, there can be polynomial extrapolation techniques such as zero-order hold (ZOH), first-order hold (FOH), second-order hold (SOH), whose behaviour is depicted in the Figure 3.1. As polynomial extrapolation doesn't include data filtering, it is prone to noise and numerical errors. Besides that, this type of methods can create signal discontinuities that affect the coupling by inducing unwanted dynamic behaviour on the system (Stettinger et al., 2013; Gomes et al., 2018).

Other methods, such as linear curve fitting and waveform properties methods, are also based on signal extrapolation. However, they do not rely on the same recent data using, instead, linear regressions or frequency aspects of the signal, respectively.

3.1.1 Zero-order hold (ZOH)

Zero-order hold (ZOH) is the lowest order polynomial extrapolation and can also be called constant extrapolation. As can be seen in Figure 3.2, this technique consists on the holding of each value until a more recent value is available. It is understandable that this technique, with low enough sampling rates, can provide a relatively smooth signal reconstruction. Also, as it uses only

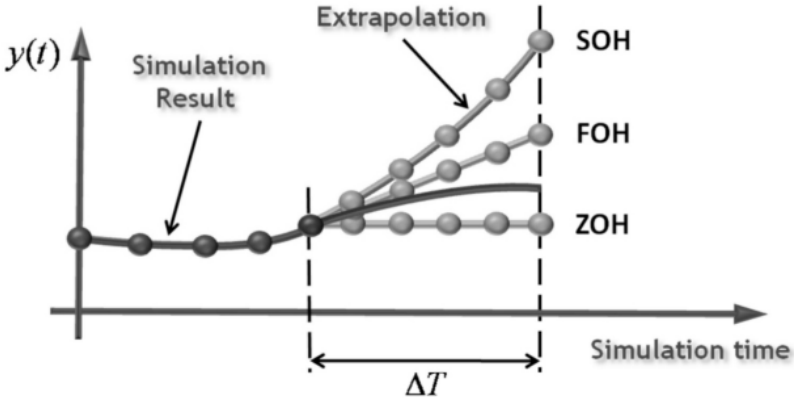


Figure 3.1: Polynomial extrapolation techniques. Extracted from Stettinger et al. (2013).

one received value at a time, values carrying noise are not kept for long, therefore not propagating the error much. However, as this is a constant extrapolation, there is no understanding of the signal evolution, hence failing to compensate the delay.

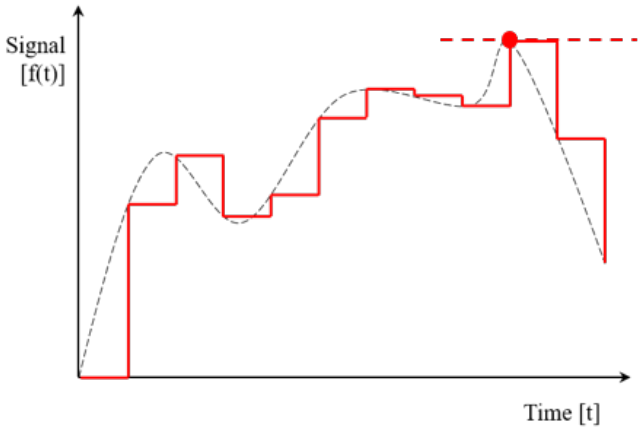


Figure 3.2: Zero-order hold signal approximation.

3.1.2 First-order hold (FOH)

First-order hold (FOH) is a first order polynomial extrapolation, also called linear extrapolation. It makes use of the last two available values to define a line on which to extrapolate. Its behaviour is depicted in Figure 3.3. As two values are being used, it is more prone to noise error propagation than ZOH. However, since there is a sense of the signal tendency, this method is more capable of compensating the delay.

FOH is slightly more complex than ZOH, mathematically speaking, therefore needing a closer look at the equations that define it. These mathematical expressions are laid out by Hoel (1965). Equation (3.1) describes the extrapolated signal as a function of time.

$$P_1(t) = \frac{y_1(t-t_0) + y_0(t_1-t)}{t_1-t_0} \tag{3.1}$$

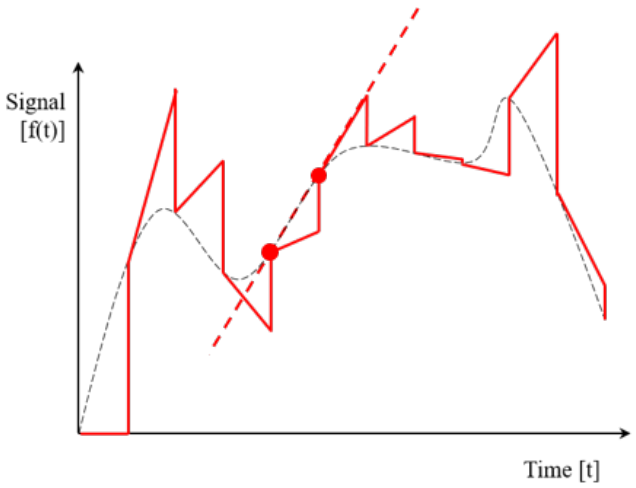


Figure 3.3: First-order hold signal approximation.

where

t	is the time for which the signal is being extrapolated,
P_1	is the value of the extrapolated signal,
t_0, y_0	are the time and value coordinates of one of the points and
t_1, y_1	are the time and value coordinates of the other point.

For simplicity, this equation can also be written as:

$$P_1(x) = y_0 L_0 + y_1 L_1 \tag{3.2}$$

where,

$$L_0(x) = \frac{x_1 - x}{x_1 - x_0} \tag{3.3}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0} \tag{3.4}$$

3.1.3 Second-order hold (SOH)

This extrapolation technique is based on second-order polynomial extrapolation, also being called quadratic extrapolation. It uses the last three past values to define a quadratic function on which to extrapolate. Its behaviour can be seen in Figure 3.4, where the continuous line represents the points being interpolated and the dotted line represents the points being extrapolated. In terms of delay compensation, this is expected to perform better than FOH given the higher order. However this also means, that values carrying noise are being used for longer which can lead to higher error propagation.

The mathematical formulation of SOH can be seen in Equation (3.5) which follows the same simplification and notation used for FOH equations (Hoel, 1965).

$$P_2(x) = y_0 L_0 + y_1 L_1 + y_2 L_2 \tag{3.5}$$

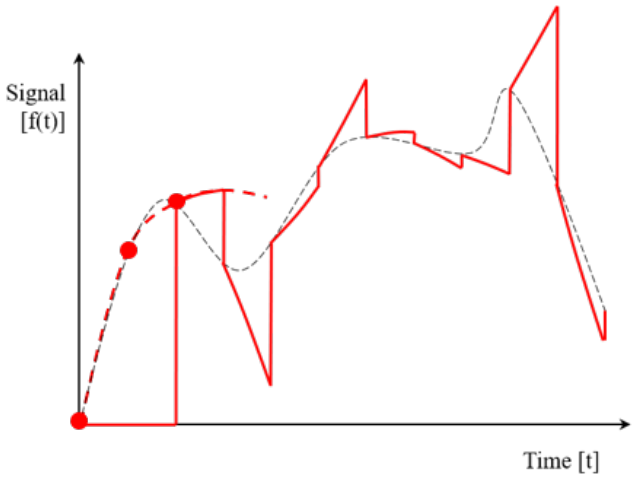


Figure 3.4: Second-order hold signal approximation.

where,

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \tag{3.6}$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} \tag{3.7}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \tag{3.8}$$

3.1.4 Linear curve fitting (LCF)

This method, proposed by Liu et al. (2017), uses the principles of linear regression for the signal approximation (Figure 3.5). This method was proposed for electrical signals and is based on the assumption that, if the time interval between sample points is small enough, the signal can be extrapolated linearly. This linear fitting means that, as more than two points are being used, the linear equation found is the one than minimizes the error for those points. With this method, it is expected to have some correction of noise, as signals further away from the trend-line lose the relevance they had in FOH method. This requires the finding of an appropriate amount of past data that compromises between the noise correction and a correct approximation of the signal. Cases of an incorrect approximation can be, for example, using so many past values that the linear regression expresses the average value of the signal instead of the actual signal. This would happen in a sine wave where data corresponding to a whole period was being used.

The linear curve fitting equation can be found by minimization of the error function between a linear equation and the actual retrieved values. Using Equation (3.9):

$$error = \sum_{i=1}^{N} (y_i - (at_i + b))^2 \tag{3.9}$$

the error is expressed as a sum of squared differences, where:

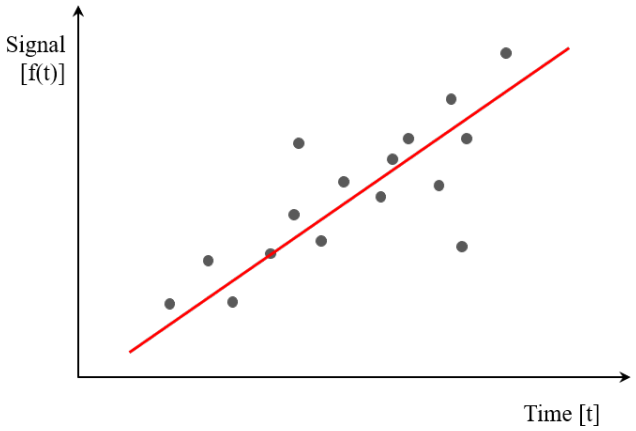


Figure 3.5: Linear regression.

t_i, y_i are the sample points coordinates,
$at_i + b$ is the linear equation, with a and b being our unknown variables,

This equation can be solved for a using the partial derivative of error with respect to both the unknown variables:

$$\frac{\partial error}{\partial a} = -2\sum_{i=1}^{N} x_i(y_i - ax_i - b) = 0 \tag{3.10}$$

$$\frac{\partial error}{\partial b} = -2\sum_{i=1}^{N} (y_i - ax_i - b) = 0 \tag{3.11}$$

Solving Equation (3.11) for b we get that

$$b = \frac{1}{N}\sum_{i=1}^{N} (y_i - ax_i) \tag{3.12}$$

Replacing b in Equation (3.10) and solving for a

$$a = \frac{\sum_{i=1}^{N}(x_i y_i) - \frac{1}{N}\sum_{i=1}^{N} y_i \sum_{i=1}^{N} x_i}{\sum_{i=1}^{N} x_i^2 - \frac{1}{N}(\sum_{i=1}^{N} x_i)^2} \tag{3.13}$$

a can now be used for the calculation of the approximated value of the signal by Equation (3.14).

$$\widetilde{y}(t) = y_0 + a(t - t_0) \tag{3.14}$$

where

t_0, y_0 are the last received sample point coordinates,
a is the linear equation slope,
t is the time for which the value is being extrapolated and
$\widetilde{y}(t)$ is the extrapolated value.

Liu et al. (2017) proved the accuracy of this method for 15 ms latency, which is not representative of many geographically distributed testing scenarios. Further testing of this linear predictor would be required to evaluate its robustness. Furthermore, the signals used in the article do not carry noise, not allowing judgement of the noise correction capabilities of this technique.

All the signal-based coupling methods described so far suffer from the same problem. Like it can be seen in Figure 3.6, the extrapolation function is abruptly changing every time new sample points are available. This means that the approximated signal is not differentiable and loses the continuity properties of the original signal. This may be relevant because non-differentiable signals can induce unwanted dynamic behaviour on the system. Interpolated extrapolation aims to combine the continuity of an interpolated signal while being able to extrapolate. Busch (2016) provides an analysis of the influences of continuous and differentiable approximated signals on the overall accuracy and numerical stability.

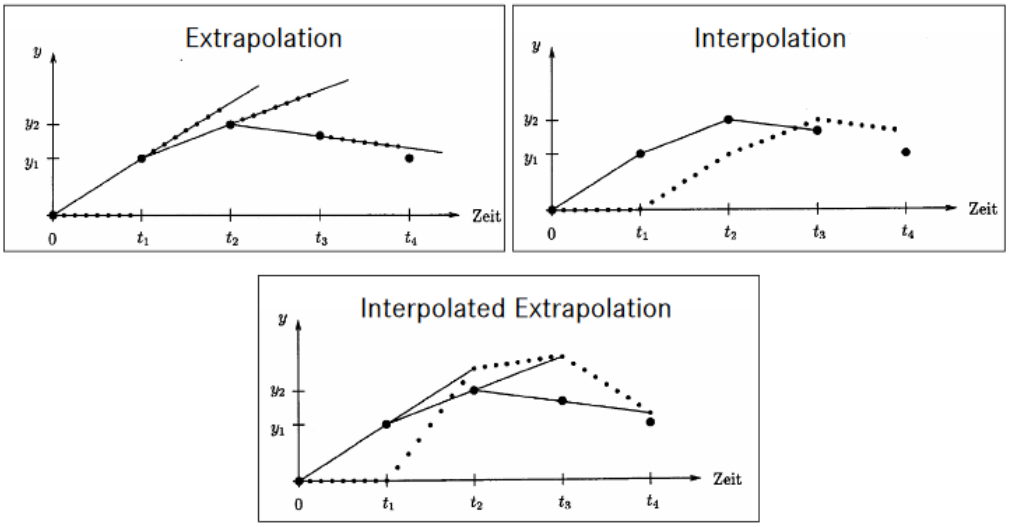


Figure 3.6: Extrapolation, interpolation and interpolated extrapolation. Extracted from Dronka & Rauh (n.d.).

3.1.5 Waveform properties

An alternative signal-based approach is the use of different representations of the signal. In this sense, there are ways to represent a waveform that allow for a better prediction than the ones provided by polynomials. This is advantageous because these types of parameters usually vary slower and are more representative of the signal rather than its instantaneous values. Stevic et al. (2017) proposed two different approaches based on the waveform parameters.

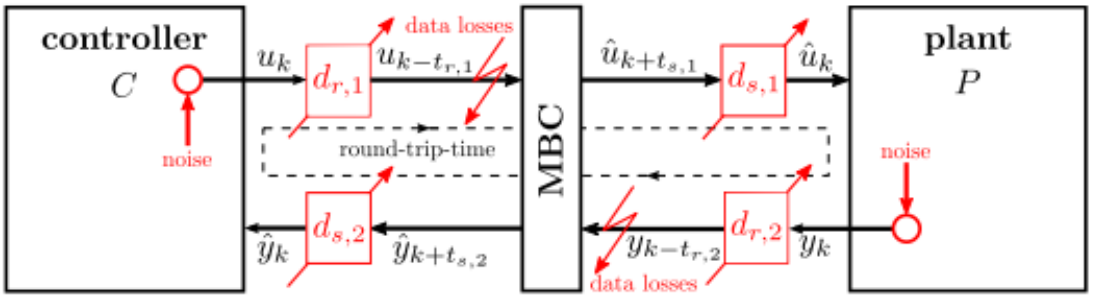
First, there is the representation of a sine wave in terms of its Root Mean Square (RMS) value, frequency and phase. This type of representation is widely used in electrical engineer. On the other hand, the wave can be on the frequency domain represented by dynamic phasors, i.e. time-varying Fourier coefficients.

The results presented by Stevic et al. (2017) are promising but fail to cope with signal transients, not showing significant improvements in comparison with the remaining signal-based coupling methods. Furthermore, waveform properties may not be made available by the subsystem which would require the implementation of techniques (such as Fourier transforms) that might raise issues by themselves.

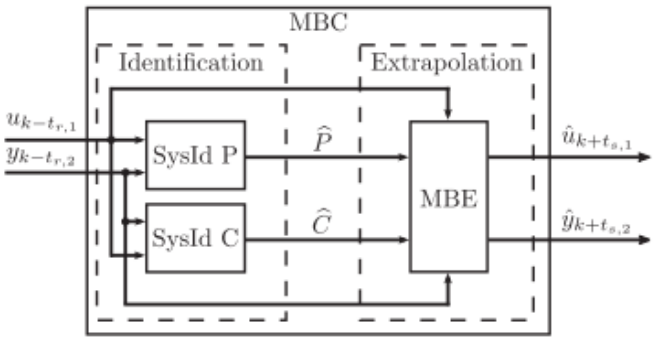
3.2 Model-based coupling

The work developed by Stettinger et al. (2013, 2014b) focuses on the development of model-based coupling. This means that, instead of using the past signal values (like in signal-based coupling), a description of the subsystems local behaviour is constructed for prediction. The subsystems are modelled as a linear time-varying system of second order through online system identification techniques.

The setup for this technique can be seen in Figure 3.7a, where both receiving and sending delays (d_r and d_s) and data losses are exemplified. Moreover, the internal structure is depicted in Figure 3.7b, where the identification and extrapolation parts are distinguishable.



(a) Element placement



(b) Internal structure

Figure 3.7: Model-based coupling element. Extracted from Stettinger et al. (2017).

The identification part is responsible for the processing of received data to find the coefficients of the second order linear equation that describe the system. This can be done by either Recursive Least Squares identification or by Extended Kalman Filter (Stettinger et al., 2017). Due to the

mathematical complexity of these methods, a comprehensive explanation is not provided herein. The author advises (Ljung, 1999) for that purpose.

In the extrapolation phase, having the knowledge of both systems an iterative extrapolation approach can be applied in order to compensate for the delay and noise of the signal. The rate at which the identified model parameters are updated results from a compromise between the accuracy and the noise correction desired (Stettinger et al., 2014b).

Furthermore, this method requires an initial system identification phase called the learning phase. In this period, while the initial estimated model is being built, signal-based coupling is used. This period should last approximately 10 times the longest time constant relevant for the system. After this, an activation period appears where both signal-based and model-based coupling are used in varying proportion until only model-based coupling is present (Stettinger et al., 2014b).

The results of this method are presented by Stettinger et al. (2014a,b, 2017) showing very promising results in comparison to signal-based coupling. However Stettinger et al. (2014b) points out that the parametrization of the system identification requires expert knowledge, hindering the implementation of this method. Furthermore, by relying on signal-based methods for initialization, model-based coupling is not able to solve problems that may rise during this phase.

A similar method using Kalman filters is presented by Brembeck et al. (2014). The proposed procedure is slightly different but it provides implementation details in accordance with FMI standard, which can be helpful. Furthermore, Feki et al. (2016) proposes another model-based method based on a polynomial extrapolation and estimated models. Although interesting results were achieved, this method was developed and tested for non-distributed co-simulation scenarios. Its ability to compensate higher delays was not proved.

3.3 Energy-conservation-based coupling

This coupling method is based on energy conservation principles. This means that, if both the subsystems are conservative systems, the coupled system should be conservative as well. This is because the interface is not supposed to affect the system, meaning that it should not interfere in the energy balances. This way, by evaluating the power exchanges between the system, the residual power created by the interface can be evaluated, as exemplified in Figure 3.8. This allows to have real-time sense of the co-simulation accuracy, therefore aiding the implementation of adaptive error correction methods. This could be, for example, the usage of variable macro-step size, according to the current accuracy of the co-simulation. This method is achieved by the incorporation of a power bond in the co-simulation interface. This power bond results, for example, from the combination of a flow with an effort (Sadjina et al., 2017, 2019).

The creation of energy happens because each SU is being solved separately, leading to numerical discrepancies. This may raise issues concerning the stability of the co-simulation, as energy is being created or destroyed even tough the systems are conservative. This is highly problematic for physical systems and is one of the current main concerns of co-simulation solutions.

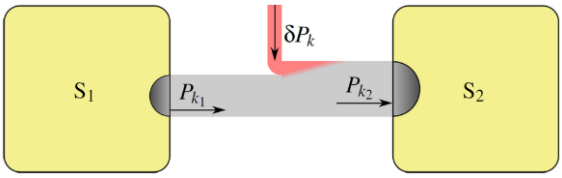


Figure 3.8: Residual power in co-simulation. Extracted from Sadjina et al. (2019).

Chapter 4

Model-in-the-loop stage

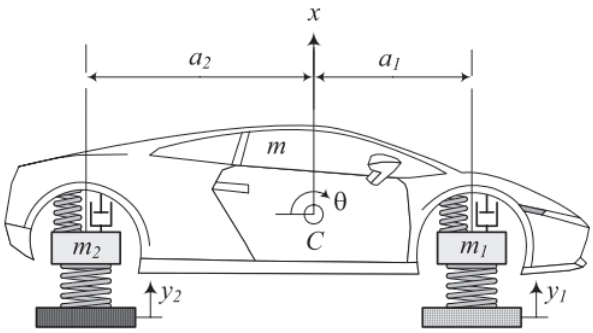
As explained in Section 1.4, the Model-in-the-loop (MIL) stage includes the steps concerning the building of the model. In this sense, every work exclusively made in the modelling software is considered part of the MIL stage. Given this, the first phase is the building of the original model used, followed by the splitting of the model and respective co-simulation scenario. The results of the non-distributed co-simulation are compared to the original model results. The major concern in this stage is the RT capability of the models, which is ensured by achieving high enough micro-step sizes, while also maintaining acceptable levels of accuracy.

4.1 Bicycle car modelling

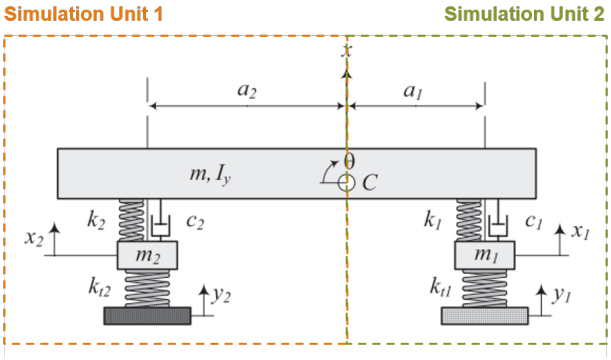
Given that the purpose of this work is the coupling of SUs, the realness and usefulness of the models built was not a concern. However, it is important that the system's dynamic behaviour is in the realm of the future applications because different types of signal impose different challenges. With this in mind, it was decided to use the concept of the bicycle car.

As can be seen in Figure 4.1a, the bicycle car model allows to study the dynamic behaviour of a car in terms of its body pitch angle (θ) and bounce (x) and in terms of the wheels hop (x_1 and x_2). This enables the study of ride comfort in the cabin of the car, allowing for suspension optimization (Jazar, 2008).

Figure 4.1b shows a conceptual model of the bicycle car. With this, it is understood that the body of the car can be modelled as a planar body with both vertical and angular displacement. The horizontal displacement of the body is not considered as there are not solicitations in that direction. Furthermore, each suspension is modelled as a spring and damper and each wheel is modelled as a mass. The tire is represented by a spring, but according to some models it can also include a damper. The suspensions, wheels and tires have only one degree of freedom (DOF), which is the vertical displacement.



(a) Body representation.



(b) Conceptual model.

Figure 4.1: Bicycle car schematics. Extracted from Jazar (2008).

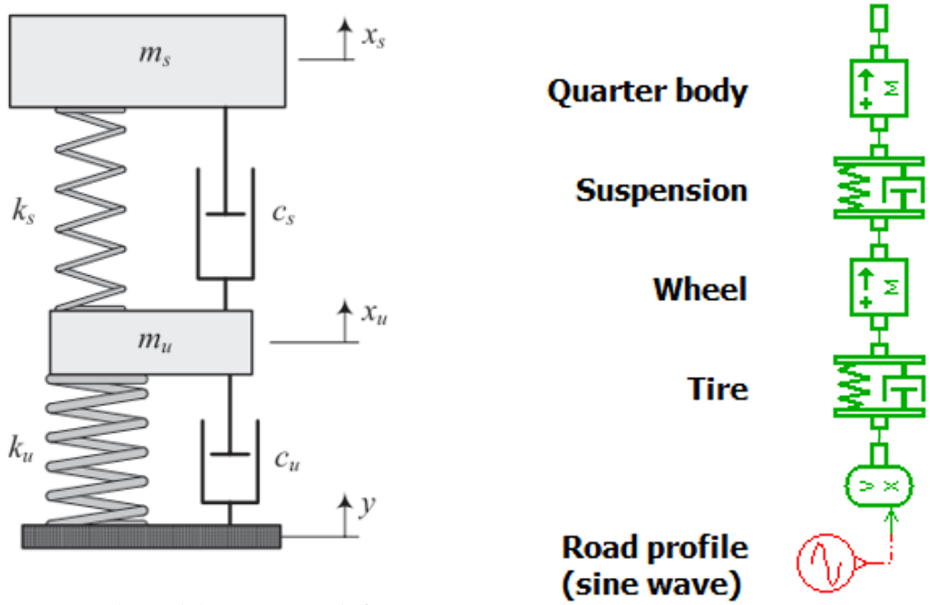
4.1.1 Quarter car

Looking again for Figure 4.1b, the reader can see the desired model splitting. The purpose is splitting the body across its centre of gravity into the rear and front parts. This means that the bicycle car can be modelled as two coupled quarter car models. An example of the quarter car model is given in Figure 4.2a, where the tire is represented by a spring and damper (like previously stated).

Looking now at Figure 4.2b, it is fairly simple to understand the model construction in AMESim. Using the one-directional mechanical library blocks, the body and wheel are modelled as rigid bodies and the suspension and tire are modelled as sets of spring and damper. In terms of the road displacement, i.e. the excitation of the system, a block converting signal to 1D mechanical variables is used. This way, the signal provided corresponds to the temporal evolution of the road profile, and the block is responsible of converting that to the road displacement and velocity, which is computed through derivation.

Starting of with this model, Jazar (2008) gives examples of reasonable parameters for all the parts of the system. The parameters chosen are compiled in Table 4.1.

For the road displacement, a sine wave generator was used (as can be seen in Figure 4.2b).



(a) Conceptual model. Extracted from Jazar (2008).

(b) AMESim model.

Figure 4.2: Quarter car models.

Table 4.1: Quarter car parameters.

Parameter		Value
Body	Mass $[kg]$	271,25
Suspension	Stiffness $[N/m]$	10000
	Damping $[N/(m/s)]$	800
Wheel	Mass $[kg]$	40
Tire	Stiffness $[N/m]$	150000
	Damping $[N/(m/s)]$	10

The sine wave can be described by Equation (4.1).

$$y(t) = 0.1\sin(2\pi t) \tag{4.1}$$

Running the performance analyser on this model, and taking a look at the eigenfrequencies of the system, it is witnessed a high frequency induced by the block linking the road displacement signal to the mechanical part of the system. This happens because the velocity variable of the road excitation is being numerically derived. This means that the bottleneck of the model at this stage is the numerical derivation instead of the mechanical system itself. To solve this issue, a different kind of component was used for the excitation of the system, which can be seen in Figure 4.3.

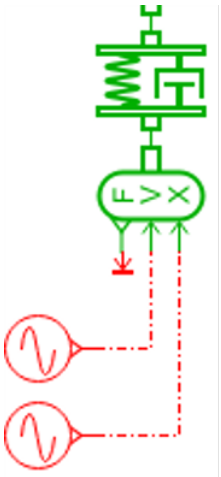


Figure 4.3: Block used for excitation of the mechanical system.

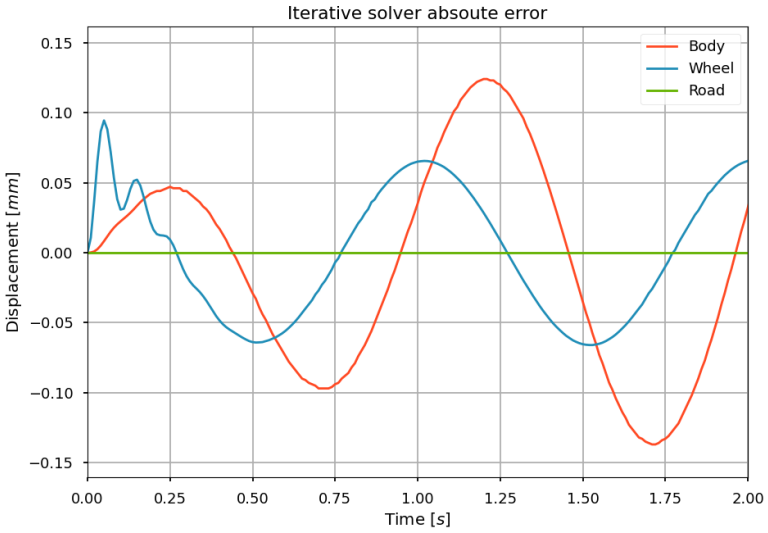
This component allows for explicitly inputting the velocity, which is easily done given that the road displacement function is available. By derivation, the road velocity can be given by Equation (4.2).

$$\dot{y}(t) = 0.2\pi\sin(2\pi t + 90) \tag{4.2}$$

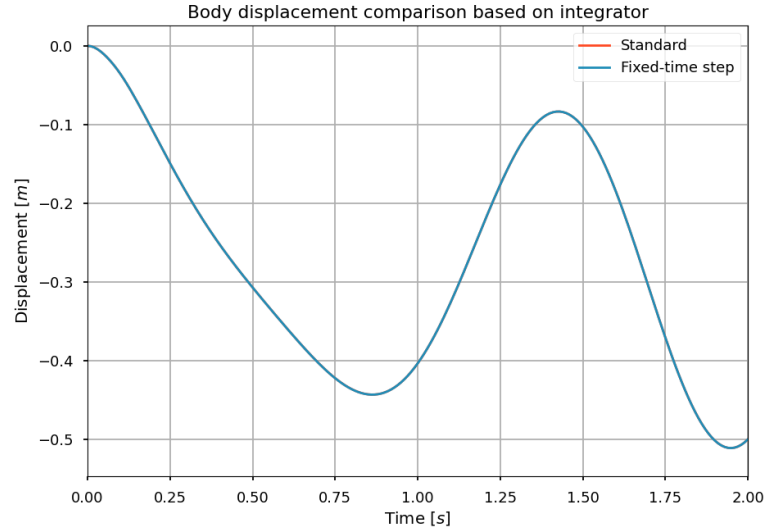
Eliminating the numerical derivation of this model, the corresponding time constant was eliminated, increasing the maximum micro-step size of the system to 47 ms. This means that, using a non-iterative integrator with a fixed integration step size smaller than 47 ms should provide results very similar to the ones obtained with an iterative solver. Figure 4.4a confirms this by showing an insignificant absolute error. In fact, a relative error in the order 0.03% for the body displacement and 0.07% for the wheel displacement was obtained. In Figure 4.4b, it is also visible that the difference between integrators' results is not noticeable.

4.1.2 Single body bicycle car

Having defined the quarter car model, the bicycle car model can be defined using AMESim 2D mechanical library. With Figure 4.5 is clear that the suspension, wheel, tire and road parts of the model are the same as in the quarter car, except for the fact that now there is a rear and



(a) Absolute error for all DOF and excitations.



(b) Body displacement comparison.

Figure 4.4: Quarter car iterative and non-iterative integrators comparison.

front version. In terms of the car body, the block used corresponds to a rigid 2D body that allows movement in 3 DOF, i.e. the vertical and horizontal displacement and the rotation in the axis perpendicular to the plane. Important to notice in this model is the block responsible for the transformation of the 1D mechanical variables into 2D mechanical variables. In that sense, the forces coming from the suspension parts are both transformed into positive vertical forces (F_y), while all the other ports of the block are linked to zero force or zero torque sources.

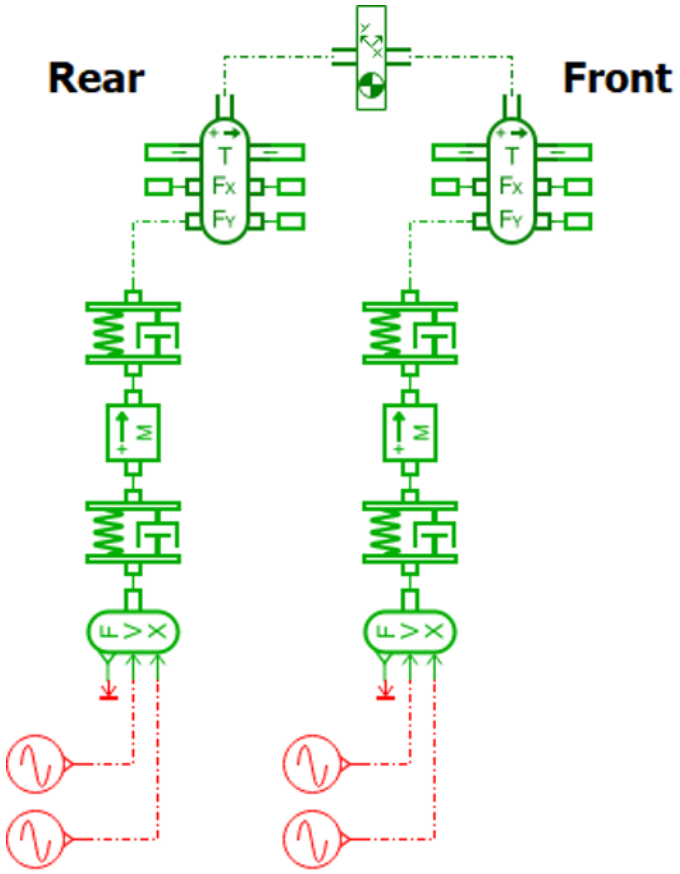


Figure 4.5: Bicycle car model in AMESim.

The parameters for this model are, again, supported by the examples provided by Jazar (2008) and are compiled in Table 4.2.

In this case, the road excitations need also to be adjusted so that they are temporally shifted one to another. This way, the front and rear road excitations, $y_1(t)$ and $y_2(t)$ respectively, are defined by Equation (4.3). It is important to notice that this phase shift between rear and front is completely arbitrary as it would depend of the vehicle velocity, which is not under consideration in this model.

Table 4.2: Bicycle car parameters.

	Parameter	Value
Body	Mass $[kg]$	542.5
	Moment of Inertia $[kgm^2]$	1100
	Rear distance $[m]$	1.47
	Front distance $[m]$	1.4
Suspension	Stiffness $[N/m]$	10000
	Damping $[N/(m/s)]$	800
Wheel	Mass $[kg]$	40
Tire	Stiffness $[N/m]$	150000
	Damping $[N/(m/s)]$	10

$$y_1(t) = 0.1\sin(2\pi t + 45) \tag{4.3}$$

$$\dot{y}_1(t) = 0.2\pi\sin(2\pi t + 135) \tag{4.4}$$

$$y_2(t) = 0.1\sin(2\pi t) \tag{4.5}$$

$$\dot{y}_2(t) = 0.2\pi\sin(2\pi t + 90) \tag{4.6}$$

Again, looking at the performance analyser, the frequencies of the model are only relative to the models dynamic, with no frequency limitation due to numerical processes. With this, a maximum fixed time step of 47 ms is obtained, which is similar to the one obtained for the quarter car. Figure 4.6 illustrates that the error between integrators increases slightly, reaching a relative error of 0.25%, 0.55% and 0.34% for the body, back wheel and front wheel vertical displacements, respectively. Moreover, it is interesting to see the phenomena happening until around 1.5 seconds, where the absolute error is 0. This suggests that the standard integrator in this phase behaved like the fixed-time step integrator, and iteration was deemed unnecessary.

4.2 Non-distributed co-simulation

Having built the original bicycle car model, the following step is actually splitting it onto the distinct SUs defined in Section 4.1.1, with the required interface definition.

4.2.1 Split body bicycle car

As the objective is splitting the body into two separate bodies, the attention goes onto the coupling of the two bodies. In the coupling it is important to keep in mind the problems mentioned in Section 2.1.2, namely the algebraic constraints and the stability problem inherited by tight couplings. Given this, the solution chosen was the inclusion of a rotary junction between the two

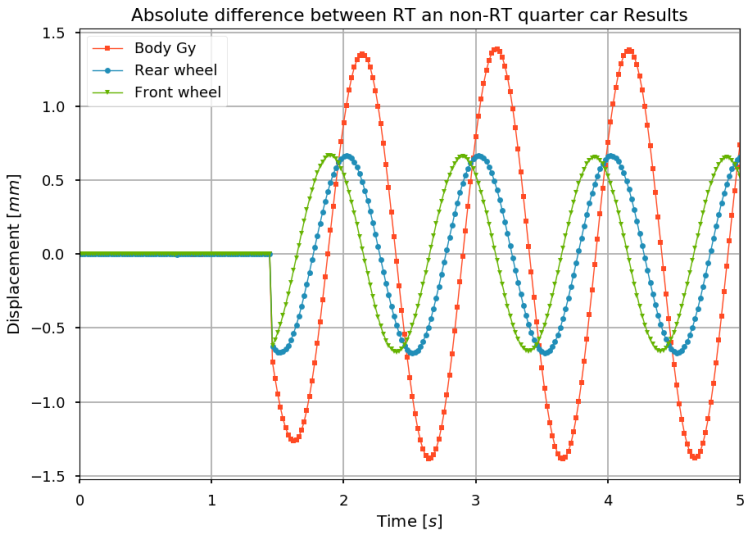


Figure 4.6: Bicycle car absolute error between integrators.

bodies, as can be seen in Figure 4.7. Finally, the chosen coupling point between SUs is in the interface between the rotary joint and the front body.

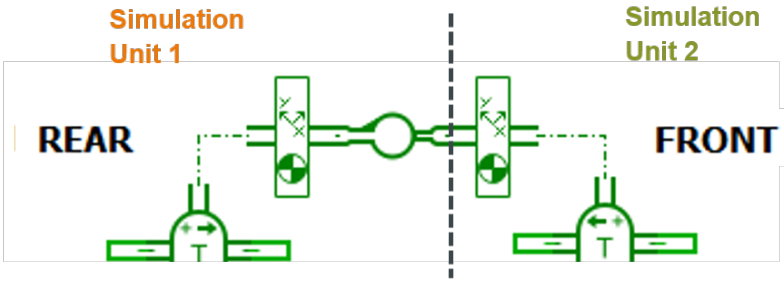


Figure 4.7: Bicycle car body splitting.

This junction constraints the relative motion between the bodies, only allowing relative movement in the rotational DOF. The extent to which this movement is allowed is defined by the rotary stiffness and damping of the junction. Nonetheless, the linear relative displacements are also allowed to some extent to avoid the tight coupling simulation issues that may arise. This way, the linear damping and stiffness, both vertical and horizontal, must also be defined. Notice that, for an ideal coupling, both the stiffness and damping are infinite, as this would impose a rigid connection both in terms of displacement and velocity. However, with increasing value of this parameters, also increases the numerical complexity imposed by the junction, therefore reducing the maximum micro-step size allowed. A compromise solution was achieved with the junction parameters visible in Table 4.3.

The results of body splitting and junction parameters, in comparison to the original model's non-iterative solution, are visible in Figures 4.8 and 4.9. In terms of the bodies vertical displacement (Figure 4.8a), the absolute error of the front body is slightly bigger than the on the rear.

Table 4.3: Rotary junction selected parameters.

Parameter	Value
Rotary stiffness $[N/rad]$	10^6
Rotary damping $[N/(rad/s)]$	10^4
X linear stiffness $[N/m]$	10^7
X linear damping $[N/(m/s)]$	10^5
Y linear stiffness $[N/m]$	10^7
Y linear damping $[N/(m/s)]$	10^5

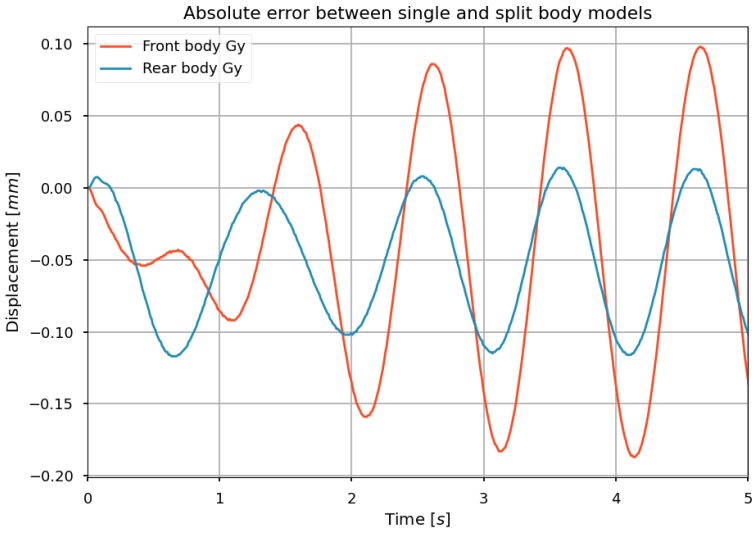
However, in terms of the relative error, this difference is barely noticeable. For the front body it is of 0.03% and for the rear body it is of 0.02%, which represents a good accuracy after splitting. In terms of the body angles (Figure 4.8b), similar conclusions can be taken. In fact, the relative error is 0.07% and 0.15% for the front and rear body angles, respectively. In terms of wheel displacement, it is visible in Figure 4.9b the finding of a new equilibrium position. This justifies an increase in the error, which reaches 10% for the front wheel and 15% for the rear wheel.

4.2.2 Interface definition

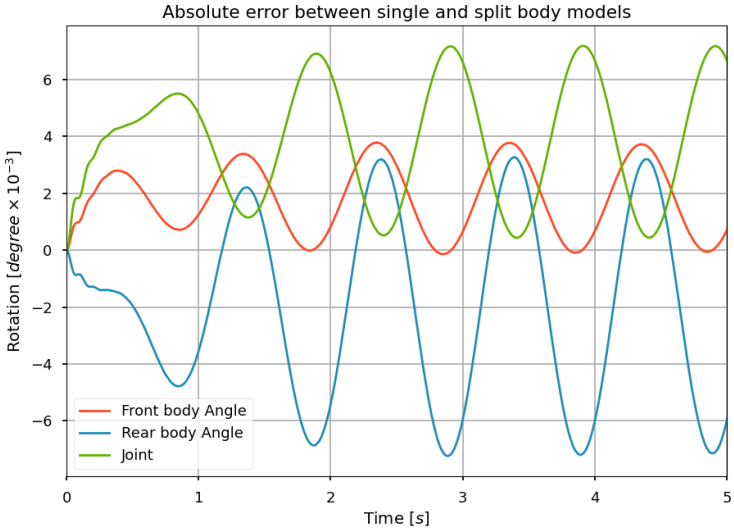
Regarding the coupling interface construction, it is necessary to understand the variables exchanged at the defined coupling point. Looking at Figure 4.10, it is visible that the rear part of the bicycle car is explicitly receiving the displacement, velocity and acceleration in all three planar DOF. On the opposite direction, the rear part is sending to the front part the torque and the vertical and horizontal forces caused by the joint.

For separation of rear and front parts, the placement of sensors to make the output variables available and the placement of input blocks (similar to the one used for the road excitation) is necessary. In the 2D mechanical library there are only two input blocks available: one for input torque and forces and the other to input either displacement, velocity or acceleration. This means that only one of these last three types of variables is inputted explicitly, with the others being either integrated or derived. With this in mind, two alternatives were attempted. First, acceleration was inputted, with displacement and velocity resulting from numerical derivation (Figure 4.11a). The other alternative was the input of displacement variables, with acceleration and velocity resulting from numerical integration (Figure 4.11b). An intermediate solution (input as velocity) was not considered because it would carry the problems of both the studied alternatives. In terms of the force and torque input block no special considerations need to be taken. For outputting variables, the adequate sensors need to be placed depending on the input block selected.

At this stage, a proper co-simulation is still not occurring. Yet the alternatives presented in Figure 4.11 allow for an evaluation of the influences of the different interfaces. First, by looking at the data provided by the performance analyser tool, comes that the alternative with acceleration input has algebraic loops. Given that the numerical integration of position and velocity always carries a numeric error, that gets propagated by the algebraic loops, resulting in an unstable model. As it is not possible removing the algebraic loops or the numeric error, this model is not viable for

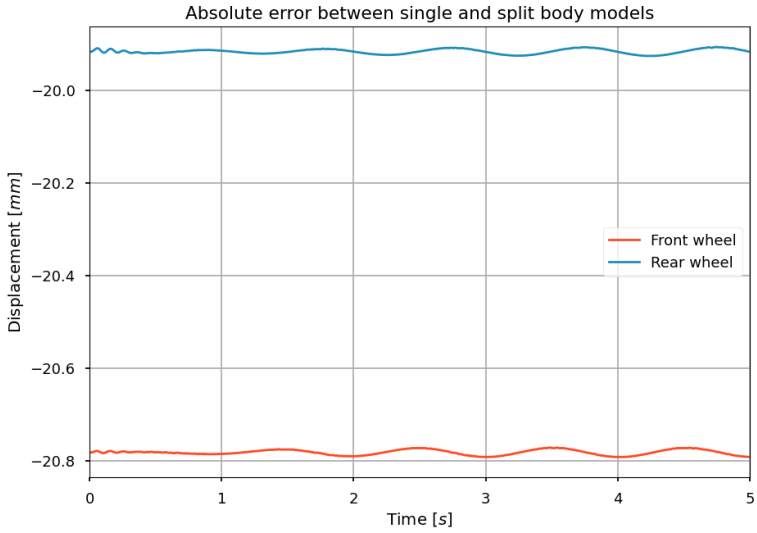


(a) Bodies vertical displacement.

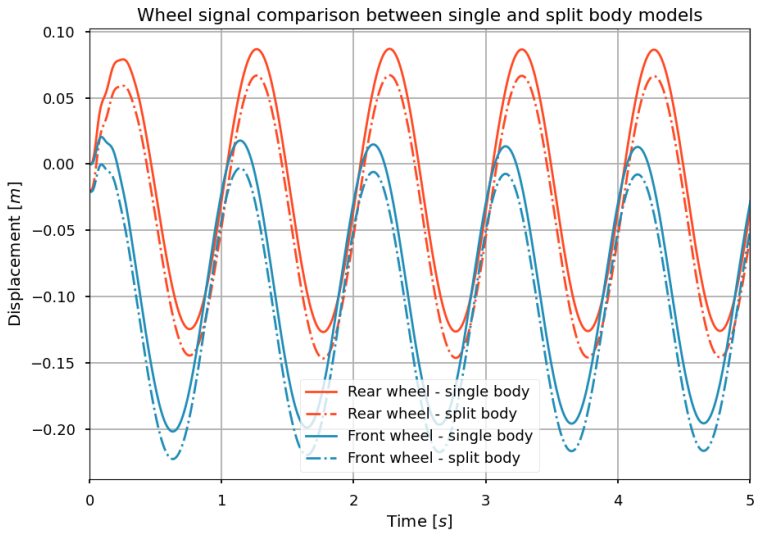


(b) Bodies angle and joint relative angle.

Figure 4.8: Rear and front body absolute error between split and unsplit model.



(a) Absolute error.



(b) Signal.

Figure 4.9: Wheels displacement comparison between split and unsplit model.

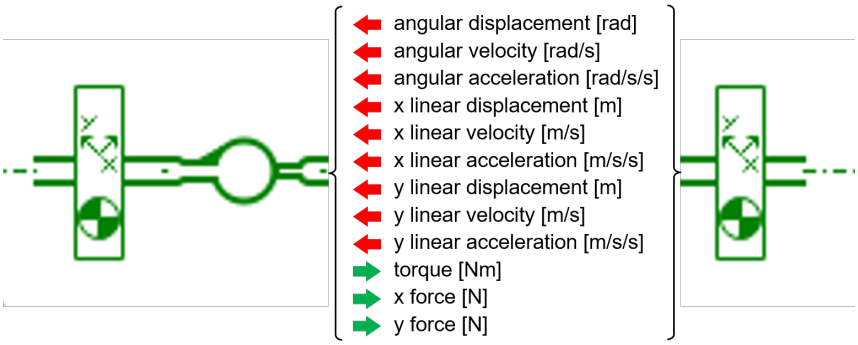
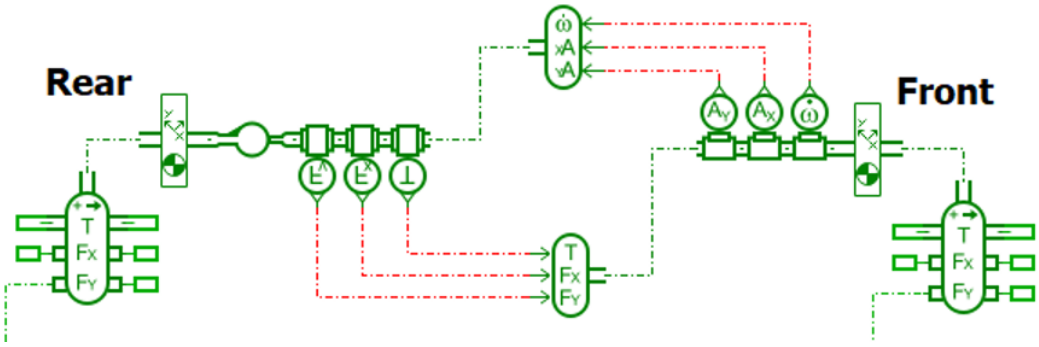
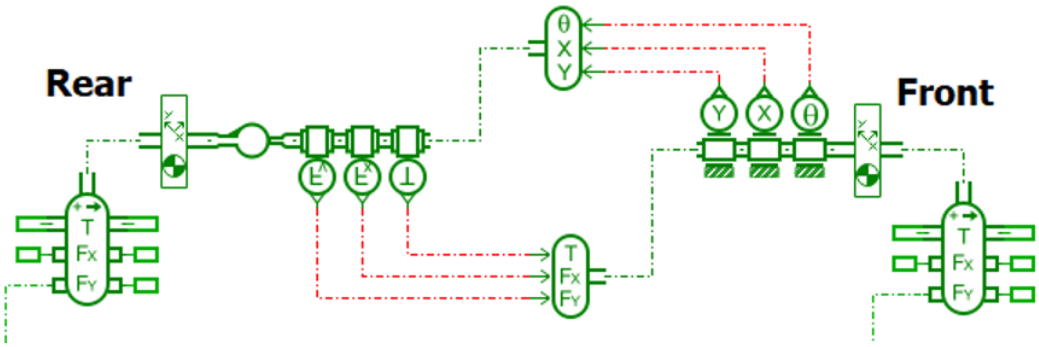


Figure 4.10: Variables exchanged at the co-simulation coupling point.



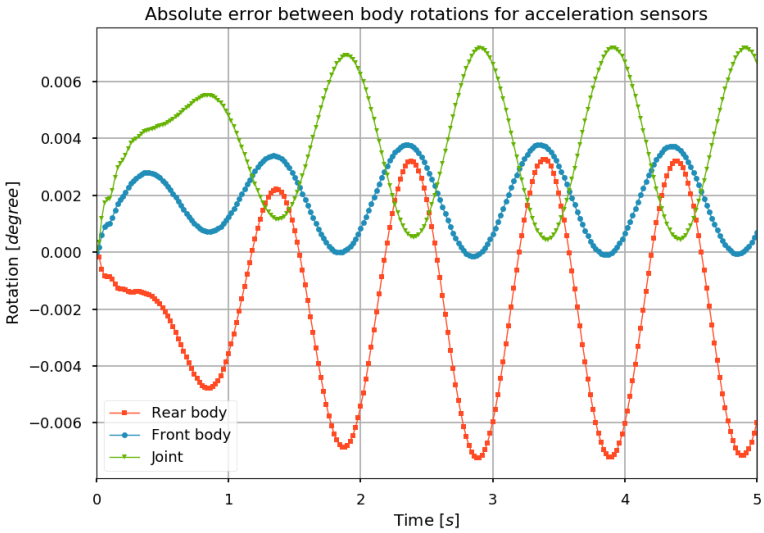
(a) Acceleration input source.



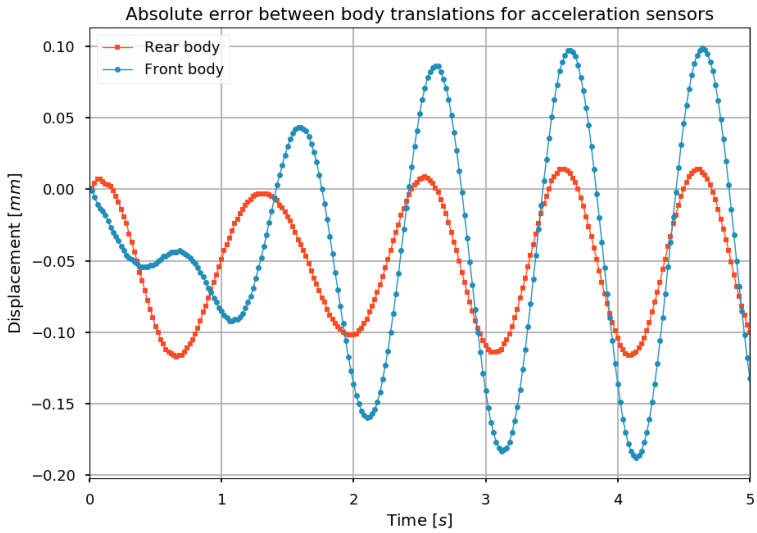
(b) Position input source.

Figure 4.11: Interface variables alternatives.

co-simulation. Nonetheless, the results obtained with it were evaluated. Looking at Figure 4.12 it is visible that a low error was obtained, making this simulation accurate. Notice however that the model does not allow the usage of a non-iterative solver, because of the algebraic loops, jeopardizing a comparison with the position input source solution.



(a) Bodies rotation error



(b) Bodies vertical displacement

Figure 4.12: Absolute error between acceleration input source and single body bicycle car models.

With the position input alternative, two derivation methods are available. One of the options is implicit derivation, which requires the usage of iterative solvers (Harish, 2020), therefore being inadvisable for RT co-simulation scenarios. On the other hand, it can be used first-order lag derivation method. This method is also found in the literature with the name *delayed differential*

equations. For a comprehensive description of this method, the author recommends the reading of Shampine & Thompson (2012); Kumar & Kumari (2020). First-order lag requires, however, the definition of an appropriate derivation time constant. In Table 4.4 and Figure 4.13 are compiled the results of the study concerning this parameter. As can be observed, with an higher time constant, the accuracy of the simulation decreases and, with a lower time constant, the maximum allowed micro-step size decreases. A compromise solution was found with a time constant of 0.002 seconds.

Table 4.4: Influence of the first-order lag time constant on the maximum time step.

Time constant [s]	Maximum Eigenfrequency [Hz]	Maximum micro-step size [ms]
0.0001	1591.55	0.278529
0.001	159.15	2.785294
0.002	96.13	4.383609
0.005	78.48	5.950925
0.01	75.72	6.187774

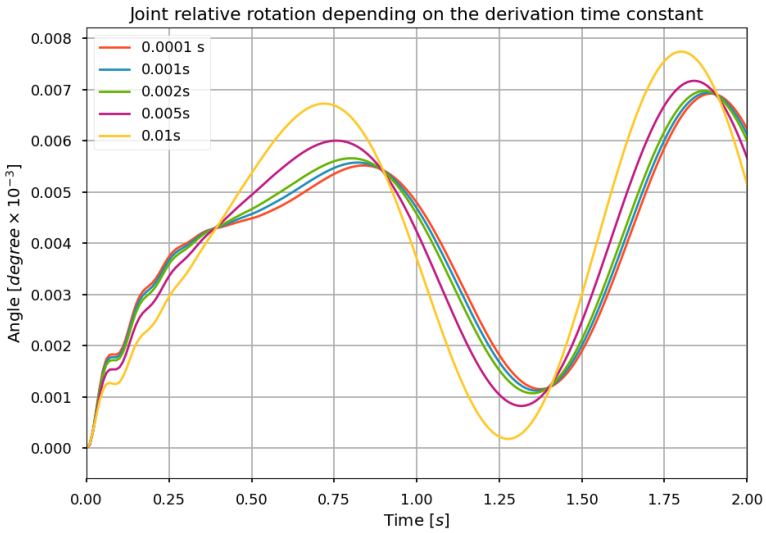
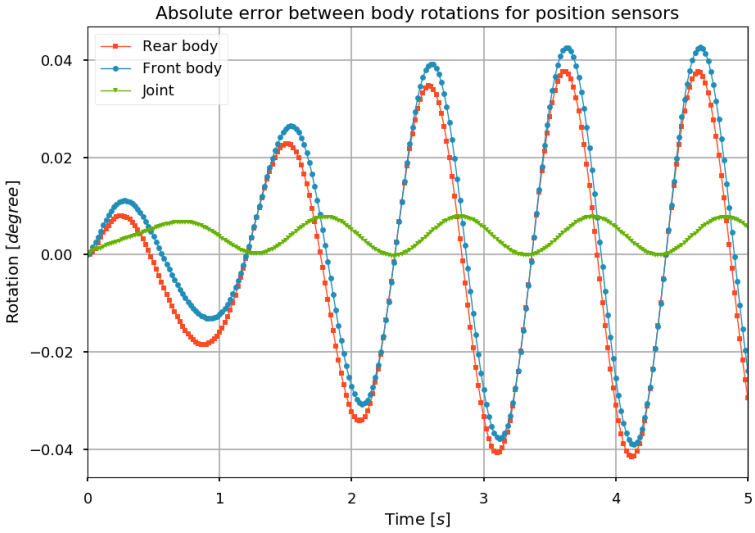


Figure 4.13: Joint relative rotation variation with the first-order lag time constant.

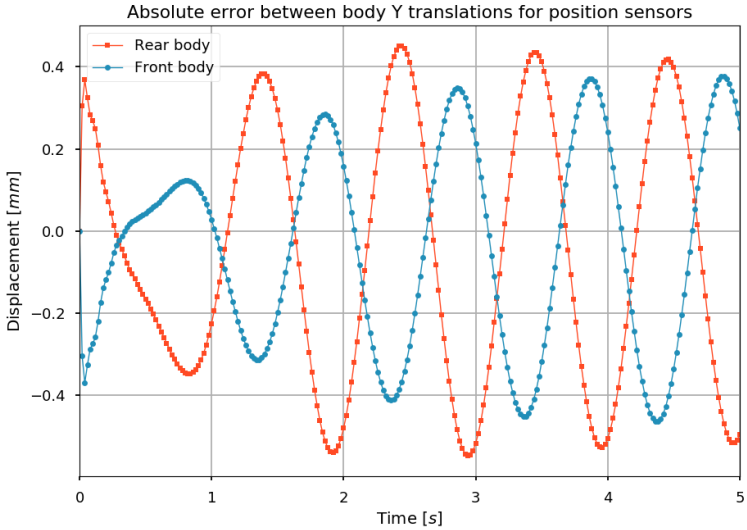
Looking now at the simulation results in Figure 4.14, one can see that this solution is almost ten times less accurate than the one provided by the acceleration input alternative. This is strongly influenced by the iterative integrator using with the latter.

4.2.3 Local AMESim co-simulation

For the last stage of the preparation of this model for co-simulation, shared memory co-simulation blocks available in AMESim were used. This block creates a shared memory location



(a) Bodies rotation error

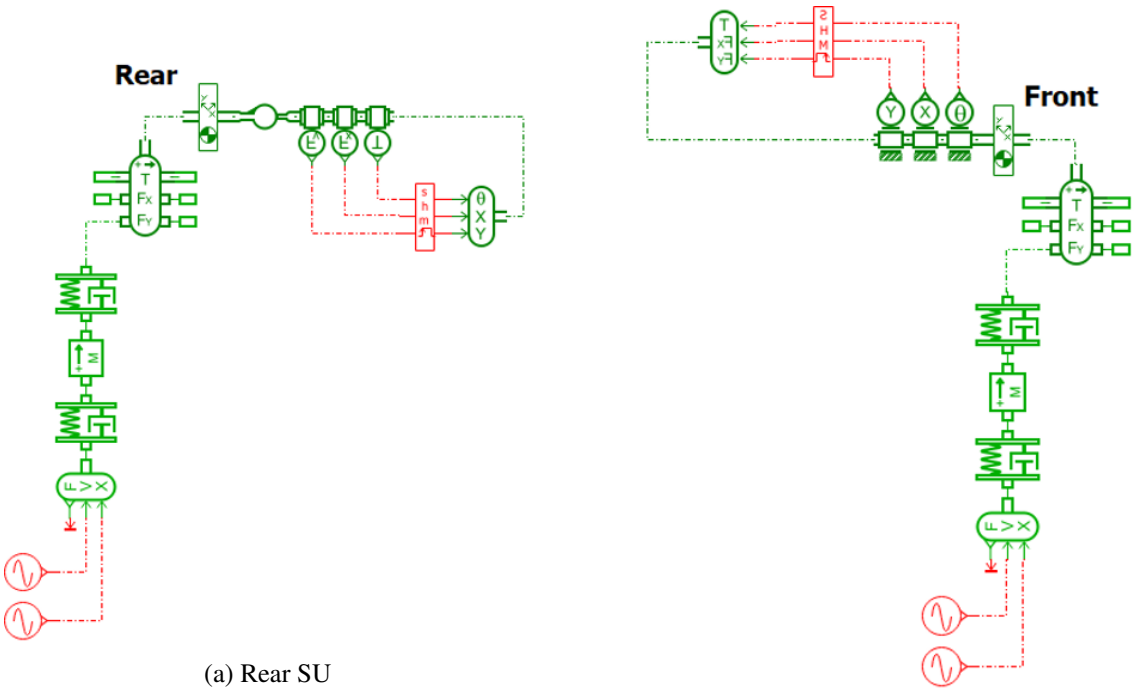


(b) Bodies vertical displacement

Figure 4.14: Absolute error between position input source and single body bicycle car models.

in the disk on which the inputs and outputs of each SU are written and read from. The resulting models can be seen in Figure 4.15. Even tough this model represents the alternative with displacement input, an equivalent model was built for the case of acceleration input.

It is important to notice that at this stage the rear and front part are not from the same model. Therefore, there is the need for the simulations to be synced. This is achieved by assigning each of the co-simulation blocks the role of master or slave. The master works as the orchestrator of the co-simulation, defining both the macro-step size and the progression of time.



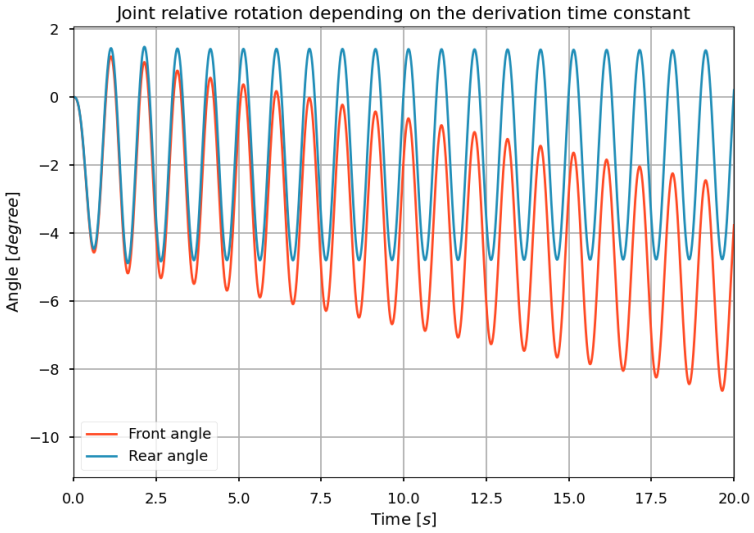
(a) Rear SU

(b) Front SU

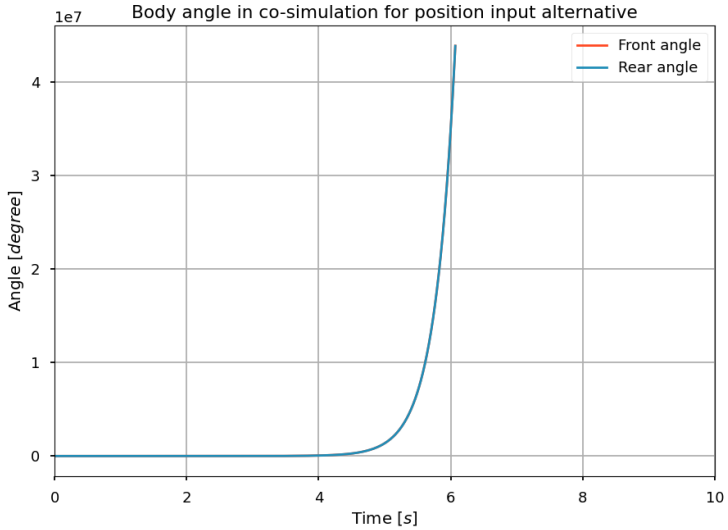
Figure 4.15: AMESim co-simulation models.

Figure 4.16 presents the results for both alternatives. Starting with the acceleration input case, it is visible the stability error caused by numeric error of the integration, with the results slowly drifting apart from the original model results. Although the results differ from the ones obtained in the previous stage, this may be due to the fact that now it was possible to use a non-iterative integrator.

With the position input scenario, the results were unexpected. In fact, a divergent solution is observable, with the absolute value of all displacements growing exponentially. This is perhaps caused by an accumulation of the numeric error of the derivative. However, at this stage, it was witnessed that the co-simulation blocks were not behaving correctly, as what was given as output by one of the SUs did not match the corresponding input. As the internal functioning of the AMESim blocks is out of the author's control, this solutions were abandoned at this stage being reconsidered again in the HIL stage in Section 6.2. Also important to state is that similar results were obtained regardless of the macro and micro-step sizes chosen.



(a) Acceleration input source



(b) Position input source

Figure 4.16: AMESim co-simulation results.

Chapter 5

Software-in-the-loop stage

This chapter focuses on the implementation of the co-simulation network impairments mitigation techniques. To aid this, an AMESim environment for their implementation and evaluation was created, allowing the usage of signal processing utilities of AMESim. This model calls two distinct executable programs. One is responsible for the simulation of the different network impairments while the other is responsible for the mitigation of the impaired signals.

5.1 Experimental setup

The AMESim experimental setup built is represented in Figure 5.1. There it can be seen that the first executable block (*EXE* block in the figure) is responsible for calling the impairment simulation program. The impairment program is responsible for sampling the signal, according to the macro-step size, and simulating the network impairments. It takes as inputs the clock signal, which indicates the current simulation time, and the wave signal to be transmitted. Following the impairment executable block comes the mitigation block which, given a sampled and impaired signal, is responsible for the signal reconstruction. The reconstructed signal is then compared to the original signal for an evaluation of its approximation error.

In Figure 5.1, it is also possible to see that, in terms of impairments, constant and variable delay and packet loss were implemented and that, in terms of mitigation methods, ZOH, FOH, SOH and LCF were implemented. The techniques were evaluated with both sinusoidal and square waves.

It is relevant to understand the variables being exchanged between the two executable blocks, as well as the parameters they take. Figure 5.2 illustrates the impairment block and the mitigation block interfaces. The focus resides on the coupling of the two blocks, where is explicit the exchange of 4 variables. The first, *Sending Time*, refers to the time at which the impairment block reads the signal being transmitted. The *Receiving Time* refers to the time at which the transmission to the mitigation block is occurring, i.e. in practice this signal represents the current time of the simulation. The *Exchange Flag* indicates if the signal is to be considered by the mitigation block or not. This exists because AMESim requires the output of the blocks to be continuous, not

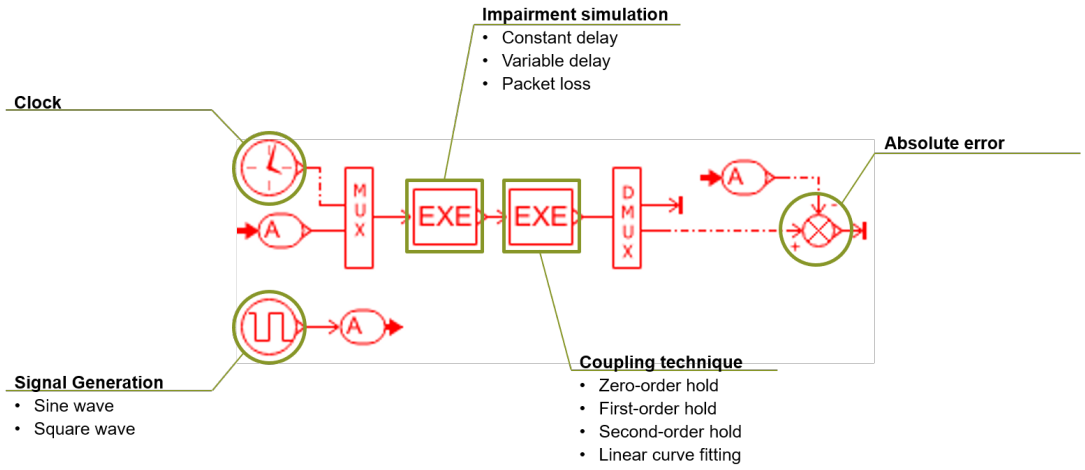


Figure 5.1: AMESim model for implementation of executables

considering the sampling and impairing caused by the communication. Given this, the impairment block is always producing output with the exchange flag defining if its output at a given time shall be considered or not. Lastly, the *Signal* corresponds to the value of the input signal at sending time, i.e. when it was read by the impairment block.



Figure 5.2: Executable blocks interfaces.

In terms of parameters, the impairment block takes the macro and micro-step sizes in seconds. The macro-step size defines the sampling periodicity of the signal while the micro-step size should be equal to the one of the non-iterative integrator. In terms of impairment parameters, the user can define the end-to-end delay, following a normal distribution, in terms of its average value and its standard deviation. It is also possible to define the packet loss ratio in a range from 0 to 1. For the mitigation block, the choice of the technique is defined in terms of the executable chosen. This means that each coupling method has its own executable program. It is also possible to define a number of points for the LCF method, with this parameter being ignored by the remaining methods.

5.2 Netwok impairments simulation

Focusing now on the network impairment simulation program, the reader can see in Algorithm 1 the pseudocode for the program. This indicates that, after the connection with AMESim and the retrieval of the parameters given to the block, the program enters an infinite loop. This loop includes the reading of inputs (line 4), the processing of inputs and the sending of outputs

(line 25). The termination of this program is determined by the parent process (AMESim), which can cancel the execution of the program at any time.

In terms of input processing, the first thing to notice is the if statement that defines if the current input is to be sampled or not (line 5). If so, the impairments generation occurs. For the delay, this happens through the generation of a random delay following a normal distribution based in the average and standard deviation of the delay. For the packet losing, this happens by the generation of a random number following a uniform distribution from 0 to 1 and subsequent comparison with the packet loss ratio. For a lost packet, the exchange flag is set to 2. Else, the exchange flag is set to 1 and the remaining values are adequately appended to the respective arrays.

After the storage (or not) of received values, the *receiveTime* array is searched and compared to the *currentTime*. If it exists any k for which *receiveTime* corresponds to the *currentTime*, the output is set to the array values of index k and is, posteriorly, sent to AMESim. Else, the values are kept unchanged, meaning that the exchange flag will be 0.

Algorithm 1 Pseudocode for the network impairment simulation program

```
    Connect to AMESim and retrieve parameters values (macroStep, microStep, avgDelay,
    stdDelay, packetLossRatio)
    k = 0
    while true do
      Get current time and value
 5:   if time is a multiple of macroStep then
        Set delay to a random number following normal distribution based in avgDelay and
        stdDelay
        Set losePacket to a random number following uniform distribution from 0 to 1
        if lose < packetLossRatio then
          exchangeFlag[k] = 2
10:       receiveTime[k] = time + delay
        else
          sendTime[k] = time
          receiveTime[k] = time + delay
          value[k] = value
15:       exchangeFlag[k] = 1
        end if
        Increment k by 1
      end if
      Initialize all outputs to 0
20:   for all k in receiveTime[k] do
        if receiveTime[k] = time then
          Set outputs to the array values with index k
        end if
      end for
25:   Send output values
    end while
```

For evaluating the implementation of the impairment block, a macro-step size of 0.1 s and a micro-step size of 0.001 s were used. The results can be seen in Figure 5.3, representing the

original signal and the sampled and impaired signal. The three sub-figures represent three different impairment scenarios.

Figure 5.3a shows the case of a constant delay, i.e. the delay standard deviation is 0, with no packet loss. All the sampled signals were transmitted and are equally spaced, attesting the constancy of the delay. Furthermore, it is visible that each peak has the same value of the transmitted signal, but 0.05 s earlier, which is coherent with the average delay defined.

In Figure 5.3b, the case for a variable delay (delay average and standard deviation of 0.05 s and 0.1 s, respectively) with no packet loss is presented. The unequal spacing of the peaks shows the variance of the delay, confirming its correct implementation. However, it is also visible that not all peaks are represented, showing that there is an unwanted packet loss. The author believes the cause is the way the delay was modelled. As normal distributions allow negative values, there may have been signals affected with a negative delay, which ultimately were not transmitted. The asymmetric nature of a real delay variance is not well-represented by a normal distribution. An improvement of this implementation should be done using other probability density functions.

Figure 5.3c represents the case of constant delay with a packet loss ratio of 0.5. It is visible that the transmitted values match the ones of Figure 5.3a, ensuring that the packet loss ratio has no influence in the delay. Furthermore, the lack of some of the peaks indicates that packet loss is occurring. The actual packet loss ratio of this scenario was 0.45, for a total of 20 sampled packets, which is fairly close to the desired packet loss ratio.

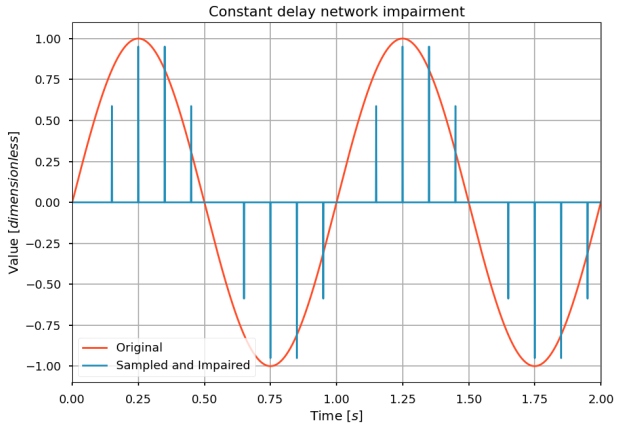
5.3 Mitigation techniques

5.3.1 Implementation

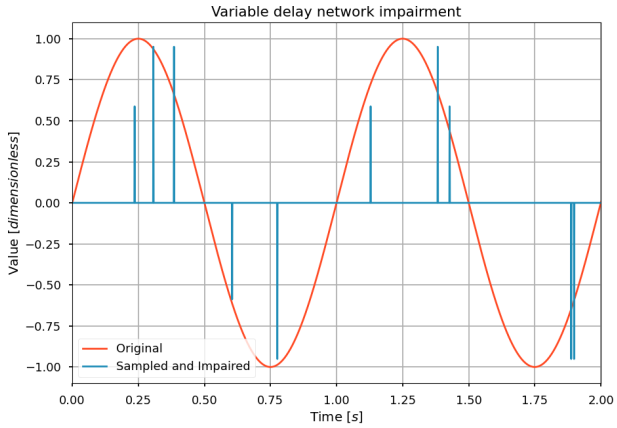
As already stated, the coupling techniques implemented were ZOH, FOH, SOH and LCF. The choice of this techniques resides on the simplicity of their implementation while still being widely used in co-simulation. Furthermore, to avoid the storage of unnecessary past data, a circular buffer was used instead of standard linear buffer.

The circular buffer, also called ring buffer, is a fixed-size buffer that wraps around, meaning that it is connected end-to-end. This buffer was implemented with a data structure, i.e. the buffer is declared as a fixed-size linear array and the data structure is the element characterizing the ring buffer state at that moment. The main principle about the ring buffer is that there are two pointers, one for the read position of the buffer and the other for the write position. The way these pointers evolve needs to address the end-to-end connection of the ring buffer. For this, instead of manually incrementing, two functions were created for updating the write position and the read position. Furthermore, there is a function for initializing the buffer state and two flags to state if the buffer is empty, or not, and full, or not.

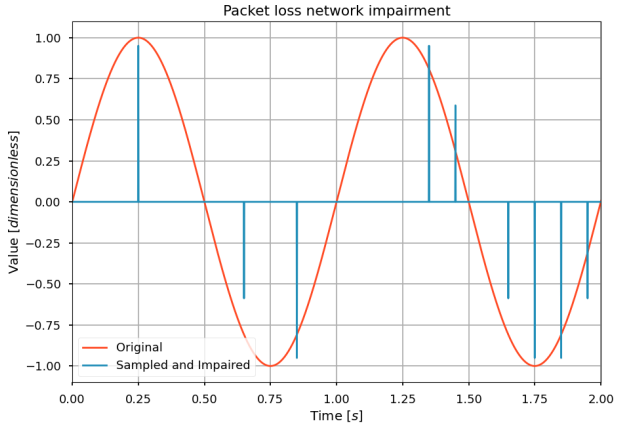
Looking at Figure 5.4, it can be seen an example of the evolution of the buffer state, for a better understanding of its working. It should be noticed that the blue and grey circle represent the write and read position, respectively, and that green and red slots are, respectively, occupied and



(a) Constant delay with avgDelay= 0.05s, stdDelay= 0s and packetLossRatio= 0



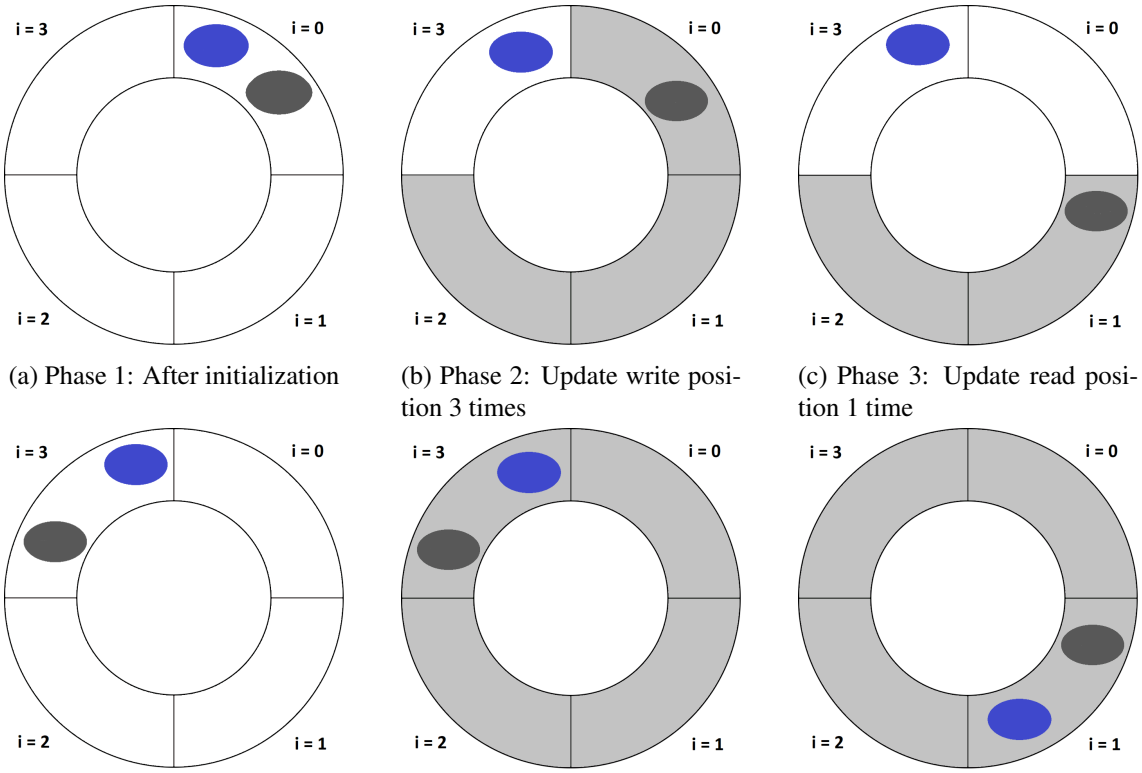
(b) Constant delay with avgDelay= 0.05s, stdDelay= 0.1s and packetLossRatio= 0



(c) Constant delay with avgDelay= 0.05s, stdDelay= 0s and packetLossRatio= 0.5

Figure 5.3: Impairment implementation validation for macro-step size of 0.1 s and micro-step size of 0.001 s.

empty cells. Another thing to bear in mind with the ring buffer is the evolution from Figure 5.4e to Figure 5.4f, where it is visible that this ring buffer, when full, overwrites older data. This is not true for all circular buffers.



(a) Phase 1: After initialization

(b) Phase 2: Update write position 3 times

(c) Phase 3: Update read position 1 time

(d) Phase 4: Update read position 2 times

(e) Phase 5: Update write position 4 times

(f) Phase 6: Update write position 2 times

Figure 5.4: Buffer state evolution example

The necessity for the implementation of a ring buffer, instead of available open-source libraries, lies with the usage of past data. This implementation includes an uncommon extra function, allowing for the retrieval of a value N indexes before the write position, with N being an arbitrary number compatible with the buffer state. This happens without the removal of the value from the buffer, unlike what happens with the function for updating the read position.

Having explained the ring buffer implementation, it is now possible to describe the general algorithm for the mitigation executable block (Algorithm 2). Like with the network impairment simulation program, the first step is the connection with AMESim, followed by the initialization of the ring buffer structure. After that, the infinite loop responsible for inputting and outputting data begins. The first step is evaluating the value of the exchange flag. If it is 1, i.e. the values are to be considered, the values are stored to the respective ring buffers. After this, the output is calculated with the selected mitigation function and is sent to AMESim. As stated in Section 5.1, there is one executable for each mitigation method, with the differences being in the mitigation function called and the ring buffer size.

Algorithm 2 Generic pseudocode for mitigation block program.

```
Connect to AMESim and retrieve parameters values (numberPoints)
Ring buffer initialization
while true do
    Get current sendTime, receiveTime, value and exchangeFlag
    if exchangeFlag = 1 then
        Write sendTime and value to respective ring buffers
        Update ring buffers write position
    end if
    Calculate the output value with a given mitigation function
    Send the output value
end while
```

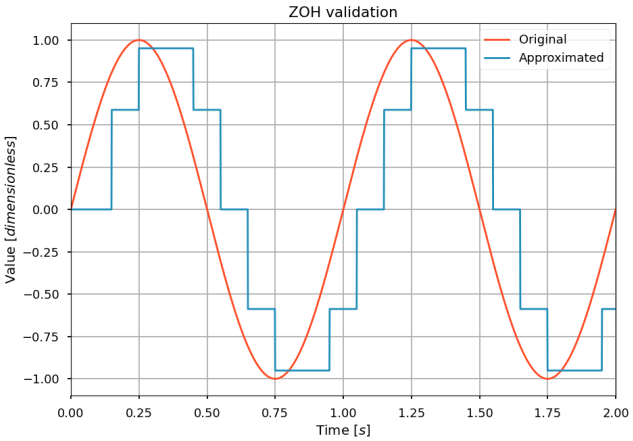
The mitigation functions are similar in process between themselves. They receive as arguments the ring buffer structure, the buffers for *sendTime* and *value*, and the time for which they are extrapolating, i.e. *receiveTime*. The first step is getting the required past values. If this happens successfully, the output value is calculated with the mathematical expressions described in Chapter 3. The difference therefore exists only on the number of past values retrieved and the expressions for the output calculation.

Figure 5.5 shows the validation of the mitigation techniques implemented. For visibility, a macro-step size of 0.1 seconds was used and the impairment applied was a constant delay with no packet loss. For ZOH and FOH the validation is made by the resemblance with Figures 3.2 and 3.3. For SOH, the validation can be made by the shape of the extrapolation curve and, for LCF, by the time it takes to update the slopes of the extrapolation (notice that five sample points were used).

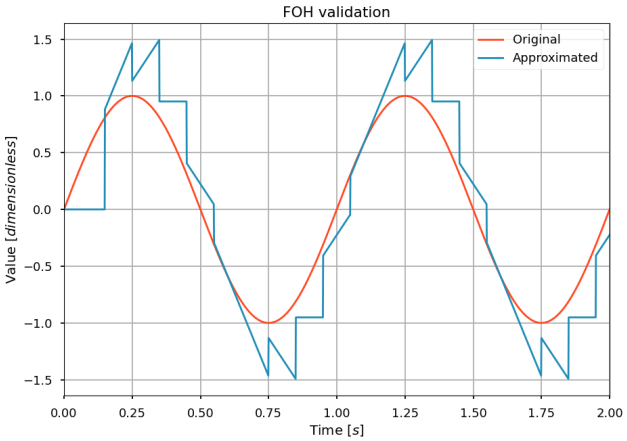
5.3.2 Performance

Given that a reasonable network impairment simulation program was created, the sensitivity of the mitigation algorithms to each of the impairments can be analysed. For this set of scenarios, a macro-step size of 0.01 s and a micro-step size of 0.001 s were used, which are values closer to the real applications. In Figures 5.6 to 5.12, the input signal was a sine wave with a frequency of 1 Hz and an amplitude of 1. The changes between these scenarios are on the impairment level.

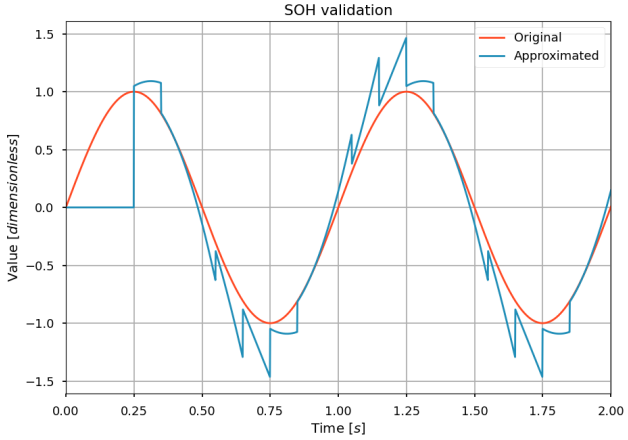
For Figures 5.6 and 5.7, the impairment applied was a constant delay with no packet loss. The first scenario has an average delay of 0.05 seconds and the second has an average delay of 0.5 s. It is visible that, for the scenario in Figure 5.6, all the coupling methods are more or less capable of generating an approximated signal. The highest accuracy is achieved with SOH, having a maximum relative error of 3 %. The lowest accuracy is achieved with ZOH, with a relative error of around 35 %. It is visible from Figure 5.6a that this method offers no delay compensation as the approximated signal maintains the imposed delay. For FOH and LCF, there is a reasonable approximation but also it is noticeable an overshoot. Depending on the applications, this might be a worse problem than the delay. Furthermore, it is important to notice the complete inability of



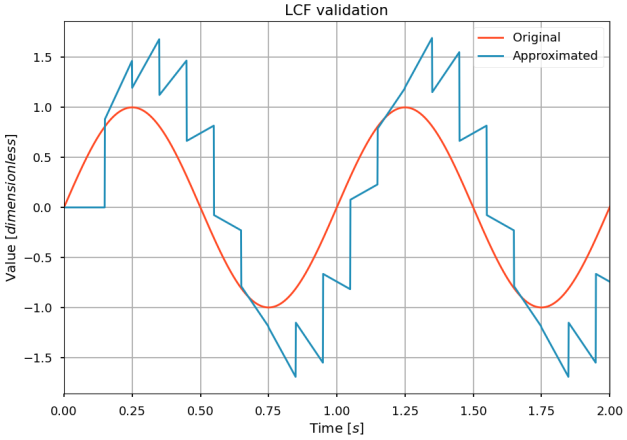
(a) ZOH



(b) FOH



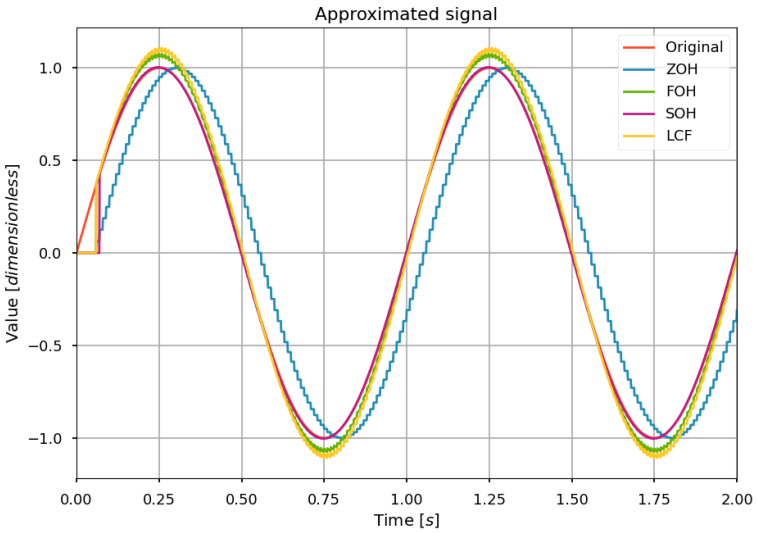
(c) SOH



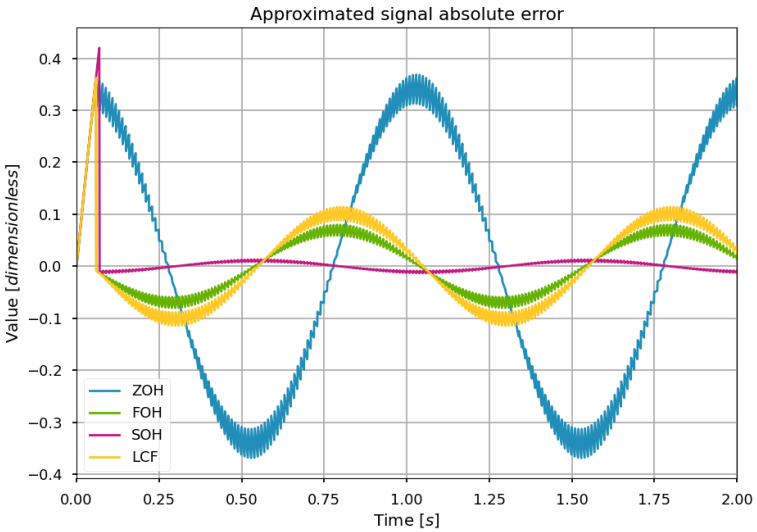
(d) LCF

Figure 5.5: Mitigation implementation validation for macro-step size of 0.1 s, micro-step size of 0.001s and a constant delay (avgDelay = 0.05 s) without packet loss

the methods in the first 0.05 seconds, which is understandable given that no previous information is available. This reinforces the attention needed for the co-simulation initialization phase. In Figure 5.7, with a delay of half the period of the signal, it is visible the inability of these methods to cope with delays of this magnitude. As the methods tested have no sense of periodicity, transmitted signals with too much delay lose their relevance. For this scenario, methods based on waveform properties could be more accurate.



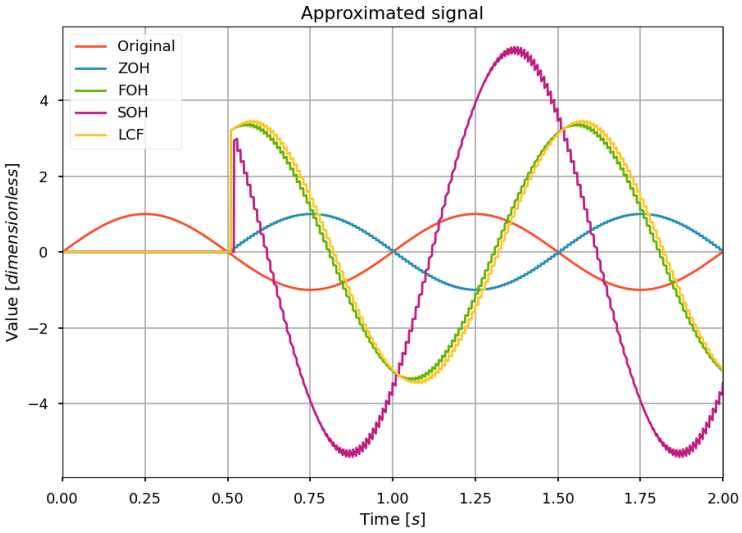
(a) Original and approximated signals



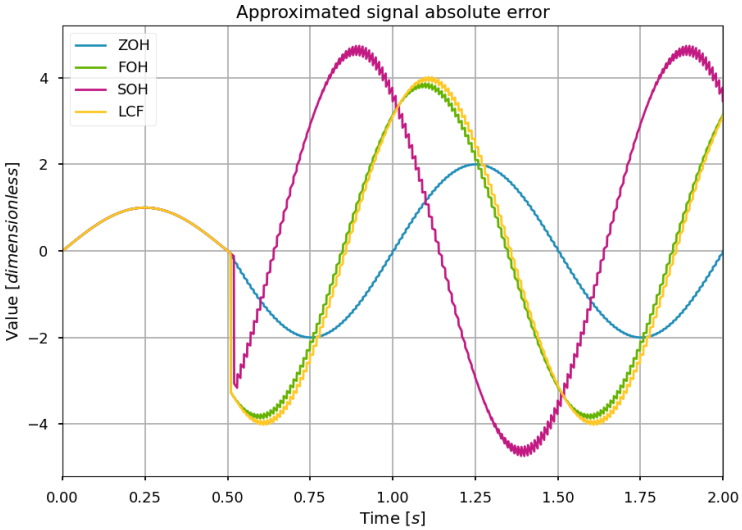
(b) Absolute error of approximation

Figure 5.6: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.

For Figures 5.8 and 5.9, a variable delay, without packet loss, was applied to evaluate the influence of the delay variation. Remember that, as discussed in Section 5.2, variable delay may induce



(a) Original and approximated signals



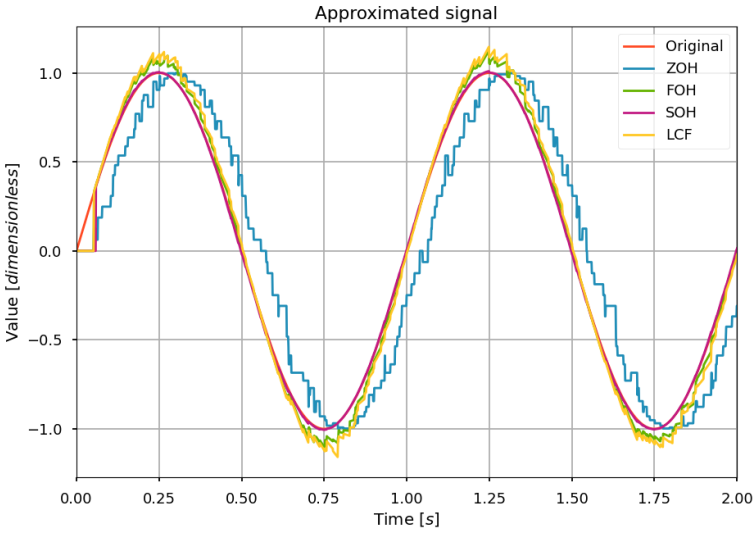
(b) Absolute error of approximation

Figure 5.7: Mitigation for impairment of avgDelay = 0.5s, stdDelay = 0s and packetLossRatio = 0.

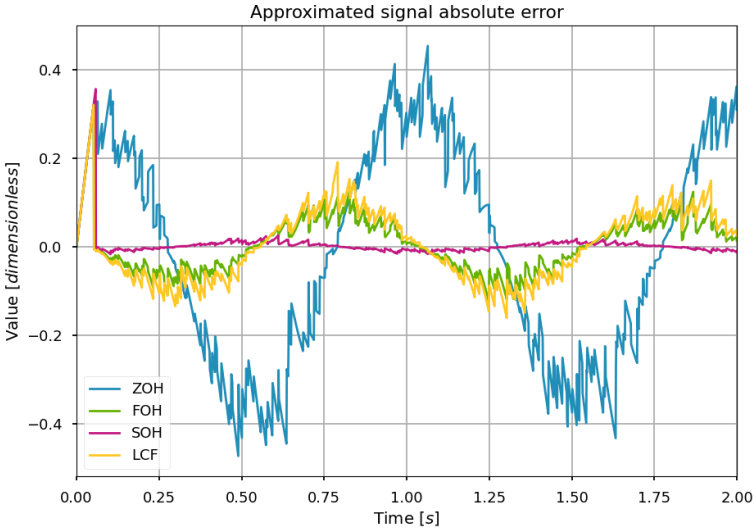
undesired packet loss, jeopardizing the results. The average delay applied was 0.05 seconds and the standard deviation for Figures 5.8 and 5.9 was 0.01 seconds and 0.1 seconds, respectively. It is visible in both scenarios that the variable delay introduces a noise in the signal approximation. This means that, besides the methods applied not being able to cope with noisy signals (see Section 3.1), delay variation behaves as communication-induced noise. Nonetheless, for Figure 5.8 the delay variation is small enough to produce good approximations. In this case, SOH continues to stand out and ZOH produces the worst approximation, with a relative error of over than 40 %. However, in Figure 5.9, the delay variation is too big, resulting in periods where either the delay is uncontrollable or there is packet loss. This is visible in the case of SOH, where the approximation is good except for small and limited periods of time. The other three methods, however, fail completely to produce a properly approximated signal. Because of the problems with the modelling of the delay variation, this results may not be trustworthy, due to possible packet loss. Furthermore, looking at ZOH approximated signal, it is observable that signal alterations occur in opposing direction of the original signal evolution, indicating the existence of out o. Although this is expected with variable delay, the way the ring buffer is used does not consider this impairment. In the mitigation algorithm, the packets order is considered based on their time of arrival. This could lead to, because of variable delay, more recent data being dropped and substituted with older data. A possible solution to this issue could be the evaluation of the *sendTime* variable prior to the buffers updating, thus only keeping the more recent data.

In Figures 5.10 and 5.11, the impairment evaluated was packet loss. For that, it was applied a constant delay with a packet loss ratio of 0.05 and 0.5, respectively. The average delay was 0.05 seconds in both scenarios. Looking at Figure 5.10b, it is visible the effect of the packet loss on the general approximation. The approximated signal drifts from the original signal when it happens, but rapidly converges when new data arrives. For small packet loss ratios, this does not appear to be significant. But, looking at Figure 5.11, it is visible that the problem with packet loss appears when multiple packets are lost in succession. Spending too much time without updating is, ultimately, what makes the accuracy of these signal-based methods drop. However, it is noticeable that SOH is the method that withstands this problem better, given that it has an higher order assessment of the tendency of the signal.

After assessing the influence each of the impairments would have on the mitigation technique, it is also important to evaluate its behaviour in a similar to real-life scenario. For that a ping routine was performed between Leuven (Belgium) and Ferrol (Spain), with the results presented in Table 5.1. However, the ping routine uses Internet Message Control Protocol (IMCP), which is a control protocol for the Internet Protocol. This means that it belongs to the network layer and not to the transport layer of the OSI model. The results are therefore not representative of the impairments of UDP, TCP or RTP. Further details, on the reason of not having results for the desired protocols are presented in Section 6.1. The results obtained allow however to confirm the asymmetry of the delay variation, as can be observed by the comparison of maximum and minimum delay against the average delay.

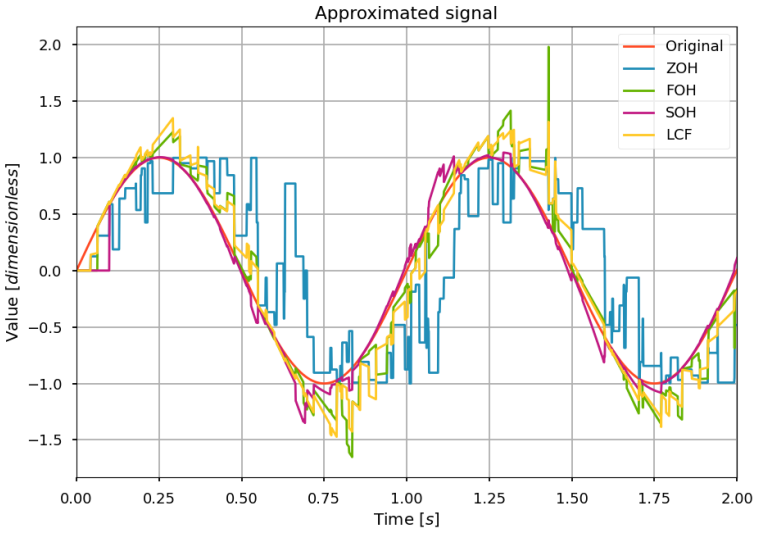


(a) Original and approximated signals

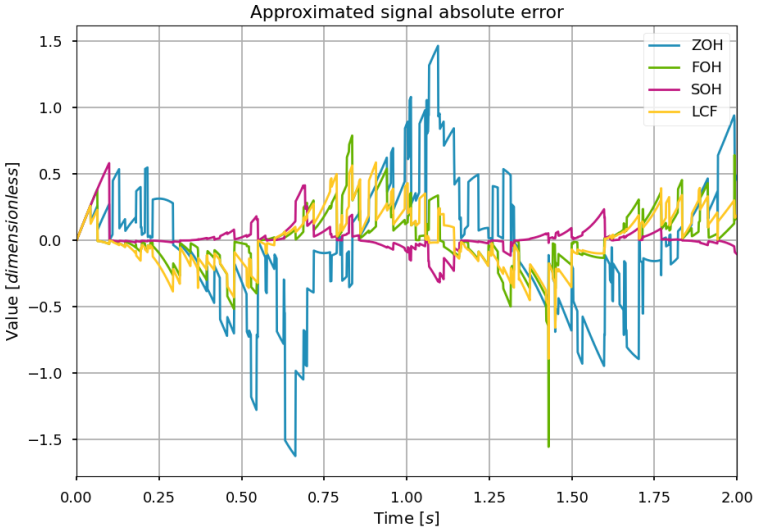


(b) Absolute error of approximation

Figure 5.8: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0.01s and packetLossRatio = 0.

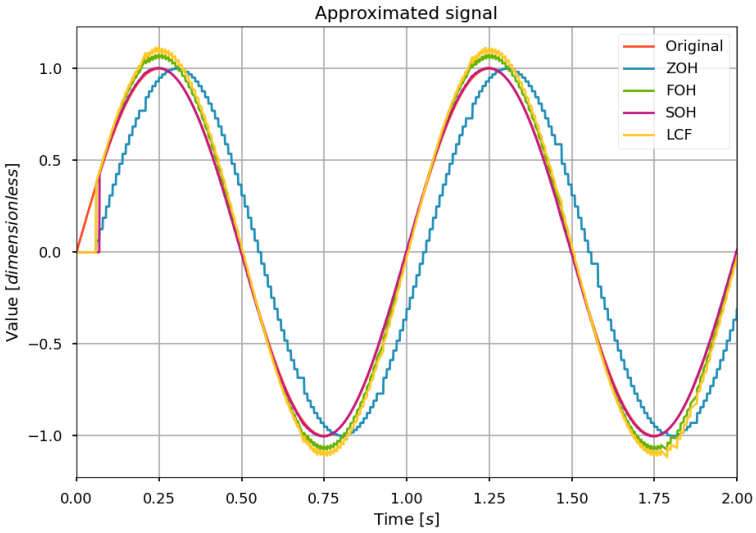


(a) Original and approximated signals

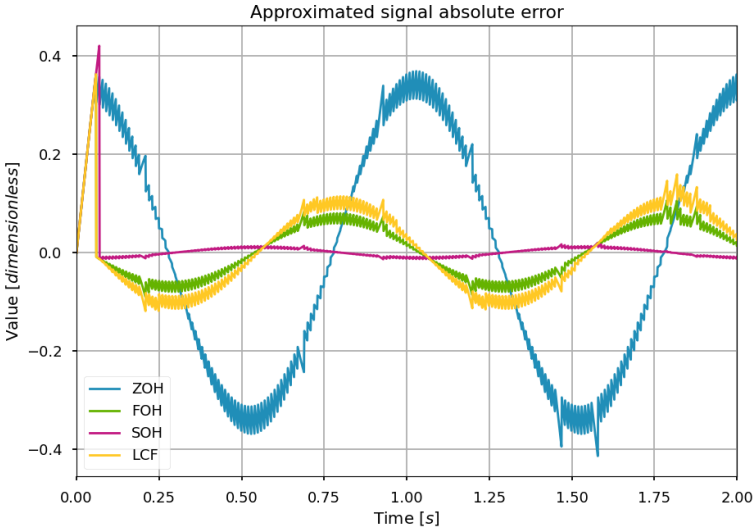


(b) Absolute error of approximation

Figure 5.9: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0.1s and packetLossRatio = 0.

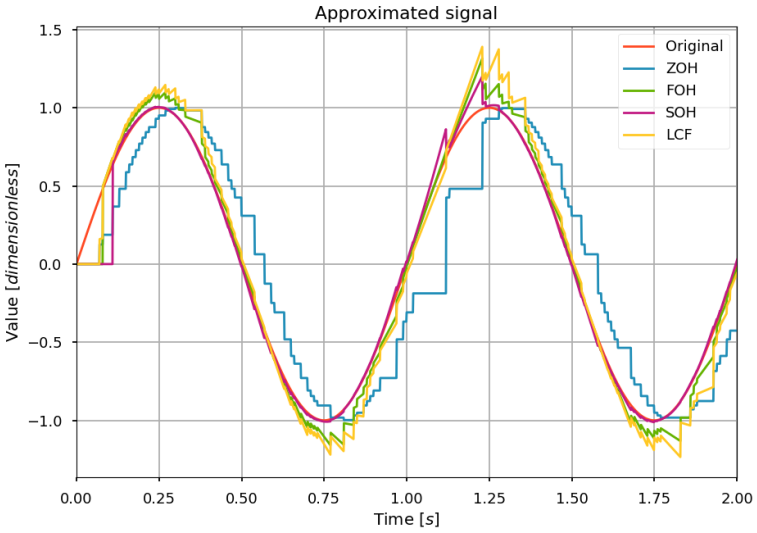


(a) Original and approximated signals

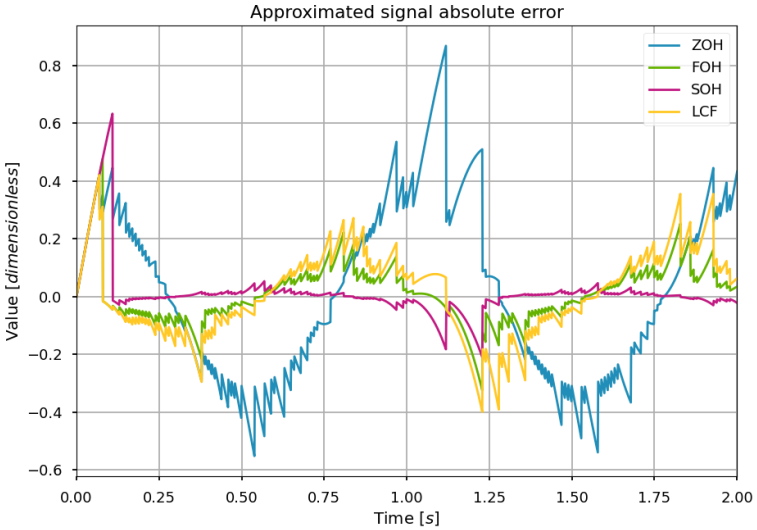


(b) Absolute error of approximation

Figure 5.10: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.05.



(a) Original and approximated signals



(b) Absolute error of approximation

Figure 5.11: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.5.

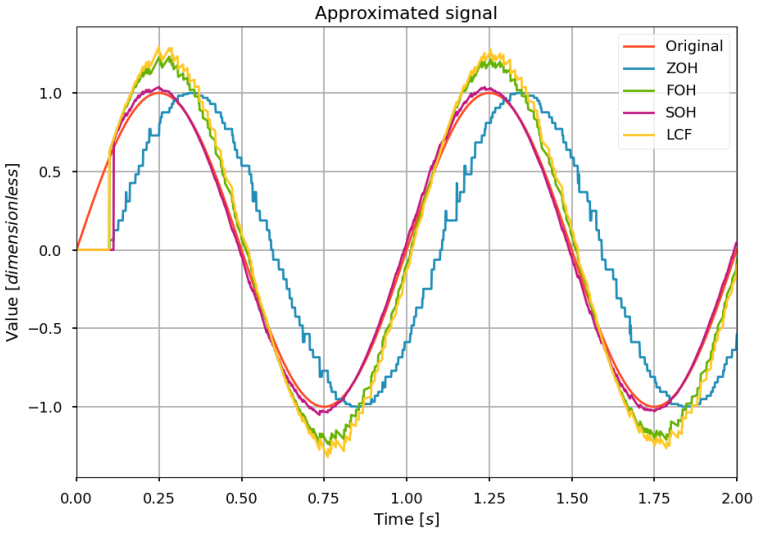
Table 5.1: Measured impairments between Leuven and Ferrol, via IMCP.

Impairment	Value
Packets transmitted	10000
Payload [bytes]	512
Minimum delay [s]	0.090
Maximum delay [s]	0.237
Average delay [s]	0.093
Delay standard deviation [s]	0.007
Packet loss ratio	0.0018

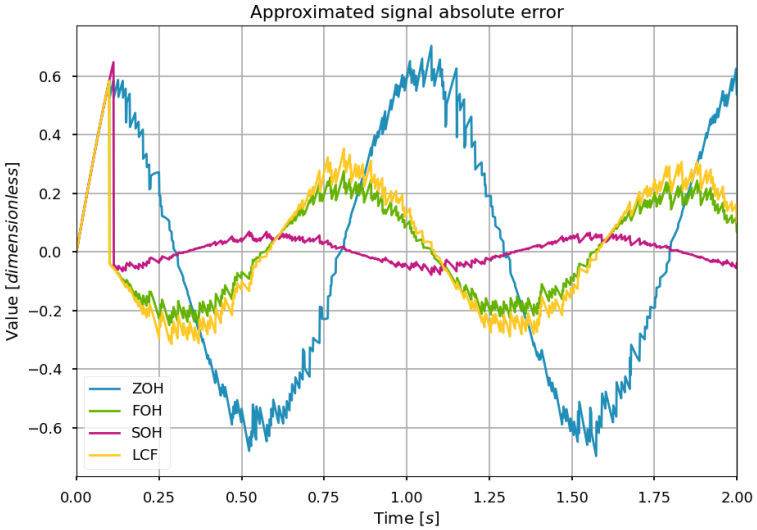
Based on the results obtained, it was applied an average delay of 0.093 seconds, a delay standard deviation of 0.007 seconds and a packet loss ratio 0.05. This increased value of the packet loss ratio was used, because the measured one was too low. The results are visible in Figure 5.12. There it can be seen that, like in the remaining scenarios, SOH is the best method for coupling of sinusoidal signals. Furthermore, the overshooting problems of LCF and FOH and of non-existent delay compensation in ZOH are still observable. In this scenario, SOH presented a maximum relative error of around 7 %.

The SOH results are much better than the remaining signals due to the nature of the signal. It is necessary, however, to assess the techniques capability with different types of signals. For this reason, a square wave was used, with the results presented in Figure 5.13. Looking at Figure 5.13b, which represents a detail of one of the square waves transition, it is visible that not a single one of the methods can compensate the delay, which makes sense since they do not acknowledge the periodicity of the signal. Again, waveform based methods could fit better this scenario. Furthermore, it is now visible that methods requiring more sample points take longer to re-approximate to the original signal. For this reason, ZOH is the method that achieves a better accuracy. Also, the methods that evaluate the signal tendency fail to preview the constancy of the signal after the transition. This means that FOH, SOH and LCF create massive overshoots.

It becomes clear now the importance of adjusting the methods to the signals being exchanged, at least with the implemented methods.

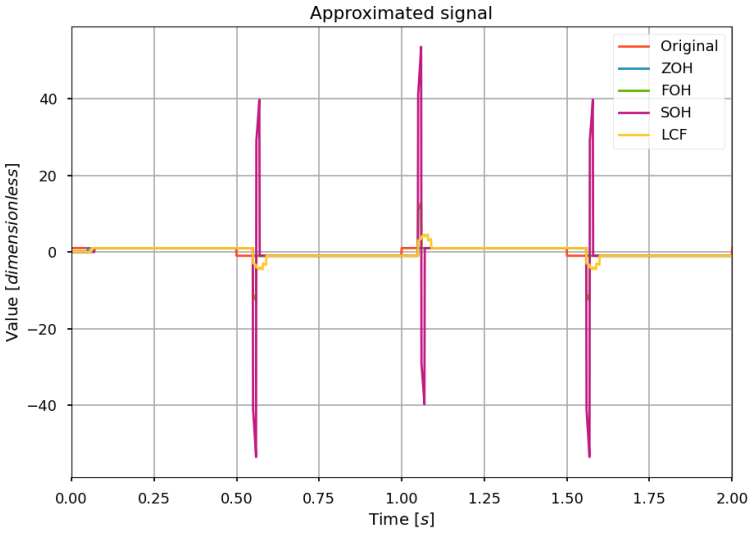


(a) Original and approximated signals

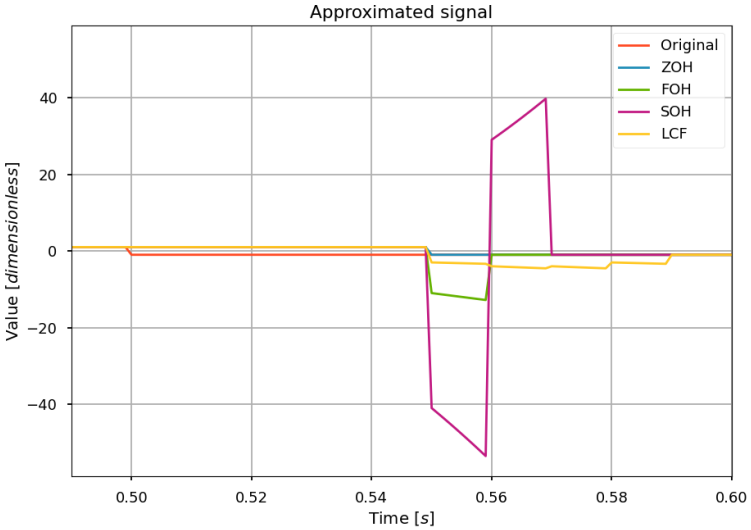


(b) Absolute error of approximation

Figure 5.12: Mitigation for impairment of avgDelay = 0.093 s, stdDelay = 0.007 s and packetLossRatio = 0.05.



(a) Original and approximated signals



(b) Detail on the square wave transition

Figure 5.13: Mitigation for impairment of avgDelay = 0.05s, stdDelay = 0s and packetLossRatio = 0.5 of a square wave.

Chapter 6

Hardware-in-the-loop stage

This chapter presents the integration of physical parts, i.e. besides the computer where AMESim and the software were running. In this case, this includes the use of real-time simulation dedicated computers as well as the inclusion of actual network infrastructures, instead of simulating the impairments. On a first stage, and before actually moving to co-simulation, a communication procedure is built and a network quality assessment program was developed. After this, it is possible to use the real-time platforms and establish the communication between themselves, via a direct Ethernet cable. This allows to achieve a proper distributed co-simulation, after the transformation of the built coupling functions, communication procedure and models into standard-compliant programs.

6.1 Network quality evaluation

Section 2.2 explained the necessity of co-simulation solutions to rely on internet-based protocols. But the choice of the transport protocol remains to be made. RTP can be discarded right away, given that it requires an extra implementation effort due to not being so commonly implemented. Furthermore, the performance in comparison to UDP is unknown to the author, therefore being difficult to judge the true advantages it could have. UDP is then obviously chosen over TCP given that it has lower delays, allowing for more up-to-date information, which is advantageous for RT applications. Having chosen the communication protocol, it is necessary to understand the actual impairments between the co-simulation locations. First, this allows to understand if the delay is small enough to make the co-simulation viable, and also it allows to have the measurements of the impairments to induce in a network emulator.

For this purpose, it was developed a measurement procedure with the communication architecture presented in Figure 6.1. First, it is visible the existence of two layers: the test control layer and the communication (or measurement) layer. The communication layer is where the actual UDP impairments measurement occurs. But, because of UDP unreliability, a test control layer responsible for the starting and ending of the measurement routine was created. This layer communicates over TCP, exactly for reliability purposes. Second, it becomes clear the existence of a

client-server architecture. This means that one of the computers, the server, is in a passive state, i.e. awaiting for communication, while the other, the client, is in an active state. In the test control layer, this means that the client is responsible for indicating the server when to start and stop the measurement routine. In the measurement layer, this means that the server is only retransmitting received messages to the sender.

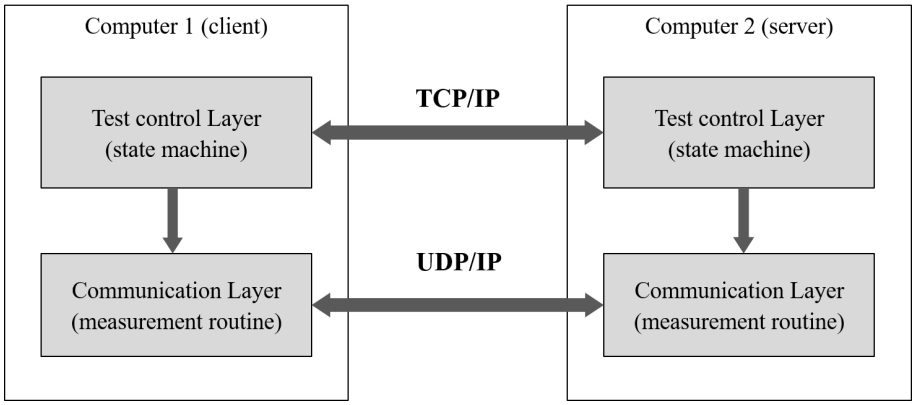


Figure 6.1: Communication architecture

Before moving into a more detailed explanation of each of the machines functioning, their interaction needs to be explained. Assuming that, theoretically, the server is constantly running and listening for TCP communication, the procedure can be described by the following steps:

1. Create client TCP port and establish a connection with the server TCP port;
2. Send START_UDP message over TCP from client to server;
3. Create server UDP port;
4. Send START_UDP_ACK message over TCP from server to client;
5. Create client UDP port;
6. Perform measurement routine;
7. Send STOP_UDP message over TCP from client to server;
8. Close server UDP port;
9. Send STOP_UDP_ACK message over TCP from client to server;
10. Terminate client.

6.1.1 Server

Both the client and server programs were developed and can be explained in terms of a state machine. The server state machine is visible in Figure 6.2a. After the server program is started, the machine goes immediately into the closed state.

In the *closed state*, the machine attempts to open the TCP server port, i.e. open it, bind it to a given address and start listening in that port. If successful the machine transitions to the listen state, else a new attempt is performed.

While in the *listen state*, the server is listening on the TCP port and accepts a client if it attempts to establish connection. Upon a successful connection, the state machine moves into the established state.

In the *established state*, the server reads the incoming communication over TCP from the client. If the message received is of START_UDP type, the machine goes into preparing state.

The *preparing state* includes the creation of a new execution thread for the handling of the UDP communication. This thread is responsible for creating the UDP port before entering an infinite loop where, all received UDP messages are retransmitted to the sender. Upon a successful thread creation, the server sends a message to the client with START_UDP _ACK and moves into the test state.

While in the *test state*, besides the UDP messages retransmission happening on the separate thread, the only function of the server is receiving TCP messages and checking for a STOP_UDP message. If so, the state machine transitions into the stopping state.

In the *stopping state*, the server terminates the thread responsible for the UDP communication and, after that, sends a STOP_UDP_ACK message. Finally, it transitions into the listen state.

6.1.2 Client

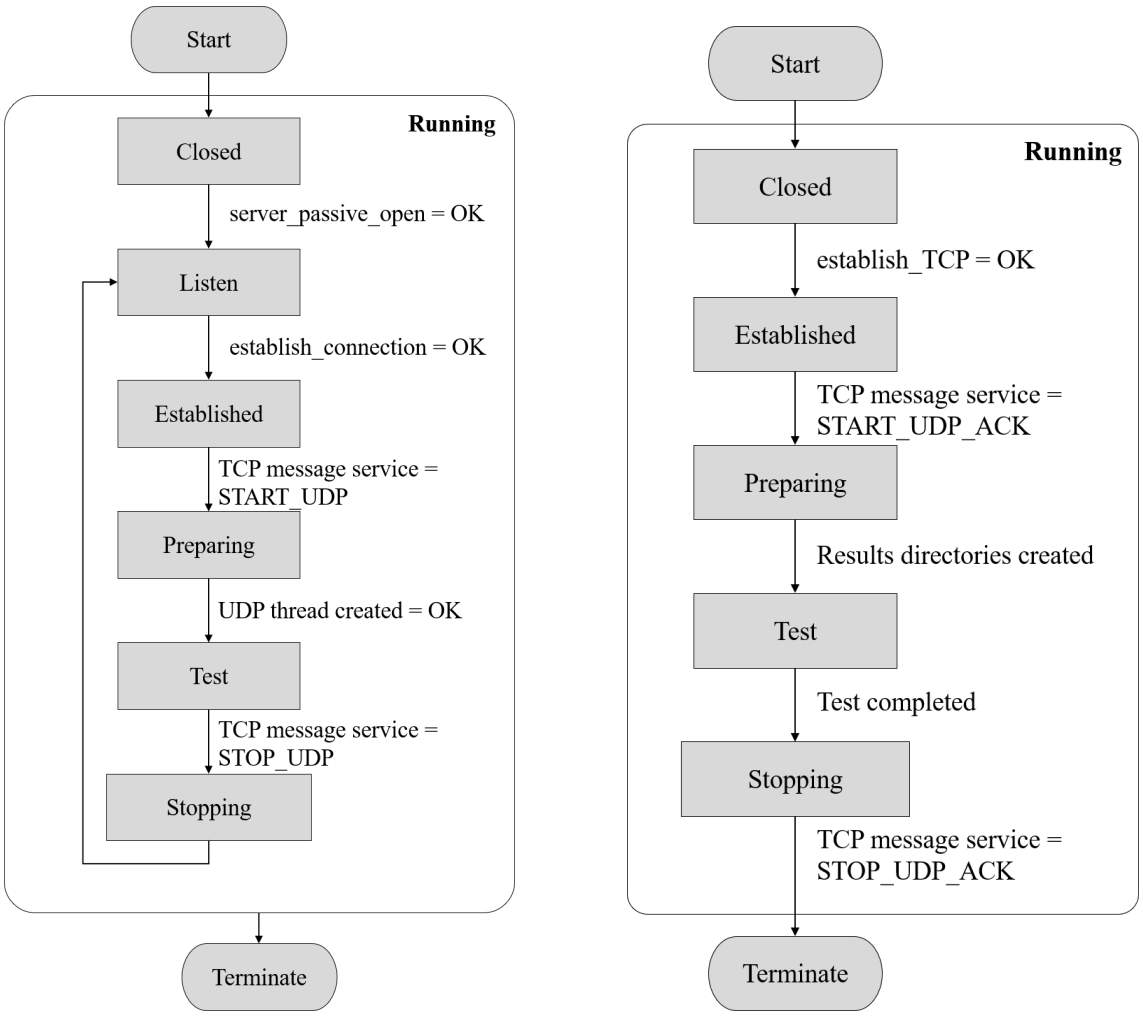
The client state machine follows the state diagram presented in Figure 6.2b. It is visible that the state machine terminates normally, unlike the server that goes back to listen state after the measurement routine. The client start in the *closed state*, which is responsible for the establishment of the TCP connection with the TCP server. This means that it creates the TCP port and attempts a connection with the server, with the success depending on the server acceptance. Upon success the machine transitions to the established state.

In the *established state*, the state machine sends a START_UDP message over TCP and awaits a START_UDP_ACK message from the server. If this happens, the machine goes into the preparing state, else it repeats the routine. Furthermore, the received message contains the address for the UDP port of the server.

On the *preparing state* occurs the creation of all the necessary directories and folders for storing of the results, as well as the creation of the client UDP port.

After this, comes the *test state*, where the measurement routine occurs. This includes the sending and receiving of UDP messages. The delay is calculated by the difference between the time of sending and time of receipt. If a given time has elapsed before the receiving of the reply, the message is considered to be lost. In this case, the time chosen was 5 seconds, but the only criteria is that it is bigger than the maximum expectable delay. The results are then written to the respective file. This routine is repeated for all the iterations and payload lengths chosen.After all measurements are executed the system moves into the stopping state.

The *stopping state* includes, first, the sending of a STOP_UDP message over TCP to the server and the awaiting of the STOP_UDP_ACK. If it succeeds, the machine closes all open ports and files before terminating the program.



(a) Server state machine

(b) Client state machine

Figure 6.2: Communication state machines

This two programs would serve to measure the impairments of the desired network for a proper simulation or emulation. However, the machines that are going to be used are still not in place and VPN has not been established, not enabling the communication between the desired locations. Furthermore, it was not found a server where the program designed could be run. For this two reasons, the programs written were only run locally and the impairments estimation relied on the ping routine from Belgium to Spain.

6.2 Real time platform

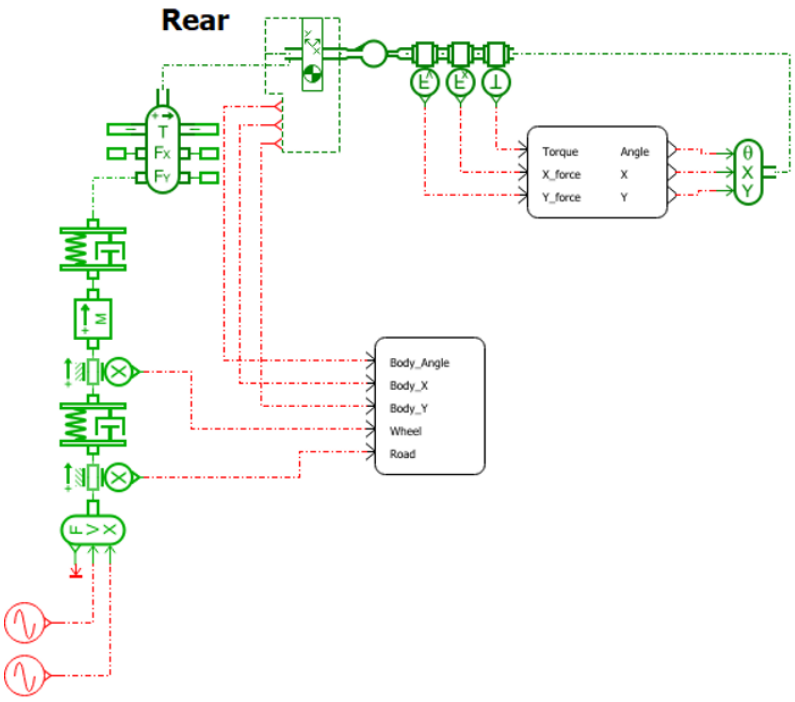
The last stage of this project consists of the implementation and running of a co-simulation across two real-time platforms, connected between themselves by an Ethernet cable. The RT platform has a Linux-based operative system designed by Siemens, specially built for RT co-simulation and testing purposes. SUs are uploaded to the platform as components, and the RT platform handles the orchestration. The orchestration uses Jacobian scheduling and non-iterative integrators. The platform offers compatibility with FMI and UserCosim, which is a AMESim specific interface standard.

6.2.1 AMESim models

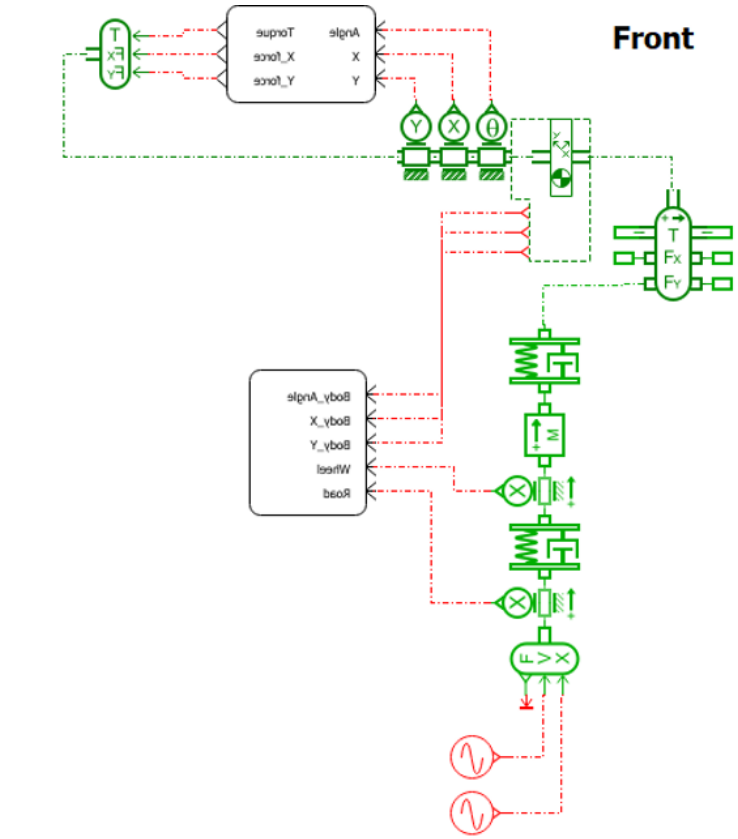
The models for this stage are very similar to the ones in Section 4.2.3. However, the interface needs to be compatible with either UserCosim or FMI, instead of relying on the shared memory created by the AMESim co-simulation block. This alteration can be seen in Figure 6.3, where the interface block substitutes the co-simulation block. Furthermore, there exists two interface blocks that allow the reading of internal variables that, otherwise, would be unavailable.

A local co-simulation was performed with these models, i.e. they were run on only one machine. From Section 4.2.3, the reader can remember that the poor results of the AMESim co-simulation could result from a poor functioning of the co-simulation blocks. This new co-simulation comes from an attempt to eliminate those issues and re-evaluate the models' validity. Looking at Figures 6.4 to 6.6, it is visible that some degree of stability has been achieved. However, this was only possible due to a very small macro and micro-step sizes (500 μs), which is uninspiring for the usage of these models in a distributed testing scenario. Furthermore, it is visible that the accuracy of the results is far beyond acceptable. In fact, for the front and rear vertical displacement there was a relative error of 121.8% and 121.7%, respectively. In the angular rotation, this error was of 81.7% for the front body and 81.4% for the rear body and, in terms of wheel displacement, the error was 89.1% and 142.5% for the front and rear wheels, respectively. This model is therefore unusable in a co-simulation scenario.

The difficulty in using this model is greatly due to the 2D Mechanical library input source blocks, which require the existence of non-explicit interface variables. Therefore, to reach a distributed co-simulation, the chosen solution was to use a model that does not rely on this library. The choice in this case relied on splitting the quarter car into the body and suspension part (Figure 6.7a) and into the wheel, tire and road part (Figure 6.7b). In terms of interface, it is visible the

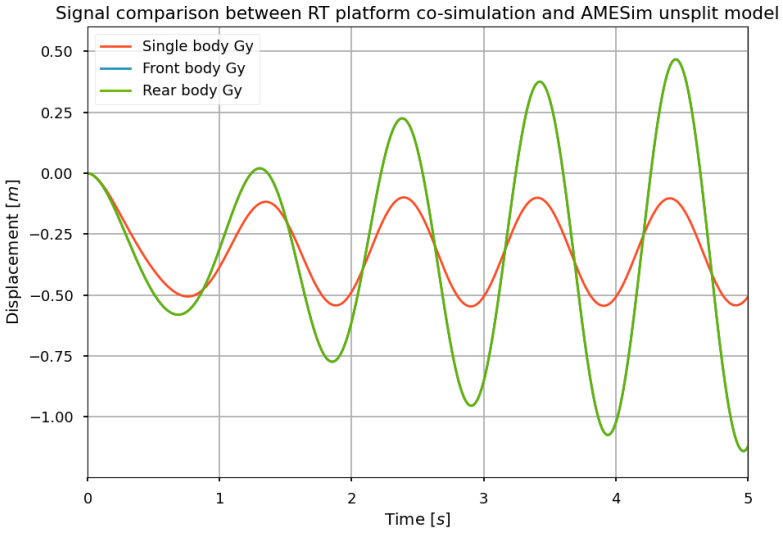


(a) Rear

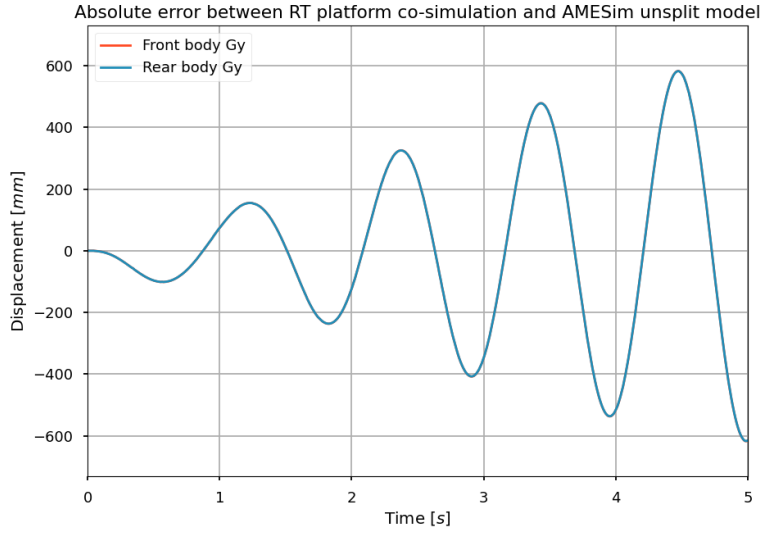


(b) Front

Figure 6.3: AMESim models with UserCosim interface

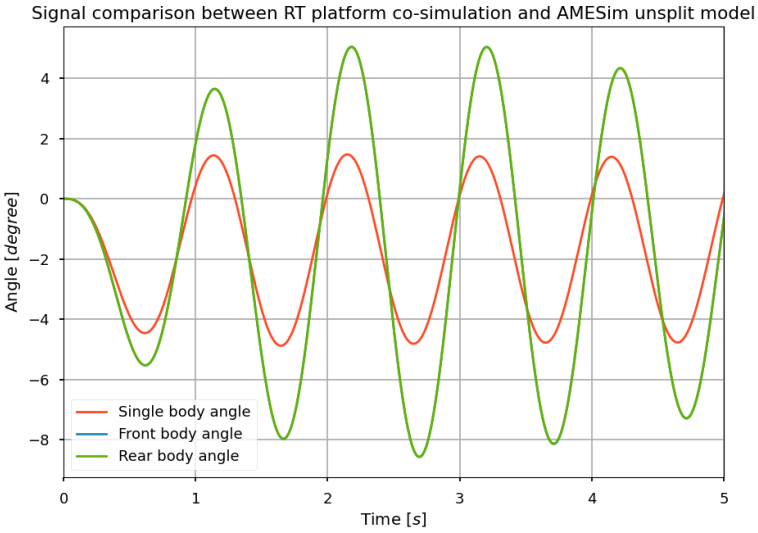


(a) Signal comparison

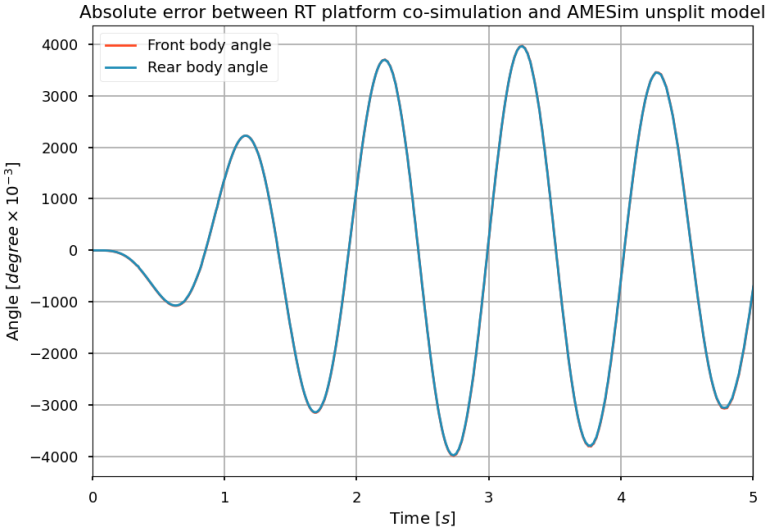


(b) Absolute error

Figure 6.4: Body vertical displacement comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim

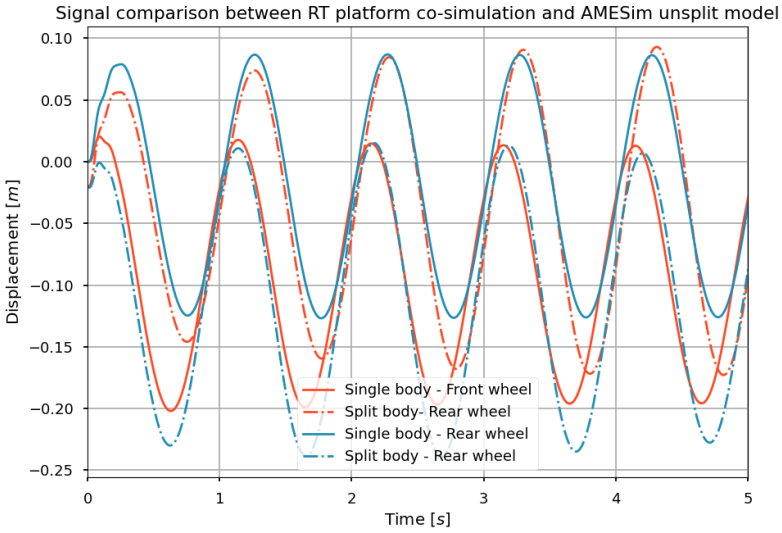


(a) Signal comparison

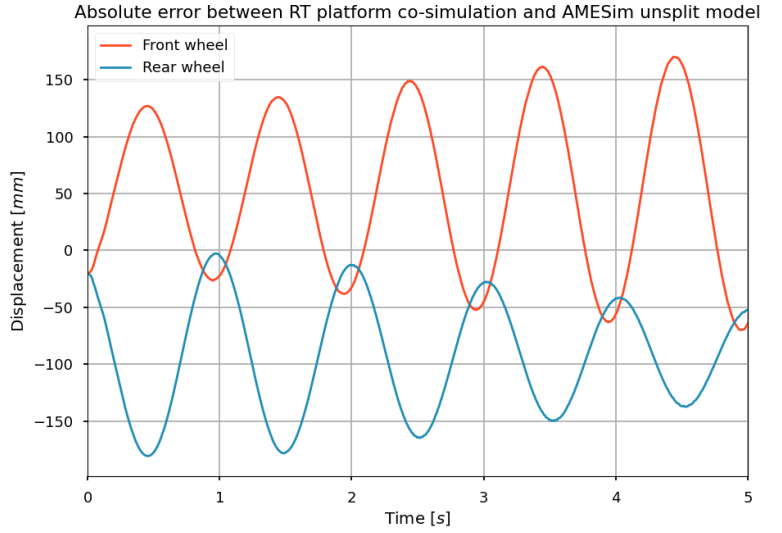


(b) Absolute error

Figure 6.5: Angular rotation comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim



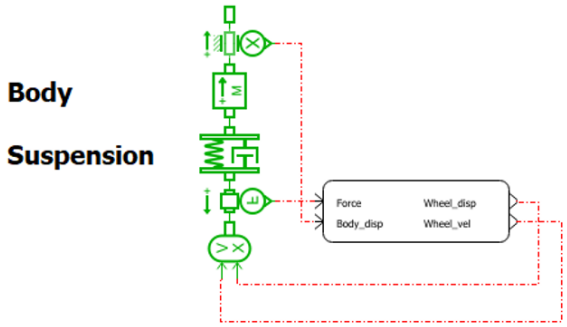
(a) Signal comparison



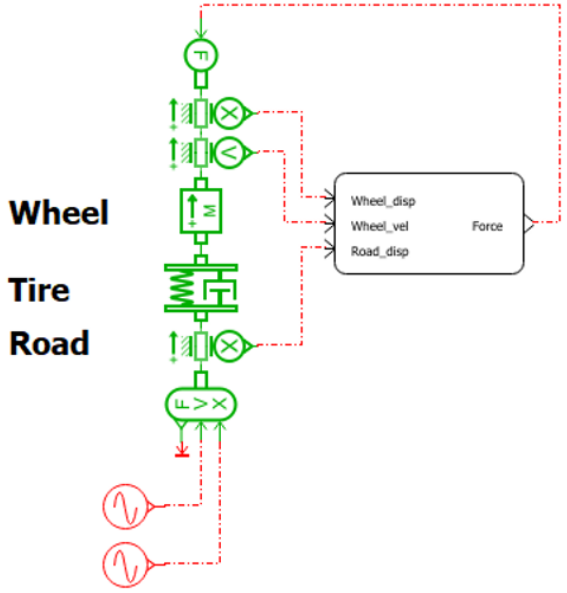
(b) Absolute error

Figure 6.6: Wheel displacement comparison between split bicycle car co-simulation in RT platform and single body bicycle car simulation in AMESim

difference with the bicycle car model. In fact, the input source to the suspension explicitly provides both the velocity and displacement (the acceleration variable is not part of the 1D mechanical library). This way, no integration or derivation is needed, avoiding the numerical constraints found in the bicycle car.



(a) Top



(b) Bottom

Figure 6.7: Split quarter car models for co-simulation

Running the models in AMESim allows to understand the influence of the model splitting on its accuracy. These results can be seen on Figure 6.8. First noticeable aspect is the coherency between the top and bottom models' wheel displacement. Second, the low error value denotes the co-simulation capability of this model. In fact the relative error for the body displacement is of 0.13% and for the wheel of 0.49%. This favours the validity of this model for co-simulation, even more because a macro-step size of 1 milliseconds was sufficient, unlike with the bicycle car.

6.2.2 Coupling elements

Because the RT platforms do not have any coupling methods included besides the direct coupling, those need to be implemented as FMUs, to be placed between the SUs. The coupler's

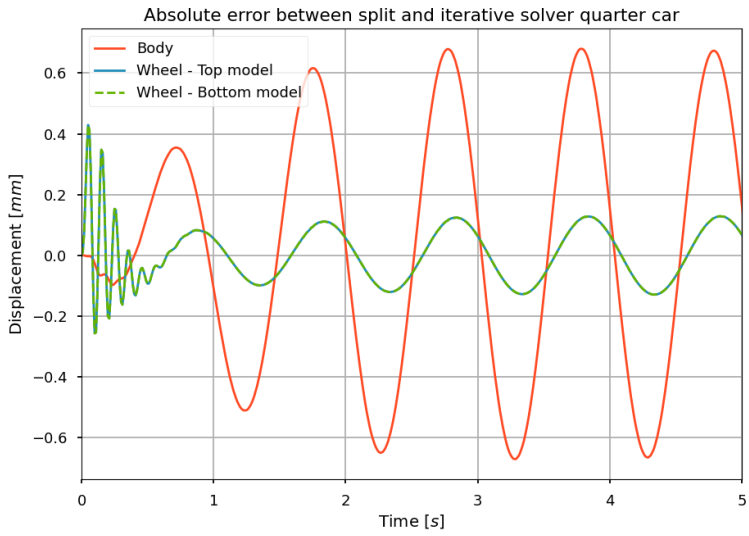


Figure 6.8: Absolute error between split quarter car model and unsplit quarter car model with iterative solver, for macro step of 1 milliseconds

architecture, as well as its surrounding, is presented in Figure 6.9. Furthermore, as the goal is for the sub-models to be placed in different machines, the coupling elements need also to address the communication procedure between them. This means that, even tough each machine contains an orchestrator, there is not a higher level orchestrator for the distributed co-simulation scenario. To deal with this communication-wise, one of the machines needs to be implemented as a master while the other as a slave. This forcefully requires the design of two different FMUs with synchronization and initialization routines.

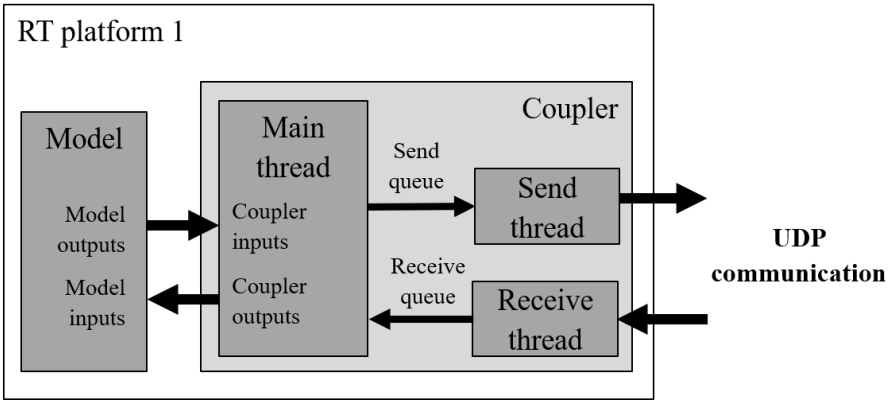


Figure 6.9: General coupler FMU architecture

The implementation of this FMUs is necessarily based on the FMI state machine presented on Figure 2.5. Having available a FMI-compliant source code, the focus resides on the *fmi2Instantiate*, *fmi2EnterInitializationMode*, *fmi2DoStep* and *fmi2FreeInstance* functions.

The *fmi2Instantiate* function is responsible for the creation of the FMU, which includes all the

variable declaration and initialization, as well as the creation of all the required files and threads. Therefore, the first architectural aspect to consider is that the communication needs to be independent from the FMU progression. Given that the coupling element is a co-simulation slave, the temporal progression happens according to the macro-steps and idling times in between them (the FMI standard follows an offline system approach). This should not be a constraint for the communication process, particularly on the receiving side. For this reason, two threads were created one for handling outgoing communication and the other for handling incoming communication. The threads communicate with the main thread through a "first in, first out" type of queue. This means that a thread pushes values to the queue while another pops values from the queue, whenever they exist, by the same order they were pushed in. The threads and queues organization can also be seen in Figure 6.9. The fmi2Instantiate function goes through the following steps:

1. Create log file for printing machine-related messages. This was created because there is no access to the RT platform terminal and, consequently, to messages usually printed there.
2. Create results file for printing the input, output and state variables at any given micro-step.
3. Declare local and remote UDP addresses, and initialize the local UDP port.
4. Create thread for sending UDP communication and await thread creation confirmation. This thread enters an infinite loop where, whenever a message can be popped from the sending queue, it is sent over UDP to the remote address.
5. Create thread for receiving UDP communication and await thread creation confirmation. This thread enters an infinite loop where the UDP port is waiting for incoming communication. The result of this waiting can be either:
    - timeout, if, for a given time, no message has been received;
    - warning, if the message received is not of the expected length, which indicates an incorrect sent message or data corruption;
    - success, if a message is correctly received. In this case, the message is pushed into the receiving queue;
6. Ring-buffer structure initialization with a buffer size depending on the coupling method chosen;
7. Creation of the buffers for storing the received time and interface values;
8. Initialization of outputs.

For the function *fmi2EnterInitializationMode* becomes important the hierarchy definition between the two RT platforms. As mentioned, one of the RT platforms will function as the orchestrator. For this, a communication procedure was created to sync the two machines. For explanation purposes, one of the machines will be hereafter called the master and the other the slave. The slave is the first RT platform to be deployed. First, its initialization routine halts until a message of type CC_SETUP is received. Once it happens, a message of type CC_SETUP_ACK is sent, and

the FMU's time is set to 0. After this, the slave proceeds its normal functioning. With the master an analogous approach is taken. First, the CC_SETUP message is sent and then it awaits the CC_SETUP_ACK message. This allows synchronization to some degree, but it is obviously influenced by the communication delays, as the slave will start its co-simulation environment earlier. A proper synchronization method, like the ones explained in Section 2.3.2 should be implemented. Furthermore, no initialization process is implemented in this phase which means that there is not compatibility assurance of initial values. Overall, more attention should be dedicated to this phase of distributed testing.

In function *fmi2DoStep*, the FMU progresses until a given time provided by the orchestrator. This means that a routine is repeated for every micro-step until that time is achieved. The routine includes, first, the attempt of popping a value from the receive queue and the corresponding update of the buffers. Secondly, the outputs are updated according to the coupling method, the received data buffers, and the current FMU time. Lastly, the message is updated to the current coupler input values (provided by the model) and the message is pushed to send queue. After all this, the current inputs and outputs are written to the results file.

fmi2FreeInstance function is responsible for, upon the ending of the co-simulation, terminating all the open files and threads. A more complex termination routine could also be implemented with the master commanding the slave to terminate as well. At the moment, this responsibility remains with the RT platform.

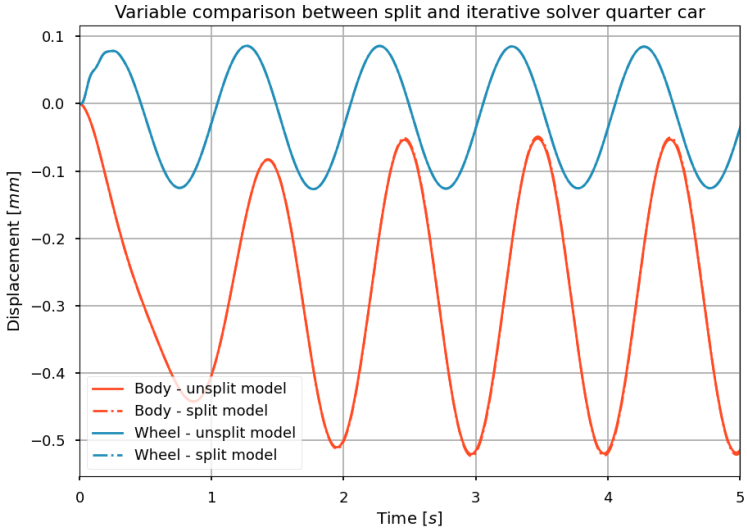
6.2.3 Distributed co-simulation

Having developed the quarter car top and bottom models and having built the coupling elements, the conditions for creating a distributed co-simulation scenario are in place. As no network impairment simulation or emulation was implemented in this stage, there is only the real physical impairments induced by the Ethernet cable directly connecting the two RT platforms. A ping routine across them allowed to measure an average delay of around 0.4 ms. The delay variation and packet loss were insignificant as there are no network routing mechanisms and no network congestion. It should be noticed that the platforms were placed next to each other. This means that, even tough the results from this section correspond to a distributed testing scenario, a proper geographical distribution was not in place. This inhibits the conclusions generalization for more demanding scenarios.

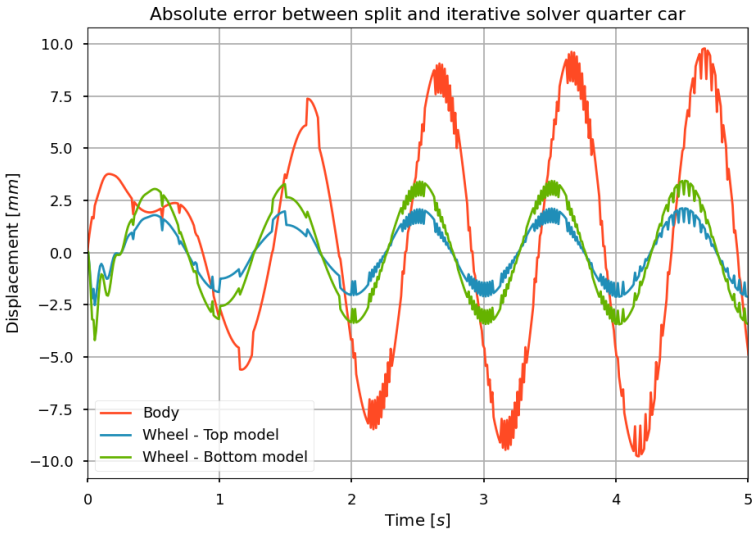
As can be seen from the results presented in Section 4.1.1, the variable's signals are relatively close to sinusoidal waves. This, allied to the approximation methods results presented in Section 5.3.2, led to the choice of SOH for the delay mitigation. The results for the quarter car distributed testing are visible in Figure 6.10, which represents the comparison with the iterative solution of the AMESim unsplit car. Notice that this results were obtained for a macro and micro-step size of 1 ms.

In Figure 6.10a, it can be seen that a good accuracy for the distributed co-simulation scenario was achieved. This is further confirmed by the absolute error values visible in Figure 6.10b and,

mainly, by the maximum relative errors. For the body displacement, this value was of 1.9% and, for the wheel displacement, 3.3%.



(a) Signal comparison



(b) Absolute error

Figure 6.10: Quarter car distributed testing results

However, as the delay is smaller than the macro-step size, it may be the case that it is not big enough to have effect on the co-simulation. This does not allow to properly assess the effectiveness of the mitigation method chosen. For that purpose, increasing the delay on the network, or at least simulating it, would be necessary. Furthermore, this work lacks the comparison of results to a scenario where no coupling methods are applied. This could allow to better perceive the effect of delay on distributed testing.

Chapter 7

Conclusion

Globally, this work allowed for an initial implementation of a distributed testing scenario. Using a split quarter car model with small communication delays led to a distributed testing scenario where a good accuracy of the simulation was maintained. However, it is understandable as well that the distributed scenario for which this was achieved was in fact a local area network, with delays around 0.4 ms.

In terms of the MiL stage, it is highlight the building of the split quarter car model. Although, this is of lower complexity than other models, it allows for earlier validation of the coupling elements designed. Regarding the building of split bicycle car model, it is understood that the 2D mechanical library, in its current state, imposes challenges concerning numerical errors of derivation and integration. It also became clear that the splitting of bodies requires the integration of stiff joints, thus increasing the computational complexity of the models.

Regarding the SiL stage, the conclusions can be separated into two parts. In terms of network impairments, a reasonable network impairment simulation was implemented allowing for constant delay and packet loss simulation. However, in terms of delay variance, a normal distribution was found to be unfitting due to the asymmetric nature of delay variation. The reason is that this type of function allows for negative values of delay, which end up representing unsent messages. This corroborates the suggestion, made by Schreiber et al. (2018), of using probability density functions based on kernel density estimators. Furthermore, it is noticeable that delay variation caused out-of-order messages, an impairment which was unaccounted for. In terms of correction methods, the signal-based ones were implemented due to the low implementation effort. This methods showed to be fitting for low impairment scenarios. For sinusoidal waves, the SOH method proved to be the most accurate and robust to all the impairments. However, the incapacity of this methods to acknowledge periodicity led them to being quite poor with more impaired communication. In fact, they can not approximate signals with delays higher than half the periodicity of the signal and suffer with high packet loss ratios. The periodicity issues mean also that all methods failed to compensate the delay for rectangular waves.

In the HiL stage, a network quality measurement program for UDP/IP was successfully created. This allows for evaluation of the real network impairments. However, at that time of this

project, the network to be used was not yet established. The programs were tested in local area networks and are ready to use, once the necessary infrastructures are in place. In terms of the distributed testing, this was achieved for low impaired networks, using SOH approximation. For further validation of this distributed testing scenario, a wide-area network, a network simulation or network emulation, would be required

Overall, an initial setup for distributed co-simulation was built with the required communication protocols and some coupling methods alternatives. It still lacks, however, the implementation of proper synchronization and initialization procedures, as well as other coupling methods, that would enhance the coupling elements capability for distributed testing.

7.1 Further Work

Obviously, given the point at which the work was stopped, the author has a few ideas of where future work should be headed. Many of them are aspects that were planned for this thesis but that, unfortunately, could not be developed. A list of these suggestions is hereby presented:

- Solve the problems with the bicycle car model. This could be done by either finding alternatives to the input sources used or by studying with the AMESim developing team the possibility of including an input source block that allows to explicitly input all interface variables;
- Devise guidelines for the proper splitting of models for co-simulation, further addressing the stability issues already acknowledge in the field;
- Implement other more promising coupling methods. The author advises to focus on the model-based coupling methods, although waveform property methods also appear to be promising;
- Develop and implement proper time synchronization mechanisms across the devices. Given the requirements of the macro-step sizes used (around 1ms), probably PTP is more advisable;
- Develop and implement proper initialization algorithms that allow the coupling methods to surpass their learning phase. This could prevent instability problems caused by poor variable initialization.
- Assess the real network impairments of the XiLforEV network, using the program created for that purpose;
- Compare UDP and RTP in terms of their impairments;
- Further develop network impairment simulation and address out of order messages;
- Assess and compare multiple coupling methods in terms of their capability of solving the network impairments issues;

- Incorporate noise on the interface signals and verify the influence it has on the co-simulation.
- Test the capability of the methods to handle noise.
- Incorporate test control layer in the distributed testing scenario, with communication over TCP, for initialization and synchronization purposes;
- Devise the guidelines for a a distributed testing scenario with particular focus on the limits of impairments, macro and micro-step sizes and noise.

References

Albers, A., Behrendt, M., Schroeter, J., Ott, S. & Klingler, S. (2013) X-in-the-loop: A framework for supporting central engineering activities and contracting complexity in product engineering processes. *Proceedings of the International Conference on Engineering Design, ICED*, 6 DS75-06 (August), 391–400.

Andersson, C. (2016) *Methods and Tools for Co-Simulation of Dynamic Systems with the Functional Mock-up Interface*. Ph.D. thesis, Lund Univeristy. Available from: https://lup.lub.lu.se/search/publication/8865908.

ASAM (2020) ASAM XIL. Available from: https://www.asam.net/standards/detail/xil/ [Accessed: 2020-09-10].

AXIOMATIC Global Electronic Solutions (2006) Q&A - What is CAN? Available from: https://www.axiomatic.com/whatiscan.pdf.

Bernard, C., Büchner, F., Ganier, C., Ivanov, V., Lehne, C., Molina, J.V., Povalej, Ž., Ricciardi, V. & Schreiber, V. (2019) *Global XILforEV architecture*. Tech. Rep. 1, XILforEV Consortium.

BluAir (n.d.) Bluetooth Range. Available from: http://www.bluair.pl/bluetooth-range [Accessed: 2020-09-10].

Brembeck, J., Pfeiffer, A., Fleps-Dezasse, M., Otter, M., Wernersson, K. & Elmqvist, H. (2014) Nonlinear State Estimation with an Extended FMI 2.0 Co-Simulation Interface. In *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, vol. 96, pp. 53–62. Linköping University Electronic Press. doi:10.3384/ecp1409653. Available from: http://www.ep.liu.se/ecp/article.asp?issue=096{%}26article=5.

Busch, M. (2016) Continuous approximation techniques for co-simulation methods: Analysis of numerical stability and local error. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 96 (9), 1061–1081. doi:10.1002/zamm.201500196. Available from: http://doi.wiley.com/10.1002/zamm.201500196.

Dronka, S. & Rauh, J. (n.d.) Co-Simulation-Interface for User-Force-Elements. Available from: http://simpack.testdomain.eu/fileadmin/simpack/doc/usermeeting06/um06{_}dc{_}research-dronka.pdf [Accessed: 2020-09-10].

Durresi, A. & Jain, R. (2005) RTP, RTCP, and RTSP - Internet Protocols for Real-Time Multimedia Communication. In R. Zurawski (ed.), *The Industrial Information Technology Handbook*, chap. 28. CRC Press.

European Commission (2013) H2020-EU.3.4. - SOCIETAL CHALLENGES - Smart, Green And Integrated Transport. *Official Journal of the European Union*, pp. 1–4.

Feki, A.B.K.E., Duval, L., Faure, C., Simon, D. & Gaid, M.B. (2016) CHOPtrey: contextual online polynomial extrapolation for enhanced multi-core co-simulation of complex systems. *Simulation*, 93 (3), 185–200. doi:10.1177/0037549716684026. Available from: http://arxiv.org/abs/1610.09261http://dx.doi.org/10.1177/0037549716684026. 1610.09261.

Gomes, C., Broman, D., Vangheluwe, H., Thule, C. & Gorm Larsen, P. (2018) Co-simulation: a Survey. *ACM Computing Surveys*, 1 (1), 35. Available from: http://www.v2c2.at/research/ee-software/projects/acorta/.

Harish, A. (2020) Implicit vs Explicit Finite Element Method (FEM): What Is the Difference? Available from: https://www.simscale.com/blog/2019/01/implicit-vs-explicit-fem/ [Accessed: 2020-09-10].

Hemminger, S. (2005) Network Emulation with NetEm. *Proceedings of the 6th Australian National Linux Conference (LCA 2005)*, (April), 1–9.

Hoel, P.G. (1965) Optimum Designs for Polynomial Extrapolation. *The Annals of Mathematical Statistics*, 36 (5), 1483–1493. doi:10.1214/aoms/1177699907. Available from: https://projecteuclid.org/euclid.aoms/1177699907.

IEEE Standards Association (2010) 1516-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules. Available from: https://standards.ieee.org/standard/1516-2010.html [Accessed: 2020-09-10].

IEEE Standards Association (2015) 1278.2-2015 - IEEE Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles. Available from: https://standards.ieee.org/standard/1278{_}2-2015.html [Accessed: 2020-09-10].

Ivanov, V., Augsburg, K., Bernad, C., Dhaens, M., Dutré, M., Gramstat, S., Magnin, P., Schreiber, V., Skrt, U. & Van Kelecom, N. (2019) Connected and Shared X-in-the-Loop Technologies for Electric Vehicle Design. *World Electric Vehicle Journal*, 10 (4), 83. doi:10.3390/wevj10040083. Available from: https://www.mdpi.com/2032-6653/10/4/83.

Jazar, R.N. (2008) *Vehicle Dynamics: Theory and Application*. Springer US, Boston, MA. doi:10.1007/978-0-387-74244-1. Available from: http://link.springer.com/10.1007/978-0-387-74244-1.

Krammer, M., Msc, M.B., Blochwitz, T., Alekeish, K., Amringer, N., Kater, C., Materne, S., Ruvalcaba, R., Schuch, K., Zehetner, J., Damm-Norwig, M., Schreiber, V., Nagarajan, N., Corral, I., Sparber, T., Klein, S. & Andert, J. (2018) The Distributed Co-Simulation Protocol for the Integration of Real-Time Systems and Simulation Environments. In *Proceedings of the 50th Computer Simulation Conference*, vol. 50, pp. 1–14. Society for Modeling and Simulation International (SCS). doi:10.22360/SummerSim.2018.SCSC.001. Available from: http://dl.acm.org/citation.cfm?id=3275383.

Kumar, D. & Kumari, P. (2020) A parameter-uniform scheme for singularly perturbed partial differential equations with a time lag. *Numerical Methods for Partial Differential Equations*, 36 (4), 868–886. doi:10.1002/num.22455. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/num.22455.

Kwon, M. (2015) A Tutorial on Network Latency and Its Measurements. In T. Soyata (ed.), *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, chap. 9, pp. 272–293. IGI Global, Hershey PA, USA. doi:10.4018/978-1-4666-8662-5.ch009. Available from: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-8662-5.ch009.

Liu, R., Mohanpurkar, M., Panwar, M., Hovsapian, R., Srivastava, A. & Suryanarayanan, S. (2017) Geographically distributed real-time digital simulations using linear prediction. *International Journal of Electrical Power & Energy Systems*, 84, 308–317. doi:10.1016/j.ijepes.2016.06.005. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0142061516308079.

Ljung, L. (1999) *System identification: theory for the user*. Prentice Hall PTR, 2nd edn.

iTrinegy Ltd (2015) *Network emulation essentials (white paper)*.

Ma, X. & Gao, J. (2012) The Comparison And Analysis Of The Streaming Media Transport Protocol In The Transmission System. In *2012 International Conference on Education Technology and Computer (ICETC2012)*, vol. 43. IACSIT Press, Singapore.

Maurer, M. (2013) Automotive Systems Engineering: A Personal Perspective. In M. Maurer & H. Winner (eds.), *Automotive Systems Engineering*, pp. 17–35. Springer.

Mills, D.L. (1991) Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications*, 39 (10), 1482–1493. doi:10.1109/26.103043.

Modelica Association Project DCP (2019) *Distributed Co-Simulation Protocol Specification Version 1.0*. Tech. rep. Available from: www.dcp-standard.org.

Modelica Association Project DCP (n.d.) DCP. Available from: https://dcp-standard.org/ [Accessed: 2020-09-10].

Modelica Association Project "FMI" (2013) *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Tech. Rep. 07006.

Modelica Association Project FMI (n.d) FMI. Available from: https://fmi-standard.org/ [Accessed: 2020-09-10].

Neagoe, T., Cristea, V. & Banica, L. (2006) NTP versus PTP in computer networks clock synchronization. In *IEEE International Symposium on Industrial Electronics*, vol. 1, pp. 317–322. doi:10.1109/ISIE.2006.295613.

Nguyen, V.H., Nguyen, T.L., Tran, Q.T. & Besanger, Y. (2019) Synchronization conditions and Real-time constraints in co-simulation and Hardware-in-the-Loop techniques for Cyber–Physical Energy System assessment. *Sustainable Energy, Grids and Networks*, 20, 100252. doi:10.1016/j.segan.2019.100252. Available from: https://linkinghub.elsevier.com/retrieve/pii/S2352467719302267.

Pace, D.K. (2004) Modeling and simulation verification and validation challenges. *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, 25 (2), 163–172.

Payne, M. (n.d) What is the Maximum Length of a USB Cable? Available from: https://www.techwalla.com/articles/what-is-the-maximum-length-of-a-usb-cable [Accessed: 2020-09-10].

Sadjina, S., Kyllingstad, L.T., Rindarøy, M., Skjong, S., Æsøy, V. & Pedersen, E. (2019) Distributed Co-simulation of Maritime Systems and Operations. *Journal of Offshore Mechanics and Arctic Engineering*, 141 (1). doi:10.1115/1.4040473. 1701.00997.

Sadjina, S., Kyllingstad, L.T., Skjong, S. & Pedersen, E. (2017) Energy conservation and power bonds in co-simulations: non-iterative adaptive step size control and error estimation. *Engineering with Computers*, 33 (3), 607–620. doi:10.1007/s00366-016-0492-8. Available from: https://link.springer.com/article/10.1007/s00366-016-0492-8.

dos Santos, F.L.M., Pastorino, R., Peeters, B., Faria, C., Desmet, W., Sandoval Góes, L.C. & Van Der Auweraer, H. (2016) Model Based System Testing: Bringing Testing and Simulation Close Together. In A. Wicks & C. Niezrecki (eds.), *Conference Proceedings of the Society for Experimental Mechanics Series*, vol. 7, pp. 91–97. Springer, Cham. doi:10.1007/978-3-319-29956-3_10. Available from: http://link.springer.com/10.1007/978-3-319-29956-3{_}10.

Sargent, R.G. (2010) Verification and validation of simulation models. In B. Johansson (ed.), *Proceedings of the 2010 Winter Simulation Conference*, pp. 166–183. IEEE, Piscataway, NJ, USA. doi:10.1109/WSC.2010.5679166. Available from: http://ieeexplore.ieee.org/document/5679166/.

Schreiber, V., Ivanov, V., Augsburg, K., Noack, M., Shyrokau, B., Sandu, C. & Els, P.S. (2018) Shared and Distributed X-in-the-Loop Tests for Automotive Systems: Feasibility Study. *IEEE Access*, 6, 4017–4026. doi:10.1109/ACCESS.2017.2789020. Available from: https://ieeexplore.ieee.org/document/8244275/.

Shampine, L. & Thompson, S. (2012) Delay-Differential Equations with Constant Lags. *CODEE Journal*, 9 (1), 1–5. doi:10.5642/codee.201209.01.10. Available from: http://scholarship.claremont.edu/codee/vol9/iss1/10/.

Shannon, C. (1949) Communication in the Presence of Noise. *Proceedings of the IRE*, 37 (1), 10–21. doi:10.1109/JRPROC.1949.232969. Available from: http://ieeexplore.ieee.org/document/1697831/.

Siemens Digital Industries Software (2019) *Model-based system testing (white paper)*. Available from: https://www.plm.automation.siemens.com/global/en/resource/virtual-sensing-techniques-and-their-applications/86020 [Accessed: 2020-09-10].

Stettinger, G., Benedikt, M., Horn, M., Zehetner, J. & Giebenhain, C. (2017) Control of a magnetic levitation system with communication imperfections: A model-based coupling approach. *Control Engineering Practice*, 58 (February 2016), 161–170. doi:10.1016/j.conengprac.2016.10.009. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0967066116302374.

Stettinger, G., Horn, M., Benedikt, M. & Zehetner, J. (2014a) A model-based approach for prediction-based interconnection of dynamic systems. *Proceedings of the IEEE Conference on Decision and Control*, 2015-Febru (February), 3286–3291. doi:10.1109/CDC.2014.7039897.

Stettinger, G., Horn, M., Benedikt, M. & Zehetner, J. (2014b) Model-based coupling approach for non-iterative real-time co-simulation. In *2014 European Control Conference (ECC)*, pp. 2084–2089. IEEE. doi:10.1109/ECC.2014.6862242. Available from: http://ieeexplore.ieee.org/document/6862242/.

Stettinger, G., Zehetner, J., Benedikt, M. & Thek, N. (2013) Extending Co-Simulation to the Real-Time Domain. In *SAE Technical Papers*, vol. 2. SAE International. doi:10.4271/2013-01-0421. Available from: https://www.sae.org/content/2013-01-0421/.

Stevic, M., Panwar, M., Mohanpurkar, M., Hovsapian, R. & Monti, A. (2017) Empirical study of simulation fidelity in geographically distributed real-time simulations. In *2017 North American Power Symposium (NAPS)*, pp. 1–6. IEEE. doi:10.1109/NAPS.2017.8107236. Available from: http://ieeexplore.ieee.org/document/8107236/.

Van der Auweraer, H., Sarrazin, M. & Santos, F. (2017) Model Based System Testing: a New Drive To Integrating Test and Simulation. In *International Conference on Structural Enginnering Dynamics*, August. Ericeira.

Vogel, S., Rajkumar, V.S., Nguyen, H.T., Stevic, M., Bhandia, R., Heussen, K., Palensky, P. & Monti, A. (2019) Improvements to the Co-simulation Interface for Geographically Distributed Real-time Simulation. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, pp. 6655–6662. IEEE. doi:10.1109/IECON.2019.8926918. Available from: https://ieeexplore.ieee.org/document/8926918/.

Watt, S.T., Achanta, S., Abubakari, H., Sagen, E., Korkmaz, Z. & Ahmed, H. (2016) Understanding and applying precision time protocol. In *2015 Saudi Arabia Smart Grid, SASG 2015*. Institute of Electrical and Electronics Engineers Inc. doi:10.1109/SASG.2015.7449285.

Wheeb, A.H. (2015) Performance Comparison of Transport Layer Protocols. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5 (12), 121–125. Available from: https://www.researchgate.net/publication/291775032_Performance_Comparison_of_Transport_Layer_Protocols#fullTextFileContent [Accessed: 2020-10-13].