Implementation in embedded systems of state observers based on multibody dynamics

Antonio J. Rodríguez

Advisors: Miguel Ángel Naya Villaverde
Roland Pastorino

Doctoral thesis

UNIVERSIDADE DA CORUÑA

Programa Oficial de Doutoramento
en Enxeñaría Naval e Industrial

Ferrol, 2020

A mi abuelo Mariano, que sé que estaría orgulloso

Acknowledgements

Finishing a doctoral thesis is supposed to be a personal achievement. However, such a work can only be accomplished thanks to the contribution of several people. It is a pleasure to acknowledge them all their merits in this thesis.

First, I would like to mention my advisors, Miguel Ángel Naya and Roland Pastorino, who have guided me through all this years, and from who I have learnt most of the things that I know. I must also mention all my colleagues at the Laboratorio de Ingeniería Mecánica of the University of A Coruña, since all of them contribute to the incredible environment at work: Javier, Urbano, Daniel, Florian, Fran González, Fran Mouzo, Alberto, Emilio, Lolo, Borja, Sarath, Mario, Álvaro, Alex, Manu, David and Amelia. I would like to highlight the help provided by Alberto and Fran Mouzo to solve most of the technical problems that I found. I must give special thanks to Emilio, whose contribution to the results obtained in this thesis was determinant. The help of Ángel Carro-Lagoa with the FPGA, from the Department of Computer Engineering, is also greatly acknowledge.

Thanks also to the MBST team from Siemens Digital Industries Software in Leuven, Belgium, for their support during my stays working with them, especially to Bart Forrier. Also thanks to Professor Wim Desmet, from the KU Leuven, for facilitate my first stay in Belgium. Thanks also to Carolina who, with Roland, made my stays in Belgium more comfortable.

Although the technical support is important, life goes on outside the work. Despite I am a bit reserved and I usually avoid talking of the work with my family and friends, they have always supported me. I would like to thank my mother (María), my sister (María), my godparents (Mariano and Madrona), my grand-mother (Marusia) and my grand-aunt (Chita), always caring about me. And at last, but not less important, thanks to Mariola, who is always beside me, encouraging me to discover new things and making life funnier.

Agradecimientos

Se supone que terminar una tesis doctoral es un gran logro personal. Sin embargo, algo así sólo se puede terminar con ayuda de mucha gente. Es un placer poder reconocerles en esta sección su contribución a esta tesis.

En primer lugar, quisiera nombrar a mis directores de tesis, Miguel Ángel Naya y Roland Pastorino, que me han guiado todos estos años y de quienes he aprendido la mayoría de las cosas que sé. También debo mencionar a mis compañeros del Laboratorio de Ingeniería Mecánica de la Universidade da Coruña, ya que ellos hacen que el ambiente en el laboratorio sea increíble: Javier, Urbano, Daniel, Florian, Fran González, Fran Mouzo, Alberto, Emilio, Lolo, Borja, Sarath, Mario, Álvaro, Alex, Manu, David y Amelia. Me gustaría remarcar la ayuda que me han dado Alberto y Fran Mouzo, a los que he recurrido con cualquier problema técnico que me encontraba. También tengo mucho que agradecer a Emilio, cuya contribución para obtener los resultados de esta tesis ha sido vital. La ayuda de Ángel Carro-Lagoa con la FPGA también ha sido muy importante.

Gracias también al equipo MBST de Siemens Digital Industries Software en Lovaina, Bélgica, por su apoyo durante mis estancias trabajando con ellos, especialmente a Bart Forrier, con quien más he colaborado. Gracias también al Profesor Wim Desmet, de la KU Leuven, por facilitar mi primera estancia en Bélgica. Y gracias a Carolina que, junto con Roland, hicieron mis estancias en Bélgica mucho más agradables.

Pero aunque la ayuda en los temas técnicos es importante, la vida sigue al salir del trabajo. Aunque evito y esquivo preguntas sobre los temas del trabajo con mi familia y mis amigos, siempre me han apoyado y han mostrado interés en mi tesis. Gracias a mi madre (María), mi hermana (María), mis padrinos (Mariano y Madrona), mi abuela (Marusia) y mi tía abuela (Chita), que siempre se preocupan por mí. Y por último, aunque no por ello menos importante, gracias a Mariola, que siempre está a mi lado empujándome a descubrir cosas nuevas y haciendo más divertida mi vida.

Abstract

Simulation has become an important tool in the industry that minimizes either the cost and time of new products development and testing. In the automotive industry, the use of simulation is being extended to virtual sensing. Through an accurate model of the vehicle combined with a state estimator, variables that are difficult or costly to measure can be estimated.

The virtual sensing approach is limited by the low computational power of in-vehicle hardware due to the strictest timing, reliability and safety requirements imposed by automotive standards. With the new generation hardware, the computational power of embedded platforms has increased. They are based on heterogeneous processors, where the main processor is combined with a co-processor, such as Field Programmable Gate Arrays (FPGAs).

This thesis explores the implementation of a state estimator based on a multibody model of a vehicle in new generation embedded hardware. Different implementation strategies are tested in order to explore the advantages that an FPGA can provide. A new state-parameter-input observer is developed, providing accurate estimations. The proposed observer is combined with an efficient multibody model of a vehicle, achieving real-time execution.

Resumen

La simulación se ha convertido en una importante herramienta para la industria que permite minimizar tanto costes como tiempo de desarrollo y test de nuevos productos. En automoción, el uso de la simulación se extiende al desarrollo de sensores virtuales. Mediante un modelo preciso de un vehículo combinado con un observador de estados, variables que son caras o imposibles de medir pueden ser estimadas.

La principal limitación para utilizar sensores virtuales en los vehículos es la baja potencia computacional de los procesadores instalados a bordo, debido a los estrictos requisitos impuestos por los standards de automoción. Con el hardware de nueva generación, el poder de cálculo de las plataformas empotradas se ha visto incrementado. Estos nuevos procesadores son del tipo heterogéneo, donde el procesador principal se complementa con un co-procesador, como una Field Programmable Gate Array (FPGA).

Esta tesis explora la implementación de un observador de estados basado en un modelo multicuerpo de un vehículo en hardware empotrado de nueva generación. Se han probado diferentes implementaciones para evaluar las ventajas de disponer de una FPGA en el procesador. Se ha desarrollado un nuevo observador de estados, parámetros y entradas que permite obtener estimaciones de gran precisión. Combinando dicho observador con un eficiente modelo multicuerpo de un vehículo, se consigue rendimiento en tiempo real.

Resumo

A simulación estase a converter nunha importante ferramenta na industria que permite minimizar custes e tempo tanto de desenvolvemento coma de test de novos productos. En automoción, o uso da simulación esténdese á implementación de sensores virtuais. Mediante un modelo preciso dun vehículo combinado cun observador de estados, pódense estimar variables que son caras ou imposíbeis de medir.

A principal limitación para utilizar sensores virtuais nos vehículos é a baixa potencia computacional dos procesadores instalados a bordo, debido aos estritos requisitos impostos polos estándares de automoción. Co hardware de nova xeración, o poder de cálculo das plataformas empotradas vese incrementado. Estos novos procesadores son de tipo heteroxéneo, onde o procesador principal compleméntase cun co-procesador, coma unha Field Programmable Gate Array (FPGA).

Esta tese explora a implementación dun observador de estados basado nun modelo multicorpo dun vehículo en hardware empotrado de nova xeración. Diferentes implementacións foron probadas para avaliar as vantaxes de dispoñer dunha FPGA no procesador. Un novo observador de estados, parámetros e entradas deseñado nesta tese permite obter estimacións de gran precisión. Combinando dito observador cun eficiente modelo multicorpo dun vehículo, conséguese rendemento de tempo real.

Contents

List of Figures

List of Tables

Acronyms

ABS Antilock Brake System. 5, 6

ADAS Advanced Driver Assistance System. 2, 6, 11, 31, 32, 48, 124

ASIC Application-Specific Integrated Circuit. 32

BRAM Block Random Access Memory. 33

CEKF continuous extended Kalman filter. 9, 10, 14

DAE differential-algebraic equation. 9

DEKF discrete extended Kalman filter. 10, 14, 15, 28

DOF degree-of-freedom. 6–8, 18, 25–29, 42, 45, 51, 52, 68, 106

DSP Digital Signal Processor. 33, 39, 46

ECU Electronic Control Unit. 5, 6, 10–12, 31

EKF extended Kalman filter. 2, 3, 8, 9, 14, 16, 17, 26, 28, 80, 107, 130

errorEKF error-state extended Kalman filter. iii, iv, vii, 10, 17, 18, 26, 27, 35, 60, 61, 66, 68–81, 83, 90, 91, 99, 101–104, 106, 124, 128, 129

FF Flip-Flop. 33, 46

FL Front-left wheel. iii–v, 64, 75, 77, 86, 88, 96

FPGA Field Programmable Gate Array. vii, viii, 3, 11, 12, 32–41, 44–50, 66, 67, 100–102, 105–107, 121, 124–131

FR Front-right wheel. iv, v, 64, 75, 77, 87, 89, 96

GJ Gauss-Jordan. vii, viii, 46–48, 67, 100–102, 127, 128

GPP General Purpose Processor. 6, 36, 48

GPU Graphic Processing Unit. 11, 32, 48

HDL Hardware Description Language. 34

HIL Hardware-in-the-loop. 1, 9, 123

HITL Human-in-the-loop. 1, 9, 123

LIM Laboratorio de Ingeniería Mecánica. 2, 124

LUT Look-Up Table. 32, 33, 39

MB multibody. iii, v, 1–5, 8–14, 16–18, 24–32, 34–36, 38, 40–42, 46, 48–52, 56, 57, 60, 61, 66–69, 80, 99–101, 104–107, 121

MCU Micro Computer Unit. 11

MPSoC Multiprocessor System on Chip. 32

PC personal computer. 2, 3, 6, 10, 24

RL Rear-left wheel. iv, v, 65, 76, 78, 87, 89, 97, 102

RMSE Root-mean-square error. vii, viii, 66, 79, 90, 91, 98, 102, 103

RR Rear-right wheel. iv, v, 65, 66, 76, 78, 79, 88, 90, 91, 97, 98, 102, 103

SPI state-parameter-input. iv, v, vii, viii, 3, 80–99, 101–104, 106, 121, 128, 129

SPKF sigma-point Kalman filter. 9, 16

SSUKF spherical simplex unscented Kalman filter. 9

UKF unscented Kalman filter. xii, 8–10, 16, 17, 80, 81, 101, 104, 106, 107, 128–130

VHDL Very High Speed Integrated Circuit Hardware Description Language. 34

List of symbols

The symbols used all along the document are defined here. The symbols employed in small parts of the document are defined locally.

Δt Time step. 23, 24, 27, 30, 103

$\bar{\mathbf{M}}$ Mass matrix projected over the body coordinates. 22

$\bar{\mathbf{Q}}$ Vector of generalized forces projected over the body coordinates. 22

$\mathbf{\Sigma}_c^P$ Covariance matrix of plan noise for continuous-time filters. 13–15

$\mathbf{\Sigma}^P$ Covariance matrix of plan noise for discrete-time filters. 14, 15, 27, 30

$\mathbf{\Sigma}^S$ Covariance matrix of sensor noise. 13–15, 17, 28

$\mathbf{\Sigma}$ Innovation covariance matrix. 15, 28

$\boldsymbol{\alpha}$ Matrix of penalty coefficients. 21, 24

$\ddot{\mathbf{\Phi}}$ Second time derivative of the constraints vector. 24

$\ddot{\mathbf{z}}^i$ Vector of independent accelerations. 26, 27, 29, 57–60

$\ddot{\mathbf{z}}$ Vector of relative accelerations. 20–24, 28

$\dot{\mathbf{\Phi}}$ Time derivative of the constraints vector. 24, 29

$\dot{\mathbf{R}}$ First derivative of the tranformation matrix from relative to body coordinates. 21, 22

$\dot{\mathbf{Z}}$ First derivative with respect to time of body coordinates. 20–22

$\dot{\mathbf{z}}^d$ Vector of dependent velocities. 29

$\dot{\mathbf{z}}^i$ Vector of independent velocities. 26, 27, 29, 57–60

$\hat{\ddot{\mathbf{z}}}^d$ Estimated vector of dependent accelerations. 29

$\hat{\ddot{\mathbf{z}}}$ Estimated vector of accelerations. 29

$\hat{\dot{\mathbf{z}}}^d$ Estimated vector of dependent velocities. 29

$\hat{\mathbf{M}}$ Mass matrix considering the estimated coordinates. 29

$\hat{\mathbf{Q}}$ Vector of generalized forces considering the estimated coordinates. 29, 56

$\hat{\mathbf{z}}^d$ Estimated vector of dependent coordinates. 28

$\mathbf{0}$ Null matrix. 22–24, 27, 29, 57–60

$\mathbf{C}$ Damping matrix. 24, 27

$\mathbf{F}$ Transition matrix of a discrete linear system. 14, 15

$\mathbf{G}$ Input matrix. 15

$\mathbf{H}$ Measurement sensitivity matrix. 14–16

$\mathbf{I}$ Identity matrix. 16, 22, 27, 28, 57–60

$\mathbf{K}$ Stiffness matrix. 24, 27

$\mathbf{M}$ Mass matrix. 21, 22, 24, 27

$\mathbf{P}$ Covariance matrix of state estimation uncertainty. 14–17, 27, 28

$\mathbf{Q}$ Vector of generalized forces. 21, 22, 24, 29, 56

$\mathbf{R}^{\boldsymbol{\Phi}}$ Projection matrix from dependent to independent relative coordinates. 27, 68

$\mathbf{R}$ Tranformation matrix from relative to body coordinates. 21, 22

χ Matrix containing the sigma-points of the unscented Kalman filter. 16, 17

$\mathbf{h}_{\mathbf{x}}$ Jacobian of measurement equation with respect to states. 28

$\mathbf{h}$ Measurement equation. 17, 28, 57–60

$\mathbf{o}$ Vector of observations. 14–17, 28

$\mathbf{u}$ Vector of inputs. 13–15

$\mathbf{v}$ Vector of measurement noise. 13–15

$\mathbf{w}$ Vector of process noise. 13–15

$\mathbf{y}$ Vector of outputs. 13–15

$\mathbf{z}^d$ Vector of dependent coordinates. 29

$\mathbf{z}^i$ Vector of independent coordinates. 26, 27, 57–60

$\mathcal{K}$ Kalman gain. 14–17, 28

$\mathcal{O}$ Matrix containing the observations of each sigma-point of the unscented Kalman filter. 17

$\tilde{\mathbf{y}}$ Vector of innovation. 15, 16, 28

$\mathbf{\Phi_q}$ Jacobian of the constraints vector with respect to the natural coordinates defining the constraints of closed loops. 23

$\mathbf{\Phi_z}$ Jacobian of the constraints vector with respect to the dependent coordinates. 23, 28, 29

$\mathbf{\Phi_z}$ Jacobian of the constraints vector with respect to the relative coordinates. 21, 22, 24

$\mathbf{\Phi}$ Vector of constraints. 21, 23, 24, 29

$\boldsymbol{\alpha}$ Angular acceleration of a body. 58, 59, 69

$\boldsymbol{\lambda}$ Vector of Lagrange multipliers. 21, 24

$\boldsymbol{\omega}$ Angular velocity of a body. 20, 22, 54, 55, 58, 59, 69

$\dot{\mathbf{s}}$ Vector which includes the translations of the body coordinates. 20

$\dot{\mathbf{z}}$ Vector of relative velocities. 20–24, 27, 28

$\hat{\dot{\mathbf{x}}}$ Estimation of the time derivative of the state. 14

$\hat{\dot{\mathbf{z}}}^i$ Estimated vector of independent velocities. 29

$\hat{\dot{\mathbf{z}}}$ Estimated vector of velocities. 29

$\hat{\mathbf{x}}$ Estimation of the state. 14–17, 27, 28

$\hat{\mathbf{z}}^i$ Estimated vector of independent coordinates. 28

$\hat{\mathbf{z}}$ Estimated vector of coordinates. 28

$\mathbf{Z}$ Body coordinates. 20–22

$\mathbf{f_x}$ Jacobian of transition function with respect to states. 27

$\mathbf{h_z}$ Jacobian of measurement equation with respect to independent coordinates. 57–60

$\mathbf{h}_{\ddot{\mathbf{z}}}$ Jacobian of measurement equation with respect to independent accelerations. 57–60

$\mathbf{h}_{\dot{\mathbf{z}}}$ Jacobian of measurement equation with respect to independent velocities. 57–60

$\mathbf{q}$ Vector of natural coordinates defining the constraints of closed loops. 22, 23

$\mathbf{x}$ State. 13–15, 26

$\mathbf{z}$ Vector of relative coordinates. 21, 23, 24, 26, 28

Chapter 1

Introduction

Simulation techniques are widely use in industrial applications for studying the different phenomena that affect a real system in a virtual environment. Simulation solutions are used during the life-cycle of the product to predict its performance, design a manufacturing process or to detect possible operational failures. Using a simulation framework allows to perform several tests over a product without building physical prototypes. Due to the advantage in terms of final product cost that it implies, simulation techniques are widely use in several fields of the industry. One of the most common techniques for simulating mechanisms and performing dynamics analysis is the multibody (MB) simulation. The use of the MB approach is extended in fields such as aerospace, machinery, robotics, biomechanics or automotive.

In the particular case of the automotive industry, MB dynamics have been of interest for several purposes. Through MB dynamics, different vehicle configurations can be accurately simulated in several maneuvers, identifying the optimal set of parameters which results in a great comfort and handling. From the simulation environment, a large amount of data can be extracted and analyzed. Also, if real-time execution is achieved, the behavior of a physical subsystem (such an active suspension) can be tested in a virtual integration context, which is known as Hardware-in-the-loop (HIL) or Human-in-the-loop (HITL), if a human user is included. The response of an element to different inputs from the vehicle can be tested and evaluated without installing it on a prototype, reducing the costs, development times and risks.

In addition, for driving assistance, different controllers are being implemented in commercial vehicles. These controllers need information on the states of the vehicle in order to take the correct actions. Some data can be obtained from sensors installed on the vehicle. However, there are measurements that are not possible to acquire because the required sensor is expensive or it is not available.

State estimation constitutes an attractive alternative for increasing the available measurements. A model of the vehicle, or part of it, is combined with an estimator and a reduced set of sensors to ensure that the simulation matches with the reality. The estimator uses the sensor measurements for correcting the possible drift of the simulation with respect to the real situation. Hence, the information that can be obtained from the simulation environment can be reliable. This approach is also known as virtual sensors, since the information of the real states of the vehicle is gathered from a virtual environment.

For virtual sensing, accurate models are required as starting point, for what MB models can be employed. However, MB models are costly from a computational point of view and have been replaced by analytical models for virtual sensing applications. Thus, implementing a state observer based on a MB model on real time and in the hardware that is on-board of vehicles is challenging. The target hardware is known as embedded hardware, and is quite less powerful than conventional personal computers (PCs). Nevertheless, new generation embedded hardware offers an increment on computational power. The automotive industry is starting to use this hardware to satisfy the computational power demanded by new applications, such as Advanced Driver Assistance Systems (ADASs), and the complexity of the control systems that come with the electrification of vehicles.

This thesis explores the opportunity that this new generation hardware offers to implement accurate virtual sensors based on MB models in automotive applications. First, after reviewing the state of the art on state estimation, an efficient filter is selected regarding its implementation and computational cost. Later, the design of the MB-based state observer is addressed and its implementation in the target hardware is studied, seeking for real-time execution. Finally, the solutions proposed are tested, and the results in terms of estimations and efficiency are analyzed.

1.1 Motivation

From some years ago, the automotive industry has been focused on improving the handling and safety of the vehicles. For that purpose, multiple control algorithms have been developed and implemented on-board. Also, during the test campaigns a lot of maneuvers are done in order to analyze the response of the vehicle. From the sensor data obtained, the parameters of the vehicle can be later modified in order to satisfy the handling and comfort requirements.

However, these applications are limited by the available sensor measurements. Installing sensors on the vehicle has a direct impact on the final cost of the vehicle. In addition, there are system variables that are difficult or costly to measure: no existing sensor, inaccessible location or inaccurate direct measurements. Combining a vehicle model with a reduced set of sensor data, additional measurements can be obtained. Thus, some real sensor can be replaced by virtual sensors, reducing the costs of the final vehicle while providing more information useful for control strategies or vehicle dynamics analysis.

The Laboratorio de Ingeniería Mecánica (LIM) of the University of A Coruña has specialized in real-time MB simulations, being vehicle dynamics one of its main applications. With this background, the LIM started a research field on the development of MB-based state observers. In [1], Cuadrado *et al.* applied an extended Kalman filter (EKF) to simple mechanisms. Then, the EKF is combined with a MB model of a vehicle [2]. Although the results were promising, the simulation was far from real time even on a conventional PC, thus limiting the use of this approach in the automotive industry.

Following the research made in the LIM, Pastorino completed his PhD. thesis [3], in which more state observers from the family of Kalman filters were considered based on the MB model of a vehicle prototype. Nevertheless, real-time performance was not achieved with none of the state observers evaluated.

After that, Sanjurjo started his PhD. thesis [4], where a new state observer based on a EKF for MB models was proposed. In his work, the filter presented shows a improvement in performance, in a way that the computational cost of the MB-based state observer was reduced, achieving real time in a PC. Also, the results in terms of estimation are consistent and reliable in the simulation environment.

To continue with the work made, this thesis deals with the implementation of a MB-based state observer in real time on an automotive platform. Thus, accurate virtual sensors could be implemented in automotive applications, such as vehicle control or vehicle test campaigns.

1.2 Objectives

The main objective of this thesis is to implement in real time a MB-based state observer on an automotive platform. For that purpose, the following partial objectives are set:

- To study the suitability of Field Programmable Gate Arrays (FPGAs) for accelerating MB simulations. FPGAs are hardware accelerators that are available in new generation automotive hardware and can be useful for achieving real-time performance. This thesis is focused on study how to take advantage of them for efficient MB simulations.

- To develop an accurate and efficient MB model-based observer of a vehicle in order to implement it in real time on automotive hardware, which has limited computational capabilities.

- An efficient code for executing MB simulations on embedded hardware is required. In [5], a library (`MBScoder`) for creating MB models for different platforms and programming language is presented. This thesis will continue the development of this library in order to add new functionalities useful for this work, such as new MB coordinates and formulations.

- To provide a framework for state estimation based on MB dynamics that could be easily implemented in real automotive systems.

Given the previous objectives, the main contributions of this work are outlined as follows:

- Guidelines to accelerate MB simulations with Field Programmable Gate Arrays (FPGAs) can be extracted from this thesis. The examples shown in this thesis can be used for future applications of FPGAs in MB dynamics.

- A new observer based on a Dual Kalman filter has been developed. It is referred during the text as state-parameter-input (SPI) observer. It shows improved accuracy in the estimations with respect to previous MB-based observers.

- The `MBScoder` is improved with a new module for modeling in relative coordinates based on a semi-recursive formulation.

- The MB-based observer developed achieves real-time performance on automotive hardware for in-vehicle applications. The entire framework has been adapted to the FMI 2.0 Standard, allowing an easy implementation in real systems.

1.3 Thesis structure

This thesis has been organized in 6 chapters:

Chapter 1 introduces the topic of this thesis and the motivation that gave rise to this thesis. After that, the objectives and contributions of this thesis are briefly described. Finally, the structure of the thesis is presented.

Chapter 2 makes a review on the state of the art of state estimation for automotive applications and on the evolution of the embedded hardware commonly installed in-vehicle, focusing on the new generation hardware.

Chapter 3 presents the filter chosen for the use-case of this thesis and the MB formulation employed in order to achieve the maximum efficiency in terms of computational cost.

Chapter 4 explains the hardware selected for implementing the MB-based state observer and the different strategies followed to achieve an efficient solution.

Chapter 5 presents the vehicle modeled with the details of the MB model developed. The simulations launched to test the behavior of the MB-based state observer are explained. The results in terms of estimations and computational cost are presented. The final implementation, based on a standardized interface is also included.

Chapter 6 consists on the conclusions extracted from the work of this thesis. The future lines of the research are drawn up.

Chapter 2

State of the art

This thesis deals with the implementation in real time of state observers based on MB models on embedded hardware for automotive applications. The success of achieving the objectives of this work relies on a proper selection of the hardware, together with an efficient state observer implementation.

In the following sections, an overview of the state observers commonly used in the automotive industry is presented. Later, the evolution of the in-vehicle hardware is commented, introducing the type of device that will be used during this thesis.

2.1 State observers on automotive industry

From several years ago, automotive industry has been implementing novel technologies on vehicles for improving the comfort and safety of the passengers [6]. For this purpose, the efforts are focused on the development of driving aids. They have been evolving from from the traditional Antilock Brake System (ABS) or electronic stability program (ESP), to advanced systems, such as the lane keeping aid, or the collision avoidance system.

All these driving aids require precise and timely information of the vehicle. While current vehicles are equipped with many sensors, there are still some magnitudes that cannot be measured directly due to technical or economical reasons. As an alternative, the estimation of those magnitudes, also known as virtual sensing, is proposed: models and a limited set of experimental data are fused to estimate system variables that are difficult (inaccessible location, inaccurate direct measurements, inexistent sensor) or costly to measure.

The virtual sensing approach can also be applied during the test campaigns of a vehicle. Virtual sensing enriches the measurement data-set, thus providing better insight into the system-under-test to engineers. Expensive sensors, such as the wheel force sensors, can be replaced reducing the overall costs.

To *install* virtual sensors on an auto-mobile, a model of the vehicle combined with a state observer has to be implemented on the vehicle computer units. In general, these units, also known as Electronic Control Units (ECUs), are responsible of the electronic control of different components of vehicles. An ECU processes inputs from sensors and drives the corresponding actuator.

ECUs are a type of embedded systems. Embedded systems can be defined as a specialized computing system that is optimized to carry out a single dedicated function. This pre-determined functionality contrasts with that of a General Purpose Processor (GPP), such as those presented in a PC, where the processor will perform a large number of very varied functions. Since an embedded system is application-specific, it can be finely optimized to deliver the chosen characteristics of a given application [7]. In embedded systems, the power consumption is critical and it has become a major aspect that limits the performance of processors for embedded applications [8]

ECUs were initially used for injection timing control of the engine, knock control, idling speed control, transmission control, and so on. At the same time, electronic control also spread quickly to functions such as the air conditioner, ABS and airbags [9]. Currently, ECUs are also responsible of executing the control of the active systems and also the ADASs, for what higher computational power is required. Due to this, new hardware alternatives for ECUs are being explored.

2.1.1 State observers based on analytical models

For the virtual sensing approach, real-time performance is a mandatory prerequisite for synchronizing real and virtual environments [10]. Due to the restricted computational power of the automotive hardware, simplified models were traditionally used for representing the main dynamics that are involved on the particular application. Thus, most of the research made in state estimation relies on analytical models, due to their simplicity and low computational cost. Analytical models usually neglect some dynamic effects, limiting the application of these approaches. For instance, in [11], an analytical model which considers only the longitudinal and vertical dynamics is used for controlling the brake distribution of a vehicle. The lateral dynamics, roll and yaw motion and wheel rotation dynamics are neglected. Therefore, the dynamics are not as precise as with a more complex model.

The interest of virtual sensors on the automotive industry is introduced by Venhovens and Naab in [12]. They developed an adaptive cruise control, a lane keeping support and they also estimated the yaw rate and tire slip angles. In these examples, analytical and simplified models were used. For the adaptive cruise control, two vehicles were modeled as two point masses, while for the other use cases, the car was modeled as a planar two-degrees-of-freedom system.

The use of the two-degrees-of-freedom model in the literature is extended. It is also known as bicycle model or one-track model. Its main functionality is to represent the lateral vehicle dynamics for estimating the slip angles of the tire.

The bicycle model is employed in [13] for estimating the side-slip angles, and comparing results between a linear and a non-linear observer. It is assumed that the two side-slip angles of one axle are nearly the same.

In [14], the bicycle model is used to consider only the lateral movement and yaw of the vehicle. A comparison of different estimation methods for the lateral dynamics is made, assuming that the transversal tire forces are linear if the lateral acceleration remains below 0.4 g.

In order to increase the accuracy and field of application of the bicycle model, different degree-of-freedoms (DOFs) are added in order to include the most relevant dynamics effects during the maneuvers of interest. Accordingly, in [15], the bicycle model is formulated including one DOF related with the longitudinal dynamics. Later, a Kalman filter is combined with the model for the on-line estimation of vehicle handling dynamics. Using a model of low-order and time-invariant ensures practical real time, retaining a useful level of accuracy.

The work presented in [16] shows the combination of a sliding mode observer [17] with a reduced-order vehicle model. In this case, the vehicle is modeled looking for a trade-off between accuracy and efficiency. Thus, five DOFs are considered in the vehicle model including the longitudinal, lateral and yaw motion of the chassis and the rotational motion of the two traction wheels. This approach assumes that the roll and pitch motion are not relevant in the vehicle load transfers.

In [18], the study of tire force estimation is addressed with a four DOF vehicle model. It only includes the longitudinal, lateral, roll and yaw motion, depreciating the effects of the rest of DOFs on the dynamics.

Freeman *et al.* estimates in [19] the speed, yaw angle and cornering stiffness through the combination of two different models and a Kalman filter. For the speed and yaw angle estimation, the vehicle model involves four DOFs: longitudinal, lateral and yaw motion and wheel rotations. For the cornering stiffness, a single track model is designed. The estimated variables are later used in a controller for preventing run-off-road situations.

In [20], a three DOF planar model is considered. Only longitudinal, lateral and yaw motions are involved, assuming that the suspension behave as rigid and neglecting its effects on vehicle dynamics. In order to achieve more accuracy, a kinematic model is later included. Both models are combined with a Kalman filter and the side-slip angle is estimated.

When modeling a vehicle, a large amount of parameters is involved. Most of those parameters, such as the geometry of the suspension system, remain constant during the life cycle of the vehicle. Meanwhile, the inertial properties of the vehicle, such as the mass, can vary depending on the number of passengers or luggage [21] and the tire-road coefficient can be affected by meteorological conditions or tire wear. Both parameters play an important role on vehicle dynamics, affecting the acceleration, braking, handling and comfort [22]. Combining state and parameter estimation can improve the quality of the estimations. As an example, in [23], the mass of the vehicle is estimated through a simple quarter-car linear model for each suspension, trying to reduce the computational cost.

Some parameters can only be estimated under certain driving conditions. Maleej *et al.* propose in [24] an event-based estimator for the mass and the road grade. It is focused on longitudinal acceleration maneuvers and, therefore, the model only includes the longitudinal dynamics. The estimation is only performed when the vehicle is accelerating. If the vehicle is in cruise motion, deceleration or stop states, the estimation is disabled.

In [22], a two DOF model is used in order to consider the lateral behavior of the vehicle. It shows that updating the mass to the estimated value improves the states estimation. Nevertheless, the quality of the mass estimation depends on the maneuver. While on long straight line maneuvers the estimation is poor, during turning maneuvers the estimator works with acceptable accuracy.

It is also of interest the work presented in [25], where a full vehicle and tire identification is presented. Based on an analytical full vehicle model, three filtering methods are tested: EKF, unscented Kalman filter (UKF) and a particles filter. While the particles filter is very slow to converge, the Kalman filters were found to be effective. In automotive applications, where the model linearization is complex, the UKF is best suited for parameter estimation.

Finally, the state and parameter estimation is combined in what is known as dual Kalman filters. Wenzel *et al.* in [21], present a dual extended Kalman filter based on a vehicle model with four DOFs, considering only the motion in the longitudinal and lateral direction and the yaw and roll angles. Then, two Kalman filters are employed in parallel for estimating the states and parameters of the vehicle. This duality allows to reduce the uncertainty of the model, improving the performance of the observer.

In a later work [26], Wenzel *et al.*, the vehicle model is improved. It considers the vehicle as one body with five degrees of freedom (longitudinal, lateral, roll, pitch and yaw motion) and one degree of freedom for each wheel. Compared with a MB model, the approach selected is more efficient.

In [27], a dual EKF for vehicle state and road friction coefficient estimation is presented. It is combined with a three DOF vehicle model, including the essential dynamic properties. Two EKFs operate in parallel, increasing the accuracy. This work shows the importance of estimating the road coefficient.

Boada *et al.* present in [28] and [29] two dual filters for automotive applications in order to estimate vehicle parameters and states. In [28], a vehicle roll model with three DOFs (side-slip angle, yaw rate and roll angle) is presented. The estimation process starts with the parameter estimation, to continue with the estimation and correction of the states, ending with the parameter correction. The dual filter proposed is later tested in a real vehicle, showing effectiveness and accuracy in the estimations. It is also remarked that the results can improve with non-linear tire models and suspension systems.

Later, in [29], Boada *et al.* present a dual Kalman filter for estimating road irregularities and vehicle mass. In this work, a full vehicle model with seven DOFs is employed, allowing either dynamic and static driving conditions. This increases the amount of conditions where the mass can be properly estimated. Following the same sequence of [28], the states and parameters are estimated and corrected. Final experiments on a real vehicle shows a high accuracy in the estimations. However, a controller should be designed in order to switch off the mass estimation at high accelerations, where the algorithm does not work properly.

Most of the research made on vehicle state estimation is based on analytical models considering the DOFs which have a notorious influence on the variables that are estimated. The simplifications are made in order to minimize the computational cost of the simulation. However, since some dynamic effects are neglected, the application of these approaches is limited.

2.1.2 State observers based on multibody models

As an alternative to analytical models, MB models can represent with fidelity the dynamic behavior of a vehicle in any situation. Thanks to the high level of detail, using MB models enriches de measurement data-set: more information can be extracted compared with analytical models.

A MB model can be defined as an assembly of two or more bodies imperfectly joined together, having the possibility of relative movement between them [30]. In order to describe a MB system, a set of parameters or coordinates must be selected for defining unequivocally the position, velocity, and acceleration of the MB system [30]. Depending on the set of coordinates, MB formulations are developed in order to derive the equations of motion, generally leading to a system of differential-algebraic equations (DAEs). The efficiency of the MB solution for a particular application depends on the coordinates and formulation selected. Thus, the modeling phase should be carefully made.

In the automotive industry, MB simulation is widely used in the designing phase of vehicles. Vehicle dynamics and durability analysis benefit from accurate MB models. Advanced MB models are also required in HIL or HITL applications, which are useful for testing and experimentally optimizing subsystem behavior in a virtual system integration context, which may even include the human user.

The applications of MB dynamics can be also extended to state estimation. However, the combination of MB models with a state observer is not trivial. While the state observers are usually formulated for first order, unconstrained and linear problems, the equations of a MB model are second order, usually constrained and non linear [4].

The implementation of state observers based on MB models starts in [1]. This firs approach implements a continuous extended Kalman filter (CEKF) applied to a four-bar linkage with different MB formulations, achieving successful results. Later, in [2], the previous work is applied to a MB model of a vehicle. Although the method achieves good results in terms of estimation, it was far from real-time performance.

In [31], the research on the implementation of non-linear Kalman filters based on MB models continues. The aim of this work is to compare different filters in terms of accuracy and computational cost. The EKF in its continuous form and some sigma-point Kalman filters (SPKFs) (UKF, spherical simplex unscented Kalman filter (SSUKF)) are implemented. It shows that the SPKFs offer more accuracy and are simpler to implement, while the computational cost is higher.

In most estimation problems, it is assumed that the inputs of the system are known. This is not true in the case of vehicle applications. In [32], the unknown inputs are taken into account in a MB-based Kalman filter. For considering the real-time problem, the methodology proposed uses reduced MB models. In this work, it is concluded that there are convergence issues if there are no position sensors as a GPS on the chassis.

In [33], it is presented a nonlinear state observer based on kinematic models. Using kinematic models, the input forces are no needed. This leads to a considerable reduction of the model uncertainties. For estimating forces, in [34], the kinematic filter is used combined with a force observer, splitting the estimation phase in two steps.

Regarding the computational cost of state estimation based on MB models, the work of Sanjurjo *et al.* should be remarked. In [35], two new state observers based on MB models are presented: the error-state extended Kalman filter (errorEKF) and the projectionEKF, a method where a projection technique is employed to satisfy the constraints of the MB model. Both filters are evaluated in terms of accuracy and efficiency. Based on a planar mechanism, the tests performed showed that the errorEKF presents high accuracy while keeping low computational cost. This work continues in [36], where the errorEKF is compared more in detail against conventional filters such as the CEKF, discrete extended Kalman filter (DEKF) and UKF.

Later, In [4], the errorEKF is applied on a MB model of the vehicle presented in [37]. In this work, real-time performance on a PC is achieved with promising results. However, the computational power of a PC is higher than the power of the embedded hardware usually employed for in-vehicle applications.

The work on the errorEKF continues in [38], where the filter presented in [36] is extended with force estimation. As in [36], the test are based on a planar mechanism. The sensors included measure positions, velocities and accelerations. Different sets of sensors and sampling frequencies are evaluated. The results show that the new version of the errorEKF outperforms in terms of accuracy the previous version. Furthermore, it allows to estimate the input forces. Although the computational time increases, it is still faster than other filters of similar accuracy as the UKF.

The force estimation is also addressed in [39]. A flexible MB model of a suspension is used combined with strain gauges in the knuckle. From the reaction forces estimated in the knuckle, the tire forces are derived. However, the computational cost of flexible MB models is an obstacle for real-time applications on embedded hardware. The linearization of the system is proposed in order to increase the efficiency of the simulation.

With respect to parameter estimation based on MB models, it is briefly addressed in [40] through an acceleration maneuver. Although the mass is accurately estimated, the solution is far from achieving real-time performance.

2.2 On-board Implementation

The state observers presented in the previous section are developed in order to provide virtual sensor information on-board a vehicle. The hardware that is available on commercial vehicles usually has low computational power. Therefore, most of the approaches presented are based on simple vehicle models, in a trade-off between accuracy and efficiency. However, computation capabilities of embedded processors have been growing exponentially due to the sustained innovation in processors technology.

2.2.1 In-vehicle ECUs

Electronic systems have been present in vehicles for a while, improving the behavior of mechanical components. In the early days of automotive electronics, each new function was implemented as an stand-alone ECU [6]. Therefore, modern vehicles have a large number of ECUs for performing a large amount of functions. In top-of-the-range vehicles, the number of ECUs is close to 100, due to the introduction of new applications such as ADAS or active safety functions. Furthermore, the incoming electric vehicle requires much more advanced computation to manage the automotive systems, including the electrical powertrain [41].

Current ECUs are a type of embedded hardware that is composed of a Micro Computer Unit (MCU) along with sensors, actuators and other chips and components that are used to build the hardware [42]. The MCUs are the most importan part. They are designed to comply with the strictest timing, reliability and safety requirements imposed by automotive standards [43]. However, MCUs have low clock speeds and platforms based on these devices cannot satisfy the high performance requirements of the newest automotive functionalities.

In order to meet the high computational performance and low-power requirements, new generation ECUs are based on advanced multi-core architecture. Having more than one core in the system allows developing parallel processing on chips, reducing the computational cost of an application [44].

In a multiprocessor platform, however, there is not an ideal core for all the possible applications. Normally, in conventional processors, this issue is solved by designing a general purpose architecture. However, few applications actually need all those resources. Therefore, such an architecture is highly over-provisioned for any single application [45], leading to a higher power consumption.

Homogeneous designs assembly multiple cores of the same architecture and, in order to cover a high range of applications, the over-provisioned problem of conventional processors is repeated [45]. Conversely, heterogeneous processors can map each application to the best suited core to meet its performance demands.

In general, heterogeneous processors complement a main core with different type of secondary processors or co-processors. The main purpose of the co-processor is to supplement the functionality of the primary processor, and it is optimized for a single specific task. By offloading computations from the main processor to one or more co-processing units, the overall system performance can be accelerated [7]. Thus, having heterogeneous processor cores provides potentially greater power savings without dramatic losses in performance [46]. Due to the potential reduced energy consumption offered by heterogeneous multi-core platforms, their usage is appealing for hard real-time systems in embedded applications [8].

In the automotive industry, it is common to find heterogeneous processors where the main processor is combined with Graphic Processing Units (GPUs) or Field Programmable Gate Arrays (FPGAs). Their are mainly employed in computer vision applications for ADAS, as presented in [43, 47–49].

This new trend gives an opportunity for simulating complex models, as MB, on automotive embedded hardware in real time. Virtual sensing applications could be based on detailed models, instead of reduced sized models, while still maintaining real-time performance.

The use of GPUs in MB dynamics has been addressed in [50–53]. The size of the problems were adequate to exploit the advantages of GPUs. However, the problem addressed in this thesis has a reduced size and it does not map with the field of application of GPUs.

With respect to FPGAs, there is no previous research on how to use them for accelerating MB simulations. Since the list of processors including FPGAs is increasing, it is important to learn how to take advantage of them. Thus, the work of this thesis will be based on heterogeneous processors with an FPGA as co-processor.

To summarize, the automotive industry is demanding more computational power on their embedded platforms. With the introduction of heterogeneous processors, the capabilities of automotive hardware is increased. Due to the reduced computational power of traditional ECUs, virtual sensing approaches were mainly based on analytical models. With the new generation embedded hardware, state observers based on MB models could be implemented in real time. This approach would increase the virtual measurements and their quality.

Chapter 3

State observers based on Multibody Models

This chapter focuses on the explanation of the Kalman filter that is employed in this thesis and the MB formulation that is implemented.

First, a brief overview on the existent Kalman filters is performed, selecting the best filter for the application addressed in this thesis. Later, the MB coordinates and formulation employed is discussed. Finally, the integration of the MB formulation in the Kalman filter is addressed.

3.1 Kalman filter review

In Chapter 2, several state observers developed for automotive applications were presented. In most of the works reviewed, Kalman filtering was the option selected for estimating variables of the vehicle. However, there are also alternatives to the Kalman filter, as the particles filter or the sliding mode observer.

The particles filter is suitable for systems that are highly non-linear (as the case of MB models). However, it leads to a huge computational cost [54]. As explained in Chapter 2, the computational power of embedded platforms in automotive applications is restricted. Thus, particles filters are discarded.

The sliding mode observer [17] is an estimator characterized by its robustness against parameter variations. It is adequate to be implemented with systems of low order [55], where the model is simplified. As stated in Chapter 2, using simple models limits the application of the filter to specific maneuvers.

Kalman filters perform the estimations through the propagation of the mean and covariance of the desired variables through the time [54]. This propagation is conditioned on the available sensor measurements. It was developed initially for linear systems where the state variables are independent. To derive the equations, a generic linear model is defined,

$$\mathbf{x} = f(\mathbf{x}, \mathbf{u}, \mathbf{w}, t) \tag{3.1}$$

$$\mathbf{y} = h(\mathbf{x}, \mathbf{v}, t) \tag{3.2}$$

$$\mathbf{w} \sim N(0, \mathbf{\Sigma}_c^P) \tag{3.3}$$

$$\mathbf{v} \sim N(0, \mathbf{\Sigma}^S) \tag{3.4}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ are the inputs to the model, $\mathbf{y}$ is the vector containing the outputs of the model, $\mathbf{w}$ is the process noise with covariance $\mathbf{\Sigma}_c^P$, $\mathbf{v}$ is the noise from the measurements with covariance $\mathbf{\Sigma}^S$, $f(\cdot)$ is the system equation and $h(\cdot)$ the equation for obtaining the measurements predicted by the model.

In order to obtain the estimation of the states through the time, the next differential equation is employed,

$$\dot{\hat{\mathbf{x}}} = \mathbf{F}\hat{\mathbf{x}}(t) + \mathcal{K}(t)\left[\mathbf{o}(t) - \hat{\mathbf{o}}(t)\right] \tag{3.5}$$

where $\mathbf{F}$ is the transition matrix representing the evolution of the model, $\mathcal{K}$ is the Kalman gain matrix which resembles to a weighting matrix for the measurements, $\mathbf{o}$ and $\hat{\mathbf{o}}$ are the real and predicted measurements respectively.

The expressions for the Kalman gain and the predicted measurements are,

$$\hat{\mathbf{o}}(t) = \mathbf{H}\hat{\mathbf{x}}(t) \tag{3.6}$$

$$\mathcal{K}(t) = \mathbf{P}(t)\mathbf{H}(t)^{\top}(t)\mathbf{\Sigma}^{S^{-1}}(t) \tag{3.7}$$

where $\mathbf{H}$ represents the evolution of the measurements with the time, and $\mathbf{P}$ is the covariance matrix of the state estimation uncertainty, obtained as,

$$\mathbf{P}(t) = \mathbf{F}(t)\mathbf{P}(t)\mathbf{F}(t)^{\top} + \mathbf{\Sigma}^P(t) \tag{3.8}$$

As the Kalman filter is developed for linear systems with independent variables, its application to MB models is not trivial. MB models are usually non-linear and defined with dependent variables, as explained in Section 3.2. However, there are variations of the Kalman filter designed for non-linear systems. Each version presents different features regarding accuracy, efficiency and implementation.

One of the most popular is the extended Kalman filter (EKF), where the non-linearities are solved by using a Jacobian matrix to propagate the mean and the covariance of the state vector [54]. Despite of its compatibility with non-linear systems, there are additional issues when combining the EKF with a MB model. First, the EKF expects first-order systems, while the equations of motion of a MB model are second order. This issue is overcome by including in the state vector the positions and velocities of the MB model, at the cost of duplicating the problem size. Second, the EKF applies to independent states, whereas most MB formulations are set in dependent coordinates related by constraint equations. Thus, tailored MB formulations must be developed in order to solve the dynamic equations in an equivalent set of independent coordinates [2]. Several Kalman filters are based on the EKF due to its efficiency and accuracy.

The continuous extended Kalman filter (CEKF) is formulated in a continuous-time form. With respect to the MB equations, this option seems appropriate, as they are also expressed in the form of continuous-time differential equations [35]. As a counterpart, simulations are normally performed in discrete time steps. Therefore, the equations of the CEKF must be modified.

The discrete extended Kalman filter (DEKF) is a version of the CEKF that operates in discrete time steps. This situation is often encountered in practice, since the dynamics of the systems are usually solved in discrete time steps, and the measurements from the sensors are also discrete in time.

In the DEKF, the estimation is divided in two stages: the prediction and the correction. During the prediction phase, the value of the state vector is estimated by solving the dynamics of the model. In this step, the measurements of the sensors, $\mathbf{o}_k$, are not taken into account. The estimated state vector before the measurements are considered is referred as the *a priori* estimate, denoted by $\hat{\mathbf{x}}_k^-$. Later, when the measurements at time step k are available, the correction stage is performed and the *a posteriori* estimation, $\hat{\mathbf{x}}_k^+$, is obtained. Of course, $\hat{\mathbf{x}}_k^+$ is expected to be a better estimate than $\hat{\mathbf{x}}_k^-$, since there is more information of the system to compute $\hat{\mathbf{x}}_k^+$ [54].

In the DEKF, the system model for a time step k is a non-linear system represented as [54],

$$\begin{aligned} \mathbf{x}_k &= f_{k-1}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}) && (3.9)\\ \mathbf{y}_k &= h_k(\mathbf{x}_k, \mathbf{v}_k) && (3.10)\\ \mathbf{w}_k &\sim N(0, \mathbf{\Sigma}^P_{c\ k}) && (3.11)\\ \mathbf{v}_k &\sim N(0, \mathbf{\Sigma}^S{}_k) && (3.12) \end{aligned}$$

which are similar to Equations 3.1-3.4, but defined for discrete time steps, and with non-linearities presented in the state equations, $f(\cdot)$, and measurement equations, $h(\cdot)$. The equations for propagating the mean and the covariance through the states are,

$$\begin{aligned} \mathbf{P}_k^- &= \mathbf{F}_{k-1}\mathbf{P}_{k-1}^+\mathbf{F}_{k-1}^\top + \mathbf{\Sigma}^P && (3.13)\\ \hat{\mathbf{x}}_k^- &= \mathbf{F}\hat{\mathbf{x}}_{k-1}^+ + \mathbf{G}\mathbf{u}_{k-1} && (3.14) \end{aligned}$$

where $\mathbf{F}$ results from the linearization of the model equations (equivalent to the Jacobian matrix), and $\mathbf{G}$ is the input matrix, representing the evolution of the system with the inputs, $\mathbf{u}$.

From this equations, the *a priori* estimate, $\hat{\mathbf{x}}_k^-$, is obtained and the correction stage can be executed. First, the mismatch $\tilde{\mathbf{y}}_k$ (innovation) between the sensor measurements and the model estimations is calculated,

$$\tilde{\mathbf{y}}_k = \mathbf{o}_k - \mathbf{H}\hat{\mathbf{x}}_k^- \qquad (3.15)$$

where $\mathbf{o}$ (observation) represent the sensors measurements and $\mathbf{H}\hat{\mathbf{x}}_k^-$ are the estimated measurements from the model according to the predicted states. For non-linear systems, the sensitivity matrix of the measurements, $\mathbf{H}$, is derived from the partial derivative of $h_k(\mathbf{x}_k, \mathbf{v}_k)$ with respect to the state evaluated on $\hat{\mathbf{x}}_k^-$. It relates the variation of the measurements with the state.

Later, for calculating the *a posteriori* values of the state vector, the expression for the Kalman gain, $\mathcal{K}$, is derived,

$$\begin{aligned} \mathbf{\Sigma}_k &= \mathbf{H}\mathbf{P}_k^-\mathbf{H}^\top + \mathbf{\Sigma}^S{}_k && (3.16)\\ \mathcal{K}_k &= \mathbf{P}_k^-\mathbf{H}^\top\mathbf{\Sigma}_k^{-1} && (3.17) \end{aligned}$$

where $\mathbf{\Sigma}_k$ is the innovation covariance matrix which represents the uncertainty in the system state projected through $\mathbf{H}$ plus the covariance matrix of the noise originated in the sensor itself, $\mathbf{\Sigma}^S$ [4].

Finally, the estimation of the state, $\hat{\mathbf{x}}$, and the covariance matrix of the estimation error, $\mathbf{P}$, are,

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathcal{K}_k \tilde{\mathbf{y}}_k \tag{3.18}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathcal{K}_k \mathbf{H})\mathbf{P}_k^- \tag{3.19}$$

Although the EKF is widely applied for non-linear systems, it can be difficult to tune. Furthermore, it often provides unreliable estimates if the non-linearities are severe, due to the implicit simplifications of the linearization process [54]. As an alternative, there is another approach for non-linear applications: the sigma-point Kalman filters (SPKFs). In this kind of filters, a set of deterministically chosen weighted sample points, also know as sigma-points, is propagated through the non-linear system functions [31]. One popular version of the SPKFs is the unscented Kalman filter (UKF), which can provide significant improvement in accuracy over the EKF [54].

The first step in the UKF is to calculate the set of sigma-points, which are a total of $n_{sp} = 2L + 1$, being L the dimension of the state vector. The sigma-points are obtained following Equation 3.20.

$$\chi_k(i) = \begin{cases} \hat{\mathbf{x}}_k & i = 0 \\ \hat{\mathbf{x}}_k + \gamma(\sqrt{\mathbf{P}_k})_i & i = 1, ..., L \\ \hat{\mathbf{x}}_k - \gamma(\sqrt{\mathbf{P}_k})_i & i = L+1, ..., 2L \end{cases} \tag{3.20}$$

where $\chi_k(i)$ is the $i - th$ sigma-point, $\gamma = \sqrt{L + \lambda}$, $\lambda = \alpha^2(L + \kappa)$, α and κ are user-defined tuning parameters, $\sqrt{\cdot}$ is the matrix square-root using lower triangular Cholesky decomposition and $(\cdot)_i$ represents the $i - th$ column [31].

The *a priori* state estimations and the covariance matrix for a time step k can be derived from the weighted mean and covariances of the sigma-points, as is expressed in Equations 3.21 and 3.22.

$$\hat{\mathbf{x}}_k^- = \sum_{i=0}^{n_{sp}-1} \mathbf{w}_i^m \chi_k^-(i) \tag{3.21}$$

$$\mathbf{P}_k^- = \sum_{i=0}^{n_{sp}-1} \mathbf{w}_i^c (\chi_k^-(i) - \hat{\mathbf{x}}_k^-)(\chi_k^-(i) - \hat{\mathbf{x}}_k^-)^\top \tag{3.22}$$

being $\mathbf{w}_0^m = \lambda/(L + \lambda)$ and $\mathbf{w}_0^c = \mathbf{w}_0^m + (1 - \alpha^2 + \beta)$, $\mathbf{w}_i^c = \mathbf{w}_i^m = 1/[2(L + \lambda)]$ for $i = 1, ..., n_{sp} - 1$ and β a scaling factor used to control the weighting of the zeroth sigma-point.

For obtaining the Kalman gain in order to estimate the *a posteriori* state vector, the sigma-points are propagated executing as many MB simulations as sigma-points for the time step k. The observations (i.e simulated sensor measurements) of each simulation are weighted and added as stated in Equation 3.23 to obtain the Kalman gain.

$$\mathcal{K}_k = \mathbf{P}_{x_k \mathbf{o}_k} \mathbf{P}_{\mathbf{o}_k \mathbf{o}_k}^{-1} \tag{3.23}$$

$$\mathbf{P}_{x_k\mathbf{o}_k} = \sum_{i=0}^{n_{sp}-1} \mathbf{w}_i^c(\chi_k^-(i) - \hat{\mathbf{x}}_k^-)(\mathcal{O}_k^-(i) - \hat{\mathbf{o}}_k^-)^\top \tag{3.24}$$

$$\mathbf{P}_{\mathbf{o}_k\mathbf{o}_k} = \sum_{i=0}^{n_{sp}-1} \mathbf{w}_i^c(\mathcal{O}_k^-(i) - \hat{\mathbf{o}}_k^-)(\mathcal{O}_k^-(i) - \hat{\mathbf{o}}_k^-)^\top + \mathbf{\Sigma}^S{}_k \tag{3.25}$$

where $\mathcal{O}_k^-(i)$ are the observations of the $i-th$ sigma-point, given through the measurement model matrix, $\mathbf{h}_k$, and $\hat{\mathbf{o}}_k^-$ are the predicted observations obtained by the weighted means of the sigma-points measurements, as presented in Equations 3.26 and 3.27 [31].

$$\mathcal{O}_k^-(i) = \mathbf{h}_k(\chi_k^-(i)) \tag{3.26}$$

$$\hat{\mathbf{o}}_k^- = \sum_{i=0}^{n_{sp}-1} \mathbf{w}_i^c \mathcal{O}_k^-(i) \tag{3.27}$$

After obtaining the Kalman gain, it is possible to calculate through Equations 3.28 and 3.29, the *a posteriori* value of the covariance matrix and the state vector for the time step k.

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathcal{K}_k(\mathbf{o}_k - \hat{\mathbf{o}}_k^-) \tag{3.28}$$

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathcal{K}_k \mathbf{P}_{\mathbf{o}_k\mathbf{o}_k} \mathcal{K}_k^\top \tag{3.29}$$

It must be remarked that the MB equations are solved for each sigma-point without being modified, simplifying the implementation of the filter. The counterpart, as commented in Chapter 2, is the additional computational cost derived from the evaluation of the MB system for each sigma-point [31]. Thus, the use of the UKF in real-time applications on embedded hardware is not recommended if the size of the state vector is high.

In [36], an error-state Kalman filter (also known as indirect Kalman filter) is presented. This filter, called errorEKF, performs the estimations based on the errors in the state variables, understanding by errors the difference between real and calculated values. This filter combines the advantages of the EKF and UKF: the MB model can be considered as an independent system similarly to the UKF, while keeping the efficiency of the EKF.

The performance of the errorEKF is tested by Sanjurjo *et al.* in [36]. Based on a four-bar and a five-bar linkage mechanism, traditional Kalman filters are compared with the errorEKF. Interesting conclusions can be extracted from the results. It shows that the efficiency of the filters is related with the MB formulation and integrator. In the UKF simulation, shifting the integrator from the trapezoidal rule to a forward Euler duplicates the performance. However, the accuracy of the estimations decreases. In terms of computational cost, the errorEKF is the most efficient approach [36].

To conclude, the errorEKF is more efficient than conventional Kalman filters. It also can be implemented with any MB system without additional development cost. Thus, in this work, this filter is selected for in-vehicle estimation based on MB models. The integration of the errorEKF and MB equations is explained in detail in Section 3.3.

3.2 Multibody modeling

As explained in 3.1, the most efficient filters require to develop a tailored MB formulation. This can result in a tedious process and the resulting combination can penalize the computational cost. On the other hand, classical filters where the MB model can be treated as a black box have a low efficiency. However, the errorEKF allows to combine the Kalman filter with any MB formulation in an efficient form. This means that the selection of the formulation can be addressed regarding only its efficiency.

3.2.1 Coordinates and multibody formulation

In order to face the real-time challenges, the MB community has been made a huge work on developing efficient formulations for the dynamics of MB systems. In order to perform an efficient calculation, different factors should be considered: modeling, coordinates selection, formulation of the equations of motion, numerical integration, and implementation [30, 56].

The first dilemma when selecting the coordinates for modeling a system is the problem of either adopting independent coordinates, whose number coincides with the number of DOFs and is thereby minimal (reducing the computational cost), or adopting an expanded system of dependent coordinates. Most of the methods are formulated for dependent coordinates, which can describe the system much more easily than independent coordinates. However, as they are not independent, they must be related with constraint equations [30]. Depending on the selected set of dependent coordinates, there are different procedures for defining each body of the system and its corresponding constraint equations. The coordinates should be selected regarding the topology of the mechanism to be modeled. A proper coordinate selection reduces the complexity, leading to simpler constraint equations, and higher performance. In order to address the problem of the coordinates selection, the methods developed can be grouped into two big families: global and topological [57].

Global methods are based on coordinates that define the system in an absolute form. These methods offer a systematic calculation independently from the mechanism topology. Hence, global methods are simple and easy to implement. As a counterpart, they are not very efficient due to the high number of variables and constraints equations that must be imposed [57]. A typical set of coordinates used in global methods are the natural coordinates, presented in [30].

Topological methods make use of relative coordinates, defining each body with respect to the previous one in the kinematic chain and leading to a system with a minimum number of dependent coordinates [30]. This type of model definition invites to produce algorithms in which the kinematic as well as the dynamic terms are calculated by means of efficient recursive procedures [57]. Nevertheless, these kind of formulations are complex to implement, because the absolute position of an element depends on the position of the previous elements in the kinematic chain. Furthermore, they lead to equations of motion with matrices that, although small, are expensive to evaluate. They also require a post-processing work to determine the absolute motion of each point and element [30].

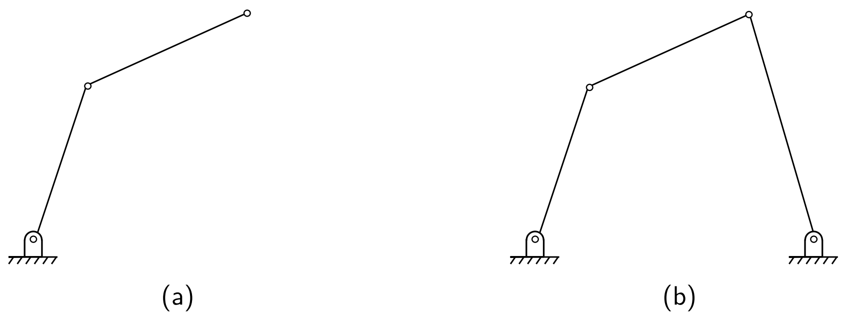


Figure 3.1: Example of open-chain mechanism (3.1a) and closed-chain mechanism (3.1b).

If the system to be modeled has a topology of an open-chain mechanism (Figure 3.1a), the relative coordinates will be independent and hence, the topological method will lead to a system with a minimum number of variables. However, when closed branches (or loops) are presented (Figure 3.1b), the coordinates are dependent and thus, constraint equations must be defined [30].

In order to generate the constraint equations, the closed loops should be opened at some point of the chain, resulting in an open-chain system. The constraint equations ensure that the equivalent open-chain mechanism behaves as the original.

As an example, in Figure 3.2, a double-wishbone suspension is presented. From Figure 3.2a, it can be seen that there are two closed loops: one is formed by the spring (B_4-B_5) and the low-control arm (B_1); the other is formed by the knuckle (B_3) and the low and upper control arms (B_1-B_2). In order to open the loop, the suspension model is cut by two joints (Figure 3.2b): the spherical joint J_4, and the revolute joint J_7. In this case, the constraint equations are based on the kinematic relations associated to the joint. Thus, the set of constraint equations depends on the selected joints. It is recommended to select the joints which lead to simpler constraint equations [58]. In the example of Figure 3.2, the constraints will impose that the position of the spherical and revolute joint, added to the direction of the rotation axis of the revolute joint, must match from both sides of the equivalent open-chain mechanism.

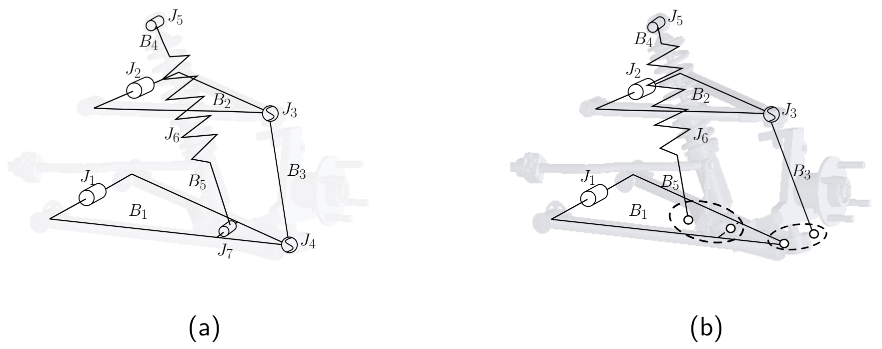


Figure 3.2: Illustrative example of the opening-chain procedure. A closed-loop mechanism (3.2a) is opened by two joints resulting in an open-chain mechanism (3.2b).

In [56–58], a topological method is presented and compared against a global method based on natural coordinates. The results presented show that the topological method offers high performance when large models are used, due to the reduction in the number of modeling coordinates. In small systems, like a suspension model, there is no advantage in using the semi-recursive formulation. When applied to a full vehicle model, the method in relative coordinates is 10 times faster than the absolute.

Accordingly to the previous explanation, the formulation presented in [56–58] is selected for the work of this thesis. In addition, the method is highly robust, which is important for ensure the convergence of the simulation after applying the corrections of the observer.

3.2.1.1 Semi-Recursive Formulation

The topological method employed in this thesis is known as semi-recursive. The set of relative coordinates used for modeling the mechanism are complemented with the so-called body coordinates, presented in Equation 3.30.

$$\mathbf{Z} = \begin{bmatrix} \dot{\mathbf{s}} \\ \boldsymbol{\omega} \end{bmatrix} \tag{3.30}$$

where $\dot{\mathbf{s}}$ is the velocity of the point of the body which is coincident with the fixed frame origin in a particular time step, thus representing the translations of the body, and $\boldsymbol{\omega}$ is the angular velocity of the body, which represents the rotations of the body.

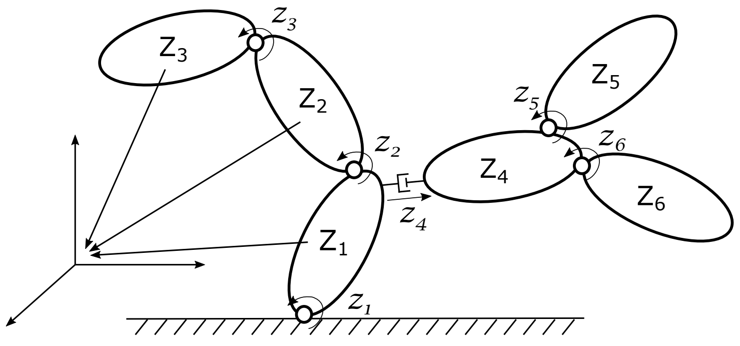


Figure 3.3: Generic mechanism which illustrates the set of coordinates that are employed in the semi-recursive formulation. The capital Z represent the body coordinates, while the lower case letter z is referred to the relative coordinates.

Applying the kinematic relations of two neighbor bodies, denoted by *i-1* and *i*, a general expression for obtaining the value of each body coordinate and its derivative in a recursive form can be derived,

$$\mathbf{Z}_i = \mathbf{Z}_{i-1} + \mathbf{b}_i \dot{\mathbf{z}}_i \tag{3.31}$$

$$\dot{\mathbf{Z}}_i = \dot{\mathbf{Z}}_{i-1} + \mathbf{b}_i \ddot{\mathbf{z}}_i + \mathbf{d}_i \tag{3.32}$$

where the form of the terms $\mathbf{b}_i$ and $\mathbf{d}_i$ depends on the type of joint that connects the bodies *i-1* and *i*. In [58], expressions of $\mathbf{b}_i$ and $\mathbf{d}_i$ are presented for different type of joints.

The dynamic equations which describe the motion of the mechanism are expressed initially in relative coordinates. In this method, the equations are stated according to the index-3 augmented Lagrangian formulation [59] in the form,

$$\mathbf{M}\ddot{\mathbf{z}} + \boldsymbol{\Phi}_{\mathbf{z}}{}^{t}\boldsymbol{\alpha}\boldsymbol{\Phi} + \boldsymbol{\Phi}_{\mathbf{z}}{}^{t}\boldsymbol{\lambda}^{*} = \mathbf{Q} \tag{3.33}$$

where $\mathbf{z}$ are the relative coordinates, $\mathbf{M}$ is the mass matrix of the mechanism expressed in terms of the relative coordinates, $\boldsymbol{\Phi}$ is the constraint vector due to the closure conditions of the loops, $\boldsymbol{\Phi}_{\mathbf{z}}$ is the Jacobian matrix of the constraints, $\boldsymbol{\alpha}$ is the penalty factor, $\mathbf{Q}$ is the vector of applied and velocity-dependent forces, and $\boldsymbol{\lambda}^{*}$ is the vector of Lagrange multipliers obtained from the following iteration process [57],

$$\boldsymbol{\lambda}^{*}_{i+1} = \boldsymbol{\lambda}^{*}_{i} + \boldsymbol{\alpha}\boldsymbol{\Phi}_{i+1}, \qquad i = 0, 1, 2... \tag{3.34}$$

However, if the relative coordinates, $\mathbf{z}$, are used, the calculation of the dynamic terms presented in Equation 3.33 is complex. Here is where the body coordinates (Equation 3.30) become relevant, since they lead to simpler expressions for obtaining the dynamic terms, as shown in [56–58]. To introduce these terms in the dynamic equations, relative and body coordinates can be related in such a way that,

$$\mathbf{Z} = \mathbf{R}\dot{\mathbf{z}} \tag{3.35}$$

$$\dot{\mathbf{Z}} = \mathbf{R}\ddot{\mathbf{z}} + \dot{\mathbf{R}}\dot{\mathbf{z}} \tag{3.36}$$

being $\mathbf{R}$ a matrix which depends on the topology of the mechanism and on the $\mathbf{b}_i$ terms from Equation 3.35, and $\dot{\mathbf{R}}\dot{\mathbf{z}}$ a vector obtained from the $\mathbf{d}_i$ terms of Equation 3.32. For the example of Figure 3.3, both matrices present the following form [56],

$$\mathbf{R} = \begin{bmatrix} \mathbf{b_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{b_1} & \mathbf{b_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{b_1} & \mathbf{b_2} & \mathbf{b_3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{b_1} & \mathbf{0} & \mathbf{0} & \mathbf{b_4} & \mathbf{0} & \mathbf{0} \\ \mathbf{b_1} & \mathbf{0} & \mathbf{0} & \mathbf{b_4} & \mathbf{b_5} & \mathbf{0} \\ \mathbf{b_1} & \mathbf{0} & \mathbf{0} & \mathbf{b_4} & \mathbf{0} & \mathbf{b_6} \end{bmatrix} \tag{3.37}$$

$$\dot{\mathbf{R}}\dot{\mathbf{z}} = \begin{bmatrix} \mathbf{d_1} \\ \mathbf{d_1} + \mathbf{d_2} \\ \mathbf{d_1} + \mathbf{d_2} + \mathbf{d_3} \\ \mathbf{d_1} + \mathbf{d_4} \\ \mathbf{d_1} + \mathbf{d_4} + \mathbf{d_5} \\ \mathbf{d_1} + \mathbf{d_4} + \mathbf{d_6} \end{bmatrix} \tag{3.38}$$

From the structure of the matrices, it can be seen that each element of the mechanism is only influenced by its previous bodies on the kinematic chain, as expected.

The expressions in body coordinates for the the mass matrix and the generalized forces of each body are,

$$\bar{\mathbf{M}}_i = \begin{bmatrix} m\mathbf{I} & -m\tilde{\mathbf{g}} \\ m\tilde{\mathbf{g}} & \mathbf{J} - m\tilde{\mathbf{g}}\tilde{\mathbf{g}} \end{bmatrix} \tag{3.39}$$

$$\bar{\mathbf{Q}}_i = \begin{bmatrix} \mathbf{f} - \boldsymbol{\omega} \times (\boldsymbol{\omega} m\mathbf{g}) \\ \mathbf{n} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{g} \times (\mathbf{f} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times m\mathbf{g})) \end{bmatrix} \tag{3.40}$$

where m is the body mass, $\tilde{\mathbf{g}}$ is the dual anti-symmetric matrix of the global position of the mass center of the body ($\mathbf{g}$), $\mathbf{J}$ is the inertia tensor of the body, $\mathbf{f}$ and $\mathbf{n}$ are the external forces and torques applied to the body respectively.

From the expressions presented in Equations 3.39 and 3.40, the dynamic matrices for the whole mechanism in body coordinates can be assembled. The mass term, $\bar{\mathbf{M}}$, is a diagonal matrix whose elements are the sub-matrices $\bar{\mathbf{M}}_i$. The vector of forces, $\bar{\mathbf{Q}}$, contains each individual term $\bar{\mathbf{Q}}_i$.

In order to obtain the equivalent dynamic matrices in relative coordinates for their use in Equation 3.33, the equation of motion in body coordinates is obtained. From the virtual power principle,

$$\mathbf{Z}^{*T}(\bar{\mathbf{M}}\dot{\mathbf{Z}} - \bar{\mathbf{Q}}) = \mathbf{0} \tag{3.41}$$

being $\mathbf{Z}^{*T}$ the virtual velocities on the body coordinates.

Substituting now the result of Equations 3.31 and 3.32 in Equation 3.41 yields,

$$\dot{\mathbf{z}}^{*T}\left\{\mathbf{R}^\top\bar{\mathbf{M}}\mathbf{R}\ddot{\mathbf{z}} - \mathbf{R}^\top(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\dot{\mathbf{R}}\dot{\mathbf{z}})\right\} = \mathbf{0} \tag{3.42}$$

or, in a more compact form,

$$\dot{\mathbf{z}}^{*T}\left\{\mathbf{M}\ddot{\mathbf{z}} - \mathbf{Q}\right\} = \mathbf{0} \tag{3.43}$$

Comparing Equations 3.33 and 3.43, the mass matrix and force vector in relative coordinates can be obtained as follows,

$$\mathbf{M} = \mathbf{R}^\top\bar{\mathbf{M}}\mathbf{R} \tag{3.44}$$

$$\mathbf{Q} = \mathbf{R}^\top(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\dot{\mathbf{R}}\dot{\mathbf{z}}) \tag{3.45}$$

Due to the form of the matrix $\mathbf{R}$, the mass matrix and force vector of the system are suitable to be assembled in a recursive form, as presented in [56], leading to an increment of computational efficiency.

So far, the calculation of the $\mathbf{M}$ and $\mathbf{Q}$ terms of Equation 3.33 has been addressed. The remaining term of Equation 3.33 is the Jacobian matrix of the constraints $\boldsymbol{\Phi}_{\mathbf{z}}$. Defining the closure condition of the mechanism in relative coordinates is not trivial [58]. Instead, in the semi-recursive formulation proposed, natural coordinates, $\mathbf{q}$, are used to impose the closure conditions of the loops at the cut-points [57].

The example presented in Figure 3.2 is used hereafter to show the procedure for the constraint definition. The cut-joint of this example is spherical. The closure

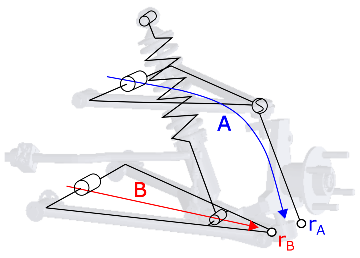


Figure 3.4: Paths for defining the closure-loop conditions in the spherical joint of the double-wishbone suspension example from Figure 3.2.

condition must impose that the position of the joint calculated following the path $\boldsymbol{A}$ and path $\boldsymbol{B}$ is the same. The constraint equations in natural coordinates are,

$$\boldsymbol{\Phi} = \mathbf{r}_{\mathrm{A}} - \mathbf{r}_{\mathrm{B}} = \mathbf{0} \tag{3.46}$$

where $\mathbf{r}_{\mathrm{A}}$ and $\mathbf{r}_{\mathrm{B}}$ are the position vectors of the spherical joint calculated following the path $\boldsymbol{A}$ and path $\boldsymbol{B}$ respectively. In Equation 3.46, there are a total of three equations, one per each coordinate of the three-dimensional space.

In order to obtain $\boldsymbol{\Phi}_{\mathbf{z}}$, the derivatives of the constraint equations should be calculated with respect to the relative coordinates. Thus, as presented in [58],

$$\boldsymbol{\Phi}_{\mathbf{z}} = \boldsymbol{\Phi}_{\mathbf{q}}\mathbf{q}_{\mathbf{z}} \tag{3.47}$$

being $\boldsymbol{\Phi}_{\mathbf{q}}$ the derivatives of the constraint equations with respect to $\mathbf{q}$, which can be easily obtained following the procedure presented in [30], and $\mathbf{q}_{\mathbf{z}}$ the derivatives of the natural coordinates with respect to $\mathbf{z}$, which can be calculated by means of kinematic relations, leading to simple analytic expressions [58].

At this point, all the terms appearing in Equation 3.33 are known and the integration of the dynamic equations can be made. For this purpose, the implicit single-step trapezoidal rule has been adopted. The corresponding difference equations in velocities and accelerations are,

$$\dot{\mathbf{z}}_{n+1} = \frac{2}{\Delta t}\mathbf{z}_{n+1} + \dot{\mathbf{z}}_n^* \quad ; \quad \dot{\mathbf{z}}_n^* = -\left(\frac{2}{\Delta t}\mathbf{z}_n + \dot{\mathbf{z}}_n\right) \tag{3.48}$$

$$\ddot{\mathbf{z}}_{n+1} = \frac{4}{\Delta t^2}\mathbf{z}_{n+1} + \ddot{\mathbf{z}}_n^* \quad ; \quad \ddot{\mathbf{z}}_n^* = -\left(\frac{4}{\Delta t^2}\mathbf{z}_n + \frac{4}{\Delta t}\dot{\mathbf{z}}_n + \ddot{\mathbf{z}}\right) \tag{3.49}$$

If Equations 3.48 and 3.49 are introduced in the dynamic equation of motion, the resulting system can be expressed as,

$$\mathbf{f}(\mathbf{z}_{n+1}) = \mathbf{0} \tag{3.50}$$

where $\mathbf{z}_{n+1}$ are the positions at the next time step, which are the unknowns.

Since Equation 3.50 is a non-linear system of algebraic equations, the Newton-Raphson iteration can be used to find a solution. Thus,

$$\left.\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\right|_{\mathbf{z}=\mathbf{z}_{n+1,i}} (\mathbf{z}_{n+1,i+1} - \mathbf{z}_{n+1,i}) = -\mathbf{f}(\mathbf{z}_{n+1,i}) \tag{3.51}$$

where the residual vector is

$$\mathbf{f}(\mathbf{z}) = \frac{\Delta t^2}{4}\left(\mathbf{M}\ddot{\mathbf{z}} + \mathbf{\Phi_z}^\top \boldsymbol{\alpha}\mathbf{\Phi} + \mathbf{\Phi_z}^\top \boldsymbol{\lambda}^* - \mathbf{Q}\right) \tag{3.52}$$

and the approximated tangent matrix is:

$$\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \simeq \mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\left(\mathbf{\Phi_z}^\top \boldsymbol{\alpha}\mathbf{\Phi_z} + \mathbf{K}\right) \tag{3.53}$$

being **C** and **K** the damping and stiffness matrices, respectively, whose full expressions can be found in [58].

After converging to a solution, the positions $\mathbf{z}_{n+1}$ satisfy the equation of motion and the constraint equations are fulfilled, as an inherit consequence of the formulation employed. However, there is not guarantee on satisfying the constraints equations at velocity ($\dot{\mathbf{\Phi}} = \mathbf{0}$) and acceleration level ($\ddot{\mathbf{\Phi}} = \mathbf{0}$), since they were not imposed. To overcome this problem, the velocities and accelerations are projected,

$$\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\dot{\mathbf{z}} = \left[\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}\right]\tilde{\dot{\mathbf{z}}} - \frac{\Delta t^2}{4}\mathbf{\Phi_z}^\top \boldsymbol{\alpha}\mathbf{\Phi}_t \tag{3.54}$$

$$\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}}\ddot{\mathbf{z}} = \left[\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}\right]\tilde{\ddot{\mathbf{z}}} - \frac{\Delta t^2}{4}\mathbf{\Phi_z}^\top \boldsymbol{\alpha}\left(\dot{\mathbf{\Phi}}_\mathbf{z}\dot{\mathbf{z}} + \dot{\mathbf{\Phi}}_t\right) \tag{3.55}$$

where $\tilde{\dot{\mathbf{z}}}$ and $\tilde{\ddot{\mathbf{z}}}$ are the velocities and accelerations obtained from the integration process, i.e non compliant with the constraints equations.

With the formulation presented, the dynamics of a MB system can be solved with a high level of efficiency. This is the feature that makes this option more suitable for developing the MB vehicle model that will be combined with a state observer on automotive platforms, where the main limitation is the low computational power available.

3.2.2 MBScoder

In order to program and develop a MB model of a vehicle or part of it, as a suspension system, different software techniques are available. Most of them are oriented to a simulation environment based on PC, where the memory and capacity of the hardware is not a limitation. However, when programming for target platforms as embedded hardware, an efficient code must be implemented.

As Pastorino *et al.* presented in [5], one of the first decision is the programming language for writing the code. Although interpreted languages as `Matlab` or `Python` reduce the development time, they have low computational efficiency. Instead, compiled languages as `Fortran`, `C` or `C++`, are more suitable for real-time MB simulations on embedded hardware due to their high efficiency, although programming in these languages is more time consuming.

To overcome this situation, commercial programs as `Matlab` or `Python` offer the option of translating code from their own interpreted language to a compiled language. However, this process leads in general to a generic code which is difficult to read by the developer, limiting the modifications that can be made in order to increase the efficiency of the generated code.

In order to reduce the development time of new MB models in a compiled language, automatic programming is proposed in [5]. Automatic programming, increases the level of abstraction for the developer without compromising code efficiency, since only the minimal source code is written.

In this context, the `MBScoder` has been developed from the work of Pastorino *et al.* in [5]. The `MBScoder` is a software tool for the simulation of the kinematics and dynamics of multibody systems, which is Free Software (licensed under LGPLv3). The `MBScoder` is intended to be used in `Python`, where the simplicity of the interpreted language for code prototyping can be exploited. From the `Python` interface, MB source code can be generated in any programming language implemented. It also offers multiple options for defining the MB model. Initially, it only included natural coordinates [30]. During this thesis, the relative coordinates based on the semi-recursive formulation [56–58] explained in Section 3.2.1.1 were included.

The steps to prepare a file for generating the source code for the simulation of a certain MB model in `Python` are:

1. Definition of the relevant points of the model. If natural coordinates are selected, the type of point (fixed or mobile) and its coordinates should be specified. If relative coordinates are chosen, the type of joint and its position is required. Also the variables representing the DOFs should be indicated.

2. Definition of the constraint equations. For natural coordinates, different options are available regarding the type of constraint: constant distance, perpendicular vectors, alignment, etc. In the case of defining the model in relative coordinates, constraint equations are required for opening the loops in closed-chain mechanism: it should be specified the joint where the closure condition should be imposed and the kinematic chain of the bodies connected by the joint.

3. Definition of the bodies of the system, including the joints presented in the body and the mass properties. In relative coordinates, it is also required a reference to the previous body of the chain in order to program the recursive procedure exposed in Section 3.2.1.1.

4. Definition of additional elements as springs or dampers, including the points of application of the external forces associated to these elements.

5. Definition of the integrator and the MB formulation that will be programmed. There are several options implemented: a model can be easily tested with different integrators and formulations.

6. Definition of the programming language for the source code generation. Three options are available: `Python`, `Matlab` or `C++`. For the `C++` option, two mathematics libraries are available in order to allow efficient algebra operations: `C++-Armadillo` and `C++-Eigen`.

The `MBScoder` has been developed in such a way that the generated source code is always minimal, clean and commented. The `MBScoder` also offers the possibility of generating a simple graphic output based on `OpenGL`. It also includes simple working examples as a reference for new models development.

To conclude, the `MBScoder` is the tool employed during this thesis for creating the source code required for the MB simulations, which should be combined later with the code of the state observer. The MB model is described in `Python` and the code for the simulation is generated in `C++-Eigen`. The `Eigen` library [60] is based on header files. Hence, only the necessary files should be included in the code, reducing the complexity of the implementation.

3.3 Error-State Extended Kalman Filter

The errorEKF uses the approach of the indirect filtering, which is commonly used in multisensor integration for navigation [61, 62]. In this kind of filters, the variables of interest are not directly estimated. Instead, the dynamic model is run without modifications and the filter estimates the drift comparing the simulated data with the measured data. Once that the drift or error is estimated, it is applied to the model in order to correct it [38].

3.3.1 Error-State EKF with force estimation

The errorEKF has been first presented in [35] by Sanjurjo *et al.*, and it has been improved in [4, 36, 38]. This filter is derived from the EKF and, therefore, most of the equations are similar. The main difference between the errorEKF and the EKF consists in the variables that form the state vector (Equation 3.56). While the state vector in the EKF contains the positions and velocities of the MB model, the state vector in the errorEKF includes the differences between real and simulated positions and velocities. Thus,

$$\mathbf{x}^{\top} = \left[\Delta\mathbf{z}^{i\top}, \Delta\dot{\mathbf{z}}^{i\top}\right] \tag{3.56}$$

where $\mathbf{z}^i$ are the independent variables, coincident with the DOFs, from the set of relative coordinates, $\mathbf{z}$, used to define the MB model (Section 3.2.1).

In [38], the errorEKF with force estimation is presented. In this version, the state vector is increased with the errors in accelerations. The results of this filter applied to planar mechanism are promising: the states and input forces are estimated with accuracy. Following the approach presented in [38], the errorEKF with force estimation is applied to a MB vehicle model in this thesis. Hence, the new state vector is,

$$\mathbf{x}^{\top} = \left[\Delta\mathbf{z}^{i\top}, \Delta\dot{\mathbf{z}}^{i\top}, \Delta\ddot{\mathbf{z}}^{i\top}\right] \tag{3.57}$$

The errorEKF is described in a schematic form in Figure 3.5. The prediction phase starts with the execution of one step of the MB simulation. The estimated errors are later obtained comparing the sensor measurements with the simulated data.

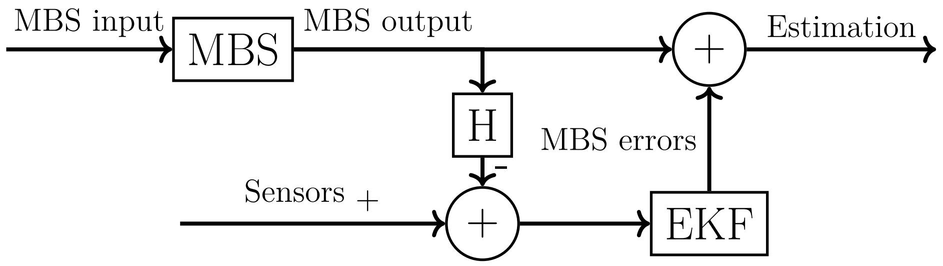


Figure 3.5: Scheme of the errorEKF applied to MB simulations [38].

The propagation phase is performed following Equations 3.58 and 3.59. Note that, since the errorEKF operates in the transformed state space of errors, its *a priori* estimate is always null. In other words, the filter initially assumes that the MB model tracks perfectly the real system.

$$\hat{\mathbf{x}}_k^- = \mathbf{0} \tag{3.58}$$

$$\mathbf{P}_k^- = \mathbf{f_x}_{k-1}\mathbf{P}_{k-1}^+\mathbf{f_x}_{k-1}^\top + \mathbf{\Sigma}^P \tag{3.59}$$

where k is the time step and $\mathbf{f_x}$ is the transition matrix, expressed in Equation 3.60.

$$\mathbf{f_x} = \begin{bmatrix} \mathbf{I} + \frac{1}{2}\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\mathbf{z}^i}\Delta t^2 & \mathbf{I}\Delta t + \frac{1}{2}\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\dot{\mathbf{z}}^i}\Delta t^2 & \frac{1}{2}\mathbf{I}\Delta t^2 \\ \frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\mathbf{z}^i}\Delta t & \mathbf{I} + \frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\dot{\mathbf{z}}^i}\Delta t & \mathbf{I}\Delta t \\ 0 & 0 & \mathbf{I} \end{bmatrix} \tag{3.60}$$

Since the evolution of the acceleration error is unknown, its terms have been set to $\mathbf{0}$. The terms $\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\mathbf{z}^i}$ and $\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\dot{\mathbf{z}}^i}$ are included to account for the acceleration error through the force models of the MB system. The full expressions can be found in [38]. In this work, these terms have been simplified in order to reduce the computational cost, yielding,

$$\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\mathbf{z}^i} = -(\mathbf{R}^{\mathbf{\Phi}\top}\mathbf{M}\mathbf{R}^{\mathbf{\Phi}})^{-1}\mathbf{K} \tag{3.61}$$

$$\frac{\partial\Delta\ddot{\mathbf{z}}^i}{\partial\dot{\mathbf{z}}^i} = -(\mathbf{R}^{\mathbf{\Phi}\top}\mathbf{M}\mathbf{R}^{\mathbf{\Phi}})^{-1}\mathbf{C} \tag{3.62}$$

where the term $\mathbf{R}^{\mathbf{\Phi}}$ is employed to transform the mass matrix of the whole system into the equivalent mass matrix for the set of independent coordinates. Following [58], each column j of $\mathbf{R}^{\mathbf{\Phi}}$ represents the velocities $\dot{\mathbf{z}}$ for an unitary value of the degree-of-freedom $\dot{\mathbf{z}}_j^i$ and null velocity for the rest of DOFs. This is analogous to solve a velocity problem for each DOF.

$$\mathbf{R}_j^{\mathbf{\Phi}} = \dot{\mathbf{z}}|_{\dot{\mathbf{z}}_j^i=1,\dot{\mathbf{z}}_k^i=0;k\neq j} \tag{3.63}$$

In order to propagate the corrections, the equations applied are similar to the equations defined in the DEKF in Section 3.1. Thus,

$$\begin{aligned}
\tilde{\mathbf{y}}_k &= \mathbf{o}_k - \mathbf{h}(\mathbf{z}_k, \dot{\mathbf{z}}_k, \ddot{\mathbf{z}}_k) && (3.64)\\
\boldsymbol{\Sigma}_k &= \mathbf{h}_{\mathbf{x}k}\mathbf{P}_k^{-}\mathbf{h}_{\mathbf{x}k}^{\top} + \boldsymbol{\Sigma}^{S}{}_k && (3.65)\\
\mathcal{K}_k &= \mathbf{P}_k^{-}\mathbf{h}_{\mathbf{x}k}^{\top}\boldsymbol{\Sigma}_k^{-1} && (3.66)\\
\hat{\mathbf{x}}_k^{+} &= \hat{\mathbf{x}}_k^{-} + \mathcal{K}_k\tilde{\mathbf{y}}_k && (3.67)\\
\mathbf{P}_k^{+} &= (\mathbf{I} - \mathcal{K}_k\mathbf{h}_{\mathbf{x}k})\mathbf{P}_k^{-} && (3.68)
\end{aligned}$$

where $\tilde{\mathbf{y}}_k$ is the error or mismatch (often called innovation) between the expected sensor readings, $\mathbf{o}_k$, and their actual values, $\mathbf{h}(\mathbf{z}_k, \dot{\mathbf{z}}_k, \ddot{\mathbf{z}}_k)$. The innovation covariance matrix, $\boldsymbol{\Sigma}_k$, represents the uncertainty in the system state. It is projected via the sensor function, $\mathbf{h}_{\mathbf{x}k}\mathbf{P}_k^{-}\mathbf{h}_{\mathbf{x}k}^{\top}$, plus an additional Gaussian noise originated at the sensor itself ,$\boldsymbol{\Sigma}^{S}{}_k$. Small values of $\boldsymbol{\Sigma}_k$ mean that the observation introduces useful information for the estimation of the states. By evaluating the Kalman gain ,$\mathcal{K}_k$, the estimation of the mean and covariance are updated in Equations 3.67 and 3.68 respectively [4].

It should be remarked that the virtual measurements, $\mathbf{h}(\mathbf{z}_k, \dot{\mathbf{z}}_k, \ddot{\mathbf{z}}_k)$, from Equation 3.64 are built by using the coordinates of the MB model instead of the states, as in the EKF. In addition, the Jacobian of the measurements model, $\mathbf{h}_{\mathbf{x}k}$, has the same expression as in an equivalent conventional Kalman filter. The partial derivatives with respect to the errors in the states have the same value than the partial derivatives with respect to the states.

After the propagation stage, the estimations of the errors in position, velocity and acceleration of the independent coordinates, $\hat{\mathbf{x}}_k^{+}$, are obtained. In a last step, the corrections should be propagated through the dependent MB variables. This is done by projecting the errors in the independent coordinates over the constraint equations.

At position level, the position problem should be solved. It is a non-linear problem, which is typically solved with an iterative method. Since it is computationally expensive, in [36] the position problem is linearized assuming that the corrections in positions are expected to be small. Hence,

$$\boldsymbol{\Phi}_{\mathbf{z}}\Delta\hat{\mathbf{z}} = \mathbf{0} \qquad (3.69)$$

If the Jacobian of the constraints, $\boldsymbol{\Phi}_{\mathbf{z}}$, and the vector of the estimated positions errors, $\Delta\hat{\mathbf{z}}$, are split in their independent and dependent parts, Equation 3.69 can be written as,

$$\Delta\hat{\mathbf{z}} = \begin{bmatrix} \Delta\hat{\mathbf{z}}^{i} & \Delta\hat{\mathbf{z}}^{d} \end{bmatrix}^{\top} \qquad (3.70)$$

$$\boldsymbol{\Phi}_{\mathbf{z}}{}^{d}\Delta\hat{\mathbf{z}}^{d} = -\boldsymbol{\Phi}_{\mathbf{z}}{}^{i}\Delta\hat{\mathbf{z}}^{i} \qquad (3.71)$$

where $\Delta\hat{\mathbf{z}}^{i}$ is the term of the state vector, $\hat{\mathbf{x}}^{+}$, referred to the positions of the DOFs.

Solving the linear system of Equation 3.71, the estimated dependent errors, $\Delta\hat{\mathbf{z}}^{d}$, are obtained. Then, the resulting vector of estimated position errors, $\Delta\hat{\mathbf{z}}$, is fed back to the MB simulation to obtain the position estimation, $\mathbf{z}$, as follows,

$$\hat{\mathbf{z}} = \mathbf{z} + \Delta\hat{\mathbf{z}} \qquad (3.72)$$

It must be noted that this method is an approximation. Therefore, a perfect fulfillment of the constraints at position level is not expected. However, since the corrections are performed every time step, the errors are usually acceptable for most applications. If the sensors employed have a low update rate, the position errors may become too big to employ this approximation, and hence the non-linear position problem should be solved instead [4]. In the particular case of this thesis, the computational efficiency is a critical factor. Thus, the linearization of the position problem is accepted as the most suitable approach.

Regarding the correction of the velocity estimation, it is performed after correcting the positions. Splitting the MB variables into their independent and dependent terms, the system for solving the velocity problem is,

$$\mathbf{\Phi_z}^d \hat{\dot{\mathbf{z}}}^d = -\mathbf{\Phi}_t - \mathbf{\Phi_z}^i \left[\; \dot{\mathbf{z}}^i + \Delta\hat{\dot{\mathbf{z}}}^i \; \right] \tag{3.73}$$

Therefore, the vector of estimated velocities for the MB model yields,

$$\hat{\dot{\mathbf{z}}} = \left[\; (\dot{\mathbf{z}}^i + \Delta\hat{\dot{\mathbf{z}}}^i) \quad \hat{\dot{\mathbf{z}}}^d \; \right]^\top \tag{3.74}$$

The corrections in accelerations of the dependent coordinates are obtained by solving the acceleration problem as presented in [30],

$$\mathbf{\Phi_z} \hat{\ddot{\mathbf{z}}}^d = -\dot{\mathbf{\Phi}}_t - \dot{\mathbf{\Phi}}^i_{\mathbf{z}^d} \dot{\mathbf{z}}^d - \mathbf{\Phi_z}^i \left[\; \ddot{\mathbf{z}}^i + \Delta\hat{\ddot{\mathbf{z}}} \; \right] \tag{3.75}$$

The final estimated acceleration vector results as follows,

$$\hat{\ddot{\mathbf{z}}} = \left[\; (\ddot{\mathbf{z}}^i + \Delta\ddot{\mathbf{z}}^i) \quad \hat{\ddot{\mathbf{z}}}^d \; \right]^\top \tag{3.76}$$

Once that the accelerations have been corrected, the forces $\Delta\hat{\mathbf{Q}}$ which would have been applied to the mechanism in order to correct the acceleration error must be calculated. However, there are infinite possibilities to calculate a vector $\Delta\hat{\mathbf{Q}}$ which produces the desired effect. The first assumption made is that all the unknown forces are applied to the independent variables, which correspond to the DOFs,

$$\Delta\hat{\mathbf{Q}}^d = \mathbf{0} \tag{3.77}$$

$$\Delta\hat{\mathbf{Q}}^i = \hat{\mathbf{M}}^i \ddot{\mathbf{z}}^i - \hat{\mathbf{Q}}^i \tag{3.78}$$

$$\Delta\hat{\mathbf{Q}} = \left[\; \Delta\hat{\mathbf{Q}}^i \quad \Delta\hat{\mathbf{Q}}^d \; \right] \tag{3.79}$$

Applying $\Delta\hat{\mathbf{Q}}$ to the generalized forces of the MB model, $\mathbf{Q}$, the error in acceleration is corrected and its expected value for the next time step is null. Since the constraints in position and velocity were also imposed, the *a priori* state vector for the next time step is null, in coherence with the Equation 3.58.

However, depending on the mechanism, applying the force corrections over the DOFs can lead to inaccurate results. Such is the case of study in this work, since the forces acting on a vehicle are highly related to the tires and not to the independent variables. This issue is discussed in Section 5.2.2.4.

3.4 Measurement noise and covariance matrices of the process

When working with Kalman filters, there are several uncertainties that must be considered. For instance, the sensors employed are not perfect, which leads to noisy measurements. In addition, the errors on the MB model with respect to the real system cannot be identified as most of them are unknown.

The measurement noise and the covariance matrices of the process are the parameters where all the uncertainties explained above are included. The importance of the covariance matrices is such that if they are not properly set, the filter can become unstable [38].

3.4.1 Measurement noise

The noise of the sensors is perfectly determined in the simulation environment. A common procedure, followed at this work, consists on generate it from a MB model which acts as the real system. Later, white Gaussian noise is generated and added to the sensors signal, accomplishing the assumptions of Kalman filtering [54]. In real sensors, although they are not perfect, the noise is usually provided by the manufacturer and can be considered as white Gaussian noise.

3.4.2 Covariance matrix

The covariance matrix includes the uncertainties of the MB model. This matrix must be tuned in order to reach a robust solution which converges to accurate solutions independently of the model errors. In addition, as reflected in [63], if fictitious process noise is added, the filter will place more emphasis on the measurements, which will improve the filter performance. As a counterpart, in case the system model is correct, this will reduce the accuracy.

The structure of the covariance matrix is as proposed in [64]. It can be obtained from the transition matrix given by Equation 3.60 yielding,

$$\mathbf{\Sigma}^P = \sigma^2 \begin{bmatrix} \frac{\Delta t^5}{20} & \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^4}{8} & \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^2}{2} & \Delta t \end{bmatrix} \tag{3.80}$$

where σ^2 is a diagonal matrix which contains the noises associated to the MB variables presented in the state vector. It must be noted that only the constant elements of the transition matrix were considered. Thus, the covariance matrix is constant during the simulation and it can be evaluated before the simulation starts [38], reducing the computational cost of the filtering.

Chapter 4

New Generation Embedded Hardware

Since the 1970s, there has been an exponential increase in the number of electronics systems in vehicles. Electronic systems have gradually replaced those that are purely mechanical or hydraulic [6]. These advances were possible due to the evolution of in-vehicle hardware capabilities.

Nowadays, the implementation of more complex ADAS strategies is pushing the development of more powerful on-board hardware to achieve real-time performance [65, 66]. In addition, embedded systems will go beyond the engine and energy control on electric vehicles [41].

As explained in Chapter 2 and Chapter 3, one of the main limitations for implementing state observers based on MB models is the computational power of conventional ECUs. Since new generation hardware is starting to be available in-vehicle, a detailed study on this devices should be performed. This new trend can give an opportunity for executing real-time MB-based state observers for in-vehicle applications.

This chapter starts with a brief overview of the new generation hardware available, selecting the platform that is used during this thesis. Afterwards, the strategies for implementing MB-based state observers on the hardware selected are explained.

4.1 Modern Hardware Analysis

Computation capabilities of conventional processors have been growing exponentially due to sustained innovation in processors technology [67]. The increment of performance is related to a higher number of computing elements per microprocessors, at the expense of reducing the power efficiency [45]. However, power consumption is critical in embedded systems and it has become the limiting factor for increasing the performance of embedded processors [8].

As stated in Section 2.2.1, in order to provide high computational performance while keeping a low-power consumption, advanced multi-core architecture have been developed. Having more than one core in the system allows developing parallel processing on chips, reducing the computational cost of an application [44]. From the multiprocessor platforms available, heterogeneous processors can map each application to the best suited core to meet its performance demands. This leads in general to a lower energy consumption without reducing the computational power. Hence, heterogeneous processors are being used for hard real-time systems in embedded applications [8].

4.1.1 Heterogeneous processors for scientific computing

In the field of heterogeneous computing, the main core is based on traditional processors. However, the co-processing units are usually based on unconventional cores such as Application-Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs) or GPUs [67–69].

ASIC devices are expensive and their development cycle is long. Meanwhile, GPUs and FPGAs are cheaper and programmable: they can be used in any application without redesigning the hardware. Although an ASIC device is tailored for a particular application providing the highest performance, FPGAs and GPUs are gaining in popularity in heterogeneous computing due to their flexibility and reduced price.

Graphic Processing Units (GPUs) were developed for parallel manipulation of tasks related to computer graphics [70]. However, their usage has been extended to increasing the efficiency of intensive computations. In a GPU, a large number of programmable cores are used to parallelize the execution of a program. In MB dynamics, they have been using in [50–53] for accelerating the simulation of large MB systems.

Field Programmable Gate Arrays (FPGAs) are programmable hardware devices that offer the high performance of custom hardware, with some of the flexibility of software. FPGAs are built from programmable logic elements which can be combined to exactly build the hardware required for a specific application. This results in orders of magnitude speed-up over conventional processors for computationally-intensive tasks [41]. The presence of FPGAs in embedded applications has been growing in last years. FPGAs are used for controlling industrial processes and motors, in machine vision to inspect manufacturing lines or in industrial networking [71, 72]. Their flexibility is one of their major advantages: it can be reprogrammed to accomplish changing requirements without high costs [72].

In automotive applications, Multiprocessor System on Chip (MPSoC) solutions embedding an FPGA or GPU are starting to be used due to the high computational cost of ADAS [73]. In contrast to GPUs, there is no research on how to exploit FPGA for accelerating a MB simulation. Due to the increasing presence of FPGAs is in automotive applications, they can be used to reduce the computational cost of MB simulations.

At this point, it is worth detailing certain characteristics of the FPGA elements. In general, an FPGA is made of wires connecting logic gates and registers [74]. The logic gates perform simple boolean logic on inputs. Combining different gates, complex operations can be performed. The registers are designed to store data that needs to be accessed quickly. The elements of an FPGA, can be classified in [7]:

1. Look-Up Table (LUT): It is a flexible resource capable of implementing a logic function, small memory elements or registers.

2. Flip-Flop (FF): It is a sequential circuit element implementing a 1-bit register, whose purpose is to synchronize logic and save logical states during clock cycles.

3. Block Random Access Memory (BRAM): It is a memory block with the special purpose of satisfy dense memory requirements.

4. Digital Signal Processor (DSP): It is a sub-unit dedicated for high-speed DSP arithmetic supporting operations such as multiply-add, multiply-accumulate (MACC), three-input add, etc.

From the available heterogeneous processors with FPGAs, the Xilinx® Zynq-7000 XC7Z020 is widely used in automotive applications [71]. It is a low-end embedded platform based on an ARM Cortex-A9 combined with an Artix-7 FPGA as co-processor. It is used in computer vision applications or control purposes and machine learning [47, 48, 70, 75–79].

The properties of the Zynq-7000 XC7Z020 are presented in Table 4.1. There are more powerful devices on the market. However, their cost are also higher and this could reduce their applications, since this cost will be reflected in the market price of the product (a vehicle in this case). Furthermore, the automotive grade Zynq-7000 devices offered by Xilinx® are based on the same processor [80]. Due to the broadly use of the Zynq-7000 XC7Z020 device and in order to provide results applicable to real use-cases, it is selected in this work.

Table 4.1: Features of the Zynq-7000 XC7Z020 processor [81].

Zynq-7000 (Device Code: XC7Z020-CLG484-1)		
Main processor	Processor Core	Dual ARM Cortex-A9
	Processor Extensions	NEON & Single\Double Precision Floating Point
	Max. Frequency	667 MHz
	L1 Cache	32KB Instruction, 32KB Data per processor
	L2 Cache	512KB
	On-Chip Memory	256KB
	External Memory	DDR3, DDR3L, DDR2, LPDDR2
Co-processor	FPGA	Artix-7
	Logic Cells	85K
	LUT	53,200
	FF	106,400
	BRAM	140 (4.9Mb)
	DSP	220

4.1.2 FPGAs considerations

Different features must be considered when programming FPGAs: programming language, numerical data types, area or resource cost (understanding it as the amount of hardware required to obtain the desired functionality) and speed or throughput (defined as the rate at which the circuit can process data). Although for the application of this thesis a maximum throughput is desired in order to increase the speed-up of the simulation, a trade-off between area and throughput should be achieved, especially in cases where the hardware resources of the FPGA are not sufficient.

With respect to the programming language, FPGAs are programmed in Hardware Description Languages (HDLs), such as Verilog or VHDL, which is quite different from the software programming languages. HDL programming limits the use of FPGAs to developers with a strong knowledge in hardware designs. In order to increase their popularity, manufacturers have been releasing translation tools which allows to program FPGAs in C-like languages or Open CL. Nevertheless, hardware design knowledge is still required in order to guide these automatic-translation tools into the desired implementation.

Regarding the numerical data types, although FPGAs can support floating-point calculations, they are oriented for fixed-point operations. The use of floating-point data involves more resources. Thus, depending on the application, this could be an issue. For MB dynamics, it is desired to work with floating-point data, since more precision and range of numbers is available. The evolution of FPGAs is resulting into devices with more resources, reducing the footprint required for the implementation of floating-point calculations [7]. Hence, high-precision calculations are becoming more common in FPGA computing.

Nevertheless, the main disadvantage of FPGAs is their limited hardware resources. If there are not enough resources for programming an algorithm, it should be partitioned or under-optimized, resulting in a less efficient implementation. As explained before, this issue is being mitigated due to the continuous FPGA technology evolution. There is a huge range of FPGAs with enough capabilities for implementing different designs.

4.2 Hardware/Software Partitioning

Hardware/Software partitioning is an important stage in the design of embedded systems and, if well executed, can result in a significant improvement in system performance [7]. In heterogeneous processors, the code should be partitioned in order to share the computational load between the main processor and the co-processor. If an FPGA is available, it can be used to accelerate the most computationally intensive tasks of a program due to their parallel processing nature. Meanwhile, the less intensive processes can be implemented in the main processor.

FPGAs are good targets for the implementation of problems which can be efficiently divided into parallel task. Due to the inherent parallel execution of the FPGAs, multiple operations can be processed simultaneously to calculate the final result in a shorter time [7]. In the case of study, the source code of the state observer plus the code of the MB system is large and involves a high number of computationally expensive matrix operations.

In order to accelerate a particular task on the FPGA, the code should be analyzed in order to offload the most demanding tasks into the FPGA. The cost and performance of the whole system depends on a proper code partitioning [82].

Traditionally, the process of partitioning a program was carried out manually by the system designer. In base to experience, the most demanding functions of the applications and the most suited to be implemented on hardware were offloaded to the co-processor. More recently, a number of algorithms and techniques have been developed enabling the partitioning process to be more automatic. Through code profiling, the most time consuming operations can be identified.

Profiling is a form of program analysis that is used to aid the optimization of a software application. It is used to measure multiple properties of an application code, which can later be used to identify the bottlenecks of the system [7]. Once identified, the bottlenecks can be optimized by rewriting the original software function in order to program it on an FPGA. In this work, a profiling analysis is executed for the code of the errorEKF based on a MB vehicle model. In Figure 4.1, the most relevant results of the profiler are shown.

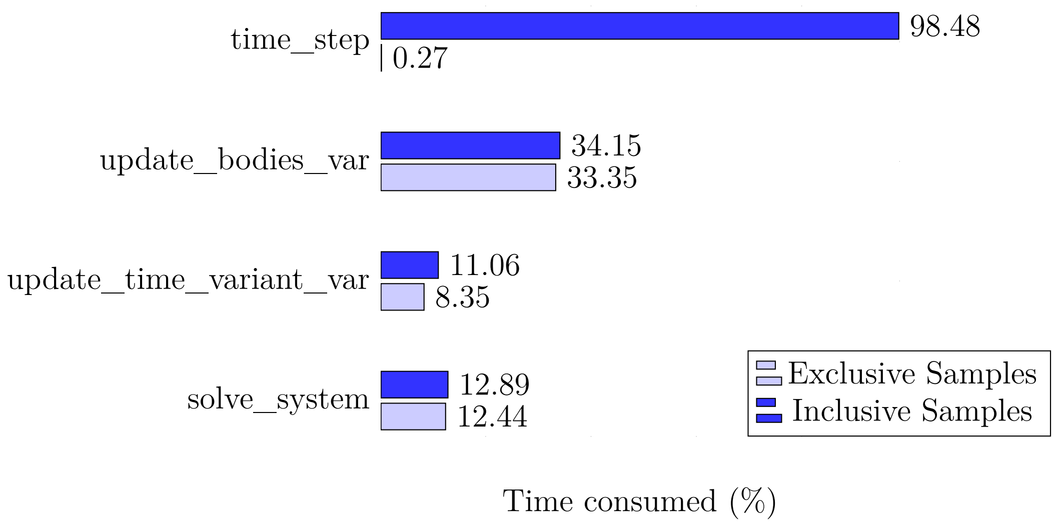


Figure 4.1: Profiling summary of a simulation of a full-vehicle MB model combined with the errorEKF. Only the most time consuming operations are included.

The main function in order to compute a time step is called `time_step()`. Its principal task is to call sequentially each of the child function required to compute a time step. This is reflected in Figure 4.1, where the percentage of exclusive samples is quite reduced. Inclusive samples are referred to the overall time spent on the function, including the time consumed by the child functions. The exclusive samples are useful to measure the time consumed by the function itself.

The main tasks executed in one time step are the integration of the MB model, the evaluation of the filter equations and the propagation of the estimations through the whole set of coordinates (Section 3.3). Three tasks stand out among the rest in terms of time consuming: `update_bodies_var()`, `update_time_variant_var()` and `solve_system()`. The other percentage of `time_step()` is dispersed in several functions, such as the calculation of the constraint equations and the $\mathbf{b}_i$ and $\mathbf{d}_i$ terms among others.

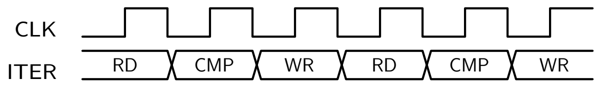
From the function selected, `update_bodies_var()` is the most time consuming and, if it is efficiently optimized, it will lead to the highest acceleration. This function is assigned to update the motion of the bodies and joints each time the MB coordinates are modified. The function `update_time_variant_var()` computes the mass matrix and vector of generalized forces of the MB model (Equations 3.44 and 3.45). It is executed each iteration of the MB integrator and for applying the filter corrections in the forces. The last function considered is `solve_system()`. It is used for solving the system of Equation 3.50 in order to get the value of the coordinates for the next time step. As is shown in Figure 4.1, these functions are self-dependent and they can be entirely programmed on the FPGA.

There are some differences within functions that should be remarked. In `update_time_variant_var()` and `update_bodies_var()`, the operations performed are simple and small matrix additions and multiplications. The main cause of the time consumed by both functions relies on the amount of operations and not on their complexity. Meanwhile, `solve_system()` involves a complex mathematical algorithm for solving system of equations. Thus, the parallelization strategy followed for each function should be different.

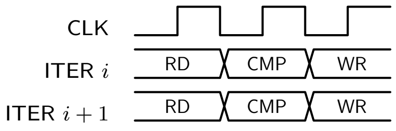
4.3 Hardware Implementations

For implementing a particular operation on the FPGA, the algorithm should be modified in order to optimize its execution. There cannot be gain in performance by just switching from a GPP based implementation to an FPGA based implementation of a given program [83]. As explained in previous Sections, the main advantage of an FPGA is the high level of code parallelization that can be reached. Due to the flexibility of FPGAs, almost any algorithm can be optimized with respect to its GPP implementation. Normally, the most time-consuming tasks in a particular application are related to the iterative execution of certain operations inside loops. Depending on the data dependency within loops iterations, two main techniques can be used to exploit the parallelism: loop unrolling and loop pipelining. The loop unrolling consists on create multiple copies of the loop body so that each loop can be executed in parallel. The loop pipelining allows operations in a loop to be implemented in a concurrent manner, so that the next iteration of the loop can start before the last operation in the current loop iteration is complete.

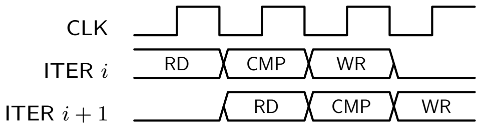
In Figure 4.2, a generic loop is taken as example. In Figure 4.2a, the sequential process of the loop is presented: the iteration $i+i$ starts right after the iteration i has finished. If the loop is unrolled (Figure 4.2b) and there is no data dependency, both iterations can be executed in parallel and all the loop will be computed at the same time. However, if there is data dependency within iterations, the unroll strategy cannot be applied, since iteration $i+1$ should wait for the data computed in iteration i. This would result in a sequential execution. Meanwhile, the loop pipelining allows to execute the iteration $i+i$ as soon as the data required is available, while iteration i is still computing. As can be seen in Figure 4.2c, this also implies a reduction of the latency of the loop execution.



(a) Sequential execution of two iterations of a generic loop.



(b) Unroll technique applied on a generic loop.



(c) Pipeline technique applied on a generic loop.

Figure 4.2: Illustrative example of unrolling and pipelining a loop, which operations are read the input data (RD), perform the computations (CMP) and write the output data (WR).

From the previous example, it can be concluded that the parallelism within loop iterations is limited by the data dependencies between iterations. In addition, the number of available hardware resources is also a huge limitation. The unroll strategy creates several copies of a loop iteration in order to execute them in parallel, consuming a huge amount of hardware resources. The pipeline technique allows to use the same chip footprint for different set of data. Therefore, in cases where the available resources are limiting the implementation, the pipeline technique is the best alternative for optimizing the loop execution.

From the profiling results, three functions stand among others as the suitable candidates to be implemented on the FPGA. As each function performs different task, the strategy followed to obtain an efficient implementation will differ from one to another. It must be noted that, due to the restrictions of the FPGA programming, the `C++-Eigen` library is not valid to be used on the FPGA. Thus, the operations performed by `C++-Eigen` in the code that is offloaded to the FPGA must be replaced manually. The software employed to program each function in the FPGA is provided by Xilinx®, the so-called `Vivado HLS`.

4.3.1 Function `update_time_variant_var()`

As explained in Section 4.2, this function updates the mass matrix of the system. The process is divided in two steps: the calculation of the body mass matrices, following Equation 3.39, and the assembly of each body mass matrix on the global matrix in a recursive form as indicated in Section 3.2.1.1. This second stage also involves the terms $\mathbf{b}_i$ and $\mathbf{d}_i$ from Equations 3.31 and 3.32, which are required for assembling the mass matrix and the vector of forces as presented in Equations 3.37 and 3.38. Two possible options to implement this function can be derived: the calculation of the body mass matrices, the assembly of the global mass matrix or the hole `update_time_variant_var()` function.

Regarding to the calculation of the body mass matrices, the independence between them allows to fully parallelize the process. The operations involved are simple, and the code for the evaluation of the mass matrix of a body i is presented in the algorithm 1.

Algorithm 1: Function to compute the individual mass matrix of the bodies which compose the MB model.

```
Input: Mass, position of the center of gravity and inertia tensor with respect
       to the center of gravity of each body.
Output: Individual mass matrix of each body.
1 for $i = 1$ to $N$ do                          // Being $N$ the number of bodies
2     $\mathbf{mass\_mat}_i(0...3, 0...3) = mass_i \cdot \mathbf{I}_{3\times3}$;
3     $\mathbf{mass\_mat}_i(0...3, 3...6) = (-mass_i) \cdot \tilde{\mathbf{rg}}_i$;          // Being $\tilde{\mathbf{rg}}$ the
        equivalent antisymmetric matrix of the vector related to
        the position of the center of gravity
4     $\mathbf{mass\_mat}_i(3...6, 0...3) = mass_i \cdot \tilde{\mathbf{rg}}_i$;
5     $\mathbf{mass\_mat}_i(3...6, 3...6) = \mathbf{Jg}_i - (mass_i \cdot \tilde{\mathbf{rg}}_i \cdot \tilde{\mathbf{rg}}_i)$;
6 end
```

Each mass matrix involves four products of a scalar (the mass of the body) and a square matrix of three rows and three columns, one matrix product, and one matrix subtraction. In order to compute the mass matrix of a body, the data that needs to be sent to the FPGA is composed of the mass, center of gravity and the tensor of inertia with respect to the center of gravity of each body. The output data is the individual mass matrix of each body, which is a square matrix of six rows and columns.

For evaluating the implementation of this function, a MB model of a vehicle which involves 29 bodies, presented in Section 5.2, has been considered. The first implementation is the code without any modification or directive of parallelization. This will be useful for comparing the following approaches and evaluate the achieved performance. The results of this implementation are shown in Table 4.2.

Table 4.2: Summary report of the FPGA implementation for computing the mass matrices of each body without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	4585	2	7	1	6
Read Input data	754	0	0	≈0	≈0
Individual Mass Matrices	696	0	7	1	6
Write Output data	3132	0	0	≈0	≈0

From the results in Table 4.2, it can be seen that most of the clock cycles of the implementation are consumed during the reading of the input data and during the writing of all the mass matrices. Both operations are substantially more costly than the evaluation of the mass matrices themselves. Due to the lack of optimization, the amount of consumed resources is almost residual.

In order to optimize the process, different actions can be applied. By default, the input and output arrays where the data is stored are defined with a single port memory resource, which limits the read and write ports of the array. If the evaluation of the mass matrices is performed in parallel, the number of read and write operations will increase, and the limited number of read and write ports of the arrays will lead to an inefficient implementation. As a solution to this problem, the arrays can be partitioned into smaller arrays (implemented as multiple memories) increasing the number of load and store ports. As a counterpart, the more the arrays are partitioned, the more resources are needed. Although a full partitioned array can eliminate the limitations of read and write operations, it could exceed the number of available resources.

In this particular case, the input and output data have the same size for each mass matrix calculation. Therefore, the arrays can be partitioned in a factor equal to the number of bodies. As a consequence, each mass matrix calculation will dispose of all the data required with independence of the rest of the bodies.

After partitioning the input and output arrays, the parallelization of the code can be addressed. In a first approach, the parallelization of all the mass matrices calculations is implemented. For this purpose, a loop unrolling is performed: the code for the mass matrix calculation of each body is copied a number of times equal to the number of bodies. The results of this implementations are presented in Table 4.3.

Table 4.3: Summary report of the FPGA implementation for computing the mass matrices of each body with a full parallelization of the mass matrices calculation.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	1453	13	197	38	163
Read Input data	377	0	0	≈0	1
Individual Mass Matrices	24	0	197	38	161
Write Output data	1045	0	0	≈0	1

With this implementation, the clock cycles required to obtain all the mass matrices is minimum and equal to the latency of calculating only one mass matrix, since all are computed in parallel. However, the required DSPs and LUTs exceed the available since the implementation involves more percentage of both resources than the 100% and, thus, this option cannot be implemented on the FPGA. To overcome the resource limitation, the parallelization should be reduced. This can be done either reducing the number of mass matrices that can be calculated at the same clock cycle, or pipelining the calculations. Using the loop pipelining, the same section of hardware can be employed for calculating several mass matrices in a concurrent manner: before the calculation of one mass matrix is finished, the calculations for other body can start. With this approach, the latency of the mass matrices calculation can be reduced without an excessive resource demand, although the parallelization level will not be as optimum as when unrolling the loops. In Table 4.4, the summary of the pipelined implementation is presented.

Table 4.4: Summary report of the FPGA implementation for computing the mass matrices of each body with a pipeline of the mass matrices calculation.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	1478	20	45	38	74
Read Input data	377	0	0	≈0	1
Individual Mass Matrices	52	0	45	36	72
Write Output data	1045	0	0	≈0	1

The loop pipelining has lead to an implementation almost similar in terms of latency to the full loop unrolling implementation, with a less demand of hardware resources. Since the latency is almost minimum and the amount of resources required does not exceed the resources available, this is the implementation selected for optimizing the function `update_time_variant_var()` during a simulation of a MB-based state observer. It can be executed on the FPGA at a frequency of 125 MHz.

However, as introduced above, the function `update_time_variant_var()` involves more calculations in order to assembly the complete mass matrix of the MB system. As presented in Table 4.4, there are resources which are not employed. Furthermore, the pipelining level can be reduced in case of more area is required. Thus, more operations can be programmed on hardware and the efficiency of the simulation could be higher. As a counterpart, more data is required and the latency of the communication will increase, which can result into a reduction of the performance. The results for the implementation of the global mass matrix calculation are presented in Table 4.5, without considering any parallelization.

Table 4.5: Summary report of the FPGA implementation for computing the mass matrix of the system without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	23310	4	291	61	225
Read Input data	630	0	0	≈0	1
Global Mass Matrix	21993	3	291	61	224
Write Output data	679	0	0	≈0	≈0

For implementing the global mass matrix calculation, the input data has been increased with the terms $\mathbf{b}_i$ and $\mathbf{d}_i$ from Equations 3.31 and 3.32, since they are required to assembly the mass matrix of the system. The output data of the implementation are the non-zero elements of the global mass matrix. The report from Table 4.5 indicates that the resources required for computing the global mass matrix of the MB model presented in Section 5.2 are not enough.

As a last alternative, only the assembly of the mass matrix is explored. This task is more time consuming than the evaluation of the individual mass matrices. Thus, the benefits of its acceleration are higher. However, this part of the mass matrix calculation involves more operations and there could be not enough resources on the FPGA. The assembly of the mass matrix is a recursive procedure, as explained in Section 3.2, and there are dependencies within bodies. Thus, the pipeline approach is the best suited to optimize this tasks. For this implementation, the individual mass matrices are computed in the main processor and they are sent to the FPGA for their assembly into the global mass matrix. The summary of the implementation without any optimization is presented in Table 4.6.

Table 4.6: Summary report of the FPGA implementation for assembling the mass matrix of the system without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	25398	8	283	59	218
Read Input data	1297	0	0	≈0	1
Mass Matrix assembly	23414	8	283	59	218
Write Output data	679	0	0	≈0	≈0

From the results of Table 4.6, it can be concluded that there are not enough resources in the selected FPGA for implementing the assembly of the mass matrix on it. Therefore, only the evaluation of the individual mass matrices of the bodies of the model can be implemented on the FPGA.

4.3.2 Function `update_bodies_var()`

Due to the nature of topological methods for MB modelling, after each integration of the MB equations, a post-processing step should be made in order to obtain the variables related to the absolute motion of each joint and body coherently with the new values of the coordinates. This process is recursive, since the position of a body depends directly on the position of its previous body on the kinematic chain. As opposite to the individual mass matrices calculation, there is a data dependency between iterations. As explained in the beginning of Section 4.3, this means that the unroll strategy of the function `update_time_variant_var()` is not adequate for this particular case, since it would end up in a sequential execution of the function. Thus, a pipeline strategy seems to be the suitable option for optimizing the function `update_bodies_var()`.

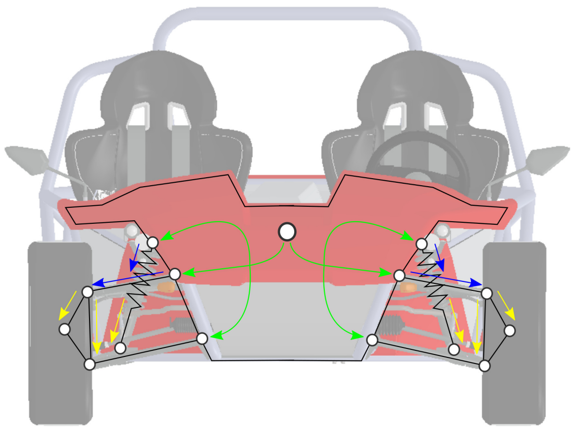


Figure 4.3: Sequence followed in order to calculate the absolute motion of each joint and body. The process starts in the chassis (root) and is made recursively through the consecutive joints and bodies (leafs). Each colour represents the calculations that can be performed in parallel: the last joints and bodies (yellow path) only can be obtained after solving the previous ones (green and blue).

In Figure 4.3, the recursive procedure is atated following the case of the MB simulation of a vehicle. In this example, the chassis is the root of the system, as it is the first body of the chain. Thus, once the motion of the chassis is derived, the motion of the suspension systems can be started to be calculated (green paths of Figure 4.3). As the suspensions are independent, they can be solved in parallel. However, inside each suspension the process is sequential. Therefore, for calculating the motion of the wheel, which is the last body of the chain, the rest of the bodies must be calculated. As in the case of the chassis, some bodies can be calculated in parallel (blue paths of Figure 4.3) and, later, the last bodies and joints can be obtained (yellow paths of Figure 4.3).

The function `update_bodies_var()` is based on kinematic relations between bodies and joints. The variables of each body that must be computed are the position and velocity of its center of gravity, its angular velocity and its tensor of inertia with respect to the center of gravity. Regarding the joints, their position must be calculated and, depending on the type of joint, different variables for representing the direction of the DOF allowed by the joint should be obtained:

- Revolute joint: a vector representing the direction of the axis of revolution.
- Prismatic joint: a vector in the translational direction of the joint.
- Spherical joint: three vectors defining the rotations made around the joint.
- Floating joint: this joint is employed for defining the chassis, since it represents the kinematic of free solids and should allow six DOFs. It can be seen as a combination of an spherical joint which allows the three rotations in the space, and three prismatic joints, one per each translational DOF. Thus, the direction of the three vectors for the rotations added to the three translational vectors should be computed.

The procedure for obtaining these values is explained below, following the example of a piece of a kinematic chain from a steering system represented in Figure 4.4.

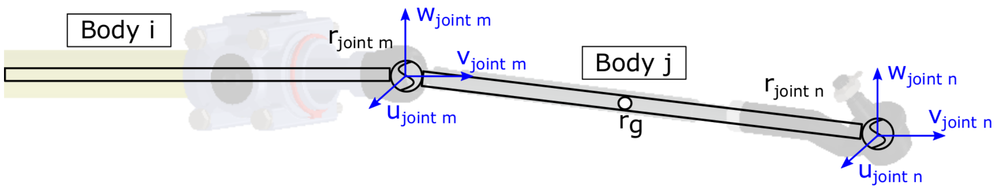


Figure 4.4: Kinematic chain of a steering system in order to illustrate the calculation of the magnitudes associated to a body and the variables required to define a joint.

It is assumed that all the magnitudes related to the body i are know, as the variables defining the joint m. In order to obtain define the motion of the body j and to determine the position and rotation of the joint n, the position of the joint m is considered as the origin of the local system of coordinates of the body j. Hence,

$$r_g = r_{joint_m} + A_j \bar{r}_g \tag{4.1}$$
$$A_j = A_i A_{ij} \tag{4.2}$$
$$\omega_j = \omega_i + A_j \left[\begin{array}{ccc} \alpha_m & \beta_m & \gamma_m \end{array} \right] \tag{4.3}$$
$$\dot{r}_g = \dot{r}_{joint_m} + \omega_j \times (r_g - r_{joint_m}) \tag{4.4}$$
$$r_{joint_n} = r_{joint_m} + A_j \bar{r}_{joint_n} \tag{4.5}$$
$$\dot{r}_{joint_n} = \dot{r}_{joint_m} + \omega_j \times (r_{joint_n} - r_{joint_m}) \tag{4.6}$$
$$u_{joint_n} = A_j \bar{u}_{joint_n} \tag{4.7}$$
$$v_{joint_n} = A_j \bar{v}_{joint_n} \tag{4.8}$$
$$w_{joint_n} = A_j \bar{w}_{joint_n} \tag{4.9}$$
$$\dot{u}_{joint_n} = \omega_j \times u_{joint_n} \tag{4.10}$$
$$\dot{v}_{joint_n} = \omega_j \times v_{joint_n} \tag{4.11}$$
$$\dot{w}_{joint_n} = \omega_j \times w_{joint_n} \tag{4.12}$$

where r_g and $\dot{r}_g$ are the position and velocity of the center of gravity respectively, $\bar{r}_g$ is the position of the center of gravity in the local coordinate system of the body i, ω is the angular velocity of the body, α_n, β_n and γ_n are the relative coordinates associated to the joint n, u_{joint_n}, v_{joint_n}, w_{joint_n}, $\bar{u}_{joint_n}$, $\bar{v}_{joint_n}$ and $\bar{w}_{joint_n}$ are the vectors representing the direction of the rotation axis of joint n in global and local system of coordinates respectively, $\dot{u}_{joint_n}$, $\dot{v}_{joint_n}$ and $\dot{w}_{joint_n}$ are the first derivatives with respect to the time of the vectors attached to the joint m, and A_{ji} is the rotation matrix of the body i with respect to the body j, which expression for a body with a spherical joint as origin of its local frame corresponds to the rotation matrix derived from the relation of the Euler angles.

It must be remarked that the procedure for other type of joints is similar, although terms as the rotation matrix of the body are different. In addition, the amount of data to be calculated is not the same for all bodies as it depends on the amount of joints that are related to a certain body.

As can be extracted from the equations presented, all the variables related to the body j must be known prior to compute the variables of the body i. In addition, the values of the relative coordinates involved in the joint n must be computed previously, allowing to determine the relative movement of the body i with respect to the body j. As in the previous implementation, the resources and latencies for the function without optimizations for the model of Section 5.2 with 29 bodies and 39 joints is presented in Table 4.7. The data streamed to the FPGA is only the full set of coordinates at position and velocity level, since they are required to start the calculations. The local values of joints and bodies are fixed parameters in the code which is programmed on the FPGA.

Table 4.7: Summary report of the FPGA implementation for updating the motion of each body without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	6674	66	1170	259	1215
Read Input data	1229	0	0	≈0	≈0
Variables Update	5388	65	1170	259	1214
Write Output data	49	0	0	≈0	≈0

The results presented in Table 4.7 show that the calculations for the updating the motion of all the bodies of a vehicle model is not viable to be implemented on an FPGA, since the required resources exceed the available considerably. Thus, barely the calculations referred to a part of the vehicle can be offloaded to the FPGA. At this point, is interesting to implement a mechanism that is repeated in the model, in order to use the implementation programmed on the FPGA more than once and accelerate even more the simulation. Hence, the suspension system is a suitable candidate, since the topology of the mechanism is almost identical for the four suspensions in the modeled vehicle. In other vehicles, the suspension system is at least the same on both sides of the front and rear suspension. Thus, each mechanism is repeated at least twice and implementing them on the FPGA is still a convenient approach. The summary for the implementation of a suspension system is shown in Table 4.8. In this case, not only the values of the relative coordinates are needed to start the calculations. Since the function is designed to be employed with each suspension, the input data to the FPGA includes the local values of the joints and bodies. In addition, the variables reflecting the motion of the chassis are required, since it is the previous body in the chain.

Without any kind of optimization, the resources required for computing the bodies of one suspension are close to the maximum resources available. As a consequence, the possible actions to parallelize the code are limited, as they will increase the demand of resources. Regarding to the implementation, the first step could be to partition the arrays where the input and output data is read and written, following the explanations of the previous implementations. Each body will require a different amount of input data due to the nature of its base joint, as it has associated one

Table 4.8: Summary report of the FPGA implementation for updating the motion of a suspension system without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	3576	6	86	22	95
Read Input data	88	0	0	≈0	≈0
Suspension Variables Update	3194	4	86	21	94
Write Output data	288	0	0	≈0	≈0

coordinate per DOF allowed and, therefore, there is not an ideal size for partitioning the input. The same situation is repeated with the output, as each body can have different number and types of joints attached, hence the number of variables computed will not match between bodies. Thus, a full partition of the input and output data is tested.

Moreover, the data dependencies of the implementation of this function arise if Equations 4.1-4.12 are analyzed. First, the rotation matrix of the body i must be computed, requiring A_j to be known. Later, r_g, ω_i, r_{joint_n} and the director vectors of the joint m can be computed in parallel. In a final step, the terms related to the velocities can be obtained. Hence, there is a minimum of three sequential steps that must be followed (Figure 4.5).

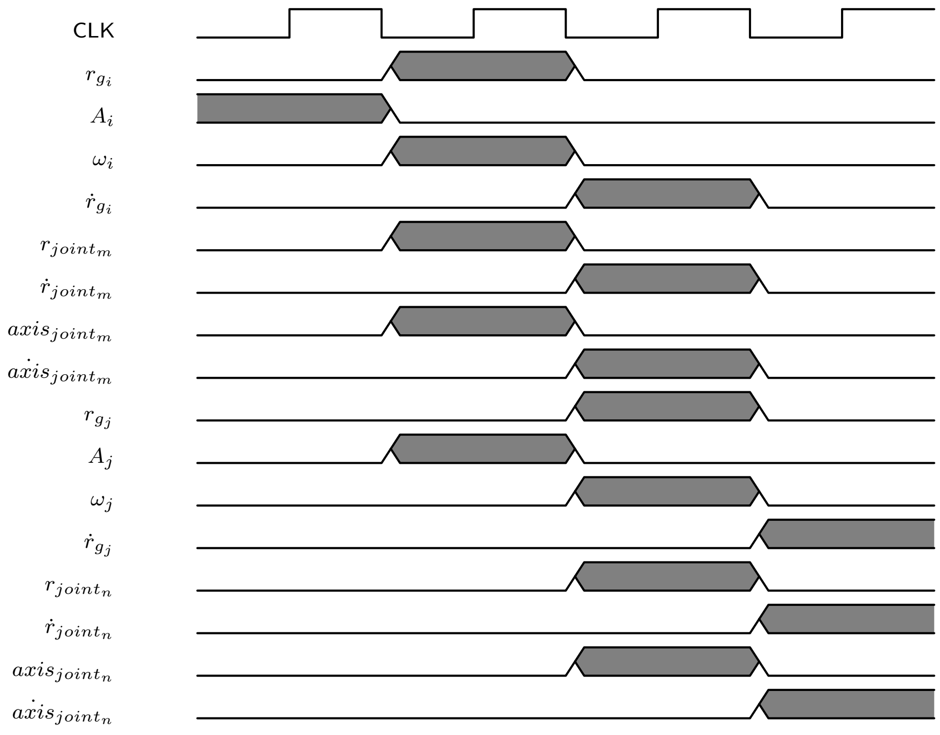


Figure 4.5: Diagram representing the pipelining process followed for updating the motion variables of two consecutive bodies, i and j, with their respective joints, m and n.

There is also a data dependency between different bodies, as can be concluded from the from Equations 4.1-4.12. The use of the unrolling strategy is therefore non convenient, since the amount of operations that can be executed at the same instant is reduced. Furthermore, the resources that can be disposed to further parallelization are scarce. On the other hand, pipelining the code could allow to reduce the clock cycles required for start the calculations of each body. For example, as soon as the rotation matrix A_j is computed, the rotation matrix A_i can be calculated and r_g, r_{joint_n}, u_{joint_n}, v_{joint_n} and w_{joint_n} can be obtained. The resultant implementation applying a pipelining to the calculations performed is shown in Table 4.9.

Table 4.9: Summary report of the FPGA implementation for updating the motion of a suspension system with a full array partitioning of the input and output arrays and a pipeline of the calculations.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	733	2	66	55	95
Read Input data	176	0	0	≈0	5
Suspension Variables Update	407	2	66	52	90
Write Output data	144	0	0	3	≈0

When pipelining the design, the latency of the function is reduced. Furthermore, the amount of DSP are considerably reduced, while the number of employed FF increases, considering that now the input and output arrays are partitioned completely. With the pipelining and partitioning directives, the software is able to reorganize the code implemented and achieve a better solution in terms of resources and latency. The achieved FPGA frequency for this implementation is of 125 MHz. This approach is tested in Section 5.2, where the performance of the implementation is measured during a simulation.

4.3.3 Function `solve_system()`

In order to solve the system of equations presented at Equation 3.50, a mathematical algorithm for solving a system of equations from the type $\mathbf{Ax} = \mathbf{b}$ must be implemented, where $\mathbf{A}$ corresponds to the term derived in Equation 3.53, $\mathbf{b}$ is the residual presented in Equation 3.52 and $\mathbf{x}$ are the increments on the MB positions of the relative coordinates. Solving this kind of systems is a common problem in more fields than MB simulations and, therefore, there are available on the literature several studies on which method is more efficient to be implemented on an FPGA.

The most common methods employed are the LU factorization, QR factorization, Cholesky factorization and Gauss-Jordan (GJ) elimination among others [84]. Algorithms based on decomposition methods such as LU, QR and Cholesky, present a high complexity for their hardware implementation [85]. Meanwhile, the Gauss-Jordan elimination is the most parallel approach. Although in software solutions it involves more arithmetic operations, it is more efficient on parallel processing hardware. The elimination operations can be performed in parallel without increasing the processing time [84]. In Table 4.10, the details of the GJ FPGA implementation without any optimization are presented.

Table 4.10: Summary report of the FPGA implementation for the Gauss-Jordan algorithm employed during the integration of the equations of motion without any optimization.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	930247	5	4	2	7
Read Input data	1892	0	0	≈0	≈0
Gauss-Jordan	928270	3	4	2	7
Write Output data	85	0	0	≈0	≈0

To optimize the design of the Gauss-Jordan algorithm on FPGAs, the implementation proposed by J.P. David in [86] is selected. In the GJ algorithm, it is required to perform several divisions. This kind of operations are costly in FPGAs. The approach proposed in [86] reduces the number of divisions performed. In addition, the division is replaced by an approximation based on floating-point arithmetic. Following the details presented in [86], the implementation presented in Table 4.11 is achieved. However, the resources required are higher than the resources available on the FPGA of the Zynq-7000.

Table 4.11: Summary report of the FPGA implementation for the Gauss-Jordan algorithm employed during the integration of the equations of motion following the approach presented in [86].

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	12515	47	156	32	150
Read Input data	1806	0	0	1	2
Gauss-Jordan	10667	32	156	30	147
Write Output data	42	0	0	1	1

In order to obtain a efficient implementation which fits on the FPGA employed in this thesis, the optimization of the algorithm should be reduced. This approach is presented in Table 4.12. Although the optimization of this solution is limited by the resources of the FPGA, it is still much more efficient than the initial implementation presented in Table 4.10. The solver of equations can be executed at a frequency of 100 MHz in the FPGA of this work.

Table 4.12: Summary report of the FPGA implementation for the Gauss-Jordan algorithm employed during the integration of the equations of motion following the approach presented in [86] with a partial partitioning of the input/output arrays.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	18276	31	54	18	95
Read Input data	1806	0	0	1	2
Gauss-Jordan	16428	21	54	16	92
Write Output data	42	0	0	1	1

4.4 Summary

In-vehicle hardware has been evolving in the last years in order to increment the computational power on-board, satisfying the demand of ADAS and other technologies. New generation devices are mainly based on heterogeneous processors where a GPP is combined with a GPU or an FPGA as co-processors. The main purpose of the co-processors is to optimize the most demanding tasks of the application, reducing the overall time consumed. This can be achieved thanks to the nature of the GPUs and FPGAs, where a high level of acceleration can be achieved through different techniques. This new processors can provide enough computational power in-vehicle for implementing virtual sensors based on MB models in real time.

In Section 4.1, an overview of the main trends in modern hardware was performed. From the options available, an heterogeneous processor with an FPGA was selected for the work of this thesis. As reviewed, the presence of FPGAs in industrial applications is increasing due to their low energy consumption. Thus, it is of interest to study the benefits that FPGAs can offer for accelerating MB simulations.

In order to optimally use the FPGA, the code of the state observer based on a MB vehicle model was profiled. The analysis shows that three functions stand out as the most time consuming tasks: updating the mass matrices of the model, updating the variables which represent the motion of the vehicle and solving the system of equations for determining the increment of the model coordinates each time step. During Section 4.3, multiple FPGA implementations are studied. From the examples shown, some guidelines for accelerating MB simulations with FPGAs were presented. In order to exploit the potential of FPGAs as hardware accelerator, the data dependencies and parallel opportunities of each tasks should be analyzed. Later, the final solution should be adapted to the resources available on the FPGA. As it was shown, a parallel approach offers a significant reduction of computational load of each function in terms of clock cycles.

In Chapter 5, the implementations proposed in this chapter are tested when applied to the simulation of MB-based state observers. The benefits of each implementation in terms of acceleration are also evaluated.

Chapter 5

Use Case: Automotive Application

As stated in previous chapters, the goal of this thesis is to provide virtual sensors based on MB models for in-vehicle applications. The main requisite for such purpose is to be able to execute the MB model of the vehicle combined with a state estimator in real time on automotive platforms, which are based on embedded hardware. The estimations must also reach a high level of accuracy.

During this work, a suitable formulation for efficient MB simulations has been selected, together with a state estimator which offers accuracy in the estimations without penalizing excessively the simulation time. Also, a new embedded processor based on heterogeneous platforms has been selected. This device, from the Zynq-7000 series of Xilinx®, has an ARM-based main processor combined with an FPGA as co-processor. It offers an increment in computational capabilities with respect to classical embedded processors. The techniques and implementations for exploiting the benefits of FPGAs have also been discussed in Chapter 4.

In this chapter, the results of the MB-based state estimator are presented based on a MB model of a real vehicle. The sensors included in the simulation are similar to the ones installed in real vehicles. The virtual measurements obtained during a simulated maneuver, together with the achieved simulation times are discussed.

5.1 Methodology

The implementations presented in this thesis are evaluated when applied to the MB model of the vehicle presented in Figure 5.1. It is a rear-wheel drive electric vehicle with four independent suspension systems, based on a double-wishbone topology. It is instrumented with multiple sensors, such as a GPS, accelerometers, gyroscopes, suspension displacement sensors, and wheel angle sensors. Furthermore, the steering wheel angle and engine torque are also measured.

The observer is designed in a simulation environment in order to perform as many maneuvers as needed, avoiding the long development time and costs that are associated with tests in the real world. In this virtual framework, three MB models are employed.

The first model replaces the physical prototype of the vehicle. It is referred as *real vehicle* and it is used as the *ground truth*. The sensors included in the model are the same as the sensors installed in the physical prototype. The sensor models are

Figure 5.1: Picture of the vehicle selected for this work, the SimRod [87].

built from the kinematics of this model. White noise is added to represent the noise properties of real sensors.

The second model represents the *model* of the *real vehicle*. In a real situation, there would be modeling errors. When building a MB model, usually the geometry can be accurately determined, but the force models have much more uncertainty. Since this work is developed in a simulation environment, errors in the force models of the *model* must be introduced. From the modeling parameters, the mass and friction coefficient are parameters prone to vary within maneuvers in real situations. Thus, the modeling errors are introduced in both parameters. This creates a drift between the *real vehicle* and its *model* similar to the existent in a real application.

The third model is the one that will be executed in combination with the estimator. It will be named *observer*. The MB model used for the *observer* is the same as the *model*. The state estimator is supposed to correct the modeling errors with the information provided by the sensors installed on the *real vehicle*.

In order to asses the performance of the proposed algorithm, different maneuvers over a flat ground are tested. The steering angle and engine torque signals are shared between the *real vehicle*, the *model* and the *observer*. The sensor measurements gathered from the *real vehicle* are fed to the *observer* in order to perform the required corrections. The results of each simulation, in terms of tire forces and sensor measurements, are compared. To validate the performance of the *observer*, the estimated magnitudes must be similar to their respective values in the *real vehicle*. The results of the *model* are used to show the drift that would have been in the results if any correction were applied.

As stated in Chapter 4, the target platform employed in this thesis embeds an ARM processor with an FPGA as co-processor. In order to evaluate the computational cost of implementing the observer on this platform, simulations with each of the implementations presented during Section 4.3 are executed. Different time steps are tested in order to determine the lower time step that can be selected while keeping real-time execution.

5.2 Complete Vehicle Model

The modeling of the vehicle has been accomplished using the `MBScoder`, presented in Section 3.2.2. The geometry of the suspension system and steering system were obtained from a virtual 3D model of the vehicle. In order to represent the vehicle dynamics with accuracy, not only the MB model of the vehicle is important. Vehicle dynamics are also highly influenced by the interaction tire-road [88] and, thus, an accurate tire model should be employed.

5.2.1 Multibody Modeling

Following the approach presented in Section 3.2.1, the MB model of the vehicle (Figure 5.2) is designed using relative coordinates. Thus, each body is defined with respect its previous body in a tree-like chain.

The initial body of the chain is the chassis frame, considered as a unique rigid body. It involves six DOFs: three translations, represented by three Cartesian coordinates of a point in the front part of the car (x, y, z) and three rotations, expressed as three Euler angles for the inertial frame of reference (α, β, γ). This DOFs are grouped into a special type of joint defined as floating joint [58].

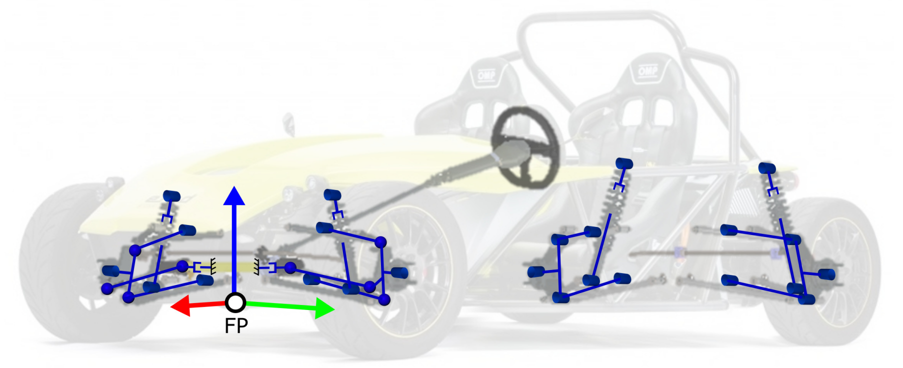


Figure 5.2: MB model of the vehicle, showing the floating joint which represents the DOFs of the chassis frame.

The MB model of the suspension system is presented in Figure 5.3a. Each suspension has three bodies attached to the chassis by three revolute joints (J_1,J_2,J_5), involving three angles representing the relative rotation of each suspension's body with respect to the chassis. Depending on whether the suspension is steerable or not, the joints connecting the knuckle to the control arms differ. For the front suspension system, the joints are spherical (J_3,J_4), allowing the rotation around the steering axis. The knuckle (B_3) of the rear suspension is connected by revolute joints. This adds a total of eight angles (three per each spherical joint and one per each revolute joint) to the set of variables of the MB model. The set spring-damper is represented by a prismatic joint (J_6), which adds one translational DOF. The wheel is represented by a revolute joint (J_8) attached to the knuckle.

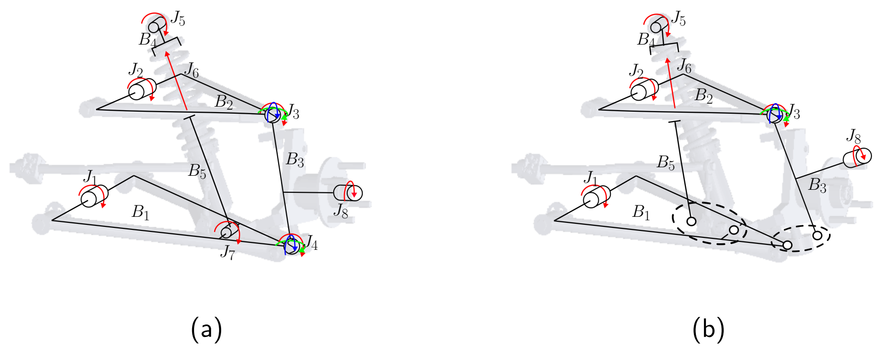


Figure 5.3: MB model of the front suspension system. The arrows represent the DOFs allowed by each joint. In 5.3a, the initial MB model is presented. Figure 5.3b shows the resulting MB model for the suspension system after opening the closed-loops following the approach of the formulation selected in Section 3.2.

As explained in Section 3.2, the closed-loops of the MB model should be opened. In this case, the spherical joint J_4 and the revolute joint J_7 are selected (Figure 5.3b). It should be noted that for opening loops, it is interesting to remove first spherical joints, since they add only three constraint equations (equal position from both sides of the chain). Revolute and prismatic joints add five constraint equations (equal position and director vector of the joint) [58].

The steering system is kinematically guided by the steering input. The steering wheel angle is converted to a rack displacement, which is represented by a prismatic joint. The rack displacement is converted into a rotation of the knuckle through two spherical joints, as can be seen in Figure 5.4a. As in the suspension model, the steering system forms a closed-loop with the suspension system. The spherical joint connecting the knuckle with the steering system is removed in order to obtain an open chain, following the formulation selected in Section 3.2.

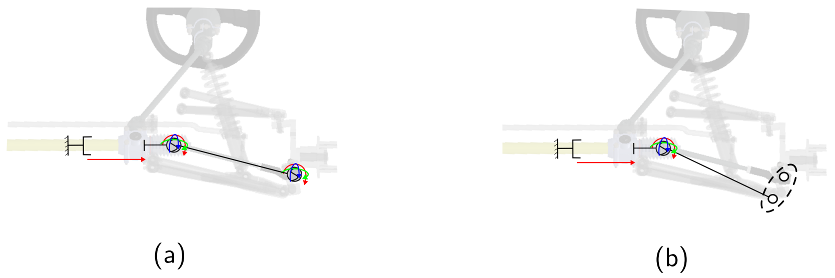


Figure 5.4: MB model of the left side of the steering system. The arrows represent the DOFs allowed by each joint. In 5.4a, the initial MB model is presented. Figure 5.4b shows the resulting MB model for the steering system after opening the closed-loops following the approach of the formulation selected in Section 3.2.

To summarize, the MB model of the vehicle selected involves a total of 42 variables representing the motion between 29 bodies. Due to the loops opened in the system, 42 constraint equations (based on closure-loop conditions) are imposed to ensure that the open-chain model behaves as the real system. Two additional constrain equations are applied for the kinematic guidance of the steering system.

5.2.2 Tire Model

Different types of mathematical tire models have been developed during the last years, each type for a specific applications depending on complexity and the required accuracy. As presented in [88], tire models can be classified in,

1. Simple tire models: The vertical behavior consists of a spring-damper model. The horizontal tire forces are calculated by means of linearization between slip and resulting forces. These tire models are used for static and quasi-static vehicle dynamics analysis and in the design of vehicle control systems.

2. Empirical models: The behavior of the tire is based on non-linear mathematical approximations of tire forces or interpolation of test data. These models are very accurate and can be used for non-linear vehicle dynamics. However, they require a huge amount of parameters related with the tire.

3. Physical models: Through physical and geometric parameters easy to obtain, these models describes the kinematics and dynamics of the tire in detail. Their application is unlimited, although they can be very complex and, often, they are tailored for a specific area of application.

4. Finite-element tire models: The tire is modeled by a detailed finite-element mesh for the complete tire structure including the compressed air. Almost any physical phenomenon can be taken into account. As a counterpart, the computational cost is too high for conventional vehicle dynamics analysis, especially if real-time performance is desired.

In this thesis, the vehicle dynamics are intended to be used together with a state observer in real-time on a vehicle. A compromise between simplicity and accuracy should be achieved in order to represent all the dynamic phenomena without sacrificing real-time performance. Simple tire models do not offer the accuracy required in this work, since non-linear handling cannot be represented. Finite-element tire models entail a huge increment of computational cost, which is already limited by the hardware employed for in-vehicle applications. Both tire models are therefore discarded.

As an alternative, tire models based on empirical data combined with equations representing the physical behavior of the tire are used. The Magic Formula, developed by Pacejka [89–92], is one of the most famous tire model. However, it is based on a huge amount of parameters, which are difficult and costly to obtain. While the tire normal behavior can be assumed as a spring-damper system, the horizontal dynamics are more complex. In [93], the TMeasy is presented as an empirical-physical tire model for low frequency applications in vehicle dynamics. The number of required parameters is reduced according to the limited availability and accuracy of the basic experimental data [93].

5.2.2.1 Normal Tire Forces

The behavior in the normal direction of the tire can be approximated with a system of the type spring-damper [94]. Hence, it is proportional to the normal tire deflection, ξ_z, and normal velocity of the contact point from the tire, $\dot{\xi}_z$. In Equation 5.1, the expression for obtaining the force is presented.

$$F_z = k_z \xi_z + c_z \dot{\xi}_z \tag{5.1}$$

where k and c are the stiffness and damping coefficients of the tire on the normal direction.

The normal deflection can be obtained through the difference between the unloaded wheel radius (including the tire) and the real distance from the wheel centre to the ground. Its velocity can also be derived from the normal velocity of the contact point belonging to the wheel. The force obtained is later limited in such a way that it can only produce compression forces over the road.

5.2.2.2 TMeasy: Generalized Tire Forces

Unlike the normal forces, the tangential tire forces require from more detailed calculations. In general driving situations, the tire suffers deflections due to the relative motion between the road and the tire itself, which is known as slip. Hence, the longitudinal and lateral forces are dependent on their respective slips (Equations 5.2 and 5.3).

$$s_x = \frac{-(v_x - r_D \boldsymbol{\omega})}{r_D |\boldsymbol{\omega}| \hat{s}_x + v_n} \tag{5.2}$$

$$s_y = \frac{-v_y}{r_D |\boldsymbol{\omega}| \hat{s}_y + v_n} \tag{5.3}$$

where s_x and s_y are the longitudinal and lateral slips respectively, v_x and v_y the longitudinal and lateral contact point velocities, r_D the dynamic radius of the wheel, $\boldsymbol{\omega}$ its angular velocity, $\hat{s}_x$ and $\hat{s}_y$ are two weight coefficients used when both effects, longitudinal and lateral, are combined. In this work, both weight coefficients are set to the unit, giving the same relevance to longitudinal and lateral effects. If the wheel is not rotating, $\boldsymbol{\omega} = 0$ and, therefore, a singularity appears on the slip equations. To avoid this singularity, v_n is added as a small fictitious velocity ($\approx 10^{-15}$). Both slips can be combined into the so-called general slip, s_g, as in Equation 5.4.

$$s_g = \sqrt{s_x^2 + s_y^2} \tag{5.4}$$

Once the general slip is known, the total tangential tire force, F, can be obtained through the tire characteristic, which relates the tire forces with the slips. Since the exact tire characteristic of the tire employed in this work is not available, a linear approximation was used from typical values presented in the literature [95]. The tire characteristic is show in Figure 5.5, where μ is the road-tire friction coefficient, F_n is the normal force to the road, and $s_{g,c}$ is the critical slip.

Hence, from the tire characteristic, the longitudinal and lateral tire forces, F_x and F_y respectively, can be calculated as,

$$F_x = F \frac{s_x}{s_g} \tag{5.5}$$

$$F_y = F \frac{s_y}{s_g} \tag{5.6}$$

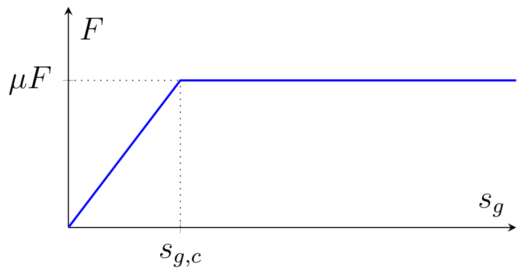


Figure 5.5: Linearised tire characteristic employed in this thesis.

5.2.2.3 TMeasy: Non-linear First Order Tire Dynamics

The tire forces derived from Equation 5.5 are referred to the steady-state forces. When the forces vary, the deflection of the tire must be considered. Otherwise, the slips introduced can lead to unrealistic tire forces, especially when the slips are high [4]. Determining the tire deflection is the problem addressed by tire models, such as the TMeasy.

Hirschberg *et al.* propose in [93] the non-linear first order tire dynamics approximation presented in Equations 5.7 and 5.8.

$$\underbrace{F_x(v_x + \dot{\xi}_x)}_{F_x^D} \approx \underbrace{F_x(v_x)}_{F_x^S} + \frac{\partial F_x}{\partial v_x}\dot{\xi}_x \tag{5.7}$$

$$\underbrace{F_y(v_y + \dot{\xi}_y)}_{F_y^D} \approx \underbrace{F_y(v_y)}_{F_y^S} + \frac{\partial F_y}{\partial v_y}\dot{\xi}_y \tag{5.8}$$

where v_x and v_y are the longitudinal and lateral velocities of the contact point, F_x^D and F_y^D are the dynamic tire forces and F_x^S and F_y^S are the steady-state tire forces.

In order to derive the expressions for the tire deflections, the Equations 5.7 and 5.8 are compared with their respective equations assuming that the tire can be considered as a spring-damper element, shown in Equations 5.9 and 5.10.

$$F_x^D = k_x\xi_x + c_x\dot{\xi}_x \tag{5.9}$$

$$F_y^D = k_y\xi_y + c_y\dot{\xi}_y \tag{5.10}$$

where k and c are the stiffness and damping coefficients of the tire on each direction respectively, F are the tire forces and ξ is the tire deflection, which should be determined. Hence, comparing the non-linear first order approximation with the spring-damper force expression yields to the differential Equations 5.11 and 5.12, which solution are the longitudinal and lateral deflections.

$$\left(v_x^* d_x + \frac{\partial F_x^S}{\partial s_x}\right)\dot{\xi}_x = -\frac{F}{s_g}(v_x - r_D\boldsymbol{\omega}) - v_x^* c_x \xi_x \tag{5.11}$$

$$\left(v_y^* d_y + \frac{\partial F_y^S}{\partial s_y}\right)\dot{\xi}_y = -\frac{F}{s_g}v_y - v_y^* c_y \xi_y \tag{5.12}$$

where $v_x^* = r_D|\boldsymbol{\omega}|\hat{s}_x + v_n$, $v_y^* = r_D|\boldsymbol{\omega}|\hat{s}_y + v_n$. The longitudinal and lateral deflections are obtained through the integration of the differential Equations 5.11 and 5.12 each time step. The solution to both differential equations are can be found in [4].

5.2.2.4 State observer: Tire Forces Distribution

The state observer presented in Section 3.3 assumes that the required corrections in forces, $\Delta\hat{\mathbf{Q}}$, should be applied on the degrees of freedom of the MB model. In some mechanisms, this assumption can bring to an accurate solution. In the case of vehicle dynamics, use-case of study in this thesis, the main forces acting over the longitudinal and lateral vehicle behavior come from the tires. Applying the force increments, $\Delta\hat{\mathbf{Q}}$, over the chassis could result into undesired tire reactions, which could lead to inaccurate tire forces estimations. To overcome this issue, it is proposed to transform the elements of $\Delta\hat{\mathbf{Q}}$ related to the chassis into tire forces increments, as illustrated in Figure 5.6.

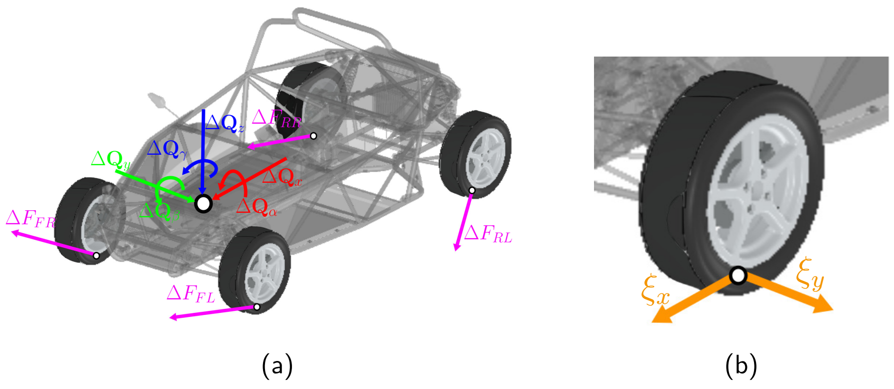


Figure 5.6: Mapping to the tires of the forces increment applied in the chassis by the errorEKF.

However, there are multiple tire forces combinations which can produce the desired effects on the chassis. In this thesis, the selected criteria is to minimize the tire forces increments. This leads to a least square problem such as,

$$\mathbf{\Delta F}_{FL} + \mathbf{\Delta F}_{FR} + \mathbf{\Delta F}_{RL} + \mathbf{\Delta F}_{RR} = \Delta\hat{\mathbf{Q}}_{xyz} \tag{5.13}$$

$$\mathbf{r}_{FL} \times \mathbf{\Delta F}_{FL} + \mathbf{r}_{FR} \times \mathbf{\Delta F}_{FR} + \mathbf{r}_{RL} \times \mathbf{\Delta F}_{RL} + \mathbf{r}_{RR} \times \mathbf{\Delta F}_{RR} = \Delta\hat{\mathbf{Q}}_{\alpha\beta\gamma} \tag{5.14}$$

Since the tire forces are related with the tire deformation (Equations 5.9, 5.10, 5.1), it is required to transform tire forces increments obtained into tire deformations. Projecting each $\mathbf{\Delta F}$ over the tire axis (Figure 5.6b) results in the required increments on longitudinal and lateral tire forces for correcting the longitudinal and lateral dynamic states of the vehicle. Considering that the tire deformation, ξ, is linear,

$$\hat{\xi}_x = \xi_x + \frac{\Delta F_x}{k_x} \tag{5.15}$$

$$\hat{\xi}_y = \xi_y + \frac{\Delta F_y}{k_y} \tag{5.16}$$

where ξ is the estimated tire deflection.

As a last step, the force models of the vehicle are updated with the estimated tire states. Since some terms of $\Delta\hat{\mathbf{Q}}$ have been derived into tire forces, the final correction in forces is obtained after subtracting the resulting tire forces. The terms related with the longitudinal and lateral displacements, as well as the yaw, are almost null since they have been distributed to the tires deformation.

5.2.3 Sensor Models

The sensors employed in this work were already introduced in Section 5.1. Some of the measurements, such as the steering wheel angle and engine torque, are inputs to the MB model. The rest of the sensors, however, are the observations employed by the filter for deriving the corrections, following Equations 3.64-3.68. Therefore, they must be modeled in order to obtain the respective measurements from the relative coordinates employed in the MB modeling. The sensor models are presented hereafter, together with their Jacobian with respect to $\mathbf{z}^i$,$\dot{\mathbf{z}}^i$ and $\ddot{\mathbf{z}}^i$, which are the variables forming the state vector (Equation 3.57). The Jacobian of each sensor is required for the filter corrections, as shown in Equation 3.65 and 3.66. The noise of each sensor is included in the measurement noise matrix of the filer (Section 3.4).

5.2.3.1 GPS Positions

The chassis of the vehicle, as explained in Section 5.2.1, is modeled by using its Cartesian coordinates and three Euler angles. Since the GPS can be placed in any part of the chassis frame, a general expression for the measurements is presented in Equation 5.17.

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \mathbf{r}_{gps} = \mathbf{r}_{ch} + \mathbf{A}_{ch}\bar{\mathbf{r}}_{gps} \tag{5.17}$$

where $\mathbf{r}_{gps}$ and $\bar{\mathbf{r}}_{gps}$ are the absolute and local (with respect to the chassis) position of the GPS respectively, $\mathbf{r}_{ch}$ are the Cartesian coordinates of the origin of the chassis, and $\mathbf{A}_{ch}$ is the rotation matrix of the chassis, related with the Euler angles, given by Equation 5.18.

$$\begin{gathered} A_{ch} = A^Z_{ch}A^Y_{ch}A^X_{ch} \\ = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} \end{gathered} \tag{5.18}$$

where α, β and γ are the Euler angles around the X, Y and Z axis of the absolute system frame respectively.

Then, the Jacobian is derived through the partial derivatives of Equation 5.17 with respect to $\mathbf{z}^i$,$\dot{\mathbf{z}}^i$ and $\ddot{\mathbf{z}}^i$, as presented in the next Equations.

$$\mathbf{h}_{\mathbf{z}} = \frac{\partial \mathbf{r}_{gps}}{\partial \mathbf{z}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{I}_3 & \frac{\partial(\mathbf{A}_{ch}\bar{\mathbf{r}}_{gps})}{\partial\alpha} & \frac{\partial(\mathbf{A}_{ch}\bar{\mathbf{r}}_{gps})}{\partial\beta} & \frac{\partial(\mathbf{A}_{ch}\bar{\mathbf{r}}_{gps})}{\partial\gamma} & ... & \mathbf{0} \end{bmatrix} \tag{5.19}$$

$$\mathbf{h}_{\dot{\mathbf{z}}} = \frac{\partial \mathbf{r}_{gps}}{\partial \dot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & ... & \mathbf{0} \end{bmatrix} \tag{5.20}$$

$$\mathbf{h}_{\ddot{\mathbf{z}}} = \frac{\partial \mathbf{r}_{gps}}{\partial \ddot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & ... & \mathbf{0} \end{bmatrix} \tag{5.21}$$

However, since the desired effect of the GPS is to correct only the positions, the terms of Equation 5.19 related with the Euler angles can be neglected, avoiding also undesired effects due to additional orientation corrections.

5.2.3.2 GPS Velocities

In addition to the positions, the GPS provides also accurate information of the vehicle speed. Velocity measurements are obtained by measuring the frequency of the signals received from the satellites. These frequencies vary when the relative velocities between the receiver and the satellites change due to the Doppler effect. Velocity measurements are usually more accurate than position measurements in single antenna GPSs.

From the GPS, only the speed measurement in the plane is obtained, together with the angle of the trajectory with respect to the north. This measurements can be easily converted to Cartesian coordinates, providing x and y components of the velocity. In Equation 5.22, the expression for relating the velocity of the GPS position the velocity of the origin of the chassis, which are part of the state vector, is presented.

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \dot{\mathbf{r}}_{gps} = \begin{bmatrix} \dot{x}_{ch} \\ \dot{y}_{ch} \end{bmatrix} + (\boldsymbol{\omega}_{ch} \times \bar{\mathbf{r}}_{gps})_{xy} \tag{5.22}$$

being $\boldsymbol{\omega}_{ch}$ the angular speed of the chassis, expressed in Equation 5.23 for the sequence of Euler angles selected in Equation 5.18, and $(\boldsymbol{\omega}_{ch} \times \bar{\mathbf{r}}_{gps})_{xy}$ the components of the velocity of the point due to the angular velocity of the chassis in the XY plane.

$$\boldsymbol{\omega}_{ch} = \begin{bmatrix} \dot{\alpha} cos\beta cos\gamma - \dot{\beta} sin\gamma \\ \dot{\alpha} cos\beta sin\gamma + \dot{\beta} cos\gamma \\ \dot{\gamma} - \dot{\alpha} sin\beta \end{bmatrix} \tag{5.23}$$

Deriving Equation 5.22 with respect to $\mathbf{z}^i$,$\dot{\mathbf{z}}^i$ and $\ddot{\mathbf{z}}^i$, the Jacobian of the GPS velocities measurements is obtained.

$$\mathbf{h_z} = \frac{\partial \dot{\mathbf{r}}_{gps}}{\partial \mathbf{z}^i} = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0}_{2\times 3} & \frac{\partial \dot{r}_{gps}}{\partial \alpha} & \frac{\partial \dot{r}_{gps}}{\partial \beta} & \frac{\partial \dot{r}_{gps}}{\partial \gamma} & \dots & \mathbf{0} \end{bmatrix} \tag{5.24}$$

$$\mathbf{h_{\dot{z}}} = \frac{\partial \dot{\mathbf{r}}_{gps}}{\partial \dot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{I}_2 & \mathbf{0}_{2\times 1} & \frac{\partial \dot{r}_{gps}}{\partial \dot{\alpha}} & \frac{\partial \dot{r}_{gps}}{\partial \dot{\beta}} & \frac{\partial \dot{r}_{gps}}{\partial \dot{\gamma}} & \dots & \mathbf{0} \end{bmatrix} \tag{5.25}$$

$$\mathbf{h_{\ddot{z}}} = \frac{\partial \dot{\mathbf{r}}_{gps}}{\partial \ddot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & \dots & \mathbf{0}_{2\times 3} & \mathbf{0}_{2\times 3} & \mathbf{0}_{2\times 3} & \mathbf{0}_{2\times 3} & \dots & \mathbf{0} \end{bmatrix} \tag{5.26}$$

5.2.3.3 Accelerometer

The observer considered in this thesis includes accelerations as part of its states. The acceleration measurements of the chassis are, therefore, important for the correction of the states. It should be noted that the accelerometers cannot measure gravity acceleration [96] and its effects are included in every measurement. As the magnitude and orientation of the gravity are known, some orientation information can be extracted from the acceleration measurements. This is useful to stabilize the magnitudes related to the roll and pitch motions [4].

The accelerometer measurements of an arbitrary point of the chassis in where the accelerometer can be placed are given by Equation 5.27. Note that the gravity vector, $\mathbf{g}$, is added in order to represent in the sensor model the real behavior of an accelerometer.

$$\mathbf{a}_{acc}^{global} = \mathbf{a}_{ch} + \boldsymbol{\alpha}_{ch} \times (\mathbf{r}_{acc} - \mathbf{r}_{ch}) + \boldsymbol{\omega}_{ch} \times [\boldsymbol{\omega}_{ch} \times (\mathbf{r}_{acc} - \mathbf{r}_{ch})] - \mathbf{g} \tag{5.27}$$

being $\mathbf{a}_{acc}$ the acceleration of the point where the accelerometer is placed, $\mathbf{a}_{ch}$ is the acceleration of the origin of the chassis (coincident with the respective variables in the state vector), $\boldsymbol{\alpha}_{ch}$ is the angular acceleration of the chassis, and $\mathbf{r}_{acc}$ is the position in absolute coordinates of the accelerometer. However, the acceleration given by the accelerometer is expressed in its local axis. The final expression for the acceleration measurements is given, therefore, in Equation 5.28.

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \mathbf{a}_{acc} = \mathbf{A}_{ch}\mathbf{a}_{acc}^{global} \tag{5.28}$$

If the axis of the accelerometer are not aligned with axis of the local reference system of the chassis, an additional multiplication should be made by the rotation matrix between the accelerometer and the chassis.

The Jacobian terms of the accelerometer with respect to the states are obtained by partial derivation of Equation 5.28, yielding,

$$\mathbf{h}_{\mathbf{z}} = \frac{\partial \mathbf{a}_{acc}}{\partial \mathbf{z}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{0} & \frac{\partial \mathbf{a}_{acc}}{\partial \alpha} & \frac{\partial \mathbf{a}_{acc}}{\partial \beta} & \frac{\partial \mathbf{a}_{acc}}{\partial \gamma} & ... & \mathbf{0} \end{bmatrix} \tag{5.29}$$

$$\mathbf{h}_{\dot{\mathbf{z}}} = \frac{\partial \mathbf{a}_{acc}}{\partial \dot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{0} & \frac{\partial \mathbf{a}_{acc}}{\partial \dot{\alpha}} & \frac{\partial \mathbf{a}_{acc}}{\partial \dot{\beta}} & \frac{\partial \mathbf{a}_{acc}}{\partial \dot{\gamma}} & ... & \mathbf{0} \end{bmatrix} \tag{5.30}$$

$$\mathbf{h}_{\ddot{\mathbf{z}}} = \frac{\partial \mathbf{a}_{acc}}{\partial \ddot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{I}_3 & \frac{\partial \mathbf{a}_{acc}}{\partial \ddot{\alpha}} & \frac{\partial \mathbf{a}_{acc}}{\partial \ddot{\beta}} & \frac{\partial \mathbf{a}_{acc}}{\partial \ddot{\gamma}} & ... & \mathbf{0} \end{bmatrix} \tag{5.31}$$

5.2.3.4 Gyroscope

The gyroscope is a sensor which provides information of the angular rates of the body where it is installed, which in this case corresponds to the chassis. The measurements of the gyroscope are equivalent to the angular velocity of the chassis, presented in Equation 5.23. Therefore,

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \boldsymbol{\omega}_{gyro} = \boldsymbol{\omega}_{ch} \tag{5.32}$$

As for the accelerometer, if the axis of the local system frame of the gyroscope are not aligned with axis of the chassis, a transformation should be applied. The angular velocity of the chassis should be multiplied by the rotation matrix representing the orientation of the gyroscope in the chassis. The Jacobian terms for the observations,

$$\mathbf{h}_{\mathbf{z}} = \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \mathbf{z}^i} = \begin{bmatrix} \mathbf{0} & ... & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \alpha} & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \beta} & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \gamma} & ... & \mathbf{0} \end{bmatrix} \tag{5.33}$$

$$\mathbf{h}_{\dot{\mathbf{z}}} = \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \dot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \dot{\alpha}} & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \dot{\beta}} & \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \dot{\gamma}} & ... & \mathbf{0} \end{bmatrix} \tag{5.34}$$

$$\mathbf{h}_{\ddot{\mathbf{z}}} = \frac{\partial \boldsymbol{\omega}_{gyro}}{\partial \ddot{\mathbf{z}}^i} = \begin{bmatrix} \mathbf{0} & ... & \mathbf{0} & \mathbf{0} & \mathbf{0} & ... & \mathbf{0} \end{bmatrix} \tag{5.35}$$

5.2.3.5 Suspension Displacement

The displacement of each suspension system is measured with wire sensors. These sensors measure linear movement and displacement through a flexible steel cable. The cable is rolled inside a drum, where a sensor (such as an encoder) provides a proportional output signal. The distance measured by the wire sensor corresponds with the spring compression, which is also a variable of the MB model.

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \left[\begin{array}{cccc} \bar{z}_{FL} & \bar{z}_{FR} & \bar{z}_{RL} & \bar{z}_{RR} \end{array}\right] \tag{5.36}$$

where $\bar{z}$ is the coordinate associated to the suspension deflection.

The Jacobian of the suspension measurements are straightforward to obtain,

$$\mathbf{h}_{\mathbf{z}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{I}_4 & ... & \mathbf{0} \end{array}\right] \tag{5.37}$$

$$\mathbf{h}_{\dot{\mathbf{z}}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{0} & ... & \mathbf{0} \end{array}\right] \tag{5.38}$$

$$\mathbf{h}_{\ddot{\mathbf{z}}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{0} & ... & \mathbf{0} \end{array}\right] \tag{5.39}$$

5.2.3.6 Wheel Angles

For testing purposes, the studied vehicle is instrumented with wheel-force transducers which also measures the rotation angle of the wheel. However, they can be replaced by Hall-effect sensors, for example, which are indeed less expensive. As for the suspension measurements, the magnitudes gathered by these sensors correspond directly with the variables employed in the vehicle modeling.

$$\mathbf{h}(\mathbf{z}^i, \dot{\mathbf{z}}^i, \ddot{\mathbf{z}}^i) = \left[\begin{array}{cccc} \varphi_{FL} & \varphi_{FR} & \varphi_{RL} & \varphi_{RR} \end{array}\right] \tag{5.40}$$

where φ is the coordinate associated to the wheel angle.

The expressions for the Jacobian are trivial to obtain,

$$\mathbf{h}_{\mathbf{z}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{I}_4 & ... & \mathbf{0} \end{array}\right] \tag{5.41}$$

$$\mathbf{h}_{\dot{\mathbf{z}}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{0} & ... & \mathbf{0} \end{array}\right] \tag{5.42}$$

$$\mathbf{h}_{\ddot{\mathbf{z}}} = \left[\begin{array}{ccccc} \mathbf{0} & ... & \mathbf{0} & ... & \mathbf{0} \end{array}\right] \tag{5.43}$$

5.2.4 Results

The vehicle model presented is combined with the errorEKF introduced in Section 3.3. Prior to implement the *observer* in the embedded hardware, the stability and accuracy of the filter should be tested.

The tuning of the covariance matrix of the filer is performed according to Section 3.4. Since it is an iterative process, the terms of the covariance matrix of the process are derived through multiple simulations in the PC environment, following a trial-error procedure. The noise of the sensors is determined based on real sensor characteristics.

The maneuver tested consists on a turn with a constant value of steering wheel angle and a constant throttle position. The errors introduced in the *model* in order to create a drift with respect to the simulation of the *real vehicle* are in the mass and tire-road friction coefficient. As discussed in Section 5.1, both parameters are prone to change on each driving situation. For this test, the mass of the *model* is set to 700 kg and the tire-road friction coefficient to 1. The *real vehicle* has a mass of 600 kg and a tire-road friction coefficient of 0.8.

It is also important to select a proper time step for the integration of the MB simulation, together with a sufficient tolerance. With respect to the time step, lower values allow to represent with more accuracy all the phenomena that can occur during a maneuver. In addition, low time steps are equivalent to a higher frequency of virtual sensor measurements, which is desired in control applications. However, reducing the time step increases the computational load of the simulation and, thus, real-time performance can be compromised. Regarding the tolerance of the simulation, since the MB model is combined with a state estimator, it is less stringent than when the objective is to accurately forward simulate the vehicle dynamics.

5.2.4.1 Estimation Results

The results of the estimation are presented in this section. First, in order to illustrate the errors caused by the drift existent between the *real vehicle* and the *model*, the paths followed by each model are presented in Figure 5.7. The position measurements derived from the GPS model are also included. It can be seen that the *observer* tracks the path of the *real vehicle*, even with the low accuracy of the GPS measurements.

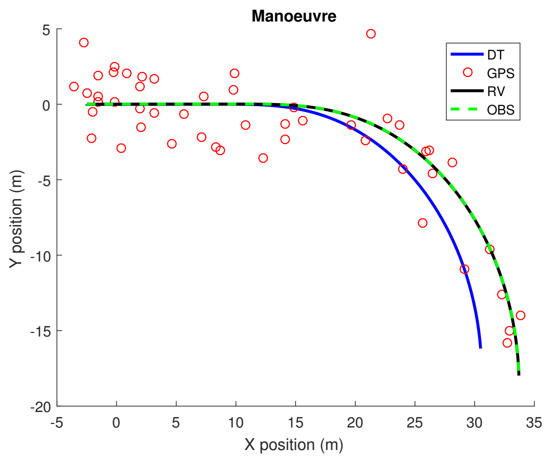


Figure 5.7: Test maneuver performed to test the state observer based on the complete vehicle model. The paths followed by each model and the GPS sensor data are represented.

Regarding the states, the measurements obtained from the *observer* are overlapped with the sensor data gathered from the *real vehicle* for correcting the drift. In addition, the noise of the estimated measurements is reduced with respect to the real measurements. In Figures 5.8-5.12, the estimated speed, accelerations and angular rates of the chassis, plus the suspension deflections and wheel angles are compared with their respective measurements in the *real vehicle*.

Although the measurements provided by the *observer* are similar to the real sensor data, there are some differences. Regarding the lateral acceleration (Figure 5.9), an offset can be appreciated when the vehicle is turning. The *observer* has more lateral acceleration than the *real vehicle*. This is caused by the error in mass: since the *model* has 100 kg more than the *real vehicle*, its accelerations will be higher. The errorEKF is not able to fully correct the drift derived from the error in mass.

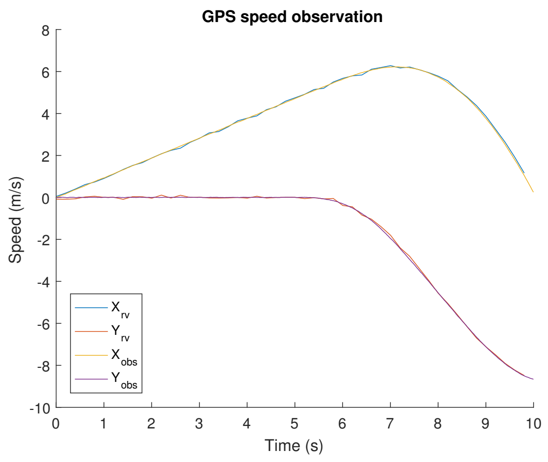


Figure 5.8: Comparison of the speeds measured in the maneuver performed to test the state observer based on the complete vehicle model.

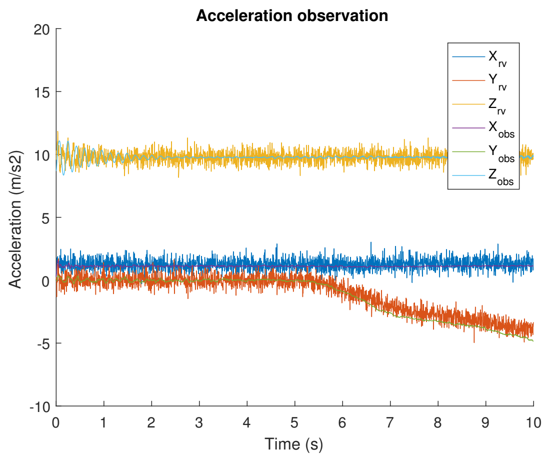


Figure 5.9: Comparison of the acceleration measured in the maneuver performed to test the state observer based on the complete vehicle model.

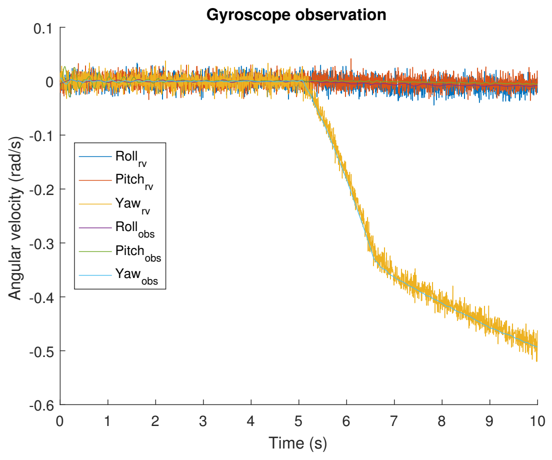


Figure 5.10: Comparison of the angular velocities measured in the maneuver performed to test the state observer based on the complete vehicle model.

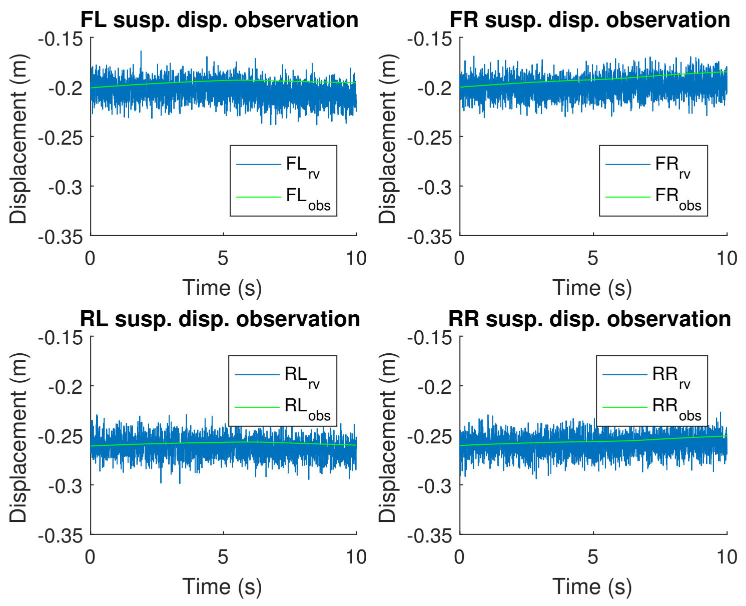


Figure 5.11: Comparison of the suspension deflections measured in the maneuver performed to test the state observer based on the complete vehicle model. The zero suspension deflection corresponds to a full compression state.

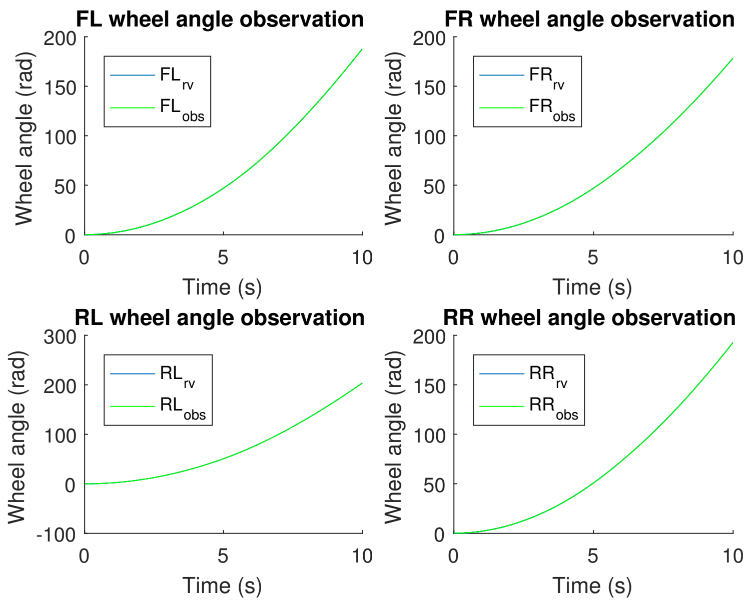


Figure 5.12: Comparison of the wheel angles measured in the maneuver performed to test the state observer based on the complete vehicle model.

The suspension deflections and wheel angle estimations are presented in Figures 5.11 and 5.12 respectively. In both cases, the *observer* tracks the measurements from the *real vehicle* and, in the case of the suspension deflections, reduces substantially the noise of the measurements. The effects of the mass error can also be appreciated in Figures 5.11, where the estimated suspension deflections show that the spring compression is higher in the *observer*.

The results presented with respect to the measurements are promising and show an accurate correction of the drift presented in the *model*. However, the interest of this approach relies on variables that can not be measured through real sensors, such as the tire forces. From Figure 5.13 to Figure 5.16, the tire forces of each wheel are presented.

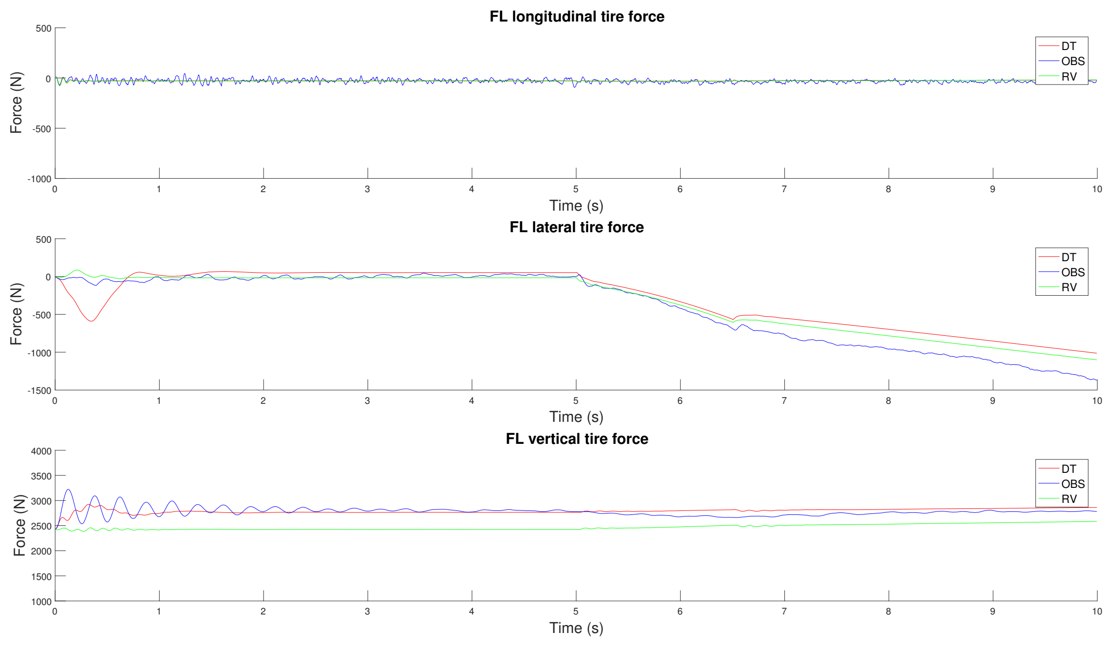


Figure 5.13: Tire forces of the Front-left wheel (FL) wheel estimated by the state observer based on the complete vehicle model in the tested maneuver.

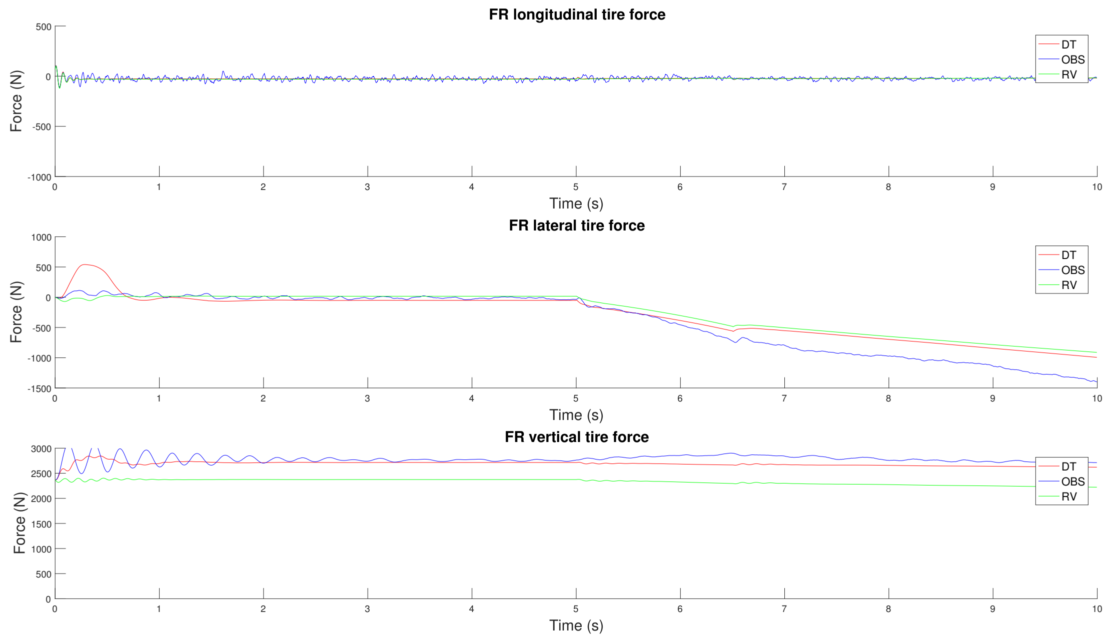


Figure 5.14: Tire forces of the Front-right wheel (FR) wheel estimated by the state observer based on the complete vehicle model in the tested maneuver.

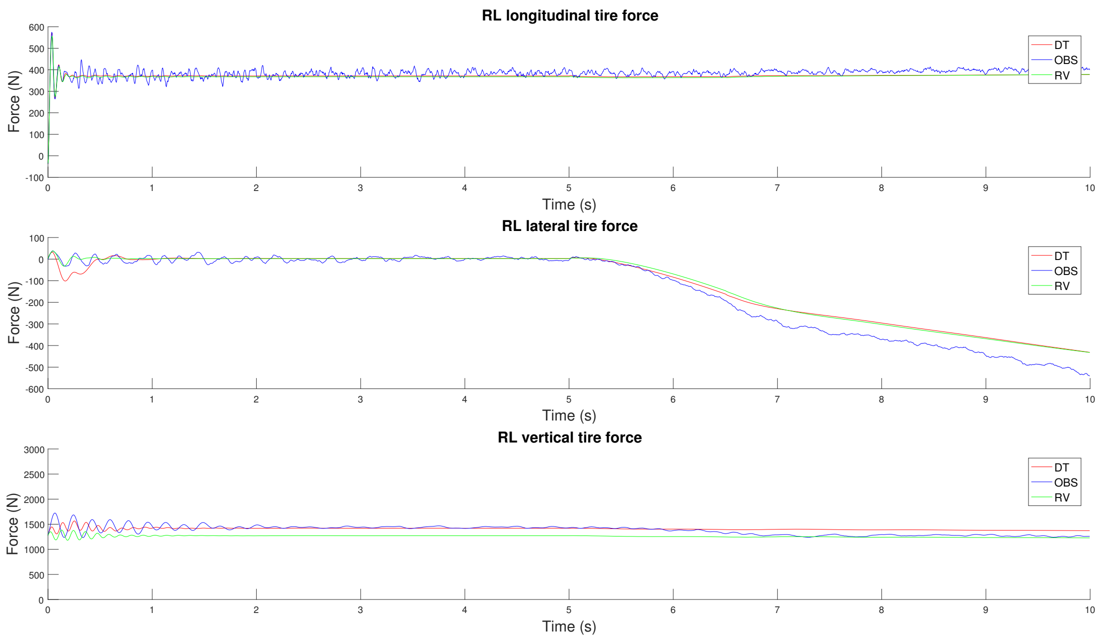


Figure 5.15: Tire forces of the Rear-left wheel (RL) wheel estimated by the state observer based on the complete vehicle model in the tested maneuver.

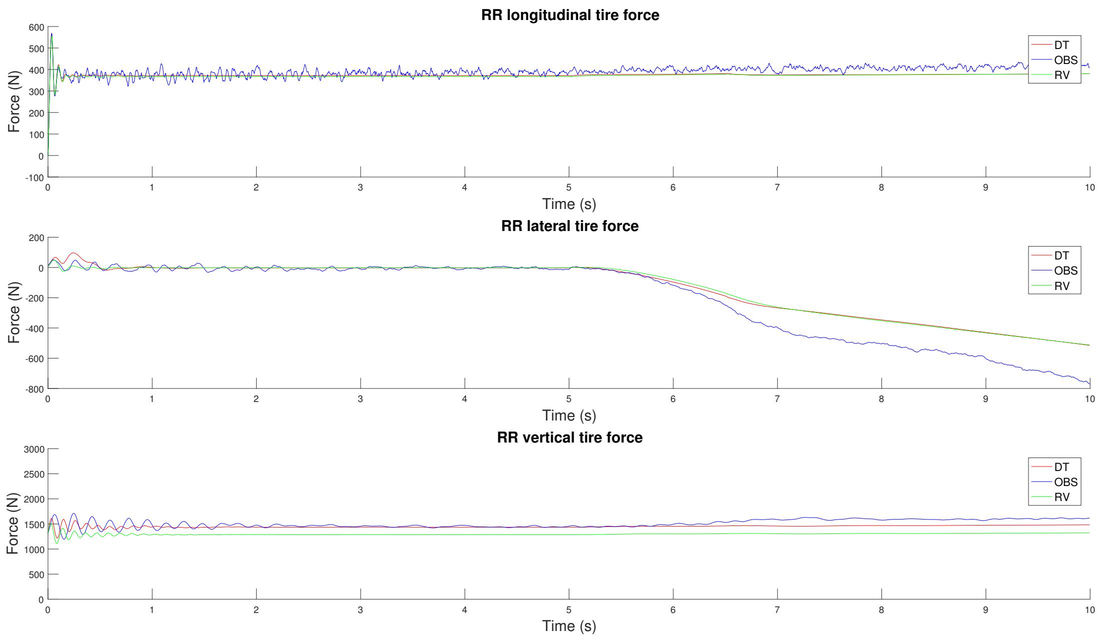


Figure 5.16: Tire forces of the Rear-right wheel (RR) wheel estimated by the state observer based on the complete vehicle model in the tested maneuver.

The results in terms of force estimation are even worse in the *observer* than in the *model*. The errorEKF is not capable to correct the errors derived from a different mass and tire-road friction coefficient. Although the filter should be improved, the implementation of the state observer based on the modeled vehicle is first addressed. If real-time execution is not guaranteed with this approach, alternative solutions should be explored.

The errors in the estimations are also presented from a quantitative point of view in Table 5.1. Through the Root-mean-square error (RMSE), the predicted values can be compared with its real value, giving a better insight on the estimations accuracy. The measurements selected are the position, acceleration and the tire forces in the RR wheel.

Table 5.1: RMSE measured in the maneuvers for testing the errorEKF combined with the complete vehicle model.

Root-mean-square error		
Magnitude	errorEKF	Sensor
Position (m)	0.0046	1.8954
X accel. (m/s^2)	0.1353	0.4372
Y accel. (m/s^2)	0.3625	0.4361
Z accel. (m/s^2)	0.2665	0.4496
RR long. tire force (N)	26.12	-
RR lat. tire force (N)	103.01	-
RR vert. tire force (N)	223.95	-

Although the estimations in position and accelerations are more accurate than the sensor data, the predicted tire forces present a high deviation from their real value, as concluded from Figure 5.16.

5.2.4.2 Implementation Results

The performance of the presented state observer in terms of computational efficiency should be tested. If real-time execution is not guaranteed, the application of the presented approach in real use-cases is not viable.

First, the execution of the state estimator combined with the MB model of the vehicle is launched in the ARM processor of the Zynq 7000 device. For all the simulations, the ARM processor runs at its maximum frequency (667 MHz) and in a bare-metal system, the Xilinx Stand-alone (bare-metal) Environment. Hence, there are no any other processes running at the same time. Only one core is used and the code is compiled under an optimization level 2, meaning that the compiler performs nearly all supported optimizations. The time measurements of this first simulation is considered as a reference value. Later, each of the hardware implementations developed in Section 4.3 is programmed on the FPGA. During each simulation, the ARM is executing all the operations required by the state observer and the MB model, except from the tasks that are implemented on the FPGA. The results in terms of time consumed are compared with the reference simulation in order to evaluate the benefits of each approach.

For this first execution, the initial time step selected is of 4 milliseconds, which corresponds with virtual sensors with a frequency of 250 Hz. If the simulation achieves real-time performance with a wide margin, the time step can be reduced. The results are shown in Table 5.2.

Table 5.2: Summary report of the comparison within the ARM execution of the vehicle MB model and each of the FPGA implementation proposed in Section 4.3.

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference	10	0.004	81.870	9.083	$1 \cdot 10^{-5}$
Reference	10	0.004	38.284	1.419	$1 \cdot 10^{-3}$
GJ FPGA	10	0.004	64.957	8.652	$1 \cdot 10^{-5}$
GJ FPGA	10	0.004	33.459	1.388	$1 \cdot 10^{-3}$
Partial Mass Matrix FPGA	10	0.004	83.622	9.153	$1 \cdot 10^{-5}$
Partial Post-Process FPGA	10	0.004	128.009	16.792	$1 \cdot 10^{-5}$

In all cases, the elapsed time during each simulation is higher than the simulated time. None of the FPGA implementations tested offers enough acceleration to achieve real time. The results show that only the Gauss-Jordan implementation offers a relevant improvement with respect to the simulation on the ARM processor. The evaluation of the individual mass matrices of the bodies does not lead to any acceleration, and the final time is slower than the reference. The worst result is obtained when the post-processing task of each suspension is offloaded to the FPGA. The problem of the numerical precision of FPGAs, commented in Section 4.1, has a strong impact when updating the motion of each suspension system on the FPGA. The simulation requires a higher number of iterations in order to converge each time step, leading to an increase of the computation time.

Although the FPGA implementation of the Gauss-Jordan algorithm allows to reduce the computational time of the simulation, it is not enough for achieving real-time execution. As an alternative, higher time steps can be tested. However, as presented in Table 5.2, the solution is quite far from real time. The time step should be highly increased, but at 8 milliseconds the simulation presents convergence issues. If the tolerance is reduced, the elapsed time is also highly reduced. However, the simulation is still more than three times slower than real time, and the tolerance is already high. Hence, additional options for reducing the computational cost of the simulation should be explored.

5.3 Simplified Vehicle Model

In order to increase the efficiency of the simulation, the vehicle MB model presented in Section 5.2 is simplified following the approach proposed in [97]. In the work presented by Cuadrado *et al.*, the vehicle is modeled with one coordinate for each DOF, and thus independent coordinates, avoiding the definition of constraint equations and reducing the computational cost of the simulation. This section introduces the new MB model developed. With respect to the tire model, the TMeasy referred in Section 5.2.2 is used and the same equations have been employed. The sensor models are also the same as the models developed for the previous vehicle model, presented during Section 5.2.3.

The chassis is modeled with three Cartesian Coordinates (x y z), representing the translation of the solid, and three Euler angles to describe the orientation of the chassis frame (α β γ).

Each suspension system is modeled with one coordinate associated to the vertical suspension deflection, $\bar{z}_i$. As explained in Section 3.2.1, since the modeling of the suspension system involves closed-loops, the motion of the suspension cannot be represented by its independent coordinate. To overcome this issue, in [97], the suspension system is replaced by kinematic tables, where the position, velocity and acceleration of the knuckle are derived as a function of $\bar{z}_i$. In the case of the front suspension system, the steering angle is also included as an input in the kinematic tables.

Finally, the position of each wheel with respect to the knuckle is defined by an angle around the wheel axis, (φ_{FL} φ_{FR} φ_{RL} φ_{RR}).

This makes a total of fourteen independent coordinates, which are grouped into vector z. It must me noted that the steering is not included, since it is an input to the model and not a coordinate itself.

$$\mathbf{z} = \left\{ \begin{array}{cccccccccccccc} x & y & z & \alpha & \beta & \gamma & \bar{z}_{FL} & \bar{z}_{FR} & \bar{z}_{RL} & \bar{z}_{RR} & \varphi_{FL} & \varphi_{FR} & \varphi_{RL} & \varphi_{RR} \end{array} \right\} \tag{5.44}$$

With respect to the implementation with the errorEKF, the variables of the state vector are now the same as the variables which define the MB model. Therefore, the terms of the transition matrix presented in Equation 3.60 are simpler to obtain: the matrix $\mathbf{R}^{\boldsymbol{\Phi}}$ is replaced by the identity matrix.

5.3.1 Suspension system: macro-joint

To obtain the kinematic tables which represent the motion of the whole suspension system, an off-line kinematic analysis based on the MB model of the suspension is launched. The values of spring deflection (and steering angle for the front suspension) vary along the suspension movement range. For each value, the position of the center of the knuckle is derived. The velocities and accelerations are obtained later through numerical differentiation of the positions. Terms due to the variation in time of the steering input have been neglected in the calculation of the velocities and accelerations of the front knuckles [97].

Due to the particularities of the macro-joints substituting the suspension links, special calculations should be performed to obtain the positions, velocities and accelerations of the knuckle [97].

The coordinates of the origin of the knuckle (for knuckle i, with suspension coordinate $\bar{z}_{FL}$), $\mathbf{r}_k$, are obtained as,

$$\mathbf{r}_k = \mathbf{r}_{ch} + \mathbf{A}_{ch}\bar{\mathbf{r}}_k \tag{5.45}$$

where $\mathbf{r}_{ch}$ is the vector of coordinates of the origin of the chassis local reference frame, $(x,\ y,\ z)$, $\mathbf{A}_{ch}$ is the chassis rotation matrix (Equation 5.18), function of $(\alpha,\ \beta,\ \gamma)$, and $\bar{\mathbf{r}}_k$ is the local position (with respect to the chassis reference frame) of the origin of the knuckle. This value is obtained from the kinematic table for a value of $\bar{z}_{FL}$ and a certain steering angle.

At velocity level, the expressions for the velocity of the origin of the knuckle, $\mathbf{v}_k$,

$$\mathbf{v}_k = \mathbf{v}_{ch} + \boldsymbol{\omega}_{ch} \times (\mathbf{r}_k - \mathbf{r}_{ch}) + \mathbf{A}_{ch}\frac{d\bar{\mathbf{r}}_k}{d\bar{z}_{FL}}\dot{\bar{z}}_{FL} \tag{5.46}$$

where $\mathbf{v}_{ch}$ is the velocity of the chassis frame and $\boldsymbol{\omega}_{ch}$ is the chassis angular velocity. The three components of the derivative of $\bar{\mathbf{r}}_k$ with respect to $\bar{z}_{FL}$ are tabulated as function of $\bar{z}_{FL}$ and the steering angle. The knuckle angular velocity, $\boldsymbol{\omega}_k$, is,

$$\boldsymbol{\omega}_k = \boldsymbol{\omega}_{ch} + \mathbf{A}_{ch}\bar{\boldsymbol{\omega}}_{k/c} \tag{5.47}$$

being $\bar{\boldsymbol{\omega}}_{k/c}$ the angular velocity of the knuckle with respect to the chassis,

$$\bar{\boldsymbol{\omega}}_{k/c} = \begin{bmatrix} 1 & 0 & sin\bar{\beta} \\ 0 & cos\bar{\alpha} & -sin\bar{\alpha}cos\bar{\beta} \\ 0 & sin\bar{\alpha} & cos\bar{\alpha}cos\bar{\beta} \end{bmatrix} \frac{d\bar{\boldsymbol{\theta}}_k}{d\bar{z}_{FL}}\dot{\bar{z}}_{FL} \tag{5.48}$$

where $\bar{\boldsymbol{\theta}}_k$ are the orientation angles of the knuckle, whose value is also included in the kinematic tables as a function of $\bar{z}_{FL}$ and the steering angle.

The acceleration of the origin of the knuckle, $\mathbf{a}_k$,

$$\begin{aligned}\mathbf{a}_k = \mathbf{a}_{ch} + \boldsymbol{\alpha}_{ch} \times (\mathbf{r}_k - \mathbf{r}_{ch}) + \boldsymbol{\omega}_{ch} \times [\boldsymbol{\omega}_{ch} \times (\mathbf{r}_k - \mathbf{r}_{ch})] + \\ 2\boldsymbol{\omega}_{ch} \times \mathbf{A}_{ch}\frac{d\bar{\mathbf{r}}_k}{d\bar{z}_{FL}}\dot{\bar{z}}_{FL} + \mathbf{A}_{ch}\left(\frac{d\bar{\mathbf{r}}_k}{d\bar{z}_{FL}}\ddot{\bar{z}}_{FL} + \frac{d^2\bar{\mathbf{r}}_k}{d\bar{z}_{FL}^2}\dot{\bar{z}}_{FL}^2\right)\end{aligned} \tag{5.49}$$

where $\mathbf{a}_{ch}$ is the acceleration of the chassis frame and $\boldsymbol{\alpha}_{ch}$ is the chassis angular acceleration. The three components of the second derivative of $\bar{\mathbf{r}}_k$ with respect to $\bar{z}_{FL}$ are tabulated as function of $\bar{z}_{FL}$ and the steering angle. The knuckle angular acceleration, α_k, is,

$$\boldsymbol{\alpha}_k = \boldsymbol{\alpha}_{ch} + \mathbf{A}_{ch}\bar{\alpha}_{k/ch} + \boldsymbol{\omega}_{ch} \times \mathbf{A}_{ch}\bar{\boldsymbol{\omega}}_{k/ch} \tag{5.50}$$

being $\bar{\alpha}_{k/ch}$ the angular acceleration of the knuckle with respect to the chassis, obtained as the time derivative of $\bar{\boldsymbol{\omega}}_{k/ch}$.

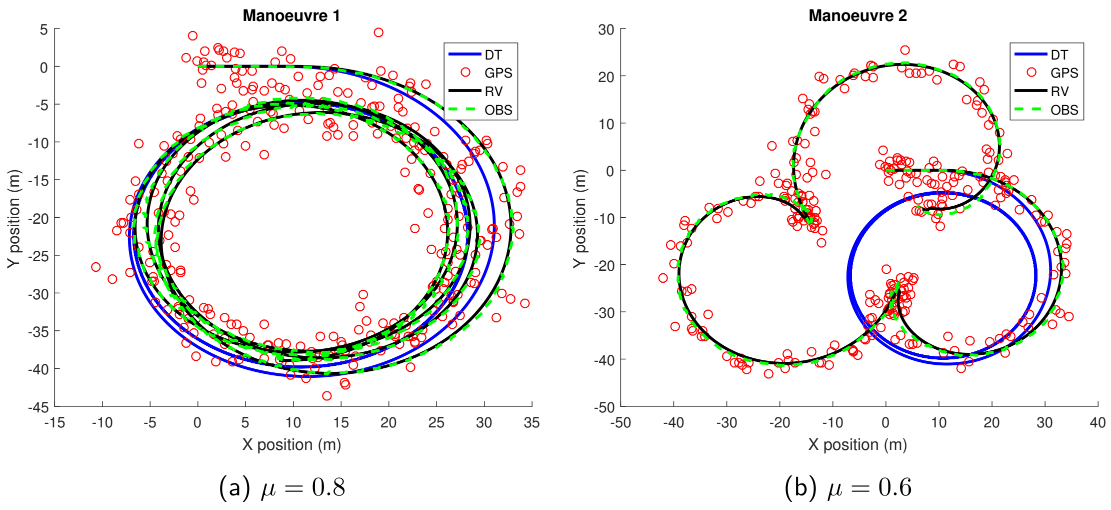
5.3.2 Results

As in Section 5.2.4, the simplified vehicle model presented is combined with the errorEKF introduced in Section 3.3. Prior to implement the *observer* in the embedded hardware, the covariance matrix of the filter is tuned to guarantee the stability and accuracy of the filter. The noise of the sensors is determined based on real sensor characteristics. All the MB models of the simulation framework have been updated to the simplified model. Thus, the modeling errors are still in the mass and in the tire-road friction coefficient. This gives a better insight of the filter performance.

The tested maneuver consists of a turn with a constant value of steering wheel angle and a constant throttle position, as for the state observer based on the complete vehicle model. In these tests, the simulated time is increased with respect to Section 5.2.4.1 in order to properly evaluate the filter behavior. The errors introduced in the *model* in order to create a drift with respect to the simulation of the *real vehicle* are applied in the mass and tire-road friction coefficient. First, the *real vehicle* has a mass of 600 kg and a friction coefficient of 0.8, while the *model* has a mass of 700 kg and a friction coefficient of 1. This maneuver is referred in the text as *Maneuver 1*. Next, in order to test a more aggressive maneuver, the friction coefficient of the *real vehicle* is reduced to 0.6, causing the vehicle to skid while turning. Since it is the second maneuver, it is referred as *Maneuver 2*.

5.3.2.1 Estimation Results

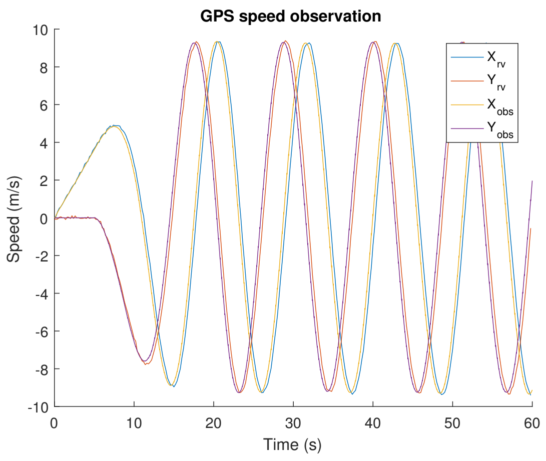
The path followed by each model is presented in Figure 5.17. In *Maneuver 1*, presented in Figure 5.17a, the trajectory of the *real vehicle* is similar to the one described by the *model*, in spite of the modeling errors. Meanwhile, *Maneuver 2*, in Figure 5.17b, the trajectory presents noticeable differences. While the *model* performs concentric circles, the *real vehicle* is not able to keep turning due to its lower tire-road friction coefficient. In every turn, the tire-road friction coefficient is too low to counteract the tangential forces required by the tire. This causes the vehicle to skid in a turn of 180°, with the corresponding loss of velocity.



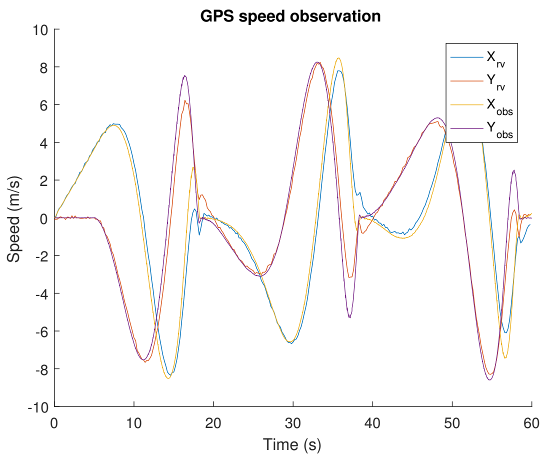
(a) $\mu = 0.8$ (b) $\mu = 0.6$

Figure 5.17: Maneuvers performed to test the errorEKF based on the simplified vehicle model. The paths followed by each model and the GPS sensor data are represented.

The measurements from the GPS that are fed to the *observer* for the corrections are also represented in Figure 5.17. In both cases, in spite of the dispersion of the positions measured by the GPS, the errorEKF is able to apply the required corrections so that the *observer* tracks the path followed by the *real vehicle*.

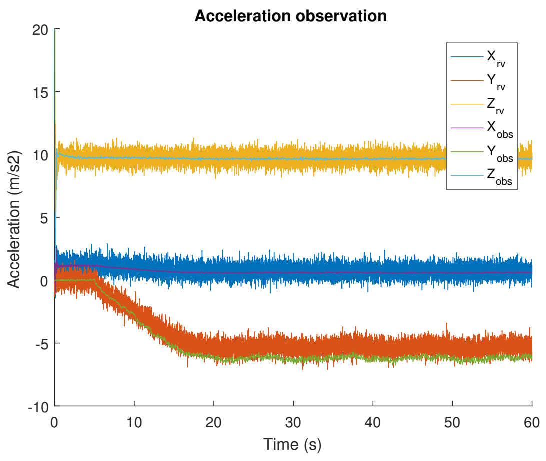


(a) $\mu = 0.8$

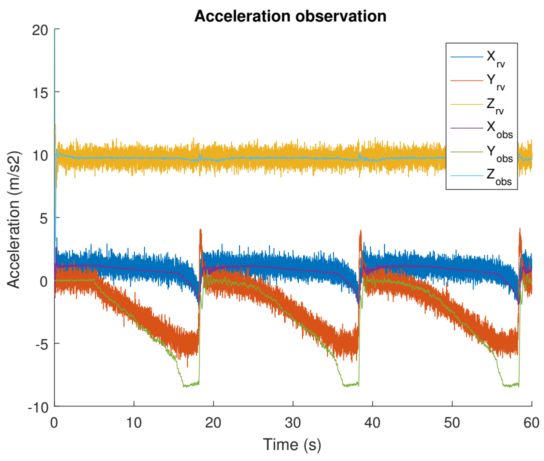


(b) $\mu = 0.6$

Figure 5.18: Comparison of the speeds measured in the maneuvers performed to test the errorEKF based on the simplified vehicle model.

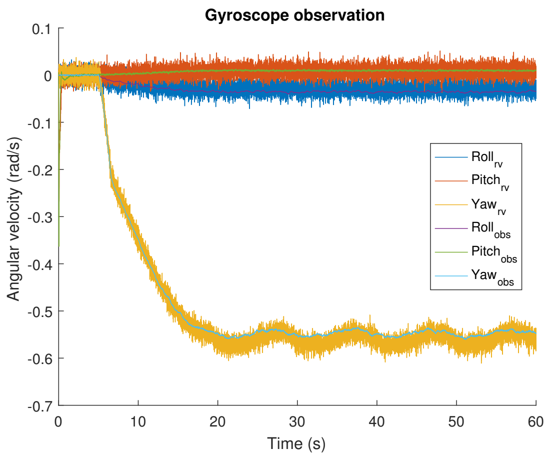


(a) $\mu = 0.8$

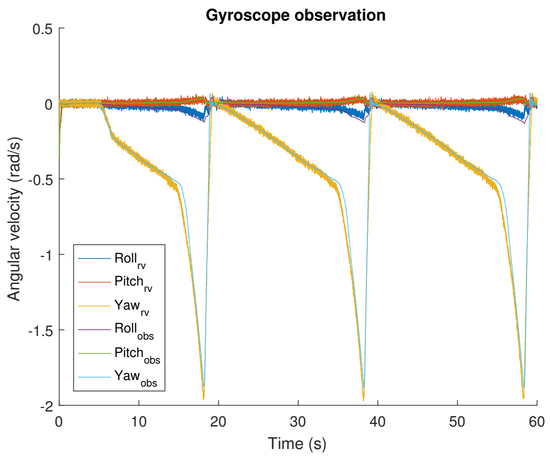


(b) $\mu = 0.6$

Figure 5.19: Comparison of the accelerations measured in the maneuvers performed to test the errorEKF based on the simplified vehicle model.



(a) $\mu = 0.8$



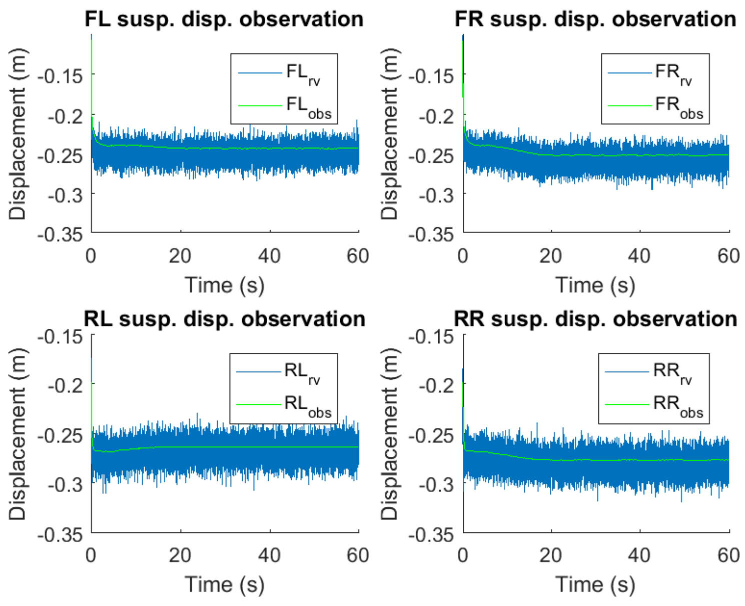
(b) $\mu = 0.6$

Figure 5.20: Comparison of the angular velocities of the chassis measured in the maneuvers performed to test the errorEKF based on the simplified vehicle model.

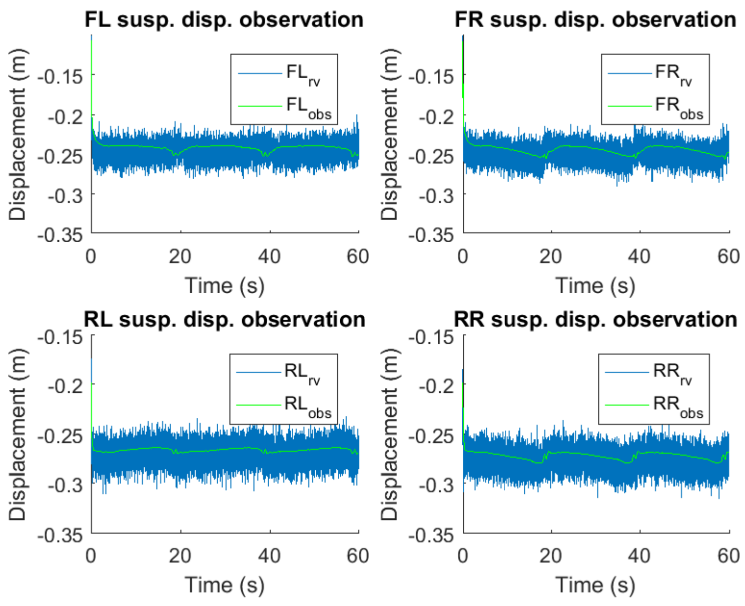
The sensor data gathered from the *observer* present the same trend that the sensor measurements provided by the *real vehicle* for the corrections. In addition, as a result of the errorEKF, the estimated measurements are less noisy than the measurements in the *real vehicle*. In Figures 5.18, 5.19, 5.20, 5.21 and 5.22, the estimated speed, accelerations and angular rates of the chassis, plus the suspension deflections and wheel angles are compared with their respective measurements in the *real vehicle*. It is also interesting to compare the measurements of both maneuvers in order to notice the effects of the different tire-road friction coefficient.

From the speed estimations, it can be seen how in the *Maneuver 2*, there is more drift between the estimated and the real speed measurements, especially when the vehicle is about to skid. The same behavior can be observed in the accelerations and angular rates of the chassis. While the estimations for *Maneuver 1* are similar to the measurements in the *real vehicle*, in *Maneuver 2* there is an important drift when the vehicle skids. Furthermore, regarding the lateral accelerations in both maneuvers, the estimated values are higher than the mean value of the measured accelerations. This shows that the errors in mass and tire-road friction coefficient are not fully corrected.

Regarding for the suspension deflections and wheel angle estimations, the *observer* tracks the measurements from the *real vehicle* and, in the case of the suspension deflections, reduces substantially the noise of the measurements. The effects of the mass error can also be appreciated in Figures 5.21a and 5.21b, where the estimated suspension deflections show that the spring compression is higher in the *observer*.

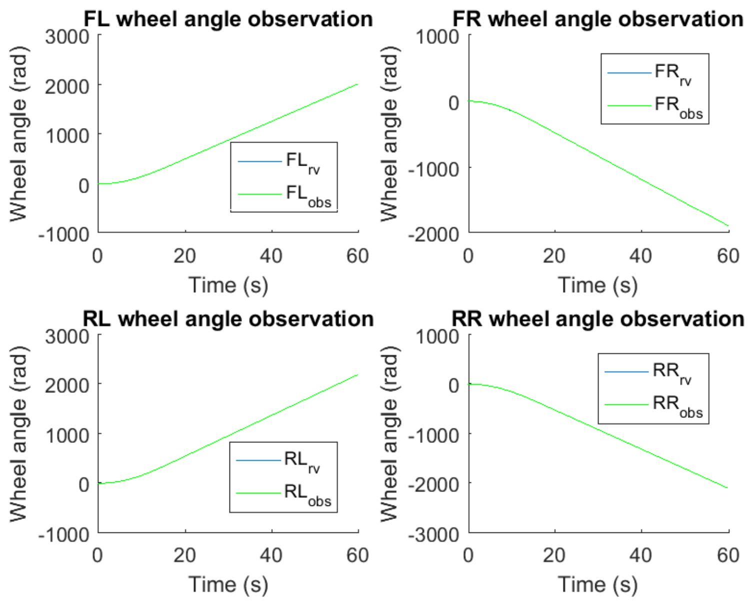


(a) $\mu = 0.8$

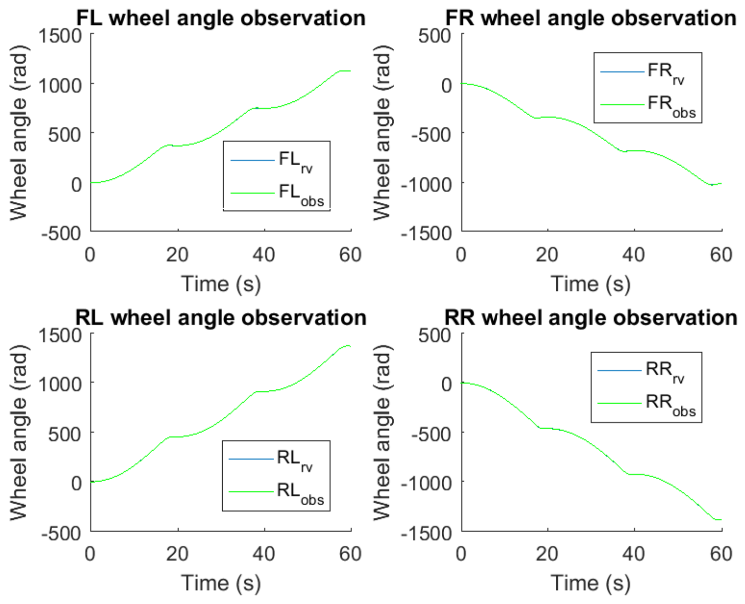


(b) $\mu = 0.6$

Figure 5.21: Comparison of the suspension deflections measured in the maneuvers performed to test the errorEKF based on the simplified vehicle model.



(a) $\mu = 0.8$



(b) $\mu = 0.6$

Figure 5.22: Comparison of the wheel angles measured in the maneuvers performed to test the errorEKF based on the simplified vehicle model.

Similarly to the estimated measurements, the tire forces derived from the *observer* present the same shape than the tire forces of the *real vehicle*. However, the drift due to the mass and tire-road friction coefficient is not fully corrected. The tire forces of each wheel for both maneuvers are presented from Figure 5.23 to Figure 5.30.

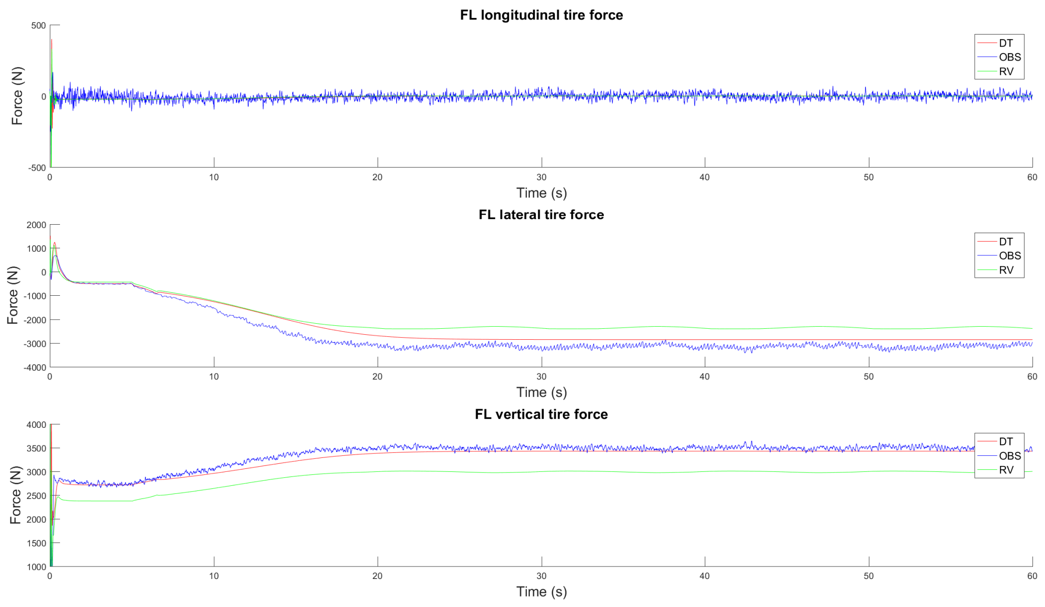


Figure 5.23: Tire forces of the FL wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.8$

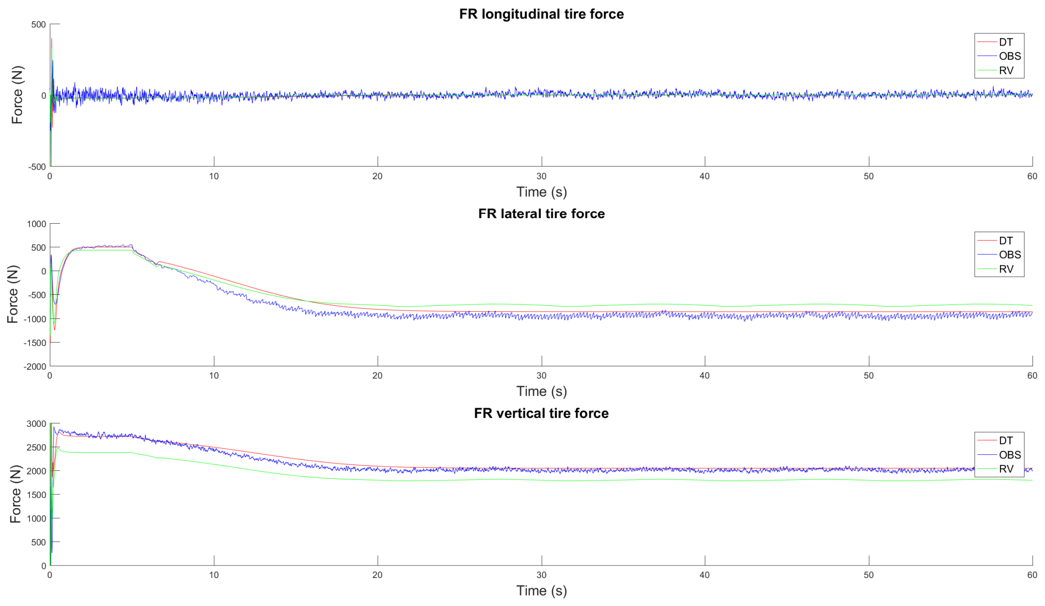


Figure 5.24: Tire forces of the FR wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.8$

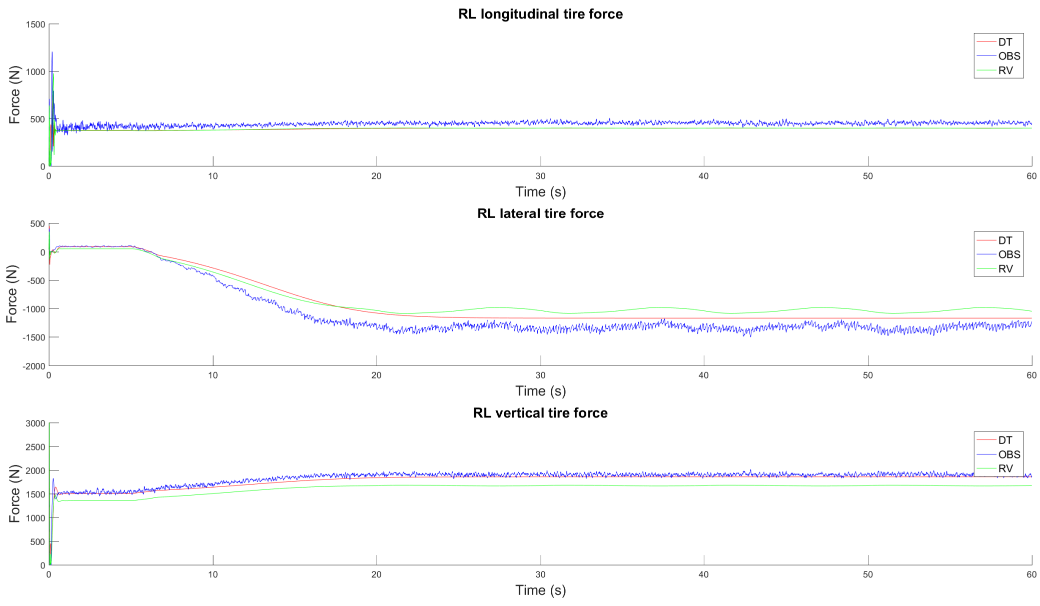


Figure 5.25: Tire forces of the RL wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.8$

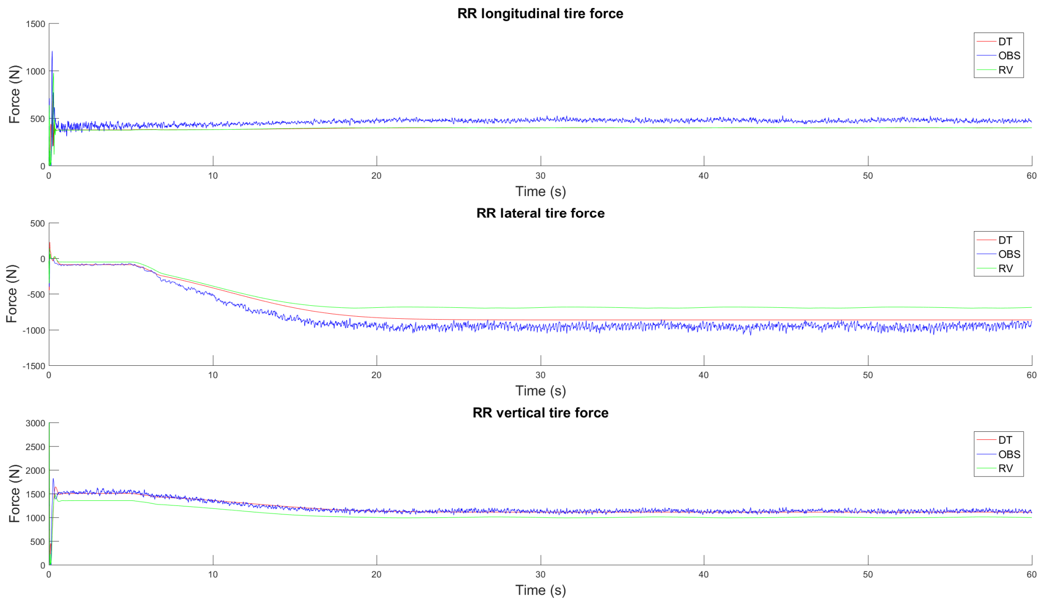


Figure 5.26: Tire forces of the RR wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.8$

Regarding the *Maneuver 1*, the tire forces of the *observer* are similar to the tire forces of the *model*. The drift in tire forces derived from the errors in the mass of the chassis and the tire-road friction coefficient is not corrected by the errorEKF.

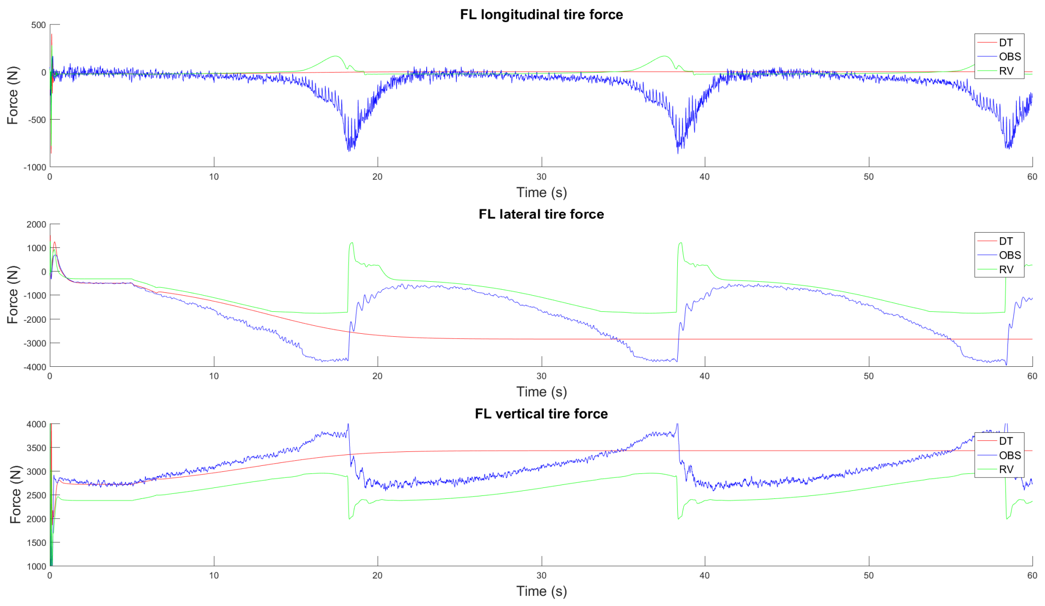


Figure 5.27: Tire forces of the FL wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.6$

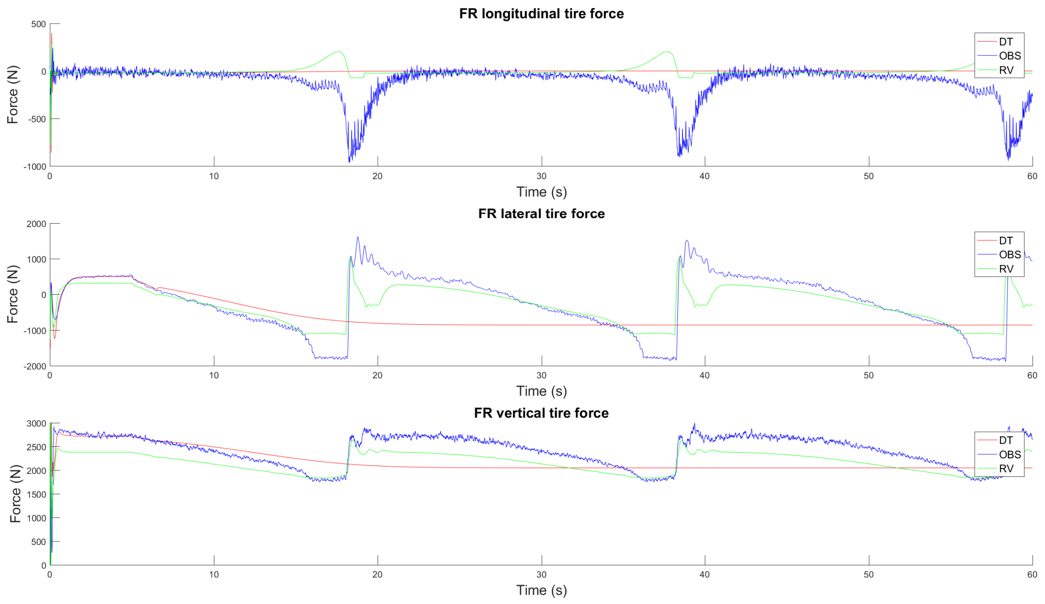


Figure 5.28: Tire forces of the FR wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.6$

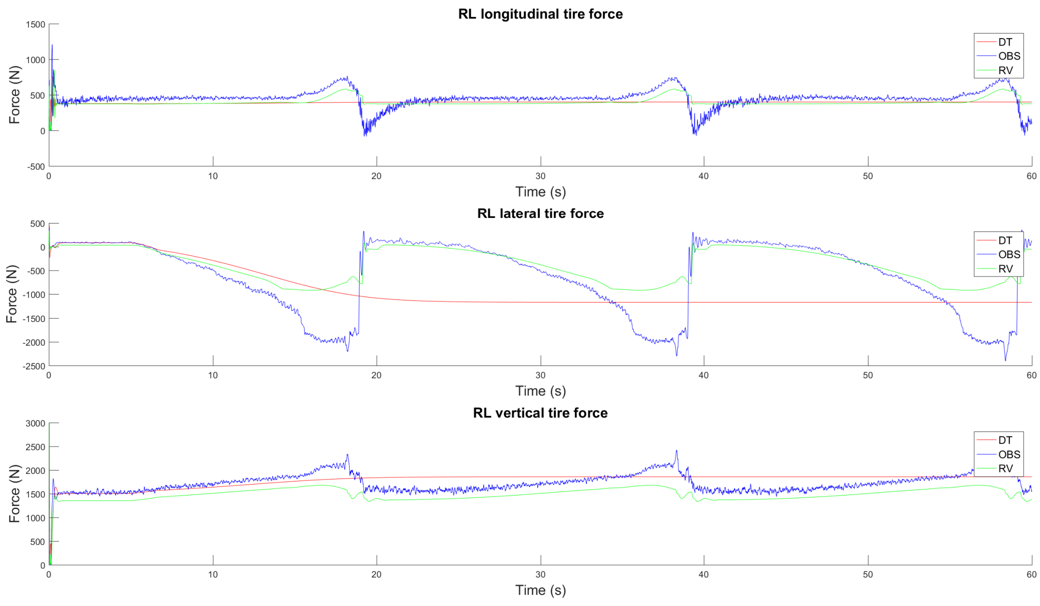


Figure 5.29: Tire forces of the RL wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.6$

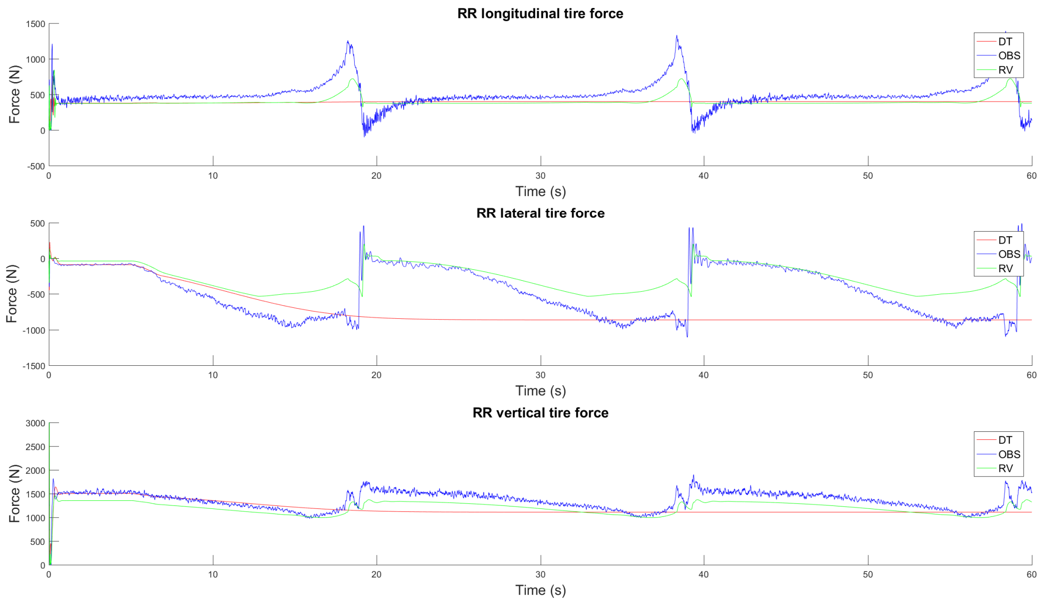


Figure 5.30: Tire forces of the RR wheel estimated by the errorEKF based on the simplified vehicle model for the maneuver with $\mu = 0.6$

In *Maneuver 2*, the *model* does not skid due to its higher tire-road friction coefficient. The drift in the tire forces between the *real vehicle* and the *model* is clearly shown in Figures 5.27-5.30. The *observer* is able to track the *real vehicle* during the skid and, therefore, the shape of the estimated tire forces is similar to the tire forces of the *real vehicle*. However, as commented earlier, there is an offset in the magnitude of the forces derived from the drift in mass and the tire-road friction coefficient, which affects to the vertical load of the vehicle and the lateral accelerations. The errorEKF is also not able to fully correct the drift between the *real vehicle* and its *model* in terms of tire forces in this maneuver.

In order to quantify the error obtained, the RMSE for the position, accelerations and tire forces is presented in Table 5.3. The estimations in positions and accelerations are compared against the error associated to the accuracy of the sensors employed for the corrections.

Table 5.3: RMSE measured in the maneuvers for testing the errorEKF combined with the simplified vehicle model.

Root-mean-square error			
Magnitude	*Maneuver 1* errorEKF	*Maneuver 2* errorEKF	Sensor
Position (m)	0.1988	0.4023	1.9075
X accel. (m/s^2)	0.1319	0.3018	0.4492
Y accel. (m/s^2)	0.7923	1.5429	0.447
Z accel. (m/s^2)	0.3496	0.3662	0.4485
RR long. tire force (N)	76.09	165.32	-
RR lat. tire force (N)	237.69	265.58	-
RR vert. tire force (N)	144.25	180.09	-

Comparing the RMSE between maneuvers, it can be seen as the estimations in *Maneuver 1* are more accurate than in *Maneuver 2*. Regarding the positions, it can be seen how the state estimator reduces the error in the GPS measurements in one order of magnitude, resulting in a more accurate positioning. With respect to the accelerations, the error in the estimated values is lower than the error in the measurements except for the lateral acceleration. As seen in Figure 5.19a and Figure 5.19b, there is an offset in the lateral acceleration due to the modeling errors introduced which leads to a higher RMSE error. The last value analyzed in terms of RMSE are the tire forces of the RR wheel. The worst result is obtained for the lateral tire forces, where the error made in the estimations is close to 30%.

5.3.2.2 Estimation Results: Alternative Observer

Although the errorEKF is capable to reduce the drift between the *real vehicle* and its *model*, the accuracy achieved is not enough in variables such as the tire forces. As stated in Chapter 2, there is a huge interest on knowing the tire forces on a vehicle. Since the sensors available for direct measure the forces are expensive, the work of this thesis is intended to provide accurate tire force estimations, following previous works mentioned during Section 2.1. Thus, the design of a new observer is addressed in this Section.

From what has been presented in Section 5.3.2.1, there is still a drift between the *observer* and the *real vehicle* after applying the corrections of the filter. Regarding to the tire forces, the response of the *observer* seems similar to the behavior of the *real vehicle* with an offset in the magnitude due to the errors in the mass and tire-road friction coefficient. This issue can be solved if the uncertainties on both parameters are reduced. For that purpose, the errorEKF, which is a state and input observer (Section 3.3) is combined with a parameter estimator.

State-Parameter-Input Observer

The state-parameter-input (SPI) observer designed in this section is based on a combination of two different Kalman filters: error-state extended Kalman filter (errorEKF) and the unscented Kalman filter (UKF). While the errorEKF is employed to estimate the states and inputs of the MB model, the UKF is in charge of parameter estimation. Even though the EKF has been also employed for parameter estimation, as commented in Section 2.1, it requires to determine the Jacobian relating the parameters (such as the mass or tire-road friction coefficient) with the variables of the MB model. In most models, this Jacobian is complex to determine. Using the UKF, the Jacobian is not needed and, thus, it reduces the complexity of the implementation [36] while leading to similar results [25].

As stated in Section 3.1, the UKF increases the computational cost of the simulation. Therefore, only a reduced set of parameters should be estimated. Most of the modeling parameters, such as the geometry of the suspension system, remain constant during the life cycle of the vehicle. Meanwhile, the inertial properties of the vehicle, such as the mass, can vary depending on the number of passengers or luggage [21], and the tire-road coefficient can be affected by meteorological conditions or tire wear. Both parameters play an important role on vehicle dynamics, affecting the acceleration, braking, handling and comfort [22]. In addition, the errorEKF gives a poor performance when the errors are in the mass and tire-road friction coefficient. Hence, the mass and the tire-road friction coefficient are estimated. The observer is described in the diagram of Figure 5.31.

In a first step, the parameter estimation is addressed through the UKF approach based on the MB model of the vehicle, whose equations can be found in Section 3.1. Later the errorEKF executes the state and input estimation based on the MB model whose parameters have been updated with the estimations of the UKF.

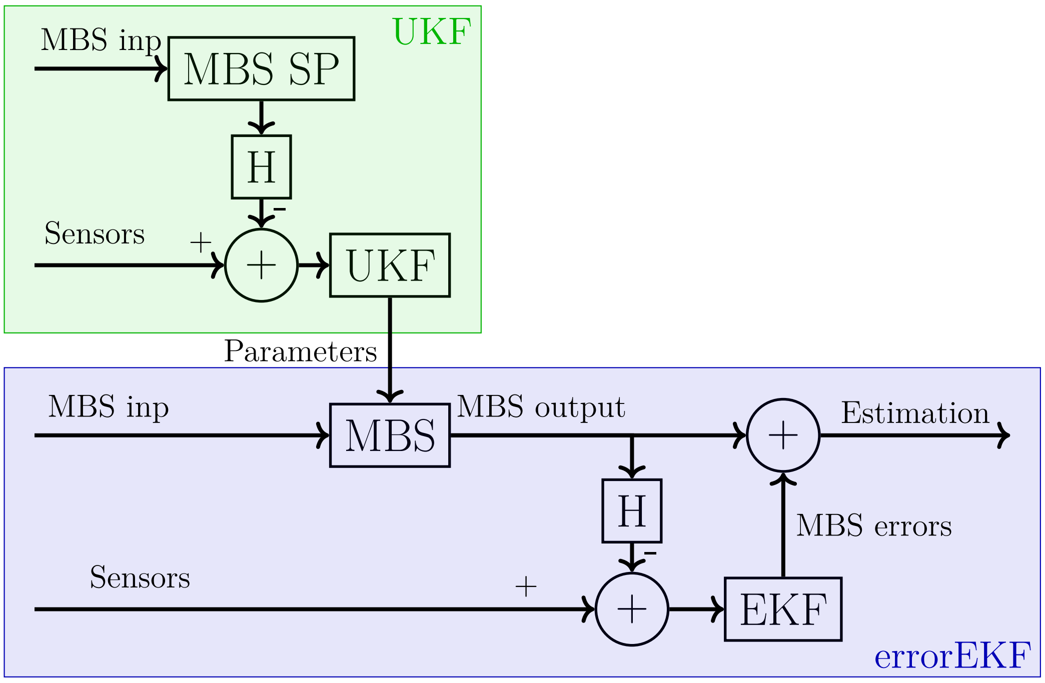


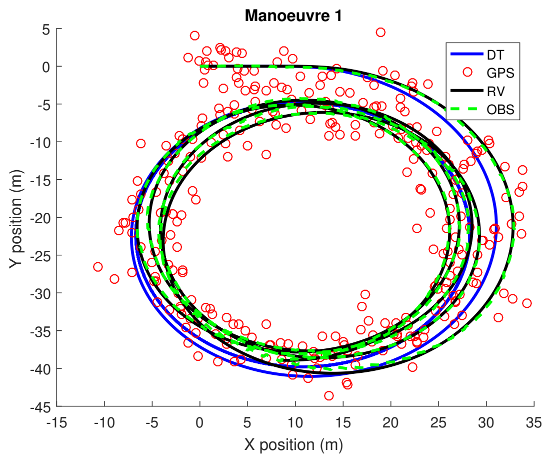
Figure 5.31: Flow diagram of a time step of the SPI observer presented, combining the errorEKF with a parameter estimator based on the UKF.

To evaluate the estimations of this new observer, the same maneuvers employed in Section 5.3.2.1 are simulated. The tests performed are focused on the simplified vehicle model, since the full vehicle model is quite far from real time.

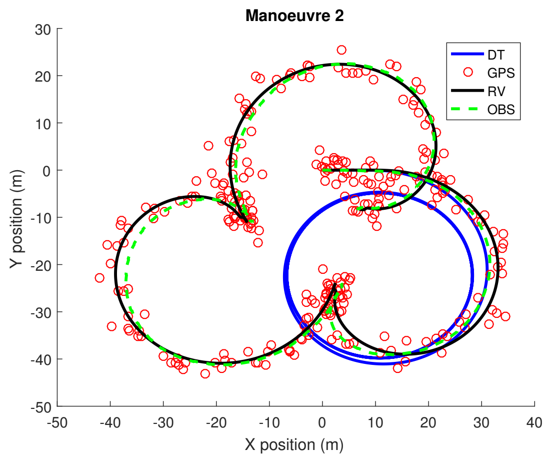
Results

Following the scheme of Section 5.3.2.1, the path followed by each model is first presented (Figure 5.32). In both cases, the behavior of the *observer* is such that it tracks the trajectory of the *real vehicle*, in spite of the existing difference between the *real vehicle* and its *model* and the low GPS accuracy.

With respect to the other sensor measurements, after a initial phase of stabilization, the *observer* removes the offset observed in previous results due to the errors in the parameters. Furthermore, the estimated measurements are again less noisy than the measurements in the *real vehicle*. In Figures 5.33, 5.34, 5.35, 5.36 and 5.37, the estimated speed, accelerations and angular rates of the chassis, plus the suspension deflections and wheel angles are compared with their respective measurements in the *real vehicle*.

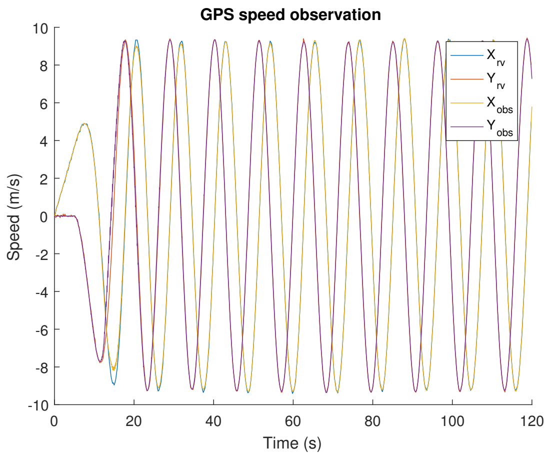


(a) $\mu = 0.8$

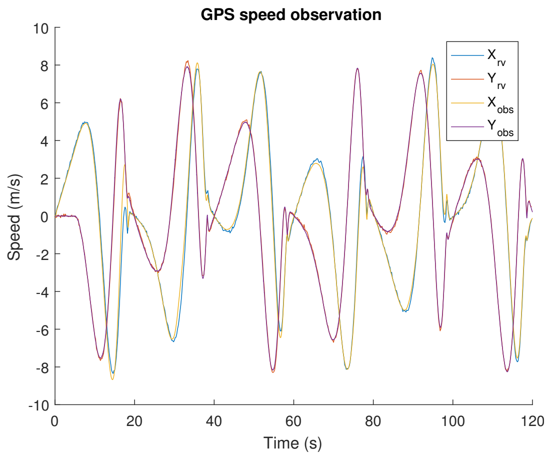


(b) $\mu = 0.6$

Figure 5.32: Maneuvers performed to test the SPI observer based on the simplified vehicle model. The paths followed by each model and the GPS sensor data are represented.

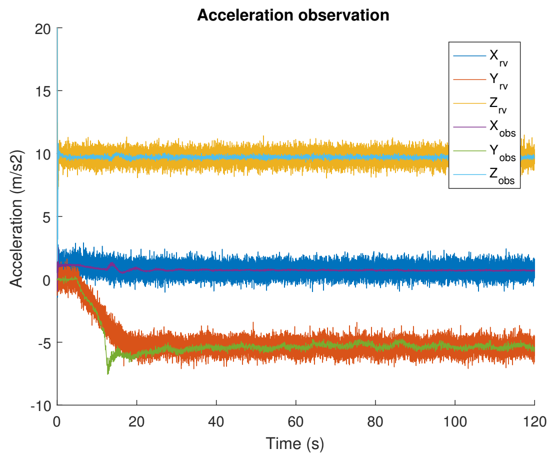


(a) $\mu = 0.8$

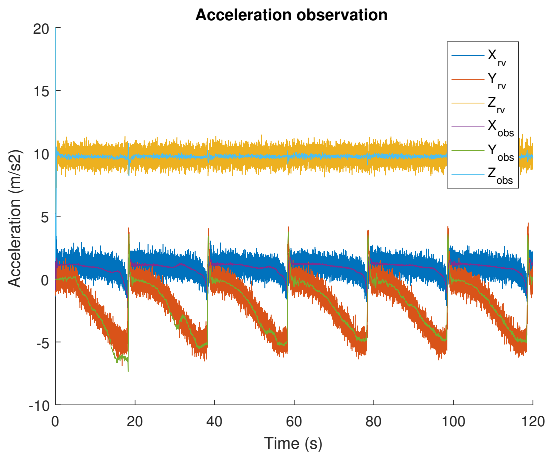


(b) $\mu = 0.6$

Figure 5.33: Comparison of the speeds measured in the maneuvers performed to test the SPI observer based on the simplified vehicle model.

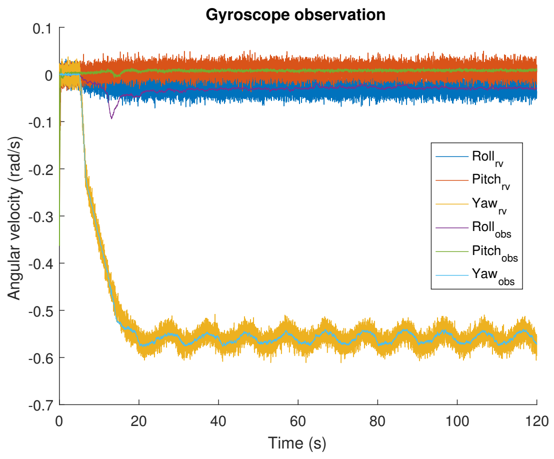


(a) $\mu = 0.8$

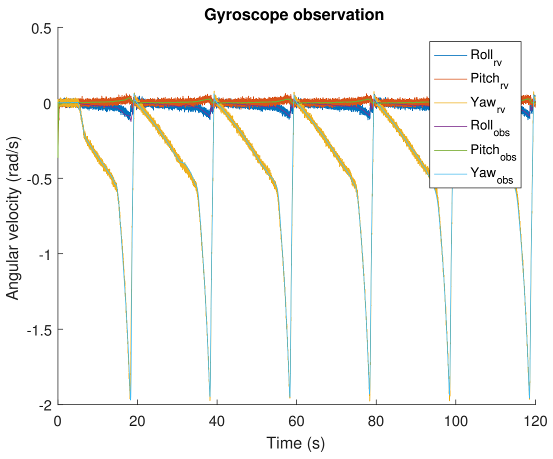


(b) $\mu = 0.6$

Figure 5.34: Comparison of the accelerations measured in the maneuvers performed to test the SPI observer based on the simplified vehicle model.



(a) $\mu = 0.8$

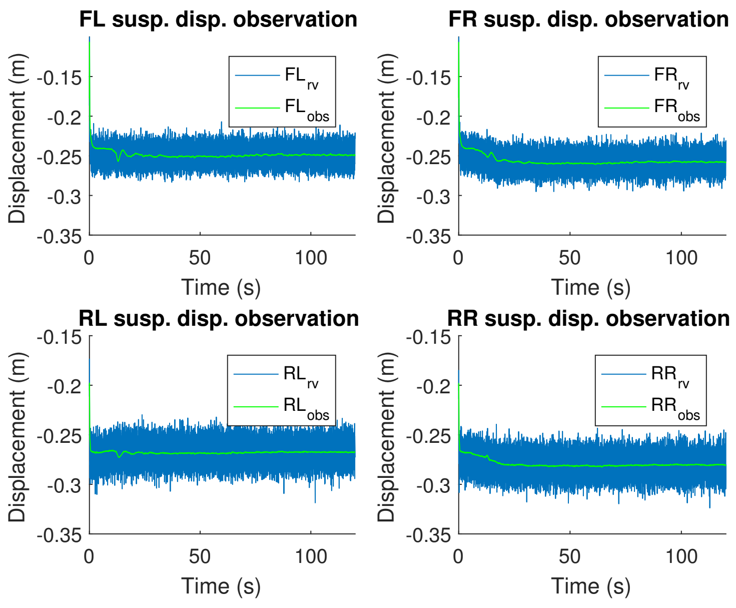


(b) $\mu = 0.6$

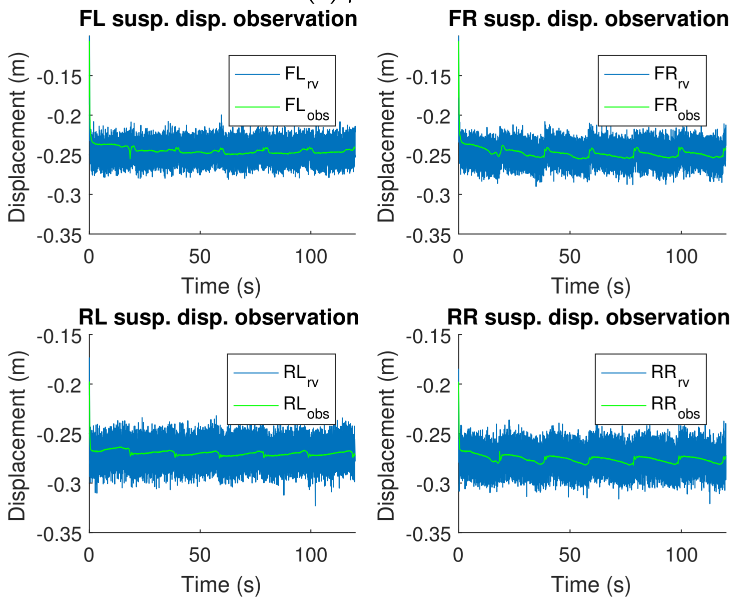
Figure 5.35: Comparison of the angular velocities of the chassis measured in the maneuvers performed to test the SPI observer based on the simplified vehicle model.

The speed estimations are as accurate as for the errorEKF. Is in the accelerometer and gyroscope measurements where the advantages of the parameter estimation are seen. In both maneuvers, the accelerations measured are around the mean value of the noisy sensor data from the *real vehicle*, removing the offset that was present in the previous results. In the angular rates, the peak value of the yaw rate is now at the same value in the *observer* and *real vehicle* measurements.

With respect to the suspension deflections and wheel angle estimations, the benefits of the parameter estimation are also shown. The *observer* tracks the measurements from the *real vehicle* and reduces the noise of the signals.

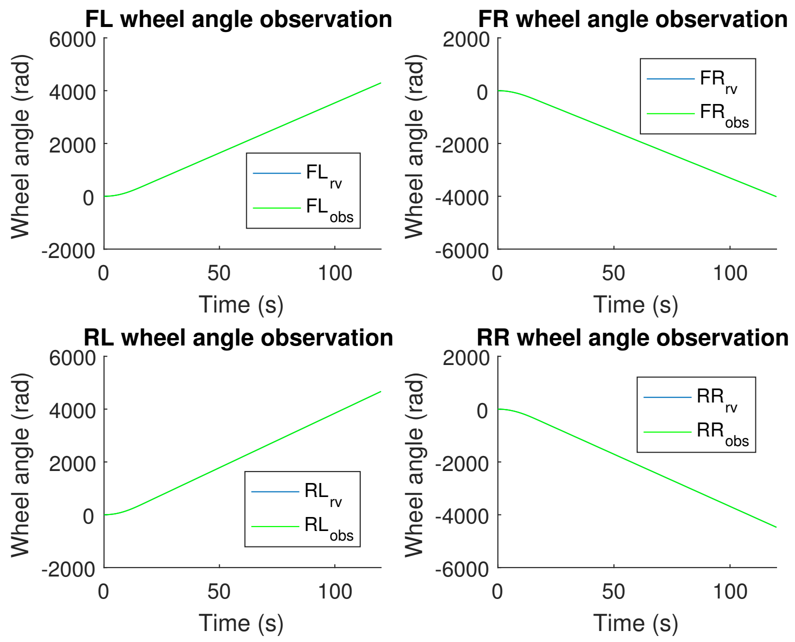


(a) $\mu = 0.8$

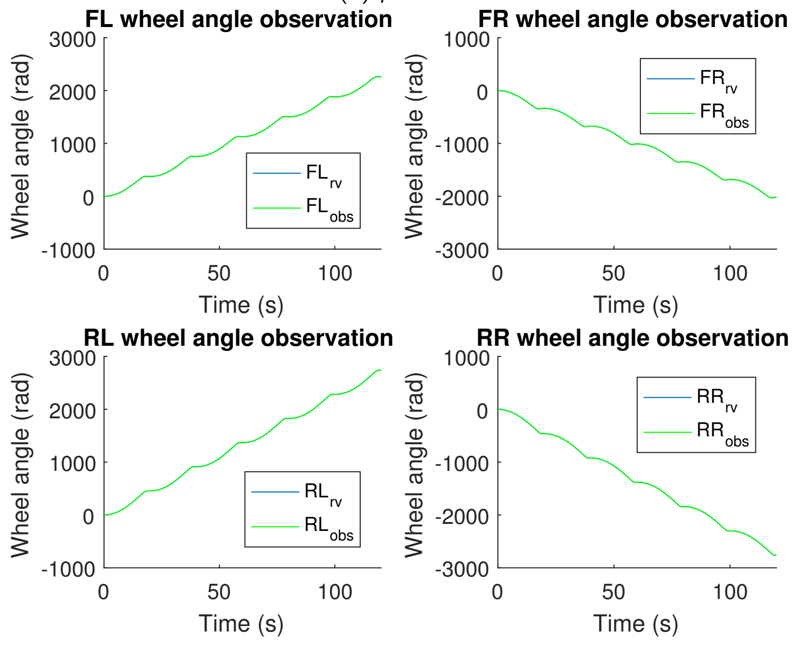


(b) $\mu = 0.6$

Figure 5.36: Comparison of the suspension deflections measured in the maneuvers performed to test the SPI observer based on the simplified vehicle model. The zero suspension deflection corresponds to a full compression state.



(a) $\mu = 0.8$



(b) $\mu = 0.6$

Figure 5.37: Comparison of the wheel angles measured in the maneuvers performed to test the SPI observer based on the simplified vehicle model.

The observed improvements when comparing the estimated with the real sensor measurements are also propagated to the tire forces. With the parameter estimation, the vertical load of the vehicle is corrected and, thus, the magnitude of the vertical tire forces is corrected. In addition, the correction of the tire-road friction coefficient yields into an accurate value of lateral tire forces. The tire forces of each wheel for both maneuvers are presented from Figure 5.38 to Figure 5.45.

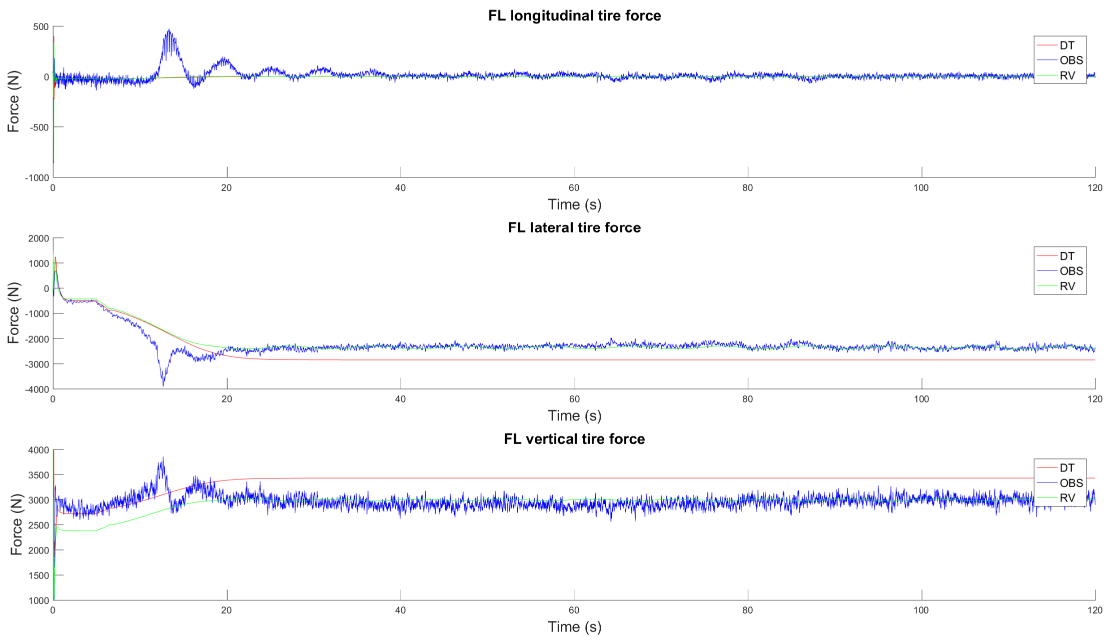


Figure 5.38: Tire forces of the FL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.8$

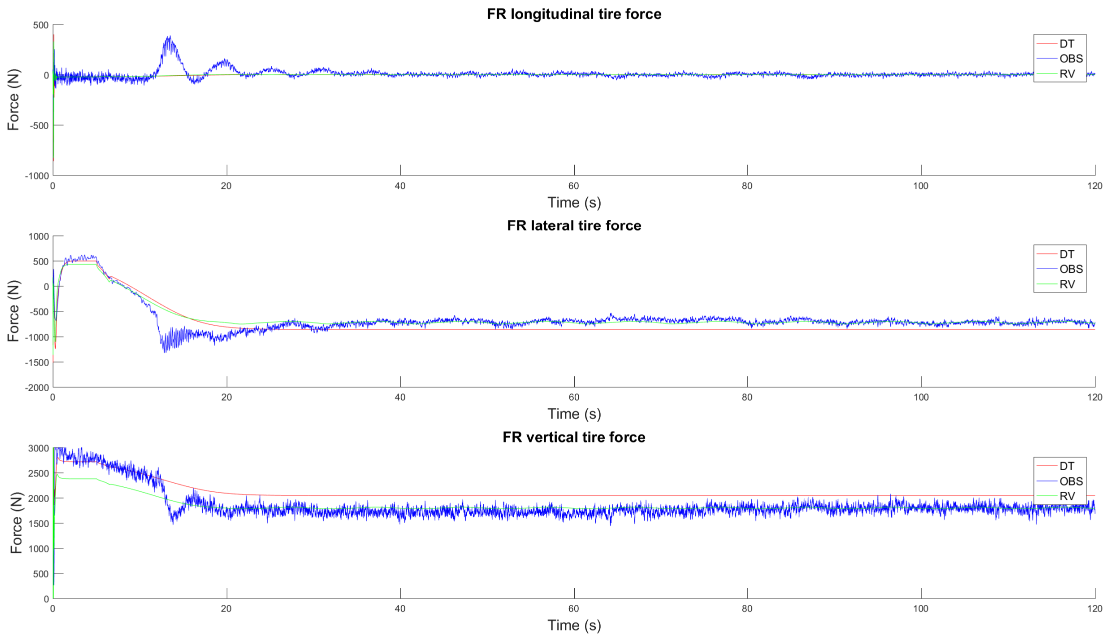


Figure 5.39: Tire forces of the FR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.8$

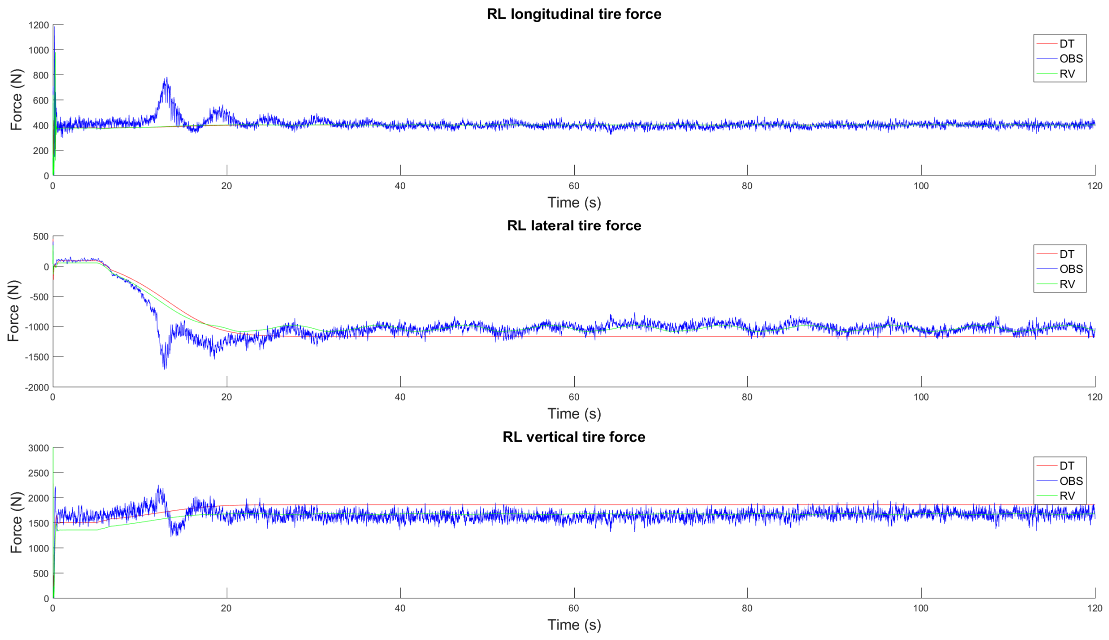


Figure 5.40: Tire forces of the RL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.8$

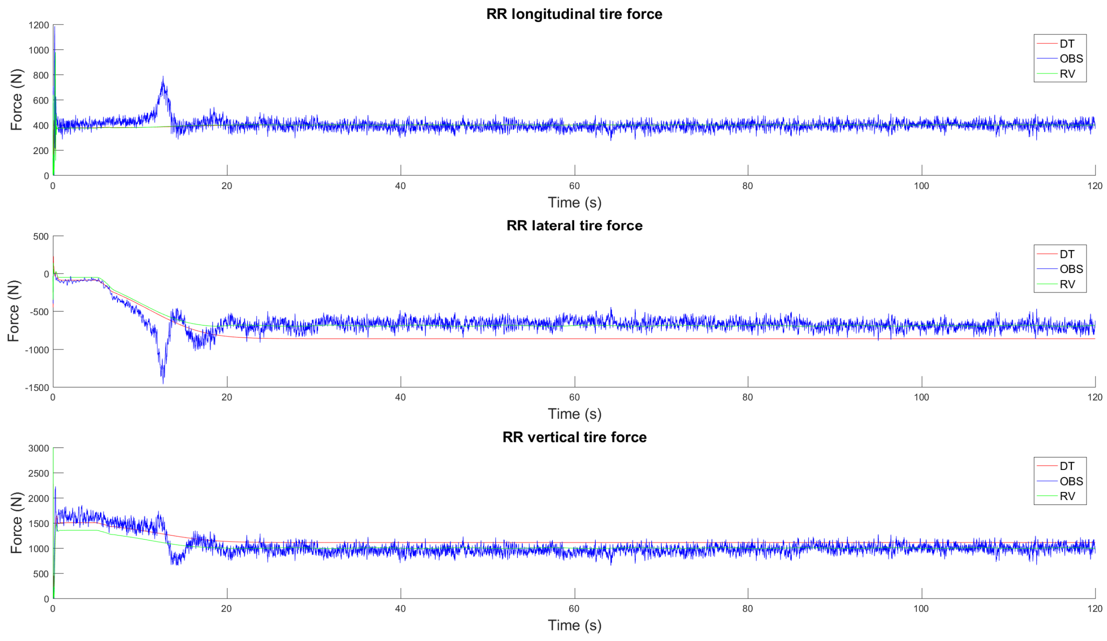


Figure 5.41: Tire forces of the RR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.8$

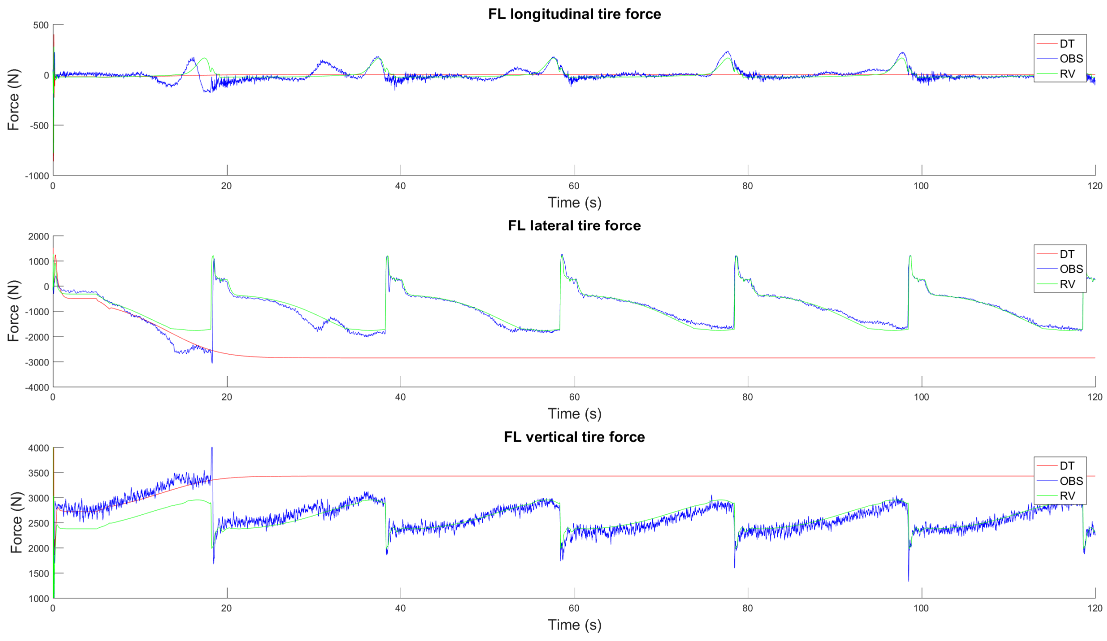


Figure 5.42: Tire forces of the FL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.6$

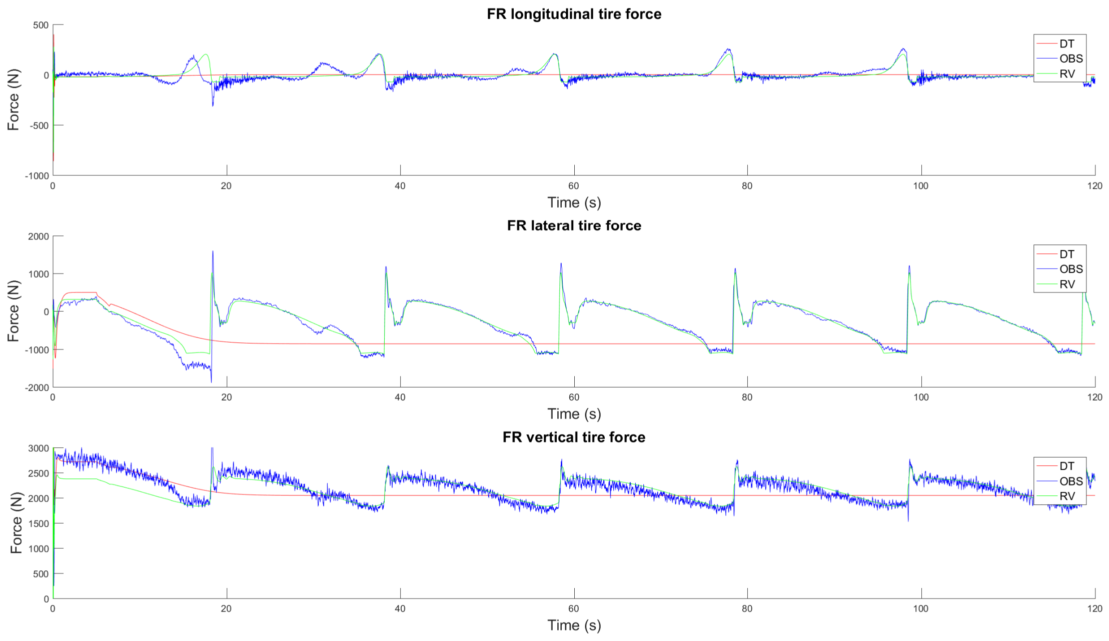


Figure 5.43: Tire forces of the FR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.6$

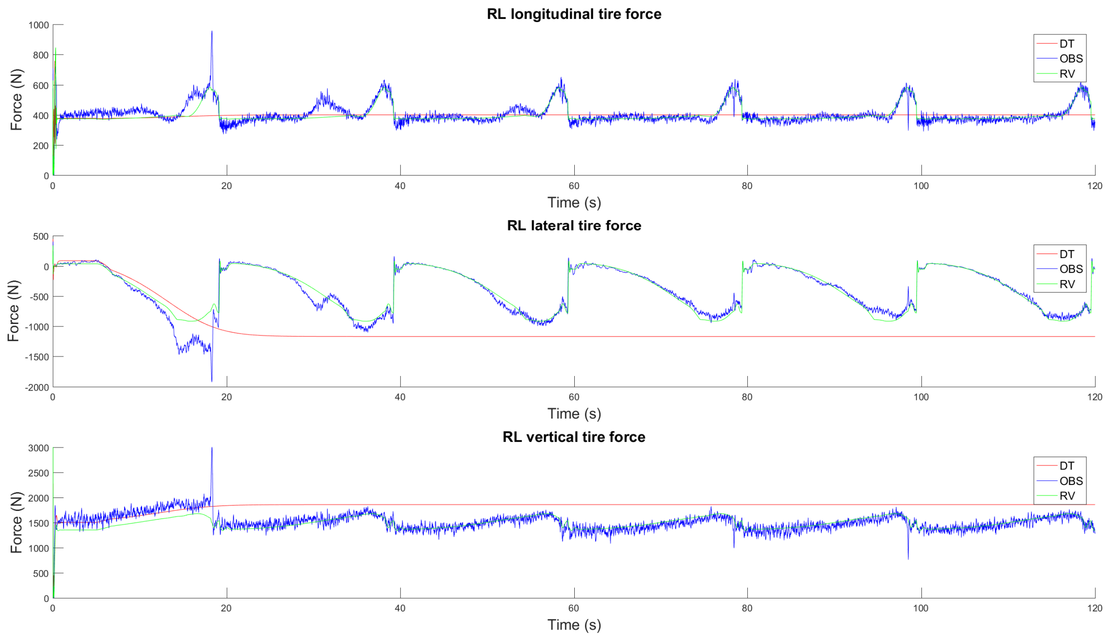


Figure 5.44: Tire forces of the RL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.6$

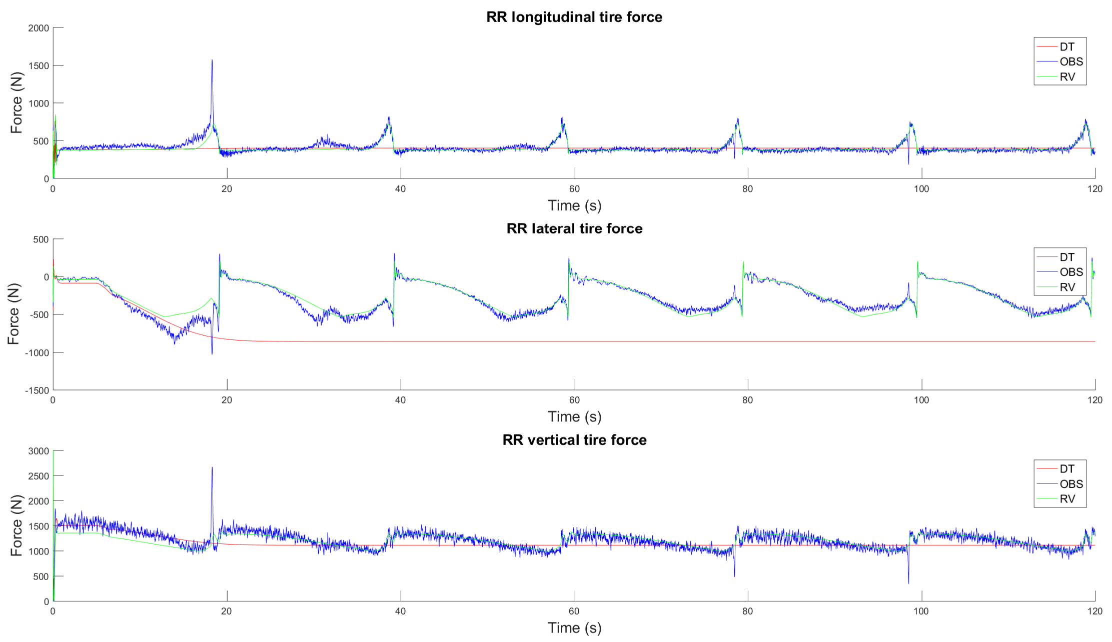


Figure 5.45: Tire forces of the RR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver with $\mu = 0.6$

Through the RMSE, the estimations of the SPI observer can be compared with the results obtained through the errorEKF. As in Table 5.3, the RMSE in position, accelerations and tire forces of the RR wheel are presented in Table 5.4. For the clarity of the reader, the RMSE in the errorEKF are also included.

Table 5.4: RMSE measured in the maneuvers for testing the SPI observer combined with the simplified vehicle model.

Root-mean-square error				
Magnitude	*Maneuver 1*		*Maneuver 2*	
	SPI	errorEKF	SPI	errorEKF
Position (m)	0.2288	0.1988	0.8143	0.4023
X accel. (m/s^2)	0.0845	0.1319	0.1197	0.3018
Y accel. (m/s^2)	0.4785	0.7923	0.5132	1.5429
Z accel. (m/s^2)	0.2678	0.3496	0.2704	0.3662
RR long. tire force (N)	47.81	76.09	49.87	165.32
RR lat. tire force (N)	99.68	237.69	74.46	265.58
RR vert. tire force (N)	127.42	144.25	111.42	180.09

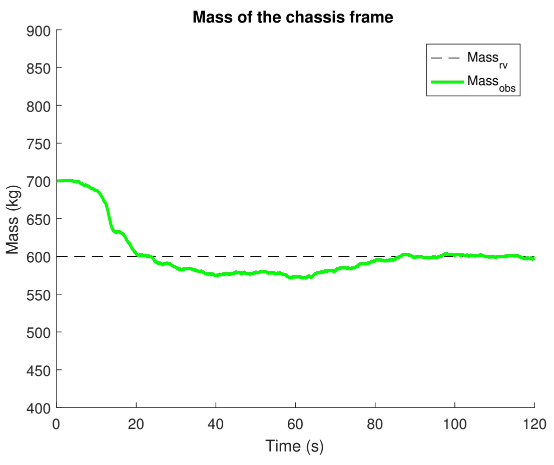
Except from the position, the SPI observer improves the accuracy of the errorEKF. With respect to the tire forces, the error is highly reduced. However, the SPI observer requires of more time to achieve a stable state. If the RMSE is calculated from a time where the filter is stabilized, the errors are reduced, as presented in Table 5.5.

Table 5.5: RMSE measured after stabilization in the maneuvers for testing the SPI observer combined with the simplified vehicle model.

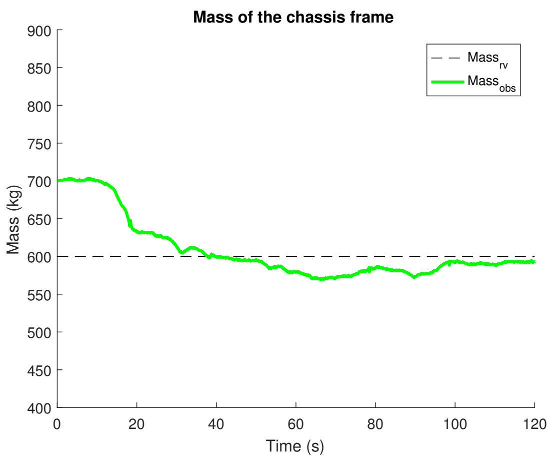
Root-mean-square error				
Magnitude	*Maneuver 1*		*Maneuver 2*	
	SPI	errorEKF	SPI	errorEKF
Position (m)	0.2196	0.1988	0.4793	0.4023
X accel. (m/s^2)	0.0343	0.1319	0.0634	0.3018
Y accel. (m/s^2)	0.1551	0.7923	0.2750	1.5429
Z accel. (m/s^2)	0.1022	0.3496	0.1121	0.3662
RR long. tire force (N)	28.27	76.09	28.41	165.32
RR lat. tire force (N)	57.77	237.69	41.27	265.58
RR vert. tire force (N)	79.48	144.25	75.30	180.09

Once that the observer is stabilized, the drift between the *model* and the *real vehicle* is fully corrected. The errors in the acceleration estimation are lower than the measurements from the accelerometer. Comparing the RMSE between both observers, it can be seen that the major improvement is on the tire forces estimation. The error in the estimated tire forces is less than 8% with the SPI observer.

The improvements presented in the estimations are related with the parameter estimation. The estimated mass and tire-road friction coefficient are shown in Figures 5.46 and 5.47 respectively. In *Maneuver 1*, the parameter estimation is smoother than in *Maneuver 2*, since the maneuver is less aggressive.

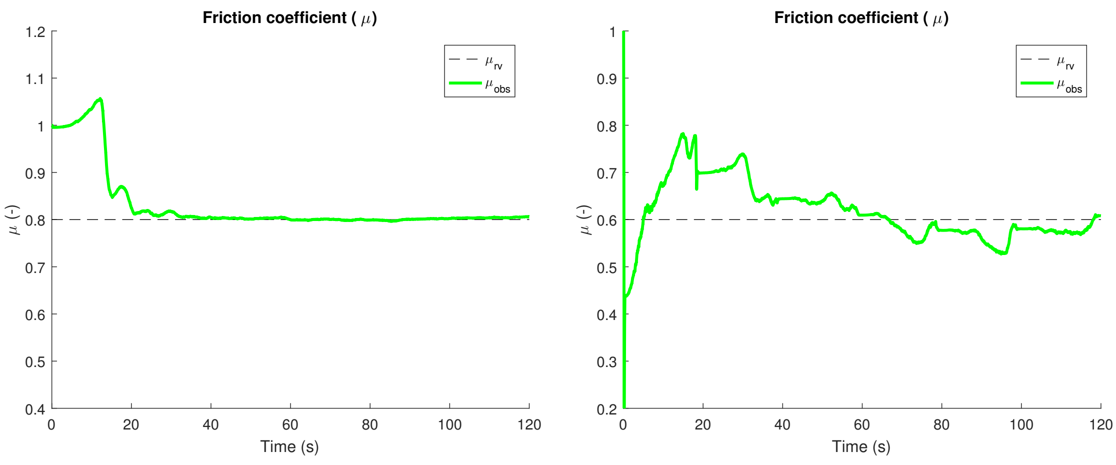


(a) $\mu = 0.8$



(b) $\mu = 0.6$

Figure 5.46: Estimated mass of the chassis in the maneuvers performed to test the SPI observer based on the simplified vehicle model.



(a) $\mu = 0.8$ (b) $\mu = 0.6$

Figure 5.47: Estimated tire-road friction coefficient in the maneuvers performed to test the SPI observer based on the simplified vehicle model.

Additional maneuver

In order to test the new filter in terms of stability, an additional maneuver, referred as *Maneuver 3*, is presented. While the mass of the chassis usually remains constant during a maneuver (the number of passengers or luggage does not change until the vehicle stops), the tire-road friction coefficient is susceptible to change due to climatological conditions or changes in the pavement. Thus, the turning maneuver of the previous sections is modified in such a way that the tire-road friction coefficient is modified during the maneuver. Initially, it is set at 0.8. After 60 seconds, it is reduced to 0.6. As a reminder, the tire-road friction coefficient of the *model* is 1, and its mass is 100 kg higher than in the *real vehicle*.

In order to represent the performed maneuver, the trajectories of each model are shown in Figure 5.48. As in the previous maneuvers, the *observer* corrects the drift in the *model* and tracks the path followed by the *real vehicle*.

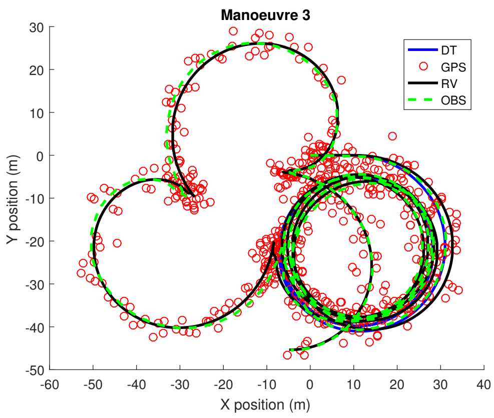


Figure 5.48: Additional maneuver performed to test the SPI observer based on the simplified vehicle model. The paths followed by each model and the GPS sensor data are represented.

With respect to the sensor measurements, the behavior is similar to the previous maneuvers. The *observer* tracks with accuracy and low sensor noise the sensor data obtained from the *real vehicle*.

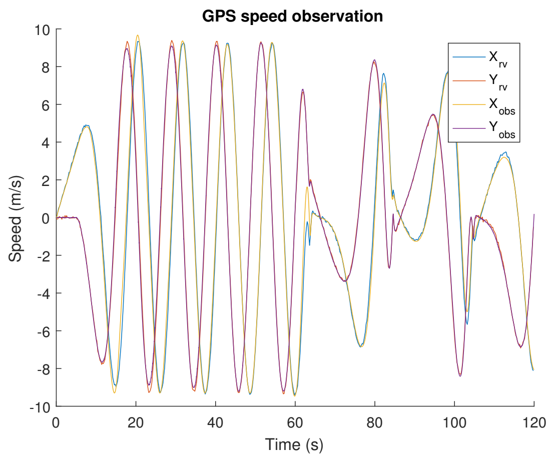


Figure 5.49: Comparison of the speeds measured in the maneuver performed to test the SPI observer based on the complete vehicle model under changes in the friction coefficient.

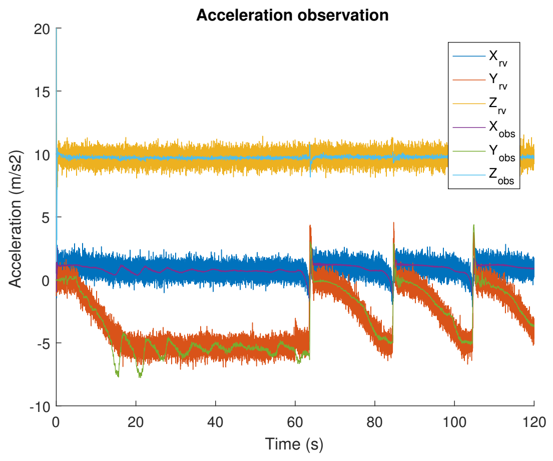


Figure 5.50: Comparison of the acceleration measured in the maneuver performed to test the SPI observer based on the complete vehicle model under changes in the friction coefficient.

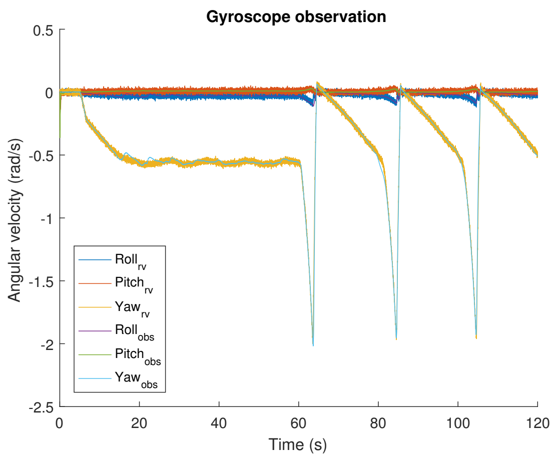


Figure 5.51: Comparison of the angular velocities measured in the maneuver performed to test the SPI observer based on the complete vehicle model under changes in the friction coefficient.

As opposite from the previous maneuvers, during the first 40 seconds of the simulation, there are high oscillations in the measurements from the *observer*, specially in the lateral acceleration. This oscillations are later reduced when the tire-road friction coefficient changes, resulting in better accuracy. This behavior is due to the tuning of the covariance matrix. Each maneuver has its optimal configuration for the covariance matrix. *Maneuver 3* can be seen as a combination of the *Maneuver 1* and *Maneuver 2* and, therefore, a trade-off was achieved for the noises of the covariance matrix. The tuning was also performed looking for a quick reaction to changes in the tire-road friction coefficient. It should be noted that the tuning of the covariance matrix is now based on a trial-error procedure. In order to increase the robustness of the estimator, alternative methods should be considered.

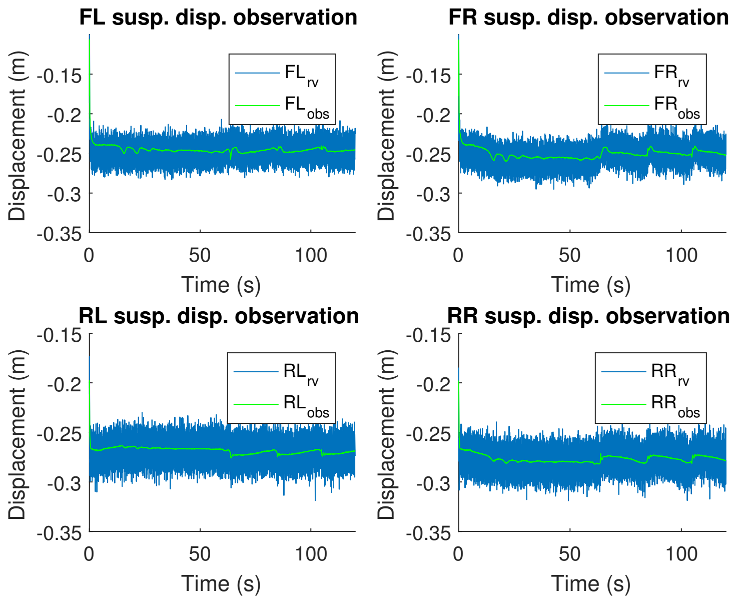


Figure 5.52: Comparison of the suspension deflections measured in the maneuver performed to test the SPI observer based on the complete vehicle model under changes in the friction coefficient.

The aforementioned oscillations can also be appreciated in the suspension deflections, shown in Figure 5.52. The rest of the simulation, the estimated deflections show less noise than the data measured in the *real vehicle*. The wheel angles estimations (Figure 5.53) are equal to the *real vehicle* measurements.

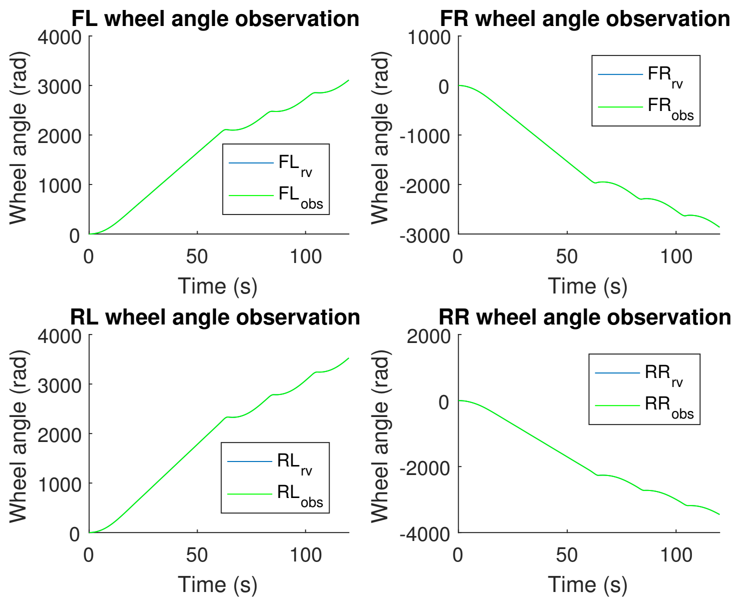


Figure 5.53: Comparison of the wheel angles measured in the maneuver performed to test the SPI observer based on the complete vehicle model under changes in the friction coefficient.

In Figures 5.54-5.57, the tire forces are presented. As for the acceleration measurements, an undesirable oscillation is observed in the first seconds of the maneuver. At the expenses of these oscillations, the estimation of the forces is quite fast when the tire-road friction coefficient changes. This quick response is useful specially for control purposes, so the proper actions can be executed in order to avoid the vehicle to skid and guarantee the safety of the passengers.

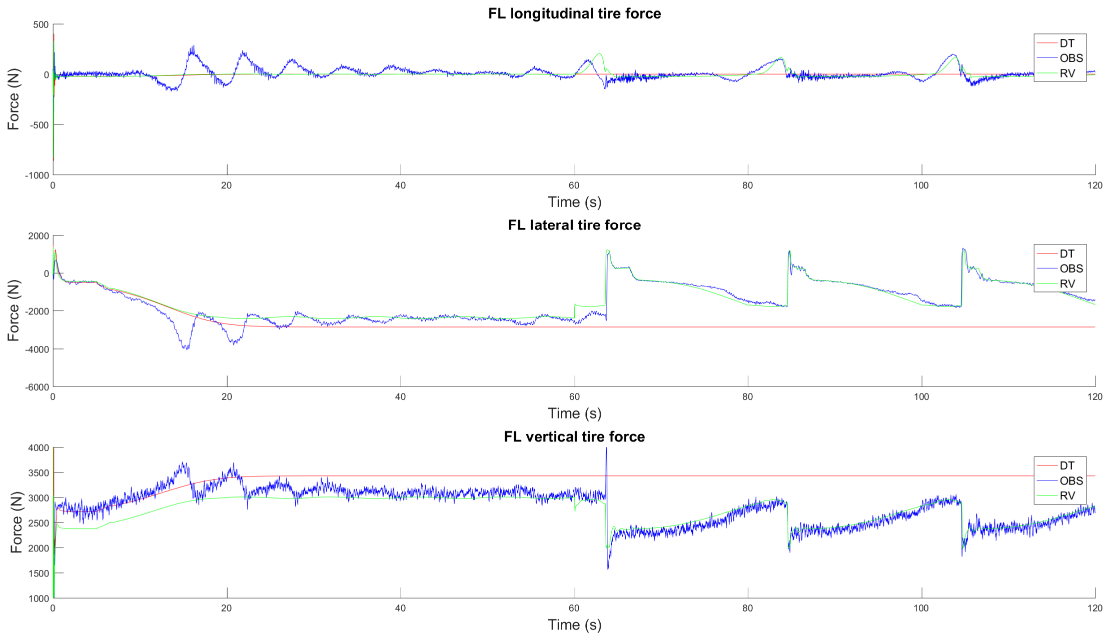


Figure 5.54: Tire forces of the FL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver under changes in the tire-road friction coefficient.

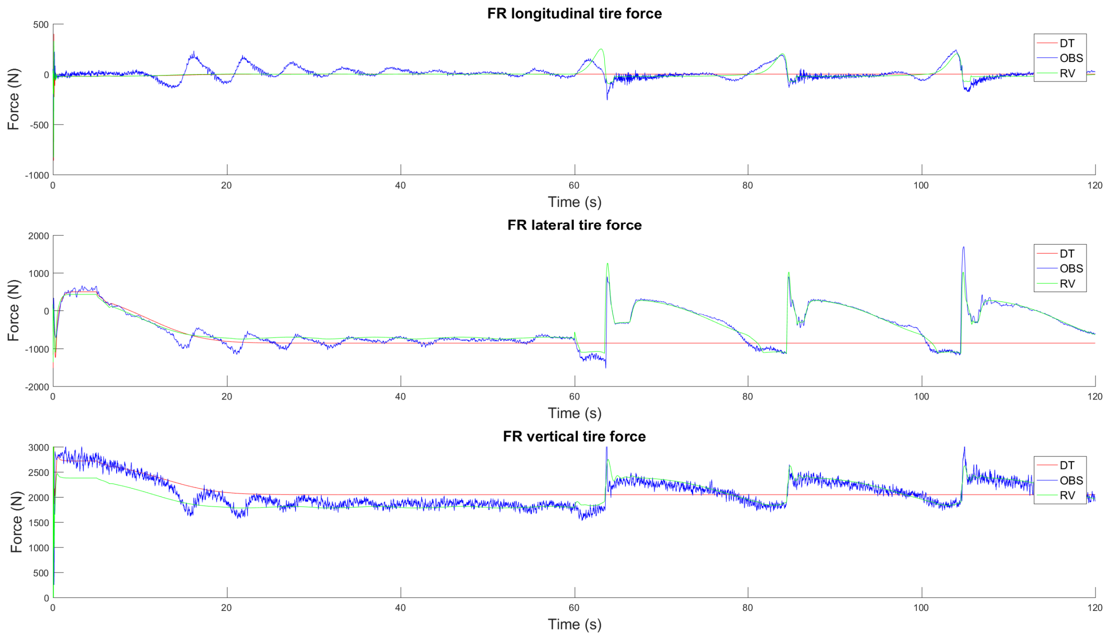


Figure 5.55: Tire forces of the FR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver under changes in the tire-road friction coefficient.

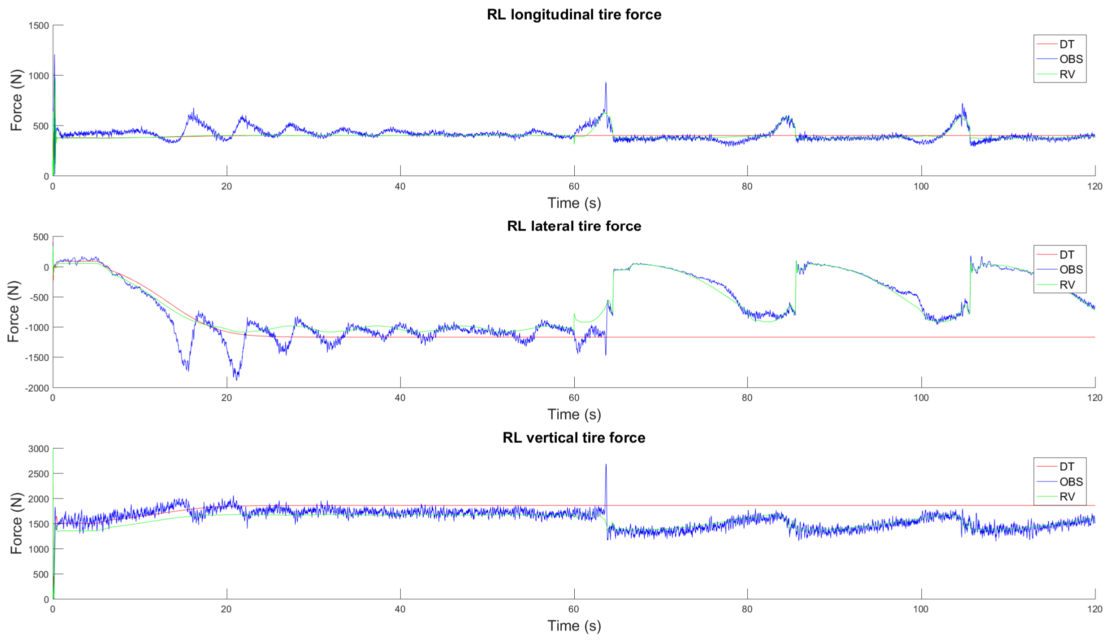


Figure 5.56: Tire forces of the RL wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver under changes in the tire-road friction coefficient.

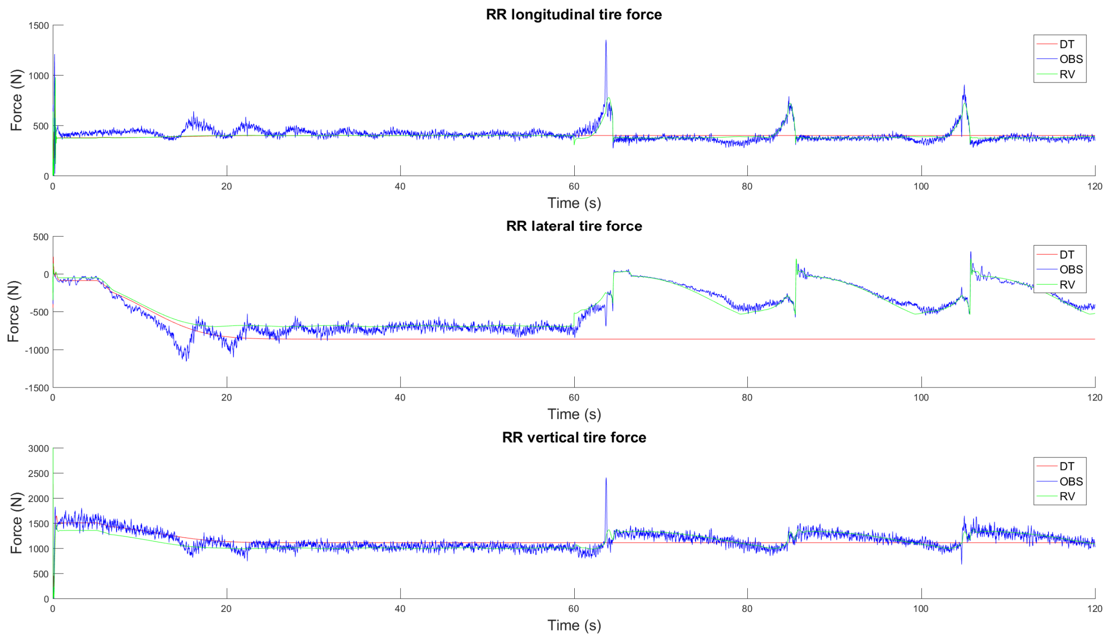
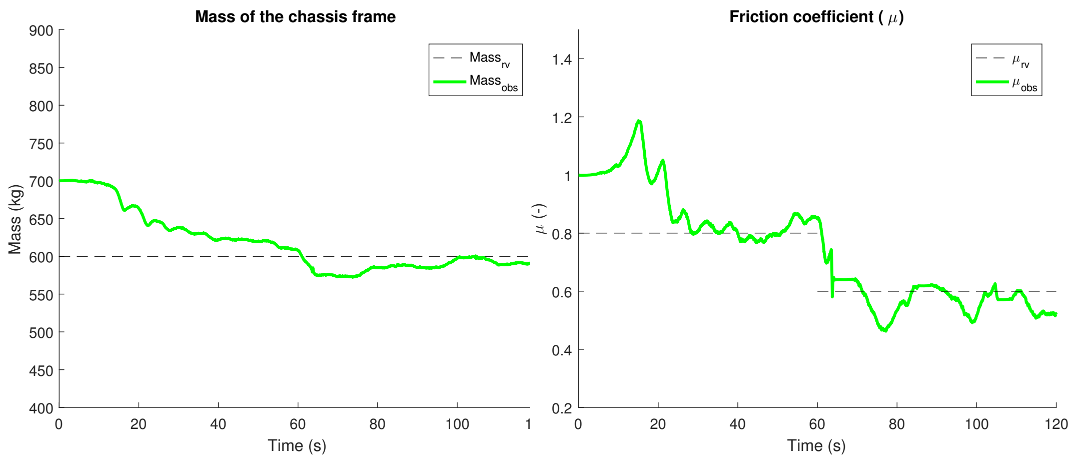


Figure 5.57: Tire forces of the RR wheel estimated by the SPI observer based on the simplified vehicle model for the maneuver under changes in the tire-road friction coefficient.

The effect of the covariance noises is also observed on the parameter estimation. The tire-road friction coefficient is tuned in such a way that the filter can increase or reduce its value faster than in the previous maneuvers. As a counterpart, the noise of the coefficient is increased. The mass and tire-road friction coefficient estimation are shown in Figures 5.58a and 5.58b respectively.



(a) Mass of the chassis. (b) Tire-road friction coefficient.

Figure 5.58: Estimated parameters in the maneuver under changes in the tire-road friction coefficient.

As in previous maneuvers, the RMSE is calculated for the position, acceleration and tire forces of the RR wheel. The errors are presented in Table 5.6.

Table 5.6: RMSE measured in the maneuver under changes in the tire-road friction coefficient for testing the SPI observer combined with the simplified vehicle model.

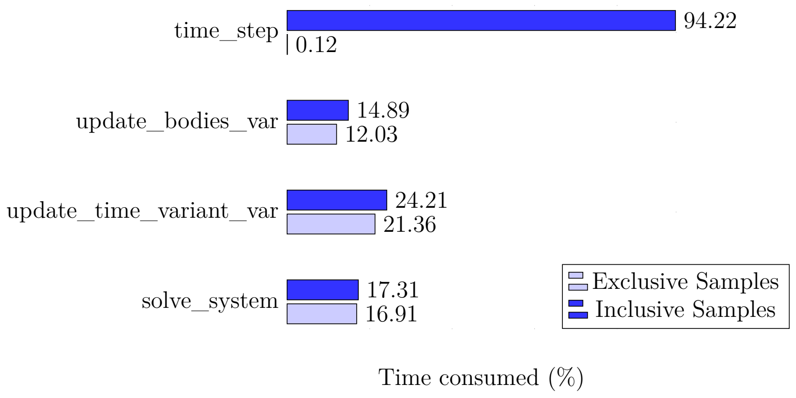
Root-mean-square error				
Magnitude	SPI	SPI ($\mu = 0.8$)	SPI ($\mu = 0.6$)	Sensor
Position (m)	0.7614	0.5565	0.9218	2.0812
X accel. (m/s^2)	0.1382	0.1456	0.1304	0.4492
Y accel. (m/s^2)	0.6062	0.6279	0.5837	0.4470
Z accel. (m/s^2)	0.2677	0.3535	0.1354	0.4485
RR long. tire force (N)	51.39	55.63	46.77	-
RR lat. tire force (N)	87.51	108.75	59.07	-
RR vert. tire force (N)	104.59	117.78	89.48	-

In *Maneuver 3*, the RMSE is higher than in *Maneuver 1* and *Maneuver 2*. As explained above, the tuning of the covariance matrix for *Maneuver 3* was made in order to achieve a trade off between *Maneuver 1* and *Maneuver 2*. Hence, the achieved results cannot offer the same level of accuracy. However, regarding the tire forces, the errors are around 15% in the lateral force, while in the longitudinal and vertical forces is about 10%. If the RMSE is calculated before and after the change in the tire-road friction coefficient, it can be seen that the errors with a tire-road friction coefficient of 0.8 are higher than when the tire-road friction coefficient is reduced. When the tire-road friction coefficient is of 0.6, the errors in the estimated tire forces are less than 10%.

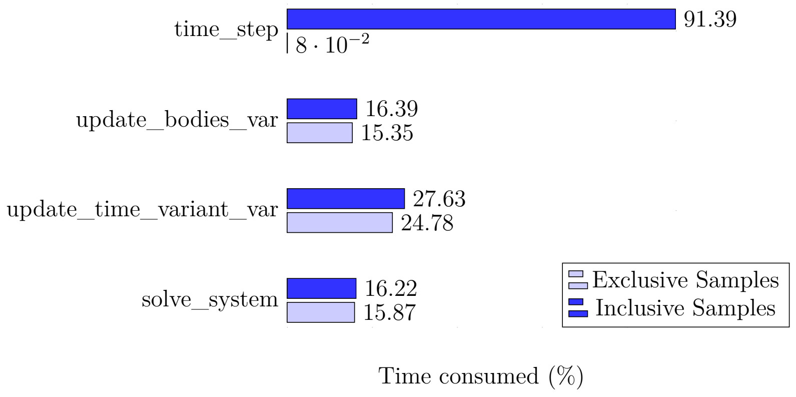
5.3.2.3 Implementation Results

Following the tests performed in Section 5.2.4.2, the implementation of the observer proposed based on the simplified vehicle model is addressed. Since the model has change, the procedure presented in Section 4.3 is followed in order to accelerate the simulation when the simplified vehicle model is used.

The first step is to profile the code in order to detect the bottlenecks of the simulation. The most time consuming functions are presented in Figure 5.59. The profiling has been made for the two observers implemented in this work.



(a) errorEKF



(b) SPI observer.

Figure 5.59: Profiling summary of a simulation of the simplified vehicle MB model combined with the errorEKF and the SPI observer. Only the most time consuming operations are included.

As for the full vehicle model, the most time consuming tasks are the functions `update_bodies_var()`, `update_time_variant_var()` and `solve_system()`. In this case, due to change of the model size, the percentages are different. The function `update_time_variant_var()` is the most time consuming and it could lead to the highest acceleration.

The implementation of each function is addressed following the guidelines presented in 4.3. It should be remarked that, although the MB model size has been reduced, the functions `update_bodies_var()` and `update_time_variant_var()` could not be fully implemented in the FPGA. From the results of Table 5.2, it was concluded that the partial implementation of both functions did not offer any acceleration. Thus, only the GJ implementation will be used for accelerating the observer based on the simplified MB model. The summary of each implementation is presented in Tables 5.7, 5.8 and 5.9. The GJ algorithm can be executed in the FPGA at a frequency of 120 MHz.

Table 5.7: Summary report of the FPGA implementation for computing the global mass matrix in the simplified vehicle model.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	3145	2	92	46	119
Read Input data	98	0	0	≈0	≈0
Global Mass Matrix	2585	1	92	45	118
Write Output data	455	0	0	≈0	≈0

Table 5.8: Summary report of the FPGA implementation for the post-processing operations in the simplified vehicle model.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	4442	10	79	20	125
Read Input data	95	0	0	≈0	3
Variables Update	3907	9	79	19	109
Write Output data	432	0	0	≈0	13

Table 5.9: Summary report of the FPGA implementation of the GJ algorithm in the simplified vehicle model.

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	1704	≈0	54	17	56
Read Input data	210	0	0	≈0	≈0
Gauss-Jordan	1458	≈0	54	16	55
Write Output data	14	0	0	≈0	≈0

As stated during Section 5.3.2.2, the SPI estimator offers a higher level of accuracy than the errorEKF. However, it implies a higher computational costs due to the UKF employed for the parameter estimation. Therefore, both approaches are implemented on the Zynq-7000 in order to compare their performance. The methodology for evaluating the observers is similar as the one followed in Section 5.2.4.2.

A simulation of the same maneuver under each observer is launched on the ARM processor. After that, the simulations where the system of Equation 3.33 is solved through the Gauss-Jordan FPGA implementation are executed. Since three maneuvers have been presented in this Section, *Maneuver 2* is selected for the implementation tests, as it demands more iterations to converge. In a first test, the time step proposed is 4 milliseconds. The results are shown in Table 5.10.

Table 5.10: Summary report of the comparison within the ARM execution of both observers based on the simplified vehicle model and the Gauss-Jordan FPGA implementation proposed in Section 4.3. The simulation is executed with a time step of 4 millisecond and with a tolerance for the integration of $1 \cdot 10^{-5}$

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference (errorEKF)	10	0.004	7.673	1.554	$1 \cdot 10^{-5}$
Reference (SPI-observer)	10	0.004	21.114	1.512	$1 \cdot 10^{-5}$
GJ FPGA (errorEKF)	10	0.004	7.465	1.511	$1 \cdot 10^{-5}$
GJ FPGA (SPI-observer)	10	0.004	20.158	1.544	$1 \cdot 10^{-5}$

From the results presented in Table 5.10, it can be concluded that the errorEKF combined with the simplified MB model runs in real time even without the FPGA for a time step of 4 milliseconds. Meanwhile, the SPI observer requires more than double the time for computing the simulation. With respect to the FPGA implementation, the acceleration achieved in both cases is less than 5%, which is a low factor compared to the acceleration appreciated with the full vehicle model. As explained in Section 4.3, this behavior is as expected, since the potential of the FPGA is higher when the system involves more variables.

With the approach presented, the errorEKF provides virtual sensor measurements at 250 Hz. However, it is interesting for automotive applications to provide estimations at a higher frequency, such as 1000 Hz, which corresponds with an integration time step of 1 milliseconds. The margin from real-time execution for the simulation with a time step of 4 milliseconds is low and, thus, it is not expected to achieve real-time execution with a time step of 1 millisecond. However, in order to measure the increment on computational cost related with the time step reduction, simulations with a time step of 1 milliseconds are launched. Furthermore, the SPI observer is also simulated with a time step of 8 milliseconds in order to evaluate its performance. The results are shown in Table 5.11.

Table 5.11: Summary report of the comparison within the ARM execution of both observers based on the simplified vehicle model and the Gauss-Jordan FPGA implementation proposed in Section 4.3 with additional time steps.

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference (errorEKF)	10	0.001	30.054	1.318	$1 \cdot 10^{-5}$
Reference (SPI-observer)	10	0.008	13.456	3.055	$1 \cdot 10^{-5}$
Reference (SPI-observer)	10	0.008	9.381	1.046	$2 \cdot 10^{-4}$
GJ FPGA (errorEKF)	10	0.001	29.330	1.299	$1 \cdot 10^{-5}$
GJ FPGA (SPI-observer)	10	0.008	12.843	3.134	$1 \cdot 10^{-5}$
GJ FPGA (SPI-observer)	10	0.008	8.957	1.047	$2 \cdot 10^{-4}$

As presented in Table 5.11, it can be noticed that the errorEKF is far from reaching real-time execution if the simulation is executed with a time step of 1 millisecond, neither with the FPGA implementation of the Gauss-Jordan algorithm. With respect to the SPI observer, increasing the time step to 8 milliseconds reduced the computational cost of the simulation. However, the simulation was still not reaching real-time execution. In this case, if the tolerance of the integration is reduced one order of magnitude, the SPI observer can run in real time. The effects of reducing the tolerance can be seen in Figure 5.60. Although the mean value of the tire forces can be approximated to the target value, the noise of the estimations is incremented.

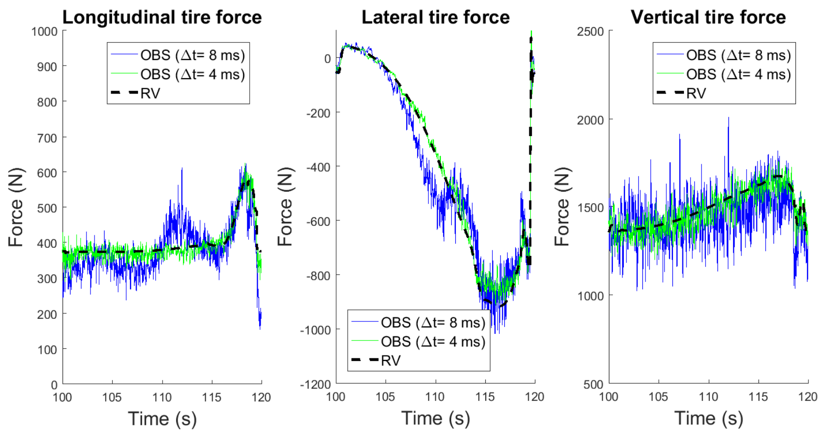


Figure 5.60: Comparison of the tire forces estimated through the SPI observer with different time steps. The results represent the tire forces of the RL wheel during a cycle of the maneuver (from skid to skid).

Through the RMSE calculation, the error derived from increasing the time step can be quantified. In Table 5.12, the error in the tire forces of the RR wheel are presented for the SPI observer executed with a time step 4 milliseconds and 8 milliseconds, together with the estimations of the errorEKF. The RMSE is calculated once that the SPI observer is stabilized.

Table 5.12: RMSE measured in *Maneuver 2* for testing the SPI observer combined with the simplified vehicle model with different time steps.

Root-mean-square error			
Magnitude	SPI ($\Delta t = 4ms$)	SPI ($\Delta t = 8ms$)	errorEKF ($\Delta t = 4ms$)
RR long. tire force (N)	28.41	57.12	165.32
RR lat. tire force (N)	41.27	63.06	265.58
RR vert. tire force (N)	75.30	135.73	180.09

The RMSE shows that, although the errors increase with respect to the SPI observations at a time step of 4 milliseconds, the estimated tire forces are still more accurate than the estimations obtained with the errorEKF.

5.4 FMI 2.0 Standard

In order to reduce the complexity of the in-vehicle implementation of the presented observer, the framework has been adapted to the FMI 2.0 Standard. The standard defines an interface to exchange models and it is supported by several commercial tools.

With the actual framework, the module of the observer can be seen as a black-box for future users. In the standard, a model is referred as Functional Mock-up Unit (FMU). Through a configuration file, also defined by the standard, the features and options of the observer are described, as the inputs and outputs of the model. In this approach, the required inputs are the sensor measurements, i.e GPS position and speed, accelerometer and gyroscope data, suspension displacement measurements, wheel angle data and the driver actions (steering, throttle and brake signals). The outputs can be configured depending on the user preferences.

The standard considers two use-cases for the FMU: model exchange and co-simulation. The main difference is that in model exchange, the integration process is performed outside the FMU. In the observer, the integration is already included in the FMU. The framework build by the user has to control the simulation time and sensor interface, and call the FMU each time step.

The FMI 2.0 Standard is widely used in the industry. Adapting the presented observer to the standard reduces the work required for the introduction of the approach in a real virtual sensing framework.

5.5 Summary

Virtual sensors increase the information available for a system, offering better insight of its behavior. In automotive applications, variables such as tire forces are of high interest. However, the sensor available in the market for measuring them is costly and, therefore, is only employed on particular tests during the design phase.

With the objective of providing accurate virtual sensors to the automotive industry, in this chapter a MB model of a real vehicle is developed using an efficient MB formulation. Combining this model with a state observer, the tire forces, together with more variables are estimated. However, the estimation of the tire forces is not satisfactory. Furthermore, the proposed approach is far from reaching real-time execution when the simulation is launched on new automotive hardware.

As an alternative, the suspension system of the vehicle modeled is simplified: their kinematics are simulated off-line and introduced into the model through look-up tables, as proposed in [97]. Although the combination of this new MB model with the errorEKF increases the efficiency of the simulation enough to achieve real time, the accuracy of the estimated magnitudes could be improved.

Based on the errorEKF, a new filter is developed and presented including parameter estimation. Through a UKF, parameters such as the mass of the chassis and the tire-road friction coefficient are estimated. With this new approach, the errors appearing in the results of the previous state observer are removed. The new filter, thus, provides accurate tire force estimation even on aggressive maneuvers. However, the parameter estimation has associated an increment of computational cost. Hence, real-time performance is only achieved with a large time step and a low tolerance of the simulation, which reduces the accuracy of the estimations.

It should be remarked that the platform employed in this work, the Zynq-7000 ZC702, has the basic heterogeneous processor from Xilinx®. There are on the market more powerful devices that offer a higher performance. Testing the presented observer in other devices could result in a better performance.

To summarize, two of the presented approaches achieved real-time execution: errorEKF based on the simplified vehicle model with a time step of 4 milliseconds, and the SPI observer based on the simplified vehicle model with a time step of 8 milliseconds. From both solutions, although the errorEKF provides estimations at 250 Hz, its estimated tire forces are quite less accurate than with the SPI observer. Hence, although the frequency of the virtual sensor is of 125 Hz, the tire forces can be estimated in-vehicle with a approximate 10% of error.

Chapter 6

Conclusions and Future Work

This chapter summarizes the work developed during this thesis and presents the conclusions. In addition, the future work is also discussed.

6.1 Conclusions

The use of MB simulation techniques in state estimation can contribute to improve the quality of the estimations. Through the MB approach, a system can be modeled with a high level of detail. This means that, in the case of automotive applications for example, estimations in many different type of maneuvers can be accurately performed. In most cases, analytical approaches have limitations when applied in maneuvers with high non-linearities. Through a MB approach, this issue can be overcome. In addition, more information can be made available when using MB models.

The main disadvantage of using MB models for state estimation in automotive applications is related to their high computational cost. The estimations, or virtual sensor measurements, must also be performed in real time on embedded hardware, where the computational power is limited. In-vehicle hardware must deal with complex requisites, such as power consumption, reducing its computational capabilities. Hence, state observers based on analytical models were commonly used.

As the technology evolves, new generation devices are being developed. Thanks to heterogeneous processors, the computational power of embedded platforms is increasing, while keeping low energy consumption. In this new trend, ARM processors combined with an FPGA as co-processor are an interesting option: the FPGA allows achieving high level of code acceleration with flexibility and low demand of energy. This new devices give an opportunity for implementing virtual sensors based on MB dynamics on automotive hardware.

For that purpose, the use of FPGAs for accelerating MB simulations was studied in this thesis. From the presented results, it can be concluded that FPGAs can accelerate MB simulations. During this work, multiple conclusions were extracted. First, due to the limited resources of an FPGA, only the most time consuming tasks of the simulation should be offloaded to it. Since FPGAs are hardware devices, the algorithm should be modified in order to exploit the advantages of FPGAs in terms of parallelization, for what a basic knowledge in hardware design is required.

Through this thesis, guidelines for a proper use of FPGAs were presented. The developed designs can be taken as reference for future implementations. Finally, it should be noted that the FPGA should be selected regarding the size of the MB model. Otherwise, the lack of resources would result in a poor acceleration. It was seen that the size of the model is also relevant. Models with a high size or repeated subsystems can lead to higher accelerations.

Regarding the virtual sensors, a new Dual Kalman filter for state, parameter and input estimation based on MB models was developed. It offers accurate estimations and it can be executed in real time on automotive hardware. Several conclusions can be derived. First, the cost of simulating a complete MB vehicle model demands much more computational capabilities than available on the target hardware of this thesis. Thus, it is suggested to model the vehicle in independent coordinates, as proposed in [97], replacing the suspension system for kinematic tables. This approach reduces significantly the computational cost while keeping all the advantages of MB dynamics.

For generating the code of the MB model, the `MBScoder` was employed. It is an automatic source-code generation tool. It allows to automatically create the required code for simulating MB model, reducing the programming time. This thesis has contributed to the development of the `MBScoder` with a module for using relative coordinates based on a semi-recursive formulation.

Initially, the errorEKF with force estimations was selected for the virtual sensing approach. This filter performs corrections in position, velocities and accelerations, allowing to estimate forces. It was previously tested in planar mechanisms showing accurate results. In this work, its application on a vehicle is addressed, showing that some modifications should be made. The original version assumes that the correction in forces is applied over the DOFs. However, the errors at the input forces must be added to the vehicle through the tires and not directly on the DOFs. Since there are four wheels, a successful procedure to achieve it is mapping the forces by means of a least squares projection. This addition must be respectful with the tire model dynamics: tire deflections must be updated correspondingly.

Although the filter should correct the drift between the model and the reference, it was seen that the errorEKF was not able to correct the errors in the mass and/or tire-road friction coefficient. Thus, a UKF-based parameter estimator was added to the errorEKF, resulting in a new Dual Kalman filter for state, parameter and input estimation. Due to the high computational cost of the UKF, only a reduced set of parameters should be estimated. In this case, vehicle mass and tire-road friction coefficient were estimated because it has been determined that they play a more sensitive role on vehicle dynamics than other dynamic parameters (inertial properties, geometrical dimensions, suspension characteristics, etc).

The estimations derived from the developed state-parameter-input observer are reasonably accurate when compared to the ground truth. Regarding the tire force estimations, which are of high interest, the error was under the 8%. With respect to the computational cost, due to the UKF, real-time performance is only achieved with a time step of 8 milliseconds. Meanwhile, the errorEKF can be executed at 4 milliseconds. However, comparing the accuracy of the estimations, the SPI at 8 milliseconds is able to provide better estimations. The tire forces can be estimated on automotive hardware with approximately a 10% of error in real time, with a sampling frequency of 125 Hz.

The developed observer has also been adapted to the FMI 2.0 Standard. The standard defines an interface to exchange models, and it is supported by several tools. Hence, the MB-model state observer presented can be integrated into multiple systems without additional complexity.

6.2 Future Work

To further extend the research done in this thesis and install virtual sensors in real automotive applications, the following research lines can be explored. The first one is related to the validation of the state observer. Now that real-time execution is achieved, it should be tested on-board a vehicle with real sensor data in real maneuvers.

The presented observer can also be improved in some aspects. On the one hand, the tuning of the covariance matrix is a difficult process full of uncertainties. Furthermore, the covariance matrix should be modified if the maneuver changes significantly. Due to the strong influence of this covariance matrix in the estimations, its tuning process should be explored. Also, the on-line tuning of the covariance matrix during a maneuver can be explored. On the other hand, the efficiency of the parameter estimator can be improved if the UKF is replaced by an EKF. The main limitation is the Jacobian matrix of the parameters with respect to the MB variables. There are different techniques for obtaining a numerical approximation of the Jacobian that can be explored. It should also be considered to derive the analytical expressions for the Jacobian.

With respect to the hardware implementations, FPGAs offer the possibility of accelerating MB simulations. However, in this thesis the potential of the solutions was limited by the reduced size of the FPGA available. Tests with higher-capacity FPGAs can be performed in order to explore the real potential of these devices. It is also of interest to develop a procedure for optimally select the best candidates of a MB simulation to be accelerated on FPGAs.

Bibliography

[1] Javier Cuadrado, Daniel Dopico, Antonio Barreiro, and Emma Delgado. Real-time state observers based on multibody models and the extended Kalman filter. *J Mech Sci Technol*, 23(4):894–900, April 2009.

[2] Javier Cuadrado, Daniel Dopico, Jose A. Perez, and Roland Pastorino. Automotive observers based on multibody models and the extended Kalman filter. *Multibody Syst Dyn*, 27(1):3–19, April 2011.

[3] Roland Pastorino. *Experimental Validation of a Multibody Model for a Vehicle Prototype and its Application to Automotive State Observers.* PhD thesis, Universidade da Coruña, 2012.

[4] Emilio Sanjurjo. *State observers based on detailed multibody models applied to an automobile.* PhD thesis, Universidade da Coruña, 2016.

[5] Roland Pastorino, Francesco Cosco, Frank Naets, Wim Desmet, and Javier Cuadrado. Hard real-time multibody simulations using ARM-based embedded systems. *Multibody Syst Dyn*, 37(1):127–143, May 2016.

[6] Nicolas Navet and Françoise Simonot-Lion, editors. *Automotive embedded systems handbook.* Industrial information technology series. CRC Press, Boca Raton, 2009. OCLC: ocn231680179.

[7] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and Robert W. Stewart. *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC.* Strathclyde Academic Media, first edition, 2014.

[8] Eduardo Valentin, Rosiane de Freitas, and Raimundo Barreto. Towards optimal solutions for the low power hard real-time task allocation on multiple heterogeneous processors. *Sci. Comput. Program.*, 165:38–53, November 2018.

[9] David Crolla. *Encyclopedia of Automotive Engineering.* John Wiley & Sons, March 2015. Google-Books-ID: ANfdCgAAQBAJ.

[10] Kay-Uwe Henning and Oliver Sawodny. Vehicle dynamics modelling and validation for online applications and controller synthesis. *Mechatronics*, 39:113–126, November 2016.

[11] Davide Tavernini, Efstathios Velenis, and Stefano Longo. Feedback brake distribution control for minimum pitch. *Vehicle System Dynamics*, 55(6):902–923, June 2017.

[12] Paul J. TH Venhovens and Karl Naab. Vehicle Dynamics Estimation Using Kalman Filters. *Vehicle System Dynamics*, 32(2-3):171–184, August 1999.

[13] U. Kiencke and A. Daiß. Observation of lateral vehicle dynamics. *Control Engineering Practice*, 5(8):1145–1150, August 1997.

[14] J. Stephant, A. Charara, and D. Meizel. Virtual sensor: application to vehicle sideslip angle and transversal forces. *IEEE Transactions on Industrial Electronics*, 51(2):278–289, April 2004.

[15] M. C. Best, T. J. Gordon, and P. J. Dixon. An Extended Adaptive Kalman Filter for Real-time State Estimation of Vehicle Handling Dynamics. *Vehicle System Dynamics*, 34(1):57–75, July 2000.

[16] Yuhang Chen, Yunfeng Ji, and Konghui Guo. A reduced-order nonlinear sliding mode observer for vehicle slip angle and tyre forces. *Vehicle System Dynamics*, 52(12):1716–1728, December 2014.

[17] S. Drakunov and V. Utkin. Sliding mode observers. Tutorial. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 4, pages 3376–3378 vol.4, December 1995.

[18] Iraj Davoodabadi, Ali Asghar Ramezani, Mehdi Mahmoodi-k, and Pouyan Ahmadizadeh. Identification of tire forces using Dual Unscented Kalman Filter algorithm. *Nonlinear Dynamics*, 78(3):1907–1919, November 2014.

[19] P. Freeman, J. Wagner, and K. Alexander. Run-off-road and recovery – state estimation and vehicle control strategies. *Vehicle System Dynamics*, 54(9):1317–1343, September 2016.

[20] Zhenpo Wang, Jianyang Wu, Lei Zhang, and Yachao Wang. Vehicle sideslip angle estimation for a four-wheel-independent-drive electric vehicle based on a hybrid estimator and a moving polynomial Kalman smoother. *Proceedings of the IMechE*, page 1464419318770923, April 2018.

[21] T. A. Wenzel, K. J. Burnham, M. V. Blundell, and R. A. Williams. Dual extended Kalman filter for vehicle state and parameter estimation. *Vehicle System Dynamics*, 44(2):153–171, February 2006.

[22] Giulio Reina, Matilde Paiano, and Jose-Luis Blanco-Claraco. Vehicle parameter estimation using a model-based estimator. *Mechanical Systems and Signal Processing*, 87:227–241, March 2017.

[23] Justus Jordan, Nils Hirsenkorn, Felix Klanner, and Martin Kleinsteuber. Vehicle mass estimation based on vehicle vertical dynamics using a multi-model filter. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 2041–2046, October 2014. ISSN: 2153-0017.

[24] Khalil Maleej, Sousso Kelouwani, Yves Dube, and Kodjo Agbossou. Event-Based Electric Vehicle Mass and Grade Estimation. In *2014 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pages 1–6, October 2014. ISSN: 1938-8756.

[25] Karol Bogdanski and Matthew C. Best. Kalman and particle filtering methods for full vehicle and tyre identification. *Vehicle System Dynamics*, 56(5):769–790, May 2018.

[26] T.A. Wenzel, K.J. Burnham, M.V. Blundell, and R.A. Williams. Kalman filter as a virtual sensor: applied to automotive stability systems. *Transactions of the Institute of Measurement and Control*, 29(2):95–115, June 2007.

[27] Changfu Zong, Dan Hu, and Hongyu Zheng. Dual extended Kalman filter for combined estimation of vehicle state and road friction. *Chin. J. Mech. Eng.*, 26(2):313–324, March 2013.

[28] Beatriz L. Boada, Daniel Garcia-Pozuelo, Maria Jesus L. Boada, and Vicente Diaz. A Constrained Dual Kalman Filter Based on pdf Truncation for Estimation of Vehicle Parameters and Road Bank Angle: Analysis and Experimental Validation. *IEEE Transactions on Intelligent Transportation Systems*, 18(4):1006–1016, April 2017.

[29] Beatriz L. Boada, Maria Jesus L. Boada, and Hui Zhang. Sensor Fusion Based on a Dual Kalman Filter for Estimation of Road Irregularities and Vehicle Mass Under Static and Dynamic Conditions. *IEEE/ASME Trans. Mechatron.*, 24(3):1075–1086, June 2019.

[30] Javier Garcia de Jalon and Eduardo Bayo. *Kinematic and Dynamic Simulation of Multibody Systems: The Real-Time Challenge.* Springer-Verlag, December 1994. Google-Books-ID: ye_SBwAAQBAJ.

[31] Roland Pastorino, Dario Richiedei, Javier Cuadrado, and Alberto Trevisani. State estimation using multibody models and non-linear Kalman filters. *International Journal of Non-Linear Mechanics*, 53:83–90, July 2013.

[32] Frank Naets, Roland Pastorino, Javier Cuadrado, and Wim Desmet. Online state and input force estimation for multibody models employing extended Kalman filtering. *Multibody System Dynamics*, 32(3):317–336, October 2014.

[33] Ilaria Palomba, Dario Richiedei, and Alberto Trevisani. Kinematic state estimation for rigid-link multibody systems by means of nonlinear constraint equations. *Multibody Syst Dyn*, 40(1):1–22, May 2017.

[34] Ilaria Palomba, Dario Richiedei, and Alberto Trevisani. Simultaneous estimation of kinematic state and unknown input forces in rigid-link multibody systems. In *2015 ECCOMAS Thematic Conference on Multibody Dynamics*, Barcelona, Spain, 2015.

[35] Emilio Sanjurjo, José L Blanco, José L Torres, and Miguel A Naya. Testing the efficiency and accuracy of multibody-based state observers. In *2015 ECCOMAS*

Thematic Conference on Multibody Dynamics, page 13, Barcelona, Spain, June 2015.

[36] Emilio Sanjurjo, Miguel Ángel Naya, José Luis Blanco-Claraco, José Luis Torres-Moreno, and Antonio Giménez-Fernández. Accuracy and efficiency comparison of various nonlinear Kalman filters applied to multibody models. *Nonlinear Dyn*, 88(3):1935–1951, May 2017.

[37] Roland Pastorino, Emilio Sanjurjo, Alberto Luaces, Miguel A. Naya, Wim Desmet, and Javier Cuadrado. Validation of a Real-Time Multibody Model for an X-by-Wire Vehicle Prototype Through Field Testing. *J. Comput. Nonlinear Dynam*, 10(3):031006–031006–11, May 2015.

[38] Emilio Sanjurjo, Daniel Dopico, Alberto Luaces, and Miguel Ángel Naya. State and force observers based on multibody models and the indirect Kalman filter. *Mechanical Systems and Signal Processing*, 106:210–228, June 2018.

[39] Enrico Risaliti, Tommaso Tamarozzi, Martijn Vermaut, Bram Cornelis, and Wim Desmet. Multibody model based estimation of multiple loads and strain field on a vehicle suspension system. *Mechanical Systems and Signal Processing*, 123:1 – 25, 2019.

[40] Jesús Vidal Gil. *Un método general, sencillo y eficiente, para la definición y simulación numérica de sistemas multicuerpo.* PhD thesis, Industriales, 2006.

[41] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, Naehyuck Chang, Sangyoung Park, Younghyun Kim, P. Leteinturier, and H. Adlkofer. Embedded systems and software challenges in electric vehicles. In *2012 Design, Automation & Test in Europe Conference & Exhibition (2012)*, pages 424–429, Dresden, Germany, March 2012. IEEE.

[42] Egil Juliussen, Richard Robinson, and Institute for Prospective Technological Studies. *Is Europe in the driver's seat?: the competitiveness of the european automotive embedded systems industry.* Publications Office, Luxembourg, 2010. OCLC: 870616326.

[43] M. M. Trompouki, L. Kosmidis, and N. Navarro. An open benchmark implementation for multi-CPU multi-GPU pedestrian detection in automotive systems. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 305–312, November 2017.

[44] A. S. Radhamani. Performance Analysis of Homogeneous and Heterogeneous Multicore Processor Using Static and Dynamic Schedulers. *Asian J. Inf. Technol.*, 15:533–541, 2016.

[45] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, November 2005.

[46] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-ISA heterogeneous multi-core architectures: the potential for processor power

reduction. In *22nd Digital Avionics Systems Conference. Proceedings (Cat. No.03CH37449)*, pages 81–92, San Diego, CA, USA, 2003. IEEE Comput. Soc.

[47] F. Brenot, P. Fillatreau, and J. Piat. FPGA based accelerator for visual features detection. In *2015 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6, June 2015.

[48] Y. Zhou, Z. Chen, and X. Huang. A system-on-chip FPGA design for real-time traffic signal recognition system. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1778–1781, May 2016.

[49] Jiaojiao Li, Jiaji Wu, Yang You, and Gwanggil Jeon. Parallel binocular stereo-vision-based GPU accelerated pedestrian detection and distance computation. *Journal of Real-Time Image Processing*, May 2018.

[50] Hammad Mazhar, Toby Heyn, and Dan Negrut. A scalable parallel method for large collision detection problems. *Multibody Syst Dyn*, 26(1):37–55, June 2011.

[51] Dan Negrut, Alessandro Tasora, Hammad Mazhar, Toby Heyn, and Philipp Hahn. Leveraging parallel computing in multibody dynamics. *Multibody Syst Dyn*, 27(1):95–117, January 2012.

[52] Radu Serban, Daniel Melanz, Ang Li, Ilinca Stanciulescu, Paramsothy Jayakumar, and Dan Negrut. A GPU-based preconditioned Newton-Krylov solver for flexible multibody dynamics. *International Journal for Numerical Methods in Engineering*, 102(9):1585–1604, 2015.

[53] José-Carlos Cano, Javier Cuenca, Domingo Giménez, Mariano Saura-Sánchez, and Pablo Segado-Cabezos. A parallel simulator for multibody systems based on group equations. *J Supercomput*, 75(3):1368–1381, March 2019.

[54] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches.* Wiley, Hoboken, 2006.

[55] Christopher Edwards, Sarah K. Spurgeon, Chee P. Tan, and Nitin Patel. *Sliding-Mode Observers*, pages 221–242. Springer London, London, 2007.

[56] Javier Cuadrado, Daniel Dopico, Miguel A. Naya, and M. Gonzalez. Real-Time Multibody Dynamics and Applications. In Giulio Maier, Jean Salençon, Wilhelm Schneider, Bernhard Schrefler, Paolo Serafini, Martin Arnold, and Werner Schiehlen, editors, *Simulation Techniques for Applied Dynamics*, volume 507, pages 247–311. Springer Vienna, Vienna, 2008.

[57] J. Cuadrado, D. Dopico, M. Gonzalez, and M. A. Naya. A Combined Penalty and Recursive Real-Time Formulation for Multibody Dynamics. *Journal of Mechanical Design*, 126(4):602, 2004.

[58] Daniel Dopico. *Formulaciones semi-recursivas y de penalización para la dinámica en tiempo real de sistemas multicuerpo.* PhD thesis, Universidade da Coruña, 2004.

[59] J. Cuadrado, R. Gutiérrez, M. A. Naya, and P. Morer. A comparison in terms of accuracy and efficiency between a MBS dynamic formulation with stress analysis and a non-linear FEA code. *International Journal for Numerical Methods in Engineering*, 51(9):1033–1052, 2001.

[60] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[61] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition*. Artech House, April 2013. Google-Books-ID: t94fAgAAQBAJ.

[62] S.I. Roumeliotis, G.S. Sukhatme, and G.A. Bekey. Circumventing dynamic modeling: evaluation of the error-state Kalman filter applied to mobile robot localization. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1656–1663, Detroit, MI, USA, 1999. IEEE.

[63] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley, Hoboken, 2006.

[64] Mohinder Grewal and Angus Andrews. *Kalman filtering: theory and practice using MATLAB®*. Wiley, New Jersey, 2008.

[65] Johannes Hiltscher, Phanindra Akula, Robin Streiter, and Gerd Wanielik. A flexible automotive systems architecture for next generation ADAS. In *Proceedings of 7th Transport Research Arena TRA*, page 10, Vienna, Austria, April 2018.

[66] Gorka Velez, Ainhoa Cortés, Marcos Nieto, Igone Vélez, and Oihana Otaegui. A reconfigurable embedded vision system for advanced driver assistance. *J Real-Time Image Proc*, 10(4):725–739, December 2015.

[67] Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 225–236, Atlanta, GA, USA, December 2010. IEEE.

[68] Young-Kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms. *ACM Trans. Reconfigurable Technol. Syst.*, 12(1):1–20, February 2019.

[69] Jose Nunez-Yanez, Sam Amiri, Mohammad Hosseinabady, Andrés Rodríguez, Rafael Asenjo, Angeles Navarro, Dario Suarez, and Ruben Gran. Simultaneous multiprocessing in a software-defined heterogeneous FPGA. *J. Supercomput.*, April 2018.

[70] Martin Dendaluce Jahnke, Francesco Cosco, Rihards Novickis, Joshué Pérez Rastelli, and Vicente Gomez-Garay. Efficient Neural Network Implementations on Parallel Embedded Platforms Applied to Real-Time Torque-Vectoring Optimization Using Predictions for Multi-Motor Electric Vehicles. *Electronics*, 8(2):250, February 2019.

[71] Yvonne Lin. Using FPGAs to Solve Challenges in Industrial Applications. Technical report, Xilinx, November 2011.

[72] Jason Chiang and Stefano Zammattio. Five Ways to Build Flexibility into Industrial Applications with FPGAs. Technical report, Intel.

[73] Óscar Mata-Carballeira, Jon Gutiérrez-Zaballa, Inés del Campo, and Victoria Martínez. An FPGA-Based Neuro-Fuzzy Sensor for Personalized Driving Assistance. *Sensors*, 19(18):4011, September 2019.

[74] Andrew Moore. *FPGAs For Dummies®, 2nd Intel® Special Edition.* John Wiley & Sons, Inc., 2017.

[75] Y. Han and E. Oruklu. Real-time traffic sign recognition based on Zynq FPGA and ARM SoCs. In *IEEE International Conference on Electro/Information Technology*, pages 373–376, June 2014.

[76] Henning Sahlbach, Daniel Thiele, and Rolf Ernst. A system-level FPGA design methodology for video applications with weakly-programmable hardware components. *J Real-Time Image Proc*, 13(2):291–309, June 2017.

[77] Sergio Saponara. Hardware accelerator IP cores for real time Radar and camera-based ADAS. *J Real-Time Image Proc*, 16(5):1493–1510, October 2019.

[78] Abdelhamid Helali, Haythem Ameur, J. M. Górriz, J. Ramírez, and Hassen Maaref. Hardware implementation of real-time pedestrian detection system. *Neural Comput & Applic*, January 2020.

[79] Inés del Campo, Victoria Martínez, Javier Echanobe, Estibalitz Asua, Raúl Finker, and Koldo Basterretxea. A versatile hardware/software platform for personalized driver assistance based on online sequential extreme learning machines. *Neural Comput & Applic*, 31(12):8871–8886, December 2019.

[80] XA Zynq-7000 SoC Data Sheet: Overview. DS188 (v1.3.2). `https://www.xilinx.com/support/documentation/data_sheets/ds188-XA-Zynq-7000-Overview.pdf`, 2018. Accessed: 2020-02-21.

[81] Zynq-7000 SoC Data Sheet: Overview. DS190 (v1.11.1). `https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf`, 2018. Accessed: 2020-02-21.

[82] P. Arato, S. Juhasz, Z. A. Mann, A. Orban, and D. Papp. Hardware-software partitioning in embedded system design. In *IEEE International Symposium on Intelligent Signal Processing, 2003*, pages 197–202, September 2003.

[83] A. Jantsch, P. Ellervee, J. Oberg, and A. Hemani. A case study on hardware/-software partitioning. In *Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines*, pages 111–118, April 1994.

[84] Z. Jiang and S. A. Raziei. An efficient FPGA-based direct linear solver. In *2017 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 159–166, June 2017.

[85] J. Arias-García, R. Pezzuol Jacobi, C. H. Llanos, and M. Ayala-Rincón. A suitable FPGA implementation of floating-point matrix inversion based on Gauss-Jordan elimination. In *2011 VII Southern Conference on Programmable Logic (SPL)*, pages 263–268, April 2011.

[86] Jean Pierre David. Low latency and division free Gauss–Jordan solver in floating point arithmetic. *Journal of Parallel and Distributed Computing*, 106:185–193, August 2017.

[87] Picture of the modelled vehicle (SimRod). `https://autotechreview.com/siemens-automotive-engineering-simulation-and-automation-test-center/atr-blog/good-digital-twins-don-t-lie`. Accessed: 2019-11-14.

[88] E. Kuiper and J. J. M. Van Oosten. The PAC2002 advanced handling tire model. *Vehicle System Dynamics*, 45(sup1):153–167, January 2007.

[89] Egbert Bakker, Lars Nyborg, and Hans B. Pacejka. Tyre modelling for use in vehicle dynamics studies. In *SAE International Congress and Exposition*. SAE International, feb 1987.

[90] Egbert Bakker, Hans B. Pacejka, and Lars Lidner. A new tire model with an application in vehicle dynamics studies. In *Autotechnologies Conference and Exposition*. SAE International, apr 1989.

[91] H. B. Pacejka and I. J. M. Besselink. Magic Formula Tyre Model with Transient Properties. *Vehicle System Dynamics*, 27(sup001):234–249, January 1997.

[92] Hans B. Pacejka, Hans B. Pacejka, and Hans B Pacejka. Preface. In *Tire and Vehicle Dynamics*, pages xiii–xvi. Elsevier, 2012.

[93] W. Hirschberg, G. Rill, and H. Weinfurter. Tire model TMeasy. *Vehicle System Dynamics*, 45(sup1):101–119, January 2007.

[94] Daniel Dopico, Alberto Luaces, Manuel Gonzalez, and Javier Cuadrado. Dealing with multiple contacts in a human-in-the-loop application. *Multibody Syst Dyn*, 25(2):167–183, February 2011.

[95] Thomas D. Gillespie. *Fundamentals of vehicle dynamics*. Society of Automotive Engineers, 1992.

[96] Agus Budiyono. Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems. *Industrial Robot*, 39(3):ir.2012.04939caa.011, April 2012.

[97] Javier Cuadrado, David Vilela, Iñaki Iglesias, Adrián Martín, and Alberto Peña. A Multibody Model to Assess the Effect of Automotive Motor In-wheel Configuration on Vehicle Stability and Comfort. In *2013 ECCOMAS Thematic Conference on Multibody Dynamics*, page 10, 2013.

Appendices

Appendix A

List of publications

This thesis has been financed by the Spanish Ministry of Economy and Competitiveness (MINECO) through the BES-2015-071372 fellowship. Several conference papers have been presented as a result of the work made in this thesis. Moreover, one journal paper related with the execution of MB models on FPGAs has been submitted and it is under the revision process as of the time of writing. In addition, another journal paper is being written presenting the results of the developed SPI observer. The list of publications is exposed hereafter.

Submitted Journal papers

A.J. Rodriguez, R. Pastorino, A. Carro-Lagoa, K. Janssens and M.A. Naya. Hardware Acceleration of Multibody Simulations for Real-Time Embedded Applications. *Multibody System Dynamics* (under review).

Conference communications

A.J. Rodriguez, R. Pastorino, M.A. Naya, E. Sanjurjo and W. Desmet. Real-time Estimation based on Multibody Dynamics for Automotive Embedded Heterogeneous Computing. In *8th ECCOMAS Thematic Conference on Multibody Dynamics*, Prague, Czech Republic, June 2017.

E. Sanjurjo, D. Dopico, M.A. Naya and A.J. Rodriguez. Indirect State and Force Estimator Based on Multibody Models. In *8th ECCOMAS Thematic Conference on Multibody Dynamics*, Prague, Czech Republic, June 2017.

A.J. Rodriguez, R. Pastorino, M.A. Naya and E. Sanjurjo. Virtual Sensing on Automotive Embedded Heterogeneous Platforms. In *15th European Automotive Congress (EAEC 2017)*, Madrid, Spain, October 2017.

A.J. Rodriguez, R. Pastorino, A. Luaces, E. Sanjurjo and M.A. Naya. Implementation of State Observers based on Multibody Dynamics on Automotive Platforms in Real-Time. In *5th Joint Int. Conference on Multibody System Dynamics (IMSD 2018)*, Lisbon, Portugal, June 2018.

E. Sanjurjo, A.J. Rodriguez, D. Dopico, A. Luaces and M.A. Naya. State and input observer for the multibody model of a car. In *5th Joint Int. Conference on Multibody System Dynamics (IMSD 2018)*, Lisbon, Portugal June, 2018.

A.J. Rodriguez, R. Pastorino, E. Sanjurjo, A. Luaces and M.A. Naya. Implementación de Observador de Estados basado en Modelos Multicuerpo en Tiempo Real en Plataformas Embebidas. In *XXII Congreso Nacional de Ingeniería Mecánica*, Madrid, Spain, September 2018.

Appendix B

Resumen extendido

Introducción

La simulación es una herramienta que permite estudiar en un entorno virtual los diferentes fenómenos que afectan a un sistema real. Los usos de la simulación van desde predecir el rendimiento de un producto durante su ciclo de vida, hasta diseñar un proceso de fabricación o detectar posibles fallos de operación. Utilizando un entorno de simulación, un producto se puede someter a múltiples pruebas sin necesidad de construir un prototipo físico. Debido a la reducción del coste final de producto que conlleva, la simulación es ampliamente utilizada en diferentes ámbitos industriales. Una de las principales técnicas para simular mecanismos y analizar su dinámica es la simulación multicuerpo. El uso de la simulación multicuerpo se extiende a aplicaciones aeroespaciales, maquinaria, robótica, biomecánica o automoción.

En el caso de automoción, la dinámica multicuerpo se ha aplicado normalmente durante la fase de diseño. Diferentes parámetros del vehículo se prueban en múltiples maniobras con el fin de encontrar una configuración óptima, buscando mejorar el comportamiento del vehículo y el confort. Además, si la simulación se ejecuta en tiempo real, el comportamiento de un subsistema real (como una suspensión activa) puede integrarse en un sistema virtual para realizar experimentos, en lo que se conoce como Hardware-in-the-loop (HIL) o Human-in-the-loop (HITL), si un humano interviene en la simulación. La respuesta del subsistema ante diferentes situaciones y sus efectos en el vehículo se pueden analizar sin necesidad de instalarlo en un prototipo, reduciendo los costes, tiempos de desarrollo y riesgos.

Por otro lado, para asistir a los conductores en la carretera, se implementan diferentes tipos de controladores. Estos controladores necesitan información del estado del vehículo en cada instante con el fin de actuar correctamente sobre el vehículo. Algunos datos se obtienen de sensores instalados en el vehículo, pero hay muchas variables que no se pueden medir porque el sensor es muy caro o no está presente. Un problema similar se da en los tests de vehículos que se realizan en circuitos. Durante el test, se necesita instrumentar el vehículo con múltiples sensores para recoger datos suficientes para analizar la dinámica del vehículo.

La estimación de estados supone una opción bastante atractiva en ambos casos. En un entorno de simulación, la cantidad de datos disponible es ilimitada. Un modelo de un vehículo podría ir simulándose a bordo del vehículo replicando la maniobra real. Si el modelo fuera perfecto, el modelo haría exactamente lo mismo que el vehículo real y se podría medir cualquier variable en el entorno de simulación. Esta solución se conoce también como sensores virtuales, ya que es equivalente a instalar múltiples sensores en un entorno virtual. Sin embargo, en situaciones reales siempre hay errores en el modelo que provocan que la maniobra simulada no sea idéntica a la real. Para corregir estos errores, se combina el modelo con un estimador de estados. Mediante la información de un conjunto de sensores instalados en el vehículo real, es posible corregir el modelo para asegurar que la simulación se corresponda con la realidad, aumentando la calidad de la información que se puede extraer de la simulación.

Para disponer de sensores virtuales de calidad, es importante reducir los errores asociados al modelo inicial. Así, los modelos multicuerpo son una gran opción ya que permiten simular con precisión los diferentes fenómenos que suceden en la realidad. Sin embargo, los modelos multicuerpo son costosos computacionalmente y, tradicionalmente, se han reemplazado por modelos analíticos para aplicaciones de sensores virtuales. Implementar un estimador u observador de estados basado en un modelo multicuerpo de un vehículo en tiempo real y en el hardware del que disponen los vehículos presenta un gran desafío. Las plataformas que llevan los vehículos se conocen como sistemas empotrados, y tienen menor capacidad de cálculo que los ordenadores convencionales. Esto se debe a que deben cumplir los estrictos requisitos impuestos por los estándares de la automoción en cuanto a frecuencia, fiabilidad y seguridad, además de un bajo consumo energético. Sin embargo, los sistemas empotrados de nueva generación han visto incrementada su potencia de cálculo. La industria de la automoción está incorporando estas nuevas plataformas para satisfacer la demanda de potencia de cálculo de nuevas aplicaciones, tales como los Sistemas Avanzados de Asistencia al Conductor (ADAS) o los complejos controladores que vienen asociados a la electrificación de los vehículos.

El hardware de nueva generación se basa en procesadores heterogéneos, que combinan el procesador principal con un co-procesador para acelerar las tareas más costosas, reduciendo la carga del procesador principal. Entre los co-procesadores más utilizados, resulta especialmente interesante el uso de Field Programmable Gate Array (FPGA). Las FPGAs son conocidas por ser hardware programable, ya que se puede crear un procesador dedicado para una aplicación concreta a través de su programación. El hecho de ser un procesador diseñado específicamente para una aplicación concreta permite paralelizar de múltiples formas un algoritmo en concreto, dando lugar a un gran rendimiento. Además, al ser programable, el proceso de desarrollo es mucho más rápido y flexible que si se construyese físicamente el mismo procesador.

En esta tesis se explora la oportunidad brindada por el hardware de nueva generación para instalar en un vehículo sensores virtuales basados en modelos multicuerpo. El Laboratorio de Ingeniería Mecánica (LIM) de la Universidade da Coruña consta de una gran experiencia en el desarrollo de sistemas multicuerpo, y una línea de investigación en el desarrollo de observadores de estado. Del trabajo previo llevado a cabo en el LIM, se ha obtenido un observador eficiente y de fácil implementación con sistemas multicuerpo, llamado error-state extended Kalman filter (errorEKF) que estima estados y fuerzas [36,38]. Este observador se combina en esta tesis con un modelo multicuerpo de vehículo basado en una eficiente formulación desarrollada por investigadores del grupo, presentada en [56]. La solución adoptada se analiza para detectar las tareas más costosas computacionalmente y diseñar su implementación en una FPGA, con el fin de incrementar el rendimiento de la simulación y alcanzar tiempo real. Finalmente, los resultados en términos de precisión de las estimaciones y coste computacional son evaluados.

Objetivos

El principal objetivo de esta tesis es implementar un observador de estados basado en modelos multicuerpo en un procesador típicamente empleado en automoción, alcanzando ejecución en tiempo real. Para tal propósito, los siguientes objetivos parciales se han establecido:

- Estudiar la idoneidad de las Field Programmable Gate Arrays (FPGAs) para acelerar simulaciones multicuerpo. Las FPGAs están presentes en los procesadores de nueva generación que se utilizan en la industria de la automoción. Esta tesis se centra en estudiar cómo utilizarlas para realizar simulaciones multicuerpo de forma eficiente.
- Desarrollar un observador eficiente y preciso basado en un modelo multicuerpo de un vehículo. La solución alcanzada deberá ser implementada en tiempo real en hardware de automoción, con bajo poder computacional.
- Es necesario disponer de un código eficiente para ejecutar simulaciones multicuerpo en sistemas empotrados. En [5], se presentó una librería (`MBScoder`) para crear modelos multicuerpo para diferentes plataformas y lenguajes de programación. Esta tesis continúa con el desarrollo de esta librería para añadir nuevas funcionalidades de utilidad en este trabajo, tales como nuevas coordenadas y formulaciones multicuerpo.
- Proporcionar un entorno para la estimación de estados basado en modelos multicuerpo que se pueda implementar con facilidad en sistemas reales de automoción.

A partir de los objetivos presentados, del trabajo realizado en esta tesis se han derivado las siguientes contribuciones:

- Se han presentado unas pautas para acelerar simulaciones multicuerpo con la ayuda de Field Programmable Gate Arrays (FPGAs). Los ejemplos mostrados durante la tesis pueden utilizarse para futuras aplicaciones de FPGAs en simulaciones multicuerpo.
- Se ha desarrollado un nuevo observador basado en un filtro dual de Kalman. Es un observador de estados, parámetros y entradas que permite obtener estimaciones con mejor precisión que otros observadores basados en modelos multicuerpo.

- Se ha mejorado la librería `MBScoder` con un nuevo módulo para desarrollar modelos multicuerpo en coordenadas relativas y una formulación semi-recursiva.
- El observador presentado puede ejecutarse en tiempo real en los procesadores que se instalan a bordo de los vehículos. La solución se ha adaptado al estándar FMI 2.0, simplificando su implementación en sistemas reales.

Estructura de la tesis

El cuerpo principal de esta tesis está organizado en los siguientes capítulos:

El **Capítulo 1** constituye la introducción de esta tesis y sirve para situar el trabajo realizado.

El **Capítulo 2** consiste en una revisión del estado del arte en cuanto a estimación de estados en la industria de automoción y en la evolución de los sistemas empotrados comúnmente utilizados en aplicaciones de automoción, centrándose en el hardware de nueva generación.

En el **Capítulo 3** se presenta el observador seleccionado para esta tesis y la formulación multicuerpo utilizada en la modelación del vehículo para alcanzar la máxima eficiencia posible.

En el **Capítulo 4** se analiza el procesador seleccionado para implementar el observador de estados basado en modelos multicuerpo y las diferentes estrategias empleadas para alcanzar una solución eficiente.

En el **Capítulo 5** se presenta el vehículo modelado y los detalles del modelo multicuerpo desarrollado. Así mismo, se explican las simulaciones realizadas para evaluar el comportamiento del observador de estados. Se presentan los resultados en cuanto a precisión de las estimaciones y coste computacional.

En el **Capítulo 6** se tratan las conclusiones extraídas del trabajo realizado. Por último, se trazan las líneas futuras de investigación para continuar con el trabajo realizado.

Metodología

Esta tesis se puede dividir en dos grandes bloques: estudio de la implementación de modelos multicuerpo en procesadores heterogéneos con FPGAs como co-procesadores; y aplicación de un observador de estados basado en un modelo multicuerpo de un vehículo en procesadores como los que se emplean en automoción. En el primer bloque, el código desarrollado para generar el modelo multicuerpo del vehículo y el observador de estados es analizado buscando los cuellos de botella de la simulación. Una vez identificados, se estudia cómo programar la FPGA para reducir el tiempo empleado para esas tareas. En el segundo bloque, el observador de estados se prueba en términos de precisión de las estimaciones y su rendimiento, para lo cual las implementaciones desarrolladas en el primer bloque serán empleadas.

Antes de comenzar con el primer bloque, el modelo multicuerpo del vehículo se ha desarrollado utilizando la librería `MBScoder`. Como se ha comentado anteriormente, se ha empleado la formulación multicuerpo presentada en [56]. Dicha formulación se basa en coordenadas relativas, que no estaban consideradas en la librería `MBScoder`. Es por esto que ha tenido que desarrollarse un módulo específico para el modelado en coordenadas relativas de acuerdo con la formulación semi-recursiva presentada en [56].

Una vez creado el modelo multicuerpo del vehículo, se combina con el algoritmo del observador de estados y se analiza el código resultante. En este trabajo, se ha seleccionado la Zynq-7000 ZC702 de Xilinx® como plataforma para ejecutar la solución propuesta. Este dispositivo incluye un procesador heterogéneo compuesto por un ARM A-9 y una FPGA Artix-7 como co-procesador. Dado que las FPGAs tienen una capacidad limitada, han de seleccionarse tareas específicas en lugar de partes generales de código. Por ejemplo, si se intenta implementar todo el código generado por la `MBScoder` en la FPGA, resultará imposible dada la gran cantidad de operaciones que se involucran. Por tanto, del análisis del código se extraen tres operaciones cuya ejecución resulta más costosa que las demás: el cálculo de la matriz de masas del vehículo, el post-procesado que se efectúa para obtener los resultados de interés y la resolución del sistema de ecuaciones que se debe realizar para integrar un paso de tiempo. Tras probar diferentes implementaciones, se ha comprobado que tanto el cómputo de la matriz de masas como el post-procesado involucran demasiadas operaciones como para programarse completamente en la FPGA. Por tanto, se reduce la cantidad de código programado en la FPGA para ambas tareas, llegando a una solución menos optimizada. Para resolver el sistema de ecuaciones comentado, se ha seleccionado el algoritmo de Gauss-Jordan en base a los estudios presentes en la literatura, logrando una implementación eficiente.

El rendimiento del observador se ha evaluado en un entorno de simulación. El vehículo base es el Kyburz eRod, un vehículo deportivo eléctrico con tracción trasera. Se han empleado dos modelos del vehículo: uno modelado con los datos reales y otro al que se le han introducido errores de modelado. Los errores se han introducido en la masa del chasis y en el coeficiente de rozamiento con la carretera, los cuales siempre presentan una gran incertidumbre. Ambos parámetros son susceptibles de cambiar entre maniobras: la masa varía con el número de pasajeros o equipaje y el coeficiente de rozamiento es dependiente de las condiciones climatológicas, desgaste de los neumáticos o el firme de la carretera. Una vez se combine el modelo con el observador, las variables estimadas por el observador deberían coincidir con las del modelo real. Las diferencias ocasionadas por el error en la masa y coeficiente de rozamiento se solventarían. Para ello, el observador necesita información de sensores instalados en el modelo del vehículo real. Los sensores simulados son los mismos que están instalados en el Kyburz eRod en el que se basa el trabajo de esta tesis: GPS (posición y velocidad), acelerómetro y giróscopo en el chasis, sensor de desplazamiento longitudinal para la suspensión, y un sensor para medir el ángulo de las ruedas. Adicionalmente, se disponen de medidas del ángulo de volante y par de aceleración que son empleadas como entradas del modelo para replicar la maniobra del vehículo real. Utilizando estos sensores en el entorno de simulación, se facilita la posterior integración del observador en el vehículo real.

Por otro lado, se evaluará el coste computacional del observador en la plataforma Zynq-7000, ejecutándose diferentes simulaciones. Como referencia, se lanzará una simulación de una maniobra genérica íntegramente en el procesador ARM. A continuación, la misma maniobra se simulará utilizando cada una de las implementaciones en la FPGA que se han desarrollado: en el procesador ARM se ejecutará todo el código del observador a excepción de las tareas programadas en la FPGA. De esta forma, se podrán evaluar los beneficios derivados de cada implementación.

Resultados

Con el fin de analizar las estimaciones proporcionadas por el observador, se han ejecutado varias maniobras genéricas. Tras una primera simulación, comparando las mediciones de los sensores en el observador con las simuladas en el modelo real, se aprecia una mejora en cuanto a precisión y reducción de ruido. Sin embargo, el mayor interés de los sensores virtuales se encuentra en poder estimar variables cuyos sensores son caros o no existen. Analizando las fuerzas en los neumáticos, que son de gran interés, se ha visto que los errores de modelado no se corrigen y que la calidad de las estimaciones es baja.

En cuanto al coste computacional, el objetivo es conseguir una ejecución en tiempo real con un paso de tiempo de integración razonable, siendo el ideal un paso de tiempo de 1 milisegundo. Un paso de tiempo menor implica un mayor coste computacional, pero da lugar a una mayor precisión de la simulación y a una mayor frecuencia de muestreo para los sensores virtuales. En este contexto, se ha probado inicialmente una simulación de 10 segundos de duración con un paso de tiempo de 4 milisegundos. La ejecución de referencia, en el procesador ARM, tuvo una duración 8 veces por encima de tiempo real. Con el uso de la FPGA, solamente se ha conseguido reducir el tiempo total con la implementación del algoritmo de Gauss-Jordan. A pesar de aumentar la eficiencia en un 20%, el coste computacional es tal que es imposible garantizar tiempo real.

Ante estos resultados, se propone simplificar el modelo multicuerpo del vehículo siguiendo la propuesta presentada en [97]. El sistema de la suspensión se reemplaza por unas tablas que recogen la cinemática de la suspensión en función de su desplazamiento vertical. En el caso del sistema de suspensión delantero, las tablas también dependen del ángulo de la dirección. Con esta solución, la eficiencia de la simulación se incrementa notablemente. El observador basado en el modelo multicuerpo simplificado alcanza tiempo real en el procesador ARM con un paso de tiempo de 4 milisegundos, siendo un 30% más rápido que tiempo real. Sin embargo, debido a que el modelo ha reducido su tamaño, los beneficios de la FPGA se reducen y solamente mejora un 3% a la ejecución íntegra en el procesador ARM.

Sin embargo, a pesar de conseguir rendimiento de tiempo real, el efecto de los errores derivados de la masa y del coeficiente de rozamiento no se corrige adecuadamente. Como alternativa, se ha diseñado un nuevo observador incluyendo la estimación de parámetros combinando el errorEKF con un filtro unscented Kalman filter (UKF). Este observador, llamado state-parameter-input (SPI) observer mejora la calidad de las estimaciones gracias a la estimación de la masa y el coeficiente de rozamiento entre los neumáticos y la carretera. El error en la estimación de fuerzas se reduce notablemente, siendo el error menor del 8%.

Como contrapartida, el hecho de incluír el UKF para la estimación de parámetros conlleva un incremento del coste computacional, y no es posible ejecutar el nuevo observador en tiempo real con un paso de 4 milisegundos en la Zynq-7000. Incrementando el paso de tiempo hasta los 8 milisegundos, se consigue alcanzar tiempo real a costa de un incremento del error en las estimaciones, que sube hasta un 10%. No obstante, esta solución sigue siendo más precisa que el observador basado en el errorEKF.

Conclusiones y trabajo futuro

En esta tesis se ha presentado un nuevo observador basado en modelos multicuerpo que permite estimar variables como las fuerzas en los neumáticos con errores inferiores al 10%. Además, se alcanza tiempo real en plataformas empotradas como las empleadas a bordo de los vehículos en la industria de la automoción.

Durante el desarrollo de esta tesis, se han evaluado las ventajas que aportan las FPGAs para optimizar simulaciones multicuerpo. Se ha presentado una estrategia a seguir para seleccionar adecuadamente las operaciones que pueden acelerarse en una FPGA. Además, se han explicado diferentes enfoques para optimizar las implementaciones en FPGAs. La metodología presentada puede ser seguida para optimizar cualquier operación que se desee en una simulación multicuerpo. En los resultados presentados, se ha llegado a obtener una mejora de un 20% con las implementaciones presentadas. No obstante, dado que la FPGA disponible en esta tesis es de un nivel básico, este rendimiento puede verse incrementado en caso de disponer de una FPGA de mejores características.

En esta tesis se ha presentado también un nuevo observador que combina la estimación de estados y entradas con la estimación de parámetros. Aunque el observador elegido inicialmente, el errorEKF, presentaba un buen rendimiento desde un punto de vista computacional y estimaba correctamente los estados, la estimación de variables como la fuerza de los neumáticos no era todo lo precisa que se esperaba. De los resultados obtenidos, se pudo observar que los errores presentes en el modelo (masa del chasis y coeficiente de fricción) no se estaban corrigiendo adecuadamente. Combinando el errorEKF con un estimador de parámetros basado en un UKF, se mejora notablemente la calidad de las estimaciones. Sin embargo, dado al incremento de coste computacional derivado del UKF, solamente se puede garantizar tiempo real si la simulación se realiza con un paso de tiempo de 8 milisegundos, mientras que con el errorEKF se alcanzaba tiempo real con un paso de 4 milisegundos. No obstante, la precisión de las estimaciones con el nuevo observador SPI mejora a las del errorEKF a pesar de la diferencia de paso de tiempo. La única desventaja, por tanto, radica en la baja frecuencia del sensor virtual, que se reduce de los 250 Hz a los 125 Hz.

Para concluir, del trabajo de esta tesis ha resultado un observador preciso basado en modelos multicuerpo para automoción que alcanza tiempo real en las plataformas empleadas en automoción. Dicho observador se ha adaptado al estandar FMI 2.0, permitiendo su fácil integración con múltiples herramientas empleadas en la industria y, por tanto, su implementación final a bordo de un vehículo. Como consecuencia de buscar aumentar la eficiencia de la simulación, se han obtenido resultados prometedores del uso de FPGAs para acelerar simulaciones multicuerpo. Dado que la mayoría de FPGAs disponibles en el mercado superan las características de la empleada en este trabajo, se espera que los resultados mejoren y que el uso de estos dispositivos sea de mayor utilidad en futuras aplicaciones.

No obstante, el trabajo realizado en esta tesis deja sin cubrir algunos aspectos que pueden constituir futuras líneas de investigación. Por un lado, sería de interés evaluar las implementaciones propuestas en una FPGA con mayor capacidad que la empleada en este trabajo, permitiendo así evaluar el verdadero potencial de estos dispositivos. También sería de interés desarrollar una metodología que permita identificar de forma óptima y automática las partes de código más adecuadas para ser implementadas en una FPGA.

En cuanto al observador, puede mejorarse en varios aspectos. Por un lado, el proceso de ajuste de ciertos parámetros del observador está lleno de incertidumbres. Si la maniobra estudiada cambia notablemente, es necesario reajustar los parámetros, lo cual resulta en un tedioso proceso. Dada la influencia de este ajuste sobre la precisión de las estimaciones, es de gran interés estudiar el proceso de ajuste y definir una metodología eficiente. Además, hay ciertos estudios de autoajuste del observador durante la maniobra, lo cual supondría un incremento de la robustez del observador. En cuanto a la eficiencia del observador, ésta puede mejorar reemplazando el UKF por un extended Kalman filter (EKF) para estimar los parámetros. El principal problema es determinar analíticamente la relación entre los parámetros y las variables del modelo multicuerpo. Existen diferentes técnicas para aproximar dicha relación que deben ser exploradas para reducir el coste computacional del observador, permitiendo reducir el paso de tiempo e incrementar la precisión de los sensores virtuales.

Trabajos derivados de la realización de esta tesis

Esta tesis ha sido financiada por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España mediante la beca BES-2015-071372. Como resultado del trabajo realizado en esta tesis, se han realizado varias comunicaciones en congresos nacionales e internacionales. Además, un artículo de revista sobre la ejecución de modelos multicuerpo en FPGAs ha sido enviado y se encuentra en revisión. Por otro lado, se está en proceso de escritura de un artículo de revista presentando los resultados del observador de estados, parámetros y entradas desarrollado en esta tesis. La lista de los trabajos derivados de esta tesis se expone a continuación.

Artículos de revista enviados

A.J. Rodriguez, R. Pastorino, A. Carro-Lagoa, K. Janssens and M.A. Naya. Hardware Acceleration of Multibody Simulations for Real-Time Embedded Applications. *Multibody System Dynamics* (under review).

Comunicaciones en congresos

A.J. Rodriguez, R. Pastorino, M.A. Naya, E. Sanjurjo and W. Desmet. Real-time Estimation based on Multibody Dynamics for Automotive Embedded Heterogeneous Computing. In *8th ECCOMAS Thematic Conference on Multibody Dynamics*, Prague, Czech Republic, June 2017.

E. Sanjurjo, D. Dopico, M.A. Naya and A.J. Rodriguez. Indirect State and Force Estimator Based on Multibody Models. In *8th ECCOMAS Thematic Conference on Multibody Dynamics*, Prague, Czech Republic, June 2017.

A.J. Rodriguez, R. Pastorino, M.A. Naya and E. Sanjurjo. Virtual Sensing on Automotive Embedded Heterogeneous Platforms. In *15th European Automotive Congress (EAEC 2017)*, Madrid, Spain, October 2017.

A.J. Rodriguez, R. Pastorino, A. Luaces, E. Sanjurjo and M.A. Naya. Implementation of State Observers based on Multibody Dynamics on Automotive Platforms in Real-Time. In *5th Joint Int. Conference on Multibody System Dynamics (IMSD 2018)*, Lisbon, Portugal, June 2018.

E. Sanjurjo, A.J. Rodriguez, D. Dopico, A. Luaces and M.A. Naya. State and input observer for the multibody model of a car. In *5th Joint Int. Conference on Multibody System Dynamics (IMSD 2018)*, Lisbon, Portugal June, 2018.

A.J. Rodriguez, R. Pastorino, E. Sanjurjo, A. Luaces and M.A. Naya. Implementación de Observador de Estados basado en Modelos Multicuerpo en Tiempo Real en Plataformas Embebidas. In *XXII Congreso Nacional de Ingeniería Mecánica*, Madrid, Spain, September 2018.