Politecnico di Milano
Master of Science in Automation and Control Engineering

LFT parameter identification of the cornering stiffnesses based on the Fiala tire model

Supervisor: Prof. Gianni Ferretti

Graduation thesis by
Tommaso Viani
ID 883487

Academic Year 2018 - 2019

Abstract

The goal of this thesis is the identification of the parameters of the Fiala model for tire-road interaction, in the framework of vehicle lateral dynamics. The considered procedure relies on the single-track model for vehicle lateral dynamics, widely used in the literature. The Fiala tire model is embodied in the considered single-track model, allowing to accurately represent the vehicle dynamics even if it is performing drifting maneuvers.
The formulated model is used in an identification algorithm, whose goal is to estimate the cornering stiffnesses of the wheels and the moment of inertia of the vehicle. Parameter identification is based on a Linear Fractional Transform (LFT) formulation of the single-track model, and on the use of a suitable MATLAB toolbox for parameter identification of nonlinear LFT models.
The procedure is run on data obtained from simulations of the same model, to check its effectiveness. The estimated parameters are compared to their corresponding values set for the simulation, expecting coinciding results. It is shown that, although the Fiala model is accurate enough to describe drifting maneuvers, identifiability is loss when drifting data are included. The suggested solution consists in recognizing and removing the detrimental data.
Finally, the algorithms are run on data retrieved from experimental tests, and the results are validated by comparing the measurements of the output variables to the simulated output of the identified model.

KEYWORDS: LFT; Linear-Fractional Transformation; Vehicle; Lateral Dynamics; Single-Track; Tire; Tire model; Tire-road interaction; Fiala; Brush model; Slip; Side slip angle; Beta; Tire slip angle; Alpha; Drifting; Full Sliding;

Sommario

Lo scopo di questa tesi è l'identificazione dei parametri del modello di Fiala per l'interazione tra uno pneumatico e la strada, nell'ambito della dinamica laterale di un veicolo. La procedura considerata fa affidamento sul modello bicicletta, largamente utilizzato nella letteratura. Il modello di Fiala di uno pneumatico è incluso nel modello bicicletta considerato, consentendo di rappresentare accuratamente la dinamica del veicolo anche se effettua manovre *drifting*.
Il modello formulato è usato in un algoritmo di identificazione, il cui compito è stimare le *cornering stiffnesses* delle ruote e il momento di inerzia del veicolo. L'identificazione dei parametri è basata su una formulazione *Linear Fractional Transform* del modello bicicletta, e sull'utilizzo di un opportuno toolbox MATLAB per l'identificazione dei parametri di modelli non lineari LFT.
La procedura è eseguita su dati ottenuti da simulazioni dello stesso modello, per verificarne l'efficacia. I parametri stimati sono confrontati ai rispettivi valori impostati nella simulazione, aspettandosi risultati coincidenti. Inoltre è mostrato che, sebbene il modello di Fiala è sufficientemente accurato per descrivere manovre *drifting*, l'identificabilità è persa quando dati corrispondenti al drifting sono inclusi. La soluzione proposta consiste nel riconoscere e rimuovere tali dati. Infine, gli algoritmi sono testati su dati ottenuti da prove sperimentali, e i risultati sono validati confrontando le misure delle variabili in uscita alle uscite simulate del modello identificato.

PAROLE CHIAVE: LFT; Linear-Fractional Transformation; Veicolo; Dinamica laterale; Bicicletta; Pneumatico; Modello di uno pneumatico; Interazione pneumatico-strada; Fiala; Modello spazzola; Slittamento; Angolo di Side Slip; Beta; Angolo di Tire Slip; Alpha; Drifting; Full Sliding;

Contents

Chapter 1

Introduction

1.1 Motivations

Modelling of dynamic systems has always covered a role of major importance in control applications. An accurate model of a system dynamics is typically determinant in control design, when a choice must be made about the class of the controller, requirements of settling time and robustness or other specifications. A correct modelling is also relevant when simulations of the system are carried out, allowing to foresee the behaviour of complex systems by implementing their mathematical models in computers and integrating them by means of tailored numerical methods. For this reason, the model should not be excessively complicated, or its implementation may be computationally heavy to be run on a computer. Controller design should also take into account for computational complexity, since a regulator is loaded on a MCU (Micro Controller Unit) with limited capabilities. In other words, trade-offs between accuracy and complexity of a model must be considered in most control applications.
This fact also holds for identification algorithms, whose goal is the estimation of the unknown parameters of the model. The choice of the parameters and the nonlinearity of the model are examples of issues which need to be tackled, to formulate a functioning and efficient algorithm.

In the framework of vehicle dynamics, there are plenty of control applications which have become more relevant in the latest period, in both industry and research. One may think of safety instruments, such as ABS and ESC, but also control design aimed to increase performance and automation of the vehicle, found for instance in sport cars. An accurate model of vehicle dynamics could be very useful in these contexts, and a relevant part of vehicle dynamics is covered by the models for tire-road interaction, or *tire model*.

The *single-track model* is widely used in the literature and it is simple and accurate enough to describe the vehicle motion. However, some of its parameters are not trivial to be measured, without using expensive measurement tools. An identifica-

tion algorithm proves to be very useful, if one needs to rely on cheap measurements units, such as an Inertial Measurement Unit (IMU). The identification algorithm exploits a mathematical model of the system to relate the measurements and the unknown parameters, so that it can correctly estimate the latter.
The adopted identification method is based on the toolbox for the identification of nonlinear systems in Linear Fractional Transform formulation, developed by Della Bona et al. [1]. The Linear Fractional Transform (LFT) is a convenient form to write nonlinear systems affected by uncertainty, allowing to separate the linear part of the system, the nonlinear part and the uncertain part. The main advantage of the toolbox is its applicability to any LFT model. The software also allows to set options for the identification procedure, which can be tuned to achieve the desired estimation accuracy and convergence time.
Two realizations of the single-track model in LFT formulation are reported in Appendix B.

The complexity of the identification algorithm is strictly related to the complexity of the model. The single-track model has already been tested in an LFT identification procedure in previous works [9, 11], providing correct and efficient results. In these works, however, a linear tire model is considered, which is simple and accurate for non aggressive driving, but it is not when the system starts drifting. For this reason, this thesis introduces a more accurate tire model, without excessively increasing the computational load of the identification algorithm. The chosen tire model is the one formulated by Fiala [10], which provides an acceptable compromise since it fits experimental data, but it is not as complicated as other tire models, such as the Magic Formula from Pacejka.

The estimation algorithm is tested on both simulated and and experimental data. Promising results are obtained on both ends, but an interesting phenomenon should be pointed out. The Fiala model is accurate for data corresponding to intense drifting, but such data lead to extremely incorrect results in the estimated parameters. The knowledge of the tire model proves to be useful in this case, allowing to detect the responsible data for the loss of identifiability, so that they can be removed and the accuracy of the results can be recovered.

1.2 Organization

This work is organized in the following chapters:

- **Chapter 2** deals with parameter identification for nonlinear models in Linear Fraction Transform formulation.
- **Chapter 3** introduces the single-track model for the vehicle lateral dynamics, the concept of tire model and the Fiala model for tire-road interaction.
- **Chapter 4** is dedicated to the developed identification algorithms, whose goal is the estimation of the unknown parameters of the single-track model.

Different ways to face the problem are discussed, mainly focusing on the choice of the tire model and the related issues.

- **Chapter 5** sums up the conclusions and suggests possible ways to further develop this work.

Chapter 2

The LFT Approach to Parameter Identification of Nonlinear Systems

The problem of parameter identification formulated over Linear Fractional Transform (LFT) model structures has been a subject of active research for more than 10 years, see, e.g., [6, 2, 5]. In particular, the parameter estimation method proposed in [5] is here extended to account for nonlinear models.

Consider a nonlinear, time invariant, multi-input multi-output, continuous-time system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\delta}^o) \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\delta}^o) \end{aligned} \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^p$, are the state and noise-free output vectors, with $\mathbf{u} \in \mathbb{R}^m$ being the input vector, and $\boldsymbol{\delta}^o \in \mathbb{R}^q$ a vector of unknown parameters, and assume as the output observation equation

$$\check{\mathbf{y}}(t_k) = \mathbf{y}(t_k) + \boldsymbol{\varepsilon}(t_k) \tag{2.2}$$

where t_k, $k = 1, \ldots, N$ denotes the sampling instant, and $\varepsilon_i(t_k)$ is a discrete-time, zero-mean, white noise of variance σ_i^2.

The identification problem can be formulated as follows: given the sampled data $\{\mathbf{u}(t_k), \check{\mathbf{y}}(t_k)\}_{k=1}^N$, find the values of parameters $\tilde{\boldsymbol{\delta}}$ minimizing the cost function

$$J(\boldsymbol{\delta}) = \frac{1}{2N} \sum_{k=1}^{N} \mathbf{e}^T(t_k, \boldsymbol{\delta}) \mathbf{W} \mathbf{e}(t_k, \boldsymbol{\delta}) \tag{2.3}$$

where $\mathbf{e}(t_k, \boldsymbol{\delta}) = \check{\mathbf{y}}(t_k) - \hat{\mathbf{y}}(t_k, \boldsymbol{\delta})$ is the prediction error between the measured output $\check{\mathbf{y}}(t_k)$ and the output $\hat{\mathbf{y}}(t_k, \boldsymbol{\delta})$ predicted by model (2.1), using parameters $\boldsymbol{\delta}$ instead of the true parameters $\boldsymbol{\delta}^o$ and $\mathbf{W}$ is a weight matrix.

As it is well known, $\tilde{\boldsymbol{\delta}}$ is a *maximum-likelihood* estimate of the model parameters $\boldsymbol{\delta}$ for output-error plants [8], and can be obtained through well known iterative optimization procedures such as, for example, the Gauss-Newton algorithm:

$$\hat{\boldsymbol{\delta}}(\nu+1) = \hat{\boldsymbol{\delta}}(\nu) - \alpha(\nu)\hat{\mathbf{H}}^{-1}(\hat{\boldsymbol{\delta}}(\nu))\mathbf{g}(\hat{\boldsymbol{\delta}}(\nu)) \tag{2.4}$$

where ν is the iteration number, $\alpha(\nu)$ is the step size, $\mathbf{g}(\boldsymbol{\delta}) : \mathbb{R}^q \to \mathbb{R}^q$ and $\hat{\mathbf{H}}(\boldsymbol{\delta}) : \mathbb{R}^q \to \mathbb{R}^{q\times q}$ are the gradient vector and a positive semi-definite approximation of the Hessian of the cost function with respect to the unknown parameters, respectively:

$$\mathbf{g}(\boldsymbol{\delta}) = \frac{1}{N}\sum_{k=1}^{N} \mathbf{E}^T(t_k,\boldsymbol{\delta})\mathbf{W}\mathbf{e}(t_k,\boldsymbol{\delta}), \tag{2.5}$$

$$\hat{\mathbf{H}}(\boldsymbol{\delta}) = \frac{1}{N}\sum_{k=1}^{N} \mathbf{E}^T(t_k,\boldsymbol{\delta})\mathbf{W}\mathbf{E}(t_k,\boldsymbol{\delta}) \tag{2.6}$$

where $\mathbf{E}(t_k,\boldsymbol{\delta}) \in \mathbb{R}^{p\times q}$ is the Jacobian of $\mathbf{e}(t_k,\boldsymbol{\delta})$ and is given by:

$$\mathbf{E}(t_k,\boldsymbol{\delta}) = \left[\begin{array}{ccc} \frac{\partial \mathbf{e}(t_k,\boldsymbol{\delta})}{\partial \delta_1} & \cdots & \frac{\partial \mathbf{e}(t_k,\boldsymbol{\delta})}{\partial \delta_q} \end{array}\right] \tag{2.7}$$

In turn, rewriting model (2.1) in a Linear Fractional Transform (LFT) formulation allows for a direct computation by simulation of the gradient and approximated Hessian of the cost function [1]:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}_1\mathbf{w}(t) + \mathbf{B}_2\boldsymbol{\zeta}(t) + \mathbf{B}_3\mathbf{u}(t) \tag{2.8}$$

$$\mathbf{z}(t) = \mathbf{C}_1\mathbf{x}(t) + \mathbf{D}_{11}\mathbf{w}(t) + \mathbf{D}_{12}\boldsymbol{\zeta}(t) + \mathbf{D}_{13}\mathbf{u}(t) \tag{2.9}$$

$$\boldsymbol{\omega}(t) = \mathbf{C}_2\mathbf{x}(t) + \mathbf{D}_{21}\mathbf{w}(t) + \mathbf{D}_{22}\boldsymbol{\zeta}(t) + \mathbf{D}_{23}\mathbf{u}(t) \tag{2.10}$$

$$\mathbf{y}(t) = \mathbf{C}_3\mathbf{x}(t) + \mathbf{D}_{31}\mathbf{w}(t) + \mathbf{D}_{32}\boldsymbol{\zeta}(t) + \mathbf{D}_{33}\mathbf{u}(t) \tag{2.11}$$

$$\mathbf{w}(t) = \boldsymbol{\Delta}\mathbf{z}(t) = \mathrm{diag}\{\delta_1^o\mathbf{I}_{r_1},\ldots,\delta_q^o\mathbf{I}_{r_q}\}\mathbf{z}(t) \tag{2.12}$$

$$\boldsymbol{\zeta}(t) = \boldsymbol{\Theta}(\boldsymbol{\omega}(t)) \tag{2.13}$$

where $\mathbf{z} \in \mathbb{R}^{n_z}$, $\boldsymbol{\omega} \in \mathbb{R}^{n_\omega}$, $\mathbf{w} \in \mathbb{R}^{n_w}$, $\boldsymbol{\zeta} \in \mathbb{R}^{n_\zeta}$ are vectors of auxiliary variables, $\mathbf{A}$, $\mathbf{B}_i$, $\mathbf{C}_i$, $\mathbf{D}_{ij}$ are 16 known constant matrices, r_i are the sizes of the corresponding identity matrices $\mathbf{I}_{r_i}$ in the $\boldsymbol{\Delta}$ block and $\boldsymbol{\Theta}(\boldsymbol{\omega}) : \mathbb{R}^{n_\omega} \to \mathbb{R}^{n_\zeta}$ is a known nonlinear vector function.

The model in the LFT formulation, although being formally equivalent to the original one, is now clearly divided in 3 parts (Fig. 2.1):

1. a *linear* part: equations (2.8 – 2.11);
2. a *nonlinear* part: equation (2.13), defined by the vector function $\boldsymbol{\Theta}(\boldsymbol{\omega}(t))$;
3. an *uncertain* part: equation (2.12), where vector $\mathbf{z}(t)$ multiplies matrix $\boldsymbol{\Delta} = \mathrm{diag}\{\delta_1^o\mathbf{I}_{r_1},\ldots,\delta_q^o\mathbf{I}_{r_q}\}$, collecting the unknown parameters.

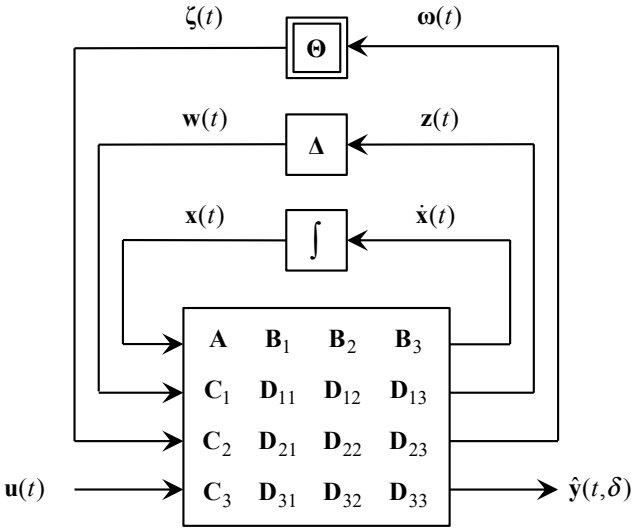

Figure 2.1: LFT formulation

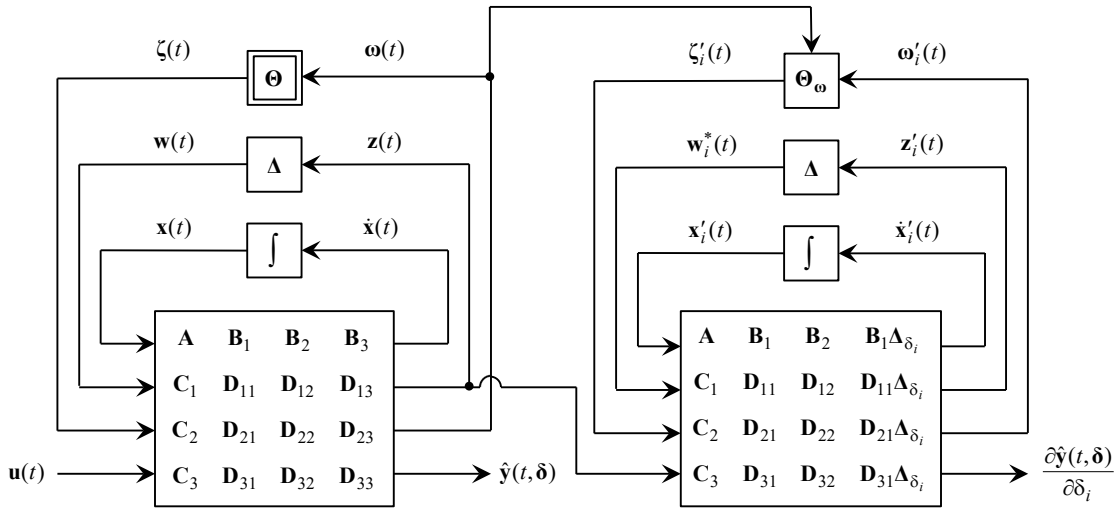

Figure 2.2: Simulation scheme for the computation of the sensitivity functions

The computation of the predicted output $\hat{\mathbf{y}}(t_k, \boldsymbol{\delta})$ (first stage of the scheme in Fig. 2.2) can be dealt with by rewriting model (2.8–2.13) as follows:

$$\begin{aligned} \mathbf{M}\dot{\tilde{\mathbf{x}}}(t) &= \tilde{\mathbf{f}}(\tilde{\mathbf{x}}(t), \mathbf{u}(t, \boldsymbol{\tau}), \boldsymbol{\delta}) && (2.14)\\ \mathbf{y}(t) &= \tilde{\mathbf{g}}(\tilde{\mathbf{x}}(t), \mathbf{u}(t, \boldsymbol{\tau}), \boldsymbol{\delta}) && (2.15) \end{aligned}$$

where $\tilde{\mathbf{x}}(t) = \left[\begin{array}{ccc} \mathbf{x}(t)^T & \mathbf{z}(t)^T & \boldsymbol{\omega}(t)^T \end{array} \right]^T$ and

$$\begin{aligned} \mathbf{M} &= \left[\begin{array}{ccc} \mathbf{I}_n & \mathbf{0}_{n\times n_z} & \mathbf{0}_{n\times n_\omega} \\ \mathbf{0}_{n_z\times n} & \mathbf{0}_{n_z\times n_z} & \mathbf{0}_{n_z\times n_\omega} \\ \mathbf{0}_{n_\omega\times n} & \mathbf{0}_{n_\omega\times n_z} & \mathbf{0}_{n_\omega\times n_\omega} \end{array} \right] && (2.16)\\ \tilde{\mathbf{f}}(\tilde{\mathbf{x}}, \mathbf{u}, \boldsymbol{\delta}) &= \left[\begin{array}{c} \mathbf{A}\mathbf{x} + \mathbf{B}_1\boldsymbol{\Delta}\mathbf{z} + \mathbf{B}_2\boldsymbol{\Theta}(\boldsymbol{\omega}) + \mathbf{B}_3\mathbf{u} \\ \mathbf{C}_1\mathbf{x} + (\mathbf{D}_{11}\boldsymbol{\Delta} - \mathbf{I}_{n_z})\,\mathbf{z} + \mathbf{D}_{12}\boldsymbol{\Theta}(\boldsymbol{\omega}) + \mathbf{D}_{13}\mathbf{u} \\ \mathbf{C}_2\mathbf{x} + \mathbf{D}_{21}\boldsymbol{\Delta}\mathbf{z} + \mathbf{D}_{22}\boldsymbol{\Theta}(\boldsymbol{\omega}) - \boldsymbol{\omega} + \mathbf{D}_{23}\mathbf{u} \end{array} \right] && (2.17)\\ \tilde{\mathbf{g}}(\tilde{\mathbf{x}}, \mathbf{u}, \boldsymbol{\delta}) &= \mathbf{C}_3\mathbf{x} + \mathbf{D}_{31}\boldsymbol{\Delta}\mathbf{z} + \mathbf{D}_{32}\boldsymbol{\Theta}(\boldsymbol{\omega}) + \mathbf{D}_{33}\mathbf{u} && (2.18) \end{aligned}$$

thus by sampling the output of a dynamic system defined by the algebraic transformation output (2.15) and by an index-1, semi-explicit DAE system defined by eq. (2.14), fed by the sampled input $\mathbf{u}(t_k)$.

The numerical integration of the DAE system (2.14) can be dealt with in MATLAB through the `ode15s.m` function, which implements a variable order BDF method and allows to define separately the mass matrix $\mathbf{M}$ and the vector function $\tilde{\mathbf{f}}(\tilde{\mathbf{x}}, \mathbf{u}, \boldsymbol{\delta})$. Moreover, in order to improve reliability and efficiency, the Jacobian matrix $\partial\tilde{\mathbf{f}}/\partial\tilde{\mathbf{x}}$ should be analytically computed.

The sensitivity

$$\frac{\partial \mathbf{e}(t_k, \boldsymbol{\delta})}{\partial \delta_i} = -\frac{\partial \hat{\mathbf{y}}(t_k, \boldsymbol{\delta})}{\partial \delta_i} = -\mathbf{y}'_i(t_k) \qquad (2.19)$$

can be computed by sampling the output $\mathbf{y}'_i(t)$ of the following LFT system (second stage of the scheme in Fig. 2.2):

$$\begin{aligned} \dot{\mathbf{x}}'_i(t) &= \mathbf{A}\mathbf{x}'_i(t) + \mathbf{B}_1\mathbf{w}^*_i(t) + \mathbf{B}_2\boldsymbol{\zeta}'_i(t) + \mathbf{B}_1\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (2.20)\\ \mathbf{z}'_i(t) &= \mathbf{C}_1\mathbf{x}'_i(t) + \mathbf{D}_{11}\mathbf{w}^*_i(t) + \mathbf{D}_{12}\boldsymbol{\zeta}'_i(t) + \mathbf{D}_{11}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (2.21)\\ \boldsymbol{\omega}'_i(t) &= \mathbf{C}_2\mathbf{x}'_i(t) + \mathbf{D}_{21}\mathbf{w}^*_i(t) + \mathbf{D}_{22}\boldsymbol{\zeta}'_i(t) + \mathbf{D}_{21}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (2.22)\\ \mathbf{y}'_i(t) &= \mathbf{C}_3\mathbf{x}'_i(t) + \mathbf{D}_{31}\mathbf{w}^*_i(t) + \mathbf{D}_{32}\boldsymbol{\zeta}'_i(t) + \mathbf{D}_{31}\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) && (2.23) \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{\Delta}_{\delta_i} &= \frac{\partial \boldsymbol{\Delta}}{\partial \delta_i} = \mathrm{diag}\{\mathbf{0}_{r_1\times r_1}, \ldots, \mathbf{I}_{r_i}, \ldots, \mathbf{0}_{r_q\times r_q}\} && (2.24)\\ \mathbf{w}^*_i(t) &= \boldsymbol{\Delta}\mathbf{z}'_i(t) && (2.25)\\ \boldsymbol{\zeta}'_i(t) &= \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \boldsymbol{\omega}'_i(t) = \boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t))\boldsymbol{\omega}'_i(t) && (2.26) \end{aligned}$$

Differentiating eqs. (2.8–2.13) with respect to parameter δ_i yields:

$$\frac{\partial \dot{\mathbf{x}}(t)}{\partial \delta_i} = \mathbf{A}\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{B}_1\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{B}_2\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} \tag{2.27}$$

$$\frac{\partial \mathbf{z}(t)}{\partial \delta_i} = \mathbf{C}_1\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{11}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{12}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} \tag{2.28}$$

$$\frac{\partial \boldsymbol{\omega}(t)}{\partial \delta_i} = \mathbf{C}_2\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{21}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{22}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} \tag{2.29}$$

$$\frac{\partial \hat{\mathbf{y}}(t)}{\partial \delta_i} = \mathbf{C}_3\frac{\partial \mathbf{x}(t)}{\partial \delta_i} + \mathbf{D}_{31}\frac{\partial \mathbf{w}(t)}{\partial \delta_i} + \mathbf{D}_{32}\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} \tag{2.30}$$

$$\frac{\partial \mathbf{w}(t)}{\partial \delta_i} = \frac{\partial \boldsymbol{\Delta}}{\partial \delta_i}\mathbf{z}(t) + \boldsymbol{\Delta}\frac{\partial \mathbf{z}(t)}{\partial \delta_i} \tag{2.31}$$

$$\frac{\partial \boldsymbol{\zeta}(t)}{\partial \delta_i} = \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \frac{\partial \boldsymbol{\omega}(t)}{\partial \delta_i} \tag{2.32}$$

which gives eqs. (2.20–2.26) by renaming the partial derivatives:

$$\mathbf{x}'_i = \frac{\partial \mathbf{x}}{\partial \delta_i}, \quad \mathbf{z}'_i = \frac{\partial \mathbf{z}}{\partial \delta_i}, \quad \boldsymbol{\omega}'_i = \frac{\partial \boldsymbol{\omega}}{\partial \delta_i}, \quad \mathbf{y}'_i = \frac{\partial \hat{\mathbf{y}}}{\partial \delta_i}, \quad \mathbf{w}'_i = \frac{\partial \mathbf{w}}{\partial \delta_i}, \quad \boldsymbol{\zeta}'_i = \frac{\partial \boldsymbol{\zeta}}{\partial \delta_i}$$

By substituting (2.25) and (2.26) in (2.20–2.23) and solving (2.21) and (2.22) with respect to $\mathbf{z}'_i(t)$ and $\boldsymbol{\omega}'_i(t)$ the following time-variant, linear system is obtained,

$$\dot{\mathbf{x}}'_i(t) = \tilde{\mathbf{A}}(\boldsymbol{\omega}(t))\mathbf{x}'_i(t) + \tilde{\mathbf{B}}(\boldsymbol{\omega}(t))\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) \tag{2.33}$$

$$\mathbf{y}'_i(t) = \tilde{\mathbf{C}}(\boldsymbol{\omega}(t))\mathbf{x}'_i(t) + \tilde{\mathbf{D}}(\boldsymbol{\omega}(t))\boldsymbol{\Delta}_{\delta_i}\mathbf{z}(t) \tag{2.34}$$

where

$$\tilde{\mathbf{A}}(\boldsymbol{\omega}(t)) = \mathbf{A} + \begin{bmatrix} \mathbf{B}_1\boldsymbol{\Delta} & \mathbf{B}_2\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} \tag{2.35}$$

$$\tilde{\mathbf{B}}(\boldsymbol{\omega}(t)) = \mathbf{B}_1 + \begin{bmatrix} \mathbf{B}_1\boldsymbol{\Delta} & \mathbf{B}_2\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{D}_{11} \\ \mathbf{D}_{21} \end{bmatrix} \tag{2.36}$$

$$\tilde{\mathbf{C}}(\boldsymbol{\omega}(t)) = \mathbf{C}_3 + \begin{bmatrix} \mathbf{D}_{31}\boldsymbol{\Delta} & \mathbf{D}_{32}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} \tag{2.37}$$

$$\tilde{\mathbf{D}}(\boldsymbol{\omega}(t)) = \mathbf{D}_{31} + \begin{bmatrix} \mathbf{D}_{31}\boldsymbol{\Delta} & \mathbf{D}_{32}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix} \mathbf{F}(\boldsymbol{\omega}(t)) \begin{bmatrix} \mathbf{D}_{11} \\ \mathbf{D}_{21} \end{bmatrix} \tag{2.38}$$

$$\mathbf{F}(\boldsymbol{\omega}(t)) = \begin{bmatrix} \mathbf{I}_{n_z} - \mathbf{D}_{11}\boldsymbol{\Delta} & -\mathbf{D}_{12}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \\ -\mathbf{D}_{21}\boldsymbol{\Delta} & \mathbf{I}_{n_\omega} - \mathbf{D}_{22}\boldsymbol{\Theta}_\omega(\boldsymbol{\omega}(t)) \end{bmatrix}^{-1} \tag{2.39}$$

It must be pointed out that, in the implementation, the Jacobian $\boldsymbol{\Theta}_\omega$ is simbolically computed directly from the definition of the function $\boldsymbol{\Theta}$.

Since the second stage requires the value of $\boldsymbol{\omega}(t)$ computed in the first stage, the two systems must be run in cascade, as in Fig. 2.2, that shows the complete procedures.

It is also important to underline the fact that the second stage must be executed as many times as the number of parameters in the $\boldsymbol{\delta}$ vector. This is a very critical part from the computational cost point of view, since the second stage is repeated many times during the whole estimation procedure.

A solution to increase the computational efficiency was implemented in the Toolbox. Given the linear ODE (2.33) it is possible to rewrite it as

$$\dot{\mathbf{x}}_i'(t) = \boldsymbol{\Gamma}_2(\omega) \begin{bmatrix} \mathbf{x}_i'(t) \\ \boldsymbol{\Delta}_{\delta_i} \mathbf{z}(t) \end{bmatrix} \tag{2.40}$$

with

$$\boldsymbol{\Gamma}_2(\boldsymbol{\omega}) = \left[\begin{bmatrix} \mathbf{A} & \mathbf{B}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{B}_1\boldsymbol{\Delta} & \mathbf{B}_2 \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \end{bmatrix} \boldsymbol{\Gamma}_1(\omega) \right] \tag{2.41}$$

$$\boldsymbol{\Gamma}_1(\boldsymbol{\omega}) = \begin{bmatrix} \mathbf{I}_{n_z} - \mathbf{D}_{11}\boldsymbol{\Delta} & -\mathbf{D}_{12} \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \\ -\mathbf{D}_{21}\boldsymbol{\Delta} & \mathbf{I}_{n_\omega} - \mathbf{D}_{22} \left.\frac{\partial \boldsymbol{\Theta}(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}}\right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{C}_1 & \mathbf{D}_{11} \\ \mathbf{C}_2 & \mathbf{D}_{21} \end{bmatrix} \tag{2.42}$$

Analyzing the use of $\boldsymbol{\Gamma}_1(\boldsymbol{\omega})$ and $\boldsymbol{\Gamma}_2(\boldsymbol{\omega})$, it can be noticed that their components can be evaluated and saved in a `CommonTerms` vector, during the first stage of the LFT parameter estimator at all the time instants of the simulation. These terms are then interpolated and evaluated in every time instant of the second stage in order to greatly reduce the overall computational cost of the parameter identification procedure.

It is also worth mentioning that, in order to deal with parameter estimation, it is essential to rewrite model (2.8−2.13) by introducing normalized unknown parameters $\bar{\delta}_i$, varying between ± 1 as parameter δ_i varies between a maximum $\delta_i^{\max}$ and a minimum $\delta_i^{\min}$ with

$$\delta_i = \frac{(\delta_i^{\max} + \delta_i^{\min})}{2} + \frac{\bar{\delta}_i(\delta_i^{\max} - \delta_i^{\min})}{2} \tag{2.43}$$

Deriving the LFT formulation could be non-trivial if carried out manually, to this aim, a symbolic computing approach, partially developed in [3] with reference to linear models, has been first fully implemented for application to the general non-linear case in [1]. Moreover, there is no unique solution.

2.1 Example

As an example consider this simple, purely academic, nonlinear model

$$\dot{x}_1 = 5x_1\delta_1 + 3\frac{x_2}{1+\delta_2} \tag{2.44}$$

$$\dot{x}_2 = u \tag{2.45}$$

$$y = x_1 + x_2 \tag{2.46}$$

where δ_1 and δ_2 are the uncertain parameters. The relevant LFT formulation can be derived according to the following steps:

1. Collect the uncertain parameters in the vector $\boldsymbol{\delta}$:

$$\boldsymbol{\delta} = \begin{bmatrix} \delta_1 & \delta_2 \end{bmatrix}^T \tag{2.47}$$

2. Each parameter δ_i has to multiply another varying quantity, otherwise, equations are modified to get the result, in our example:

$$\frac{x_2}{1+\delta_2} \rightarrow \frac{x_2^2}{x_2 + x_2\delta_2} \tag{2.48}$$

3. Define:

$$w_j = \delta_i z_j \quad , \quad \begin{cases} i = 1, \ldots, q, \\ j = r_{i-1}+1, \ldots, r_{i-1}+r_i \end{cases} \tag{2.49}$$

in our example:

$$w_1 = \delta_1 z_1 \tag{2.50}$$
$$w_2 = \delta_2 z_2 \tag{2.51}$$

4. Compute

$$\mathbf{z} = \mathbf{C}_1\mathbf{x} + \mathbf{D}_{11}\mathbf{w} + \mathbf{D}_{12}\boldsymbol{\zeta} + \mathbf{D}_{13}\mathbf{u} \tag{2.52}$$

keeping linear dependencies of $\mathbf{z}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:

$$z_1 = x_1 \tag{2.53}$$
$$z_2 = x_2 \tag{2.54}$$

5. Compute

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}_1\mathbf{w} + \mathbf{B}_2\boldsymbol{\zeta} + \mathbf{B}_3\mathbf{u} \tag{2.55}$$
$$\mathbf{y} = \mathbf{C}_3\mathbf{x} + \mathbf{D}_{31}\mathbf{w} + \mathbf{D}_{32}\boldsymbol{\zeta} + \mathbf{D}_{33}\mathbf{u} \tag{2.56}$$

keeping linear dependencies of $\dot{\mathbf{x}}$ and $\mathbf{y}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:

$$\dot{x}_1 = 5w_1 + 3\zeta \tag{2.57}$$
$$\dot{x}_2 = u \tag{2.58}$$
$$y = x_1 + x_2 \tag{2.59}$$

6. Define

$$\boldsymbol{\zeta} = \boldsymbol{\Theta}(\boldsymbol{\omega}) \tag{2.60}$$

in our example

$$\zeta = \frac{\omega_1^2}{\omega_1 + \omega_2} \tag{2.61}$$

7. Compute

$$\boldsymbol{\omega} = \mathbf{C}_2\mathbf{x} + \mathbf{D}_{21}\mathbf{w} + \mathbf{D}_{22}\boldsymbol{\zeta} + \mathbf{D}_{23}\mathbf{u} \tag{2.62}$$

keeping linear dependencies of $\boldsymbol{\omega}$ from $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{u}$ while introducing nonlinear terms in $\boldsymbol{\zeta}$, in our example:

$$\omega_1 = x_2 \tag{2.63}$$
$$\omega_2 = w_2 \tag{2.64}$$

The final LFT formulation is thus given by:

$$\dot{\mathbf{x}} = \begin{bmatrix} 5w_1 + 3\zeta & u \end{bmatrix}^T \tag{2.65}$$
$$\mathbf{z} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \tag{2.66}$$
$$\boldsymbol{\omega} = \begin{bmatrix} x_2 & w_2 \end{bmatrix}^T \tag{2.67}$$
$$y = x_1 + x_2 \tag{2.68}$$
$$\mathbf{w} = \begin{bmatrix} \delta_1 z_1 & \delta_2 z_2 \end{bmatrix}^T \tag{2.69}$$
$$\zeta = \frac{\omega_1^2}{\omega_1 + \omega_2} \tag{2.70}$$

Chapter 3

Dynamic model of the vehicle

The identification algorithm discussed in Chapter 4 is based on the so-called *white box identification*, meaning that it relies on a mathematical model derived from the laws of physics. The single-track model is obtained from basic force balances and, under mild assumptions, it is simple enough to be implemented in a computationally efficient algorithm.

In this chapter, a formulation from literature of the single-track model is shown; then, the focus is moved on the description of tire-road interaction, i.e. a mathematical model for the friction forces exchanged between tires and road. The Fiala tire model, also known as "brush" model, has been chosen for this purpose: it is derived from the laws of physics and it has a simpler expression than the best known Pacejka model, leading to a less complicated dynamic model and to a more efficient identification algorithm.

3.1 The single-track model

The main idea of the single-track model is to consider the wheels as if they were lumped together on the same front or rear axle at its centerline. The main quantities involved in the dynamic model are depicted in Figure 3.1., where

- u and v are the longitudinal (parallel to the vehicle main direction) and lateral (orthogonal to the latter) components of the speed of the Center of Gravity.
- The side-slip angle β is defined as:

$$\beta = \arctan\left(\frac{v}{u}\right)$$

- ψ is the yaw angle, between the vehicle main direction and an inertial reference frame;

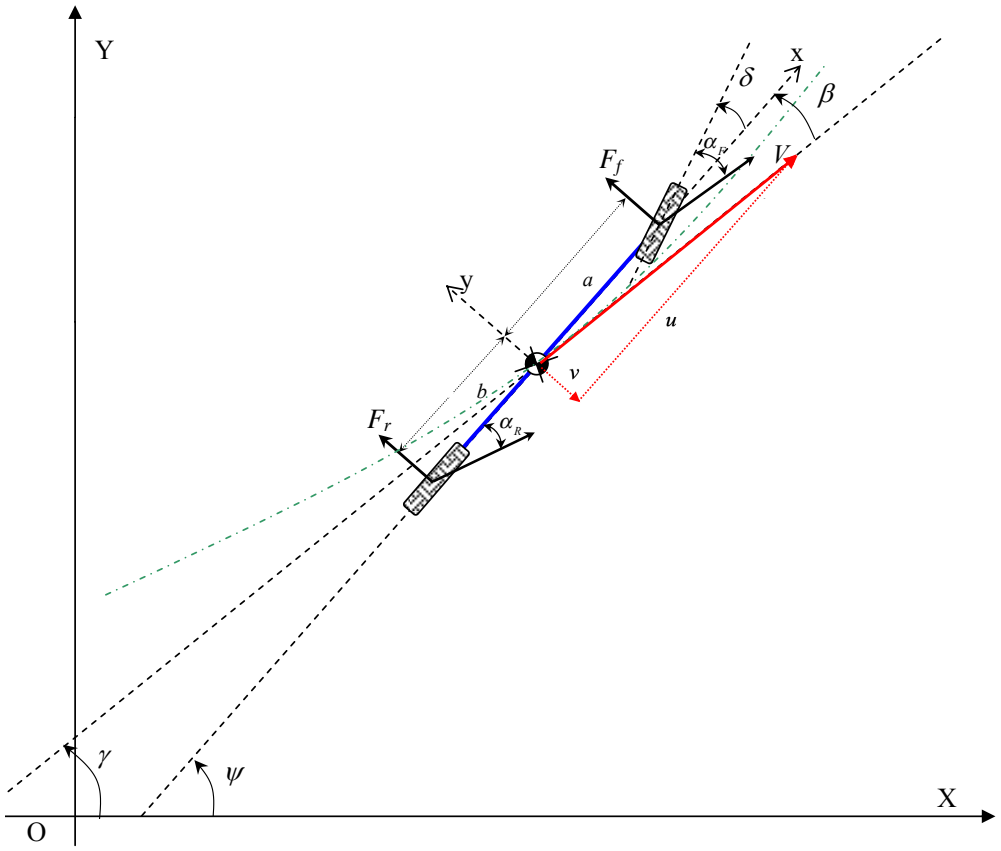

Figure 3.1: Single-track vehicle model (the figure show the quantities used to derive the motion model).

- r is the yaw rate;
- a and b are respectively the distances of the front axle and the rear axle from the CoG;
- δ is the steering angle. A steering gain G must be considered if δ is retrieved from the steering wheel, in this case it may differ from the actual steering angle.
- A wheel's tire-slip angle is defined as the angle made up of the wheel's speed vector, and its projection along the wheel's main direction. Front and rear tire slip angles can be computed as follows:

$$\alpha_f = -\beta - a\frac{r}{u} + G\delta \qquad \alpha_r = -\beta + b\frac{r}{u}$$

  playing an important role when defining the tire model.

To further simplify the model, the following assumption are enforced:

- ground slope, longitudinal load transfer, and pitching and rolling motions are neglected;

- a linear tire model is considered, i.e.,

$$F_f = C_f \alpha_f \qquad F_r = C_r \alpha_r$$

  where F_f, F_r are the front and rear lateral forces and C_f, C_r are the front and rear cornering stiffness;

- sideslip and steering angle are small enough to introduce the approximations $\sin(x) \approx x$ and $\cos(x) \approx 1$;
- the vehicle is rear-wheel drive and braking forces are neglected;
- the vehicle velocity is slowly varying, i.e., the Newton equation related to the longitudinal motion is considered at steady state.

Under the previous assumptions, single-track model's equations can be expressed through the following dynamical system:

$$\begin{aligned}
\dot{v} &= \frac{1}{m}(F_f + F_r) - ur \\
\dot{r} &= \frac{1}{I_z}(aF_f - bF_r) \\
\dot{\psi} &= r \\
F_f &= C_f \alpha_f \\
F_r &= C_r \alpha_r \\
\alpha_f &= -\beta - a\frac{r}{u} + G\delta \\
\alpha_r &= -\beta + b\frac{r}{u} \\
\beta &= \arctan\left(\frac{v}{u}\right)
\end{aligned} \tag{3.1}$$

where m and I_z are the vehicle mass and moment of inertia.
The time evolution of the kinematic states is given by:

$$\begin{aligned}
\dot{x} &= u\cos\psi - v\sin\psi \\
\dot{y} &= u\sin\psi + v\cos\psi
\end{aligned} \tag{3.2}$$

where x, y are the Cartesian coordinates of the vehicle center of mass with respect to an inertial reference frame.

3.2 Fiala model (brush model) for tire-road interaction

Experimental evidence shows that the linear tire model is not accurate for large values of the tire-slip angle. Therefore the single-track model in (3.1) may be incorrect, if the vehicle undergoes intense drifting.

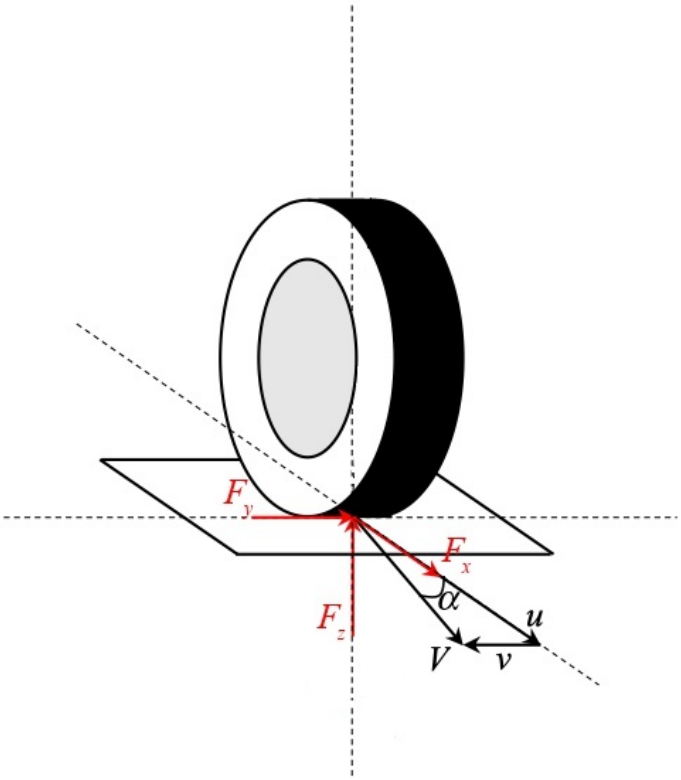

Figure 3.2: Tire scheme. The involved quantities in the tire model are shown.

Pacejka formulated a more accurate nonlinear tire model, which relates F_y, i.e. the lateral force exchanged between tire and road due to friction on the contact patch, and α, the tire-slip angle, as follows:

$$F_y = D \sin\left[C \arctan\left[B\alpha - E\left[B\alpha - \arctan\left[B\alpha\right]\right]\right]\right] \tag{3.3}$$

where B, C, D, E are suitably chosen coefficients.

Pacejka model is also referred to as "Magic Formula", since it has no physical background but it accurately fits experimental data.

On the other hand, the *Fiala* model is derived from the laws of physics: the main idea consists in considering the contact patch, the surface between tire and road, covered with brushes which bend when they touch the road (for this reason it is also known as *brush model*). The formula of the Fiala model is:

$$F_y = \begin{cases} Cz\left(1 - \frac{|z|}{z_{sl}} + \frac{z^2}{3z_{sl}^2}\right) & |z| < z_{sl} \\ \mu F_z \text{sign}(\alpha) & |z| \geq z_{sl} \end{cases}$$

where C is the tire cornering stiffness, α is the tire-slip angle, $z = tan(\alpha)$, μ is the dynamic friction coefficient between the materials of tire and road. z_{sl} is defined as

$$z_{sl} = \frac{3\mu F_z}{C} \tag{3.4}$$

By performing suitable substitutions in (3.3), the Fiala model formula becomes

$$F_y = C \begin{cases} z\left(1 - \frac{|z|}{z_{sl}} + \frac{z^2}{3z_{sl}^2}\right) & |z| < z_{sl} \\ \frac{z_{sl}}{3}\text{sign}(\alpha) & |z| \geq z_{sl} \end{cases} \tag{3.5}$$

If α maintains sufficiently small values, the approximation $z = tan(\alpha) \approx \alpha$ can be introduced.

The quantity z_{sl} is a bound on z delimiting the *Full Sliding* region. According to the Fiala model, when the tire-slip angle α reaches a sufficiently high value, the lateral force acting on the tire during cornering is no more dependent on α and it is capped at a fixed value: this condition is called Full Sliding (for the sake of brevity it will be addressed as FS hereon).
While the vehicle is not in the FS region, it can be noticed that F_y is a cubic function of α. Therefore it is much simpler than the Pacejka formula in (3.3), so it is more suitable for a computationally efficient identification algorithm.

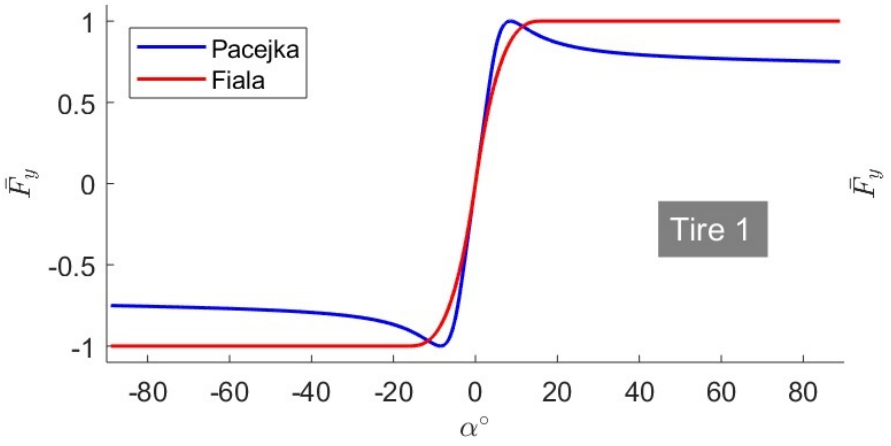

Figure 3.3: A comparison between Pacejka and Fiala models.

Figure 3.3 shows a graphical comparison between the two models. The most noticeable difference is the absence of the decreasing slope of the FS zone in the Fiala model. However, it will be shown in the following chapter that FS data are detrimental for the identification of the cornering stiffnesses and need to be discarded, therefore choosing the Pacejka formula to accurately model FS data would bring no benefit and would only increase the complexity of the algorithm.
Regarding the implementation of the Fiala model in MATLAB, it graphically resembles a saturation. However, `min()` and `max()` MATLAB functions are not compatible with the toolbox for LFT identification. Instead, the function `abs()`

is compatible and can be exploited to represent the model:

$$\begin{aligned} z &= \tan(\alpha) \\ y(z) &= z\left(1 - \frac{|z|}{z_{sl}} + \frac{z^2}{3z_{sl}^2}\right) \\ f(z) &= \frac{1}{2}\left(\left|y(z) + \frac{z_{sl}}{3}\right| - \left|y(z) - \frac{z_{sl}}{3}\right|\right) \\ F_y &= Cf(z) \end{aligned} \tag{3.6}$$

Chapter 4

Parameter identification in the LFT formulation of the Single Track Model

The identification algorithm is based on the single-track model. It has already been stated that the purpose of this work is to provide an estimate of the cornering stiffnesses; however, a few adjustments allow to simplify the identification problem and also estimate the vehicle moment of inertia around its z-axis, I_z.

4.1 Setup of the identification problem

4.1.1 Definition of the accessible inputs and outputs of the system

Consider the equations of the single-track model (3.1):

$$
\begin{aligned}
\dot{v} &= \frac{1}{m}(F_f + F_r) - ur \\
\dot{r} &= \frac{1}{I_z}(aF_f - bF_r) \\
\dot{\psi} &= r \\
F_f &= C_f \alpha_f \\
F_r &= C_r \alpha_r \\
\alpha_f &= -\beta - a\frac{r}{u} + G\delta \\
\alpha_r &= -\beta + b\frac{r}{u} \\
\beta &= \arctan\left(\frac{v}{u}\right)
\end{aligned}
$$

In every implementation of the identification algorithm in this work, the following 5 variables are defined as *accessible*, i.e. they are known functions of time and they provide data to perform the identification.

- u, the longitudinal velocity;
- δ, the steering angle;
- r, the yaw rate;
- a_y, the lateral acceleration of vehicle CoG. a_y is given by:

  $$a_y = \dot{v} + ur.$$

  Comparing a_y with the first equation of (3.1), it can be computed as:

  $$a_y = \frac{1}{m}(F_f + F_r). \tag{4.1}$$

- β, the sideslip angle.

4.1.2 Identification goal

The final purpose of the identification procedure is to estimate C_f and C_r. However, a suitable choice of the coefficients to identify allows also to estimate I_z. Model (3.1) can be rewritten as:

$$\begin{aligned}
\dot{v} &= \frac{C_f}{m}\left(-\beta - a\frac{r}{u} + G\delta\right) + \frac{C_r}{m}\left(-\beta + b\frac{r}{u}\right) - ur \\
\dot{r} &= a\frac{C_f}{I_z}\left(-\beta - a\frac{r}{u} + G\delta\right) - b\frac{C_r}{I_z}\left(-\beta + b\frac{r}{u}\right) \\
\dot{\psi} &= r
\end{aligned}$$

Assuming that all other coefficients are known, the goal is to identify the *three ratios*:

$$\delta_1 = \frac{C_f}{m} \qquad \delta_2 = \frac{C_r}{m} \qquad \delta_3 = \frac{C_f}{I_z}$$

Once the three ratios are estimated, the computation of C_f, C_r and I_z is straightforward.

$$C_f = m\delta_1 \qquad C_r = m\delta_2 \qquad I_z = m\frac{\delta_1}{\delta_3} \tag{4.2}$$

4.2 Identification from simulated data: linear model

From this section onwards, results of the implemented LFT identification algorithms are shown and discussed. They are all based on the LFT identification toolbox, but they differ from each other for the choice of the tire model: linear and Fiala models will be considered.

The linear model lends itself to a simple explanation of the implemented code, which is similar for the algorithms in the following sections. Moreover, it provides satisfactory results for the identification when tested on experimental data, if the tire slip angles are small.

Listing 4.1: Definition of the known parameters and loading of the accessible variables

```
% Data
mass = 2; % [kg] vehicle mass
a = 0.15; % [m] CoG-front axle distance
b = 0.11; % [m] CoG-rear axle distance
Gsteer = 1;

d1lim = [0 100];
d2lim = [0 100];
d3lim = [0 1000];

load('data')
```

The first code section is shown in Listing 4.1. The known parameters are defined as well as the boundaries on the unknown parameters `d1lim`, `d2lim`, `d3lim`. These values also determine the starting values of the parameters, used in the first iteration of the identification algorithm. They are computed as the average of each parameter's boundaries.
The accessible variables are loaded by means of `load('data')`. The initial values of the state variables of the LFT form are also retrieved in this way: these are required to numerically integrate the system using the function `lftSolver`.

Listing 4.2: Definition of lftfun

```
%% Load lft function
lftfun = singleTrackLftLinear(a, b, Gsteer, d1lim, d2lim
   , d3lim);
lftfun.DeltaVal = diag([0 0 0]);

M = max(output);
m = min(output);
for i=1:length(M)
nominal(i)=(M(i)-m(i));
```

```
end
lftfun.ISO.Nominal = nominal;
```

In Listing 4.2 the struct `lftfun` is defined, containing the implementation of the LFT single-track model. A more thorough explanation of the involved variables is given in Appendix A; a detailed formulation of the single-track LFT form is in Appendix B.

Listing 4.3: LFT toolbox settings

```
%% LFT toolbox settings
LFTsolverOptions = lftSet('SensAlgorithm', 'ode15s',...
'SolutionTimeSpan', t,...
'SolutionInterpMethod', 'spline',...
'OversamplingMethod', 'spline',...
'RelTol', 1e-3,...
'AbsTol', 1e-6...
);
LFToptimOptions = lftOptSet('StartOptimSample', 1,...
'Display', 'iter',...
'StepTolerance', 1e-6,...
'MaxIter', 60,...
'TolFun', 1e-9...
);
```

The settings for the LFT solver are saved in the struct `LFTsolverOptions`. The chosen numerical method is `ode15s`, a stiff method which allows to use a variable step. Each step should satisfy a prescribed tolerance on the local error, given by `RelTol` and `AbsTol`: these are suitably chosen to attain satisfactory results in the identification, and terminate the algorithm in a reasonable time. Indeed, the identification procedure integrates the LFT system at each iteration along the time span associated to `SolutionTimeSpan`, so this operation should not be too long.
The settings for the LFT identification algorithm are stored in `LFToptimOptions`. These include the criteria to terminate the algortithm, `StepTolerance`, `MaxIter` and `TolFun`.
Further options are available for `lftSet` and `lftOptSet` but they are not the main focus of this treating. The interested reader can find them in Appendix A.

Listing 4.4: Setting of AlgebraicStartingValues

```
% Setting of AlgebraicStartingValues
algebraicStartingValues = zeros(9,1);
algebraicStartingValues(7) = 1e-6;
algebraicStartingValues(6) = u(1);

InitialConditions = struct('StateInitialConditions',
   initialConditions,...
```

```
'AlgebraicStartingValues', algebraicStartingValues...
);
```

The passage in Listing 4.4 allows a proper initialization of the algorithm. `'AlgebraicStartingValues'` is a vector collecting arbitrary initial values for the z and ω variables of the LFT system.

$$ASV = \begin{bmatrix} z_1(0) & z_2(0) & z_3(0) & \omega_1(0) & \omega_2(0) & \omega_3(0) & \omega_4(0) & \omega_5(0) & \omega_6(0) \end{bmatrix}^T$$

It is not important how these values are set, since `lftSolver` automatically computes a set of consistent initial conditions when it is called. However, some of these variables may appear at the denominator in the LFT system: if a division by zero is detected at the initial time instant, the algorithm returns error and terminates. By setting the corresponding elements of `AlgebraicStartingValues` to a non-zero value, this issue is removed.
In the case of `singleTrackLinear`, the interested variables are u and α_f: they appear at the denominator in some elements of $\theta(\omega)$ (see B.1 and B.2 for details).

Listing 4.5: Identification algorithm

```
Input = struct('Type', 'interpolated', 'Samples', input,
    'Time', t);

% Identification algorithm
[DELTA_opt, fval, J, grad, H, CN, history] = lftOptDelta
   (lftfun, Input, InitialConditions, LFTsolverOptions,
   output, LFToptimOptions);
[delta0, deltaOpt] = norm2abs(lftfun, lftfun.DeltaVal,
   DELTA_opt);

H = H(:,:,end), CN = CN(end)

%% Results
Cf = mass*deltaOpt(1)
Cr = mass*deltaOpt(2)
Iz = mass*deltaOpt(1)/deltaOpt(3)
```

In the last section of code, the identification procedure is run. First of all, the matrix collecting the input variables of the LFT system, u and δ, and the time vector are stored in the struct `'Input'`, which is fed to the function performing the identification, `lftOptDelta`. The latter is iteratively called by `norm2abs` which, at each iteration, chooses values of the unknown parameters δ_1, δ_2 and δ_3 to minimize the cost function J. When it terminates, the function returns `delta0`, i.e. the values of the unknown parameters at the start of `norm2abs`, and `deltaOpt`, the optimal values.
The function `lftOptDelta` returns, at each iteration, the Hessian and its Condition Number. Their values at the final iteration are saved in `H` and `CN`: they provide an interesting index to evaluate the quality of the estimate.

Finally, C_f, C_r and I_z are computed, accordingly to (4.2).

4.2.1 Simulation setup

The identification algorithm is run on simulated data, to show that it gives correct results in an ideal setup. The known parameters are set to similar values of the experimental setup of Section 4.4, so that comparable results can be attained.

Vehicle mass	m	2	kg
Distance of front axle from C.O.G.	a	0.15	m
Distance of rear axle from C.O.G.	b	0.11	m

The unknown parameters are set to arbitrary values; these are the parameters the algorithm longs to estimate.

Front wheels' cornering stiffness	C_f	3	N/rad
Rear wheels' cornering stiffness	C_r	4	N/rad
Vehicle moment of inertia	I_z	0.03	kgm^2

The input variables u and δ are shown in Figure 4.1. $u(t)$ is constant and equal to 1m/s, $\delta(t)$ is a sinusoid of amplitude 0.2rad and frequency 1rad/s.

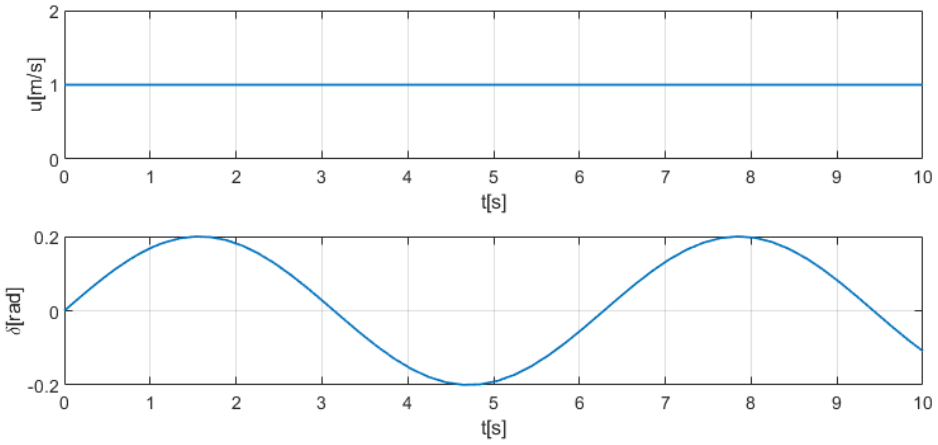

Figure 4.1: Simulated input variables.

The simulated system has been implemented in Simulink and it is integrated with `ode15s`, the same method used by `lftSolver`. The results of the identification procedure are summarized in the following table.

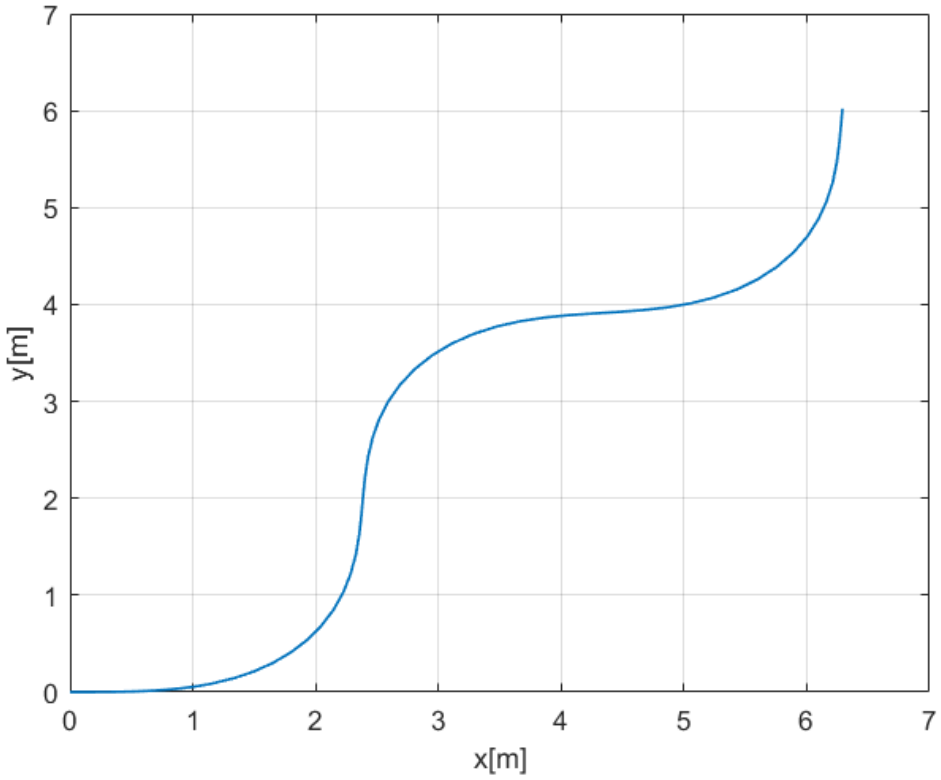

Figure 4.2: Simulated trajectory of the vehicle.

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	3	3.0176	+0.59%
Rear Tire Cornering Stiffness	C_r	4	4.0058	+0.14%
Moment of Inertia	I_z	0.03	0.0298	−0.67%

4.3 Identification from simulated data: Fiala model

A possible step to improve the identification algorithm is introducing the Fiala tire model in the LFT single-track model. Thus, the tire model should be more accurate for higher values of the tire-slip angles, w.r.t. the linear one.
In Appendix B.2 a LFT formulation embedding Fiala model is reported. Besides a different choice of the nonlinear variables, the main change is given by the introduction of parameter z_{sl} which, recalling (3.4), is equal to:

$$z_{sl} = \frac{3\mu F_z}{C}$$

μ and F_z can be considered fixed if the vehicle is driven at reasonably low speed; however the cornering stiffness is unknown, so a priori information on z_{sl} should

not be available. In the simulated setup, z_{sl} is considered constant and equal to 1 for the sake of simplicity and to remark that the algorithm works properly. This assumption is dropped when treating experimental data: a method to retrieve z_{sl} from data is suggested in Subsection 4.3.3.

The implemented code slightly differs from the one of the previous section. In the following, only the differing lines are discussed.

Listing 4.6: Definition of known parameters - Fiala

```
% Data
mass = 2; % [kg] vehicle mass
a = 0.15; % [m] CoG-front axle distance
b = 0.11; % [m] CoG-rear axle distance
Gsteer = 1;
zsl = 1;

d1lim = [0 100];
d2lim = [0 100];
d3lim = [0 1000];
```

Same choice of the known parameters and of the boundaries on the unknown ones is made, except for the introduction of z_{sl}.

Listing 4.7: Definition of lftfun - Fiala

```
%% Load lft function
lftfun = singleTrackLftFiala(a, b, Gsteer, zsl, d1lim,
   d2lim, d3lim);
```

The struct `lftfun` should be assigned by calling the proper function implementing the modified single-track model in its LFT form, named `singleTrackLftFiala`. The settings for the LFT solver and identification are the same as for the linear model, hence they are omitted.

Listing 4.8: Setting of AlgebraicStartingValues - Fiala

```
% Setting of AlgebraicStartingValues
algebraicStartingValues = zeros(11,1);
algebraicStartingValues(6) = 1e-6;
algebraicStartingValues(10) = u(1);

InitialConditions = struct('StateInitialConditions',
   initialConditions,...
'AlgebraicStartingValues', algebraicStartingValues...
);
```

Since vector $\boldsymbol{\omega}$ has changed, `algebraicStartingValues` should be updated accordingly.
The remaining code lines are identical to Listing 4.5.

4.3.1 Simulation setup

As in Section 4.2, the identification algorithm is run on simulated data. The choice of known and unknown parameters for the simulation is unchanged.
Known parameters:

Vehicle mass	m	2	kg
Distance of front axle from C.O.G.	a	0.15	m
Distance of rear axle from C.O.G.	b	0.11	m

Unknown parameters:

Front wheels' cornering stiffness	C_f	3	N/rad
Rear wheels' cornering stiffness	C_r	4	N/rad
Vehicle moment of inertia	I_z	0.03	kgm^2

Two different sets of simulated data are generated, to be tested on the identification algorithm. The first set is computed with the same input variables of section 4.2, and is referred as *Non-drifting conditions*. The second set is obtained from higher longitudinal speed and steering angle, which cause the lateral forces acting on the tires to enter the *Full Sliding zone*. The latter is particularly interesting as it shows that, if any tire is in full sliding, the identification procedure does not estimate correctly the unknown parameters.

The simulation is run on Simulink, using the numerical method `ode15s`, the same of `lftSolver`, setting the same tolerance options on both.

4.3.2 Non-drifting conditions

When the system is fed by the inputs of Figure 4.3, the simulated output allow to estimate correctly the parameters, as shown in the table below.

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	3	3.041	+1.37%
Rear Tire Cornering Stiffness	C_r	4	3.992	−0.20%
Moment of Inertia	I_z	0.03	0.0301	+0.33%

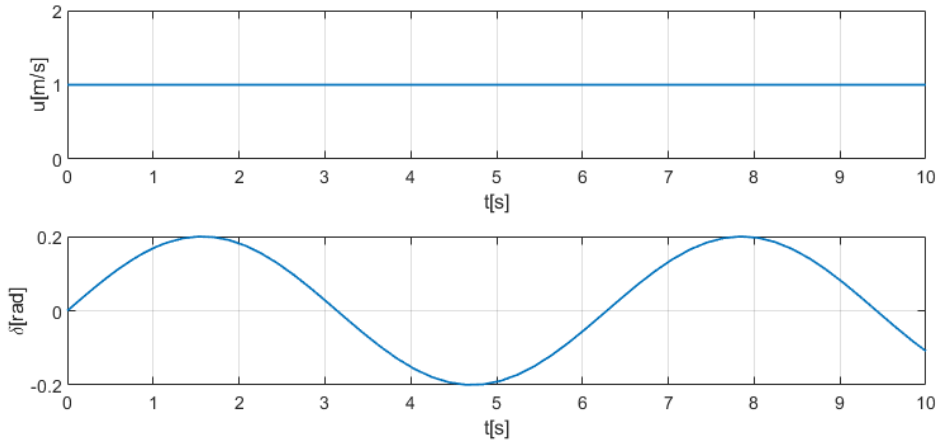

Figure 4.3: Simulated input variables.

4.3.3 Drifting conditions: the Full Sliding zone

In this section, the case in which a wheel of the single-track is in *Full Sliding* (FS) will be treated. In Chapter 3, it has been remarked that FS is a condition achieved in Fiala tire model when

$$\tan(\alpha) \geq z_{sl}$$

where α is the tire slip angle of the considered tire. When this situation occurs, the lateral force no longer depends on the tire slip angle but it is capped to a fixed value which, quoting equation 3.5, can be computed as

$$F_{FS} = C\frac{z_{sl}}{3}$$

where C is the wheel's cornering stiffness.
In other words, the involved tire is *drifting*.

In our simulation environment, z_{sl} is set to 1, $C_f = 3$ and $C_r = 4$. Naming α_f and α_r the front and rear tire slip angles, F_f and F_r the front and rear lateral forces, FS can be recognized if the following conditions are satisfied:

$$\tan(\alpha_f) \geq 1 \qquad F_f = 1N \qquad \tan(\alpha_r) \geq 1 \qquad F_r = \frac{4}{3}N \approx 1.3N \tag{4.3}$$

Under the former two conditions, the front wheel is in FS. The latter two mean that the rear wheel is in FS. If all the four conditions are satisfied, both wheels are in FS.

To show that in the simulation of section 4.3.2 FS is not occurring on neither wheel, in Figures 4.4 and 4.5 the tire slip angles and lateral forces of both wheels are plotted.

Comparing with the aforementioned (4.3) conditions, it is evident that FS is not attained.

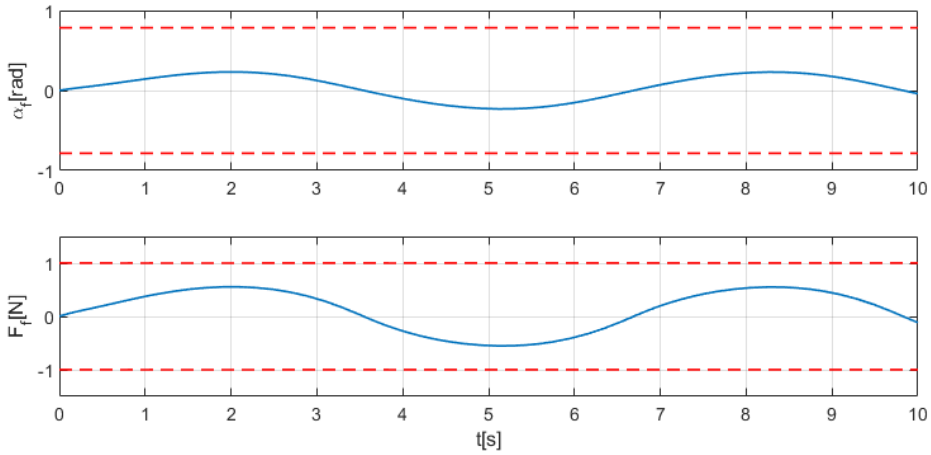

Figure 4.4: Front tire slip angle and lateral force. Full Sliding not occurring.

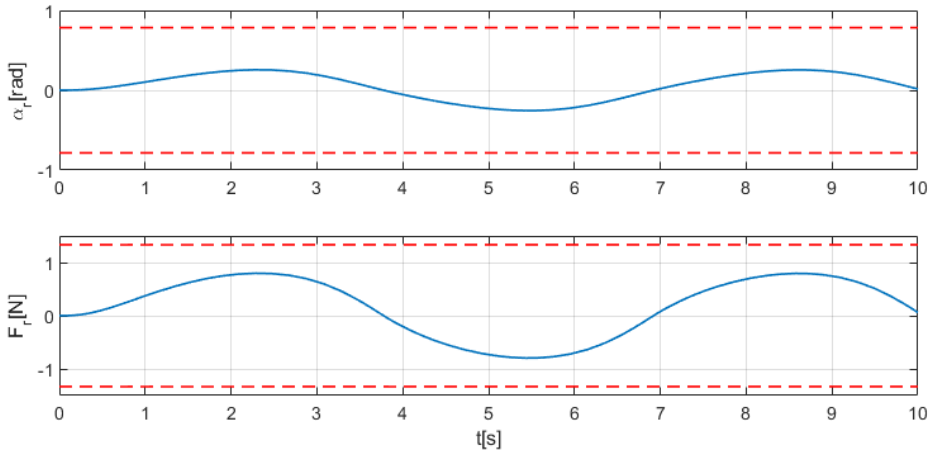

Figure 4.5: Rear tire slip angle and lateral force. Full Sliding not occurring.

To obtain greater values of α and F, it suffices to increase the value of either the simulated longitudinal speed u or the steering angle δ. For instance, the amplitude of δ can be increased to 0.4 rad/s, maintaining the same frequency and the same u as in the previous cases, to obtain the set of simulated data of Figure 4.6.

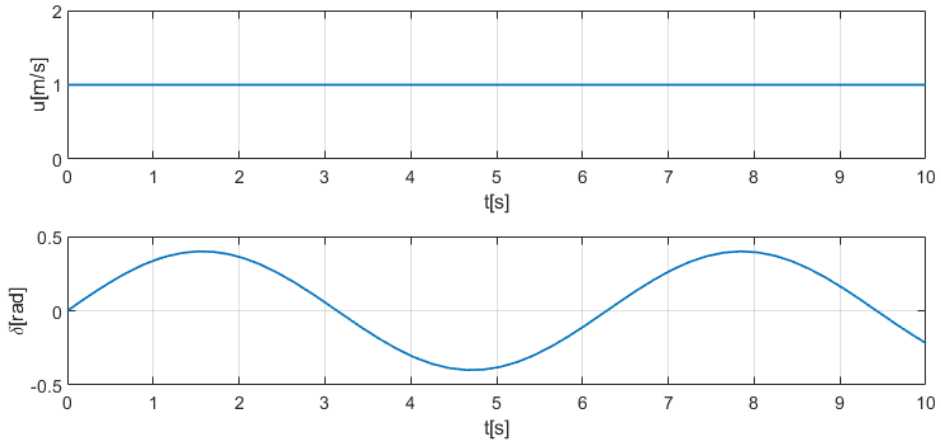

Figure 4.6: Simulated input variables. δ is increased w.r.t. Figure 4.3

Front and rear α and F change consequently, as shown in Figures 4.8 and 4.9. In this case, FS is achieved on the rear wheel: it is evident from the saturation of the rear lateral force, and it can be checked that the two latter conditions of 4.3 are satisfied.
The LFT identification algorithm is again run on the simulated data. However, the table below reports remarkable errors in the estimated parameters, which are not met when Full Sliding does not occur.

Parameter	δ	Real Value	Estimated Value
Front Tire Cornering Stiffness	C_f	3	35.5056
Rear Tire Cornering Stiffness	C_r	4	8.7704
Moment of Inertia	I_z	0.03	0.3132

These results are not only extremely incorrect, but they are also very sensitive to the initialization of the algorithm, meaning that they consistently change if a different starting value of the unknown parameters is set. In some cases, the algorithm does not even terminate.
More interestingly, the Condition Number of the Hessian at the final iteration is subject to a sheer increase w.r.t. the non-drifting situation. Indeed

$$CN = 8.0520 \cdot 10^4$$

is greater than $CN = 2.4946 \cdot 10^3$ obtained from the non-drifting data. This result proves that the problem is ill-posed and requires an alternative procedure to be tackled. The suggested solution is explained in Section 4.4.

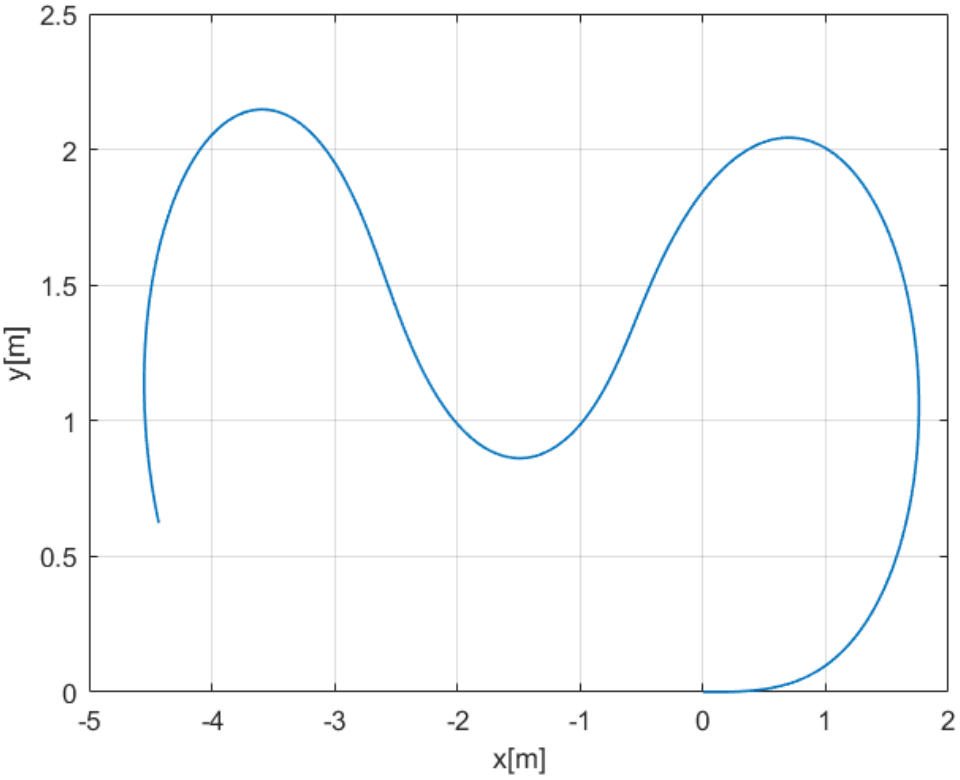

Figure 4.7: Simulated trajectory of the drifting vehicle.

4.4 Removal of Full Sliding data

FS data are a conceivable cause of the incorrect estimate obtained in the previous section. When any tire is in FS, the dynamics of single-track model's equations are no more dependent on the input variables. For this reason, these variables end up providing no information for the identification, whose accuracy is compromised.
A possible solution consists in removing detrimental FS data. The whole set of data could be divided in multiple segments of non-FS data, and the identification algorithm could be run on each one of them; the results of each procedure could be averaged to provide the final estimate.
However, a criterion is necessary to recognize FS data. This is not a trivial task, as it requires an estimate of the lateral forces without knowing the wheels' cornering stiffnesses.

The following sections explain the algorithm to discriminate FS data and the reasoning behind it. Interestingly, the algorithm does not require any knowledge on z_{sl}, in fact, the procedure also allows to estimate it.

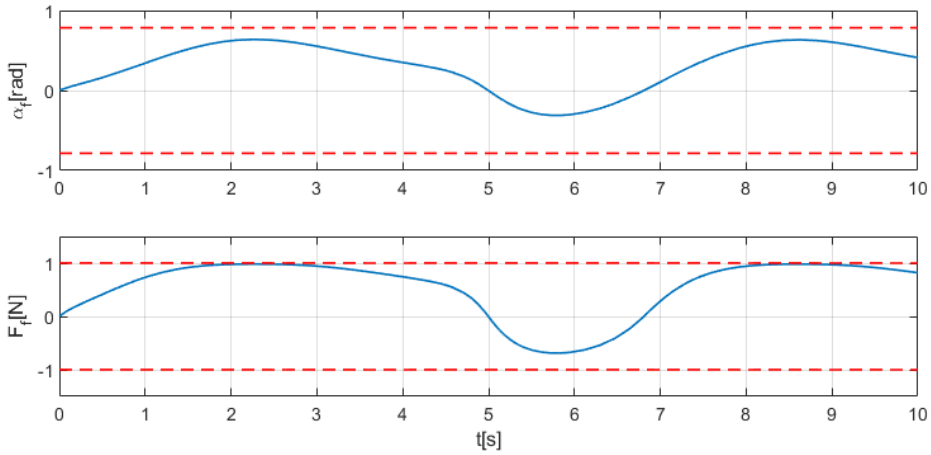

Figure 4.8: Front tire slip angle and lateral force. Full Sliding not occurring.

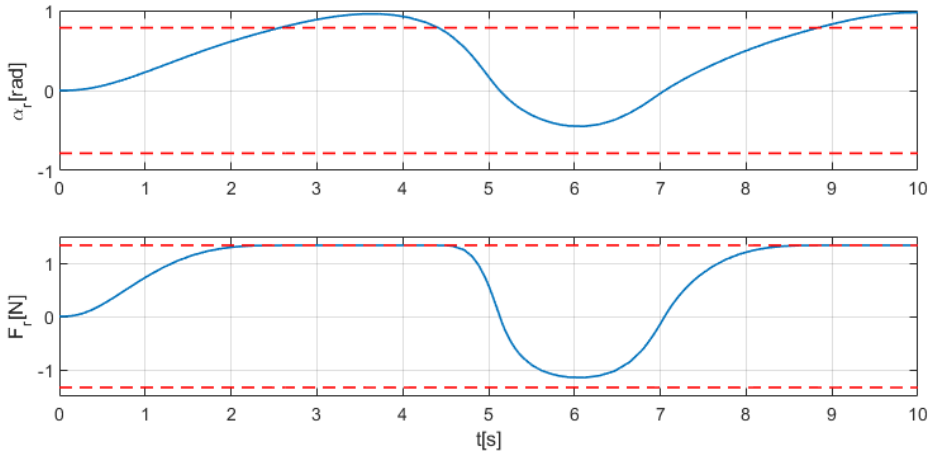

Figure 4.9: Rear tire slip angle and lateral force. Full Sliding is occurring.

4.4.1 Estimation of the lateral forces

Recalling the second equation of the single-track model (3.1)

$$\dot{r} = \frac{1}{I_z}(aF_f - bF_r)$$

and the definition of a_y from 4.1

$$a_y = \frac{1}{m}(F_f + F_r)$$

these equations can be exploited to compute the lateral forces:

$$\begin{aligned} F_f &= \frac{bma_y + I_z\dot{r}}{a+b} \\ F_r &= \frac{ama_y - I_z\dot{r}}{a+b} \end{aligned} \tag{4.4}$$

Thus, F_f and F_r can be estimated without making any assumption on the tire model and without any knowledge on the cornering stiffnesses. However, $\dot{r}$ and I_z are needed in the computation. The former can be obtained from the derivative of r (performed on sampled data throughout a suitable FIR filter); the latter is an unknown parameter and it has to be estimated.
The estimate can be obtained by means of the LFT identification of Section 4.2, based on the linear tire model. Since it is accurate only for small values of the tire slip angle, the procedure is run on data corresponding to low α_f and α_r. For instance, we can include only data for tire slip angles lower than a third of their maximum values.
To verify if this idea is feasible, the identification is run on the simulated data from Section 4.3.3, based on Fiala model, which at a certain point undergo FS. The identification returns the following results:

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	3	2.6485	-11.72%
Rear Tire Cornering Stiffness	C_r	4	3.7005	-7.49%
Moment of Inertia	I_z	0.03	0.0299	-0.33%

As we could expect, the estimated cornering stiffnesses are subject to a consistent error, due to the discrepancy between linear and Fiala models. Since lateral forces obtained from Fiala are always lower than the linear ones in absolute value, considering the linear model for the identification leads to smaller estimated cornering stiffnesses. However, this doesn't hold for I_z, which is retrieved from δ_1 and δ_3:

$$\delta_1 = \frac{C_f}{m} \qquad \delta_3 = \frac{C_f}{I_z} \qquad I_z = m\frac{\delta_1}{\delta_3}$$

C_f from δ_1 and δ_3 should be subject to the same error and it is inherently canceled in the computation of I_z. Therefore, the estimate of I_z is correct, which is the purpose of this identification.
The estimated lateral forces can now be computed, and they are shown in Figure 4.10. Comparing them with Figures 4.8 and 4.9, it can be noticed that they perfectly resemble each other, proving the correctness of the estimate.

The lateral forces are plotted as functions of their respective tire slip angles in Figures 4.11 and 4.12: since we are referring to simulated data, we expect to recognize the Fiala model. These plots are extremely useful to visualize the criterion to discriminate FS data, exposed in the next section.

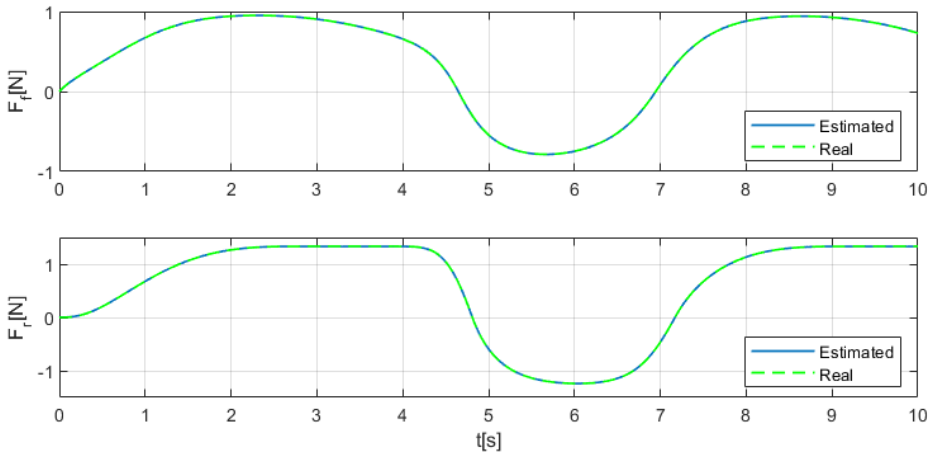

Figure 4.10: Estimated lateral forces.

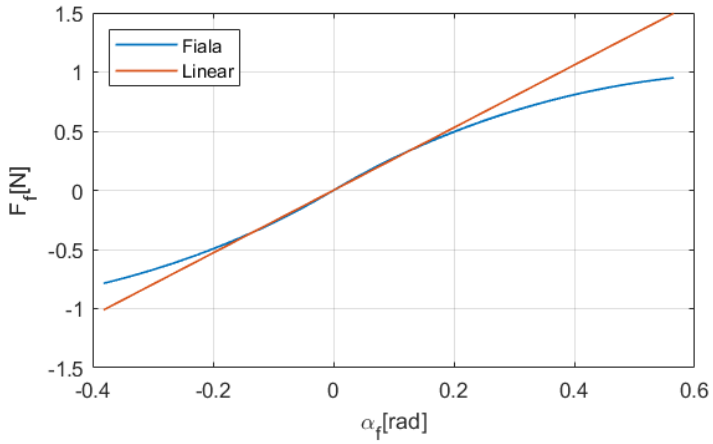

Figure 4.11: Estimated $F_f(\alpha_f)$.

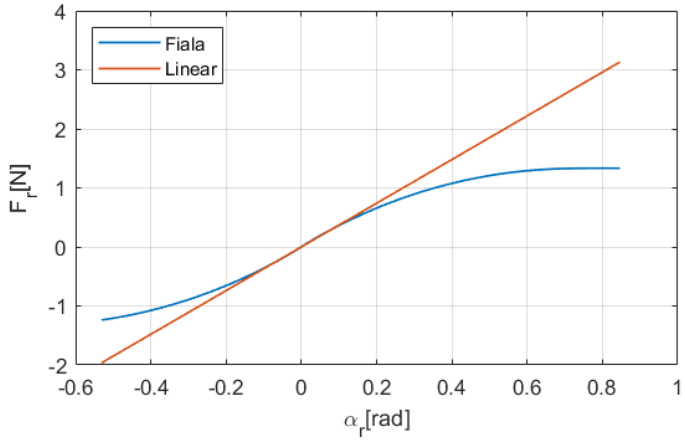

Figure 4.12: Estimated $F_r(\alpha_r)$.

4.4.2 Criterion to discriminate Full Sliding data

By definition, FS is occurring when $z \geq z_{sl}$, where $z = \tan(\alpha)$. The quantity z_{sl} is unknown, therefore this condition can't be verified. However, FS can also be recognized from the lateral force which, according to Fiala model, saturates to a specific value, $Cz_{sl}/3$. By assuming that the tire slip angle is small enough to introduce the approximation $z = \tan(\alpha) \approx \alpha$, the boundary on $F_{fia}(\alpha)$ can be set as

$$F_{sl} = F_{fia}(\alpha_{sl}) = C\frac{\alpha_{sl}}{3}$$

α_{sl} is unknown, but we can define F_{lin} as the lateral force computed from the linear model and notice that

$$\frac{F_{lin}(\alpha_{sl})}{3} = \frac{C\alpha_{sl}}{3} = F_{sl}$$

It can be proven that:

$$\begin{aligned} \left|\frac{F_{lin}(\alpha)}{3}\right| &< |F_{fia}(\alpha)| \qquad \text{iff } |\alpha| < \alpha_{sl} \\ \left|\frac{F_{lin}(\alpha)}{3}\right| &\geq |F_{fia}(\alpha)| \qquad \text{iff } |\alpha| \geq \alpha_{sl} \end{aligned} \tag{4.5}$$

The second inequality is obvious, since F_{fia} cannot be greater than F_{sl} while F_{lin} is equal to F_{sl} for $\alpha = \alpha_{sl}$ and greater for higher values of α. The proof of the first inequality is less trivial, but it can be easily understood by noticing that $F_{lin}(\alpha)$ and $F_{fia}(\alpha)$ have the same derivative over α in the origin, so $F_{lin}/3$ is lower than F_{fia}, for values of α up to α_{sl} (for which they are equal).
Equation (4.5) is a valid criterion to detect FS data, but F_{lin} requires the cornering stiffness C to be computed, which is unknown. The estimated C_f and C_r from the LFT identification of Section 4.4.1 are useful for this scope, although they are slightly lower than the actual ones. To compensate for this fact, equation (4.5) can be modified by multiplying a corrective coefficient k_{corr} to the left hand side of the inequalities (only the first one is rewritten for the sake of brevity).

$$\left|\frac{k_{corr}C}{3}\alpha\right| < |F_{fia}| \qquad \text{iff } |\alpha| < \alpha_{sl} \tag{4.6}$$

$k_{corr} \in [1; 3)$ can be suitably tuned to satisfy the user's requests. In particular:

- k_{corr} close to 1 is advisable if the user decides to account as many data as possible, with the risk of including FS data; this case is shown in Figure 4.13, with reference to the data from Figure 4.12. In this situation, Fiala model for identification should provide better results, as it is more accurate than the linear one for large tire slip angles, close to α_{sl}.
- k_{corr} close to 3 should be selected if the user desires to consider only data corresponding to low tire slip angles. In this way, the linear model may fit data rather well, and it can be used in a less complex identification procedure,

which would also terminate faster since it considers fewer data. However, the user should be careful of not excluding too many data or the quality of the estimate could be compromised. $k_{corr} = 3$ is a pathologic case, as the criterion (4.5) becomes

$$|C\alpha| < |F_{fia}|$$

which is not possible, since $|F_{lin}(\alpha)|$ is greater or equal to $|F_{fia}(\alpha)|$, $\forall\alpha$. If such k_{corr} is chosen, the criterion would exclude all the data.

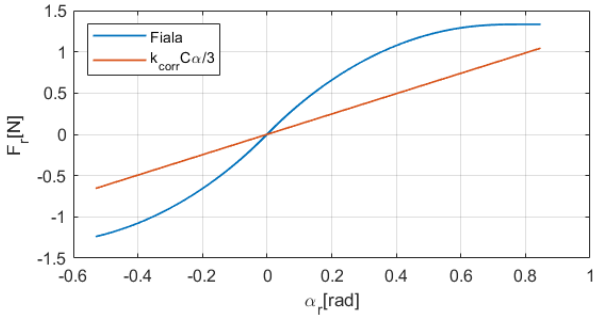

Figure 4.13: $k_{corr} = 1$. All data above the red line are taken, including the FS ones.

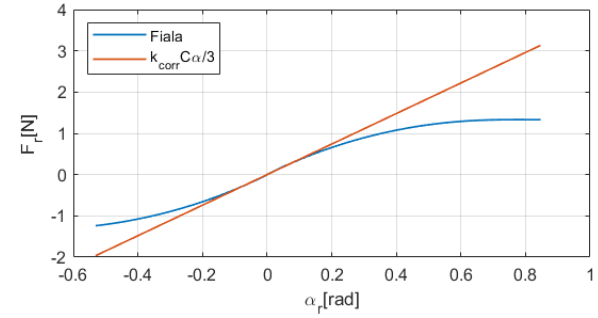

Figure 4.14: $k_{corr} = 3$. All data below the red line are excluded, almost no data are taken.

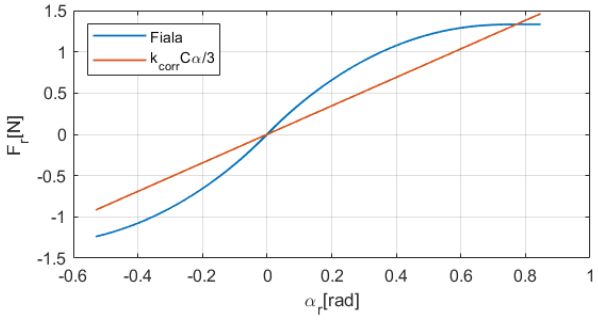

Figure 4.15: $k_{corr} = 1.4$ is a good compromise, allowing to discard only FS data.

4.4.3 Comments to implementation

In this section a brief explanation of how the explained ideas have been implemented is given, helped by portions of code.
The initial LFT identification to estimate I_z is the same explained in Section 4.2. However, it is not run on the whole set of data, but only on the first segment of data in which longitudinal speed is nonzero and the tire slip angles are lower than a third of their maximum value. This choice allows to avoid divisions by zero in the LFT model and selects only data that roughly fit the linear tire model. The results of the identification are saved in the variables `CfLFT`, `CrLFT`, `IzLFT`.
The variables $\dot{r}$, α_f and α_r are computed from the model and $\dot{r}$ is used to estimate the actual lateral forces as in (4.4). They are saved in the vectors `fialaF` and `fialaR`. The lateral forces given by the linear tire model are computed by multiplying `CfLFT` and `CrLFT` respectively by α_f and α_r. These are saved in the vectors `linearF` and `linearR`.
Listing 4.9 is the iteration that performs the distinction of FS data and non FS data. For each sample, the algorithm does the following operations:

- It checks if u or α_f are zero and, if they are, the data sample is discarded. Such values cause divisions by zero in the LFT model and they should be neglected. When a datum with nonzero u and α_f is detected, the corresponding time instant is saved in the vector `part`. This vector records the initial time instant of each data segment, which is necessary to select the correct initial conditions of each one of them, required by the LFT identification.
- If u and α_f are nonzero, criterion (4.5) is tested on the current data. If it is satisfied for both the front and rear wheels, non FS data are recognized and they are saved in `struct_y{j}`, `struct_u{j}` and `struct_time{j}`, which respectively record the output, the input, and the current time instant. The variable `j` is the index of the current data segment.
- If the (4.5) is not satisfied, a FS datum is detected. The highest value among $|\alpha_f|$, $|\alpha_r|$ and the previously stored α_{sl} is saved and assigned to the variable α_{sl}.
  The variable α_{sl} is set to zero at the start of the algorithm, so that it can be replaced as soon as FS is recognized. If it is not, the maximum value among $|\alpha_f|$ and $|\alpha_r|$ is assigned to α_{sl} (this case means that FS never occurs and all the partitioning becomes pointless).

Listing 4.9: Data partitioning algorithm

```
% Partitioning
i = 1;
j = 1;
part(1) = 1;
eps = 1e-6;
alphasl = 0;
while i < n
    if (abs(alphaf(i))<=eps || abs(u(i))<=eps) % neglect
       data which cause divisions by zero in the model
    j = j + 1;
    while (abs(alphaf(i))<=eps || abs(u(i))<=eps)
       i = i + 1;
    end
    part(j) = i;
    else % no division by zero, the algo can compare
       linear and Fiala to detect FS
       if (abs(corr*linearF(i)/3) <= abs(fialaF(i)) &&
          abs(corr*linearR(i)/3) <= abs(fialaR(i))) % low
           slip, data are stored
           struct_y{j}(i - part(j) + 1,:) = output(i,:);
           struct_u{j}(i - part(j) + 1,:) = input(i,:);
           struct_time{j}(i - part(j) + 1,:) = t(i,:);
           i = i + 1;
       else   % full sliding, i++ until both tires go
          back in low slip mode and data are not stored
           alphasl = max([abs(alphaf(i)) abs(alphar(i))
              alphasl])
           j = j + 1;
           while ((abs(corr*linearF(i)/3) > abs(fialaF(i))
               || abs(corr*linearR(i)/3) > abs(fialaR(i)))
               && i < n)
              i = i + 1;
           end
       part(j) = i;   % the start of the new segment is
          stored. It's required to compute the initial
          conditions
       end
    end
end

zsl = tan(alphasl);
```

Once the segments and the vector `part` are stored, a vector of initial conditions is defined for each segment and it is assigned the correct values.

Listing 4.10: Assignment of the initial conditions.

```
% Storing the initial conditions of each partition
for K = 2:j
initialConditions{K} = [v(part(K)) r(part(K))]';
end
```

The lateral speed of the vehicle C.o.G. `v` can be easily computed as $u\tan(\beta)$. The LFT identification algorithm of Sections 4.2 or 4.3 is run on each data segment. The choice of the algorithm depends on which tire model is desired. The results of each identification are saved in the struct `delta_Opt`. Also the cost function's value `J` at the end of each identification is saved, and proves to be useful to compute the average of all the results.

Finally, the results are averaged to obtained the final estimate, which is computed as a weighted mean of the results stored in `delta_Opt`. The weight can be chosen as:

- the number of data of each segment, assuming that an identification is more accurate, if it is run on more data. In that case, the vector $\boldsymbol{\delta}$ of estimated parameters is computed as:

$$\boldsymbol{\delta} = \frac{\boldsymbol{\delta_1} l_1 + ... + \boldsymbol{\delta_n} l_n}{l_1 + ... + l_n}$$

  where $\boldsymbol{\delta_i}$ is the i-th result stored in `delta_Opt`, l_i is the length of the i-th segment.

- the value of `J` associated to each segment, assuming that the more accurate an identification, the lower J. In that case, $\boldsymbol{\delta}$ is computed as:

$$\boldsymbol{\delta} = \frac{\boldsymbol{\delta_1}/J_1 + ... + \boldsymbol{\delta_n}/J_n}{1/J_1 + ... + 1/J_n}$$

  where J_i is the value of the cost function at the end of the i-th identification.

- A mixed strategy can be envisaged, computing $\boldsymbol{\delta}$ as:

$$\boldsymbol{\delta} = \frac{\boldsymbol{\delta_1} l_1/J_1 + ... + \boldsymbol{\delta_n} l_n/J_n}{l_1/J_1 + ... + l_n/J_n} \tag{4.7}$$

4.4.4 Simulated results

The algorithm embodying data partitioning and LFT identification is tested on the data from Section 4.3.3, in which the system undergoes FS at a certain time. The system's inputs and trajectories can be found in Figures 4.6 and 4.7.

The partitioning divides the data set in two segments, removing the data corresponding to FS, as shown in 4.16. FS is detected and z_{sl} is retrieved, obtaining:

$$z_{sl} = 0.9744$$

which is close to the real value $z_{sl} = 1$.

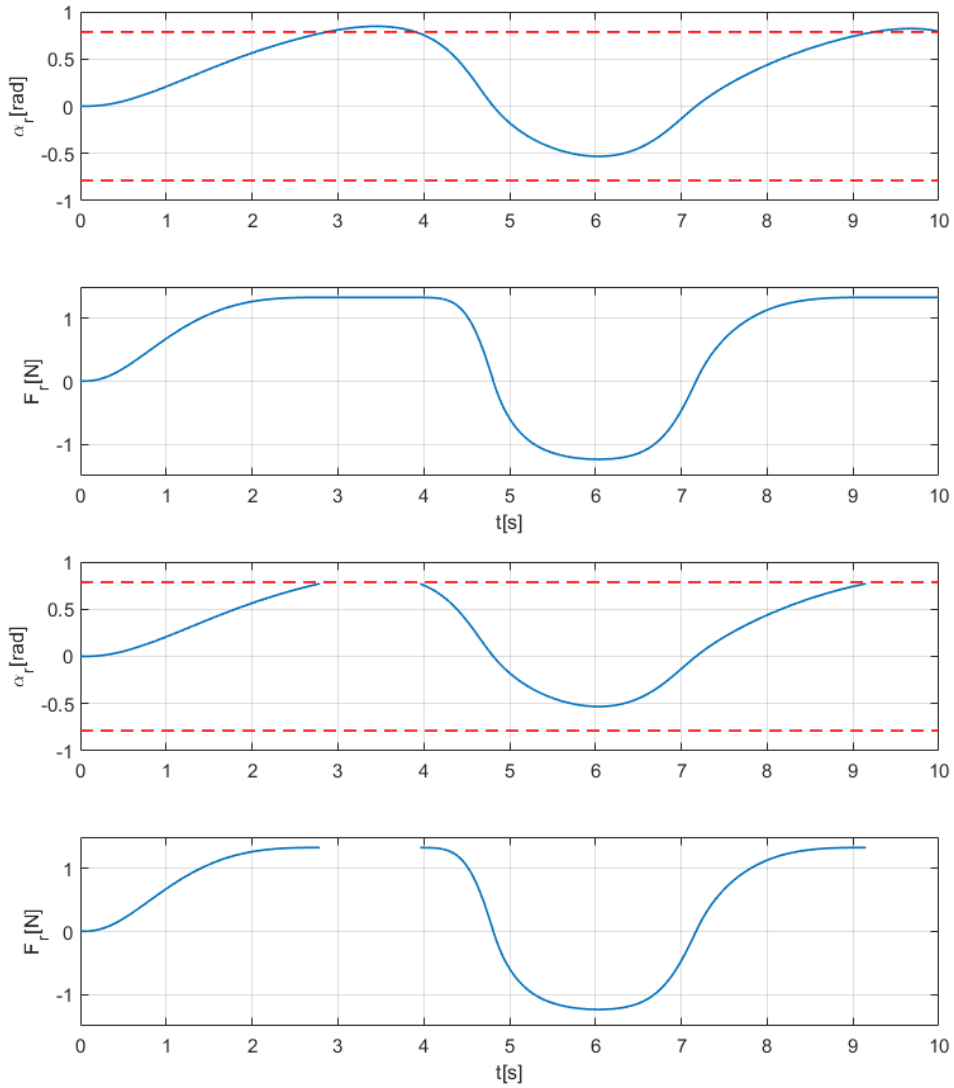

Figure 4.16: Comparison of α_r before and after the partitioning.

The LFT identification is run the two segments, attaining the following results, respectively for the first and the second segment.

Parameter	δ	Real	Estimated	% Error	J
Front Tire Cornering Stiffness	C_f	3	3.0390	+1.30%	
Rear Tire Cornering Stiffness	C_r	4	4.0651	+1.63%	$5.01115 \cdot 10^{-6}$
Moment of Inertia	I_z	0.03	0.0293	−2.33%	

Parameter	δ	Real	Estimated	% Error	J
Front Tire Cornering Stiffness	C_f	3	3.0697	+2.32%	
Rear Tire Cornering Stiffness	C_r	4	4.0903	+2.26%	$1.0265 \cdot 10^{-5}$
Moment of Inertia	I_z	0.03	0.0300	0%	

By averaging the results, according to the mixed strategy (4.7), the final estimates are obtained.

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	3	3.0535	+1.78%
Rear Tire Cornering Stiffness	C_r	4	4.0770	+1.92%
Moment of Inertia	I_z	0.03	0.0297	−1%

4.5 Identification from experimental data and validation

In this section the discussed identification algorithms are run on data retrieved from tests on the experimental setup of Section 4.5.1.
It should be noted that the available data do not include cases in which the system is drifting: the values of α_f and α_r remain quite low in the considered tests and FS is not detected. Therefore, the algorithm for removal of FS data will not be validated in this section. However, the estimates of the lateral forces from experimental data are discussed, allowing to verify if the results make sense and if they suggest future tests on drifting data.

4.5.1 Experimental setup

A 1:10 scale car-like vehicle (Fig. 4.17) inspired by the ones used by ETH and Georgia Institute of Technology researchers [7, 12], has been adopted to test the autonomous drifting control strategy.
The platform is a RWD car, actuated by a current controlled brushless motor, and equipped with four independent suspensions and an electric steering servo along with a rear differential. The car can be either manually controlled with a standard RC radio system, or can autonomously drive thanks to the installation of an embedded PC Odroid XU4 that runs a ROS control architecture. In both situations, an Arduino UNO provides a bidirectional communication with steering, propulsion, and wheel encoders.

Moreover, the car is equipped with an IMU, providing linear acceleration, angular velocity and attitude measurements, and a marker that allows to track vehicle position and orientation at a frequency of 100 Hz using a 12-camera OptiTrack motion tracking system, which has been used to estimate vehicle longitudinal and lateral velocities.

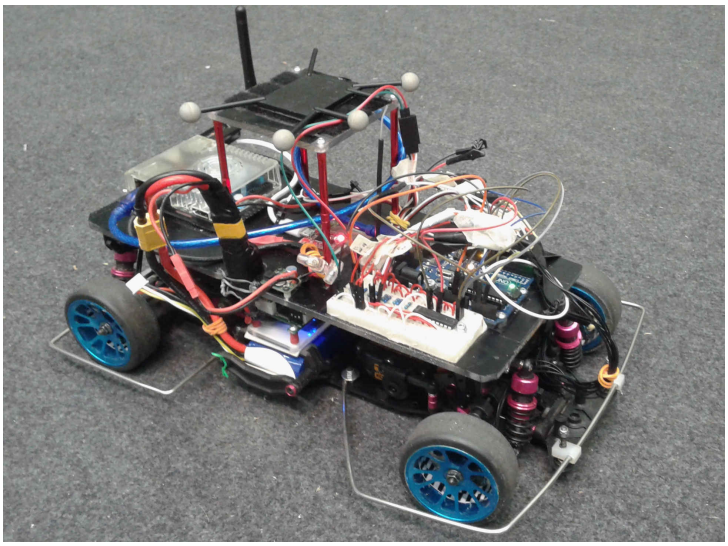

Figure 4.17: The experimental platform.

The steering servo dynamics has been identified by mounting the IMU on one of the front wheels and by recording the yaw rate in response to a step of the angular position of the servo. It has been verified that servo dynamics is well reproduced by a first order low-pass filter, with cut-off frequency of 50 rad/s and a delay of 0.04 s, when the wheels are not touching ground, and 0.06 s when the car is lying on ground.

Vehicle mass and COG position have been measured with a weight balance, while the estimation of the other parameters has been conducted using the methodology described in [4], which is based on the minimization of the error between measured yaw rate and lateral velocity and the ones obtained by forward simulating vehicle

dynamic model (3.1). In particular, the lower bound on the value of the yaw moment of inertia (0.015 Kgm^2) has been taken as the yaw moment of inertia of the car without the hardware component mounted. This value has been estimated by hanging the car to the tip of an industrial manipulator equipped with a torque sensor, and measuring the reaction torque to a trapezoidal angular speed profile applied to the vehicle yaw axis. The upper bound has been taken as the yaw moment of inertia obtained by concentrating the whole mass of the vehicle at the front, m_f, and rear, m_r, axles: $J_z^{max} = m_f a^2 + m_r b^2$.

Car parameters have been identified using an eight-shaped trajectory. The results are reported in Table 4.1.

Table 4.1: Experimental Platform Data

Mass	1.9	Kg
Yaw moment of inertia	0.03	$\mathrm{Kg\,m}^2$
Distance of COG from front axle	0.1368	m
Distance of COG from rear axle	0.1232	m
Tyre-ground friction coefficient (carpet)	0.35	
Front wheels' cornering stiffness (carpet)	47.86	$\mathrm{N\,rad}^{-1}$
Rear wheels' cornering stiffness (carpet)	127.77	$\mathrm{N\,rad}^{-1}$
Tyre-ground friction coefficient (wood flooring)	0.22	
Maximum steering angle	$\pm$ 45	deg
Steering servo actuator bandwidth	8	Hz
Steering servo actuator delay	0.09	s

4.5.2 Linear

The algorithm based on the linear tire model, discussed in Section 4.2, is run on data retrieved from a test carried at low speed and steering angle. The input variables and the trajectory of the vehicle are shown in Figures 4.18 and 4.19.

The identification procedure sets the known parameters to the following values:

Vehicle mass	m	1.9	kg
Distance of front axle from C.O.G.	a	0.1368	m
Distance of rear axle from C.O.G.	b	0.1232	m
Steering gain	G	1.1	

The results of the estimation are shown in the table below.

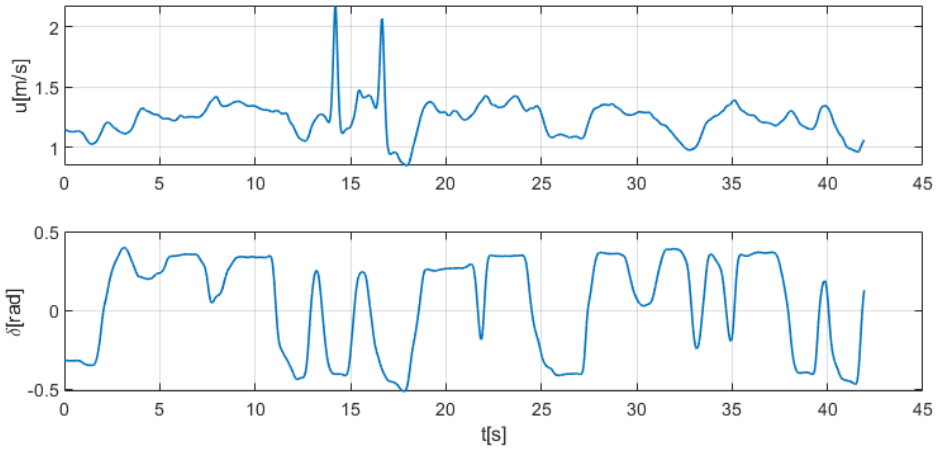

Figure 4.18: Input variables.

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	47.86	49.4435	+3.3%
Rear Tire Cornering Stiffness	C_r	127.77	105.1553	−17.7%
Moment of Inertia	I_z	0.03	0.0725	+142%

A mismatch can be noticed w.r.t. the nominal values. To check if the accuracy of the results is acceptable, data can be validated by comparing the output measurements to the output variables of the LFT model, when fed by the same inputs. Figure 4.20 shows that the identified model is sufficiently accurate to describe the behavior of the system.

FS is not expected in these data, but the algorithm for lateral forces estimation has been tested to verify if it works correctly on data corresponding to low tire slip angles. Figures 4.21 and 4.22 report the estimated lateral forces as functions of α_f and α_r, showing that they fit to the linear tire model. Values corresponding to low longitudinal speed or low α_f could lead to noticeable spikes, due to divisions by values close to zero in the LFT model.

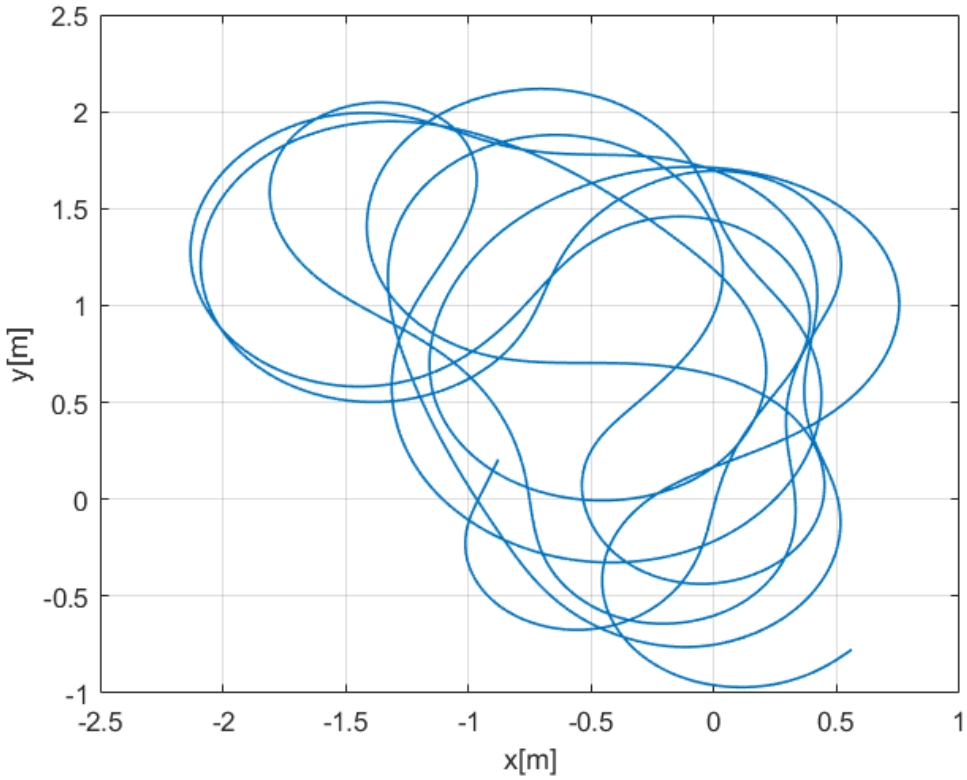

Figure 4.19: Trajectory of the vehicle.

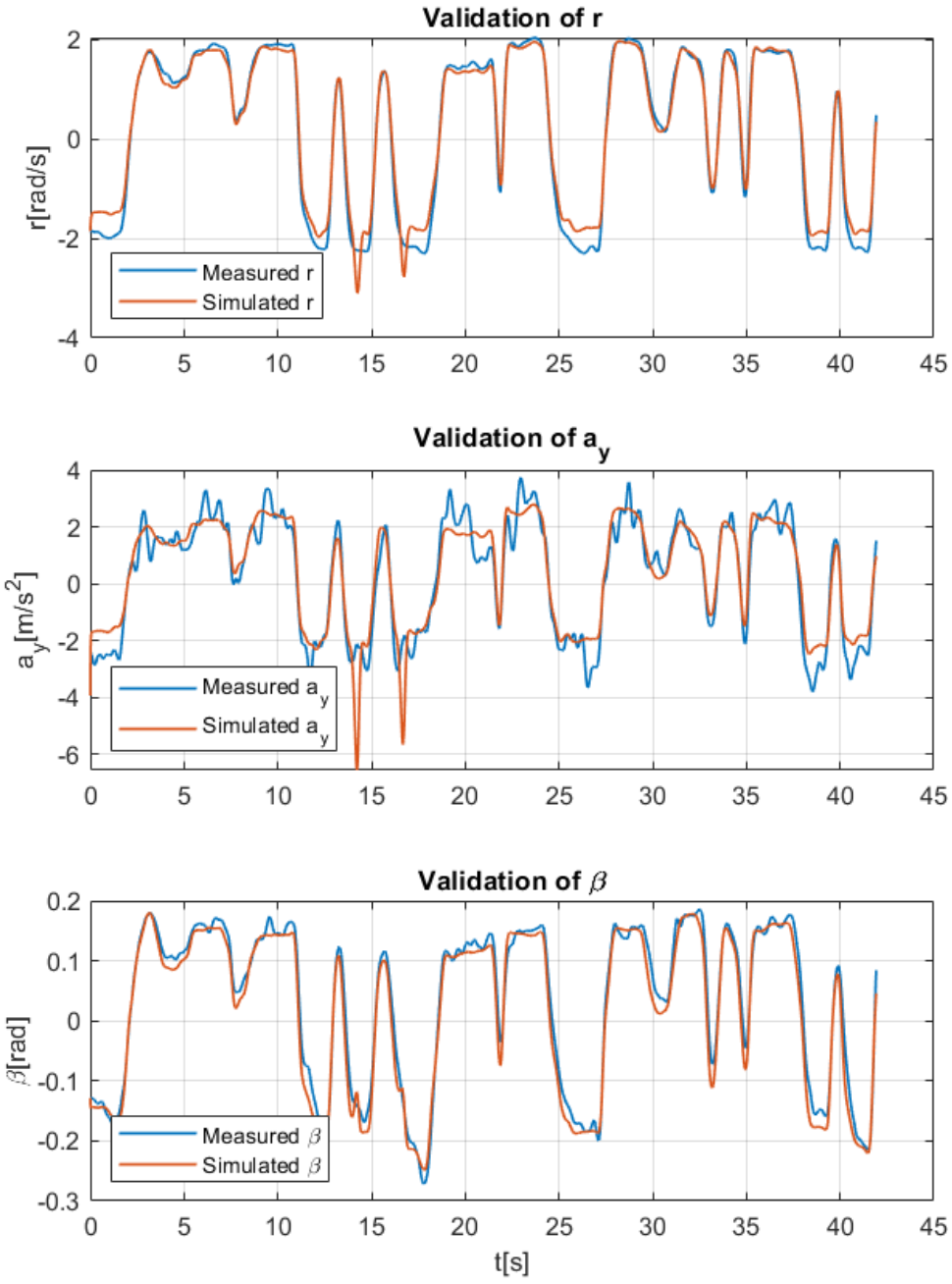

Figure 4.20: Comparison between measured output variables and simulated output variables.

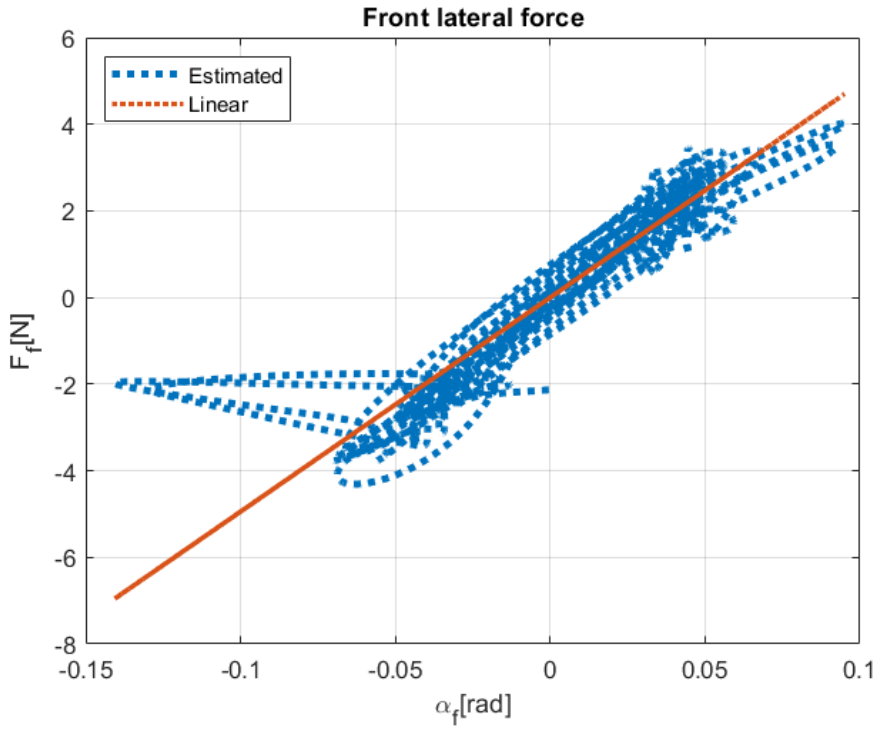

Figure 4.21: Front lateral force as function of α_f.

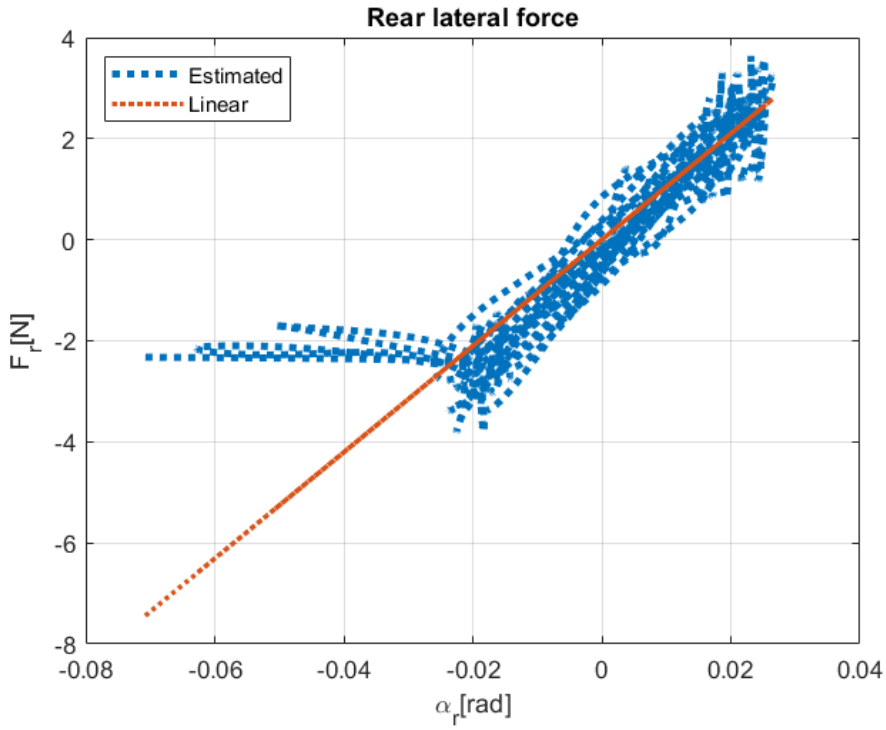

Figure 4.22: Rear lateral force as function of α_r.

4.5.3 Fiala: non-drifting

To test the identification algorithm based on the Fiala model from Section 4.3, a different data set is chosen, from an experiment conducted at higher speed and sterring angle than the previous test. The input variables and the trajectory are reported in Figures 4.23 and 4.24.

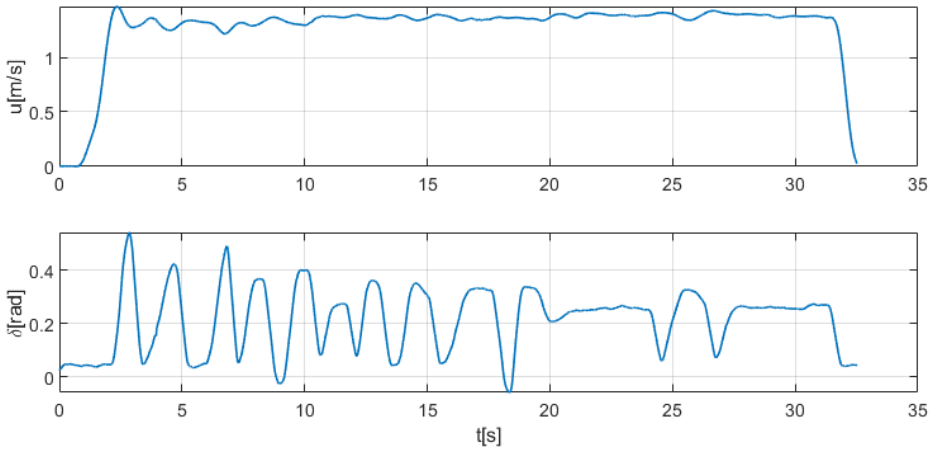

Figure 4.23: Input variables.

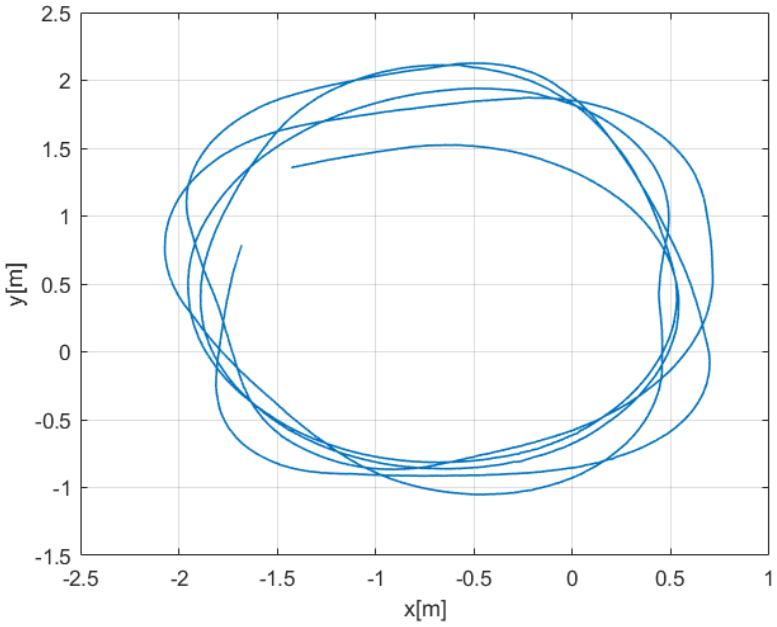

Figure 4.24: Trajectory of the vehicle.

The identification procedure sets the known parameters to the following values:

Vehicle mass	m	1.9	kg
Distance of front axle from C.O.G.	a	0.1368	m
Distance of rear axle from C.O.G.	b	0.1232	m
Steering gain	G	1.0255	
Full Sliding boundary	z_{sl}	0.2149	

The results of the estimation are shown in the table below.

Parameter	δ	Real Value	Estimated Value	% Error
Front Tire Cornering Stiffness	C_f	47.86	44.8325	-6.32%
Rear Tire Cornering Stiffness	C_r	127.77	113.2773	-11.34%
Moment of Inertia	I_z	0.03	0.0860	$+187\%$

As in the previous section, a comparison between measurements and simulated data is conducted to validate the results.

Figure 4.25 shows that, although a slight error is obtained in the estimated parameters, the identified model is accurate.

In Figures 4.26 and 4.27, $F_f(\alpha_f)$ and $F_r(\alpha_r)$, obtained from their estimated values, are reported. It is evident that the front lateral force fits the Fiala model more accurately than the linear one, justifying the usage of the identification procedure based on the Fiala model.

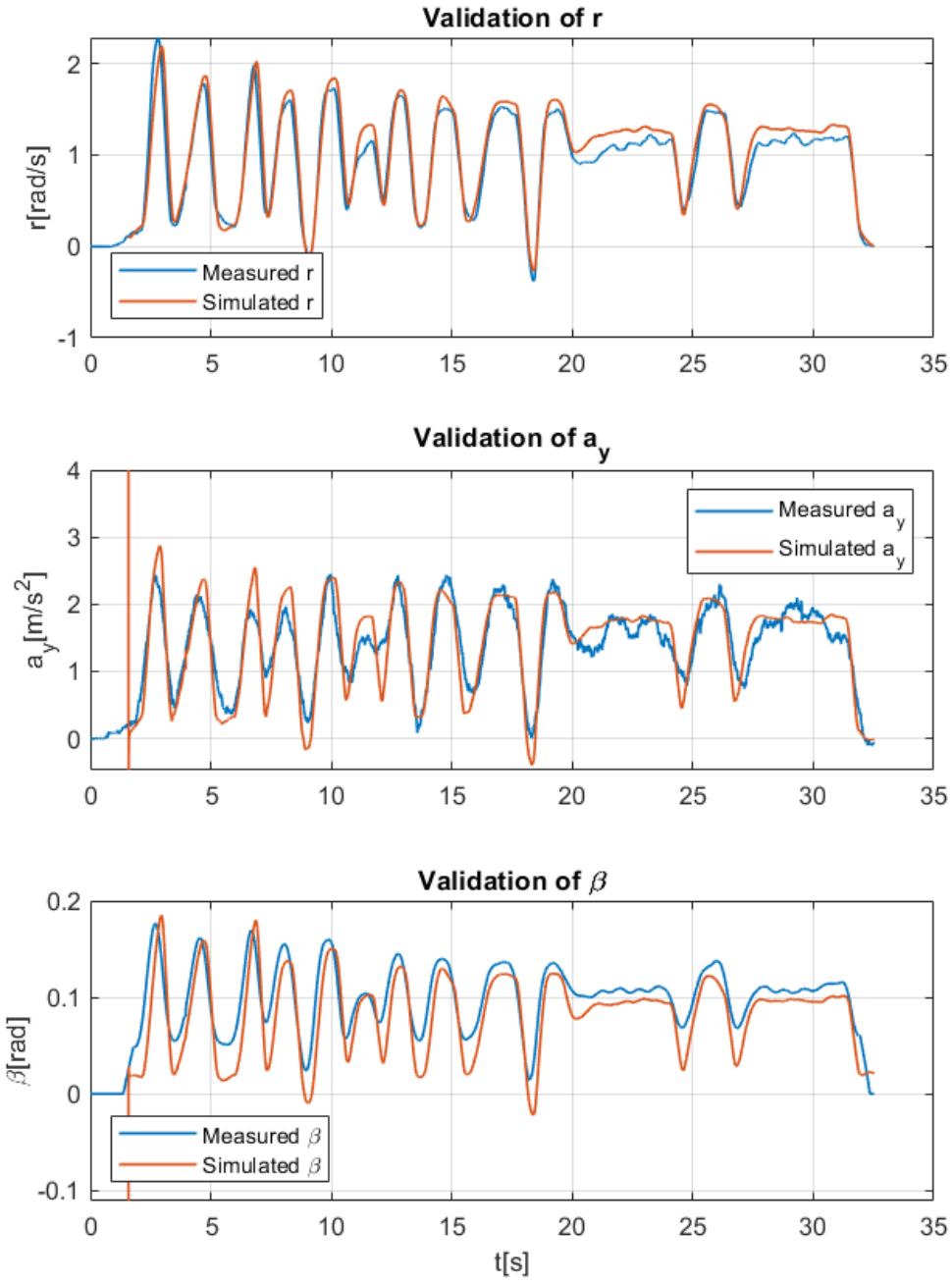

Figure 4.25: Comparison between measured output variables and simulated output variables.

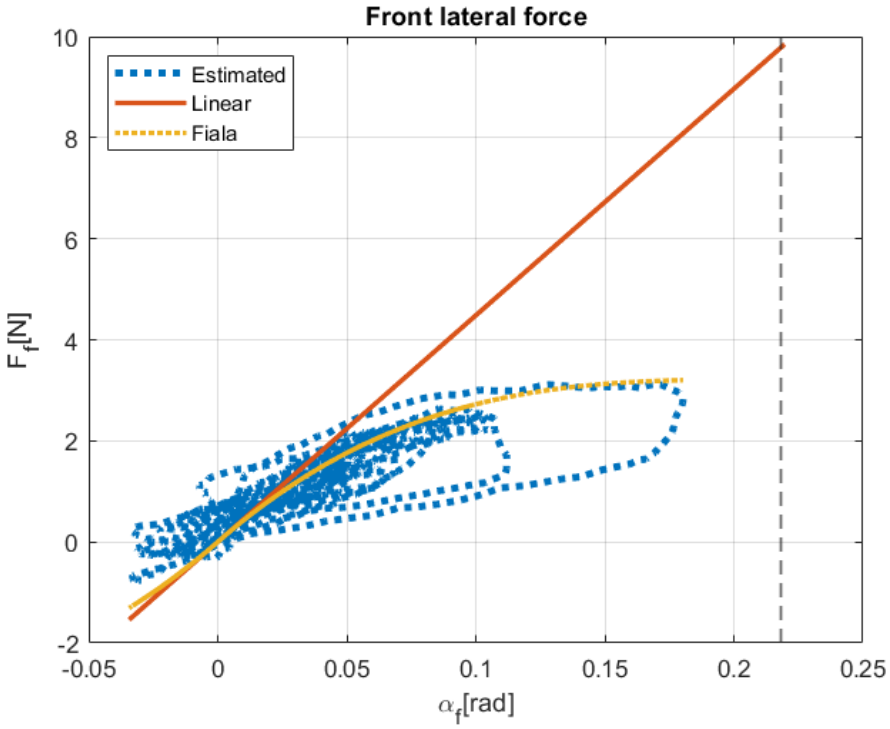

Figure 4.26: Front lateral force as function of α_f. The FS boundary is represented as the vertical dashed line.

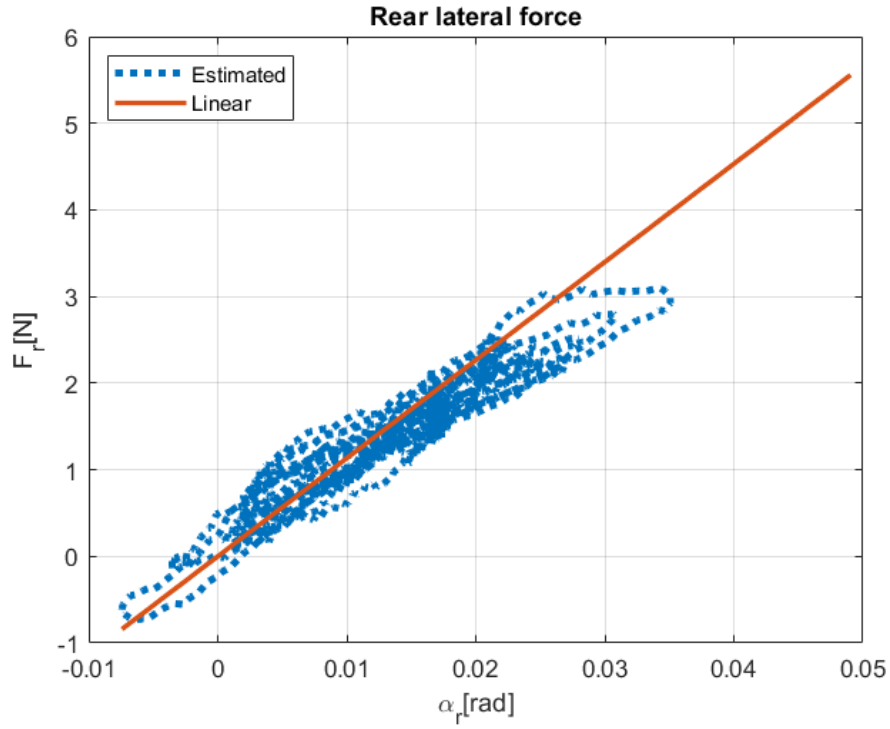

Figure 4.27: Rear lateral force as function of α_r.

Chapter 5

Conclusions and future improvements

In this work an identification algorithm based on the single-track model for vehicle lateral dynamics and on the Fiala tire model has been implemented and tested on simulated data with promising results. The problem of Full Sliding has been introduced and discussed, and a method to tackle it has been suggested and discussed. We concluded that, if such data are removed from the whole data set, identifiability can be recovered. The algorithms have been tested on experimental data, showing they can achieve a good performance in estimating the required parameters, either when the linear and the Fiala models are considered as identification models.

In future, drifting experimental data could be considered, to test the effectiveness of the partitioning algorithm and its robustness.
Other future improvements of the algorithm could include the identification of other parameters of the single-track model, such as the distances of front and rear axles from the C.o.G. a and b, and the steering gain G.
An interesting test could be carried out to verify the robustness of the identification. Indeed, the side-slip angle is retrieved from expensive optical measurements in the experimental setup. A cheaper mean to obtain measurements of the vehicle position is the GPS, but it is susceptible to noise: one may think that geolocation is typically subject to a certain radius of uncertainty. This disturbance can be modelled as a white noise with nonzero mean, which could be implemented in a simulated setup to test the robustness of the algorithm towards it.
All the considered algorithms are meant for offline applications, since the whole data set is known prior to the identification. Therefore, an implementation of the LFT toolbox for online applications would be interesting, as it would allow to estimate the parameters in real time, while the data are being collected. In the case of drifting maneuvers, the partitioning algorithm could be useful to the online algorithm, which can collect enough data from start-up to the first aggressive cornering to correctly estimate the unknown parameters. Thus, the system would have enough information to recognize Full Sliding.

Appendix A

User Manual

A.1 A nonlinear model

The nonlinear system chosen to help understanding the use of the toolbox is a simple mass, spring and damper system, depicted in Fig. A.1 and described by the following model:

$$\dot{x}_1 = x_2 \tag{A.1}$$

$$\dot{x}_2 = -\frac{k_1}{m}x_1 - \frac{k_2}{m}x_1^3 - \frac{c}{m}x_2 + \frac{1}{m}u \tag{A.2}$$

$$y = x_1 \tag{A.3}$$

where x_1 is the position of the mass, x_2 is the velocity, the constants m, k_1, k_2 and c are respectively the mass, the linear stiffness coefficient, the nonlinear stiffness coefficient and the damping coefficient, the input u is an external force applied to the mass and the output y of the system is the position of the mass.

The mass and the linear stiffness coefficient are assumed as known and are equal to $m = 1$ Kg and $k_1 = 20$ N/m respectively, while the nonlinear stiffness and the damping coefficients have to be identified, thus $\delta_1 = c$, $\delta_2 = k_2$.

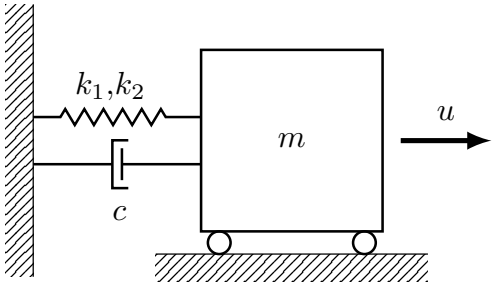

Figure A.1: A nonlinear mass, spring, damper system

An equivalent LFT reformulation of the model is thus given by:

$$\dot{\mathbf{x}} = \begin{bmatrix} x_2 \\ -\frac{k_1}{m}x_1 - \frac{1}{m}w_1 - \frac{1}{m}w_2 + \frac{1}{m}u \end{bmatrix} \tag{A.4}$$

$$\mathbf{z} = \begin{bmatrix} x_2 & \zeta \end{bmatrix}^T \tag{A.5}$$

$$\omega = x_1 \tag{A.6}$$

$$y = x_1 \tag{A.7}$$

$$\mathbf{w} = \begin{bmatrix} \delta_1 z_1 & \delta_2 z_2 \end{bmatrix}^T \tag{A.8}$$

$$\zeta = \omega^3 \tag{A.9}$$

and the only non-null matrices relevant to formulation (2.8–2.13) are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k_1}{m} & 0 \end{bmatrix} \quad , \quad \mathbf{B}_1 = \begin{bmatrix} 0 & 0 \\ -\frac{1}{m} & -\frac{1}{m} \end{bmatrix} \quad , \quad \mathbf{B}_3 = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \tag{A.10}$$

$$\mathbf{C}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad , \quad \mathbf{D_{12}} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{A.11}$$

$$\mathbf{C_2} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{A.12}$$

$$\mathbf{C_3} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{A.13}$$

In order to perform parameter identification two scripts must be created:

- `spring_lft.m`: the function script, called in the main script, where the LFT form is defined.
- `main.m`: the main script where the data are loaded, the identification problem is defined and all the main functions are called.

A.2 `spring_lft.m`

As already mentioned, the `spring_lft.m` script is a function called in the main script where the LFT model is defined.

In this first line (Lis. A.1) the function output `lftfun` is declared, which is the description of the LFT model. The input parameters are the known parameters of the model, m and k_1, and as many (2-dimensional) row vectors as the unknown parameters, containing the limits for the constrained search optimization. In this example we have two vectors `k2lim` and `clim`.

Listing A.1: Declaration of the function

```
function [lftfun] = spring_lft(k1,m,k2lim,Clim)
```

Then, we have to specify the number of inputs $\mathbf{u}$, states $\mathbf{x}$, outputs $\mathbf{y}$ and omega variables $\boldsymbol{\omega}$ of the LFT model, these definitions will later be used to correctly create and size the linear time invariant matrices of the LFT model.

Listing A.2: Definition of the number of inputs **u**, states **x**, outputs **y** and omega variables $\boldsymbol{\omega}$ of the LFT model

```
%definition of the number of inputs, states, outputs and
    omega of the LFT
u_count = 1;
x_count = 2;
y_count = 1;
om_count = 1;
```

A.2.1 DeltaSym and theta_sym

We will now define the matrix $\boldsymbol{\Delta}$, named `DeltaSym`, as shown in Lis. A.3.

Listing A.3: Definition of the matrix $\boldsymbol{\Delta}$

```
%definition of the matrix delta
DeltaSym = {
    'parName' 'indStartDiag' 'indStopDiag' 'LowerBound'
       'HigherBound' 'toIdentify', 'lb', 'ub';
    'c' 1 1 Clim(1) Clim(2) 1 -1 1;
    'k2' 2 2 k2lim(1) k2lim(2) 1 -1 1;
        };
delta_count = 0;
for i=2:size(DeltaSym,1)
    delta_count = delta_count + DeltaSym{i,3} - DeltaSym
       {i,2} + 1;
end
```

In this matrix the first row, not to be modified, simply acts as a guide for the construction of the matrix itself.

The first column contains the names of the unknown parameters, in our case `'c'` (second row) and `'k2'` (third row).

The second and third column are used to indicate the first and last position of the unknown parameter on the $\boldsymbol{\Delta}$ matrix's diagonal. In our case the unknown parameters appears only once in the $\boldsymbol{\Delta}$ matrix, respectively $c = \delta_1$ in position (1,1) and $k_2 = \delta_2$ in position (2,2).

In the fourth and fifth column the lower and upper bound of the unknown parameters are defined.

The sixth column can be filled with 1 or 0 if we, respectively, want or don't want to identify the parameter. In the latter case the parameter will be maintained fixed at its initial value.

The seventh and eighth column should always be respectively filled with -1 and 1, defining the lower and upper bound of the limits in normalized form.

At last, once completed the definition of the `DeltaSym` matrix, the number of elements in the diagonal of matrix $\boldsymbol{\Delta}$ is computed.

The next lines (Lis. A.4) create the column vector $\boldsymbol{\zeta}$ named `theta_sym`, after having defined an array of symbolic $\boldsymbol{\omega}$.

Please note that in case of two or more $\boldsymbol{\omega}$ the initial `if`' condition can be skipped.

Finally, the number of components of the vector $\boldsymbol{\zeta}$ is determined.

Listing A.4: Definition of the vector $\boldsymbol{\zeta}$

```
%definition of the matrix theta (= ZETA)
if om_count == 1
    om = sym(['om1_1']);
else
om = sym('om', om_count);
end

theta_sym = [
    om(1)*om(1)*om(1);
    ];
theta_count = size(theta_sym,1);
```

A.2.2 LTI matrices

Initially, all matrices of the LFT formulation are filled with zeros (Lis. A.5), then the only non-null elements are defined (Lis. A.6).

Listing A.5: Construction of the LTI matrices

```
%construction of the matrices of the LTI part
LTI = struct(...
    'A', zeros(x_count,x_count),...
    'B1', zeros(x_count,delta_count),...
    'B2', zeros(x_count,theta_count),...
    'B3', zeros(x_count,u_count),...
    'C1', zeros(delta_count,x_count),...
    'D11', zeros(delta_count,delta_count),...
    'D12', zeros(delta_count,theta_count),...
    'D13', zeros(delta_count,u_count),...
    'C2', zeros(om_count,x_count),...
    'D21', zeros(om_count,delta_count),...
    'D22', zeros(om_count,theta_count),...
    'D23', zeros(om_count,u_count),...
    'C3', zeros(y_count,x_count),...
    'D31', zeros(y_count,delta_count),...
    'D32', zeros(y_count,theta_count),...
```

```
    'D33', zeros(y_count,u_count));
```

Listing A.6: Definition of the LTI matrices

```
%definition of the matrices
% dx1 = x2
LTI.A(1,2) = 1;
% dx2 = -k1/m*x1-1/m*w1-1/m*w2+1/m*u
LTI.A(2,1) = -k1/m;
LTI.B1(2,1) = -1/m;
LTI.B1(2,2) = -1/m;
LTI.B3(2,1) = 1/m;

% z1 = x2
LTI.D12(2,1) = 1;
% z2 = zeta
LTI.C1(1,2) = 1;

% om1 = x1
LTI.C2(1,1) = 1;
% LTI.C2(2,1) = 3;

% y1 = x1
LTI.C3(1,1) = 1;
```

We are finally able to construct the `lftfun` struct, which is the output of the function, by initially defining it and then calling the `lft_finalize` function.

Listing A.7: Definition of the `lftfun` struct

```
%initial definition of the LFTfun
lftfun = struct(...
    'LTI', LTI,...
    'DeltaSym', {DeltaSym},...
    'DeltaVal', zeros(delta_count,delta_count)...
        );

% save additional data for reference purpose
lftfun.theta_sym = theta_sym;
lftfun.u_count = u_count;
lftfun.x_count = x_count;
lftfun.y_count = y_count;
lftfun.om_count = om_count;
lftfun.theta_count = theta_count;
lftfun.delta_count = delta_count;

%function to finalize the definition of LFTfun
lftfun = lft_finalize(lftfun);
```

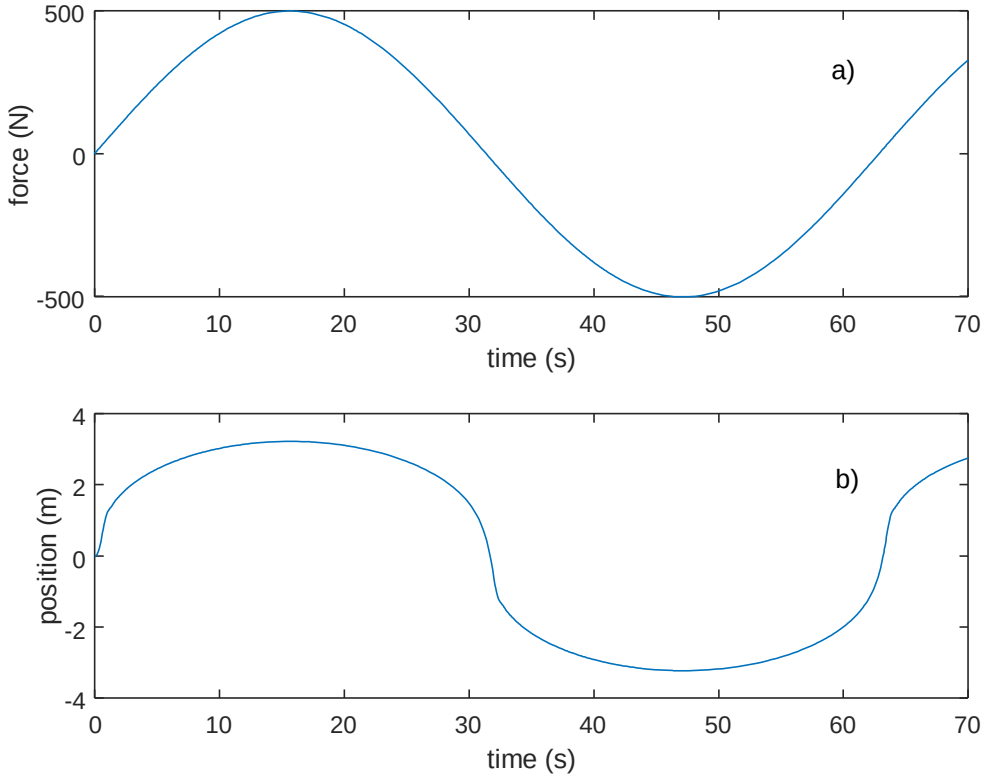

Figure A.2: Input force a) and position b)

```
end
```

A.3 `main.m`

The `main.m` script is where all the data are loaded, where the solver options are defined and, finally, where all the main functions are called to tackle the identification process.

The data used for parameter identification will be collected by applying the input (Fig. A.2 a))

$$u(t) = 500 \sin(0.1t) \tag{A.14}$$

and sampling the output of system (A.1–A.3) (Fig. A.2 b)), assuming the following values for the parameters to be identified:

$$c = 8 \text{ Ns/m} \tag{A.15}$$
$$k_2 = 13 \text{ N/m}^3 \tag{A.16}$$

Of course, since this is an ideal case, an almost perfect convergence to the actual values of the parameters is expected.

The time vector and the input values are stored in the file `F.mat`, while the output values are stored in the file `lft_y.mat`. Note that the output values are always assumed to be evaluated at the time instants of the time vector in the input matrix.

As shown in Lis. A.8, after a very generic clean up of the workspace, command window and of the possible figures open, the script begins by loading the input and output data of the system, that will be used for identification.

Listing A.8: Loading of the input and output data

```
clc, clear all, close all

%import input (Force 'F') NB every input or state comes
   with its timevector
load F.mat;
t = F(:,1);
F = F(:,2);

%import the output (position 'x')
load('lft_y');
```

Then, before defining the `lftfun` struct, the known parameters of the model, in our case m and k_1, are defined, as well as the limits of the constrained search on the optimal values of the parameters (Lis. A.9).

In this case, wide ranges have been defined, in order to show the capabilities of the tool. For the identification of multiple parameters of more complex systems, in order to speed up the estimation procedure, it is recommended to choose reasonable ranges in which the LFT identification Toolbox should find the optimal solution.

Listing A.9: Definition of the `lftfun` struct

```
%definition of the constant and known parameters
k1=20; %[kN/m]
m=1; %[Kg]

%definition of the limits of the uncertain parameters
%in this case the values to be found is 'k2=13' and 'c
   =8'
k2lim = [1 200];
Clim = [1 200];

%load lft function and solver options
lftfun = spring_lft(k1,m,k2lim,Clim);
```

In the case of multiple outputs their influence on the cost function (2.3) can be

weighted through the matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & W_p \end{bmatrix} \tag{A.17}$$

in turn, the weights W_i can be defined by defining the vector `lftfun.ISO.Nominal` as the vector

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_p \end{bmatrix}^T \tag{A.18}$$

$$\alpha_i = \frac{1}{\sqrt{W_i}} \tag{A.19}$$

where `lftfun` is the structure containing the description of the LFT model.

One possible choice is:

$$W_i = \frac{1}{(M_i - m_i)^2} \tag{A.20}$$

where M_i is the highest value of y_i and m_i is the lowest one, which can be easily implemented as:

Listing A.10: Definition of the `lftfun.ISO.Nominal` vector

```
M = max(lft_y);
m = min(lft_y);
for i=1:length(M)
    alpha(i) = M(i) - m(i);
end
lftfun.ISO.Nominal = alpha;
```

The next piece of code defines the options for the LFT Solver (see Section A.4), the (struct) input and the initial conditions of the model's state variables.

Listing A.11: Definition of options for the LFT solver, the input and the initial conditions of the model's state variables

```
%load LFT solver options
LFTsolverOptions = lftSet('RelTol', 1e-4,...
    'AbsTol', 1e-8,...
    'SolutionInterpMethod', 'spline',...
    'SolutionTimeSpan', t,...
    'SensAlgorithm', 'ode15s',...
    'OversamplingMethod', 'spline' );

%define the input
Input = struct('Type', 'interpolated', 'Samples', F, '
   Time', t);
```

```
%define the initial conditions of the model's state
   variables
InitialConditions = struct('StateInitialConditions', [0;
    0]);
```

Similarly, the options for the LFT optimizer (see Section A.5) and the initial guess of parameters in normalized form[1] are defined before running the estimation through the optimizer.

The identified parameters (not normalized) are finally presented through the `norm2abs` function and the Hessian and its condition number are shown.

Listing A.12: Definition of options of the LFT optimizer, the initial guess of parameters in normalized form and start of the identification

```
%load LFT estimator options
LFToptimOptions = lftOptSet('Display', 'iter',...
    'MaxIter', 120, ...
    'TolFun', 1e-8 ...
    );

% estimate
% initial guess of parameters in normalized form
lftfun.DeltaVal = diag([0 0]);

%estimation
[DELTA_opt,fval,J,grad,H,CN, history] = lftOptDelta(
   lftfun,Input,InitialConditions,LFTsolverOptions,lft_y
   ,LFToptimOptions);

norm2abs(lftfun, lftfun.DeltaVal, DELTA_opt);

H=H(:,:,end), CN=CN(end)
```

A.3.1 Results

The results obtained by running the `main.m` script are shown in the command window as in Fig. A.3

In the first part of the command window informations on all iterations are shown: the first column reports the number of the iteration, the second the value of the cost function, the third the norm of each step, the fourth the first-order optimality measure and the last the number of conjugate gradient iterations taken.

At the end of the identification the `fmincon` function, that is the function used to minimize the cost function, will print the *stopping criteria* that stopped the

[1] An initial guess equal to zero means that the initial guess of the parameter is equal to the mid-point in between the search limits.

```
LFT identification task

                                Norm of      First-order
 Iteration        f(x)          step          optimality   CG-iterations
     0           0.712623                        0.374
     1           0.141291        1.02064         0.257           1
     2          0.0107467       0.362213        0.0711           1
     3        0.000548436       0.142858        0.0057           1
     4        1.99397e-05      0.0622938      0.000652           1
     5        7.03737e-08      0.0157541       3.2e-05           1
     6        1.76169e-08    0.000873076       2.46e-07          1
     7        1.75299e-08    2.00117e-07        1.8e-06          1

Local minimum possible.

fmincon stopped because the final change in function value relative to
its initial value is less than the selected value of the function tolerance.

<stopping criteria details>

-----------------------------------------------------------------------------------------
|  Name            |  Initial Value  |  Optimal Value  |  Lower Bound  |  Higher Bound  |
-----------------------------------------------------------------------------------------
|  c               |  100.5          |  7.9995         |  1            |  200           |
|  k2              |  100.5          |  12.9996        |  1            |  200           |
-----------------------------------------------------------------------------------------

H =

    1.9477    0.2527
    0.2527   30.0364


CN =

   15.4409

>>
```

Figure A.3: Results of identification

identification. In our case the stopping criteria is the final change in function value relative to its initial value, which is less than the selected value of the function tolerance: 10^{-8}. It is also possible to click on `<stopping criteria details>` to have more details about the stopping criteria trigger.

Then, the `norm2abs` function prints the optimal values found by the LFT Identification Toolbox for the unknown parameters, along with their name, the initial guess values and the search limits. In the example, the values found by the LFT Identification Toolbox, after 7 iterations, are:

$$c = 7.9995 \text{ Ns/m} \quad \text{(A.21)}$$

$$k_2 = 12.9996 \text{ N/m}^3 \quad \text{(A.22)}$$

which correspond respectively to a 0.0062% and 0.0031% error, with respect to the real values of the parameters.

A.4 LFT Solver

The LFT solver function, which is not directly used in the main script, is the most important part of the LFT Toolbox since it is the heart of the optimization function.

The LFT solver function, as its name suggests, solves the nonlinear, time invariant LFT model given the initial state, the input and the options. The solver function is called as follows:

Listing A.13: Call of the LFT solver

```
[output,internalSolution,CommonTerms] =
    lftSolver(lftfun,Input,InitialConditions,
       lftSolverOptions);
```

The outputs of the function are:

- `output`: a matrix of $1+p$ columns, where p is the number of system outputs. The first column is the time vector t, while the other p columns are the output components of the system evaluated at time instants in vector t.
- `internalSolution`: a matrix of $1+m+n+n_z+n_\omega+p$ columns, where m, n, n_z and n_ω are respectively the number of inputs, states, $\mathbf{z}$ variables and $\boldsymbol{\omega}$ variables. The first column is the vector of the time instants chosen by the solver and in the other columns we find, in order, the inputs, the state variables, the $\mathbf{z}$ variables, the $\boldsymbol{\omega}$ variables and the outputs, evaluated at the time instants of the first column.
- `CommonTerms`: a struct containing the matrices of the LFT model both fixed, relative to the linear part of the system and time-variable, relative to the non-linear part of the system.

The inputs are the following:

- `lftfun`: struct containing the LFT model (in our example defined by the function `spring_lft.m`).
- `Input`: struct composed by:
  - `Type`: defines the type of the input: `discontinuous`, `interpolated` or `continuous`.
  - `Samples`: defines the values, for each time instants, of the inputs.
  - `Time`: defines the time vector of the inputs.
- `InitialConditions`: contains the vector of the initial conditions of the state variables of the LFT system.
- `lftSolverOptions`: struct containing the solver options composed by:
  - `RelTol`: relative tolerance on the solution (default 10^{-3}).
  - `AbsTol`: absolute tolerance on the solution (default 10^{-6}).
  - `MaxOrder`: maximum order of accuracy for `ode15s` solver (default 5).
  - `BDF`: variable setting the use of BDF method (default `off`).
  - `InitialStep`: initial stepsize (default 10^{-3}).
  - `MaxStep`: maximum stepsize (default not defined).
  - `NumberOfSteps`: maximum number of steps (default not defined).
  - `ShowIntTime`: boolean variable setting the printing in the command window of the time instant of each step (default `false`).
  - `SolutionInterpMethod`: type of interpolation that should be executed on the input values (default `linear`).
  - `OversamplingMethod`: type of interpolation that should be executed on the output values (default `linear`).
  - `SolutionTimeSpan`: time vector on which the output should be evaluated.
  - `Sensitivity`: For `Sensitivity = 0` the solver simply simulates the LFT system. For `Sensitivity = 'pX'`, with `X` equal to the number of the unknown parameter, it is possible to execute the sensitivity of the outuput with respect to the chosen unknown parameter (default `0`).
  - `SensAlgorithm`: algorithm to be used to compute the sensitivity (default `ode15s`).
  - `CommonTerms`: vector of common terms of the first stage (solver in simulation configuration) that can be passed to the sensitivity solver in order to be almost 100 times more rapid than the first stage.

A.5 LFT Optimizer

As for the `LFTsolverOptions` in Sec. A.4 we are going to explain all the components of this struct:

- `TolFun`: lower bound on the change in the value of the objective function during a step (default 10^{-3}).
- `MaxStep`: maximum amplitude of the step in the optimization procedure (default not defined).
- `NumberOfSteps`: maximum number of steps of the optimization procedure (default 2).
- `PolynomialDegree`: degree of the interpolating polynomial in case of continuous type input (default empty).
- `RelTolX`: relative tolerance (default 10^{-3}).
- `MinNormGrad`: minimum value of the norm of the gradient (default 10^{-3}).
- `EpsilonLambda`: maximum value of tolerance for the zero (default 10^{-6}).
- `MaxIter`: maximum number of Newton iterations for the calculation of the step (default 20).
- `Display`: flag for printing in the command window the Newton iterations for the search of the optimal step (default `off`).
- `StartOptimSample`: first time instant, as position in the time vector, from which the algorithm should begin the optimization process (default 1).

To summarize, the function requires the following inputs:

- `lftfun`: LFT model of the system.
- `Input`: input variables struct.
- `InitialConditions`: initial conditions struct.
- `LFTsolverOptions`: solver options struct.
- `lft_y`: outputs matrix (in our example a vector).
- `LFToptimOptions`: optimizer options struct.

The output of the function are:

- `DELTA_opt`: row vector containing the identified parameter in normalized form.
- `fval`: final cost function's value.
- `J`: column vector containing the values of the cost function at each step.
- `grad`: matrix containing the value of the gradients at each step.

- `H`: struct containing the Hessian matrices at each step.
- `CN`: vector containing the condition number at every step.
- `history`: matrix containing the values of the unknown parameters at each step.

Appendix B

LFT formulation of the single-track model

B.1 Linear tire model

Starting from the single-track model in canonical representation:

$$\begin{cases} \dot{v} = \frac{C_f}{m}\left(-\beta - a\frac{r}{u} + G\delta\right) + \frac{C_r}{m}\left(-\beta + b\frac{r}{u}\right) - ur \\ \dot{r} = \frac{C_f}{I_z}a\left(-\beta - a\frac{r}{u} + G\delta\right) - \frac{C_r}{I_z}b\left(-\beta + b\frac{r}{u}\right) \\ \dot{\psi} = r \\ \dot{x} = u\cos\psi - v\sin\psi \\ \dot{y} = u\sin\psi + v\cos\psi \end{cases}$$

If only r, a_y and β are chosen as output variables, the system can be reduced to the first two equations only.

$$\begin{aligned} \dot{v} &= \frac{C_f}{m}\left(-\beta - a\frac{r}{u} + G\delta\right) + \frac{C_r}{m}\left(-\beta + b\frac{r}{u}\right) - ur \\ \dot{r} &= \frac{C_f}{I_z}a\left(-\beta - a\frac{r}{u} + G\delta\right) - \frac{C_f}{I_z}b\left(-\beta + b\frac{r}{u}\right) \end{aligned}$$

The state vector $\mathbf{x}$ and output vector $\mathbf{y}$ can be defined as:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} v & r \end{bmatrix}^T \\ \mathbf{y} &= \begin{bmatrix} r & a_y & \beta \end{bmatrix}^T = \begin{bmatrix} x_2 & w_1 + w_2 & \zeta_1 \end{bmatrix}^T \end{aligned}$$

while the unknown parameter matrix $\boldsymbol{\delta}$ and the auxiliary vectors $\mathbf{w}$ and $\mathbf{z}$ are given by:

$$\begin{aligned}\boldsymbol{\delta} &= \begin{bmatrix}\frac{C_f}{m} & \frac{C_r}{m} & \frac{C_f}{I_z}\end{bmatrix}^T \\ \mathbf{w} &= \begin{bmatrix}z_1\delta_1 & z_2\delta_2 & z_3\delta_3\end{bmatrix}^T \\ \mathbf{z} &= \begin{bmatrix}-\zeta_1 - a\zeta_2 + Gu_2 \\ -\zeta_1 + b\zeta_2 \\ -a\zeta_1 - a^2\zeta_2 + aGu_2\end{bmatrix}\end{aligned}$$

the vector $\boldsymbol{\zeta}$ and the auxiliary vector $\boldsymbol{\omega}$ are:

$$\begin{aligned}\boldsymbol{\zeta} = \theta(\boldsymbol{\omega}) &= \begin{bmatrix}\arctan\left(\frac{\omega_1}{\omega_3}\right) \\ \frac{\omega_2}{\omega_3} \\ \omega_2\omega_3 \\ \frac{\omega_5\omega_6}{\omega_4}\end{bmatrix} \\ \boldsymbol{\omega} &= \begin{bmatrix}x_1 & x_2 & u_1 & w_1 & w_2 & w_3\end{bmatrix}^T\end{aligned} \tag{B.1}$$

and finally vector $\dot{\mathbf{x}}$:

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}w_1 + w_2 - \zeta_3 \\ w_3 - \frac{b}{a}\zeta_4\end{bmatrix}$$

B.2 Fiala tire model

If the Fiala model is chosen in place of the linear tire model, the equations of the single-track Model become:

$$\begin{cases}\dot{v} = \frac{C_f}{m}f(\alpha_f) + \frac{C_r}{m}f(\alpha_r) - ur \\ \dot{r} = \frac{C_f}{I_z}af(\alpha_f) - \frac{C_r}{I_z}bf(\alpha_r)\end{cases}$$

where $f(\alpha)$ is the Fiala tire model (3.5):

$$f(\alpha) = \begin{cases}z\left(1 - \frac{|z|}{z_{sl}} + \frac{z^2}{3z_{sl}^2}\right) & |z| < z_{sl} \\ \frac{z_{sl}}{3}\text{sign}(\alpha) & |z| \geq z_{sl}\end{cases}$$

recalling that $z = \tan(\alpha)$.
The state vector $\mathbf{x}$ and output vector $\mathbf{y}$ can be defined as:

$$\begin{aligned}\mathbf{x} &= \begin{bmatrix}v & r\end{bmatrix}^T \\ \mathbf{y} &= \begin{bmatrix}r & a_y & \beta\end{bmatrix}^T = \begin{bmatrix}x_2 & w_1 + w_2 & \zeta_1\end{bmatrix}^T\end{aligned}$$

the unknown parameter matrix $\boldsymbol{\delta}$ and the auxiliary vectors $\mathbf{w}$ and $\mathbf{z}$ are given by:

$$\begin{aligned}\boldsymbol{\delta} &= \begin{bmatrix}\frac{C_f}{m} & \frac{C_r}{m} & \frac{C_f}{I_z}\end{bmatrix}^T \\ \mathbf{w} &= \begin{bmatrix}z_1\delta_1 & z_2\delta_2 & z_3\delta_3\end{bmatrix}^T \\ \mathbf{z} &= \begin{bmatrix}\zeta_2 \\ \zeta_3 \\ a\zeta_2\end{bmatrix}\end{aligned}$$

the vector $\boldsymbol{\zeta}$ and the auxiliary vector $\boldsymbol{\omega}$ are:

$$\begin{aligned}\boldsymbol{\zeta} &= \begin{bmatrix}\arctan\left(\frac{\omega_1}{\omega_7}\right) \\ f(-\omega_6 - a\frac{\omega_2}{\omega_7} + G\omega_8) \\ f(-\omega_6 + b\frac{\omega_2}{\omega_7}) \\ \omega_2\omega_7 \\ \frac{\omega_4\omega_5}{\omega_3}\end{bmatrix} \\ \boldsymbol{\omega} &= \begin{bmatrix}x_1 & x_2 & w_1 & w_2 & w_3 & \zeta_1 & u_1 & u_2\end{bmatrix}^T\end{aligned} \tag{B.2}$$

where $f(\cdot)$ is the implemented form of the Fiala model, shown in equation (3.6). Finally, vector $\dot{\mathbf{x}}$ is defined as:

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}w_1 + w_2 - \zeta_4 \\ w_3 - \frac{b}{a}\zeta_5\end{bmatrix}$$

Ringraziamenti

Il conseguimento di questa laurea non sarebbe stato possibile senza l'aiuto e il supporto delle persone che mi sono state vicine e vorrei dedicare alcune righe per ringraziarle.
Rivolgo i primi ringraziamenti al Prof. Ferretti e al Prof. Bascetta, che mi hanno grandemente supportato nei mesi dello svolgimento della tesi, fornendomi sempre consigli utili per proseguire il lavoro e per destreggiarmi nelle difficoltà, concedendomi sempre la massima disponibilità. E' stato un vero piacere lavorare con loro.
Grazie alla mia famiglia, a mio papà Raffaele e mia mamma Monica che mi hanno sempre aiutato quando ne ho avuto bisogno e hanno sempre creduto in me. Ringrazio anche mio fratello Matteo, che nonostante ora viva in Olanda non manca mai di contattarmi quotidianamente per farmi sapere che mi vuole bene.
Ringrazio anche il resto della mia famiglia, mia zia Caterina, i miei cugini Davide ed Elisa e il mio caro nonno Franco che avrà sicuramente letto la tesi. Grazie per avermi fatto trascorrere dei momenti stupendi in quel del Buillet.
Gli ultimi ringraziamenti vorrei rivolgerli ai miei amici, che per me sono un bene inestimabile e con i quali ho formato un legame stupendo. Grazie a Jan, Coach, Luca, Danilo, Dimitri, Guglielmo, Pedro, Ilaria e Giulia che mi sopportano dai tempi del liceo e che continuano a regalarmi gioie. Grazie a tutte le altre Dodsie, Francesca, Marta, Chiara e Pietro, con cui non ci vediamo più molto spesso ma ai quali sono sempre affezionato. Grazie a Simone, Eleonora e Dario per essermi stati amici e per avermi sempre aiutato durante la triennale a Genova. E per ultimi, ma solo in ordine cronologico, vorrei ringraziare gli amici che ho conosciuto al Politecnico, Aldo e Riccardo, due persone estremamente in gamba con cui spero di poter collaborare nuovamente in futuro.
Vorrei chiedere scusa, come mio solito, a coloro che non ho citato, spero che non ci rimangano male. Grazie a tutti!

Bibliography

[1] A. D. Bona, G. Ferretti, E. Ficara, and F. Malpei. LFT modelling and identification of anaerobic digestion. *Control Engineering Practice*, 36:1 – 11, 2015. [cited on pages 10, 14, and 18]

[2] F. Demourant and G. Ferreres. Closed loop identification of a LFT model. *Journal Européen des Systémes Automatisés*, 36(3):449–464, 2002. [cited on page 13]

[3] F. Donida, C. Romani, F. Casella, and M. Lovera. Integrated modelling and parameter estimation: an LFT-Modelica approach. In *2009 IEEE Conference on Decision and Control*, pages 8357–8362, Dec 2009. [cited on page 18]

[4] L. Fagiano, M. Lauricella, D. Angelosante, and E. Ragaini. Identification of induction motors using smart circuit breakers. *IEEE Transactions on Control Systems Technology*, (99):1–9, 2018. [cited on page 50]

[5] K. Hsu, T. Vincent, G. Wolodkin, S. Rangan, and K. Poolla. An LFT approach to parameter estimation. *Automatica*, 44(12):3087–3092, 2008. [cited on page 13]

[6] L. H. Lee and K. Poolla. Identification of Linear Parameter-Varying Systems Using Nonlinear Programming. *Journal of Dynamic Systems, Measurement, and Control*, 121:71–78, Mar. 1999. [cited on page 13]

[7] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015. [cited on page 50]

[8] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, Upper Saddle River, NJ, USA, 1999. [cited on page 14]

[9] A. Ros. Sideslip estimation using a lft-based estimator with commercial positioning and inertial sensors. Master's thesis, Politecnico di Milano, 2017. [cited on page 10]

[10] B. Schramm, Hiller. *Vehicle Dynamics - Modeling and Simulation*. Springer,

1999. [cited on page 10]

[11] M. Viganò. Lft-based parameter identification of the lateral dynamics of vehicles. Master's thesis, Politecnico di Milano, 2019. [cited on page 10]

[12] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *IEEE International Conference on Robotics and Automation*, pages 1433–1440, 2016. [cited on page 50]

List of Figures