3
PyCar: A versatile vehicle simulation package

Currently, there are many vehicle simulation softwares, e.g. CarSim [55], MSC Adams Car [56] and VI-CarRealTime [57]. In general, they are employed for vehicle dynamic analysis and design of control strategies for active safety systems. However, in many cases, these softwares are difficult to use and/or a license is required. In addition, in the previous section, the fully non-linear and three-dimensional road vehicle model was derived using nine rigid bodies and the Jordain's principle. This model includes, a handling tire model and a non-linear suspension model. Finally, the chassis, the tire and the suspension system, were included in one global package that was named in this thesis as PyCar.

PyCar is an essential part of this thesis and it was developed in order to test control strategies, to analyze the vehicle dynamics and the suspension system kinematics. Due to this characteristic, it was denominated as a versatile vehicle dynamic simulation package. Furthermore, PyCar was built using, predominantly, Python as a programming language. The choice of this programming language was because it is well accepted in the scientific community as well as there is a solid support behind its main numerical package, e.g. SciPy [58], Numpy [59] and Matplotlib [60].

3.1 Computational vehicle model

The global multibody vehicle model is completely described by the Equation (2-71). This nonlinear system of differential equations can be analyzed by numerical time integration. Furthermore, the computational effort to solve this system is generally large due to the complexity of the vehicle model. Consequently, a trade-off between accuracy and time consuming is required in order to choose the correct solver. A theoretical review of methods to solve a system of differential equations can be found in [61, 62]. In [63], numerical time integration methods for a multibody vehicle model are considered.

The commercial programming platform Matlab® offers seven solvers for technical computing as presented in [64], i.e. `ode23`, `ode45`, `ode113`, `ode15s`, `ode23s`, `ode23t`, `ode23tb`. Each of them has their advantages and disadvangaes against a stiff and non-stiff problems [61]. Generally, `ode45` is the best function to apply as a “first try” for the majority of problems. However, if the programmer/user suspects that the system is a stiff problem, probably the `ode15s` solver is the best option. Especifically, because the multibody vehicle model is highly non-linear, i.e. a stiff problem, implicit solvers, e.g. `ode15s` or `ode23s`, are needed. Furthermore, because PyCar was implemented in Python then, a equivalent Matlab® implicit solver need to be employed.

3.1.1 Numerical integration

PyCar use the `VODE` [65] solver for the numerical solution of the multibody vehicle model described in this thesis. Probably, this set of subroutines for numerical solution of initial-value problems (IVP) is the best know ODE solver for stiff and non-stiff problems. Neverthless, before solving the system of differential equations of the multibody vehicle model of the form

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{z}} = \boldsymbol{g}(\boldsymbol{q}, \boldsymbol{z}, t),\ \boldsymbol{z}(t_0) = \boldsymbol{z}_0,\ \boldsymbol{z} \in \mathbb{R}^n \tag{3-1}$$

the solution of the $\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{z}, t)$ need to be computed first in order to have IVP of the form

$$\dot{y} = f(y, t),\ y(t_0) = y_0,\ \boldsymbol{y} \in \mathbb{R}^n \tag{3-2}$$

and finally, compute the its numerical solutions by applying the `VODE` solver. The solution of $\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{g}(\boldsymbol{q}, \boldsymbol{z}, t)$ is performed using a LU decomposition implemented by the Numpy package.

3.1.2 Virtual test driver

The driver can be thought as a sensor, controller and actuator over the vehicle. It senses the environment and the current state of the vehicle, then takes a decision based on its capabilities (experience) and finally actuates over the vehicle through the steering wheel, brake and throttle pedal, and the clutch. In addition, an average driver can recognize the linear behavior of the vehicle, i.e. when the vehicle is in the stability region. On the other hand, skilled drivers can operate the vehicle at its limits without allowing the car to become unstable.

In this thesis, an average driver is modeled using a Proportional-Derivative (PD) controller. In addition, the driving torque is controlled by the driver using a simple PI-controller where the vehicle longitudinal is considered as input. For steering control of this driver model, only one input is required, i.e. the cross-track error (ε). This input is defined as the difference between the current lateral position y and the desired one y_0, i.e. $\varepsilon = y(t) - y_0(t)$. Then, the driver model is proportional to this error and the variation of it. In addition, the controller has a delay τ, this means

$$\delta(t+\tau) = -K_p\varepsilon - K_d\dot{\varepsilon} \tag{3-3}$$

where K_p and K_d are the proportional and derivative gain of the controller respectively. Moreover, by expanding the function $\delta(t+\tau)$ using the Taylor series about time t and truncating it after the linear term, it is obtained

$$\tau\dot{\delta}(t) + \delta(t) = -K_p\varepsilon - K_d\dot{\varepsilon} \tag{3-4}$$

The delay τ of this first-order driver model includes the reaction time of the driver as well as the actuator delay. In addition, it is possible to assume a delay $\tau = 0.25\,s$ for an average driver and $\tau = 0.08\,s$ for skilled drivers [69].

Figure 22 shows a double lane-change maneuver performed by the PD driver defined in Equation (3-4). The vehicle model employed for this simulation is the simple handling model defined in Section 2. For this maneuver, a reference lateral position $y_0(t)$ is defined as follows

$$y_0(t) = \begin{cases} 5 & \text{for} \quad 1 < t < 8 \\ 0 & \text{for} \quad \text{other values} \end{cases} \tag{3-5}$$

It can be noticed that the PD driver is capable of following the reference trajectory quite well as shown in the plot of $\varepsilon - x$. In addition, a steering ratio $k_\delta = 1/17$ is considered, δ_w represents the steering wheel angle. Some characteristic parameters for the simple handling vehicle model and for the PD driver controller are shown in Table 6.

3.2 Animation environment

It is well know that the visualization of the simulation results by animation of the multibody system is an important feedback for the engineer. PyCar also offers a web-based visualization tool. Figure 23 shows a screen-shot of PyCar animation tool. In this 3D environment it is possible to create obstacles as well as represent different type of surfaces, this is particularly useful for μ-split scenarios. In addition, the time simulation is displayed in the left upper part of the panel as well as some useful bottoms to command the animation in the right upper part.

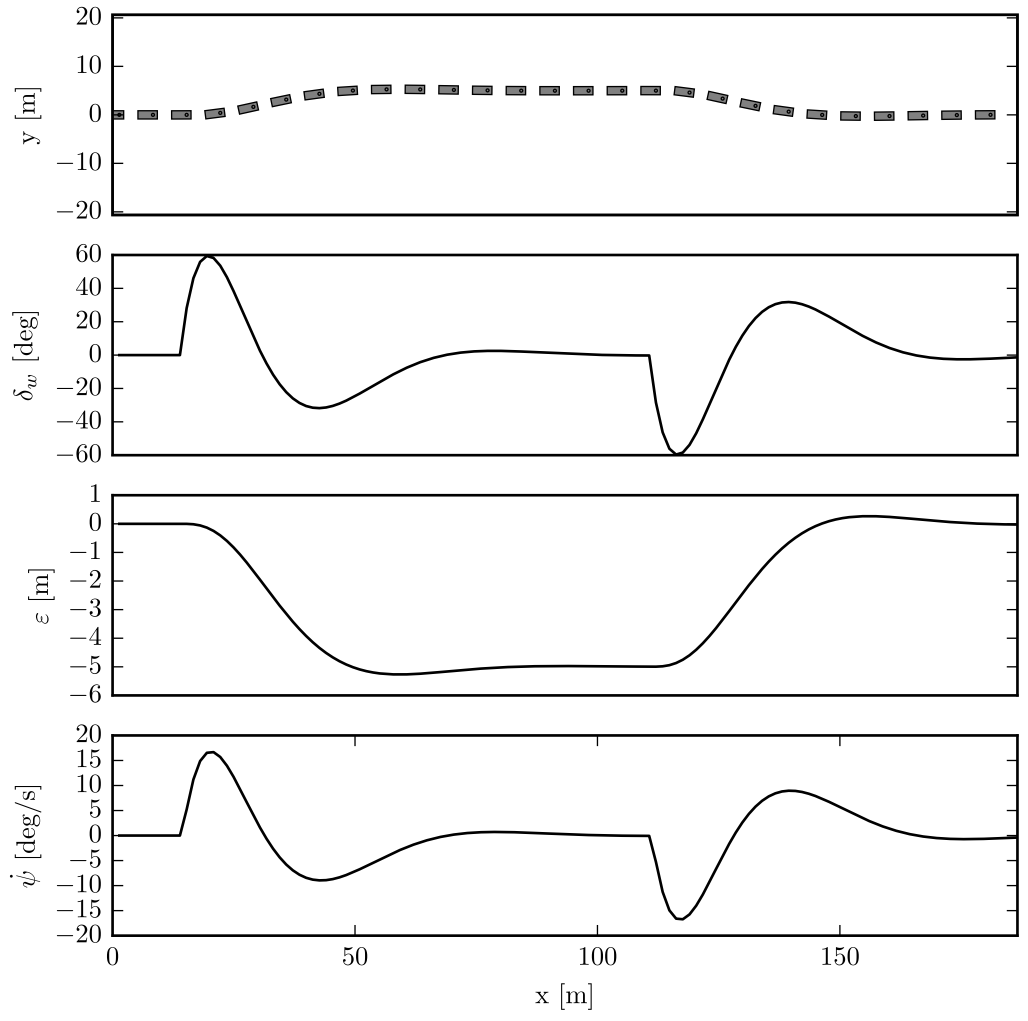

Figure 22: Double lane-change maneuver performed by the PD driver model using the simple handling model at 50 km/h.

3.3 Model validation using PyCar

In order to validate the multibody vehicle model through PyCar, a scaled car is employed, see Figure 24. This scaled car was built about 5 years ago at OTH Regensburg [16] and it is constantly improving. In addition, this scaled vehicle is highly oversteer, see Appendix B.1. For measurements of the lateral states, e.g. yaw rate $\dot{\psi}$ and lateral acceleration a_y, sensors were mounted in this scaled car. In addition, this scaled model is controlled using a Radio Control Joystick. Inertia and geometric data of the scaled vehicle are presented in Table 7.

For the validation, a double-lane change was performed in a surface of a static coefficient of friction of 0.7 approximately. Figure 25 shows the input steering angle performed by the driver using the Radio controller. This input is measured in %, it represents the maximum and minimum angle of the servo

Table 6: Simple handling model and Proportional-Derivative driver model parameters.

SHM			
Parameters	**Symbol**	**Value**	**Unit**
Vehicle mass	m	1724	kg
Vehicle z-axis inertia	Θ	1100	$kg \times m^2$
Distance c.o.g to front axle	l_f	1.35	m
Distance c.o.g to rear axle	l_r	1.15	m
Cornering stiffness of front axle	K_f	9×10^4	$N/-$
Cornering stiffness of rear axle	K_r	13.8×10^4	$N/-$
PD driver model			
Proportional gain	K_p	0.3	rad/m
Derivative gain	K_d	0.4	$rad \times s/m$
Delay	τ	0.25	s
Steering ratio	k_δ	1/17	-

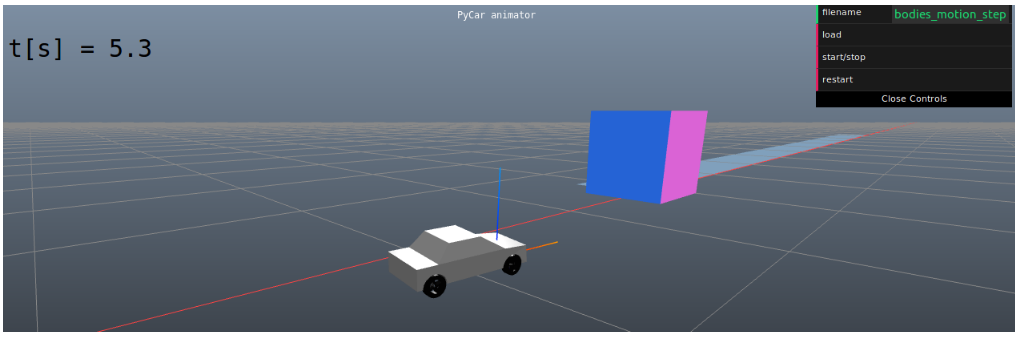

Figure 23: PyCar animation screen-shot.

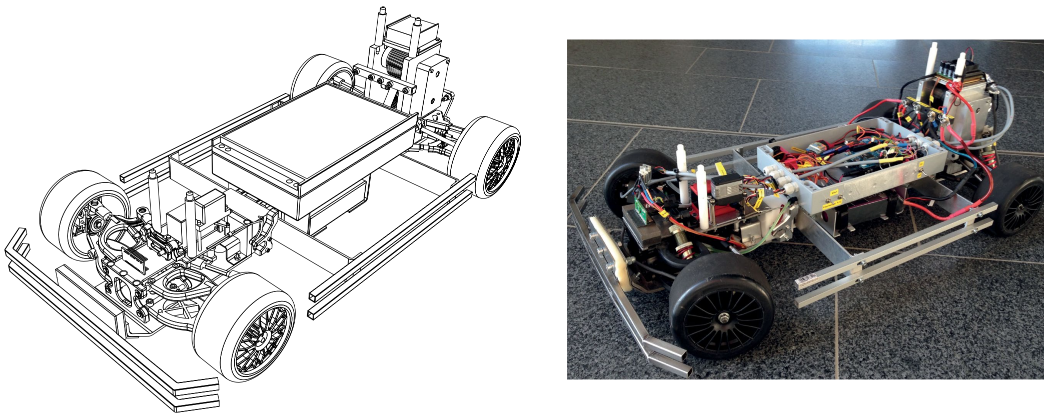

Figure 24: Three-dimensional CAD model and real scaled car.

motor that controls the steering angle. Moreover, the velocity of the maneuver was controlled to 5 m/s approximately.

Table 7: Scaled vehicle main inertia and geometric data.

Parameter	**Value**	**Unit**
Tire radius	0.0618	m
Wheel base	0.5080	m
Track width	0.3210	m
Height of c.o.g.	0.1397	m
Mass	12.215	kg
Inertia x-axis	0.1038	$kg \times m^2$
Inertia y-axis	0.5730	$kg \times m^2$
Inertia z-axis	0.6291	$kg \times m^2$

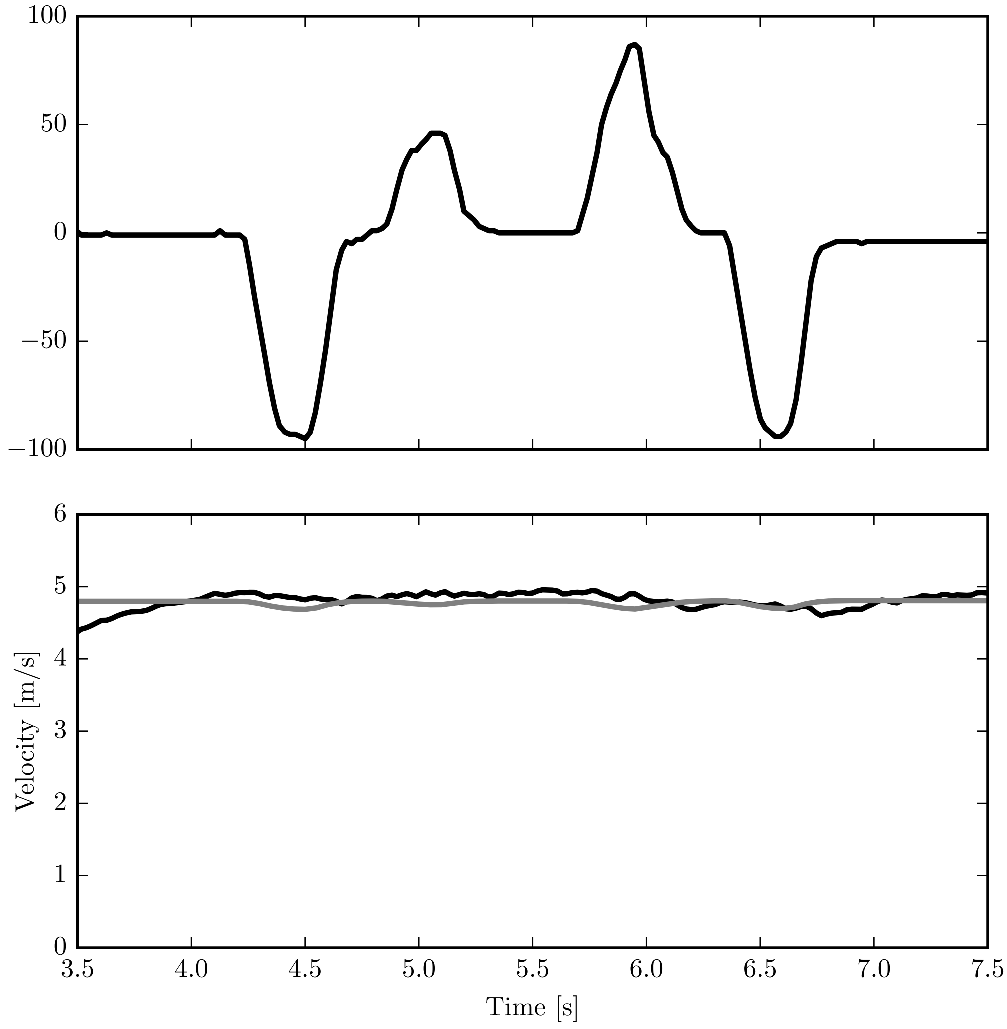

Figure 25: **Top**: input steering angle for a double lane change maneuver performed in a surface with a static coefficient of friction of 0.7 approximately. **Bottom**: controlled longitudinal velocity (black: measurement, gray: PyCar).

Figure 26 shows the lateral acceleration and yaw rate obtained by the sensors as well as the ones computed using PyCar. In qualitative terms, a good agreement between the measurements and states computed by PyCar can be observed.

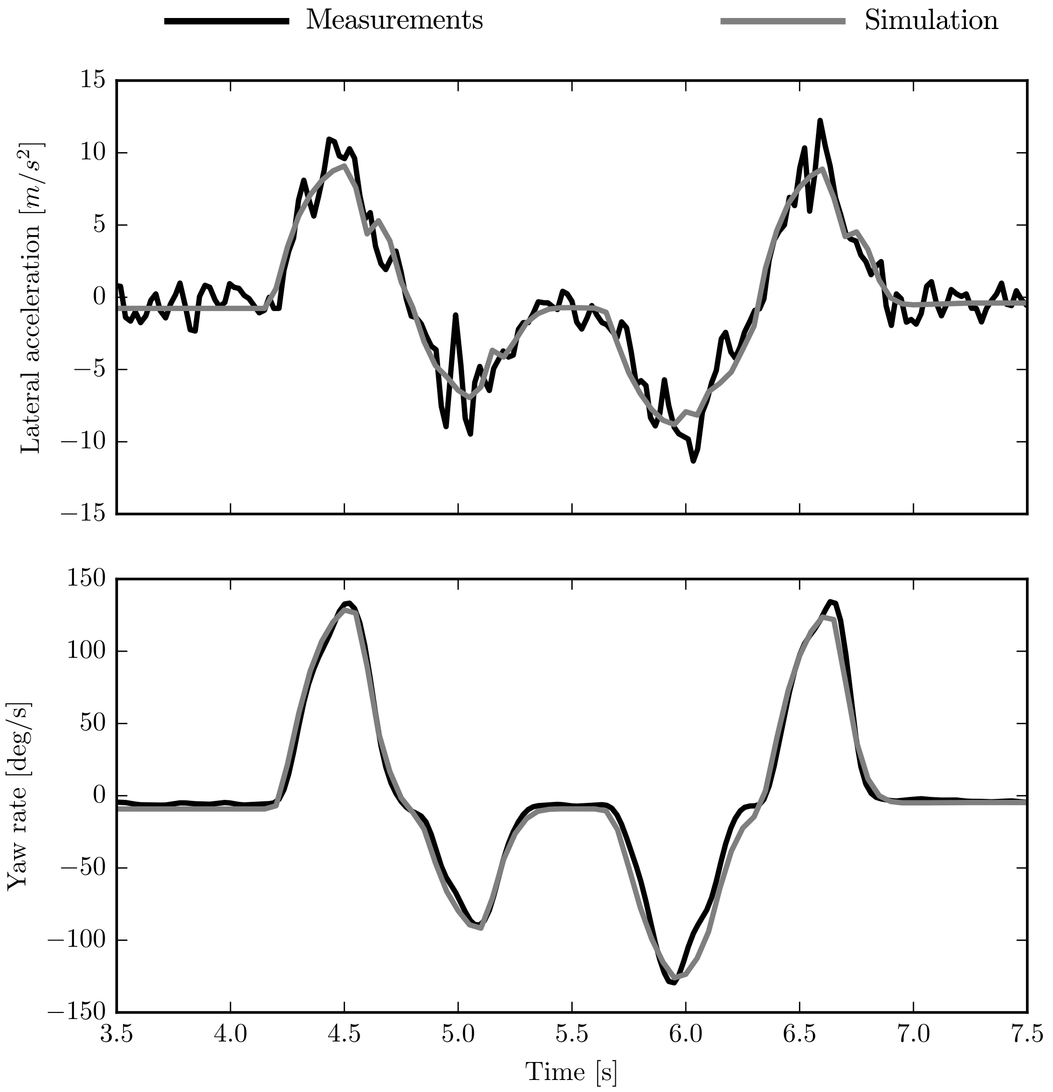

Figure 26: **Top**: lateral acceleration versus time. **Bottom**: yaw rate versus time in a double lane change maneuver (black: measurement, gray: PyCar).