Automotive Model-Based Differential System Development for Steering Control of an Autonomous Vehicle

BY

GABRIELE DE MATTEIS
B.S, Politecnico di Torino, Turin, Italy, 2013

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Chicago, 2016

Chicago, Illinois

Defense Committee:

Michael Scott, Chair
Sabri Cetinkunt, Advisor
Massimo Violante, Politecnico di Torino

ACKNOWLEDGMENTS

I want to thank my advisors, my family, my friends and all people that supported me during these years of study.

GDM

TABLE OF CONTENTS

CHAPTER **PAGE**

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

FIGURE **PAGE**

CHAPTER 1

INTRODUCTION

Autonomous vehicles have been discussed for many years, but, today, many companies and research are focusing on this field. The sharp increase in the grade of automation in vehicles is evident: for the past fifteen years many new faculties, for both higher safety and comfort, have been introduced. The next step is to produce a fully autonomous vehicle, able to drive itself among cars driven by humans in city traffic and on the highways. Beyond being an high technology record or a researchers competition, a fully autonomous vehicle also represents the "opportunity to improve high safety, decrease congestion, lower emissions, expand mobility, and create new economic opportunities for jobs and investment" (1), and many investors support this research.

1.0.1 Purpose

Given this context this thesis focuses on the problem of steering optimization for a two-wheel drive (2WD) electrical vehicle, employing an algorithm that reproduces a differential gear. The vehicle is designed for "The Freescale™ Cup", a competition organized by Freescale™. A white painted track with black borer lines is used in the race. The vehicles admitted are scaled with a 1:32 factor. They are characterized as fully-autonomous and use a line scan camera, in order to recognize the border lines of the track. A on-board Control Unit (CU) elaborates the information coming from the camera and provides command signals to the actuators, in order to keep the vehicle inside

the track. Any given trajectory is not stored, rather it elaborates its desired position at each time instant. The objective of the work is to realize a model-based software, performing the action of a differential gear for the vehicle. The idea of model-based design is increasing in relevance as it allows one to simulate a system, develops a controller and implement it with simplicity; as such, it is being employed by many industries. A graphical interface allows the utilization of common block-schemes, making the system description simple and easy to catch. The aim of this work is to contribute to the development of a model-based handling system for the vehicle. The software components already developed were: the vision algorithm which elaborates camera outputs and detects border lines, navigation that elaborates measured and desired lines position, servo management which, using navigation output, assigns a command signal to a servo motor, implying front wheels steering. The "MathWorks™" company provides a powerful software, MATLAB®, that includes one of the most famous model-based software: Simulink®. The company provides a support package for the utilized hardware, which includes a library of drivers for the board's peripherals. However the library does not have any implementation of speed sensor and any driver for peripherals useful to this scope. For this reason was necessary to find a way to drive peripherals and speed sensor. The solution was the integration of *C code* with the Simulink® ambient using the *S-Funcions*.

1.0.2 Implementation

In order to realize the differential algorithm, the kinematic of the vehicle has been used. Given the steering angle and the inverse kinematic, one can compute the ideal

speed for the posterior wheel, steering with constant curvature. To be sure that the required speeds were reached, a speed control for the DC motor was needed, which required the implementation of a speed sensor. The control realization should be based on the system model but, as in the case of study, the system parameters are unknown. For that reason, an identification method must be applied, in order to model a system and control it. Now follows a brief description of the points faced in the work.

1.0.2.1 Vehicle kinematics and equations for differential algorithm

At first the kinematics of steering vehicles has been studied, such as the Ackermann geometry and the bicycle model. Next, the equations regulating the kinematic behavior of the vehicle in steering conditions have been computed. Then the presence of the posterior wheels have been introduced, leaving behind the bicycle model, and considering a speed distribution, along the rear wheels axis. Finally, inverse kinematic relations, taking into account the wanted vehicle speed and the steering angle, have been used to compute the speed of the posterior wheels (the wanted set).

1.0.2.2 Model based speed sensor design

As previously stated, it was necessary to implement a speed sensor. First, a study was conducted on how the model-based software operates. With this information, the method employed in the sensor driver development. Then the physical working principle of the Hall sensor was studied; how is the output signal produced and with which shape? Furthermore the sensor must calculate the wheel speed given a predefined logic. Two principal logics have been exposed, both event based: *Fixed Window* and *Sliding Window*. These methods diverge in how they treat data in the speed computation, once the

obtained values have been validated according to the frequency of the output signal, as measured by the sensor and measured with the oscilloscope. Finally a brief explanation on the sensor block interface has been presented.

1.0.2.3 DC motor speed control design: model and control design

Once the sensor has been obtained, the further step was to produce a controller. Before that, it is necessary to model the system under study. At the beginning, all the components that took part in the system, from the commanded supply group to the wheel that is actuated, have been considered. That process includes: the *DC motor*, *H-Bridge* and *Tyre-Soil Contact*. In order to obtain a transfer function, the linear model of the DC motor, the dynamic consideration on the H-Bridge and the phenomena of tyre slip, were introduced. Since there was not a data sheet providing information about the DC motor, and given the presence of non linearities in the system and difficulties in modeling the tyre-soil contact behavior, the final transfer function was estimated using input and output of the system. In this way the system as constructed takes into consideration phenomena that, in classical modeling, would have been omitted. After the definition of the model's order, the coefficients regulating and modeling the system were estimated, and the final transfer function obtained. Thus the system was studied, the controller designed and synthesized, in order to match the specifications required. Finally, the PI controller was implemented in the hardware control unit and the input and output were evaluated, all in order to check if the requirements were reached.

1.0.2.4 Results and Complete system scheme

Finally, after obtaining the needed components, the whole system was assembled, implemented and subsequently tested with the steering angle handling system activated. Since the final goal of the work was the differential system, the test has been repeated with a fixed steering angle rather than the steering management already present. Before the new test execution, a previous verification of the actuator position output was performed. In that way it is possible to test only the differential action in a vacuum. The experiment was performed for different speed values, with and without differential activation. In order to verify the action of the differential, the vehicle cornering was recorded using a camera.

For the sake of completeness has to be said that the operations of data acquisition have been performed using an acquisition board (FRDM - KL25Z) communicating with the CU by mean of SPI protocol and with the computer by UART.

CHAPTER 2

SYSTEM COMPONENTS AND UTILIZED TOOLS

The system, as studied, involves both electrical and mechanical parts. This chapter will present all the components constituting the vehicle. In order to develop control of the system's parts and the software, many tools have been used, all of which are presented in this chapter.

2.1 Mechanical components

It is obvious that a vehicle is principally a mechanical system. The composition of the system usual for a normal vehicle, apart of the dimension and some components, from abnormalities steming from the vehicle's dimension. An example of this is the suspension which was not configurable in these small sizes. As such the vehicle is composed by:

- Wheels: 4 (see fig.1)
- Platform (see fig.1)
- Wheels Axis: Rear (see fig.3)
- Ackermann steering mechanism: Front (see fig.2)
- Central suspension (see fig.2)
- DC Motor: 2 (see fig.3)

Those mechanical components characterize geometric quantities defining the vehicle size. These value are presented in table I and refers to fig.5a and 5b.

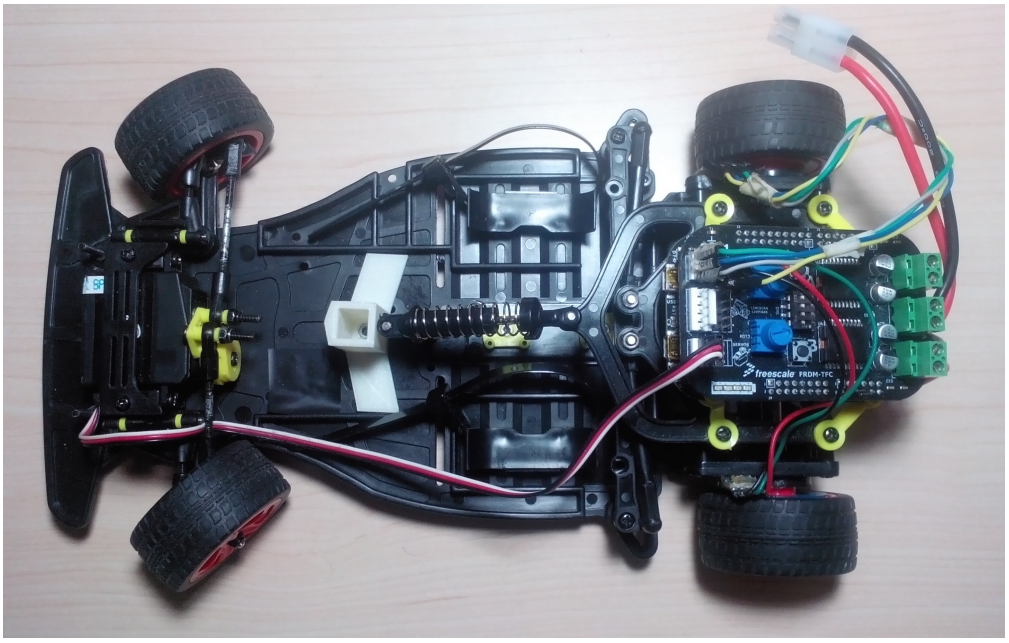


Figure 1: Superior view of the vehicle.

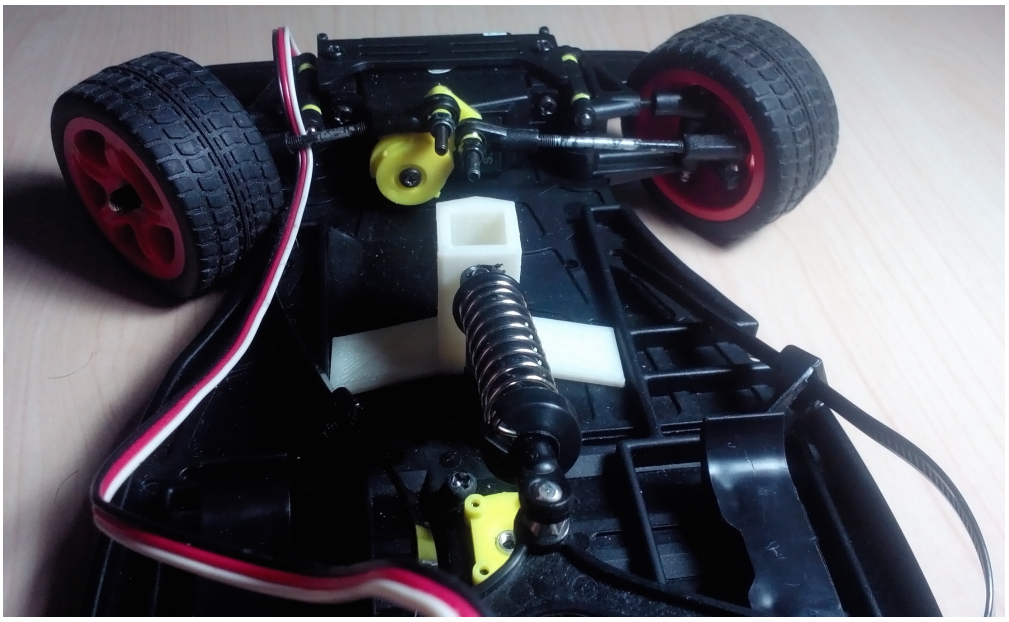


Figure 2: Front part of the vehicle.

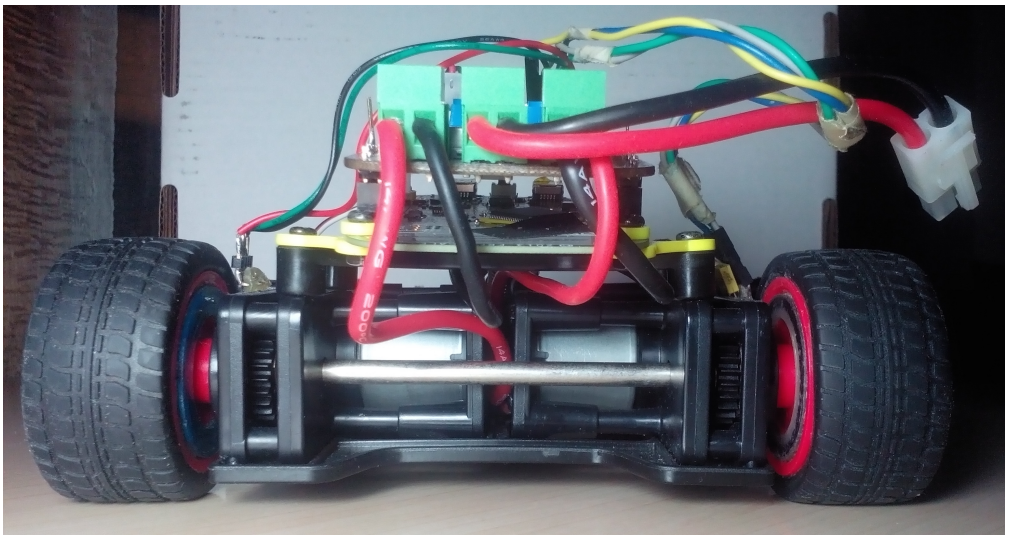


Figure 3: Back view of the vehicle.

TABLE I: DIMENSION OF GEOMETRIC CHARACTERISTICS OF THE VEHICLE.

Characteristic	Description	Value
L	Represent the distance between the rear and front wheels axis (fig.5a)	0.200 m
L_t	Total length of the vehicle (fig.5a)	0.273 m
l_w	Wheel track, is the distance between right and left wheels, also assumed as vehicle width (fig.5a)	0.137 m
l_r	Distance of center of mass from rear axis (fig.5a).	0.060 m
l_f	Distance of center of mass from front axis (fig.5a).	0.140 m
d_w	Wheel diameter (fig. 5b).	0.050 m
r_w	Wheel radius ($d_w/2$).	0.025 m
w_w	Wheel width (fig. 5b).	0.024 m

2.2 Electrical components

In order to let the system move and be controlled, an electrical actuator, micro controller board, and vision sensors are utilized. The specific components are exposed here.

- DC motor - rn250-cn (fig. 5a) : The motor characteristics are :
  - Voltage source: 7.2 V
  - No load speed: 10 000 rpm
  - Stall current: 3800 mA
  - Stall torque: 88 gcm

- Servo Motor - Futaba® S3010 (fig.5b) : This is used to set the steering angle that is imposed upon the wheels. From (2):
  - Supply voltages (4.8 V, 6.0 V)
  - Torque (5.2 kg−cm, 6.0 kg−cm)
  - Speed (0.20 sec/60°, 0.16 sec/60°

- Line Scan Camera - TSL1401 (fig.5c): The camera is used for detecting the lines on the borders of the track. See (3).
  - Sensor Element 128×1
  - Dots Per Inch (DPI) 400

  - Dynamic Range $4000:1 (72\,\text{dB})$
  - Clock Frequency from $5\,\text{kHz}$ to $8000\,\text{kHz}$

- Battery (fig.5d) : Provides power supply for the hardware components.
  - Supply voltage $7.2\,\text{V}$
  - Current $3000\,\text{mAh}$

- Freescale™ TFC Shield (fig.5e): This provides the instruments to control the actuators (motor). This owns the circuit dedicated to the motor supply, speed sensor input, servo output and more. Components, reported at (4), are:
  - Channel Motor Driver ICs (MC33887APVW): 2
  - Channel Servo Outputs: 2
  - Line Scan Camera connectors: 2
  - Input for speed sensors: 2
  - Potentiometers: 2
  - Push button: 2
  - Position DIP Switch: 4
  - LEDs: 4

- Freescale™ Freedom-Kl25z (fig. 5f): This is the core board, the MCU and all the peripherals that constitute the control unit of the system are welded on it. The components, as shown in (5), are:
  - CPU ARM Cortex M0+
  - FLASH Memory 128 KB
  - SDRAM 16 KB
  - UART: 1 low power, 2 standard
  - SPI 2×8bit
  - I2C 2
  - PWM 1×3 ch
  - PWM 2×2 ch
  - SAR ADC 16 bit
  - DAC 12 bit

These components identify the electrical and electronic part of the system. In order to also utilize those and interact with the software that has been developed it is important to introduce the software tools that allows this.

2.3 Software Tools

In order to produce the control algorithm and analyze the system for controller synthesis and generate a prototype, many computational libraries, tools and software are utilized. The tools which have been used are discussed below. The main utilized software has been MATLAB® by MathWorks®. This software is a powerful instrument

that provides various tools which help with computational work in engineering and many other fields. MATLAB®'s engineering related software related to control system area in particular, has been a frequent help. In addition, many hardware-related tools has been utilized as well. These utilities are:

- **Simulink®** is one of the most famous tool provided by the MATLAB® suite. It is a model-based interface with a huge library that help in simulation, control system design, code generation and more.

- **Embedded Coder®** a tool that automatically generates C or C++ code for a specific hardware. This program introduces new configurations and optimization for MATLAB Coder™ and Simulink Coder™ that allow a better integration with the hardware. In addition, several support packages are available for specific hardware.

- **Freescale™ FRDM-KL25Z Microcontroller Support from Embedded Coder®** provides a library and drivers for many devices related with the used board and the shield (*FRDM-KL25Z* and *TFC Shield*). The supported peripherals, as reported in (6), are:

  - Board components:
    - digital I/O: 14
    - analog inputs: 6

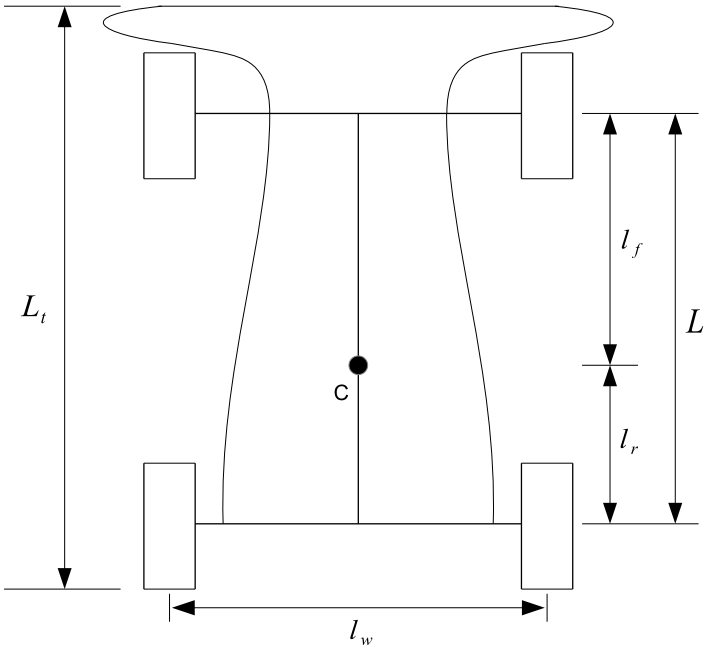
    * analog outputs: 1
    * serials Tx/Rx from UARTs: 3
    * RGB LED: 3

  - Shield components:
    * line scan cameras: 2
    * dip-switch: 1
    * potentiometers: 2
    * push buttons: 2
    * servo motors: 2
    * DC motors: 2
    * RGB LED: 4

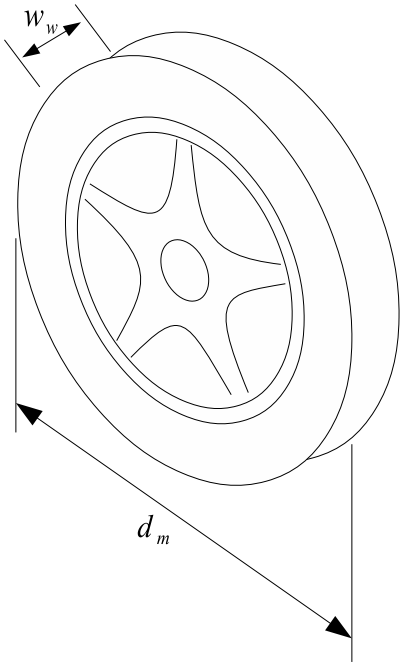
- **Serial port terminal CoolTerm** is a simple application that allows input data to be read from serial. It has been utilized mainly for acquisition.

The exposed elements are the component constituting the vehicle and software utilized for studying the system and are necessary for the realization of the control algorithm. These are the instruments and the set up available from a wide selection of options. The description of the algorithm and design that utilize these components follows next

Figure 4: Vehicle dimensions.



(a) *Geometry of the vehicle.*



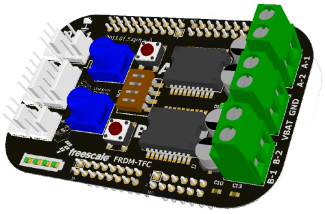
(b) *Wheel representation.*

(a) *DC motor image.*

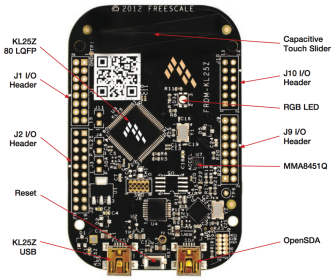
(b) *Servo Motor used for steering.*

(c) *Line Scan Camera module.*

(d) *Battery Pack.*



(e) *TFC Shield.*



(f) *FRDM - kl25z board.*

Figure 5: Electrical components.

CHAPTER 3

LATERAL BEHAVIOR

In vehicular motion, cornering is critical, with a poor twin causing instability and unsafe conditions. Consider a generic curve and a car approximated as a point. Assume, in addition, that the vehicle is traveling with constant speed, and is perfectly centered in the road as in fig.6. With these assumptions, the point has the same distance from the rotation center for the whole trajectory and a constant angular speed rotation is obviously derivable.

$$\dot{\psi} = \frac{V}{R} \tag{3.1}$$

Now consider a car whose rear wheels and axis aren't the mean point coincident with the previous distance R but which shares the same angular speed $\dot{\psi}$. Consider the two wheels aligned on the radius of the curve (see figure 7). It is evident that without relative rotation speeds of the wheel who respect to the axis center, the tangential speeds of the wheels will be different, changing the distance from the rotation center, so:

$$V_1 = \dot{\psi} \cdot R_1 \quad V_2 = \dot{\psi} \cdot R_2. \tag{3.2}$$

Those simple relations study simply the rear wheels. In addition to this has to be considered the relations extant in a steering vehicle, that includes the front wheels,

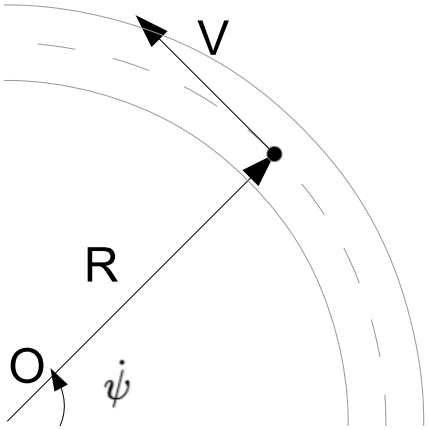


Figure 6: Tangential speed rotatory motion of car as a point.

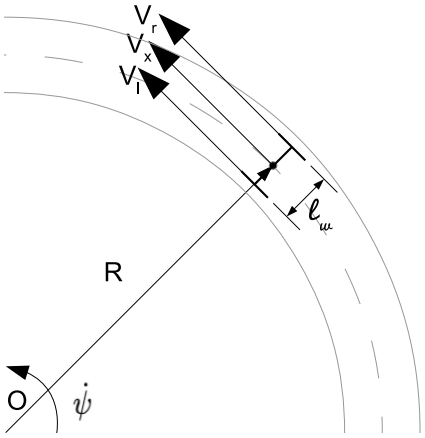


Figure 7: Rear wheels in cornering.

which, in the case of this study, are responsible for steering. These relations constitute the kinematics of the vehicle.

3.1 Kinematics

Kinematics of vehicles has been studied by many authors, one of the most common of which is the *Ackermann* steering kinematics (7). Consider a vehicle with only front steering wheels and a trapezoidal tire rod (fig. 8). The steering angle provided are quite different for right and left wheel, so the steering angle should be considered as the average of the two:

$$\delta = \frac{\delta_r + \delta_l}{2} \tag{3.3}$$

Thanks to the geometry (shown in figure 8) it is possible to derive the kinematics of the vehicle. Form the trigonometry result:

$$\frac{L}{R\cos\beta} = \tan\delta \tag{3.4}$$

$$\frac{l_r}{R\cos\beta} = \tan\beta \tag{3.5}$$

From equations 3.4 and 3.5 follows:

$$\frac{L}{\tan\delta} = R\cos\beta \tag{3.6}$$

$$\frac{l_r}{\tan\beta} = R\cos\beta \tag{3.7}$$

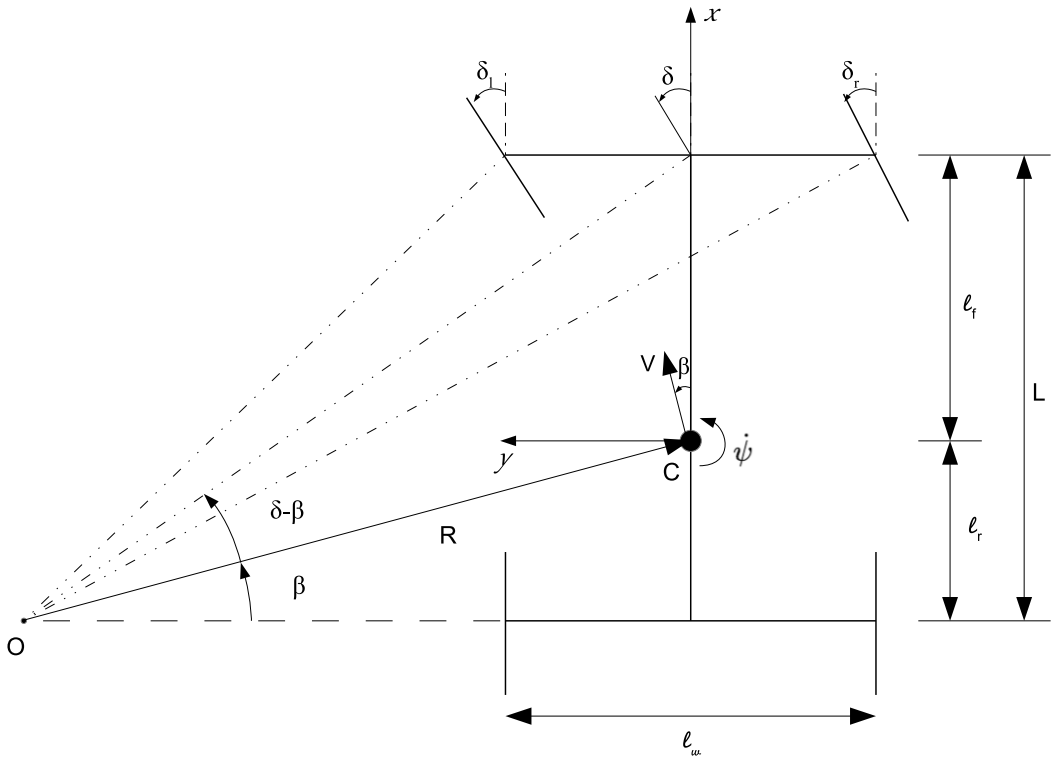


Figure 8: Ackermann geometric model.

Putting equal equation 3.6 with 3.7 result:

$$\frac{L}{\tan\delta} = \frac{l_r}{\tan\beta} \tag{3.8}$$

and obviously :

$$\tan\beta = \frac{l_r}{L}\tan\delta \tag{3.9}$$

From equation 3.9 finally follows:

$$\beta = \arctan\left(\frac{l_r}{L}\tan\delta\right) \tag{3.10}$$

Equation 3.10 obtains the slip angle β, that defines the orientation of speed, with respect of the longitudinal direction of the vehicle. In ideal conditions this angle is equal to zero, but in reality this value is higher and could cause instability in cornering. From equation 3.4 it is possible to define the rotation radius R

$$R = \frac{L}{\cos\beta\tan\delta} \tag{3.11}$$

Now, from figure 8 and the equations 3.10 and 3.11, it is possible to find the three speeds that characterize the lateral kinematics of the vehicle:

$\dot{x}$ Longitudinal speed

$\dot{y}$ Lateral speed

$\dot{\psi}$ Yaw rate

Longitudinal and lateral speed could be simply derived by the geometry and result:

$$\dot{x} = V\cos(\beta + \psi) = V\cos\left[\arctan\left(\frac{l_r}{L}\tan\delta\right) + \psi\right] \tag{3.12}$$

$$\dot{y} = V\sin(\beta + \psi) = V\sin\left[\arctan\left(\frac{l_r}{L}\tan\delta\right) + \psi\right] \tag{3.13}$$

The yaw rate derive from equation 3.1 substituing 3.11:

$$\dot{\psi} = \frac{V}{L}\tan\delta\cos\beta = \frac{V}{L}\tan\delta\cos\left[\arctan\left(\frac{l_r}{L}\tan\delta\right)\right] \tag{3.14}$$

In circular uniform motion condition the speed $\dot{y}$ is null and also the slip angle β, but in real motion slip angle and lateral speed could be present causing understeering or oversteering. That happens because of the dynamic of the vehicle that is influenced by many factors included tyre slip, inertia and engine actuation. The previous study defined the kinematic equations considering the speed V without taking into account the wheel's speeds. Since in the case of study both rear wheels are independently actuated, it is important to define the speed of each wheel.

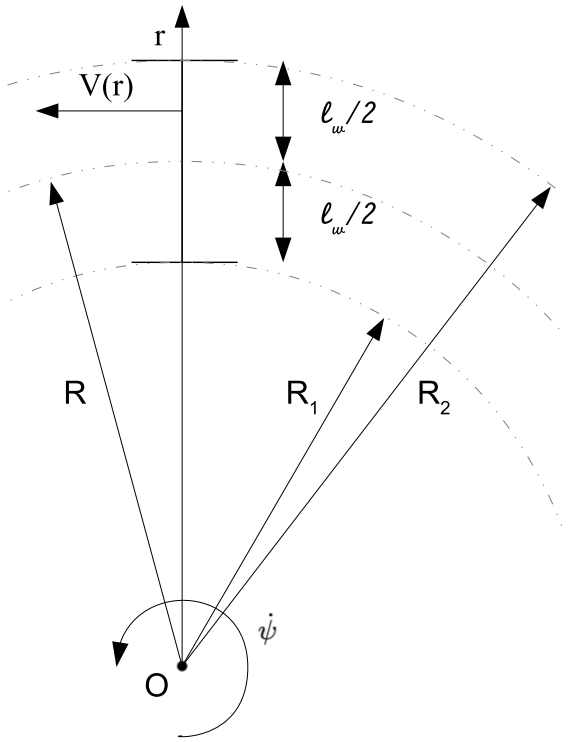


Figure 9: Speed radius dependance.

3.2 Speed distribution on rear wheels

Focusing on rear axis, consider that longitudinal speed $\dot{x} = V_x$ is directed perpendicular with the axis and is present on it at the middle point. Since, as seen before, the tangential speed is dependent from yaw rate and rotational radius as already discussed, the result is that the speed of each wheel is different. First of all consider the speed distribution on the axis (figure 9), from 3.2 follows:

$$V(r) = \dot{\psi} \cdot r \tag{3.15}$$

So the speeds result:

$$V_x = V(R) = \dot{\psi} \cdot R \quad V_l = V(R_1) = \dot{\psi} \cdot R_1 \quad V_r = V(R_2) = \dot{\psi} \cdot R_2 \tag{3.16}$$

From figure 10 is possible to find that:

$$R_1 = R - \frac{l_w}{2} \quad R_2 = R + \frac{l_w}{2} \tag{3.17}$$

For that result substituting 3.17 in 3.16

$$V_l = \dot{\psi} \cdot \left(R - \frac{l_w}{2}\right) \quad V_r = \dot{\psi} \cdot \left(R + \frac{l_w}{2}\right) \tag{3.18}$$

and expanding:

$$V_l = \dot{\psi}R - \dot{\psi}\frac{l_w}{2} \quad V_r = \dot{\psi}R + \dot{\psi}\frac{l_w}{2} \tag{3.19}$$

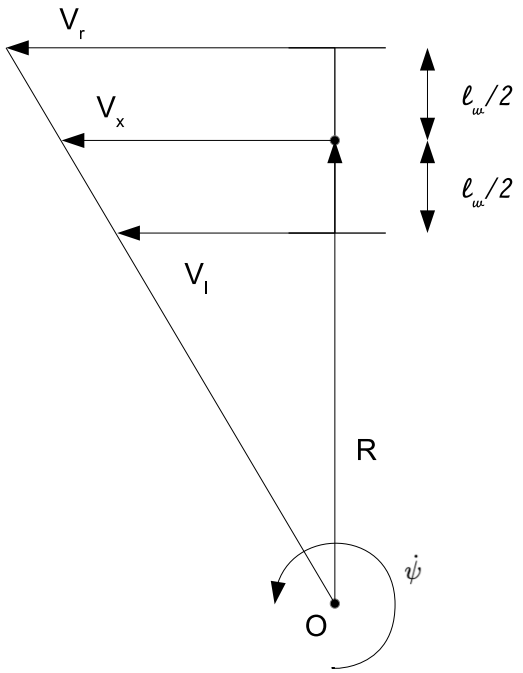


Figure 10: Speed distribution.

From that result substituting 3.16 in 3.19

$$V_l = V_x - \dot{\psi}\frac{l_w}{2} \quad V_r = V_x + \dot{\psi}\frac{l_w}{2} \tag{3.20}$$

It follows that having speed a triangular distribution, the wheels being equidistant from the middle of the axis, V_x is also the mean between right and left wheels speed:

$$V_x = \frac{V_r + V_l}{2} \tag{3.21}$$

Equations 3.12, 3.13, 3.14 with 3.20, 3.21 constitute the set of equations that define the kinematic of the vehicle with independently actuated wheels. From these facts, a set of equations that constitute the algorithm used in order to define wheels' speed in steering conditions is derived.

3.3 Inverse Kinematic: equations for differential

From the kinematic equations it is possible to find inverse relations that allow for the evaluation of the speed of both wheels. These speeds will constitute the reference for speed control of rear wheels and are computed according to Ackermann model in order to provide ideal cornering. Before inverse kinematics equations could be proper a brief description of *mechanical differential* in order to understand the working principle.

3.3.1 Mechanical Differential

The classic differential is a mechanical gear that allows wheels adequately distribute speed to the value that is needed to perform corner with a fixed angular speed. Torque distribution is provided by the engine in equal part.

When a vehicle is steering could be one of the two conditions:

- **Locking**: inner and outer wheels are obliged to turn with the same speed. Due to the difference of required speed during a turn, wheels start sliding causing a dangerous loss of traction. In that way the trajectory is not properly followed and, under high speeds, it may not be possible to turn at all. In that condition torque is distributed not equally but to satisfy the requirements.
- **Open**: inner and outer wheels adequate their speed in order to have the chosen vehicle speed. Outer wheel will turn with higher speed than inner one, avoiding

sliding that are cause of instability in corners. Torque distributed equally as half of the provided by the engine.

Over the years, many other methods which complement the *open wheels* differential has been posited. Some of these are :

- **Auto-Locking**: this differential combine the *open wheels* and *locking wheels* automatically, in that way in lack of traction conditions it is possible to provide more torque to the wheel that have more traction.
- **TorSen**: this differential has some additional gears that that allows for the assignment of specific amount of torque to the wheel with more traction, and less torque to the slipping wheel. In normal traction conditions, this works as an open differential. Speeds are the same as in the open differential.
- **Active**: Active differentials use yaw rate sensors and electronic control unit in order to give torque to the wheels reducing understeering effects.

3.3.2 Inverse Kinematic

From the kinematic equations that define wheels speed are derived. Considering that we have defined V_x, δ, L, l_f, l_r, l_w, from equations 3.10, 3.11, 3.20, it follows the inverse kinematic equations:

$$\begin{aligned}
\beta &= \arctan\left(\frac{l_r}{L}\tan\delta\right) \\
R &= \frac{L}{\cos\beta\tan\delta} \\
\Delta V &= \dot{\psi}\frac{l_w}{2} \\
V_r &= V_x + \Delta V \\
V_l &= V_x - \Delta V
\end{aligned} \tag{3.22}$$

Considering equation 3.14, eq. 3.22 become :

$$\Delta V = \frac{V}{L}\tan(\delta)\cos(\beta)\frac{l_w}{2} \tag{3.23}$$

Being $V_x = V\cos(\beta)$ the complete equations result as:

$$\begin{aligned}
\beta &= \arctan\left(\frac{l_r}{L}\tan\delta\right) \\
R &= \frac{L}{\cos\beta\tan\delta} \\
\Delta V &= \frac{V_x}{L}\tan(\delta)\frac{l_w}{2} \\
V_r &= V_x + \Delta V \\
V_l &= V_x - \Delta V
\end{aligned}$$

3.3.3 Differential algorithm

In order to produce an algorithm that could be used in a MCU, inverse kinematics equations should be linearized and simplified. In that case trigonometric function will be eliminated with small angle approximation for linearizing equations. Considering angles δ and β the result is that:

$$\cos(\beta) \approx 1$$

$$\sin(\beta) \approx \beta$$

$$\tan(\beta) \approx \beta$$

$$\cos(\delta) \approx 1$$

$$\sin(\delta) \approx \delta$$

$$\tan(\delta) \approx \delta$$

Now follows a simplified equations set that constitute the algorithm used to compute the wheels speed:

$$R' = \frac{L}{\delta} \tag{3.24}$$

$$\Delta V' = \frac{V_x}{L}\delta\frac{l_w}{2} \tag{3.25}$$

$$V_r = V_x + \Delta V' \tag{3.26}$$

$$V_l = V_x - \Delta V' \tag{3.27}$$

CHAPTER 4

MODEL-BASED CUSTOM DRIVERS

The second important point of this thesis work is the method used in order to develop the control system: the *Model-Based Design*. As the name suggests it is a method that focus on the model of the system and that model is visualized thanks to block and other features that help to simulate, analyze and study a system. There are many software platforms that are model-based, many of those are widely used in industry thanks to their ease of use and quick response time. Model-Based design is an efficient approach for communicating among the design stages following the "V" diagram that represent the development cycle. The four steps that characterize model-based design are:

1. Plant modeling. Strives towards the realization of a mathematical model that describe the physical system to be controlled.
2. Controller design. Once the model is known, one is able to create the dynamic characteristic from which the controller is synthesized.
3. Simulation. The system and the controller are tested in real time or offline. Simulation acts as a final test of system specifications and requirements. Errors can be located and rectified.
4. Deployment of the controller. The controller found in simulation has to be converted and integrated. This operation could be performed with automatic code generator. It is important to note that the controller developed in simulation

never works exactly as simulated, so the controller must be updated to match the required specifications.

In this thesis work the four steps has been employed to develop the motor controller but another important topic related to the model-based approach has been the automatic code generation. The controller for the vehicle handling has been designed by model based visual approach and thanks to the automatic code generation, code has been built and compiled. The software utilized for design is Simulink®. The automatic code generation is provided by Embedded Coder® a feature that adds functionalities to MATLAB® Coder™ and Simulink® Coder™, in order to produce efficient and optimized embeddable code for the specific target embedded system as constructed. The usage of Embedded Coder® Support Package lead to the inclusion of several peripherals of the board, a feature which necessitated the search of further drivers. Specifically a speed sensor driver is not present. A speed sensor driver is necessary to realize a "block" that with the Embedded Coder® realize specific code for the hardware. Simulink® has two tool to accomplish this: S-Function and Target Language Compiler.

4.1 S-Functions

S-function fills a distinct gap in the Simulink system. *S-function* provides a language to describe a Simulink block, and provides a way to execute this from his engine. The S-Functions are mainly used to expand capabilities of Simulink, adding blocks designed for specific functions. In this case, S-Function is used to realize drivers in order to interface with the target hardware. There are five types of S-Function implementation:

- **Level-1 MATLAB S-function** is simple means of connect MATLAB and S-Function API.
- **Level-2 MATLAB S-function** is more powerful method than Level-1, and allows a wider range of S-function API and code generation.
- **Handwritten C-MEX S-function** is the more flexible method to develop an S-Function. It is written in C,C++ or Fortran, and is composed by a set of callbacks (with a specific name) that contains instructions to be executed during the simulation by the Simulink Engine. So an algorithm could be written directly as C-MEX S-Function or with a wrapper file calling existing functions. In order to inline code an additional file called *TLC* is required.
- **The S-function Builder** is a visual interface that help in creating S-Functions, although it has less capabilities thab handwritten C-MEX S-Functions (for example it supports C or C++ language only). At the same time, it avoids to use S-Function API, instead create wrapper file and TLC file automatically, if it is desired.
- **The Legacy Code Tool** is a tool that use a set of MATLAB instructions and help to realize S-Function when coupled with legacy code. For inlining code the TLC file could be generated automatically. This method is less powerful than the previous two C-MEX methods.

Among all the possible methods the *Handwritten C-MEX* has been chosen. The reason of the decision is mainly its flexibility and the option to inline code in C language. This gives a wide range of possible decision, allowing for a deal of freedom in programming

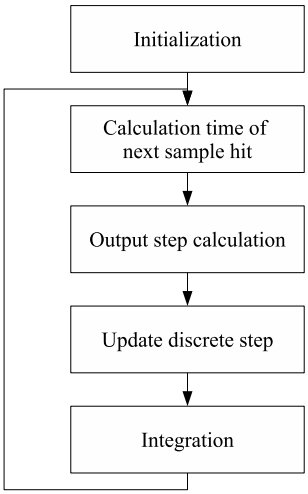


Figure 11: Simulation loop.

choice while still allowing for the code to be inlined similarly as if directly programmed MCU. As said before an S-Function contains a set of instruction that the Simulink Engine executes in a certain order in the Simulation. In order to develop the S-Function the programmer must know *callback* and how these work.

4.1.1 Simulink Engine

Simulation causes the engine to execute many stages in a loop (fig.11), where each stage provides the input for the next. These stages are:

- **Initialization**: that precede first simulation loop and prepare the variables used in the simulations.
- **Calculation time of next sample hit**: if the time step size is variable, it calculates step size for the next sample hit.

- **Output states calculation in major step time**: compute output states at the current time step of the simulated system, from previous loop execution states value.
- **Update discrete states in major step time**: it update states for the next loop execution.
- **Integration in minor step time**: performed in continuous states simulations, executes subtasks that compute derivatives, output and zero-cross detection. In practice, this stage computes states.

First, the initialization is executed. "In this phase, the Simulink engine incorporates library blocks into the model, propagates signal widths, data types, and sample times, evaluates block parameters, determines block execution order, and allocates memory." (8). Then the engine enters into the loop steps. At each stage of the loop blocks are executed in the order chosen in the initialization phase. Some tasks are only executed depending on the simulation settings, but generally output, derivatives and states are computed by the engine at each simulation time.

4.1.2 Callbacks

Looking at a lower level each step of the simulation is defined by callback. Callback functions are used by the engine to interpret at which step of the simulation a blocks' specific operation has to be performed. Simplifying the concept: callbacks constitute the code representation of block specific loop steps. This definition, however, is not exactly proper, as one to one link between simulation step and callback cannot be established.

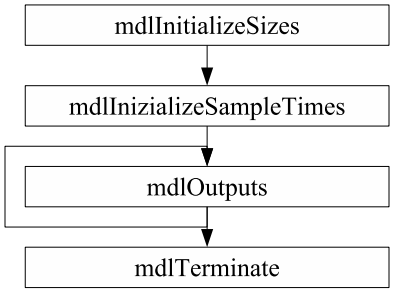


Figure 12: Essential callbacks.

Each step has one mandatory callback, but many more are possible at each stage. Others are optional and provide additional functions. Mandatory callbacks are:

- **mdlInizializeSizes**: set up input and output ports, parameters, work vector and other important component.
- **mdlInitializeSampleTimes**: set up the sample time.
- **mdlOutputs**: calculate the output at the current time step.
- **mdlTerminate**: perform the operations needed for terminating the simulation.

These callbacks must be implemented, and don't correspond to each step of the simulation. Some stages are not required all the time (see dotted rectangles in fig.13). The loop is begun by firstly calling *mdlInitializeSizes* and *mdlInitializeSampleTimes*, which configure input and output ports, parameters, sample times and other attributes. *MdlOutputs* follows this, calculating outputs of the S-function, at the end of the simulation, *mdlTerminate* executes functions for terminating the simulation.

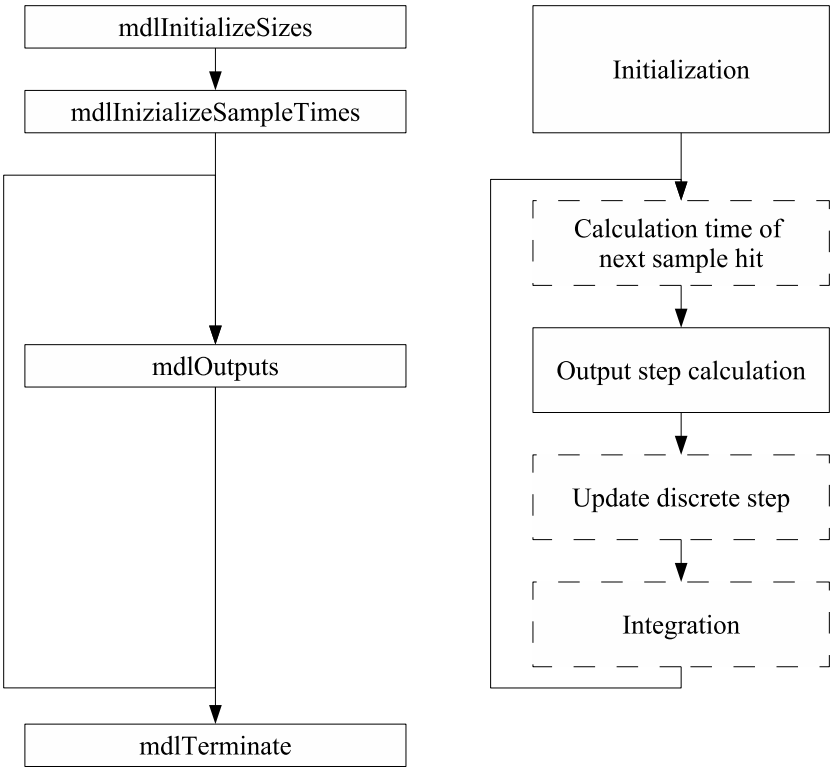


Figure 13: Callbacks and stages correspondance.

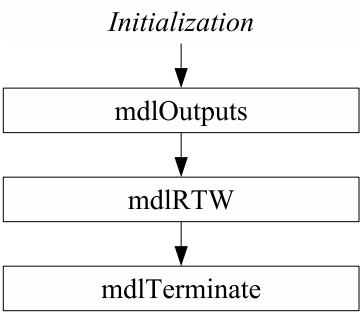


Figure 14: Callbacks in code generation.

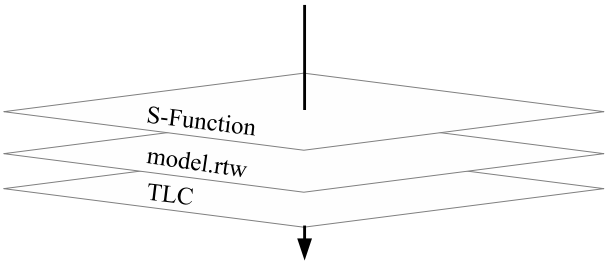


Figure 15: Code generation levels.

This is the simulation loop, and not all additional callbacks are required in code generation (fig.14). Callbacks are called in the same order but in this case another method is required: *mdlRTW*. This method is called in order to generate the *model.rtw* file. This file contains information about the model and the components that constitute it, and acts as an intermediary between the C-MEX S-Function and the TLC (figure 15),.

4.2 Target Language Compiler

TLC (Target Language Compiler) is a tool offered by Simulink® Coder™. This tool generates code, for the purpose of improving performance and execution speed, realizing specific target code, and modifing the length of the produced code eventually include existing code. The Target Language Compiler use TLC files, one for block (if present for a specific block) that contains instruction and functions for the blocks operations and TLC files with parameter informations and header specified that are model-wide. TLC files are used to *inline* the S-Funcion in order to have the customization required for

the application, but if specific code is not required the code could be generated without TLC files, meaning *noninlined* code.

4.2.1 Inlined and Noninlined S-Functions

Before explaining the process by which Target Language Compiler is used to generate code, it is proper to distinguish case in which case the TLC files should be used. It has been established that TLC files are used in order to inline S-Function, so it is important to distinguish *Inlined S-Funcion* and *Noninlined-S-Function*.

- **Inlined S-Function**: an S-Function is *Inlined* if, after the code generation process, the related code is customized and optimized for some specific goal, for example to increase execution performance and reduce memory usage. A TLC file is necessary to do this.
- **Noninlined S-Function**: an S-Funcion is *Noninlined* if the code generation is performed without optimization of the produced code, instead directly using the S-Function API that provides functions and other overhead support. This approach doesn't require the usage of TLC files.

Talking of *Inlined S-Function* we can distinguish two type of *inlining* methods:

- Fully Inlined S-Function
- Wrapped Inlined S-Function

Both types of inlining imply optimizations in performances and memory usage, and both reduce the overhead introduced with noninlined S-Functions. The difference between the

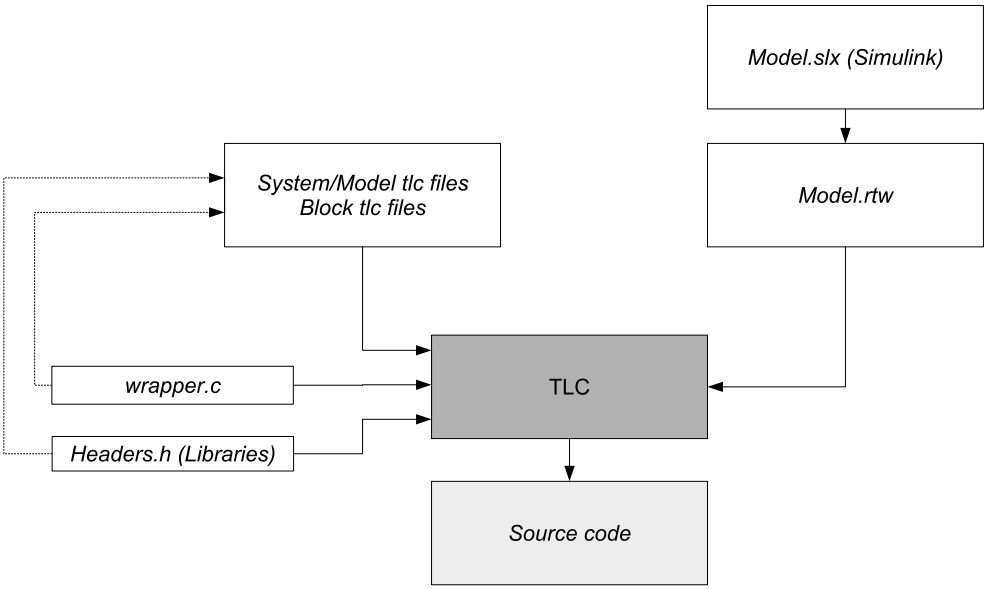


Figure 16: TLC code generation.

two methods lies in the use of TLC files. Fully Inlined S-Functions are those in which the TLC file contains all the related code. Wrapper Inlined on the other hand has another file called “wrapper” with “.c” extension, in addition to the TLC files. This file contains C code algorithm already present in the program and the TLC file. This allows for a shorter model code, as code that is called by multiple blocks or the same block multiple times can be referenced directly to the wrapper file, rather than be repeated for multiple iterations.

4.2.2 TLC process

The process by which the TLC generates code is simple. From the simulink model, a model.rtw file is generated . This file contains the information of the model and is

used by the TLC with TLC files to generate the source code, and a file used to build the application. As stated before, there are two type of TLC files:

- **System/model TLC files**: contains information about the model and specifications for code generation.
- **Block TLC files**: provide algorithm to embed for the specific block.

In the process, System TLC file is called first and prepares the code generation using the information provided by *model.rtw* file as well as the settings specified inside the TLC system file. After that, the Target Language Compiler utilizes the TLC file for the inlined S-Functions and interpret code from non-inlined S-Functions. Finally, the code may be generated and ready to be used.

4.2.3 Block Methods

Although the process is simple, it is important to understand how TLC files interact with the Target Language Compiler, communicating which code to use and when. These operations are executed by *block methods* that are the analogous of S-Function's Callbacks for TLC. These functions called by the model-wide TLC file during code generation. There are functions that are required and functions that are optional as seen for callbacks. The following functions are required:

- **BlockTypeSetup**: this function is executed once for block type present in the model.
- **Output**: this function is the main algorithm of the block, and generates the output of the block. It is placed in *mdlOutputs*.

These functions are called by the model-wide TLC. This file calls functions and places the code inside functions which are then integrated in the generated code. Additional useful functions are:

- **BlockInstanceSetup**: this function is executed for each block type present in the model.
- **Start**: in this function the initialization code that has to be executed at the start up is placed, and represents *mdlStart*.

These functions constitute the core of the driver; in other words, it constitutes the future generated code.

4.2.4 TLC file Structure

The utilized functions have been described, but there are other essential parts in the tlc files that are necessary for the system to work. TLC provides many capabilities consonant with the traditional capabilities of a programming language. Some additional, brief specifications could be useful to understand the use of tlc files.

4.2.4.1 Directives

Directives are identified with "`%`" and could be different. Main directives are:

```
%assign
```

That directive create a variable in TLC language.

```
%if expr
%elseif expr
```

```
%else
%endif
```

This is equivalent to the "`if`" in C language.

```
%<expr>
```

This expands an expression or a variable value; when executed, it uses what is contained inside the variable or expression.

```
%function namefunction(block, system)
%end
```

It is possible to also create a function used in TLC language, and which could, in turn, be called by the block specified inside the TLC file.

```
%%
```

This represents the comment; everything after it is considered commented and is ignored.

There are more directives, but these are the most utilized. It is important to understand that directives expand the capabilities of code customization. Only directives are considered by the TLC, what is outside the directive or comment symbol is reported in the code as it is written. Obviously the code that resides in functions area, or conditional on an if statement, is used only if the condition is satisfied or not, depending on where it has been placed. For example:

```
%if %<a> == 1
```

```
CODEA

%else

CODEB

%endif
```

If condition $a = 1$ is satisfied *CODEA* is reported in the generated code. If $a \neq, 1$ *CODEB* is written.

4.2.4.2 Structure

In order to inline an S-Function, the whole structure of the TLC file is simply:

1. **File name and language definition**: in oder to specify that, the directive `%implements` is used the following way: `%implement "filename" "C"`
2. **Functions**: Here follows the methods specified, in order:
   - `%function BlockTypeSetup(block, system)`
   - `%function BlockInstanceSetup(block, system)`: (optional)
   - `%function Start(block, system)`: (optional)
   - `%function Outputs(block, system)`

The resulting structure is quite simple. Phase "1" is mandatory and the name of the file must be the same as the S-Function file. It is important to remember that all methods of phase "2" are not necessary. It is useful to say that in *BlockTypeSetup* and *BlockInstanceSetup* methods, in general, include libraries, global variables, function prototype depending on the capability that these are required to to satisfy. There are

in addition TLC functions that could be used to include files, libraries, add prototypes, access S-Function parameter information and other useful possibilities.

Some consideration about S-Functions and Target Language Compiler. It is important to understand that in code generation, if an S-Function is inlined, the code written in TLC is presented in generated code. So, for example, an S-Function could be empty but could generate code if TLC file is present, but could not perform in a simulation as S-Function code used in Simulink Engine for simulations. Although S-Function must have specified port numbers and parameters, it must access *model.rtw* file that is generated by the Simulink Engine in order to generate a working TLC file. Moreover S-Function API and TLC provide a wide array of functions and capabilities, so the method and order of declaring, including and specifying functions, variables, and libraries are not unique.

This chapter is a brief introduction to the argument of model based design, giving basic information about the techniques and processes utilized. In order to better understand and produce personal custom drivers, sources (8) and (9) are raccomended.

CHAPTER 5

SPEED SENSOR

One of the main objectives of this work has been the development of a Model-Based Speed sensor. As said this constitute the starting point of the DC Motor feedback control since it allows for the measurement of the speed each loop-time instant. In order to implement the speed sensor, many trials were conducted using different conceptual approaches. First, a polling method has attempted. This method was found to record improper results, as it required a high working frequency in order to give a reliable measurement; this requirement is not feasible, as is explained below. Next, an event based method has been attempted. Following this method, hardware interrupts and timers were necessary providing, so custom drivers that provides direct access to the board hardware peripherals were produced. Before going through the development of the code, an understanding of the working principle of the speed sensor on the two main approaches used to create the speed sensor is necessary.

5.1 Speed sensor set up

The components that constitute the sensing equipment consist of: a *phonic wheel* and *Hall sensor*. These interact via a digital signal that is recognized and used by the processor in order to measure speed.

TABLE II: PHONIC WHEEL CHARACTERISTICS

Element	Value
Magnets	48
N - pole	24
S - pole	24
N - N angle (S - S)	15°
Diameter (external)	3.6 cm

5.1.1 Phonic Wheel

The phonic wheel is a ring shaped object whose structure of elements is designed to produce a high and low state. These elements could be, for example, slots or magnets, if the sensor is based on light or hall sensor. This of study utilized a hall based sensor, with a phonic wheel composed by magnets with alternate polarity N - S (NORTH - SOUTH). The phonic wheel is composed by two parts that constitute the external chase. The internal half contains slots for magnets, the external is the cover and is fixed in order to fix magnets in their place. The whole phonic wheel is fixed inside the vehicle wheel. There are 48 magnets, half (24) have an exposed N pole and half (24) an exposed S pole. It is important to understand that the distance of each pole from the previous one is fixed. In table II are presented some specification of the phonic wheel.

5.1.2 Hall-Sensor

The Hall-Sensor is a component that, exploits the Hall effect: a voltage drop occurs when a magnetic field variation is detected. This experiment the sensor produces a

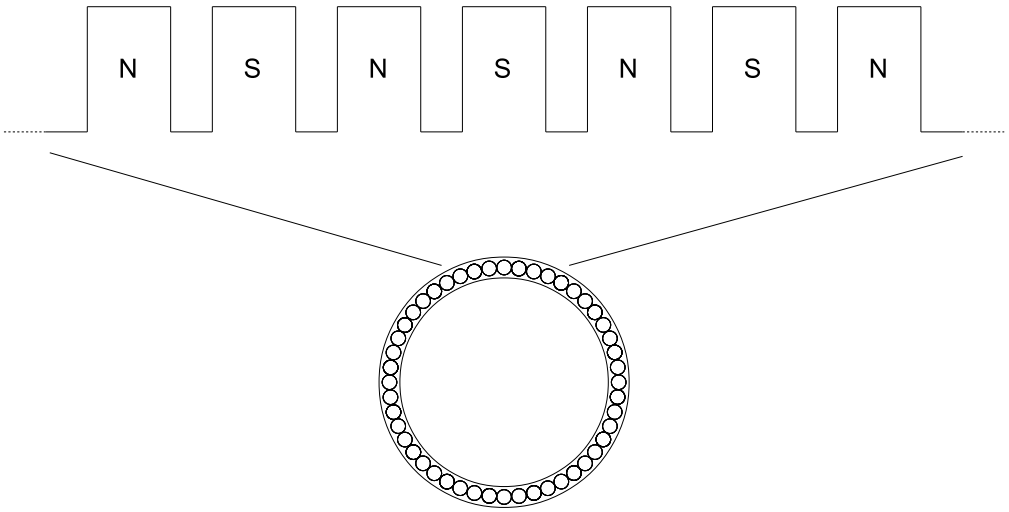


Figure 17: Representation of phonic wheel linear magnets disposition.

TABLE III: HALL SENSOR SPECIFICATIONS

Characteristic	Value
Supply VoltageV_{CC}	$3.3\,\mathrm{V} - 18\,\mathrm{V}$
Low Output Voltage$V_{OUTPUT(ON)}$	$160\,\mathrm{mV} - 500\,\mathrm{mV}$
Output Leakage Current$I_{OUTPUT(OFF)}$	$< 1\,\mu\mathrm{A} - 10\,\mu\mathrm{A}$
Supply Current $I_{CC(OFF)}$	$3.5\,\mathrm{mA} - 6.0\,\mathrm{mA}$
Supply Current $I_{CC(ON)}$	$4.5\,\mathrm{mA} - 6.0\,\mathrm{mA}$

voltage drop of $3.3V$, if a magnet with N polarity perturbs the field and $0V$ with S polarity influence. The component used integrates the two sensor inside the same silicon, but only one is used. The other hall effect sensor is the quadrature sensor that detects the rotation verse of the wheel.

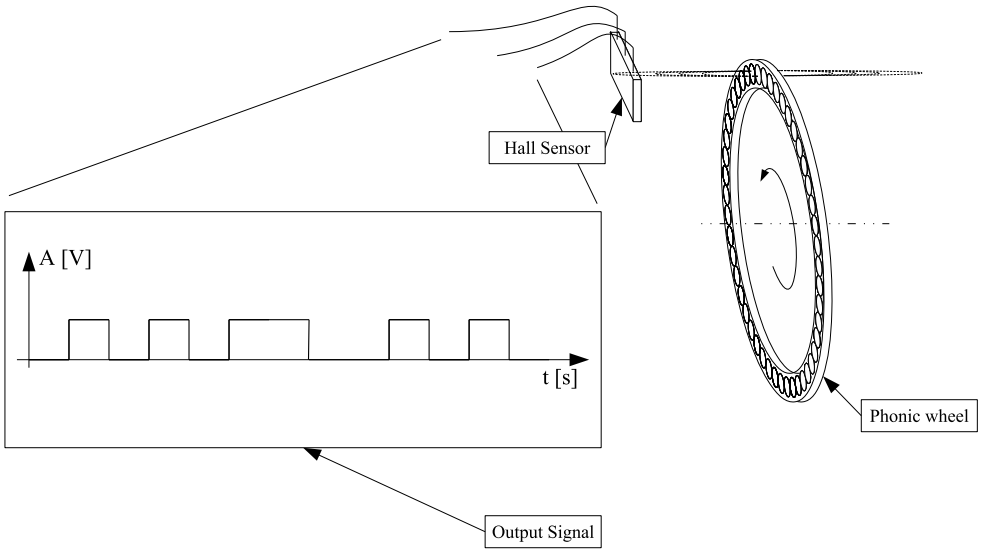


Figure 18: Functional sequence of the signal generation process.

5.1.3 Working principle

The composition of the measure system has been seen. It is possible now to analyze the working principle, that allows for a measurement of speed. As said, the phonic wheel has 48 magnets. These are half with N pole exposed and the other half with the south pole exposed. N-poles cause high state output from the Hall-sensor and S-poles low level state. So alternating N and S pole, the Hall-sensor produces a square wave, as in figure 19. This period is related to the time between the passage of two poles of the same type. In order to measure speed, the board receives this square wave and detects the number of passed magnets and time period. In this work the embedded software detects rising edges of the square wave, while a timer measures the time of the magnets passage; since

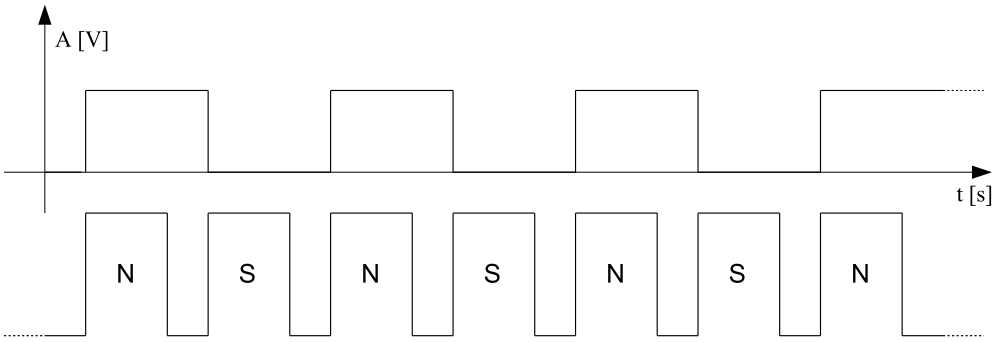


Figure 19: Relation between magnet pole and wave status.

one magnet has fixed distance from the next, it measures speed as the ratio of the space and time interval.

Knowing the characteristic of vehicle and phonic wheels, the arc of circumference resulting from wheel radius r_w and angle between magnets α_m that arc of circumference is $d_m = 6.54\,\text{mm}$. So in general:

$$v = \frac{n_m \cdot d_m}{\Delta t} \quad \text{with } n_m = \text{ number of magnets} \tag{5.1}$$

That is a general overview of the sensor working principle and how hardware generates the input for the software. It is now possible to analyze the software implementation.

5.2 Speed sensor implementations

The speed sensor could be software implemented in many ways resulting in differing in modules, module utilization, data management and algorithms. In this work two methods of implementation, differing in data and algorithm utilized, have been examined.

As the module utilized under both methods is the same, the details of the algorithm utilized, should be examined first. The module used is the timer.

5.2.1 Timer

The used module is the TPM a Timer/PWM Module. It has a Module Counter that in this application has been set in free running mode, up counting. When the counter reaches MOD value (module value) an overflow occurs, and the counter is configured to restart counting up from zero. Each overflow triggers an interrupt request.

The counter could be clocked using different inputs which may be pre-scaled, such as the BUS clock. In this case a pre-scaled clock has not been used. Channels of the TPM are in parallel and one of two channels of the TPM module (TPM1) are utilized. The channels could receive inputs from external sources, and the TPM1 module has been configured to receive input from the hall sensor. The *input capture mode logic* has been set on the rising edge input capture which allows for the triggering of the channel's interrupt. Moreover, when the input event is present, the channel value register (CnV with n = number of the channel) locally copies the value of the counter, and this information is used, in turn, to find the time values. This exposes only the utilized configurations of the TPM module; in order to have a better knowledge of it and the other peripherals of the board see (5)

In order to measure speed, a time interval and a certain number of magnets that are related to the interval are required. The number of magnets is determined by the data handling of the input used. As said, the speed sensor implementations are different in data input handling. These are: *Fix Window* and *Sliding Window*.

5.2.2 TLC interaction

Let's examine in detail how the speed sensor is implemented in a TLC file, focusing on the requirement of the method and how these are used. It could be useful to analyze the code, step by step.

```
%% File : SpeedSensor.tlc
%%
%implements  SpeedSensor "C"
%%
%% Function: BlockTypeSetup ================================================
%%
%function BlockTypeSetup(block, system) Output


        /* General purpose: Include libraries of the MCU, function wrappers, */
        /* variables, function prototype instantiation remembering this is   */
        /* executed only once for block type */


        /* Specific implementation: */
        // Library Inclusion


%endfunction
```

That portion of code show the first part of it where the method `BlockTypeSetup` is used to include the common libraries. The next step is the `Start` method.

```
%% Function: Start ============================================================
%%
%function Start(block, system) Output
```

```
        /* General purpose: Used to include and initialize variables or more  */
        /* before block instance execution. That method is used for each      */
        /* instance */

        /* Specific implementation: */
        // ISR prototype
        // External variables
        // Global
        // TPM set-up
        // IRQ setup
        // ISR function
        //      - Overflow detection
        //      - Sensor routine with speed measurement

%endfunction
```

This call is the core of the speed sensor. This method is executed before the normal execution, and it acts as a sort of "last initialization". This process declares variables and sets-up the TPM timer and Interrupts. Inside the Interrupt Service Routine is present the algorithm of the speed sensor. The speed value measured inside this routine is stored in a global variable. For this reason, the Interrupt Service Routine constitute a sort of buffer, as the value is stored until another measure is done. The speed values are read from the loop with a fixed time step, but since the measurements are done asynchronously, these values are stored in the global variable.

```
%% Function: Outputs ============================================================
```

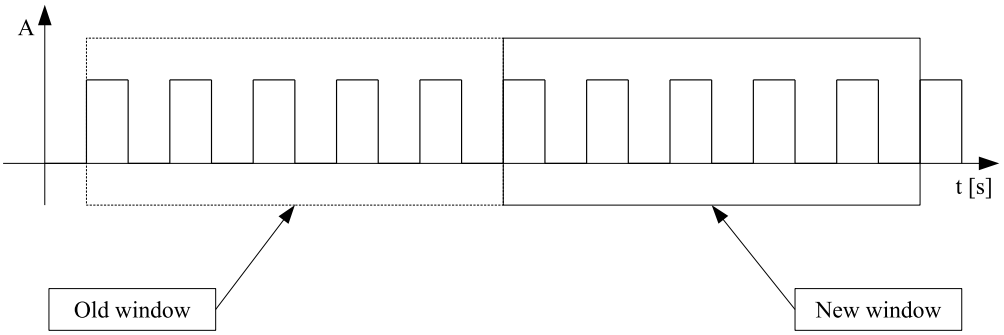


Figure 20: Working principle of the fixed-window.

```
%%
%function Outputs(block, system) Output


        /* General purpose: gives output of the block */


        /* Specific implementation: */
        // Speed measure in output port


%endfunction
```

This last method is used in order to transmit the output speed value stored in the global variable to the block output when the loop is executed. In this way, the implemented method synchronizes the event based speed measurement with the time step of the loop execution.

5.2.3 Fixed Window

With this method each speed measurement is obtained with a specified number of input events (identified by magnets passage). With a fixed amount of magnets (n), the

time interval is detected by evaluating the difference of time between the "n^{th}" magnets and the first. In detail, when the first magnet is passed, the TPM's counter value is read and stored in a variable. A software counter (a variable) is increased by one for each magnet passed until the "n^{th}" occurs. When this happens, the time interval is detected by difference of the actual time value and the first, taking into account possible overflows of the module counter. In order to evaluate if and how many overflows have occurred, another counter variable is increased each time one has happened. If no overflows have occurred, then the time interval is calculated simply as difference between counter value at the windows limit and the beginning. The equations utilized are

$$\Delta t = t_{end} - t_{start} \qquad \text{without overflows;} \tag{5.2}$$

$$\Delta t = t_{end} + (t_{MOD} \cdot n_{ovf}) - t_{start} \qquad \text{with overflows;} \tag{5.3}$$

$$v = \frac{n \cdot K_{speed}}{\Delta t} \tag{5.4}$$

Notice that in equation 5.4 term K_{speed} is a constant that depends on the clock frequency, number of magnets and arc of circumference d_m. After the first measure the last magnet is considered as the new first and old t_{end} become t_{start}. The time interval speed is measured. Each event triggers an ISR only when the window width is reached, and the speed measurement is performed in it (fig.21).

The name fixed window is understandable: segmentation is based on time samples to be moving, jumping of n measures each interval. Under this methodology, the speed measure is obtained using only new time samples each time speed is evaluated. In that

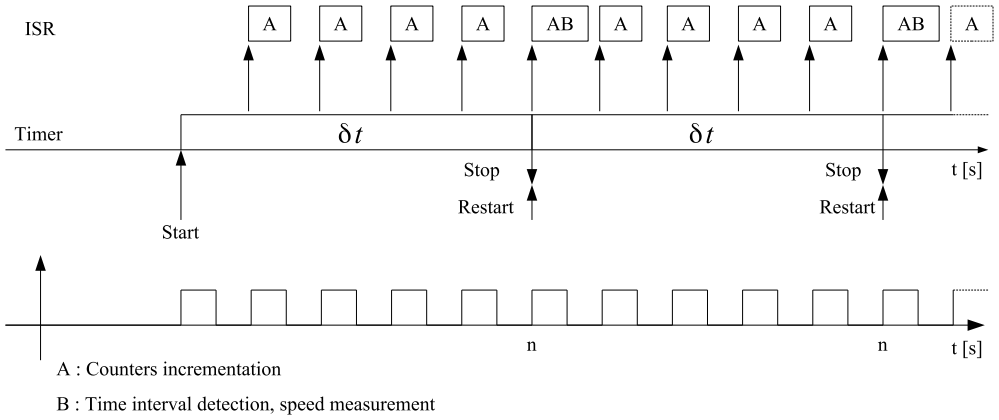


Figure 21: Event that triggers timer start/stop and ISR in fixed-window approach.

way the detection of the variations of speed is faster, but on the other hand the robustness is not assured as peaks could result from signal noise or disturbances introduced by magnetic field, these factors are not well attenuated. It is important to properly choose the length of the window, because if the number of magnets is too high some variations could be attenuated too much from values that are not affected by the variation. In addition with a wider window the computational cost is reduced because the computation of the speed is performed each "n" magnets, rather than at each magnet.

5.2.4 Sliding Window

The Sliding Window method, like the fixed window method, is an event based measurement. The speed is evaluated depending on a specified number of events. These events are in practice identified by the rising edge of the square deriving from the detection of magnets by the Hall-sensor. The name Sliding derives from the concepts that measure is obtained with one new measure and "n-1" old. So each time instant the new

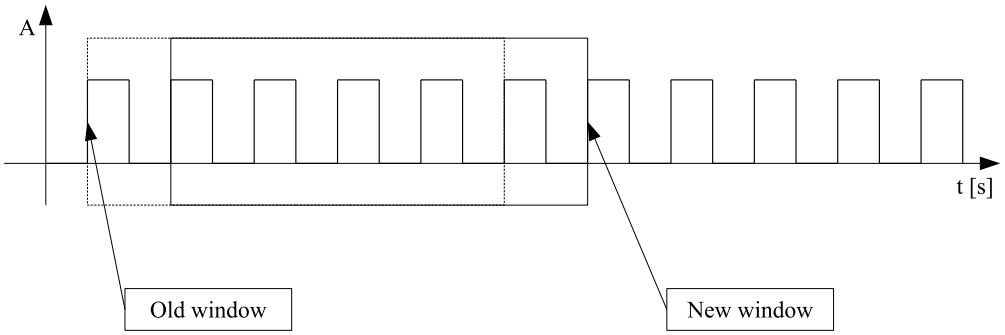


Figure 22: Working principle of the sliding-window.

measure is add to the set of old measures. The oldest is removed and the other are shifted. The window in that way slides including new value and leaving the oldest. The implementation of this is performed using a buffer. Choose the value of "n" is used as dimension of the buffer. When it is full, the first measurement is done and after that each new time interval is added to the buffer and the oldest one is discarded. Then the buffer elements are summed up constituting the time interval used in order to evaluate speed. Equations 5.2 and 5.3 are still valid, but in that case the time interval used in 5.4 derives from the sum of all time intervals. Moreover it follows that for each interval

the t_{end} and t_{start} will change each new event, end time become the next start and so on. The equations become:

$$\delta t_i = t_i - t_{i-1} \qquad \text{without overflows;} \tag{5.5}$$

$$\delta t_i = t_i + (t_{MOD} \cdot n_{ovf}) - t_{i-1} \qquad \text{with overflows;} \tag{5.6}$$

$$\Delta t = \sum_{i=1}^{n} \delta t_i \qquad \text{with } i = 1, \ldots, n; \tag{5.7}$$

$$v = \frac{n \cdot K_{speed}}{\Delta t} \tag{5.8}$$

$$\delta t_{i-1} = \delta t_i \qquad \text{with } i = 2, \ldots, n \text{ when buffer full.} \tag{5.9}$$

In the equations index i represents the event and n the window length. The time event intervals are stored inside the buffer. The equations 5.5, 5.6 and 5.8 are the equivalent of 5.2,5.3 and 5.4 for the sliding window. The main difference resides in 5.9, which describe the shifting of the time intervals in the buffer. So, the first measurement is equal to a fixed window measure because the buffer is still empty but small intevals are stored as described by equations 5.5, 5.6 and 5.7. When the buffer is full, the index of equations become $i = n$, since the last time interval has to be added. These operations are performed inside an ISR: buffer elements are stored and old elements discarded, counter and flags are incremented or decremented, and, in the end, speed evaluated. The last operation that constitutes the core of the measurement, is executed in this

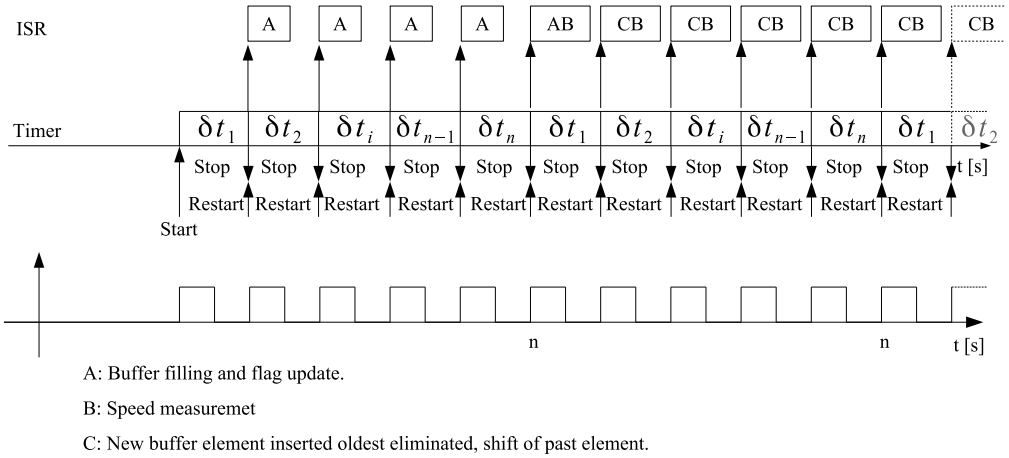


Figure 23: Event that triggers timer start/stop and ISR in sliding window apprach.

approach each time the rising edge in the sensor generated signal (neglecting the first measure that require "n" events since buffer has to be filled).

The characteristic of this kind of speed measurement is that the measure is affected by one new value and $n-1$ old intervals. It results that in case of variations the detections is slower but it could constitute a robustness point that provide a slight attenuation if unwanted variations and disturbances occur. It is obvious that this method requires more computational performances and memory allocation, as computations have to be performed for each magnet detected and a buffer is required. These in addition depends also from the window width: the more window is wider, the more is computationally expensive. Furthermore it is interesting that time needed to provide a measure, neglecting computation time of the algorithm, is not dependent on the window length, except for the first measure, since a speed value is provided for each magnet.

5.2.5 Validation and Comparison

The characteristics of the two implementations have been analyzed. It is proper to validate both and analyze differences in precision and response. In order to validate the two methods, a percentage of the duty cycle as assigned to PWM in order to control the motor. Then, in steady state conditions, the square wave has been analyzed via the output of the hall sensor. Observing the pulses' period, with reference to eq. 5.1, leads to the calculation of the speed measure. That measure constituted the value considered true. Thus the speed has been measured with both implementation methods. Using the oscilloscope, the signal generated by the hall sensor has been analyzed and, from the measured frequency value, speed reference has been measured. (The test has been performed in free wheel condition without friction, with PWM duty cycle at 50% and window size of $n = 5$). The frequency obtained from the oscilloscope has been converted in the speed considering the value of d_m, which represents the arc of circumference of vehicle's wheel and related to the angle between two magnets in the phonic wheel as explained in 5.1.3. From that value, the corresponding wheel speed is calculated, taking into account the number of magnets that contribute to the speed measurement. So the equation that convert the frequency value in speed measure in mm/s is:

$$v_{ref} = f \cdot d_m \quad \text{with } f = \text{frequency, } v_{ref} = \text{speed} \tag{5.10}$$

In the specific test the values used are shown in the table.

Measure procedure was the following:

TABLE IV: VALUE USED FOR SENSOR SPEED TEST

Characteristic	Description	Value
d_m	Describes the arc of wheel circumference between two magnets	6.54 mm
N_m	Represents the number of phonic wheel magnets used in the speed evaluation	24
f	Refers to the frequency related to the period between two magnets measured with the oscilloscope	641 Hz
v_{ref}	Is the speed value evaluated from measured frequency that constitute the reference value for the sensor measures	4192 mm/sec
n	Sensor window size	5
PWM	PWM duty cycle percentage	50%

1. Signal analysis and frequency measurement with the oscilloscope
2. Speed acquisition from speed sensor in *Fixed Window* mode with n window size
3. Speed acquisition from speed sensor in *Sliding Window* mode with n window size

The graph in fig 25 shows measured speed in both implementations of the sensor and compared to the reference speed value. The first second has to be neglected representing obviously the transitory part needed to reach steady state value. The speed measured values seem to oscillate around a mean value in both cases. That mean value is almost equivalent to the speed reference value related to the frequency measured and calculated as in 5.10. This could be seen better in fig 26, where the mean values of both measures are displayed separately. Those values are close to the reference one, table V reports the

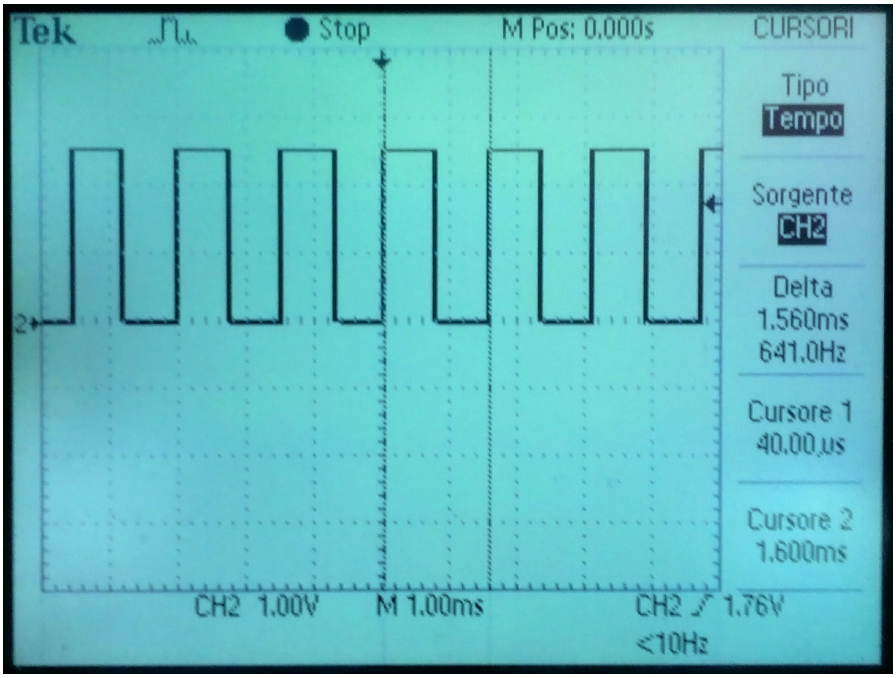


Figure 24: Oscilloscope measured signal.

mean values of measured speeds and the variation coefficients of the two method related measures. As shown from the graphs, there are low disturbances on the speed value that contribute to the coefficients of variation values shown in table V. These disturbances could be due to the magnetic field leakage from the DC motor affecting the hall sensor.

Another analysis of the behavior could based on the measurements obtained for very high and low speed. From graphs in fig. 27, it is possible to observe that at high speed 27a the measures are affected both case by peaks with either window method. These could be due to bumps in the rotation or also from magnetic field disturbances introduced on the hall sensor, but in higher quantity than for lower speed. The absence of direction of speed could make the situation worse. Fig. 27b instead presents measures

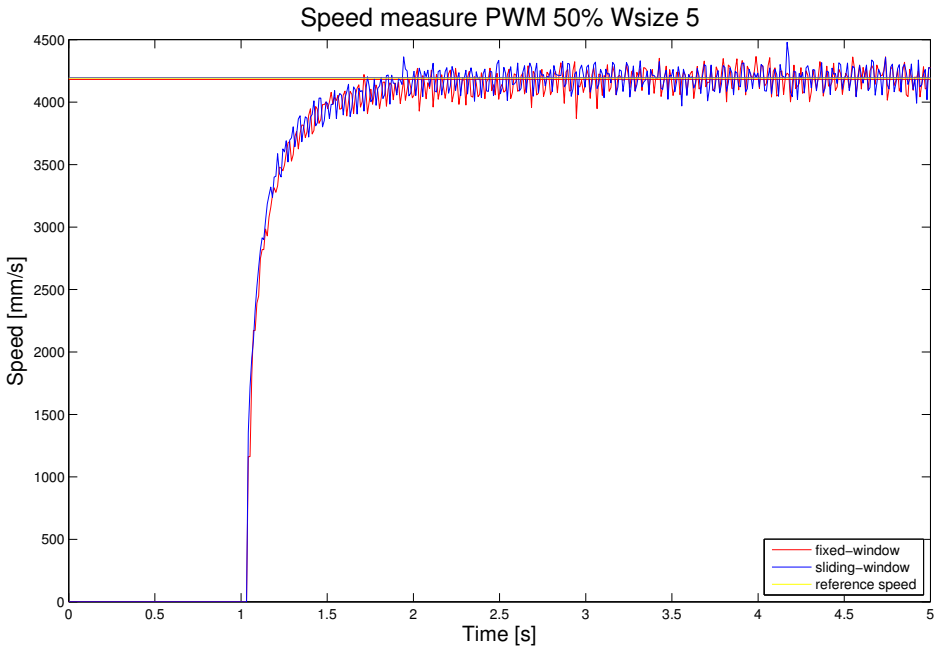


Figure 25: Speed measures comparison.

TABLE V: VALUE USED FOR SENSOR SPEED TEST

Description	Value	Coefficient of variation (CV)
Fixed-window speed mean	4182 mm/sec	2.22%
Sliding-window speed mean	4196 mm/sec	2.15%
Reference speed	4192 mm/sec	—

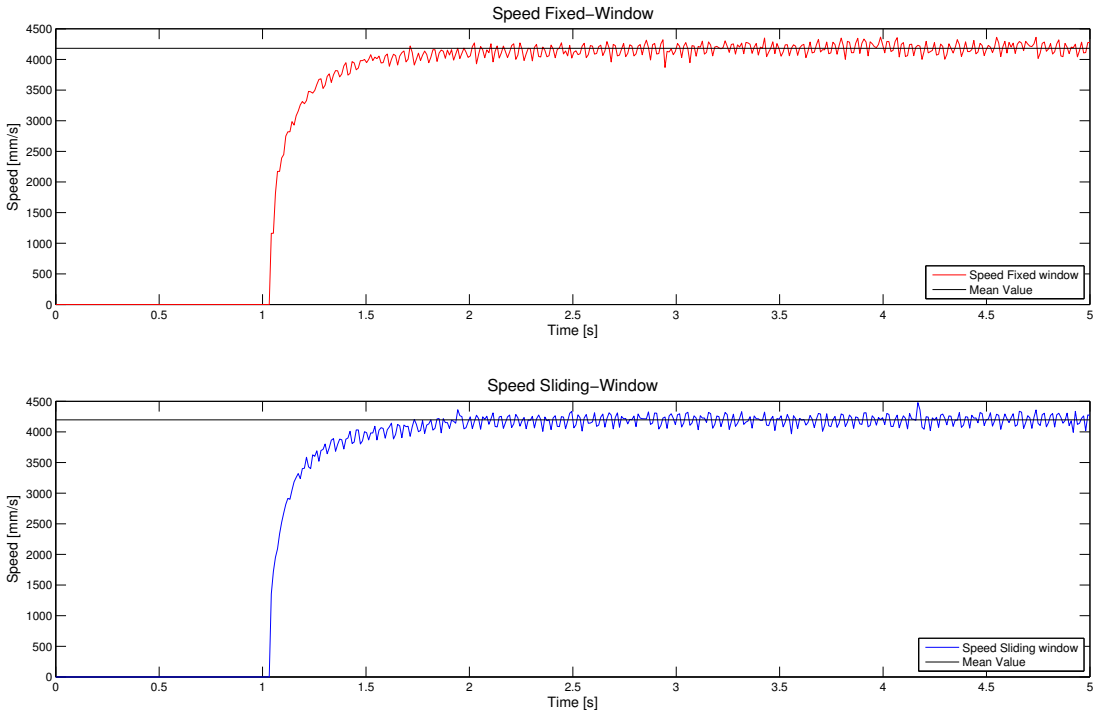


Figure 26: Speed measures separated.

of a soft disturbance, but not peaks for high speed. The result of the measures appears to be almost equal with both methods.

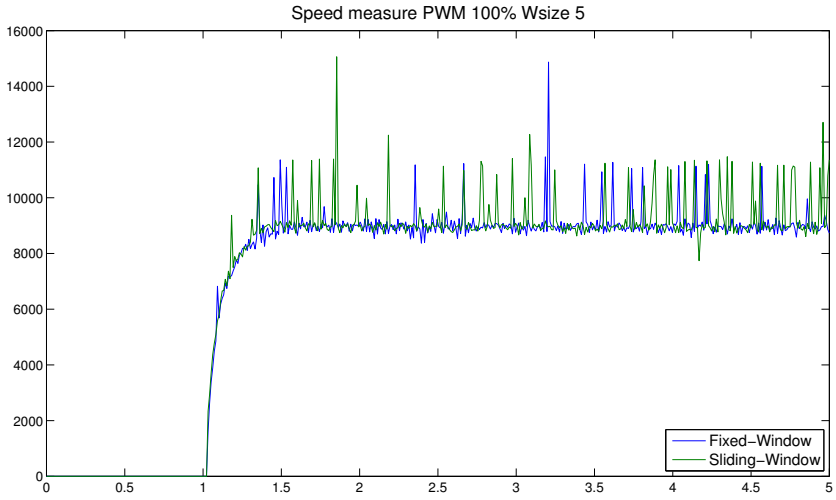
5.2.6 Speed sensor Block

The speed sensor implementation, as discussed, performs the measurement in two different ways. The development of the driver finally allows the production of a block that perform that function. The block realized in that case has the following aspect. In order to ease the application of changes to the block characteristics, a mask has been developed. The mask provide an interface that quickly configures the speed sensor in the way that is needed for a given application.

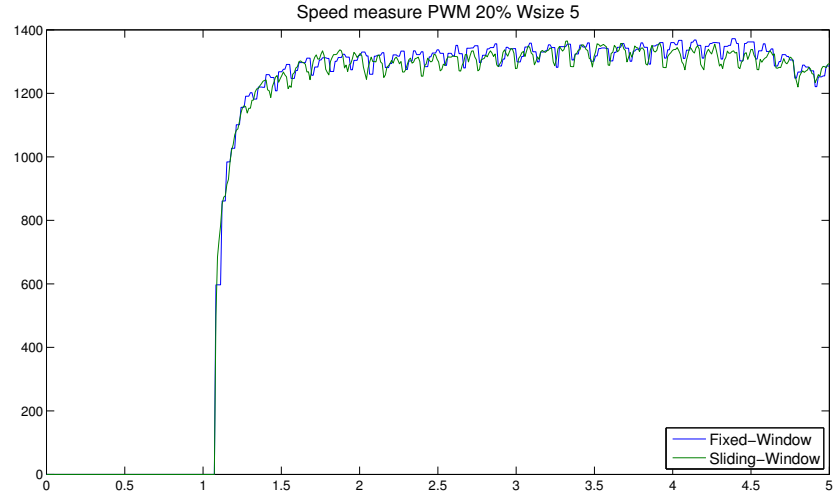
The selection of the block is, as usual for Simulink, possible by dragging and dropping it in the model area. The block appear as in figure 28. In order to select parameters mask has been designed to set different parameters to implements different characterisitcs of the driver. The mask is represented in fig 29, resulting from the selection of four different parameters:

- Sensor
- Timeout
- Method
- Window Size

Let's examine in details what those selections means.



(a) *High speed*



(b) *Low speed*

Figure 27: Speed method comparison.

Figure 28: Speed sensor block aspect.

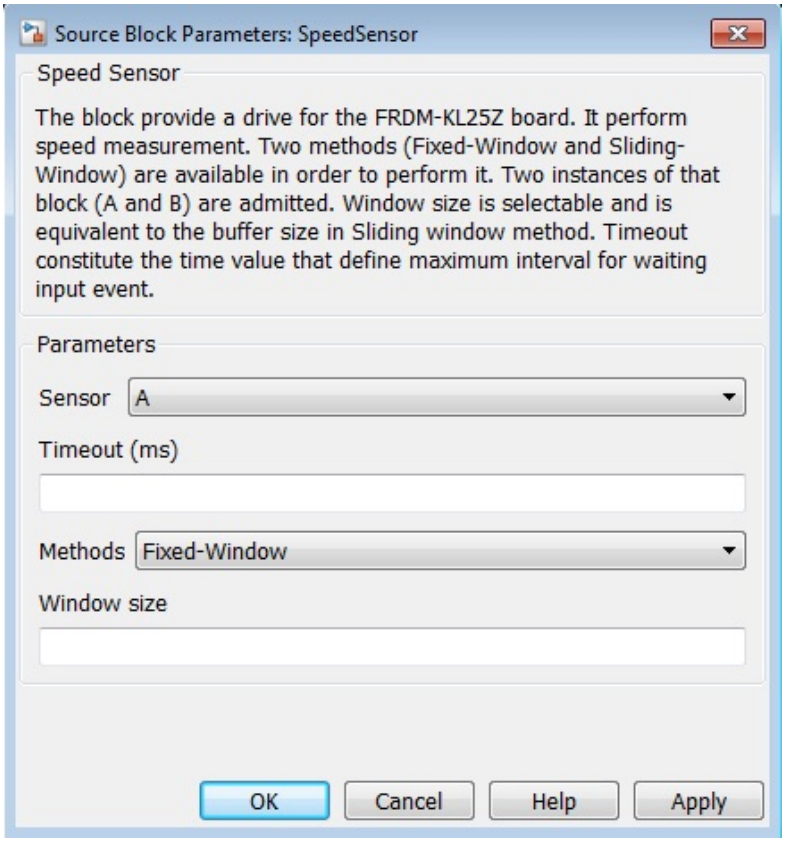


Figure 29: Speed sensor mask.

5.2.6.1 Sensor

A popup menu allows one to choose which of the two sensors available on the vehicle should be used. The sensors names are strictly related with those assigned to motors, according to the *FRDM-KL25Z Support Package* provided by MathWorks. The possible choices are shown in fig. 30.The sensor are specifically :

- A: for the right side motor
- B: for the left side motor

The selection A or B modifies the implement, which, in turn, select one of the different TPM module channels. Since the channel is physically linked to an input, it is of the at most importance to correctly connect the chosen sensor to the related pin.

5.2.6.2 Timeout

Timeout has to be specified in a text input window, as seen in fig 31. This parameter defines the maximum time interval that could pass between the passage of two consecutive magnets. If the expressed value is reached, the vehicle is considered in still condition and speed is recorded as equal to zero. In the code, the parameter is converted to a number that represents the equivalent number of overflows. When this limit is reached, the vehicle is considered in still condition, and the speed is null.

5.2.6.3 Methods

Methods are selectable with a popup menu 32. The available implementations of the sensor that are:

- Fixed Window

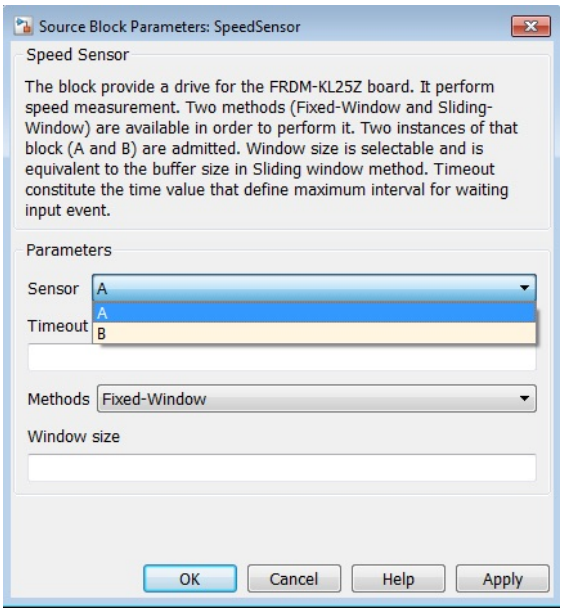


Figure 30: Speed sensor mask: Sensor selection.

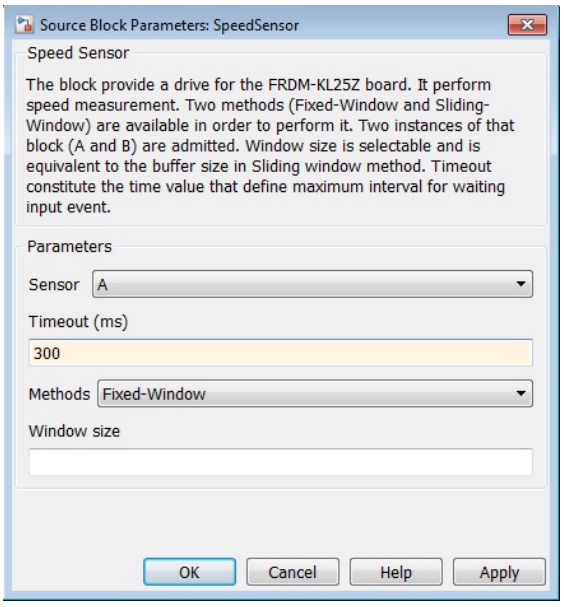


Figure 31: Speed sensor mask: timeout.

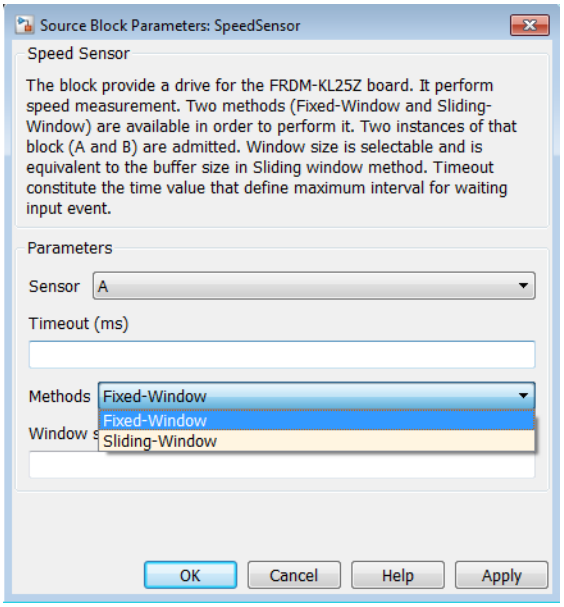


Figure 32: Speed sensor mask: methods.

- Sliding Window

Those have been discussed in sections 5.2.3 and 5.2.4. The differences introduced are mainly related to how the input signal is treated in order to evaluate speed. Principally, the former is simpler and provides a new measurement, and is able to detect the speed variation quicker, while the latter, on the other hand, is more robust and mediates the measurement with reference to previous interval values.

5.2.6.4 Window Size

Window size could be defined by typing the value in a text input window as could be seen in fig 33. The window size is defined as the number of consecutive values used in the measurement. In *Sliding Window* mode, the length of the buffer must also be defined.

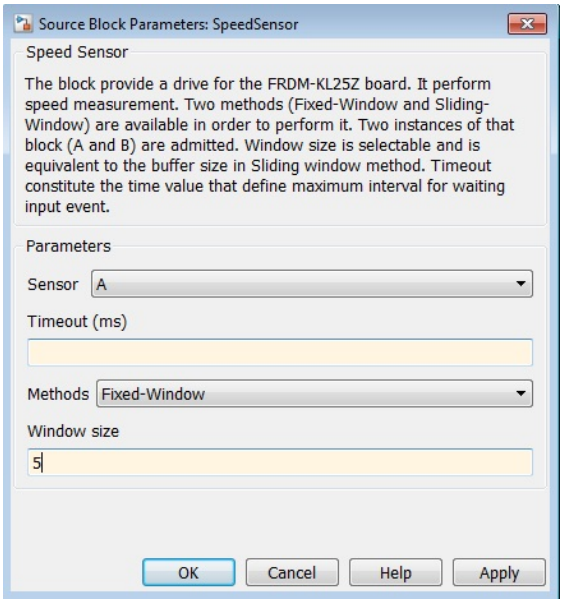


Figure 33: Speed sensor mask: window size.

It is important to decide the proper value of the window size because this determines the value of speed measured. In *Fixed Window* it is directly related to the time needed to have a new measure and in *Sliding Window* method, the time is needed to detect speed variations, as explained in sections 5.2.3 and 5.2.4.

CHAPTER 6

MOTOR CONTROL

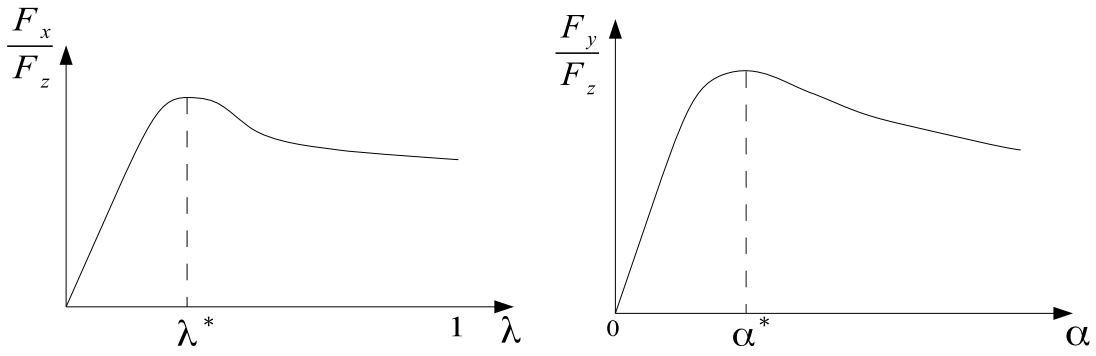
This chapter will explaine how the motor has been designed and how a control algorithm for speed control is realizable. First we will consider how the transfer function has been evaluated, followed by a discussion of the estimation methods used, a crucial point in the operation as high non linearity is present and there is not a proper data-sheet.

6.1 System

The control system that has to be controlled is not constituted simply by the motor. The wheel speed must consider many factors that influence vehicle behavior. Obviously the main part is the Motor, which constitutes the actuator and performs the energy transformation from electrical to kinetic. The motor dynamic is the principal part, but the dynamic of the supply circuit must also be considered. The wheel soil friction model is present and substantially modifies the behavior of the vehicle.

6.1.1 Supply System

The supply system provides the voltage to the motor. The voltage is regulated, and modifies the duty cycle of the PWM. The core of the supply system are H-Bridges; the maximum voltage that could be provided in output is the input one. That value could be reduced modifying the percentage of the duty cycle that vary between $0\% - 100\%$, number which respectively represent the highest and lowest outputs. The H-bridge is constituted



(a) *Longitudinal Slip* (b) *Slip Angle*

Figure 34: Tyre slip graphs

by four switches used for reducing the output power. The switches are constituted by MOSFET transistor. The transistors are characterized by a high frequency dynamic behavior, and its response depends on the load. In general the high frequencies harmonics introduced by the switches are cut off by the mechanical system itself, as their high inertia values act as a low-pass filter. So the dynamic of the H-Bridge is considered negligible.

6.1.2 Wheel-Road contact

The system behavior is affected by the contact of the wheel with the soil. The contact force that derives from this phenomenon influence traction behavior. Longitudinal and lateral traction forces are characteristic of this. These forces depend on slip ratio (difference between vehicle and wheel speed over wheel speed), slip angle (angle between velocity direction and imposed direction) and normal force (the component perpendicular to the ground). The general characteristic that describes this behavior is shown in fig 34a, where λ represents the slip ratio. A similar shape is valid in order to represent

the lateral force trend with respect to the slip angle as in fig 34b. That is not easy to describe. There are many equations that approximate this behavior. The most famous is the so called *"Magic Formula"* introduced by Pacejka (10):

$$F_x = D\sin\{C\arctan[B\lambda - E(B\lambda - \arctan(B\lambda))]\} \tag{6.1}$$

$$F_y = D\sin\{C\arctan[B\alpha - E(B\alpha - \arctan(B\alpha))]\} \tag{6.2}$$

Equations 6.1 and 6.2 refers to longitudinal and lateral traction, respectively. The traction forces don't influence dynamically the system but introduce non linearities that may influence the behavior of the whole system. The problem of tyre slip is widely studied in traction control. The aim of a traction control system is to assure maximum traction in every asphalt condition, assuring the maximum traction forces. To reach this scope, the slip must be pushed below the critical value, which is related to the peak of traction force. The region before the critical slip is the *linear region* while for higher slip values there is the *non-linear region.*

6.1.3 DC motor

The available motors are two brushed DC motor with permanent magnet. The supply information is reported in table VI. As known, the DC motor dynamic is provided by two couples of equations; the former couple is constituted by *mechanical equations* and the latter by *electrical equations*:

TABLE VI: DC MOTOR SUPPLY INFORMATIONS

Quantity	Value
Supply Voltage	7.2 V
Stall current	3800 mA

$$T = K_t i \tag{6.3}$$

$$J\ddot{\theta} + b\dot{\theta} = T \tag{6.4}$$

$$e = K_e \dot{\theta} \tag{6.5}$$

$$L\frac{di}{dt} + Ri = V - e \tag{6.6}$$

Solving these differential equations results in the transfer function of the motor that relates the input voltage and rotation speed. From equations 6.3, 6.4, 6.5 and 6.6, considering $K_i = K_e = K$ and applying *Laplace Transform*, it follows:

$$\frac{\dot{\theta}(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2} \tag{6.7}$$

Equation 6.7 represents the transfer function of the motor that links the supply voltage to the final rotation speed. In order to realize a controller, the starting point is the transfer function. The transfer function is modified with gains that constitute the controller in order to match the required specifications. To achieve this goal, one needs

the numerical values of the parameters presented in equations 6.3, 6.4, 6.5 and 6.6. The first critical point in the case of study was the absence of a data sheet that led the motor be a totally unknown system. In addition, the instrumental equipment did not perform torque measurement, and it was not possible to measure the back electromotive force. That prevented the acquisition of the motor characteristic. Without these elements, would be impossible to create a model of the motor and realize a controller from it. In order to create a model, an estimation method is necessary. The result of this method allows for the evaluation of a transfer function using the in time input and output measurement from a system.

6.2 Identification of the system and parameters estimation

As said the Identification of the transfer function is required due to the absence of any information about real components. In addition, the system is highly non linear, a fact which prevents the generation of an accurate model from the states equations. Instead thanks to the identification, it is possible to use information directly from real system, and to estimate a transfer function representing the real behavior of the system. In order to perform estimation, a defined model of the system is necessary. Models could be of different type, but in this case an ARX model (Auto Regressive with eXogenous variable) has been chosen. The ARX model is composed as the name explain from an AR variable with an exogenous (or independent) variable (11).

$$\begin{aligned} y(t)+a_1 y(t-1) + a_2 y(t-2) + ... + a_{n_a} y(t-n_a) = \\ b_1 u(t-1) + b_2 u(t-2) + ... + b_{n_b} u(t-n_b) + e(t) \end{aligned} \tag{6.8}$$

The $y(t)$ variable is the auto regressive and the $u(t)$ is independent. The coefficients that precede the variables define the poles and zeros of the related transfer function, with the a_i coefficients specifically influencing poles and b_i the zeros. So, defining the number of poles and zeros implies the imposition on the order of numerator and denominator. There is also another term $e(t)$ representing the error, this term is considered negligible.

This model takes into account the collected measures of the input signal and the output signal from the system. The input is the command signal to the motor "block"; in other words the duty cycle and the output is the speed measured by the speed sensor, which is, in turn, as signal that comes out from the block, as explained in chapter 5.

Using an iterative process with statistical proprieties and end quantities the parameters constituting the transfer function have been estimated. It is a step by step procedure.

The evaluation of the parameters have been practically performed using the *System Identification Toolbox* available in the MATLAB® suite that let with simple passages to set up the wanted estimation for the desired use.

6.2.0.1 Model definition

Focusing on the first step, the choice of poles and zeroes the selected values are:

- **Poles number** $n_p = 2$
- **Zeros number** $n_z = 1$

The related transfer function should be of the type:

$$G(z) = \frac{b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{6.9}$$

The choice of the number of poles and zeros is strictly related to the physical system that has to be modeled. Considering equations 6.7; the result is that the system is characterized by two poles. Those poles characterize the main dynamic of the system, the motor. The whole system is also influenced by the very fast H-bridge dynamic. In general, fast dynamics could be neglected in mechanical systems because these are often characterized by high inertia that determines a low pass filter behavior which cuts the high frequency filtering of the poles. For that reason the number of the poles has been chosen equal to the number of poles in the motor transfer function.

6.2.0.2 Input and Output values

As said the estimation requires input and output in order to estimate the system's transfer function. In order to perform the estimation, a duty cycle of $PWM = 0.5 = 50\%$ has been chosen. The inputs used are shown in fig. 35

It should be noticed that the system is not completely linear. For this reason, the working point has been chosen as it is the mean value in the admitted range.

6.2.0.3 Parameter estimation

The estimation process has been performed utilizes the *System Identification Toolbox*. This toolbox helps with the estimation of the parameters of the transfer function. The tool provides various types of estimates and each of those uses an identification algorithm. The dialog window appears as in fig 36. The procedure used is:

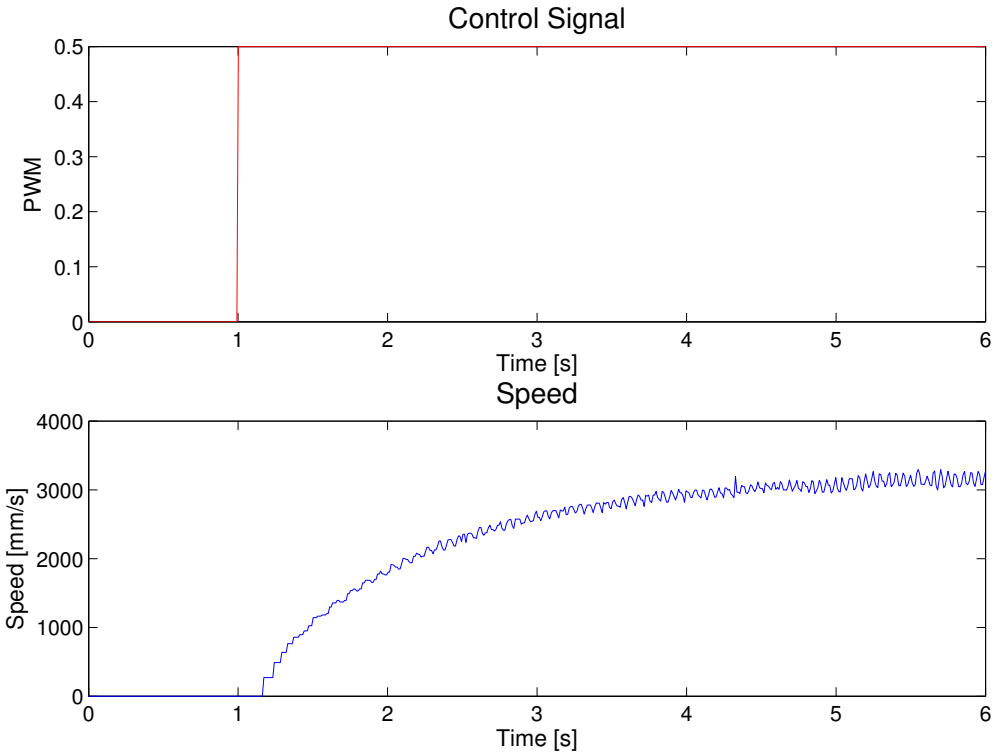


Figure 35: Input and output of the system.

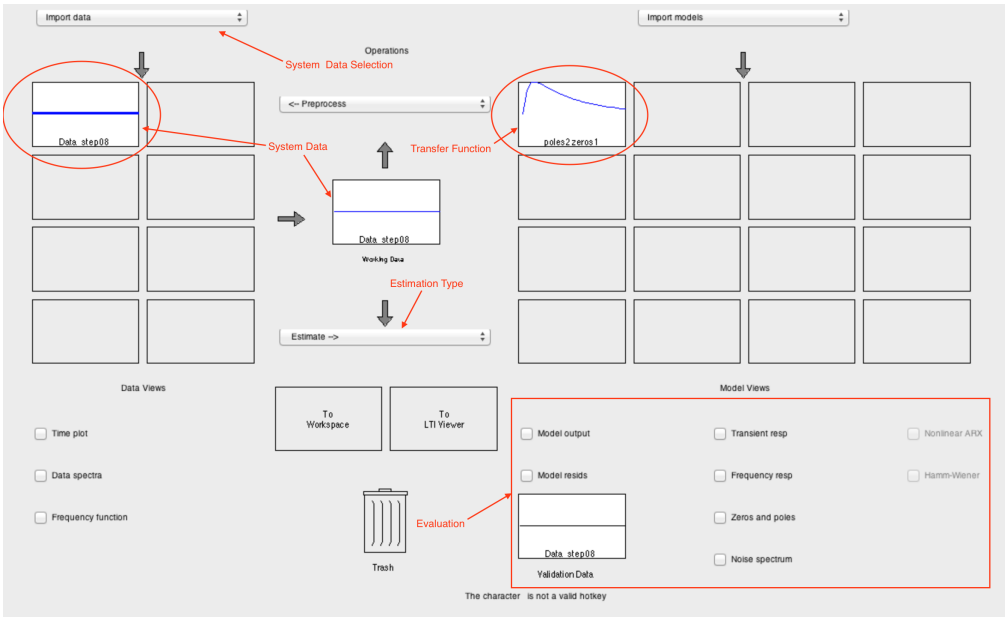


Figure 36: System identification toolbox interface.

- Data selection: input/output system values inclusion as time series data.
- Estimation type: transfer function estimation with selection of poles and zeros number with possibility of bandwidth selection.
- Estimation evaluation: window with estimation result and evaluation tool of the estimated transfer function.

The toolbox doesn't allow to choose manually the estimation algorithm, but provides information about the used one. The method actually utilized is the "Nonlinear least squares method".

6.3 Transfer Function

In order to estimate the transfer function, data has been acquired under the following settings:

- Duty cycle of $PWM = 50\%$
- Acquisition time 6 s
- Only straight line movement

From the estimation the final transfer function results:

$$G_A(z) = \frac{7.784z}{z^2 - 1.86z + 0.8609} \quad \text{with } T_s = 0.01\,\text{s} \tag{6.10}$$

Using the tool provided by *System Identification Toolbox*, it is possible to evaluate if the estimated method matches the required characteristic, and if the measure is precise enough. The graph in fig.37 shows, that this transfer function fit very well the measured output, using the original input. The transfer function output fits the system measured output for 94.89%. For that reason the estimated transfer function is considered valid.

In addition, estimation could be performed with transfer functions at higher orders, as in fig 38a and 38b, where the former shows results obtained increasing the zeros number and the latter modifying poles and zeros number. It is possible to observe that, obviously, the system is better approximated with higher order transfer functions. Yet the possible benefits don't justify the use of function of higher order: the maximum fit is of 95.44%, increasing the benefits on the order of only 0.55%.

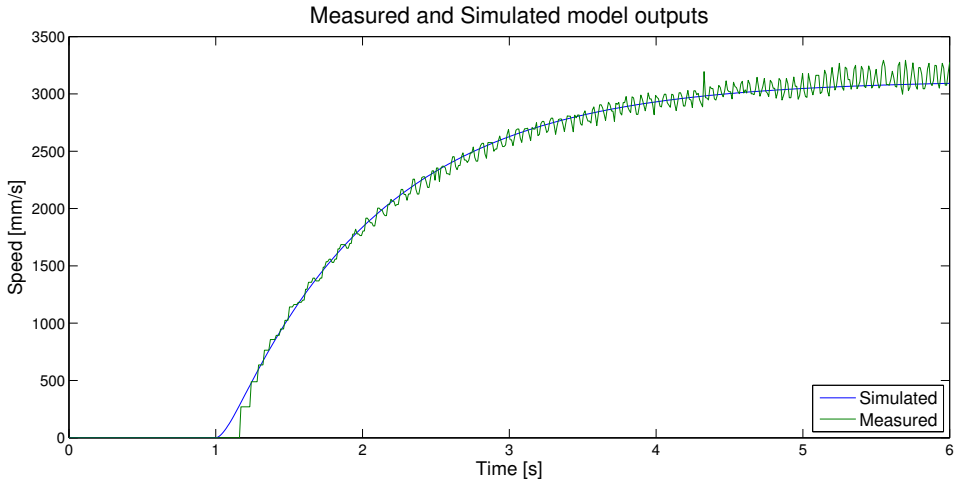


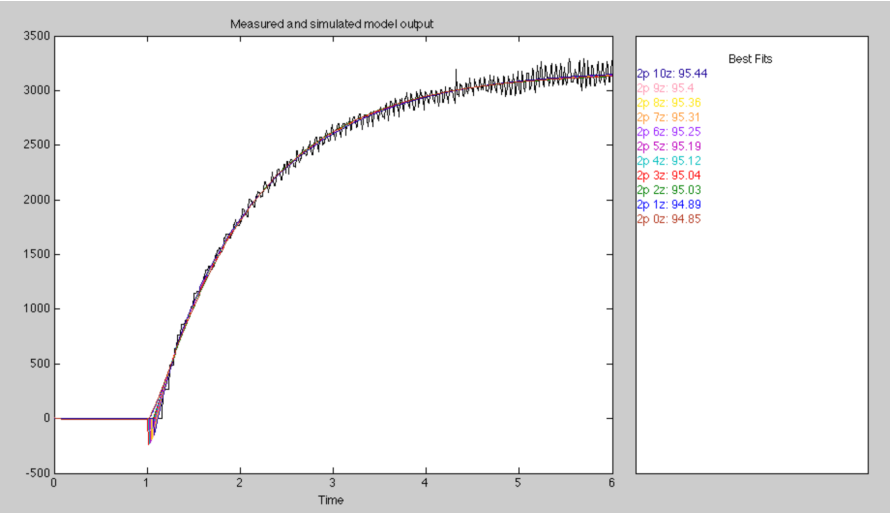
Figure 37: Transfer function output and system measured output.

6.4 Controller Syntesis

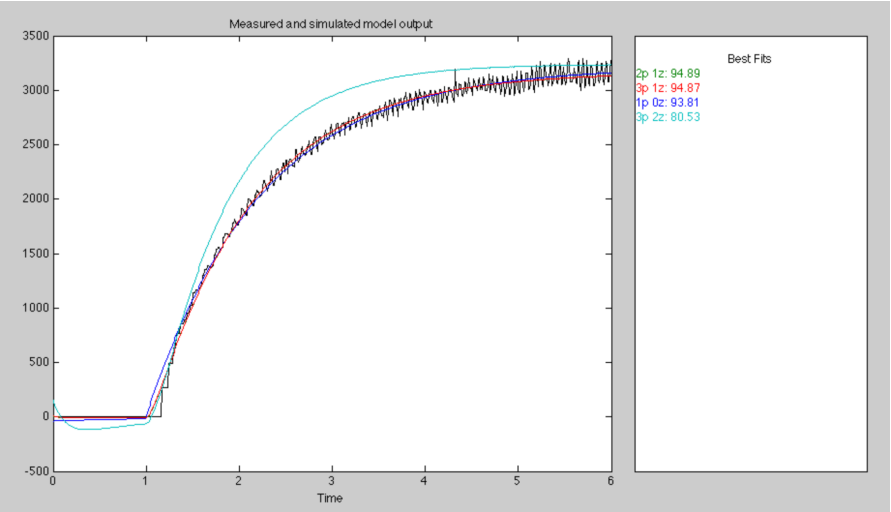
The system has to respect the requirements in terms of speed that the differential algorithm needs. For this reason, it is necessary to realize a control system that effectively assures that the motion speed is correct.

6.4.1 Model-based design method

The model-based design ambient provides many tools, that aid in realizing the control system, allowing for easy solution to a common problem. Model-based design fits in the context of the fast prototyping, which is facilitated with the employment of the model-based design software. This approach is common in industries, where reduction of design time is directly correlated with lower cost and higher profits. This approach uses the tuning method based on the process step transient response of the closed-loop. The software computes the gains of the desired controller type, to assure phase margin.



(a) *Different zeros number*



(b) *Different zeros and poles number*

Figure 38: Comparison of different order transfer function

Internally, it uses a tuning method called *relay method*(12). After that, it is possible to adjust gains, thanks to the interface which provides plots of tuned and original response. Moving the two cursors, one for the speed of the transient and the other for robustness, the controller gains are modified. The *PID Tuner* tool, from the MATLAB® suite, is utilized. A PI controller has been chose and a phase margin of 60° set. The derivative action has been avoided, as it disturbs the speed measurement from the sensor. Then, the procedure used for the realization of the controller, is:

- **Model of the loop**: With the transfer function 6.9 constituting the process and the Simulink® library has developed a closed loop introducing the *PID* block.
- **Controller choice**: From the configuration interface of the *PID* block, PI controller has been selected.
- **Tuner**: Then it is possible to choose *Tune* option that execute the preliminary auto-tuning operation.
- **Adjustment**: Therefore the controller gains could be adjusted to satisfy the required behavior.

Adjustments to the automatic tuned response, have been performed in order to generate the transient parameter reported in table VIII.

The obtained values are shown in VII.

TABLE VII: GAINS OBTAINED USING TUNING TOOL PROVIDED BY MATLAB®

Gain	Value
K_p	0.000941
K_i	0.000851

6.4.2 Controller analisys

Once obtained, the quality of the controller has to be verified. The frequency response, step response, stability, output signal and command signal are examined to see if they imply the behavior desired.

6.4.2.1 Frequency Response

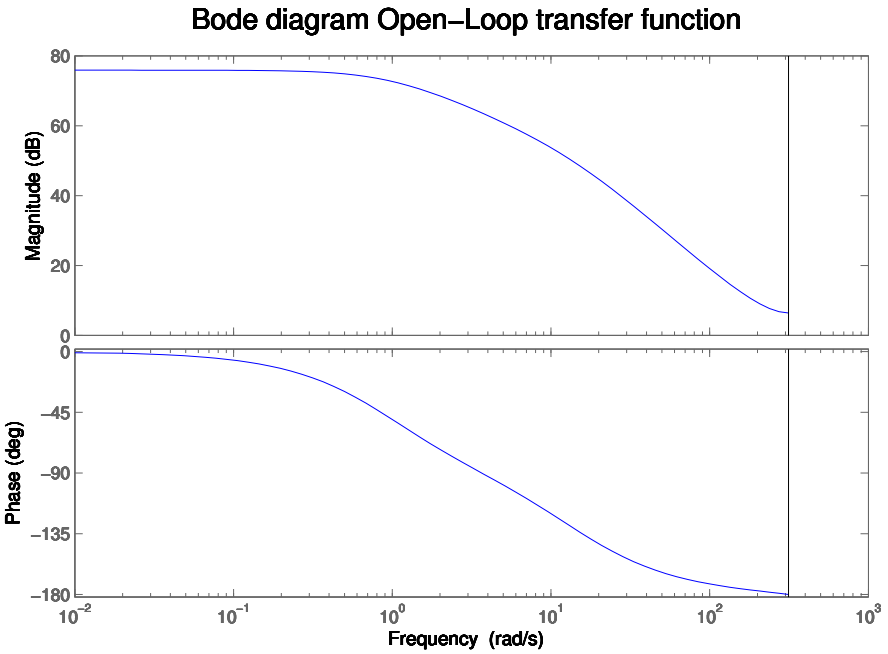
If the resulting frequency response is similar to the target, we can assume the differences reside primarily in the gain at low frequencies in the uncontrolled system response as seen in fig 39a and 39b.

6.4.2.2 Step Response

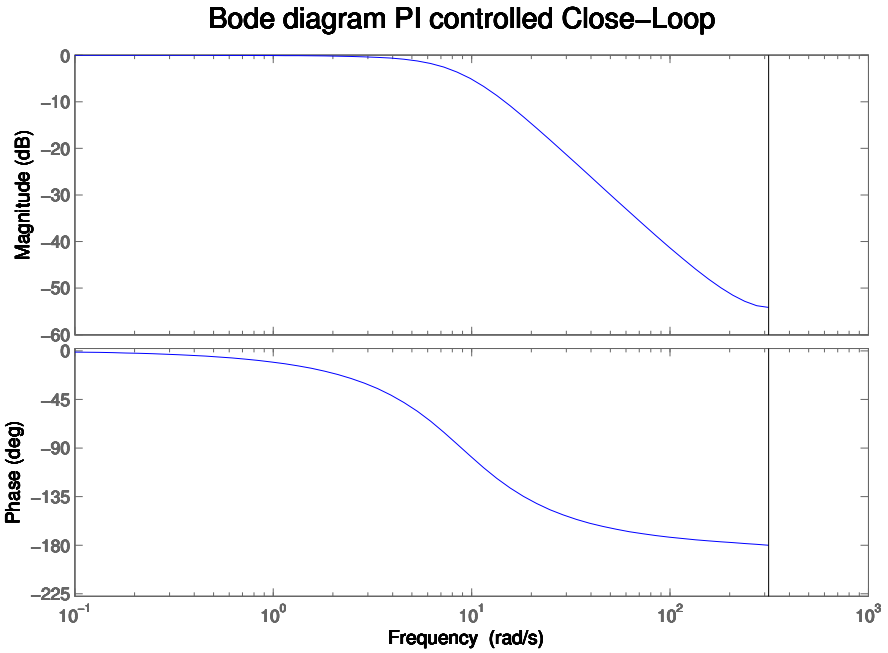
Step response shows the transient behavior, and allows for the verification of the specifications presented in Tab.VIII.

The adjustments have been performed on the automatic tuned response, in order to generate the transient parameter reported in Table VIII.

The information deriving from this process are reported in table IX. Comparing the values of rising time and overshoot with the desired values in table VIII, it is evident that requests are satisfied.

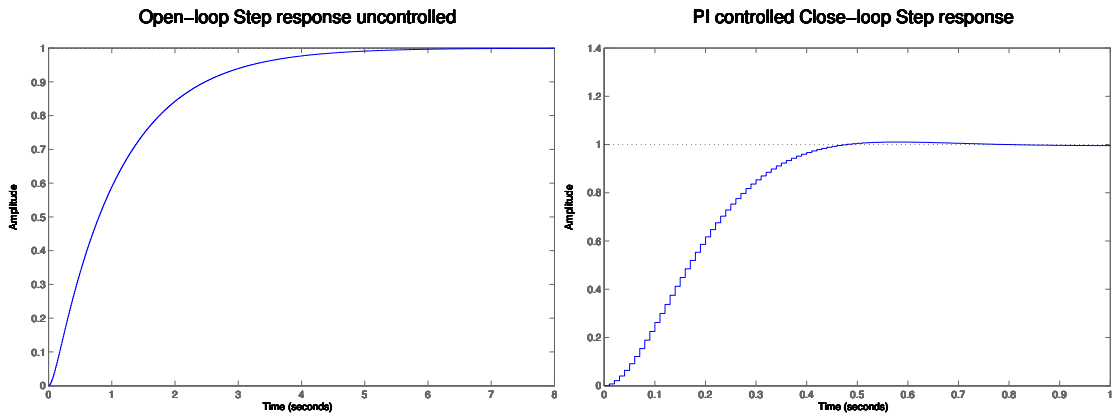


(a) *Uncontrolled system*



(b) *Close-Loop with PI*

Figure 39: Frequency response



(a) *Uncontrolled Open-Loop* (b) *PI Close-Loop*

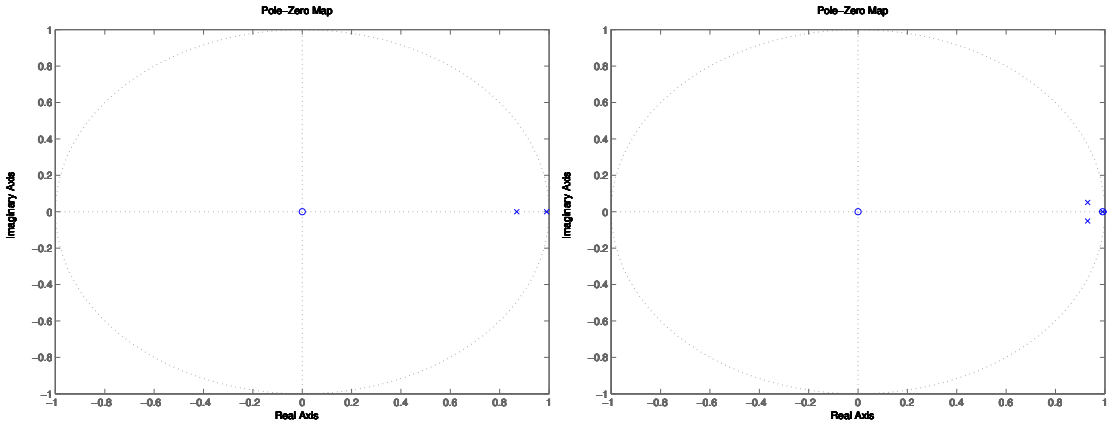
Figure 40: Step response

TABLE VIII: CHARACTERISTICS OF TRANSIENT RESPONSE DESIRED

Characteristic	Value
r_t	0.3 s
$\hat{s}$	1%

TABLE IX: CHARACTERISTICS OF TRANSIENT RESPONSE OBTAINED

Characteristic	Value
r_t	0.2800 s
$\hat{s}$	1.0417%



(a) *Uncontrolled Open-Loop* (b) *PI Close-Loop*

Figure 41: Poles and zeroes map

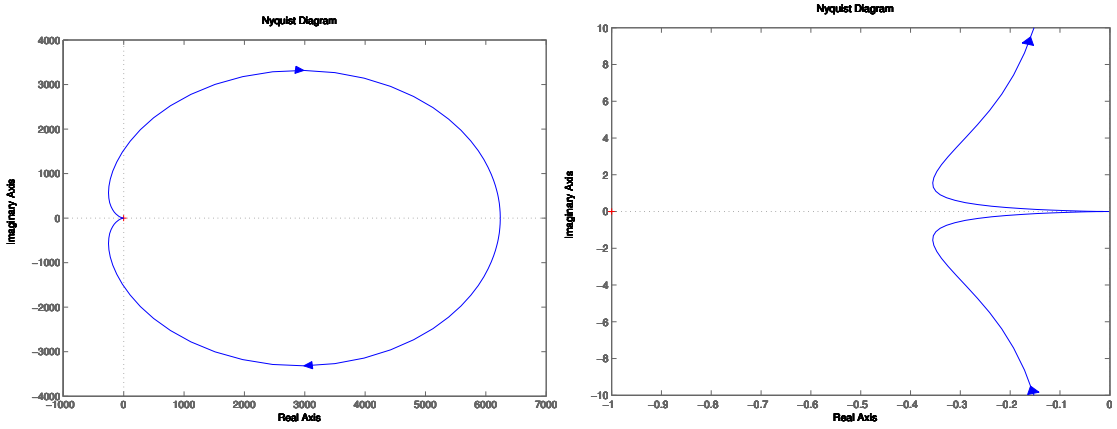
6.4.2.3 Stability

It is important to also verify that the control loop creates a system with a convergent behavior. This results from the step response, but it is proper to analyze the stability of the loop in order to define also the tolerance of the stability.

At first poles and zeros values must be analyzed; these are reported in table X. The results show that both the open loop and the controlled close-loop systems present stable poles inside the unit circle. On the other hand, the close-loop system without a controller has a pole outside the unit circle. Fig 41 shows the pole-zero plots. In addition, from the close-loop system with a PI controller it is possible to see that the first pole has a module near to the unity but there is a zero that attenuate this influence. From the roots it can be concluded that the PI controller produce a asymptotically stable transfer function.

TABLE X: POLES AND ZEROS

Function	p_i	z_i
System	$0,99046$	0
	$0,86919$	–
Close-Loop	$-5,77532$	0
	-0.14907	–
Close-Loop with PI	$0,99105$	0.99095
	$0,93065 + 0,05086i$	0
	$0,93065 - 0,05086i$	–



(a) *Uncontrolled* (b) *PI action*

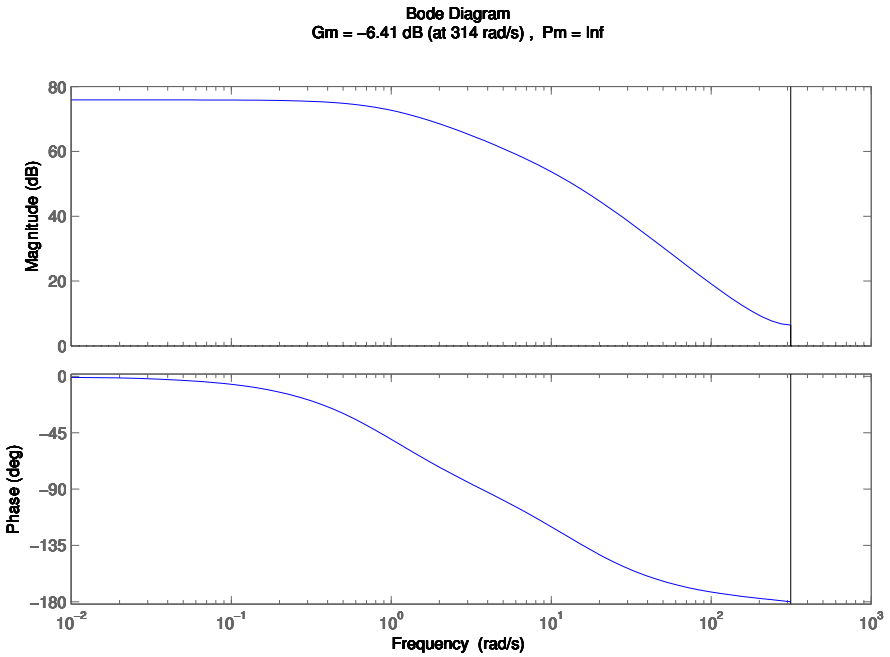
Figure 42: Nyquist graphs.

TABLE XI: BODE MARGIN OF CONTROLLED SYSTEM

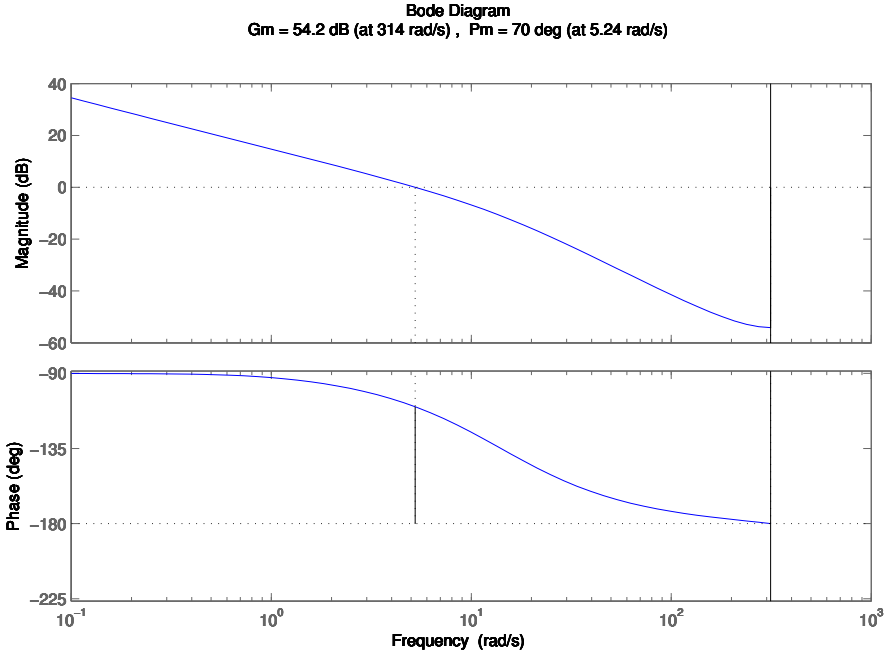
Margin	Value
Gain K_m	54.2 dB (at 3.14 rad/s)
Phase ϕ_m	70° (at 5.4 rad/s)
Stability margin s	0.018

Here are shown the bode plot with *phase margin* and *gain margin*. It also could be seen from the Nyquist plots in fig.42 that the non controlled system executes a clock wise turn around the critical point -1, which implies that the system is unstable. With the controller, the system instead crosses the real axe far to the right of the critical point: the controller stabilizes the system. The instability is caused by the sampling frequency imposed. this frequency limits the bandwidth causing the instability of the system. That is highlighted in figure 43a where a *phase margin* tending to infinity is evident. The expected one, instead, is $-90°$, being the system with two poles and one zero. Additional parameters are available from the bode diagram of the controlled and uncontrolled system. These parameters, *phase margin* and *gain margin*, are reported in figure 43. The values are reported also in table XI. It is evident that the system controlled affers higher margins, assuring a certain robustness.

It therefore, follows that the control system effect leads to a faster and more stable system.



(a) *Uncontrolled*

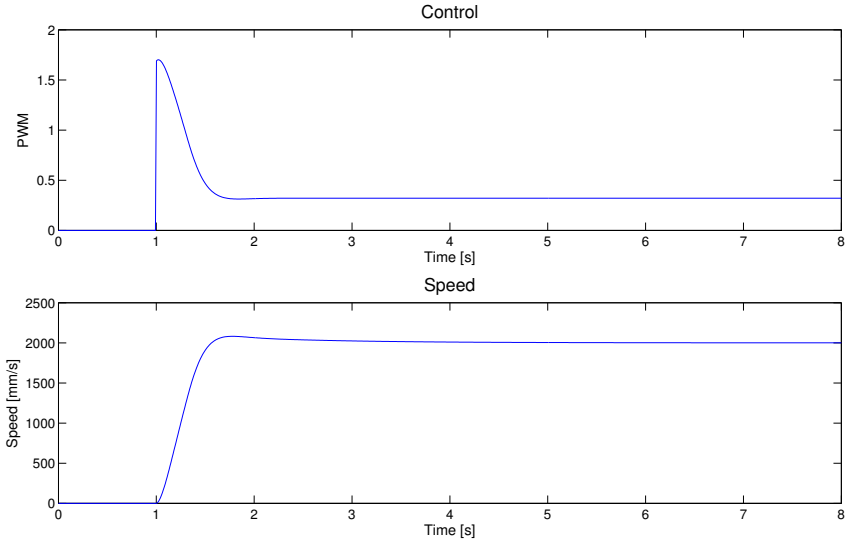


(b) *PI action*

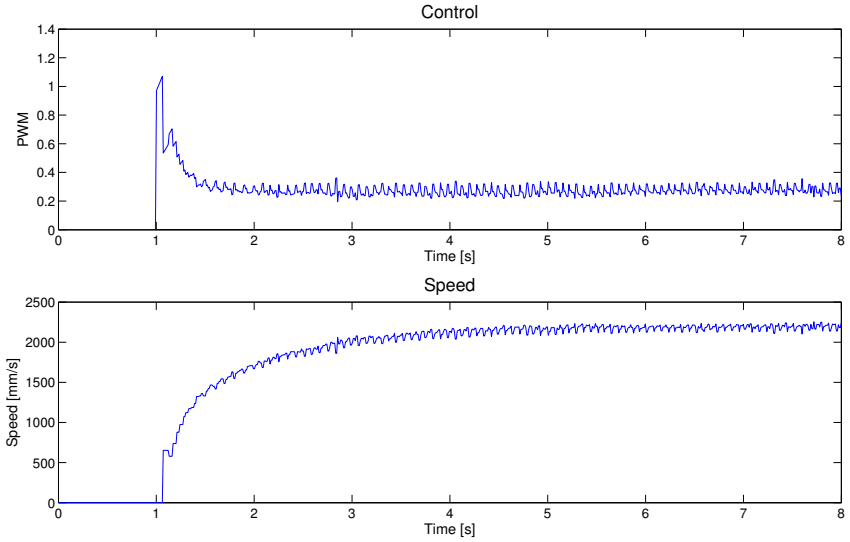
Figure 43: Bode stability margin.

6.4.2.4 Output signal and control signal

Once the controller is verifiably stable, it is time to see if the actual behavior matches the simulated one. Figures 44a and 44b show the command signal and output signal simulated and measured from the real system. The comparison is made using output obtained imposing a speed step of a fixed speed value 2000 mm/s. From the graphs obtained, it is evident that the real system has some differences between the simulated ones. A residual of steady state error is present, and the transient is a little slower than expected. For the purpose of this work, the application has been considered acceptable, but a modified controller may improve performance gains. Acting on the integral gain, the steady state error may be partially alleviated and proportional gains may increase overall performance.



(a) *Symulated*



(b) *Measured*

Figure 44: Output and control signals.

CHAPTER 7

DIFFERENTIAL IMPLEMENTATION AND RESULTS

Chapter.3, details the kinematics that lead to the realization of algorithms 3.24, 3.25, 3.26, 3.27, which computes speeds for posterior wheels in cornering; in other words, an algorithm this acts as a mechanical differential. This chapter shows the implementation of the algorithm using the model-based method and the results that this methodology obtains in terms of gains on the cornering behavior.

7.1 Implementation

The implementation of an algorithm, in a model based ambient, could be performed using the library provided by the software, linking many blocks together sequentially. That method could be chaotic at times and time consuming. Simulink® provides a very simple option and similar to an S-Function, which allows for the creation of custom functions in MATLAB® language. These functions are executed in each simulation as normal blocks are, and help in integrate the simple algorithm that requires many operations. This method has been used in order to create the algorithm block. In the block realization, the constant parameters are defined first. These have been introduced in a *data dictionary* file. It is related to a model and contains all constant values that the user stored. Thanks to this, each time the model is executed, the constant values are called from the data dictionary file. The constant parameters in this case are L,l_w,r_w,

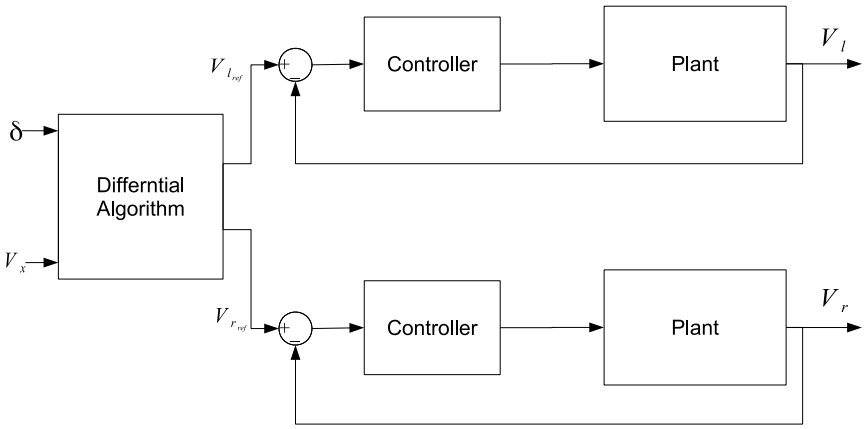


Figure 45: Block scheme of the system

presented in tab.I. Figure 45 shows the block scheme of the whole differential system, from the differential algorithm to the speed output.

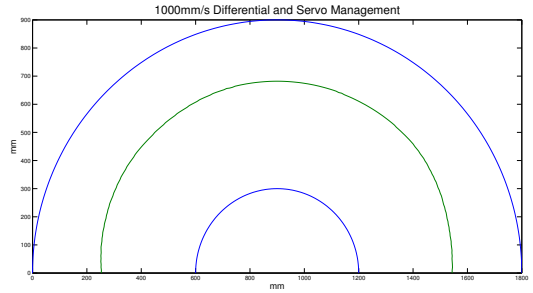
7.2 Results

The discussed differential system can now be now analyzed, given the results obtained by the vehicle behavior in cornering.

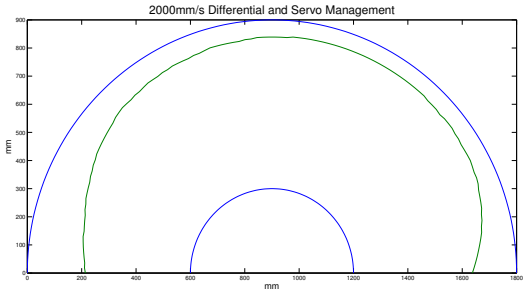
7.2.1 Differential with Servo management

The first test has been performed with the differential algorithm and the steering management system activated.

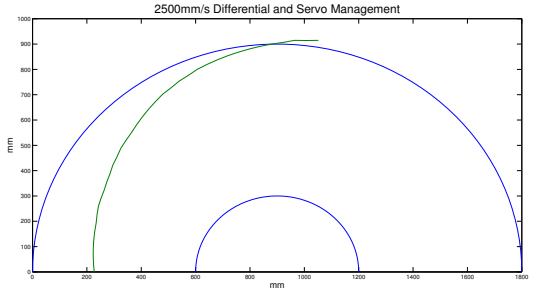
The graphs in figure 46 show that the differential action at high speed does not produce benefits to vehicle behavior. Indeed, the graphs 46c and 46d show that the vehicle trajectory ends at the track limits. The meaning of this behavior is that the vehicle is not able to corner, but the dynamic forces bring it out. For lower speed,



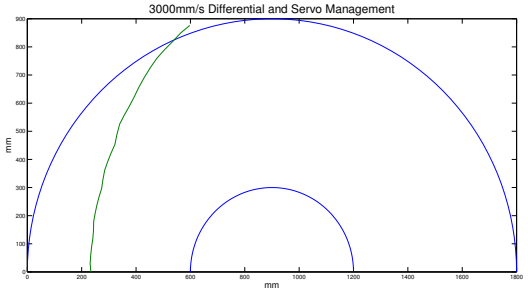
(a) *1000 mm/s*



(b) *2000 mm/s*



(c) *2500 mm/s*



(d) *3000 mm/s*

Figure 46: Differential action with servo management

TABLE XII: STEERING ANGLE EFFECTIVE VALUE

Reference angle	Steering left δ_l	Steering right δ_r	Mean δ
10.00°	11.00°	12.00°	11.50°
15.00°	17.00°	19.00°	18.00°
20.00°	25.00°	27.00°	26.00°
25.00°	30.00°	34.00°	32.00°
30.00°	35.00°	44.00°	39.50°

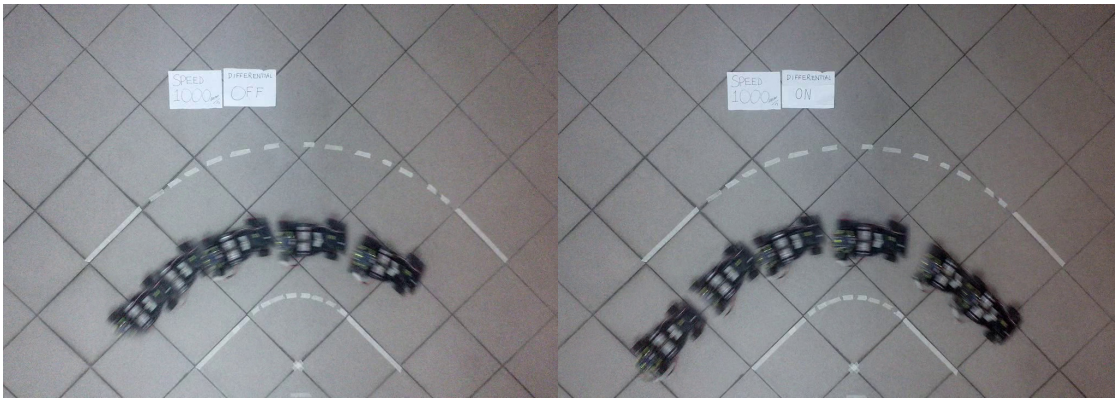
graphs 46a and 46b show that the action works well. The servo management is active, preventing to analyze the behavior strictly related to the differential action.

7.3 Differential with fixed angle

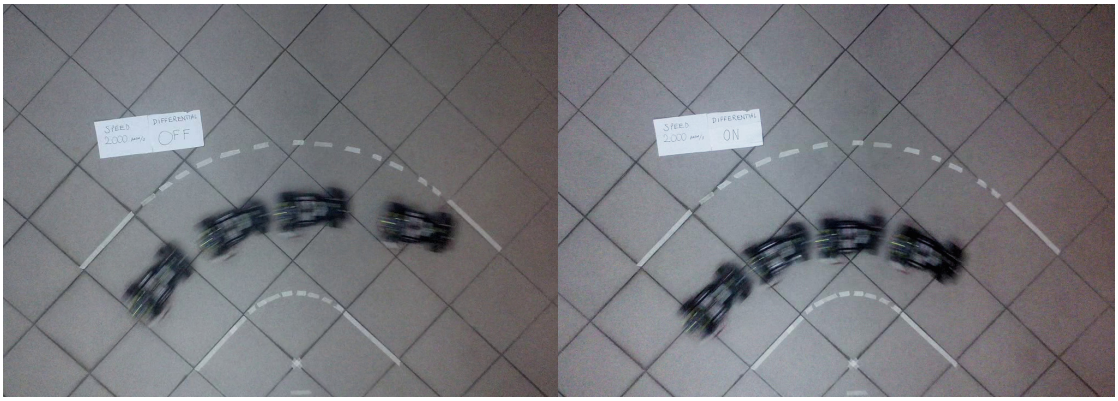
This test analyzes the behaviors of the vehicle in steering conditions, with a fixed steering angle. To evaluate the effective configurable angle for the servo, the differences between the expected steering angle and the output effective angle is observed and calculated. Table XII reports the assigned and the measured value. From these values, a gain factor that corrects the output angle is calculated. The gain value is $K_\delta = 1/1.2493$ and it has to be used to compute the servo reference value $\delta_u = K_\delta \cdot \delta_{ref}$. This assures that the real angle will be nearer to the reference value assigned, δ_{ref}.

Next, the gain for compensating the steering angle error have been evaluated, and a value of 18° has been chosen in order to assure a fixed curvature radius of 60 cm.

The action of the differential, with only fixed angle, is visible from the results in all cases studied, by the addition of differential action. In all figures, the differences between

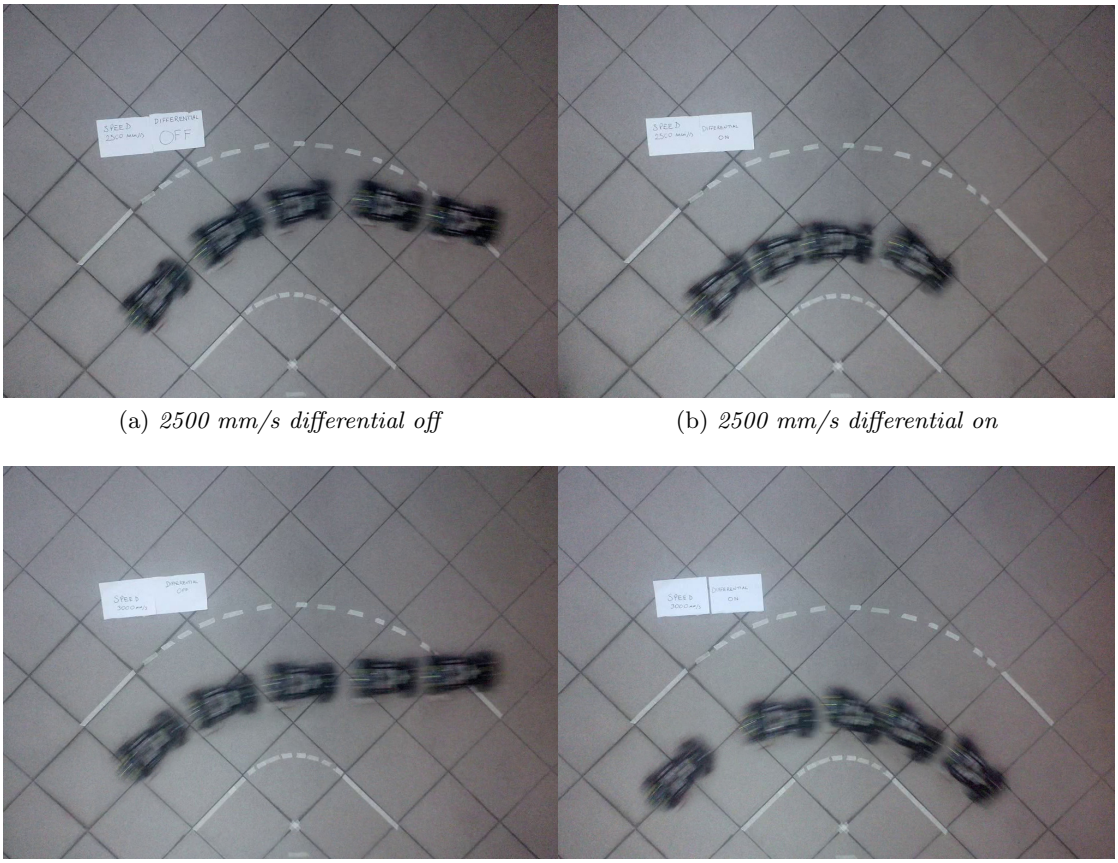


(a) *1000 mm/s differential off* (b) *1000 mm/s differential on*



(c) *2000 mm/s differential off* (d) *2000 mm/s differential on*

Figure 47: Differential action with fixed angle part 1



(a) *2500 mm/s differential off*

(b) *2500 mm/s differential on*

(c) *3000 mm/s differential off*

(d) *3000 mm/s differential on*

Figure 48: Differential action with fixed angle part 2

the behavior with and without the differential is visible. In all cases the vehicle was able to complete the corner.

7.3.1 Conclusion

In the case studied, the cornering behavior of the system has been improved by the addition of the differential action. Better results have been obtained in tests with a fixed angle. This is reasonable, as there exists a basic of incongruence between the steering angle management and the differential algorithm, as the former has been designed with a different objective. A re-design of the steering management in order to assign the angle with a different criteria might also be considered.

CONCLUSION

This work has proposed a model based solution to a design problem. In particular, have a differential algorithm that reproduce the behavior of a mechanical differential on the electric 2WD vehicle has been proposed. The implementation and testing of the algorithm required many intermediate components to be induced in order to constitute a closed-loop chain. The most important addition is the speed sensor, which has been realized by interfacing the board peripherals with the model-based ambient. After that, a control system has been developed and implemented in order to test the differential algorithm. As seen, the algorithm provides benefits on the behavior in steering condition of the vehicle, and these improvements are more evident with a fixed angle. The interest of this thesis goes beyond the simple realization of the algorithm, as the heavy part of the work has been in realizing the speed sensor driver, which required new language study and communication with the software engine. Another crucial point that has been underlined in this work is the choice of a model-based design . The simplicity and rapidity that characterize the realization of a controller with model based tools has been emphasized. This example is only one of the many cases in which model based design helps system design. In general the ease of use presented by the graphical interface is a huge point in favor of the model based design. For sake of completeness, implementation of the system requires several considerations. Many parts have been put together during the development. In that process, several issues arose, and some have been solved, while others have not been resolved practically. One of these unresolved issues occurs during

the time step, which defines the working frequency of the system. That parameter should be elevated in term of frequency (and obviously very small in term of period). In the case of this study, this options was not possible, as many of the systemic limits were not immediately evident. Going deep in the problem it has been discovered that the system was slowed down by the *Line Scan Camera* block, provided by the *FRDM-kl25z support package*. For that reason, the working frequency of the system has been limited to 100 Hz. That limit is extended to the controller (a dramatic limitation) and all parts of the system.

Future Works

When approaching problem holistically, it is normal to detect some aspects that could be improved. The principal modification and enhancements that could be done are:

- Steering management: as seen from the final tests, the algorithm used for assigning the steering angle does not well join with differential algorithm. A re-design of the steering management could solve that incompatibility, improving more the benefits of the differential.
- Steering angle control : correlated with the previous point it could be a good enhancement to realize a control of the steering angle.
- Filtration of the speeds: from the graph of the output speed values, the noise presence is evident, can be, at times detrimental to the system. A filtration of the output signal could improve the control performances as well as the differential of the system.

- Yaw rate control or Electronic stability: in order to assure better cornering behavior it would be advisable to realize a controller of the lateral dynamic, controlling the yaw rate or realizing an electronic stability control that acts not only on the yaw rate but also considers the slip angle.
- Speed sensor improvement: A strong limit introduced by the sensor his the absence in the implementation, of the sign detection.
- Controller improvement: The controller effect, as previously stated, is not highly precise in output. An additional benefit could be reached, working on the controller, in order to have better performances and more precision.

Many other improvement could be done, but the principals, in my opinion, are the exposed ones. In the end this thesis works it is concluded but opens many doors to futures works, with the aim to improve performances of the vehicle and face new cases of interest.

CITED LITERATURE

1. ed. U. G. printing office How autonomous vehicles will shape the future of surface transportation: Hearing before the subcommittee on highways and transit of the committee on transportation and infrastructure house of representatives. United States Congress House Committee on Transportation and Infrastructure Subcommittee on Highways and Transit, 2014.

2. Futaba: Futaba s3010 - standard high-torque bb servo.

3. TAOS: TSL1401RLF 128 1 LINEAR SENSOR ARRAY WITH HOLD, 2007.

4. Freescale: Freescale cup shield for the freedom kl25z.

5. Freescale™: KL25 Sub-Family Reference Manual, 2012.

6. Mathworks: Freescale frdm-kl25z microcontroller support from embedded coder.

7. Rajamani, R.: Vehicle Dynamics and Control. Springer US, 2nd edition, 2012.

8. The MathWorks Inc.: Simulink® Developing S-Functions, 2015.

9. The MathWorks Inc.: Simulink® Coder™ Target Language Compiler, 2015.

10. Pacejka, H. B.: Tyre and Vehicle Dynamics. Butterworth-Heinemann, 2nd edition, 2006.

11. Ljung, L.: System Identification Theory for the User. Patience Hall, 2nd edition, 1999.

12. Åström, K. J. and Hägglund, T.: Advanced PID Control. ISA Instrumentation Systems and Automation Society, 2006.

13. Allegro®: A1230 Ultra-Sensitive Dual-Channel Quadrature Hall-Effect Bipolar Switch.

14. Freescale™Semiconductor: H-Bridge with Load Current Feedback.

CITED LITERATURE (continued)

15. Chen, Y. and Wang, J.: Design and evaluation on electric differentials for overactuated electric ground vehicles with four independent in-wheel motors. IEEE Transactions on vehicular technology, 2012.

16. Snider, J. M.: Automatic steering methods for autonomous automobile path tracking. Technical report, The Robotics Institute Carnegie Mellon University, 2009.

17. Madarás, J., Bugár, M., and Danko, J.: Driving dynamic of an electric vehicle with electronic differential assisted drive. Scientific Proceedings Faculty of Mechanical Engineering STU in Bratislava, 2013.

18. Wu, X., Xu, M., and Wang, L.: Differential speed steering control for four-wheel independent driving electric vehicle. IEEE International Symposium on Industrial Electronics, 2013.

19. Kelly, A.: A vector algebra formulation of kinematics of wheeled mobile robots. Technical report, The Robotics Institute Carnegie Mellon University, 2010.

20. Everitt, B. S. and Skrondal, A.: The Cambridge Dictionary of Statistics. Cambridge University Press, 1998.

21. Bona, B.: Dynamic Modelling of Mechatronic Systems. Celid, 1st edition, 2013.

22. Uicker, J., Pennock, G., and Shigley, J.: Theory of Machines and Mechanisms. Oxford University Press, 2003.

VITA

NAME	Gabriele De Matteis
EDUCATION	
Dec 2015	Master of Science in Mechanical Engineering, University of Illinois at Chicago, USA
Oct 2015	Master of Science in Mechatronics Engineering, Polytechnic of Turin, Italy
Oct 2013	Bachelor's Degree in Mechanical Engineering, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2013 - IELTS examination (6.5/9)
	A.Y. 2014/15 Six Month of study abroad in Chicago, Illinois
	A.Y. 2013/14. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Fall 2014	Italian scholarship for TOP-UIC students
TECHNICAL SKILLS	
Advanced level	Model-based Design
Intermediate level	System identification
Advanced level	Fluid automation
Basic level	PLC programming
Basic level	Logistic
Intermediate level	Control system
Basic level	Digital filter
Advanced level	C programming
Advanced level	3D Cad

VITA (continued)

Advanced level	MATLAB/Simulink

WORK EXPERIENCE AND PROJECTS

Oct 2014 - Dec 2014	Digital filters design: Wiener filters, adaptive filters, linear predictors
Mar 2014 - Jun 2014	HIL control of a robotic arm: realization using Simulink, Simmechanics, Matlab, Solidworks, ArduinoUNO
2014	Design and simulation of electronic systems: BJT-based amplifier, Low-pass active filter, R/2R D/A converter, Counter A/D converter, Transmission lines, Non-linear circuits, FPGA design and simulations using VHDL
	Design and simulation of fluid automation systems: pneumatic cylinders and valves, basic command and pneumatic circuits, pneumatic sequencer, electro-pneumatic systems, PLC
2013	Industrial plant design: internal transport, supply transport, machinery, shelving units
2013-2014	Other Experiences:
	Design of gear box