UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

Model-based development of a self-balancing, two-wheel transporter

Relatore: Roberto Oboe

Laureando: Dino Spiller
1084324-IMC

ANNO ACCADEMICO: 2017-2018

SUMMARY

The current work aim to propose an evolute theoretical model for the *two-wheel self-balancing transporter*, together with a practical implementation of the device, that serves as proof for the model itself.

Inside this document the mathematical modeling, the control strategies (LQR for the equilibrium and PI for steering) and the management the system non-idealities that affects the real-device.

An automated powerful tool was realized and described, to make it possible the easy compare of simulation and real-device performances/behavior. It can be considered a useful tool for the proof of future studies and system improvements.

The results of this work are a good correspondence between the theoretical model simulation and the real device behavior, in addition to the good stability performances of both the simulated and real device.

Perché la vita é un brivido che vola via,
é tutto un equilibrio sopra la follia.
(Vasco Rossi)

THANKS

To *Prof. Roberto Oboe*: thanks for all your kindness and support, especially for encouraging me in the moment when I believed I couldn't ever make the device work. You made the things "lighter", with your sympathy and that finally helped me. Sometimes I think I'd love to be like you.

To *Simone*: you were not simply a colleague. You always had the patience of listening me when I told you my difficulties about this project and even gave me useful keys to see difficult situations from the right perspective.

To *Maria, my mother*: you always believed in me and see something that sometimes even I couldn't. I'm your seventh son, but you always made me feel like I was the only.

To *Antonio, my father*: thanks for your example. What I learned from you is the love for the honesty and that for the study: an activity you never stopped in every moment of your life.

To *Nicola and Andrea*: you are the "sun" of my life. You're one of the main motivation of this work: it's a toy for you! I will never forget the times passed with you sleeping on my knees, when I studied and the times you told me "don't give up, papa!". I wish you'll be blessed by equally lovelies children.

And over all, to *Michela*: It's you I miss, every day, when I can't wait to be back at home. This work is only the "iceberg top" of what you helped me to do these years of university: your patience, your encouragements and your love made it possible the completion of this hard work. I'll always remember the moments we had a break together, making our "terrazza-parties": pleased by the aroma of your presence, of your beauty and your love, that is even better than that of the coffee we had together.
Without you none of the words in this story might have been written... even those in this book.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

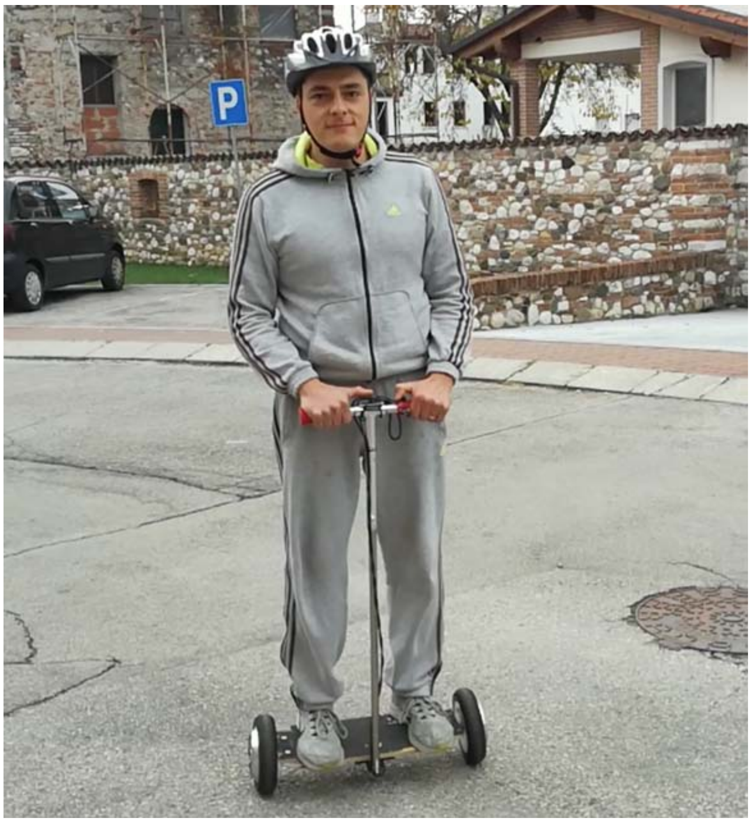


Figure 1: Me, while riding the self-balancing vehicle described in this thesis.

A self-balancing scooter (also "hoverboard", self-balancing board) is a self-balancing personal transporter consisting of two motorized wheels connected to a pad on which the rider places his feet and stands up. The rider controls the speed by leaning forwards or backwards, and direction of travel with a steering command.

The self-balancing feature is the result of a complex computer algorithm that stabilizes the under-actuated system formed by the person standing over the vehicle. The person can "perturb" this system by leaning forwards or backwards and making the controller try to stabilize it again, with the consequence of back or forward movement.

In recent years, the two-wheeled self-balancing vehicles have been recognized as a powerful personal transporter and commercial versions like *Segway PT* [37] are available in the market since the year 2001.

Another successful newer example is the -so called- *Hoverboard* [33]: a confirmation of the interest in this type of vehicle as a companion in the everyday life. Probably considerable as an evolution of the first, it has introduced some interesting news in the drive mechanism such as the revolutionary absence of the drive-bar. Paying the cost of a lower-usability, this has indeed, the advantage of being small, lightweight and portable.

Both those vehicles, together with many other similar alternatives are derivations of the *self-balancing robot*: a non-linear multi- variable and naturally unstable system.

Controlling such a system is a challenge, therefore it attracts attention of many modern control researchers, included the author.

Control stability, robustness and safety have been studied over the years both by manufacturers of such vehicles and by the research community.

1.0.1 *Existing literature*

Many studies concerning the control of the two-wheel self-balancing transporter (*TWSBT*) have focused on different aspects of the problem.

Some studies have focuses on the problem of modellization: starting from balancing a single-wheel transporter (only 2-DOF) [8], to a more-complex problem, which is the decoupling of the two-wheel robot into two subsystems that are the balancing and the Yaw angle subsystems [39] and [15] .

Other studies like [16],[12] and [3] have focused on the comparison between balancing control techniques, attempting to find the stability, points-of-force and limits of different techniques, like PID, LQR, LQG.

Some other studies like [7] and [34] have focused on the solution of self-balancing problem with alternative methods like fuzzy logic and neural networks.

Further interesting materials have been found, concerning some less-conventional themes about this problem: the model of the interaction between human and the self-balancing vehicle [20] and [19], the problem of the security in case of necessity of sudden braking [23] and finally some studies on the sensor-fusion scheme, for the accelerometer-gyroscope pitch angle detector [25].

1.0.2 *Motivation and attempts*

The first motivation of this thesis' argument is the love for my son Nicola: it is a way to build "a self-balancing toy" for him. At the same time, the TWSBT problem is probably the most powerful and economic way to test the knowledge of the theoretical concepts learned in the Mechatronic Engineering master degree course. The realization of such "toy" was the way to face in a rigorous way arguments like power electronics, mechanics, system and motion control, both in theory and in practice, with the real-world problems.

The success of this study, and the quality of the work performed might become in a short time a starting point for a high-quality open-source project, where advanced control techniques and high quality hardware might be used or simply studied by the community.

The intent of this thesis is the building of a reliable model, with a high-performance controller, differently from the popular PID, used in many hobby projects.

The analysis of the literature just exposed, served as inspiration for the successful implementation of control strategies, using methods and models proposed by such studies.

The device model, control scheme and the algorithms were simulated using Matlab and Simulink, but during the thesis a real prototype of the vehicle was realized, using state-of-the-art electronics boards taken from an open-source electronic project [35], equipped with a real-time embedded operating system, a couple of motors and wheels and other home-made hardware.

This prototype served as a test-bed for the proof of the model the control algorithm performances.

1.0.3 *Thesis outline*

In chapter 2, a complete description of the system requirements and the consequent component chosen.
In chapter 3, the description of the derivation of the dynamic model for the self-balancing, two-wheel vehicle. Also in this chapter, the block-model derived from equations, useful for the successive simulation of the vehicle.
In chapter 4, the description of an evoluted model, where differently from the first, the rider has a freedom degree against the angle of the base. Even here a block model was built: the model's parameters regarding the rider and its behavior, were derived from physiological studies.
In chapter 5 and the control schemes adopted, together with the motivations that drove the choices, the stability check and the simulation of the control scheme applied to the model.
In chapter 6 a discussion about the non-idealities of the system, whose presence has a direct consequence on the controlled system. They are important since they made the difference between a pure-theoretic model and a real-one, and also their study made it possible to design a more-realistic model and do better-tune of the controller.
In chapter 7 the practical implementation of the algorithms: the software structure, the algorithm code, and the telemetry system.
In chapter 8 the validation of the model which is essentially the comparison between simulation blocks' behavior and real-world prototype.
In the last chapter, the conclusions and future work.

2 DEFINITION OF SYSTEM SPECIFICATIONS

2.1 INTRODUCTION

The current thesis aims to design a self-balancing human transporter and to test on it some strategies about control (discussed in the successive chapters).
The goal of this project is to implement it by using a simple and robust architecture, using as less components as possible, possibly cheap.
In such a way, the whole work can be converted in an open-source project, useful for hobbyists, students and everyone that needs an open-source platform for testing self-balancing control strategies.
In this chapter it will be discussed the specifications of the project and then the calculus that drove the choices about components.

2.2 PROBLEM DEFINITION

The human transporter being designed needs to verify the following expected features:

- The vehicle must carry persons, so the dimensions of the vehicle must take it into account
- Maximum weight to transport: $M_{MAX} = 90kg$
- Maximum Velocity in horizontal ground: $V_{MAX} = 15km/h$
- Speed at full-weigth, 10% ground inclination: $V_{incl,MIN} = 10km/h$
- Minimum deceleration capability: $A_{dec,MIN} = 0.5m/s^2$
- Battery autonomy (under normal-use): $W_{batt,MIN} = 1hour$

All the previous specification are the drivers for the components choice.

2.3 DIMENSIONS

As for that in the specifications, the transporter has to carry persons. The choice is that of making the vehicle usable, but as compact as possible.
Using a simple table, a meter, and by making simple usability tests , it was chosen to have a base width of about $0.4m$. This will translate in a wheel-to wheel distance of about $D = 0.45m$. In order to use cheap, non-custom materials, the base structure was done using a skateboard table, and reducing its width to $0.4m$. This because souch tables are cheap, but made-up with high-performance and lightweight materials, capable of carrying up to $100kg$ even under vibrations and jumps.
The other motivation about the use of skateboard table is that their geometry is familiar to the people, so it is a kind of warranty in the usability of the vehicle.
The use of skateboard table as base, fixes the base length to that of the standard

tables: $L = 0.2m$. This implicitly defines approximately the dimension of the wheel (its diameter must be approximately equal to the base length). The wheel and motor choice will be discussed later, but the important thing is that the wheel diameter is chosen to be approximately equal to $L = 0.2m$, thus the wheel radius is half of this: $r_w = 0.1m$. The height of the base is that of the skateboard table ($H = 0.01m$), the space for the battery is under the skateboard table, so the choice of that was made taking this into account.

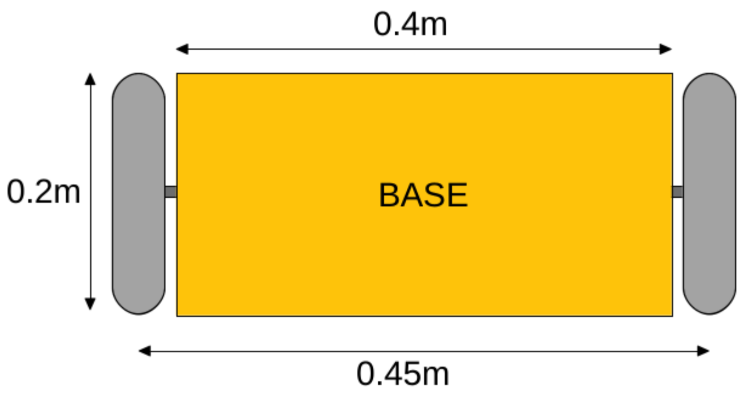


Figure 2: Base dimensions (observed from upside)

2.4 MOTOR TORQUE

For the definition of motor torque needs, two requirements holds. In the following sections the discussion about these:

2.4.1 *Speed at full weight, inclined ground*

This specification gives a precise indication about motor torque needs as long as motor power need. About the first, let's consider the situation as an application of the inclined plane (see figure 3).

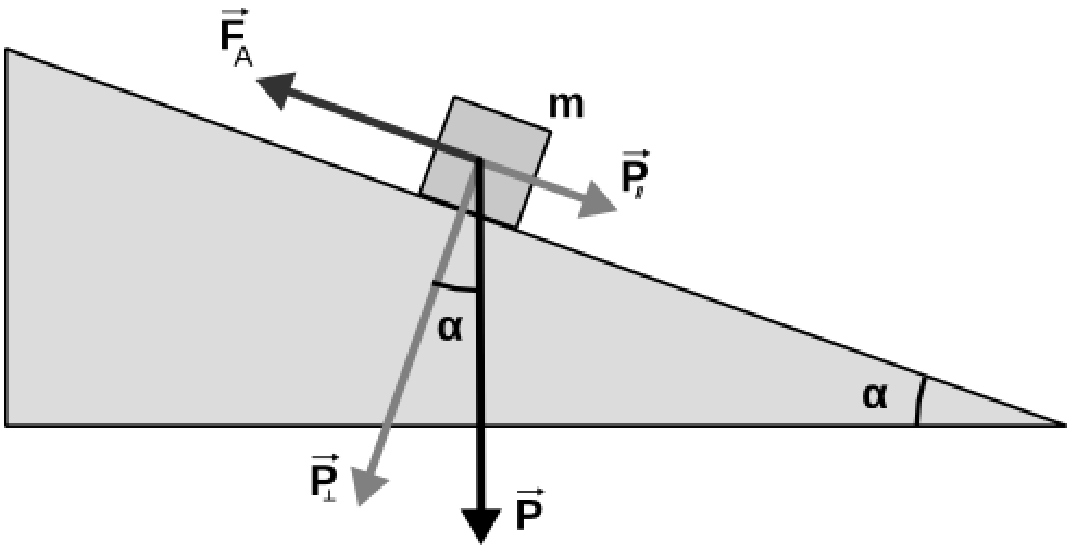


Figure 3: The requirement of a certain minimum speed at full weight, along an inclined plane. Figure from [38].

The force P is given by:

$$P = m * g$$

Thus, considering the critical condition, let's apply the maximum weight $M_{MAX} = 90\text{kg}$:

$$P_{MAX} = M_{MAX} * g = 883\text{N}$$

Since the maximum inclination specified is 10%, which corresponds to an angle of $\alpha = 5.71°$, the resulting weight-force, acting parallel to the plane is:

$$P_{\|MAX} = P_{MAX} * \sin(\alpha) = 883\text{N} * 0.997 = 87.86\text{N}$$

Having the wheel a radius of $0.1m$, as stated in the previous section, and having two motors, the torque need is:

$$\tau_{MAX} = P_{\|MAX} * r_w/2 = 87.86\text{N} * 0.1m/2 = 8.79\text{N}m/2 = 4.39\ [\text{N}m] \quad (1)$$

At the same time, it is possible to derive the power need for the motor: supposing slip absence, the $V_{incl,MIN}$ specification translates into:

$$\begin{aligned} V_{incl,MIN} &= 10km/h = 10/3.6m/s = 2.78\ [m/s] \\ &\text{thus:} \\ \omega_{w,inclinMAX} &= \frac{V_{inclinMAX}}{r_w} = 27.8\ [rad/s] \\ &\text{defining a power-need of:} \\ P_{mot_{min}} &= \tau_{MAX} * \omega_{winclinMAX} = 122\ [W] \end{aligned} \quad (2)$$

2.4.2 *Maximum deceleration*

The other requirement that defines the motor torque need is the maximum deceleration capability. By ignoring the equilibrium problem, let's apply the D'alembert principle to the transporter:

$$\begin{aligned} F_{dec,MAX} &= M_{MAX} * A_{dec,MIN} \\ &\text{thus:} \\ \tau_{dec,MAX} &= F_{dec,MAX} * r_w/2 = 90kg * 0.5m/s^2 * 0.1m/2 = 2.75\ [\text{N}m] \end{aligned} \quad (3)$$

In this last equation it is possible to observe that the requrement is less-strict than that of the inclined plane, thus, using the torque calculated for the inclined plane, it is possible to obtain stronger decelerations:

$$A_{dec,MAX} = \frac{\tau_{MAX}*2}{M_{MAX}*r_w} = \frac{4.39*2}{90*0.1} = 0.976\ [m/s^2] \quad (4)$$

2.4.3 *Motor choice*

The previous requirements are theoretical, thus a consistent margin of about 2 is taken:

The other important requirement of the motor (as it will be clear in the successive chapters), is the presence of a position-feedback. In the case of a DC motor, it

Motor Torque needed	8	[Nm]
Motor Power needed	250	[W]

Table 1: Motor Requirements (with a margin of approx. 2)

is often used an encoder, but since the application does not need a high-precision positioning, it is possible to use simple BLDC motors, with HALL sensors incorporated. The hall-sensors are very important since the motors, during the standstill self-balancing condition, will work almost in steady position (not generating enough BEMF) and sensor-less driving is not possible with simple drivers. Given the previous, two BLDC alternatives are considered:

1. Hub motor (direct-drive motor, incorporated into the wheel)
2. Small RC motor with gear-reduction

2.4.3.1 *Hub motor*

The hub-motor is that used in many modern self-balancing toys. Its characteristics are:

- maximum voltage = $36V$
- maximum current = $10A$
- maximum power = $360W$
- Ke=Kt= $0.694Nm/A$
- maximum velocity = $30km/h$
- motor configuration = $27N, 30P$

This motor has three built-in hall sensors, thus allowing to have a position-reading definition of 3 hall-sensors * 30 poles $= 90$ steps of position reading for every wheel rotation.

Figure 4: HUB motor, used in many self-balancing toys. Figure from [10].

The "pros" and "cons" of that solution are:

pros	**cons**
robust construction , with wheel incorporated	high weigth (2.5kg)
silent and very-regular torque	expensive(approx. 70 €)
low current need	high voltage need

Table 2: Pros& cons of hub motor

2.4.3.2 *RC motor*

The RC motor is a kind of motor used in RC models. It has been considered because of its high torque and high power density, in a very lightweight device. The drawback of this solution is the need to design a wheel and a reduction system (of about 1:6), but here are reported the characteristics:

- maximum voltage = $29.6V$
- maximum current = $80A$
- maximum power = $2200W$
- Ke=Kt= $0.035Nm/A$ (considering a $KV = 270RPM/V$)
- motor configuration= $12N, 14P$

This motor has three built-in hall sensors, thus allowing to have a position-reading definition of 3 hall-sensors * 14 poles = 42 steps of position reading for every motor rotation. Supposing a gear-reduction of about $1:6$, the total steps of position reading for every wheel rotation is: $42 * 6 = 252$ steps.

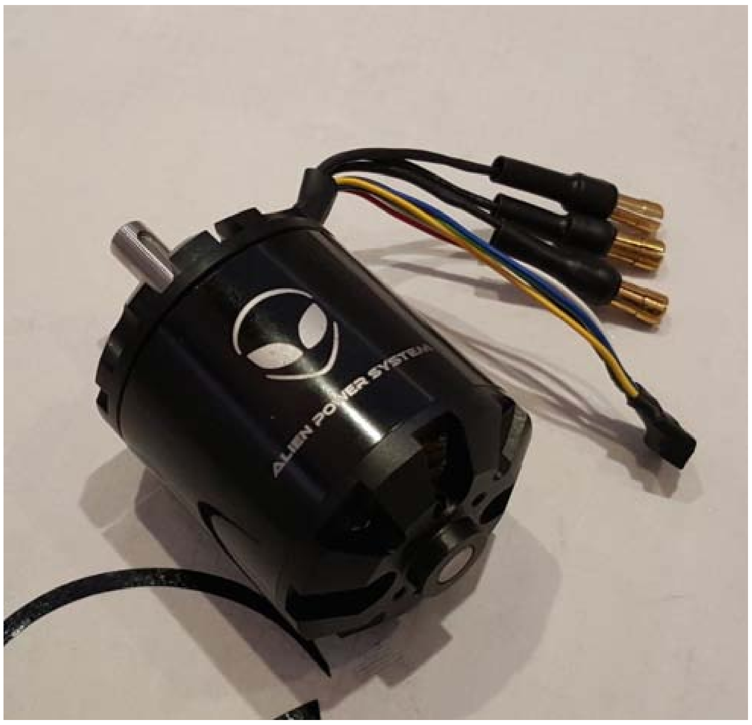


Figure 5: RC model motor, image from [1]

The "pros" and "cons" of this solution are:

For the purpose of the project it was chosen to use the **hub motor**, because it is very simple to mount it into the structure and since there wasn't the possibility to

pros	cons
very lightweight (320g)	need for external gearbox and wheel
cheap (approx. 30 €)	very high current needed

Table 3: Pros& cons of RC motor

prototype complex gearboxes/wheel, the RC motor was not used.
Nevertheless, since the RC motor represents a very-lightweight solution and also cheap, even if not used in our prototype, it was considered into the simulations and for the possibility of using it in cheap open-source projects.

2.5 BATTERY

For the choice of the battery, it is requested to have an autonomy of about 1h in "normal use". The goal is to define what is the power consumption under normal use. It is reasonable to suppose an overall, average consumption of about 150W for the vehicle (consider that the braking is regenerative).
Another important thing to consider choosing the battery is that the needs are very different in the case of HUB motor and in that of RC motor. In this last case the current requirements are very difficult to accomplish: 80A is a high-current and imposes a charge/discharge capable battery, while with the HUB motor, a current of 10A is not critical, even for lead batteries.
Giving the previous it was chosen to use:

- 6S (22.2V) 30C discharge Li-Po battery for the RC motors (the discharge rate is over-sized, but it was the only model available)
- 12V Lead battery to put in series to the previous, for the HUB motors

The battery capacity (for the RC motor) needed is:

$$\begin{aligned} E_{batt,MIN} &= P_{vehicle,AVG} * 1h = 150\ [Wh] \\ \text{which means:} & \\ I_{batt,MIN} &= E_{batt,MIN}/V_{batt} = 150Wh/22.2V = 6.76\ [Ah] \end{aligned} \tag{5}$$

This requirement drives the choice of a Li-Po battery capacity of **22.2V, 8 Ah**.
For the Lead battery, a lesser capacity is admissible, thus choosing a 12V, 6Ah model.

Figure 6: Battery used for the high-current RC motor. Figure from [6].

The last requirement is the vertical dimension: the wheel radius is $r_w = 10cm$ is the maximum height admissible. The batteries chosen satisfied this dimension constraint.

2.6 MOTOR DRIVER

This is probably the key component of the device. It must meet many requirements:

1. possibly cheap and easily available
2. BLDC driving capability (many producers name it "ESC")
3. Hall sensor management (many ESC drivers operates only in sensor-less mode)
4. High current drive capability (possibly up to 80A)
5. High voltage management (36V is not a common value)
6. Fast response (in the following chapters it will result a loop time of 4-10ms)
7. Flexible and robust communication interface
8. Current control mode (many ESCs has only tunable velocity controller)
9. Management of different motor types

All the previous requirements were met with a controller developed in Open-Source by Benjamin Vedder [35].
This project is not a simple ESC controller: it is a complete Open Hardware high-quality project which has unique characteristics about the capabilities of tuning motor control parameters such as :

- tunable PWM frequency
- tunable current control gain
- tunable velocity speed parameters
- tunable startup PWM-boost parameters

All this in a fully-open and community maintained project, with another very-important feature: a user-forum for user's matters, such as tuning issues, problem issues, feature-request issues, best practice issues.

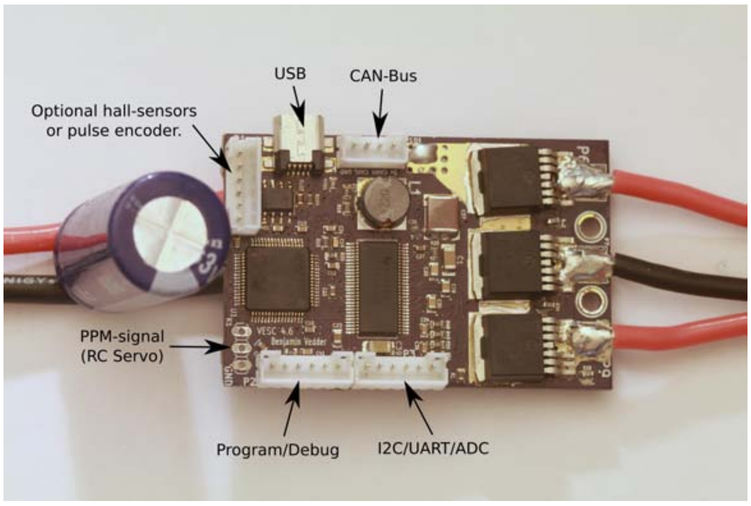


Figure 7: Open-hardware motor controller, a project of Benjamin Vedder [35] image also from [35]

In addition to the previous requirement meets, this device has several further point-of-force:

1. very powerful micro-controller (STM32F405), with embedded floating-point hardware accelerator
2. USB interface for motor diagnosis and configuration of motor characteristics
3. UART interface for external communication
4. High-speed CAN interface, useful for inter-board communication
5. I2C interface for external sensors interface
6. Management of HALL sensors and also ENCODER, with embedded calibration and position incremental output
7. Open-source software (easily customizable for integrating self-balancing control routines)
8. Uses an embedded real-time OS (Chibi-OS), very useful and easy to integrate suspensive routines and sync mechanisms

This solution was chosen because it has all the features needed for the control purpose, in a compact solution and with the unique possibility to customize the firmware, allowing the integration of control loop without external boards. The presence of an embedded real-time Operating System, made it easier to manage wait conditions, threads, mutexes and inter-task synchronization via events.

2.7 IMU

In this project there is the need to measure base inclination and rotation velocity. The most popular solution to this task is the use of a sensor-fusion scheme, based on accelerometer and gyroscope (as it will be explained in the following chapters). A component integrating these sensor is called Inertial Movement Units (IMU). IMUs are very popular (almost always MEMS-based) devices, available in the market with different features, sizes and interfacing system (analog, i2C, SPI...).

Giving that the micro-controller has an available I2C interface, it was chosen to use an IMU device with that interface, avoiding the problem of different ADC channels and bias management.
In the panorama of possible solutions, the choice was to use a single-component one, integrating both accelerometer and gyroscope and with an evaluation board compact and suitable for easy connection with out controller board. Another important thing is the choice of an IMU with ready-to-use libraries. This is very helpful for time-saving and error-avoidance.

Having the previous, the choice was the Invensense's MPU6050-based open-hardware module:

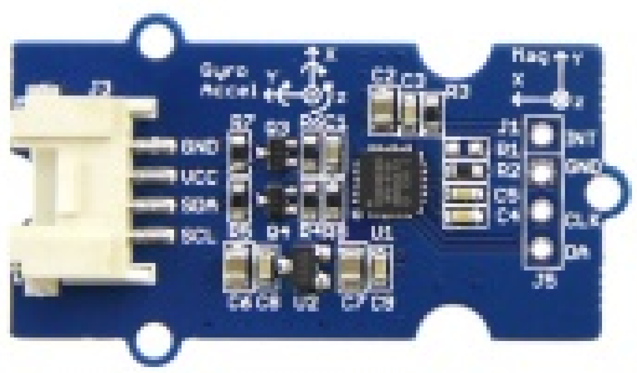


Figure 8: Open-hardware IMU i2C evaluation module of Invensense's MPU6050. Figure from [28].

Its features are:

- I2C interface
- 400kHz Fast Mode I2C for communicating with all registers
- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of $\pm 250, \pm 500, \pm 1000$, and $\pm 2000° / sec$
- Digital-output 3-Axis accelerometer with a programmable full scale range of $\pm 2g, \pm 4g, \pm 8g and \pm 16g$
- 3-axis silicon monolithic Hall-effect magnetic sensor with magnetic concentrator

In addition to the features needed, the chosen component owns a 3D magnetic sensor (often called "compass"), but it was never used in the application. Nevertheless, this feature might be an interesting and powerful one, in a future scenario, giving to the controller the possibility to manage a further sensor, useful for features like slip-detection or others.

2.8 JOYSTICK

The Joystick was hardly-used during the experiments: is was very helpful for driving the transporter back-and-forth, making rotations or simply standing, while collecting data. In the final prototype the joystick was replaced by a simple steering potentiometer. The Joystick used is a simple dual-potentiometer one, $10k\Omega$ in both axis

Figure 9: Joystick used during experiments. Image from [2]

2.9 REMOTE COMMUNICATION

During the experiments it was necessary to collect data in order to validate assumptions, algorithms and eventually to post-process real-life data. The choice was to do this in the most simple way: a serial-to-bluetooth board connected to the micro-controller's UART and sending data remotely to the host computer, via Bluetooth. The board selected is the HC-06:

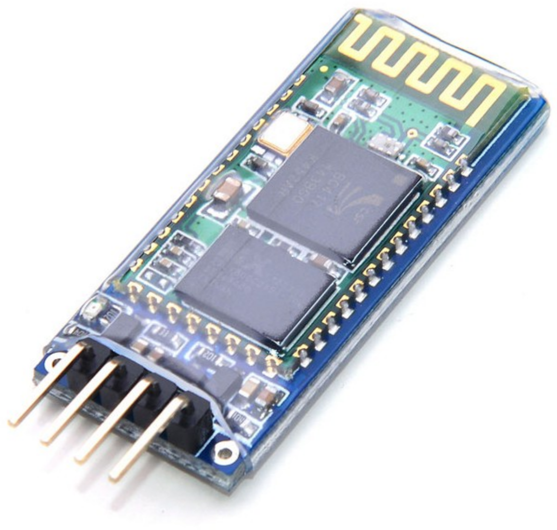


Figure 10: Serial-to-bluetooth (HC-06) module used for remote logging of data coming from the self-balancing transporter, to the host PC. Figure from [5]

It was chosen because of its very-interesting features:

- very cheap (sold as a component of Arduino project it is available even at only 3$
- very compact: only 2.7 cm x 1.3 cm
- fully compliant with Serial Port Profile (SPP), thus very easy to integrate with every operating system
- baudrate up to 115200bps (and over)
- UART level and Vdd=3.3V: the same as that of the controller board

The need to use a wireless module for remote communication/logging is obvious: having a moving device, it is strongly forbidden to use a wired communication scheme. The use of Bluetooth has the advantage to be embedded in almost every laptop (while other wireless standards such as zig-bee or others needs a counterpart USB key and drivers.

2.10 OVERALL ARCHITECTURE AND SHEMATIC

After having described the requirements, here it is described the overall connection scheme with the corresponding descriptions. The scheme of connections is visible in the figure:

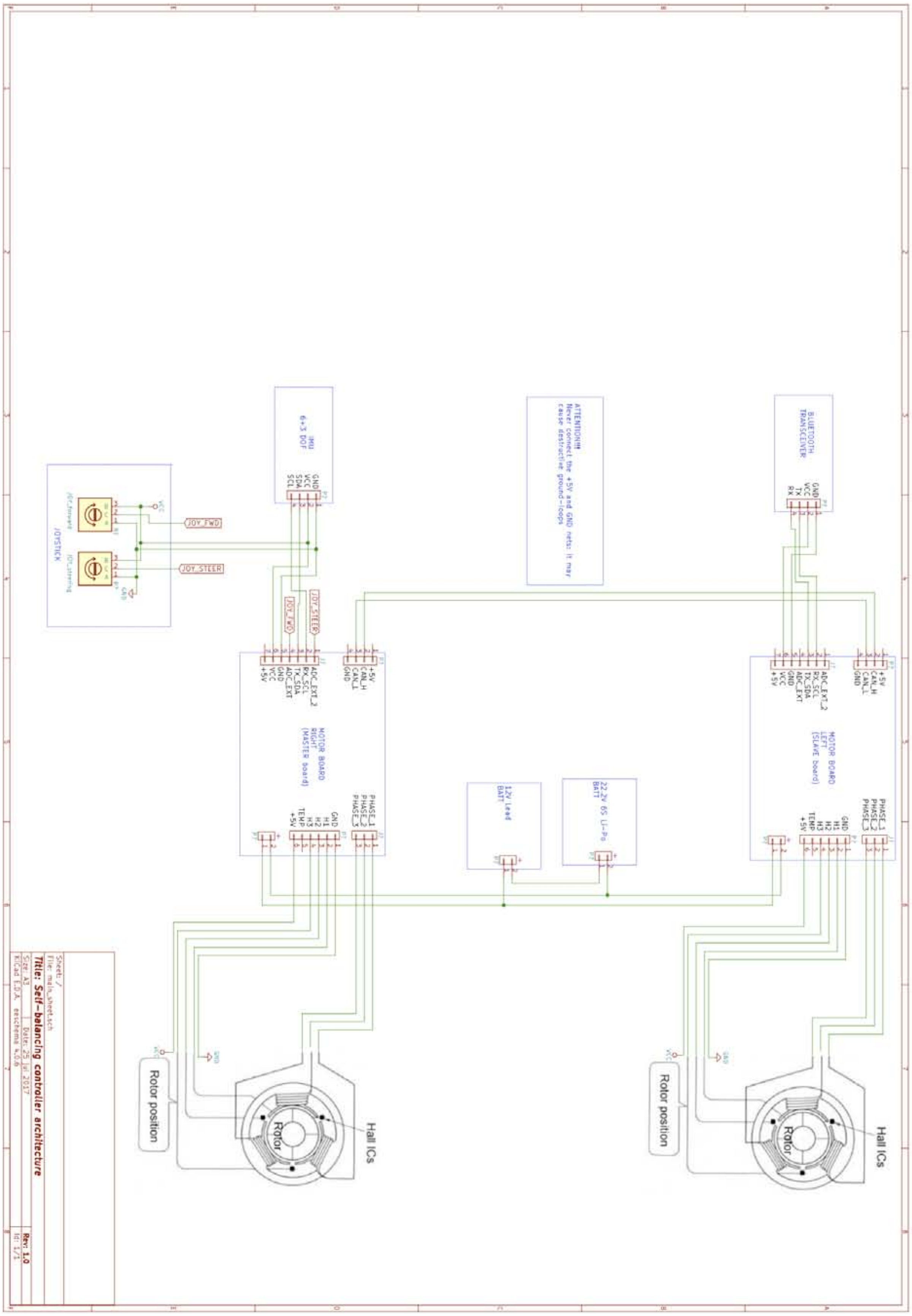

JOYSTICK
JOY_FWD
JOY_STEER
IMU 6+3 DOF
ATTENTION!!! Never connect the +5V and GND nets: it may cause destructive ground-loops
BLUETOOTH TRANSCEIVER
MOTOR BOARD RIGHT (MASTER board)
MOTOR BOARD LEFT (SLAVE board)
12V Lead BATT
22.2V 6S Li-Po BATT
Rotor position
Rotor
Hall ICs
Title: Self-balancing controller architecture
Rev: 1.0

Figure 11: Hardware architecture schematic of the self-balancing transporter. Note the absence of a "central board" and the choice of a master-slave architecture

The open-hardware and open-source feature of the controller board, makes it possible to integrate the self-balancing control algorithm directly inside the motor-control board, without the need for an external "super-part" board. In order to implement correctly the control, it is compulsory to communicate with the other board for:

- retrieve information about motor's rotor position
- set the desired current

Those needs drived the choice of a "Master-Slave" architecture, described below.

2.10.1 *Master board*

Master board is "the heart" of the self-balancing controller: in order to work, this board must collect all the available sensor's data and translate them into state variable's values.
Giving that this board drives (in real-time) one HALL-sensor-ed motor, it implicitly has the real-time info about its motor's rotor position.
The rotor's position of the opposite motor is available thanks to the inter-board communication via CAN bus: it is very fast (500kbaud/s) and robust against EMI interferences and data corruption. It makes it possible to design a fast control-loop, thanks to the fast variable update (approx. every 300 microseconds).
The board collects also the base inclination thanks to the i2C IMU attached to it. Even this interface is very fast (400kHz SCL), making it possible to have a very-fast angle readout (approx. every 100 microseconds).
The Joystick attached to this board is used as reference for steering and forward acceleration references. It doesn't need a very-fast update (like the balancing control-loop): it is read at a speed of about 10Hz, which is sufficient for reference speed.
Once the algorithm produces its output, the current to the Master board is imposed simply by an internal function, while that for the other board is set via a CAN-bus communication message.

2.10.2 *Slave Board*

The slave board duties are the feedback of its rotor position and the reception/actuation of the current reference. This board has another "unusual" duty: since the interfaces of the master-board were all occupied by the sensors, this board was used to communicate to host PC: the master board sends and receives UART commands to/from the PC via a "TUNNELLING" inside the CAN bus (a function utility already present in the original open-source project).
This last function obviously can't be too fast, since the communication has many "hops" to travel in: this is not a tragic drawback since that interface is used simply for sending state variable and command signal entities to the PC. This data send can be performed at a lower-rate than the loop speed, thus the choice was taken as suitable.

3 SYSTEM MODELLING

3.1 INTRODUCTION

The problem of controlling the Two-Wheel Self Balancing Transporter (TWSBT) starts with the modeling of the system. In this chapter it will be introduced the kinematic model, the dynamic model (using Lagrange approach) and the decoupling of the system in balancing and steering subsystems. At the end of the chapter it will be presented some considerations about the results and the block-models derived from the equations.

3.2 KINEMATIC MODEL

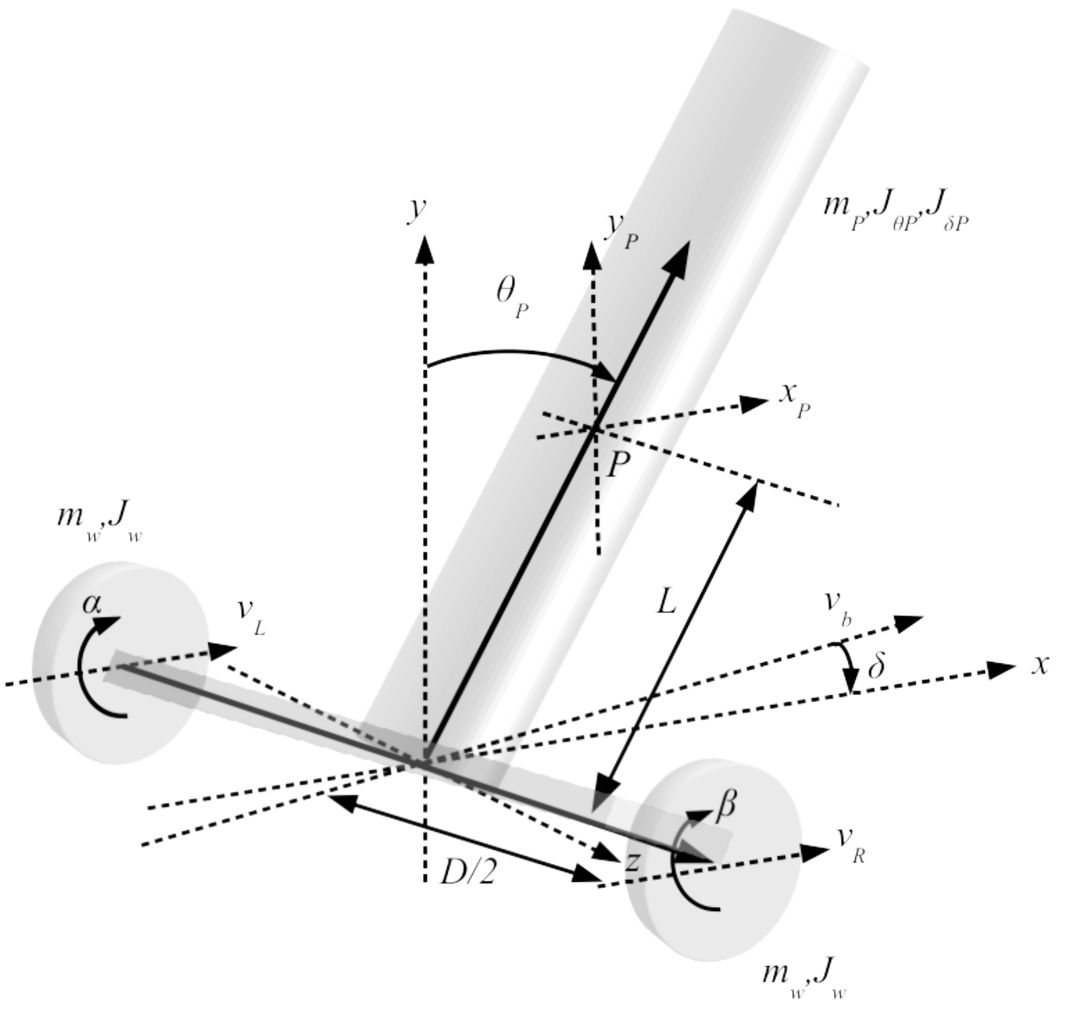


Figure 12: Graphical representation of the entities of the system.

The conventions used for this model are summarized in table 4.

m_P	mass of the rider	[kg]
m_w	mass of the wheel (identical in Left and Right)	[kg]
$J_{\theta P}$	inertia of the rider, referred to the pitch rotation	$[kgm^2]$
$J_{\delta P}$	inertia of the rider, referred to the yaw rotation	$[kgm^2]$
J_w	inertia of the wheel	$[kgm^2]$
α_m, β_m	angle of the (Left, Right) motor (referred to the base)	[rads]
α, β	angle of the (Left, Right) wheel (referred to the ground)	[rads]
θ_P	angle of the person (referred to the ground, where 0 is the vertical upper position)	[rads]
v_L, v_R	velocity of the (Left, Right) center of the wheel	[m/s]
x_b, v_b	horizontal position and velocity of the base center (origin)	[m],[m/s]
x_P, y_P, z_P	coordinates of the rider's center of mass	[m]
L	distance between base and center of mass of the rider	[m]
D	distance wheels	[m]
r	radius of the wheel	[m]
C_L, C_R	torques applied to the wheels, by the motor (after the gearbox)	[Nm]
ρ	reduction ratio between the motor rotation and wheel rotation	
τ_L, τ_R	torque of the (Left,Right) motor	[Nm]
ψ	viscous friction coefficient	

Table 4: Conventions

3.2.1 *Assumptions*

The following assumptions are used for the problem:

1. The friction is considered linear and proportional to the motor's rotation speed, even if different from reality
2. The rider is modeled as a rigid body (cylinder) of "2L" height
3. The efficiency of the gearbox is equal to 1
4. The reduction has no elasticity
5. The friction produced by the air, with the system's components, is neglected
6. The vertical coordinate of the base is taken as system's vertical origin

The relation between motor's position (referred to the rider's angle) and the wheel's angle (referred to the ground) is:

$$\begin{aligned}\alpha &= \theta_P + \rho\alpha_m \\ \beta &= \theta_P + \rho\beta_m \\ \dot{\alpha} &= \dot{\theta}_P + \rho\dot{\alpha}_m \\ \dot{\beta} &= \dot{\theta}_P + \rho\dot{\beta}_m\end{aligned} \tag{6}$$

The congruence equations for the wheels and base are:

$$\begin{aligned}v_L &= r\dot{\alpha} \\ v_R &= r\dot{\beta} \\ v_b &= \frac{v_L+v_R}{2} = r\frac{\dot{\alpha}+\dot{\beta}}{2} \\ \dot{\delta} &= \frac{v_L-v_R}{D} = r\frac{\dot{\alpha}-\dot{\beta}}{D}\end{aligned} \tag{7}$$

while, concerning the rider's center of mass:

$$\begin{aligned}x_P &= x_b + L\sin\theta_P\cos\delta \\ y_P &= L\cos\theta_P \\ z_P &= z_b + L\sin\theta_P\sin\delta \\ \dot{x}_P &= \dot{x}_b + L\dot{\theta}_P\cos\theta_P\cos\delta - L\dot{\delta}\sin\theta_P\sin\delta \\ \dot{y}_P &= L\dot{\theta}_P\sin\theta_P \\ \dot{z}_P &= \dot{z}_b + L\dot{\theta}_P\cos\theta_P\sin\delta + L\dot{\delta}\sin\theta_P\cos\delta \\ v_P^2 &= \dot{x}_P^2 + \dot{y}_P^2 + \dot{z}_P^2 = \\ &= v_b^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P[\dot{x}_b\cos\theta_P\cos\delta + \dot{z}_b\cos\theta_P\sin\delta] + \\ &\quad +2L\dot{\delta}[-\dot{x}_b\sin\theta_P\sin\delta + \dot{z}_b\sin\theta_P\cos\delta]\end{aligned} \tag{8}$$

where:

$$\begin{aligned}\dot{x}_b &= v_b\cos\delta \\ \dot{z}_b &= v_b\sin\delta\end{aligned} \tag{9}$$

thus:

$$\begin{aligned}v_P^2 &= v_b^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P[v_b\cos\theta_P\cos^2\delta + v_b\cos\theta_P\sin^2\delta] = \\ &= v_b^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P v_b\cos\theta_P\end{aligned} \tag{10}$$

3.3 DYNAMIC MODEL

The resulting torque, after the gearbox of each motor, suffers of the viscous friction and it is modeled as:

$$\begin{array}{rcl} C_L & = & \frac{1}{\rho}(\tau_L - \psi\dot{\alpha}_m) = \frac{1}{\rho}\tau_L - \frac{\psi}{\rho^2}(\dot{\alpha} - \dot{\theta}_P) \\ C_R & = & \frac{1}{\rho}(\tau_R - \psi\dot{\beta}_m) = \frac{1}{\rho}\tau_R - \frac{\psi}{\rho^2}(\dot{\beta} - \dot{\theta}_P) \end{array} \quad (11)$$

kinetic energy of the wheels:

the following equations accounts for both translational and rotational components of the wheel motion. The kinetic energy associated with rotation of the wheel around its vertical axis is neglected.

$$\begin{array}{rcl} T_L & = & \frac{1}{2}m_w v_L^2 + \frac{1}{2}J_w\dot{\alpha}^2 = \frac{1}{2}(m_w r^2 + J_w)\dot{\alpha}^2 \\ T_R & = & \frac{1}{2}m_w v_R^2 + \frac{1}{2}J_w\dot{\beta}^2 = \frac{1}{2}(m_w r^2 + J_w)\dot{\beta}^2 \\ T_w & = & T_L + T_R = \frac{1}{2}(m_w r^2 + J_w)(\dot{\alpha} + \dot{\beta})^2 \end{array} \quad (12)$$

kinetic energy of the rider:

the kinetic energy of the rider is built-up with three components:
The translational kinetic energy is the following:

$$\begin{array}{rcl} T_{tP} & = & \frac{1}{2}m_P v_P^2 = \frac{1}{2}m_P\left(v_b^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P v_b \cos\theta_P\right) \\ & = & \frac{1}{2}m_P\left[\left(r\frac{\dot{\alpha}+\dot{\beta}}{2}\right)^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P r\frac{\dot{\alpha}+\dot{\beta}}{2}\cos\theta_P\right] \end{array} \quad (13)$$

The rotational kinetic energy due to the rotation of the rider around the wheel's centre (θ_P) is made-up by the following:

$$T_{\theta P} = \frac{1}{2}J_{\theta P}\dot{\theta}_P^2 \quad (14)$$

The rotational kinetic energy due to the rotation around the vertical axis (y) is made-up by the following:

$$T_{yP} = \frac{1}{2}\left(J_{\delta P} + m_P L^2 \sin\theta_P^2\right)\dot{\delta}^2 = \frac{1}{2}\left(J_{\delta P} + m_P L^2 \sin\theta_P^2\right)\left(r\frac{\dot{\alpha}-\dot{\beta}}{D}\right)^2 \quad (15)$$

The total kinetic energy of the rider is given by:

$$\begin{array}{rcl} T_P & = & T_{tP} + T_{\theta P} + T_{yP} = \\ & & \frac{1}{2}m_P\left[\left(r\frac{\dot{\alpha}+\dot{\beta}}{2}\right)^2 + L^2\dot{\theta}_P^2 + 2L\dot{\theta}_P r\frac{\dot{\alpha}+\dot{\beta}}{2}\cos\theta_P\right] + \\ & & \frac{1}{2}J_{\theta P}\dot{\theta}_P^2 + \\ & & \frac{1}{2}\left(J_{\delta P} + m_P L^2 \sin^2\theta_P\right)\left(r\frac{\dot{\alpha}-\dot{\beta}}{D}\right)^2 \end{array} \quad (16)$$

kinetic energy of the motors:

The kinetic energy of the motors is:

$$\begin{array}{rcl} T_m & = & T_{mL} + T_{mR} = \frac{1}{2}J_m(\dot{\alpha}_m^2 + \dot{\beta}_m^2) = \\ & & = \frac{1}{2}\frac{J_m}{\rho^2}(\dot{\alpha}^2 + \dot{\beta}^2 + 2\dot{\theta}_P^2 - 2\dot{\alpha}\dot{\theta}_P - 2\dot{\beta}\dot{\theta}_P) \end{array} \quad (17)$$

Total kinetic energy:
The overall kinetic energy of the system is:

$$\begin{aligned} T = {} & T_P + T_w + T_m = \\ & \frac{1}{2}m_P\left[\left(r\frac{\dot\alpha+\dot\beta}{2}\right)^2 + L^2\dot\theta_P^2 + 2L\dot\theta_P r\frac{\dot\alpha+\dot\beta}{2}\cos\theta_P\right] + \\ & \frac{1}{2}J_{\theta P}\dot\theta_P^2 + \\ & \frac{1}{2}\left(J_{\delta P} + m_P L^2 \sin^2\theta_P\right)\left(r\frac{\dot\alpha-\dot\beta}{D}\right)^2 + \\ & \frac{1}{2}(m_w r^2 + J_w)(\dot\alpha+\dot\beta)^2 + \\ & \frac{1}{2}\frac{J_m}{\rho^2}(\dot\alpha^2 + \dot\beta^2 + 2\dot\theta_P^2 - 2\dot\alpha\dot\theta_P - 2\dot\beta\dot\theta_P) \end{aligned} \tag{18}$$

potential energy (of the rider):

This contribute of potential energy is given by simply the vertical position of the rider's center of mass:

$$U = m_P g L \cos\theta_P \tag{19}$$

3.4 LAGRANGIAN

Given the previous, it is possible to write the expression of the Lagrangian for the system

$$L = T - U \tag{20}$$

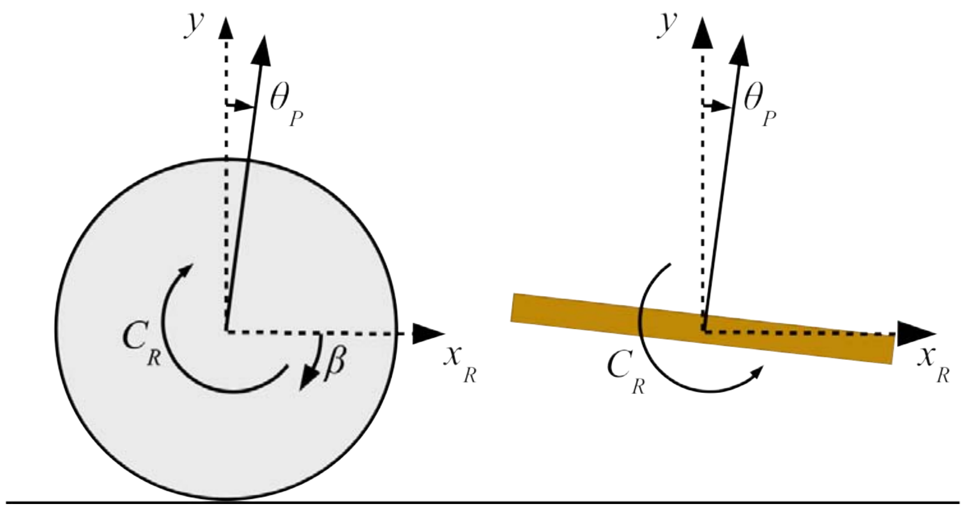


Figure 13: Detailed description of torques and references on right wheel. In the left, the wheel, while in the right the TWSBT base: note that the torque generated by the motor applies equal and opposite in the two bodies. The symmetric holds for the left wheel.

Having this, the Lagrange equations becomes:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot\alpha}\right) - \frac{\partial L}{\partial \alpha} &= C_L \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot\beta}\right) - \frac{\partial L}{\partial \beta} &= C_R \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot\theta_P}\right) - \frac{\partial L}{\partial \theta_P} &= -(C_L + C_R) \end{aligned} \tag{21}$$

The right-terms of the Lagrange equations represents the active contributes to the dynamics. The first and second equations' terms are quite obvious, while for the third equation, the active contribute is the sum of the torques of the two motors, as easily noticeable observing figure 13.

Which permits to write the complete form of the motion equations (112) this way:

$$\begin{aligned}
&\left[\frac{m_P r^2}{4}+\frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2}+m_w r^2+J_w+\frac{J_m}{\rho^2}\right]\ddot{\alpha}+\\
&\left[\frac{m_P r^2}{4}-\frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2}+m_w r^2+J_w\right]\ddot{\beta}+\\
&\left[\frac{m_P L r\cos\theta_P}{2}-\frac{J_m}{\rho^2}\right]\ddot{\theta}_P+\frac{\psi}{\rho^2}\dot{\alpha}-\frac{\psi}{\rho^2}\dot{\theta}_P+\\
&-\frac{m_P L\dot{\theta}_P^2 r\sin\theta_P}{2}+\frac{2m_P L^2 r^2\sin\theta_P\cos\theta_P}{D^2}\dot{\theta}_P\left(\dot{\alpha}-\dot{\beta}\right) = \frac{\tau_L}{\rho}\\
\\
&\left[\frac{m_P r^2}{4}-\frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2}+m_w r^2+J_w\right]\ddot{\alpha}+\\
&\left[\frac{m_P r^2}{4}+\frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2}+m_w r^2+J_w+\frac{J_m}{\rho^2}\right]\ddot{\beta}+\\
&\left[\frac{m_P L r\cos\theta_P}{2}-\frac{J_m}{\rho^2}\right]\ddot{\theta}_P+\frac{\psi}{\rho^2}\dot{\beta}-\frac{\psi}{\rho^2}\dot{\theta}_P+\\
&-\frac{m_P L\dot{\theta}_P^2 r\sin\theta_P}{2}+\frac{2m_P L^2 r^2\sin\theta_P\cos\theta_P}{D^2}\dot{\theta}_P\left(\dot{\beta}-\dot{\alpha}\right) = \frac{\tau_R}{\rho}\\
\\
&\left[\frac{m_P L r\cos\theta_P}{2}-\frac{J_m}{\rho^2}\right]\ddot{\alpha}+\left[\frac{m_P L r\cos\theta_P}{2}-\frac{J_m}{\rho^2}\right]\ddot{\beta}+\\
&+\left[m_P L^2+\frac{2J_m}{\rho^2}+J_{\theta_P}\right]\ddot{\theta}_P-m_P g L\sin\theta_P+\\
&+\frac{2\psi}{\rho^2}\dot{\theta}_P-\frac{\psi}{\rho^2}\dot{\alpha}-\frac{\psi}{\rho^2}\dot{\beta}-\frac{m_P L^2 r^2\sin\theta_P\cos\theta_P}{D^2}(\dot{\alpha}-\dot{\beta})^2 = -\frac{\tau_L}{\rho}-\frac{\tau_R}{\rho}
\end{aligned} \tag{22}$$

which are the motion equations for the system. Having this non-linear system, it is now possible to linearize it, in order to build a linear controller.

3.5 SYSTEM LINEARIZATION

The linearization of the system is performed around the vertical equilibrium condition ($\theta_P \approx 0$) and for small horizontal rotations ($\delta \approx 0$), which has as consequence:

$$\begin{aligned}
\sin\theta_P &\approx \theta_P\\
\cos\theta_P &\approx 1\\
\sin^2\theta_P &\approx 0\\
\dot{\theta_P}(\dot{\alpha}-\dot{\beta}) &\approx 0\\
(\dot{\alpha}-\dot{\beta})^2 &\approx 0\\
\dot{\theta}_P^2 &\approx 0
\end{aligned} \tag{23}$$

The system (43), after linearization, becomes the following:

$$\begin{aligned}
&\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{12} & m_{11} & m_{13} \\ m_{13} & m_{13} & m_{33} \end{bmatrix} \begin{Bmatrix} \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\theta}_P \end{Bmatrix} + \begin{bmatrix} c_{11} & 0 & -c_{11} \\ 0 & c_{11} & -c_{11} \\ -c_{11} & -c_{11} & 2c_{11} \end{bmatrix} \begin{Bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\theta}_P \end{Bmatrix} + \\
&+ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & k_{33} \end{bmatrix} \begin{Bmatrix} \alpha \\ \beta \\ \theta_P \end{Bmatrix} = \begin{Bmatrix} \frac{\tau_L}{\rho} \\ \frac{\tau_R}{\rho} \\ -(\frac{\tau_L}{\rho} + \frac{\tau_R}{\rho}) \end{Bmatrix}
\end{aligned}$$

where:

$$\begin{aligned}
m_{11} &= \left[\frac{m_P r^2}{4} + \frac{r^2 J_{\delta P}}{D^2} + m_w r^2 + J_w + \frac{J_m}{\rho^2}\right] \\
m_{12} &= \left[\frac{m_P r^2}{4} - \frac{r^2 J_{\delta P}}{D^2} + m_w r^2 + J_w\right] \\
m_{13} &= \left[\frac{m_P L r}{2} - \frac{J_m}{\rho^2}\right] \\
m_{33} &= \left[m_P L^2 + \frac{2J_m}{\rho^2} + J_{\theta_P}\right] \\
c_{11} &= \frac{\psi}{\rho^2} \\
k_{33} &= -m_P g L
\end{aligned} \tag{24}$$

Which makes it possible to build-up the state-space system this way:

$$\begin{aligned}
& M\ddot{q} + C\dot{q} + Kq = u \\
q &= \begin{Bmatrix} \alpha \\ \beta \\ \theta_P \end{Bmatrix}, \; x = \begin{Bmatrix} q \\ \dot{q} \end{Bmatrix} = \begin{Bmatrix} \alpha \\ \beta \\ \theta_P \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\theta_P} \end{Bmatrix}, \; u = \begin{bmatrix} \frac{1}{\rho} & 0 \\ 0 & \frac{1}{\rho} \\ -\frac{1}{\rho} & -\frac{1}{\rho} \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix} \\
M &= \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{12} & m_{11} & m_{13} \\ m_{13} & m_{13} & m_{33} \end{bmatrix} \\
C &= \begin{bmatrix} c_{11} & 0 & -c_{11} \\ 0 & c_{11} & -c_{11} \\ -c_{11} & -c_{11} & 2c_{11} \end{bmatrix} \\
K &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & k_{33} \end{bmatrix}
\end{aligned} \tag{25}$$

And finally the state-space representation:

$$\begin{aligned}
\dot{x} &= Ax + Bu \\
x &= \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \\
\dot{x} &= Ax + Bu = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M^{-1}(-Kq - C\dot{q} + u) \end{bmatrix} = \\
&= \begin{bmatrix} 0_3 & I_3 \\ -M^{-1}K & -M^{-1}C \end{bmatrix} x + \begin{bmatrix} 0_3 \\ M^{-1} \end{bmatrix} u = \\
&= \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M^{-1}(-Kq - C\dot{q} + u) \end{bmatrix} = \\
&= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & a_{43} & a_{44} & 0 & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & 0 \\ 0 & 0 & a_{63} & a_{64} & 0 & a_{66} \end{bmatrix} \begin{Bmatrix} \alpha \\ \beta \\ \theta_P \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\theta_P} \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ b_{41} & b_{42} \\ b_{51} & b_{52} \\ b_{61} & b_{62} \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix}
\end{aligned} \tag{26}$$

The non-null terms in the [A] and [B] matrices are not reported. In the next section it will be clear why.

3.6 SYSTEM DECOUPLING

The system just introduced has the problem of a deep coupling between the *BALANCE* and the *STEERING* components. The scope of this section is to introduce a decoupling strategy for it, in a way it will be possible to design two separate controllers (one for the balance and one for the steering), presented in the successive chapter.

First of all, the state coordinates will be changed, using the expressions (7), so that it will be easier to talk about base velocity and steering angle, instead of the α and β quantities. The base-change matrix is:

$$\begin{aligned}
q_s = \begin{Bmatrix} X_b \\ \delta \\ \theta_P \end{Bmatrix} &= \begin{bmatrix} r/2 & r/2 & 0 \\ r/D & -r/D & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \alpha \\ \beta \\ \theta_P \end{Bmatrix} = \begin{bmatrix} S \end{bmatrix} q \\
&\text{so that:} \\
\begin{Bmatrix} \alpha \\ \beta \\ \theta_P \end{Bmatrix} &= \begin{bmatrix} S \end{bmatrix}^{-1} \begin{Bmatrix} X_b \\ \delta \\ \theta_P \end{Bmatrix} = \begin{bmatrix} S \end{bmatrix}^{-1} q_s \\
&\text{thus:} \\
&\begin{bmatrix} M \end{bmatrix} \ddot{q} + \begin{bmatrix} C \end{bmatrix} \dot{q} + \begin{bmatrix} K \end{bmatrix} q = u \\
&\text{becomes:} \\
&\begin{bmatrix} M \end{bmatrix} \begin{bmatrix} S \end{bmatrix}^{-1} \ddot{q}_s + \begin{bmatrix} C \end{bmatrix} \begin{bmatrix} S \end{bmatrix}^{-1} \dot{q}_s + \begin{bmatrix} K \end{bmatrix} \begin{bmatrix} S \end{bmatrix}^{-1} q_s = u \\
&= \begin{bmatrix} M_s \end{bmatrix} \ddot{q}_s + \begin{bmatrix} C_s \end{bmatrix} \dot{q}_s + \begin{bmatrix} K_s \end{bmatrix} q_s
\end{aligned} \tag{27}$$

Following the procedure, like in the formulas (28), the new system matrices will be:

$$\begin{aligned}
\dot{x}_s &= \begin{bmatrix} q_s \\ \dot{q}_s \end{bmatrix} = A_s x_s + B_s u = \\
&= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & as_{43} & as_{44} & 0 & as_{46} \\ 0 & 0 & 0 & 0 & as_{55} & 0 \\ 0 & 0 & as_{63} & as_{64} & 0 & as_{66} \end{bmatrix} \begin{Bmatrix} X_b \\ \delta \\ \theta_P \\ \nu_b \\ \dot{\delta} \\ \dot{\theta}_P \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ bs_{41} & bs_{42} \\ bs_{51} & bs_{52} \\ bs_{61} & bs_{62} \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix}
\end{aligned} \tag{28}$$

At this point the important decoupling scheme. Since the "common-mode" torque oft the two motors contributes for the base movement and the "differential-mode" contributes for the base rotation, two new entities are introduced: the balancing and the rotation torques:

$$\begin{aligned}
\begin{Bmatrix} \tau_\theta \\ \tau_\delta \end{Bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix} = \begin{bmatrix} D \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix} \\
&\text{so that:} \\
\begin{Bmatrix} \tau_L \\ \tau_R \end{Bmatrix} &= \begin{bmatrix} D \end{bmatrix}^{-1} \begin{Bmatrix} \tau_\theta \\ \tau_\delta \end{Bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{Bmatrix} \tau_\theta \\ \tau_\delta \end{Bmatrix}
\end{aligned} \tag{29}$$

Appliying the previous to the state-space causes the decoupling of the problem, but in order to manage better the problem, a more-convenient view of the state-space system comes with a re-arrangement of the state variables order. The final result is the following:

$$\begin{aligned}
\dot{x}_N &= A_N x_N + B_N u_N = \\
\begin{Bmatrix} \dot{X}_b \\ \dot{\theta}_P \\ \ddot{X}_b \\ \ddot{\theta}_P \\ \dot{\delta} \\ \ddot{\delta} \end{Bmatrix} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & an_{32} & an_{33} & an_{34} & 0 & 0 \\ 0 & an_{42} & an_{43} & an_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & an_{66} \end{bmatrix} \begin{Bmatrix} X_b \\ \theta_P \\ \dot{X}_b \\ \dot{\theta}_P \\ \delta \\ \dot{\delta} \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ bn_{31} & 0 \\ bn_{41} & 0 \\ 0 & 0 \\ 0 & bn_{62} \end{bmatrix} \begin{Bmatrix} \tau_\theta \\ \tau_\delta \end{Bmatrix}
\end{aligned} \tag{30}$$

In this final system it is clearly visible that the problem can be divided in the following two separate systems:

$$
\begin{aligned}
& & & \text{Equilibrium subsystem:} \\
\dot{x}_\theta & = & & A_\theta x_\theta + B_\theta \tau_\theta = \\
\begin{Bmatrix} \dot{X}_b \\ \dot{\theta}_P \\ \ddot{X}_b \\ \ddot{\theta}_P \end{Bmatrix} & = & & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & an_{32} & an_{33} & an_{34} \\ 0 & an_{42} & an_{43} & an_{44} \end{bmatrix} \begin{Bmatrix} X_b \\ \theta_P \\ \dot{X}_b \\ \dot{\theta}_P \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \\ bn_{31} \\ bn_{41} \end{bmatrix} \tau_\theta \\
& & & \text{and Steering subsystem:} \\
\dot{x}_\delta & = & & A_\delta x_\delta + B_\delta \tau_\delta = \\
\begin{Bmatrix} \dot{\delta} \\ \ddot{\delta} \end{Bmatrix} & = & & \begin{bmatrix} 0 & 1 \\ 0 & an_{66} \end{bmatrix} \begin{Bmatrix} \delta \\ \dot{\delta} \end{Bmatrix} + \begin{bmatrix} 0 \\ bn_{62} \end{bmatrix} \tau_\delta
\end{aligned} \tag{31}
$$

3.7 STABILITY ANALISYS

The linearized, decoupled system has two subsystems that are interesting to analyze. In the following rows, some considerations about the linearized models (31), considering these parameters : a rider of 80kg and 1.80m height.

Concerning the *equilibrium* subsystem, it is a controllable system, even if underactuated. It owns the following poles and it is possible to note the presence of, as expected, an unstable (positive real) pole:

$$\text{poles of equilibrium} = 0 \ , \ 2.3267 \ , \ -2.3315 \ , \ -0.0167 \tag{32}$$

Represented in Gauss' plane in figure14.

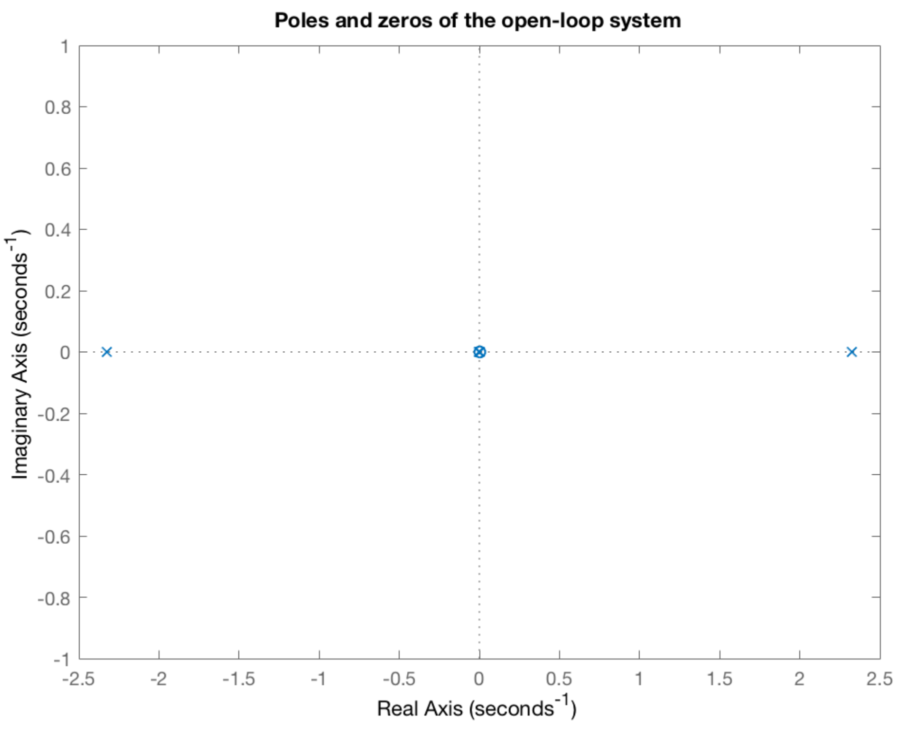


Figure 14: Poles and zero-es of equilibrium subsystem (open-loop unstable poles).

The unstable nature of the system is also confirmed by the nonlinear model, by plotting a free evolution obtained by applying a small perturbation to the tilt angle (5 degrees) and by holding $\tau_\theta = 0$. Those values, applied to the nonlinear model **??** gives the evolution visible in 15.

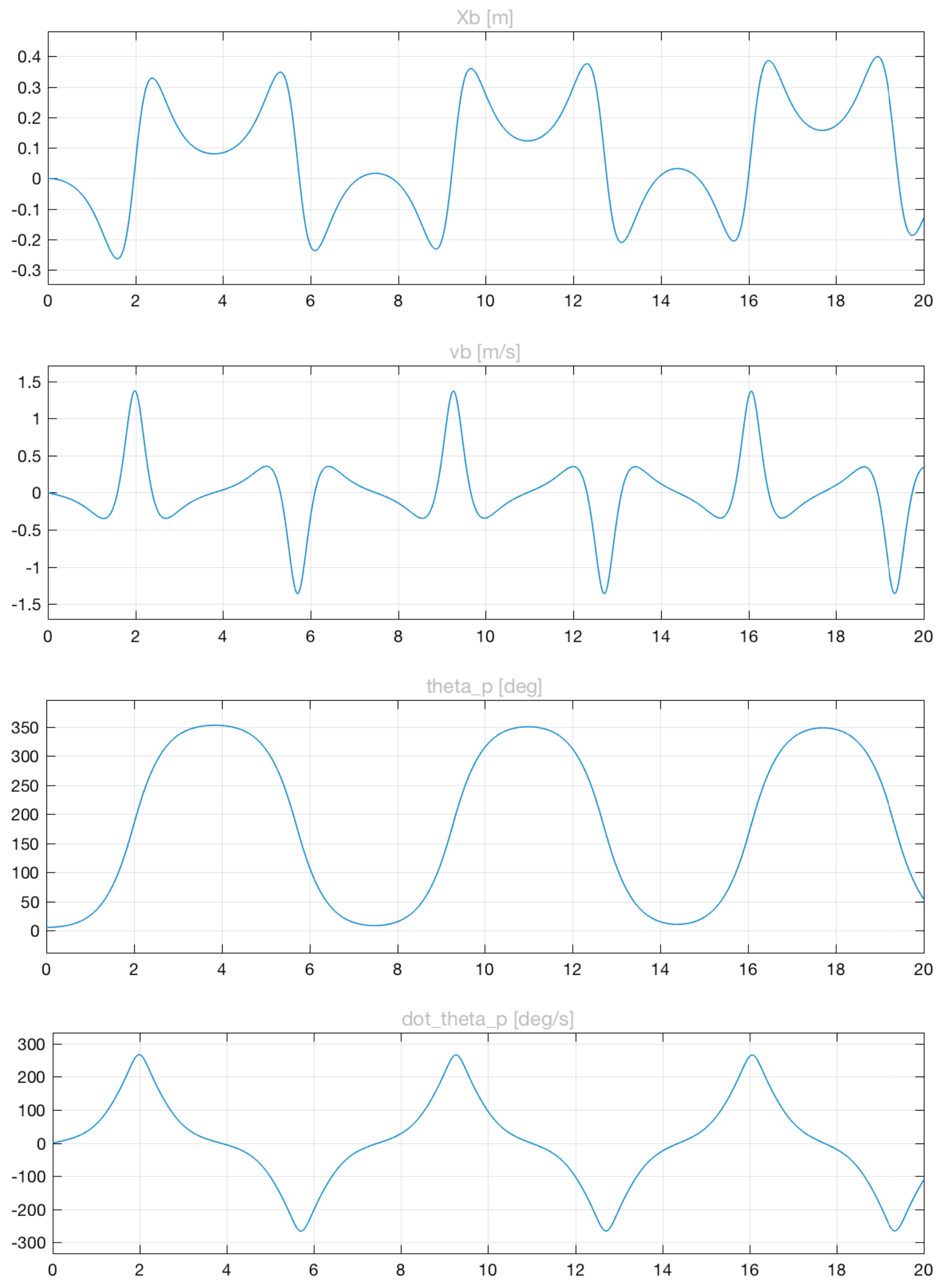


Figure 15: Free evolution of the model 43 with $\theta_P(0) = 5°$ and $\tau_\theta = 0$.

Concerning the *steering* subsystem, the poles are two: one (weakly) stable and one at the origin. The transfer function of the steering angle, against torque is:

$$W_\delta(s) = \frac{\delta}{\tau_\delta} \quad = \quad \frac{1}{s}\frac{7.2}{s+0.09861} \tag{33}$$

The root-locus and the Bode plot for the system are visible in figure 16.

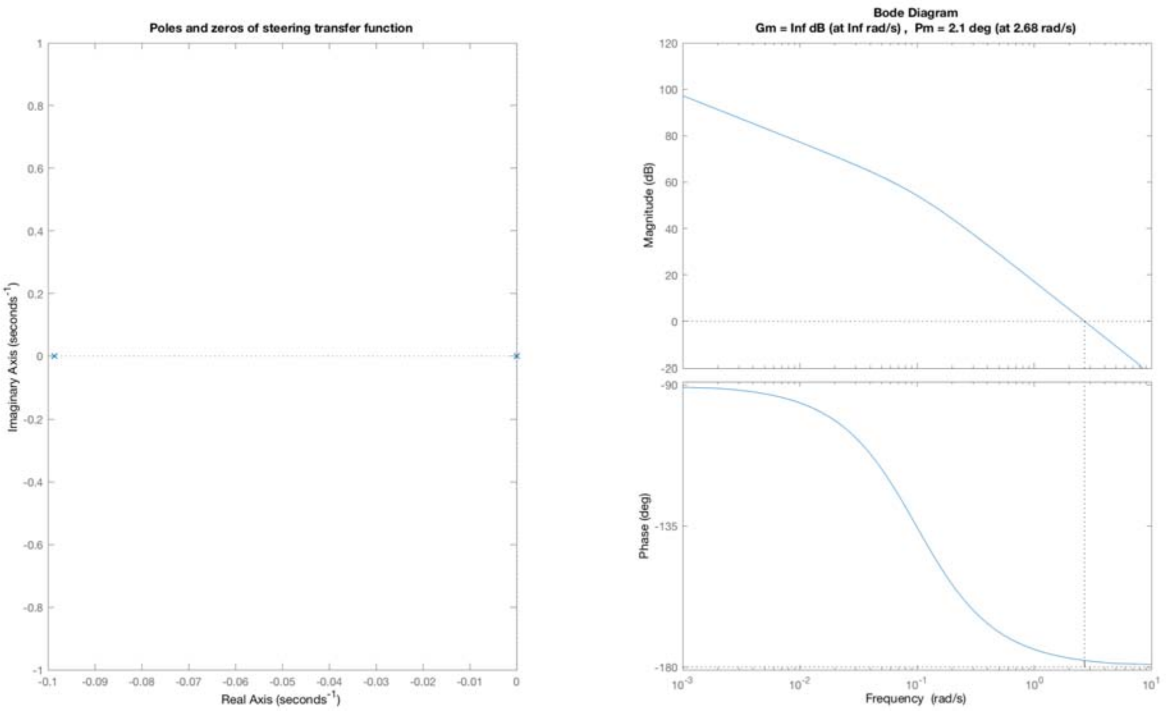


Figure 16: Root locus and Bode plot with stability margins of the $\frac{\delta}{\tau_\delta}$ transter function (33).

3.8 CONCLUSIONS

In this chapter it was introduced the simplest model for the study of the self-balancing inverse-pendulum system. This model was validated with simulations, using the simulink block-model. The properties of the system's subsystems (such as poles, zeros and stability) was also analyzed making this model usable in order to design a suitable controller.
In the following chapter a further step into the system modeling, by introducing a model extension, making it more realistic and interesting for study.

4 A MORE REALISTIC MODEL

4.1 INTRODUCTION

The system model presented in chapter 3 is the most commonly used for the *"self-balancing robot"* problem, which consist of a rigid body standing up over two wheels.
Even if useful in many studies, this model doesn't take in care one of the most important peculiarities of the TWSBT, visible in figure 17: a further freedom-degree, that is a different angle between the person and the base.

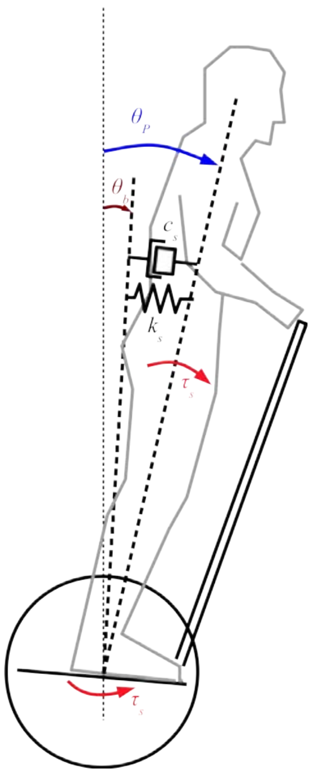


Figure 17: A more-complex model comes with the introduction of a further freedom-degree: the separation of the angle between the rider and the base.

In this new system, even if the body and the base might assume two different absolute angles, they remain coupled with the *"spring"* group, which consists of a spring-damper couple, representing the rider's ankle action. This coupling, in the absence of τ_s, provides the regime-equality of base and rider angles. The torque τ_s is the model of the action provided by the rider when it plans to impose a velocity and, as it happens in the reality, makes it possible to impose a difference between the angles, even at regime condition.
The values of K_s and c_s were not measured, because they depend on many factors, in fact the spring-damper model is a hard simplification of the complex muscular system. Some physiological studies as [20] and [19] have studied the problem of the stiffness and damping values for a human body trying to stand vertically and gave numeric values useful for the model buildup. In [9] an application of this

model to the self-balancing vehicle was used. This was taken as a starting point for this chapter.
Thanks to this, it is possible to rewrite in a convenient way the extended model of the system that will be called *MODEL2*, that differs from the first, mainly by the presence of the additional freedom degree.

The presence of this further freedom degree will change the lagrangian expression, introducing a further term of potential energy:

$$\begin{aligned} U_s &= \tfrac{1}{2}k_s(\theta_b-\theta_P)^2 \\ &\quad \text{so:} \\ U &= m_P g L \cos\theta_P + \tfrac{1}{2}k_s(\theta_b-\theta_P)^2 \end{aligned} \tag{34}$$

and a dissipation term, given by the damper, whose contribute is subtracted from the active force τ_s, resulting in:

$$C_s = \tau_s - c_s(\dot\theta_P - \dot\theta_b) = \tau_s + c_s(\dot\theta_b - \dot\theta_P) \tag{35}$$

The torques supplied by the motors now are referred to the base angle (not the rider's one):

$$\begin{aligned} C_L &= \tfrac{1}{\rho}(\tau_L - \psi\dot\alpha_m) = \tfrac{1}{\rho}\tau_L - \tfrac{\psi}{\rho^2}(\dot\alpha - \dot\theta_b) \\ C_R &= \tfrac{1}{\rho}(\tau_R - \psi\dot\beta_m) = \tfrac{1}{\rho}\tau_R - \tfrac{\psi}{\rho^2}(\dot\beta - \dot\theta_b) \end{aligned} \tag{36}$$

Also the kinetic energy of the motors varies in:

$$\begin{aligned} T_m &= T_{mL} + T_{mR} = \tfrac{1}{2}J_m(\dot\alpha_m^2 + \dot\beta_m^2) = \\ &= \tfrac{1}{2}\tfrac{J_m}{\rho^2}(\dot\alpha^2 + \dot\beta^2 + 2\dot\theta_b^2 - 2\dot\alpha\dot\theta_b - 2\dot\beta\dot\theta_b) \end{aligned} \tag{37}$$

And it is introduced the kinetic energy of the base (translational and rotational):

$$\begin{aligned} T_b &= \tfrac{1}{2}J_b\dot\theta_b^2 + \tfrac{1}{2}m_b v_b^2 = \\ &= \tfrac{1}{2}J_b\dot\theta_b^2 + \tfrac{1}{2}m_b\left(r\tfrac{\dot\alpha+\dot\beta}{2}\right)^2 \\ &= \tfrac{1}{2}J_b\dot\theta_b^2 + \tfrac{1}{4}m_b r\dot\alpha^2 + \tfrac{1}{4}m_b r\dot\beta^2 + \tfrac{1}{2}m_b r\dot\alpha\dot\beta \end{aligned} \tag{38}$$

The new coordinates vector and the Lagrangian becomes:

$$\begin{aligned} q &= \begin{Bmatrix} \alpha \\ \beta \\ \theta_b \\ \theta_P \end{Bmatrix} \\ L = T - U &= \tfrac{1}{2}m_P\left[r^2\tfrac{\dot\alpha^2}{4} + r^2\tfrac{\dot\beta^2}{4} + r^2\tfrac{\dot\alpha\dot\beta}{2} + L^2\dot\theta_P^2 + L\dot\theta_P r(\dot\alpha+\dot\beta)\cos\theta_P\right] + \\ &\quad \tfrac{1}{2}J_{\theta P}\dot\theta_P^2 + \tfrac{1}{2}\left(J_{\delta P} + m_P L^2\sin^2\theta_P\right)\left[r^2\tfrac{\dot\alpha^2}{D^2} + r^2\tfrac{\dot\beta^2}{D^2} - 2r^2\tfrac{\dot\alpha\dot\beta}{D^2}\right] + \\ &\quad \tfrac{1}{2}(m_w r^2 + J_w)(\dot\alpha^2 + \dot\beta^2 + 2\dot\alpha\dot\beta) + \\ &\quad \tfrac{1}{2}\tfrac{J_m}{\rho^2}(\dot\alpha^2 + \dot\beta^2 + 2\dot\theta_b^2 - 2\dot\alpha\dot\theta_b - 2\dot\beta\dot\theta_b) + \\ &\quad \tfrac{1}{2}J_b\dot\theta_b^2 + \tfrac{1}{4}m_b r\dot\alpha^2 + \tfrac{1}{4}m_b r\dot\beta^2 + \tfrac{1}{2}m_b r\dot\alpha\dot\beta \\ &\quad - m_P g L\cos\theta_P - \tfrac{1}{2}k_s(\theta_b - \theta_P)^2 \end{aligned} \tag{39}$$

thus:

$$\begin{aligned} \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial\dot\alpha}\right) - \tfrac{\partial L}{\partial\alpha} &= C_L \\ \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial\dot\beta}\right) - \tfrac{\partial L}{\partial\beta} &= C_R \\ \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial\dot\theta_b}\right) - \tfrac{\partial L}{\partial\theta_b} &= -(C_L + C_R) - C_s \\ \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial\dot\theta_P}\right) - \tfrac{\partial L}{\partial\theta_P} &= C_s \end{aligned} \tag{40}$$

The final result of these last expressions are the equations of motion in the new coordinates:

$$
\begin{aligned}
\left[\frac{m_P r^2}{4} + \frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2} + m_w r^2 + J_w + \frac{J_m}{\rho^2} + \frac{1}{2} m_b r\right]\ddot{\alpha} + \\
\left[\frac{m_P r^2}{4} - \frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2} + m_w r^2 + J_w + \frac{1}{2} m_b r\right]\ddot{\beta} + \\
-\frac{J_m}{\rho^2}\ddot{\theta}_b + \frac{m_P L r \cos\theta_P}{2}\ddot{\theta}_P + \\
+\frac{\psi}{\rho^2}\dot{\alpha} - \frac{\psi}{\rho^2}\dot{\theta}_b + \\
-\frac{m_P L \dot{\theta}_P^2 r \sin\theta_P}{2} + \frac{2 m_P L^2 r^2 \sin\theta_P \cos\theta_P}{D^2}\dot{\theta}_P\left(\dot{\alpha}-\dot{\beta}\right) &= \frac{\tau_L}{\rho} \\
\\
\left[\frac{m_P r^2}{4} - \frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2} + m_w r^2 + J_w + \frac{1}{2} m_b r\right]\ddot{\alpha} + \\
\left[\frac{m_P r^2}{4} + \frac{r^2\left(J_{\delta P}+m_P L^2 \sin^2\theta_P\right)}{D^2} + m_w r^2 + J_w + \frac{J_m}{\rho^2} + \frac{1}{2} m_b r\right]\ddot{\beta} + \\
-\frac{J_m}{\rho^2}\ddot{\theta}_b + \frac{m_P L r \cos\theta_P}{2}\ddot{\theta}_P + \\
+\frac{\psi}{\rho^2}\dot{\beta} - \frac{\psi}{\rho^2}\dot{\theta}_b + \\
-\frac{m_P L \dot{\theta}_P^2 r \sin\theta_P}{2} + \frac{2 m_P L^2 r^2 \sin\theta_P \cos\theta_P}{D^2}\dot{\theta}_P\left(\dot{\beta}-\dot{\alpha}\right) &= \frac{\tau_R}{\rho} \\
\\
\left[-\frac{J_m}{\rho^2}\right]\ddot{\alpha} + \left[-\frac{J_m}{\rho^2}\right]\ddot{\beta} + \\
+\left[\frac{2J_m}{\rho^2} + J_b\right]\ddot{\theta}_b + \left[\frac{2\psi}{\rho^2} + c_s\right]\dot{\theta}_b + \\
-c_s\dot{\theta}_P - \frac{\psi}{\rho^2}\dot{\alpha} - \frac{\psi}{\rho^2}\dot{\beta} + k_s\theta_b - k_s\theta_P &= -\frac{\tau_L}{\rho} - \frac{\tau_R}{\rho} - \tau_s \\
\\
\left[\frac{m_P L r \cos\theta_P}{2}\right]\ddot{\alpha} + \left[\frac{m_P L r \cos\theta_P}{2}\right]\ddot{\beta} + \\
+\left[m_P L^2 + J_{\theta_P}\right]\ddot{\theta}_P - m_P g L \sin\theta_P - k_s\theta_b + k_s\theta_P + \\
+c_s\dot{\theta}_P - c_s\dot{\theta}_b - \frac{m_P L^2 r^2 \sin\theta_P \cos\theta_P}{D^2}(\dot{\alpha}-\dot{\beta})^2 &= \tau_s
\end{aligned}
\tag{41}
$$

4.2 LINEARIZATION OF THE NEW SYSTEM

As before, the system linearization is performed around the vertical equilibrium condition ($\theta_P \approx 0 \approx \theta_b$) and for small horizontal rotations ($\delta \approx 0$), which has as consequence:

$$
\begin{aligned}
\sin\theta_P &\approx \theta_P \\
\sin\theta_b &\approx \theta_b \\
\cos\theta_P &\approx 1 \\
\sin^2\theta_P &\approx 0 \\
\dot{\theta}_P(\dot{\alpha}-\dot{\beta}) &\approx 0 \\
(\dot{\alpha}-\dot{\beta})^2 &\approx 0 \\
\dot{\theta}_P^2 &\approx 0
\end{aligned}
\tag{42}
$$

The system (43), after linearization, becomes the following:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{12} & m_{11} & m_{13} & m_{14} \\ m_{13} & m_{13} & m_{33} & 0 \\ m_{14} & m_{14} & 0 & m_{44} \end{bmatrix} \begin{Bmatrix} \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\theta}_b \\ \ddot{\theta}_P \end{Bmatrix} + \begin{bmatrix} c_{11} & 0 & -c_{11} & 0 \\ 0 & c_{11} & -c_{11} & 0 \\ -c_{11} & -c_{11} & c_{33} & c_{34} \\ 0 & 0 & c_{34} & -c_{34} \end{bmatrix} \begin{Bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\theta}_b \\ \dot{\theta}_P \end{Bmatrix} +$$
$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & k_{33} & -k_{33} \\ 0 & 0 & -k_{33} & k_{44} \end{bmatrix} \begin{Bmatrix} \alpha \\ \beta \\ \theta_b \\ \theta_P \end{Bmatrix} = \begin{Bmatrix} \frac{\tau_L}{\rho} \\ \frac{\tau_R}{\rho} \\ -(\frac{\tau_L}{\rho} + \frac{\tau_R}{\rho}) + \tau_s \\ -\tau_s \end{Bmatrix} = \begin{bmatrix} \frac{1}{\rho} & 0 & 0 \\ 0 & \frac{1}{\rho} & 0 \\ -\frac{1}{\rho} & -\frac{1}{\rho} & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \\ \tau_s \end{Bmatrix}$$

where:

$$\begin{aligned} m_{11} &= \left[\frac{m_P r^2}{4} + \frac{r^2 J_{\delta P}}{D^2} + m_w r^2 + J_w + \frac{1}{2} m_b r + \frac{J_m}{\rho^2}\right] \\ m_{12} &= \left[\frac{m_P r^2}{4} - \frac{r^2 J_{\delta P}}{D^2} + m_w r^2 + J_w + \frac{1}{2} m_b r\right] \\ m_{13} &= \frac{-J_m}{\rho^2} \\ m_{14} &= \frac{m_P L r}{2} \\ m_{33} &= \frac{2 J_m}{\rho^2} + J_b \\ m_{44} &= \left[m_P L^2 + J_{\theta_P}\right] \\ c_{11} &= \frac{\psi}{\rho^2} \\ c_{33} &= \left[\frac{2\psi}{\rho^2} + c_s\right] \\ c_{34} &= -c_s \\ k_{33} &= k_s \\ k_{44} &= -m_P g L + k_s \end{aligned} \tag{43}$$

4.2.1 *decoupling*

The presence of θ_s doesn't change the decoupling scheme adopted, whose modification is very limited:

$$q_s = \begin{Bmatrix} X_b \\ \delta \\ \theta_b \\ \theta_P \end{Bmatrix} = \begin{bmatrix} r/2 & r/2 & 0 & 0 \\ r/D & -r/D & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \alpha \\ \beta \\ \theta_b \\ \theta_P \end{Bmatrix} = \left[S\right] q$$

so that:

$$\begin{Bmatrix} \alpha \\ \beta \\ \theta_b \\ \theta_P \end{Bmatrix} = \left[S\right]^{-1} \begin{Bmatrix} X_b \\ \delta \\ \theta_b \\ \theta_P \end{Bmatrix} = \left[S\right]^{-1} q_s \tag{44}$$

and the decoupling matrix is:

$$\begin{Bmatrix} \tau_\theta \\ \tau_s \\ \tau_\delta \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{Bmatrix} \tau_L \\ \tau_R \\ \tau_s \end{Bmatrix} = \left[D\right] \begin{Bmatrix} \tau_L \\ \tau_R \\ \tau_s \end{Bmatrix}$$

Following the same steps done for the derivation of the (31), the state-space expression of the system with this further freedom degree is:

$$\begin{aligned}
\dot{x}_\theta &= A_\theta x_\theta + B_\theta \tau_\theta = \\
\begin{Bmatrix} \dot{X}_b \\ \ddot{X}_b \\ \dot{\theta}_b \\ \ddot{\theta}_b \\ \dot{\theta}_P \\ \ddot{\theta}_P \end{Bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & an_{22} & an_{23} & an_{24} & an_{25} & an_{26} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & an_{42} & an_{43} & an_{44} & an_{45} & an_{46} \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & an_{62} & an_{63} & an_{64} & an_{65} & an_{66} \end{bmatrix} \begin{Bmatrix} X_b \\ \dot{X}_b \\ \theta_b \\ \dot{\theta}_b \\ \theta_P \\ \dot{\theta}_P \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ bn_{21} & bn_{22} \\ 0 & 0 \\ bn_{41} & bn_{42} \\ 0 & 0 \\ bn_{61} & bn_{62} \end{bmatrix} \begin{Bmatrix} \tau_\theta \\ \tau_s \end{Bmatrix} \\
\text{and:} & \\
\dot{x}_\delta &= A_\delta x_\delta + B_\delta \tau_\delta = \\
\begin{Bmatrix} \dot{\delta} \\ \ddot{\delta} \end{Bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & an_{88} \end{bmatrix} \begin{Bmatrix} \delta \\ \dot{\delta} \end{Bmatrix} + \begin{bmatrix} 0 \\ bn_{83} \end{bmatrix} \tau_\delta
\end{aligned} \tag{45}$$

4.3 TRANSFER FUNCTION AND STABILITY ANALISYS

In the following rows, some considerations about the linearized new models (45). The main system parameters adopted for this study are:

- a rider of 80kg
- 1.80m height
- $k_s = 1440$
- $c_s = 350$

For the values of k_s and c_s, the values were copied from [19].

Concerning the *equilibrium* subsystem, it is possible to tell that the rider's angle is not accessible: only the base angle is known (accelerometers can read this, but nothing can read the rider's angle). Having this, it is possible do define the output matrix as the subset of state variables readable from the system:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{46}$$

Thanks to the definition of C, it is possible to say that the equilibrium subsystem is an *UNOBSERVABLE* and, observing A and B matrices, even an *UNCONTROLLABLE* one. The poles of the system are:

$$\begin{aligned} \text{poles of equilibrium} &= 0\,,\ -10345\,,\ 2.2479\,, \\ &\quad -0.0092\,,\ -4.1154\,, \\ &\quad -2.2499 \end{aligned} \tag{47}$$

The presence of a high-frequency pole is given by the base: it has a very small weight, and even a very small inertia momentum (compared with that of the rider),

thus resulting in a contribute of a very-high frequency.
As for the first model, the main scope of the control will be to maintain upside the rider.

5 CONTROL

5.1 INTRODUCTION

In the previous chapters there were built two linearized models of the vehicle: one (simplified) where the rider was -linked with the vehicle base, and one where it had a further freedom degree. Even the first, simplified model have demonstrated to be a multi-input, multi-output (*MIMO)* system.

Further transforms permitted to separate the inverse-pendulum system into two distinct subsystems algebraically independent, thus that might be separately controlled. The *steering subsystem* is a single-input, single-output (*SISO*) system, that has a simple dynamic description, while the *equilibrium subsystem* is a single-input, multi-output (*SIMO*) system.

This chapter describes the control strategies adopted for the two subsystems, together with considerations about the stability of that controllers.

5.2 CONTROL OF EQUILIBRIUM SUBSYSTEM

The *equilibrium subsystem* is a single-input, multi-output (*SIMO*) system, thus it is necessary to adopt a suitable controller.

In the panorama of possible controllers, some possibilities are:

- Pole-placement technique
- Self-learning controllers
- Linear Quadratic (optimum) controllers

Since the poles and their meaning is such a system is not obvious, the Pole-placement was discarded. Also the choice of self-learning controllers was discarded, since the aim was to build a precise-behavior system, where it was always possible to know and discuss the control behavior. Also there was not enough time to properly train an algorithm.

The final choice was the adoption of an *optimum controller* (*LQR*) because it was very easy to relate the controller weights, with the system's coordinates. The drawback of this approach is that it is not that simple to relate the controller's gains, with the coordinate's dynamics.

Regarding this missing relation, some studies like [13] and [22] have proposed the use of a *genetic algorithm* in order to properly tune the LQR coefficients, for obtaining the desired system dynamics. In this section it will be described the LQR controller designed for the self-balancing vehicle and some discussion about the coefficient choice and the consequent stability.

5.2.1 *The optimum (LQR) controller*

The classic approach in controlling a system is to find a feedback controller in order to make it place the poles/zeroes of the system in a way to obtain *stability* and some other performances such as *bandwidth*. Many times it is of interest to have a controller that can guarantee adequate rejection to disturbances. The limit of this approach is that it is difficult to evaluate the effect of bandwidth, pole position, disturbance rejection, in other aspects such as single state deviation or energy consumption. This is the main scope of the optimum controller: designer has a tool to privilege the economicity or the deviation from one or a set of state variables, giving the possibility to have a very powerful and flexible control that satisfies different characteristics.

All the previous is possible with a "cost functional", which is a quadratic form that describes the "cost" (always non-negative) when privileging one aspect instead of another. For a discrete-time proper system defined as:

$$\begin{cases} x(k+1) = Fx(k) + G(k) \\ x(0) = x_0 \end{cases} \tag{48}$$

The cost functional is:

$$J(u, x_0) = \sum_{k=0}^{T-1} [x(k)^T Qx(k) + u(k)^T Ru(k)] dk + x(T)^T Sx(T)$$

Where it is possible to observe three terms in a quadratic form, associated with a matrix.

- The first term is that with Q, which is **positive semi-definite**. It describes the cost related to the deviation of the state during the sequence $0 - T$. Having Q diagonal, for every single element in the diagonal, corresponds the cost of the deviation of that singular state variable. *Note: the higher the values, the more precise the controller.*

- The second term is that with R, which is also **positive semi-definite**. It describes the cost related to the input sequence (the power consumption for obtaining a result). *Note: the higher the values, the more economic the controller.*

- The third term is that with S, which is also **positive definite**. It describes the cost related to the error in the final state. *Note: the higher the values, the more precise the controller.*

5.2.1.1 *Infinite-horizon control*

The previous functional is valid in the case that T is a finite number (control over a finite-time horizon).
In the case $T \to \infty$ there is the advantage that the input can be built with a linear and constant feedback from the state, but some problem arises:

1. In order to have J non-infinite, it is important to verify the existence of an input sequence with which J assumes a finite value.

2. Having a control in the form of a feedback, it must be internally stable.

3. The feedback matrix might be found by solving the Algebraic Riccati Equation (ARE), which admits several solutions. Only one of such solution is the optimum: the problem is to determine the solution that matches the requirements.

In the same discrete-system in 48, at infinite-time horizon, the cost functional becomes:

$$J(u, x_0) = \sum_{k=0}^{\infty} (x(k)^T Q x(k) + u(k)^T R u(k)) \tag{49}$$

Where **Q is symmetric and positive-semidefinite** and **R is symmetric and positive-definite**.
- *lemma 1* If the system 48 is stabilizable, there exist input-sequences for whom the index 49 is finite.
- *lemma 2* If the system 48 is stabilizable, the succession of matrices $M(0), M(-1)...$ defined as:

$$\begin{aligned} &M(0) = 0 \\ &M(k) = Q + F^T M(k+1) F + \\ &-F^T M(k+1) G [R + G^T M(k+1) G]^{-1} G^T M(k+1) F \end{aligned}$$

converges (for $k \to \infty$) to a symmetric, positive semi-definite matrix that satisfies the ARE:

$$M = Q + F^T M F + -F^T M G [R + G^T M G]^{-1} G^T M F \tag{50}$$

- *lemma 3* Assume M is a solution symmetric, positive semi-definite solution of ARE. For every input succession $u(0), u(1), ...u(T-1)$ and every initial state x_0, it results that:

$$\sum_{k=0}^{T-1} (x(k)^T Q x(k) + u(k)^T R u(k)) = \tag{51}$$

$$x_0^T M x_0 - x^T(T) M x(T) + \tag{52}$$

$$+ \sum_{k=0}^{T-1} [(R + G^T M G) u(k) + G^T M F x(k)]^T * \tag{53}$$

$$[R + G^T M G][(R + G^T M G) u(k) + G^T M F x(k)] \tag{54}$$

Observing also the theorem:
Theorem 1:
Having the system 48 and M, the solution found with *lemma 2*, then

1. the control law

$$\begin{aligned} u_\infty(k) &= K_\infty(k) x(k) \\ K_\infty &= -(R + G^T M_\infty G)^{-1} G^T M_\infty F \end{aligned}$$

   minimizes the index J

2. For every other solution symmetric, positive-semi-definite $\bar{M}$ of ARE, the matrix $\bar{M} - M_\infty$ is positive-semidefinite

The theorem gives a way to calculate an optimum controller with linear-feedback function that creates an input sequence, directly from the system state. The drawback is that the solution found might be not-stabilizing. It depends on some properties of the matrices F and G .

Another aspect is that if M_∞ is positive-semi-definite (not definite), the optimum input for some state configuration might be the NULL sequence, since there exists non-null initial states for which it exists a minimum value of the cost J. This corresponds to have no action on the system, by the controller.
Theorem 2:
If M_∞ is the solution of ARE that minimizes the 49 for the system 48. Having a C matrix such that:

$$Q = C^T C$$

Then M_∞ is positive-definite only if the couple (F, C) is observable.

5.2.1.2 *Optimum control and stabilization*

The stabilization of the system is the most important requirement for our control, since it must act in an unstable system and make it stable. Here are some important aspects of the internal stability requirements for the control:
Theorem 3:
Given the system 48, with (F, G) stabilizable, and M_∞ the optimum solution of ARE. This solution is also stabilyzing if the couple (F, C) is observable.
Theorem 4:
Given the system 48, with (F, G) stabilizable and $Q = C^T C$. It is true that:

- ARE has at most one symmetric positive-semi-definite solution associated with a stabilizing control.
- This solution exists only if the couple (F, C) is observable.
- If a stabilizing solution M_S exists, for every symmetric positive-semi-definite solution $\bar{M}$ the matrix $M_S - \bar{M}$ is symmetric positive-semi-definite.
- If a stabilizing solution M_S exists, it is possible to derive it starting with as a limit for $k \to \infty$ of the ARE solution assuming $M(0)$ an arbitrary symmetric positive-definite matrix.

Corollary:
If the solution M_S of ARE is stabilizing, it is the only symmetric positive-semi-definite solution of the equation.

At this point a scenario is possible: M_S and M_∞ both exists and different. It is obvious that the stabilyzing solution is the obligatory choice. In this case the control will be:

$$u_s(k) = k_s x(k) \tag{55}$$

This solution, in the space of the stabilyzing laws, is that associated with the minimum cost index.

5.2.2 *Application of LQR to the equilibrium subsystem*

The introduced state-feedback control scheme was used to stabilize the equilibrium subsystem, whose linearized form is the following state-space system (reported from equation (31)):

$$\begin{aligned} \dot{x}_\theta &= A_\theta x_\theta + B_\theta \tau_\theta = \\ \begin{Bmatrix} \dot{X}_b \\ \dot{\theta}_P \\ \ddot{X}_b \\ \ddot{\theta}_P \end{Bmatrix} &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & an_{32} & an_{33} & an_{34} \\ 0 & an_{42} & an_{43} & an_{44} \end{bmatrix} \begin{Bmatrix} X_b \\ \theta_P \\ \dot{X}_b \\ \dot{\theta}_P \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \\ bn_{31} \\ bn_{41} \end{bmatrix} \tau_\theta \end{aligned} \tag{56}$$

This continuous-time system must be discretized, in order to apply in it, the mentioned controller. Given a system of matrices A, B,C, D, the corresponding discretized matrices are:

$$\begin{aligned} A_d &= e^{AT_s} \\ B_d &= \left(\int_{\tau=0}^{T_s} e^{A\tau} d\tau \right) B \\ C_d &= C \\ D_d &= D \end{aligned} \tag{57}$$

This discretization of the system was automatically performed by the MATLAB program with the "c2d" command:

```
%%  Model discretization (for the equilibrium subsystem)
sys_cont=ss(A_e,B_e,C_e,D_e,'statename',states,'inputname',inputs,'outputname
    ',outputs);
sysPd = c2d(sys_cont, Ts);          %   discrete-time model (LTI obj)
[Ad,Bd,Cd,Dd] = ssdata(sysPd);  %   discrete-time model (state-space matrices)
```

which performs a continuous-to-discrete transform, based on the "zero-order-hold" technique, which basically holds the input for the entire T_s period, as shown in figure 18.

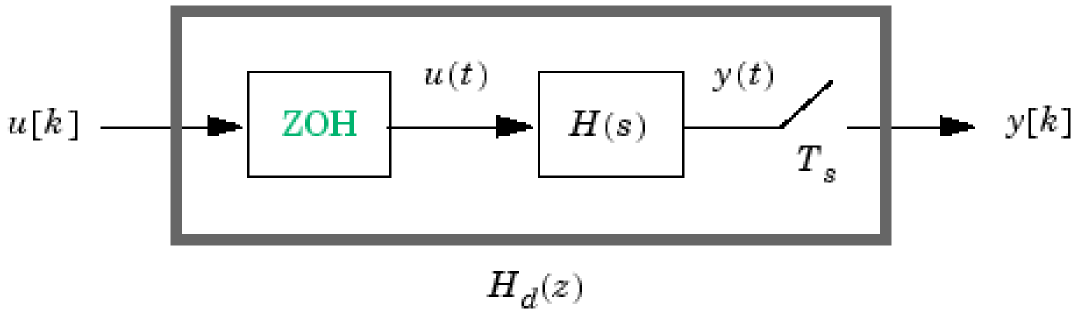


Figure 18: Zero-order-hold process, used (as default method) by MATLAB, for the continous-to-discrete conversion of a system. Figure from MATLAB's website.

Now, having the discretized system, an LQR controller can be easily built, with the MATLAB command "dlqr":

```
% DLQR parameters for the model: discrete-time variant of LQR
Kd = dlqr(Ad,Bd,Q,R)
```

which solves the Discrete Riccati Equation (*DRE*) and returns Kd, that is the vector of gains to be applied to the state-errors (the vector of differences between the reference value of the state and the true, measured value), as stated in equation (55), where k_s is replaced with the Kd, and the reference now is not 0, like in (55), but is x_{ref}.

This results in:

$$u(k) = Kd * [x_{ref}(k) - x(k)]$$

which means:

$$\tau_\theta = Kd * \left[\begin{Bmatrix} \dot{X}_b \\ \dot{\theta}_P \\ \ddot{X}_b \\ \ddot{\theta}_P \end{Bmatrix}_{ref} (k) - \begin{Bmatrix} \dot{X}_b \\ \dot{\theta}_P \\ \ddot{X}_b \\ \ddot{\theta}_P \end{Bmatrix} (k) \right] \tag{58}$$

This equation was modeled in the block-scheme with the components in figure 19.

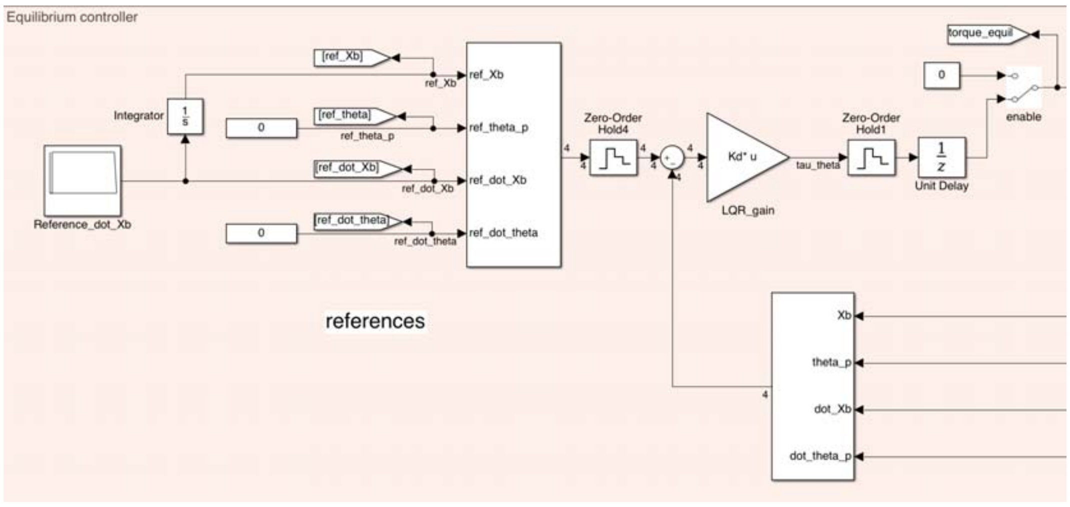


Figure 19: Control scheme for the equilibrium of the self-balancing transporter: a discrete-time LQR acts on the "equilibrium torque". The unit delay represents the processing limit of the micro-controntroller, which must collect data (via serial-link), calculate the necessary torque (current), and send back the current reference to the motors.

As noticeable in figure 19 the only meaningful reference supplied to the system is that of linear velocity: the linear position is obtained as a time-integration of that velocity reference.

The references for both the pitch angle and its velocity are set to a constant value that is the only one that makes sense: zero. Infact it has no sense to supply a different pitch reference, since it will determine a violation of equilibrium condition, with a consequent constant acceleration, to balance the perturbed equilibrium.

5.2.3 *Stability analysis*

As explained in the previous subsections, the optimum controller is not necessarily a stabilizing one. In this section it will be investigated the choice of the weights of the Q matrix, their effect on the system performances and into the control stability and the considerations made, in order to guarantee a stable controller.

Unfortunately (as stated in the introduction of this chapter) the weigths set in the Q matrix, and those in R has not a direct, simple correlation with the system dynamics and the stability, but they has the advantage of a simple interpretation, regarding "what we want" from the system.

Being the states:

$$x = \begin{Bmatrix} X_b \\ \theta_P \\ \dot{X}_b \\ \dot{\theta}_P \end{Bmatrix} \tag{59}$$

setting the "weights" of the Q matrix:

$$Q = \begin{bmatrix} w_{X_b} & 0 & 0 & 0 \\ 0 & w_{theta_P} & 0 & 0 \\ 0 & 0 & w_{\dot{X}_b} & 0 \\ 0 & 0 & 0 & w_{\dot{\theta}_P} \end{bmatrix} \tag{60}$$

means "*how much we care about an error in that state coordinate*". *Example:* a high value on w_{theta_P} means we carry much about the angle deviation from the vertical position, while a high weight in w_{X_b} means we want the vehicle to deviate the less possible, from the position set.

A high weight in the velocities means a little deviation from the velocities set-point. This implies that if the velocities are set to 0, we will have a "slow" control, while with a low weight, we let the control follow the other references without "carrying much" about how much the velocity deviates from null: it corresponds to a "fast control".

The R matrix (in our case it is a simple scalar-value) represents "*how much we care about the input usage*", which is similar to say "*how economic must be the control*". A high value on R means we want an "economic" control, which implies many times a "poor-performance" control. A low value means obviously the opposite: a high-performance control, with the "defect" of spending much energy.

A final consideration about the relation between the matrices Q and R: the ratio between them represents a "control invariant", in the sense that if (for instance) we double the value of R and half all the values of Q, the control dynamics stays the same.

Now it remains the *stability* issue: a wrong choice of the Q and R coefficients might determine a critically stable, or even an unstable system. This deeply depends on the form of the matrix to be controlled (A_d). In chapter 3 we already discussed about the eigenvalues of the A matrix, which corresponds to the (unstable) poles of the free-evolution system.

What we can do here is to evaluate the stability of the controlled system by analizing the eigenvalues of the controlled system's matrix:

$$A' = [A - k_s B] \tag{61}$$

We can tell that:

- setting $w_{X_b} = 0$ creates an unstable controller, in fact this means "don't care about position" and it is obviously an unstable condition (MATLAB automatically outputs an error about this, refusing to compute an unstable controller).
- a combination of the weights of Q and R might result in real-negative (or null) or complex-conjugated poles: they might be over-damped or under-damped, making the output prone to self oscillations.

The last statement is visible in figures 20 and 21. In the first figure, the choice of Q and R yelded to both real and complex-conjugated, under-damped poles, while in the second figure, the poles are all negative.

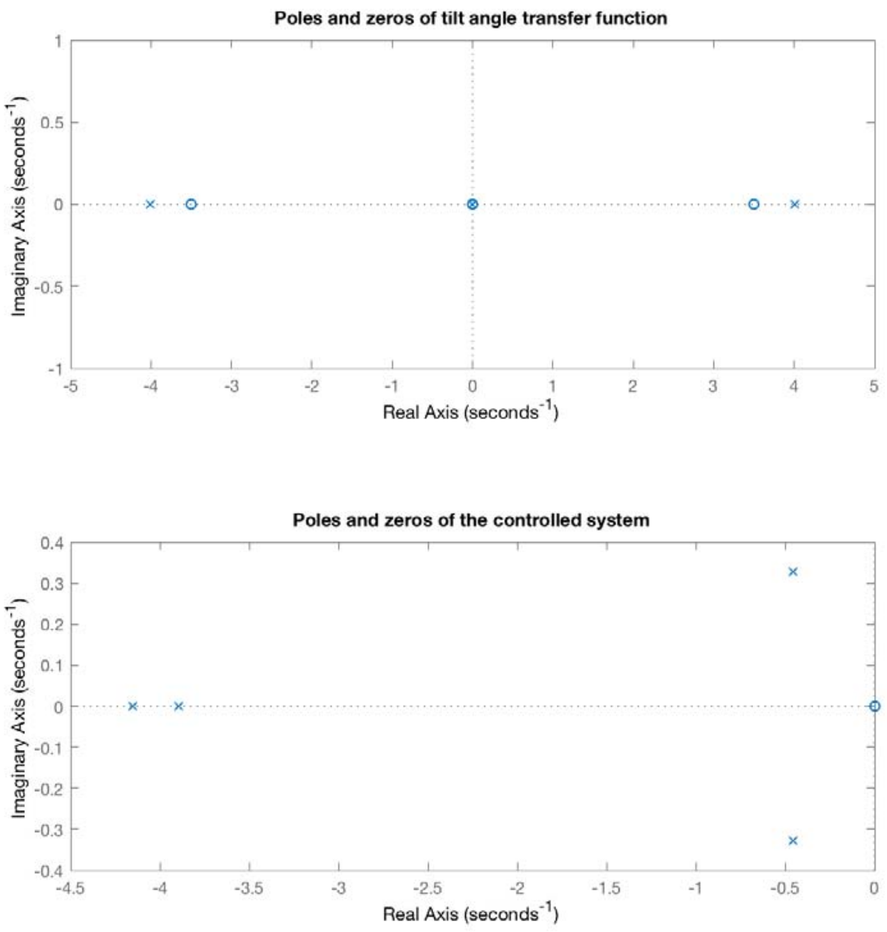


Figure 20: Poles of the system: in the upper figure, the uncontrolled system, in the lower the controlled one. The choice of the coefficients of Q and R yielded to both real and complex-conjugated, under-damped poles.

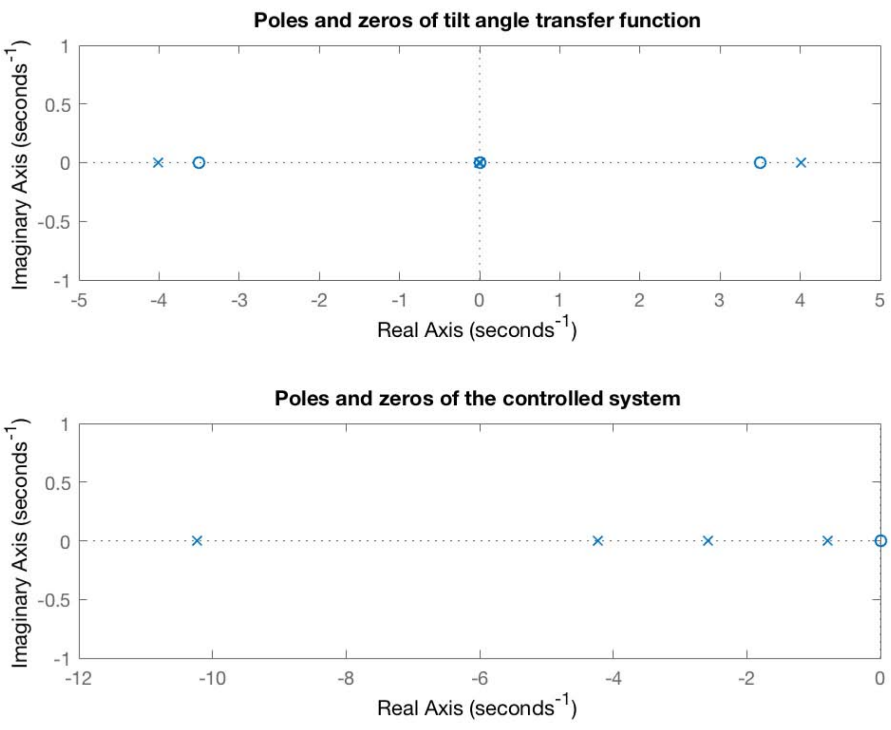


Figure 21: Poles of the system: differently from the situation of figure 20, the controlled-poles are now all negative-real.

At this point the simplest idea about the settings of the weights of matrices Q and R is to try-and error until we reach a controller where poles are all real-negative and "as negative as possible", for getting a high performance device.

Obviously it is not that simple: this is an *ideal model*, which means that at the moment we did not considerate the dynamics of other components of the system which can be either limited-band current loops, non ideal sensors etc. These kind of components will be discussed in the next chapter, but what we can tell here is that it is important to avoid highly-negative poles, which might determine an interaction with other component's dynamics and consequent stability compromises.

5.2.4 *LQR tuning*

Even having investigated about the choice of the "weights" of the Q matrix and having stated that a $w_{X_b} = 0$ determines an instable controller, the "true life" is that when using the self-balancing transporter, the rider doesn't care about maintaining the vehicle to the initial position: he only wants the vehicle to maintain the vertical position and move forward or back when he lean in the desired direction.
In this scenario a different controller must be designed: a controller where the weight relative to the horizontal position is $w_{X_b} \approx 0$. Also it is important to set $w_{\dot{X}_b} = 0$, infact since in this scenarion both the references of X_b and $\dot{X}_b$ will be 0, setting $w_{\dot{X}_b} \neq 0$ means the controller will try to "slow" the vehicle until it reaches 0 velocity.

The "real life" controller has another constraint: the weight relative to the vertical angle must be sufficiently high, in order to give the rider the sensation to be in a "rigid ground", instead of an excessively floating platform that might induce him to do corrective actions, in contrast of the inclination of the base.

Obviously an excessive "readiness" might result in high gains with "nervous" behavior and stability risks.

The LQR tuning is then the activity of finding a good compromise between "readiness" and stability, with regard to the human perception.

The scope of this thesis is not only to "build a comfortable vehicle", but to build an affordable model and to validate it. In order to do this, another scenario was created: a rigid body, solidly-linked with the base and some experiments regarding the behavior of the vehicle, compared with simulation results. In this case it is important to validate the behavior of the vehicle with $w_{X_b} \neq 0$ and $w_{\dot{X}_b} \neq 0$.

Having two different scenarios of control, the choice of the coefficients in the matrices was made by following the two different control strategies:

1. Control weights for "real-life" transporter $w_{X_b} \approx 0$
2. Control in "simulation", with $w_{X_b} \neq 0$

After several tries, the final choices for the equilibrium LQR parameters are those reported in table 5.

	[Q]	R
"real-life" control, with $w_{X_b} \approx 0$	$diag([1e-10; 3; 0; 0.1])$	0.1
"simulation" controller, with $w_{X_b} \neq 0$	$diag([1; 3; 10; 0.1])$	0.1

Table 5: Values set to the Q and R matrices of the equilibrium controller. Two sets are provided, in order to accomplish the task of "real-life" or "simulation".

The results of both the simulation and the experimental data is discussed in chapter 8.

5.3 CONTROL OF STEERING SUBSYSTEM

The problem of controlling the steering subsystem consists in the control of the state-space system described in (31) and here reported:

$$\begin{aligned} \dot{x}_\delta &= A_\delta x_\delta + B_\delta \tau_\delta = \\ \begin{Bmatrix} \dot{\delta} \\ \ddot{\delta} \end{Bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & an_{66} \end{bmatrix} \begin{Bmatrix} \delta \\ \dot{\delta} \end{Bmatrix} + \begin{bmatrix} 0 \\ bn_{62} \end{bmatrix} \tau_\delta \end{aligned} \tag{62}$$

Which can be conveniently rewritten by using the second row of the (62):

$$\ddot{\delta} = an_{66}\dot{\delta} + bn_{62}\tau_\delta$$

And using the Laplace transform of this expression (with a slight abuse of notation), it results:

$$\begin{aligned} s\dot{\delta} &= an_{66}\dot{\delta} + bn_{62}\tau_\delta \\ \dot{\delta}\left(s - an_{66}\right) &= bn_{62}\tau_\delta \\ \frac{\dot{\delta}}{\tau_\delta} &= \frac{bn_{62}}{(s - an_{66})} \end{aligned}$$

Which takes to this transfer-function:

$$\frac{\dot{\delta}}{\tau_\delta} = \frac{bn_{62}}{(s - an_{66})} \tag{63}$$

Which is a first-order, one-pole transfer function.

After this brief introduction, different control strategies might be adopted, here we analyze the following two possibilities:

1. *LQR* steering controller
2. *PI* steering control

Those two strategies have been implemented and made available for simulation, as visible in figure 22.

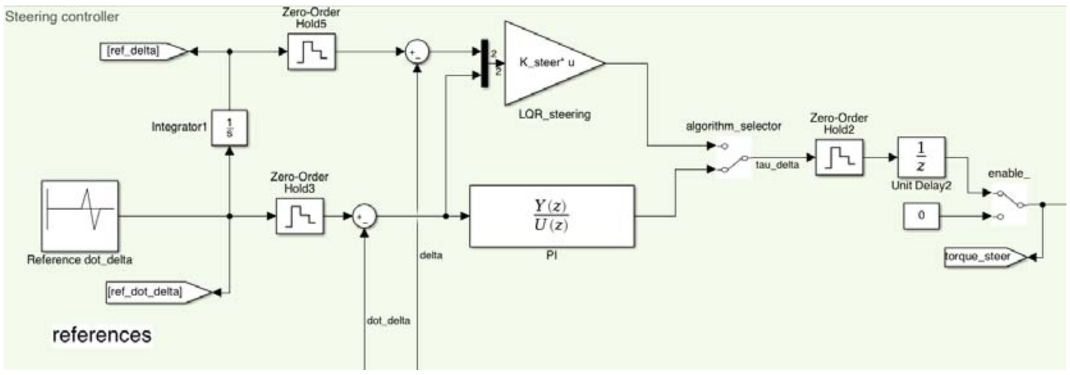


Figure 22: Steering control block: two different algorithms are available for simulation.

As noticeable in figure 22, the reference for both the systems is the *steering angle rate* and not the steering angle: this last quantity is in fact important only for the LQR controller and it is derived by time-integration of the previous reference. In this controller, differently from the equilibrium one, we did not differentiate the strategy between "real life" and "simulation", because the needs are the same. The only difference in the two situations is the reference: in simulation the reference comes from a "repeating sequence" block, while in "real-life", this comes from the position of a potentiometer (the steering command of a joystick).

In the following sections, a discussion about the application of the LQR and the PI control strategies.

5.3.1 *LQR steering controller*

The theory behind the LQR control was already discussed at the beginning of this chapter: here a brief discussion about the weights of the Q and R matrices of the controller. Even the discretization of this system is omitted, since the procedure is identical to that on the previous section. In this case the Q matrix is defined as:

$$Q = \begin{bmatrix} w_\delta & 0 \\ 0 & w_{\dot{\delta}} \end{bmatrix} \tag{64}$$

where w_δ and $w_{\dot{\delta}}$ are, respectively the weights related to *steering angle error* and *steering rate error*. Even here it is important to verify the stability of the controller, consequently to the choice of the previous weights. This deeply depends on the values of the state-evolution matrix. Like for the equilibrium controller, it is useful to plot the poles of the controlled system, as visible in figure 23.

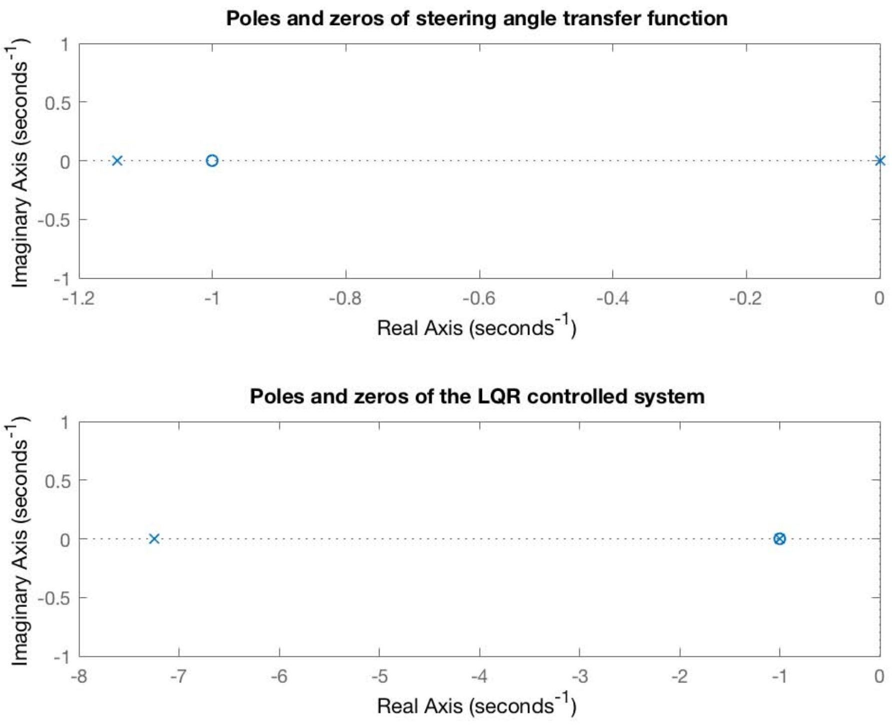


Figure 23: Poles of the steering system: too-negative poles might determine unstable controller.

The calibration of the "weights" can take into account the mentioned poles and try to reach a convenient trade-off between performances (with strong-negative poles) and stability ("slower" poles, but without the risk to work "near other system's components dynamics").

A good compromise for the coefficients of the Q and R matrices is reported in table 6. The results of both the simulation and the experimental data is discussed

	[Q]	R
LQR gains for steering controller	$diag([1,2])$	0.1

Table 6: Values set to the Q and R matrices of the steering controller.

in chapter 8.

5.3.2 *PI steering controller*

Looking at (63) it is clearly visible that the system to control is not a *MIMO*, but simply a *SISO* system, and additionally its transfer function is a first-order one. Because of this, a simple proportional controller might be sufficient, with the aid of an integral action, to avoid a steady-state error.

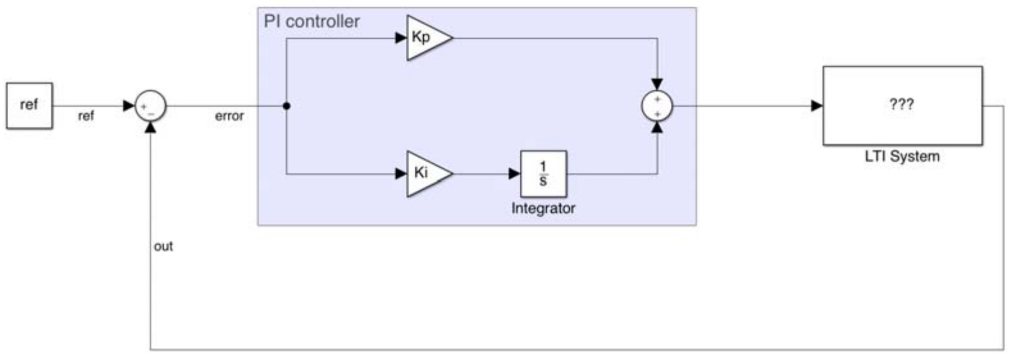


Figure 24: PI controller blocks.

With this basic idea a PI controller is designed. The first thing to do is to choose a suitable proportional gain (Kp), in odrer to reach the desired bandwidth (increase Kp to increase bandwidth).

Being the (63) a first-odrer system, theoretically the gain might be undefinitely be increased without compromising stability margins.

The truth is that in the system are present other (non-ideal) components, discussed in the following chapter, that might limit the maximum bandwidth and whose dynamics might interact with that of the controller (seamlessly adding poles), compromising stability in case of too-high gains.

In order to properly choose Kp and the consequent system bandwidth, it must be also considered the integral action. Because of this a brief mathematical digression is presented. Let's describe the PI transfer function, as easily derivable from figure 24

$$W_{PI} = K_p + \frac{Ki}{s} = K_i \frac{1 + s/(K_i/K_p)}{s} = K_i \frac{1 + s/\omega_{PI}}{s} \tag{65}$$

where ω_{PI} is the pulsation of the zero relative to the PI controller. The loop gain is then the product of the gains of PI controller and the system's transfer function:

$$\begin{aligned} W_{LOOP} &= W_{PI} * \frac{\dot{\delta}}{\tau_\delta} = K_i \frac{1+s/\omega_{PI}}{s} \frac{bn_{62}}{(s-an_{66})} = \\ &= \frac{K_i * bn_{62}}{-an_{66}} \frac{1+s/\omega_{PI}}{s} \frac{1}{(s/(-an_{66})+1)} \end{aligned} \tag{66}$$

Which is the combination of a static gain, a pole at the origin and one real-pole, plus a real zero. This loop gain's frequency behavior is visible in figure 25.

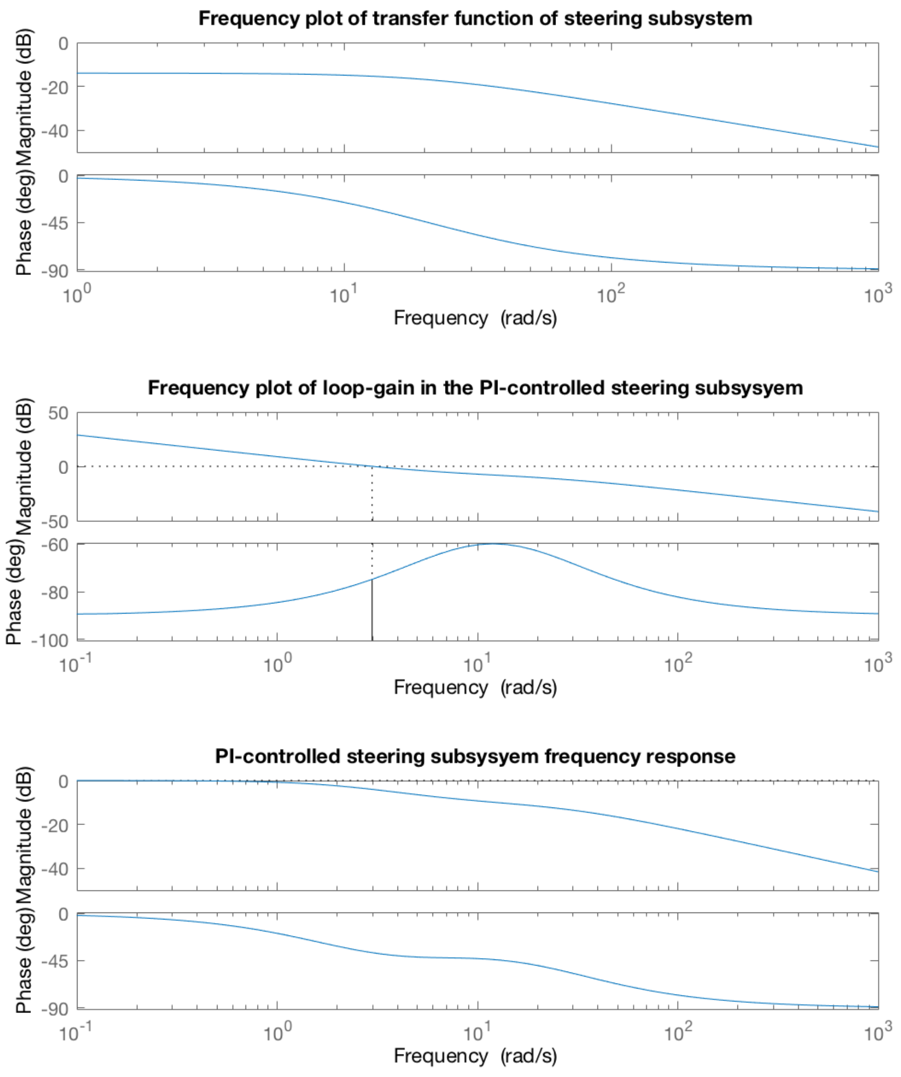


Figure 25: Frequency representation of the steering subsystem. In the upper graph the uncontrolled system, in the middle the loop-gain of the PI-controlled system, and in the lower the frequency response of the overall controlled system.

Now it is more simple to present the principle of gains selection: the Ki have been chosen so that the zero at ω_{PI} will present the minimum interaction possible with the pole at $-an_{66}$: ideally one third "slower". The Kp gain was chosen also in a way to guarantee a loop-gain bandwidth of (at most) equal to the original system's pole in $-an_{66}$.

$$K_i = K_p \frac{-an_{66}}{3} \tag{67}$$

This combination of criterion, whose result is in figure 25, took to the choice of gains (calculated by imposing a "cylinder body" of $1m$ height and $30kg$ weight) of: Even for the PI steering controller, both the simulation and the experimental results will be presented in chapter 8.

	K_p	K_i
PI coefficients for the steering controller	2	13.8241

Table 7: Values chosen for the PI steering controller, in the case of a rigid-cylindric-body $1m$ height and $30kg$ weight.

5.4 CONCLUSIONS

In this chapter it was described the control algorithms chosen for the two subsystems that forms the *TWSBT*. It was described the motivation of the controller type and the tuning strategies. Also it was reported a discussion about the stability.

All the controllers described here are designed on the base of an "ideal" system: it is a simplistic approach and lacks of many real-life limits that might modify a lot the controller performances and its stability, but a first model-based control design is a good base for further tuning, based on real-life non-idealities.

In the following chapter, a detailed description of those real-life non-idealities, and how they were modeled, and used to properly tune the described controllers.

6 SYSTEM NON-IDEALITIES

6.1 INTRODUCTION

In chapeter 3 it was introduced a basic model, useful for the study of the dynamics of the system. Such a system was useful for a first approach to the problem such as the design and testing of equilibrium and steering controllers, but it lacked of a high number of real-life phenomenons that acts on the system. In the successive chapter, a first evolution of the model aimed to build a more realistic one, by treating the human driver as "something different from a solid body, solidal with the base", that is a first step in the direction of non-idealities description.

Further model evolutions might consider the complex continous elastic behavior of the mechanical parts, but with a realistic consideration, this complication can be neglected, since the vibration modes will seamlessly act at high frequencies that does not influence much the equilibrium and the steering controllers and even if, they can't be managed, because they acts outside the controller's bandwidth.

In the following subsections it will be described other system non-idealities that might concretely influence the control strategy and how those unidealities were managed in the design of the self-balancing transporter. In addition, in order to try to design a model, as realistic as possible, these unidealities were also (where possible) added to the model in chapter 3.

6.2 MOTOR TORQUE

In an ideal model, the torque generated by a motor is constant and proportional to the supplied current:

$$\tau_{mot} = k_t * i_{mot} \tag{68}$$

thus, driving the motor with a good current controller, ideally corresponds to a direct control of the motor's torque.

Unfortunately the real-world is different: the relation between current and torque is linear only for some kind of motors, and only in a limited frequency range. Most cheap motors suffer of non-constant torque constant, exactly as the Brushless-DC motors employed in this project (see figure 26).

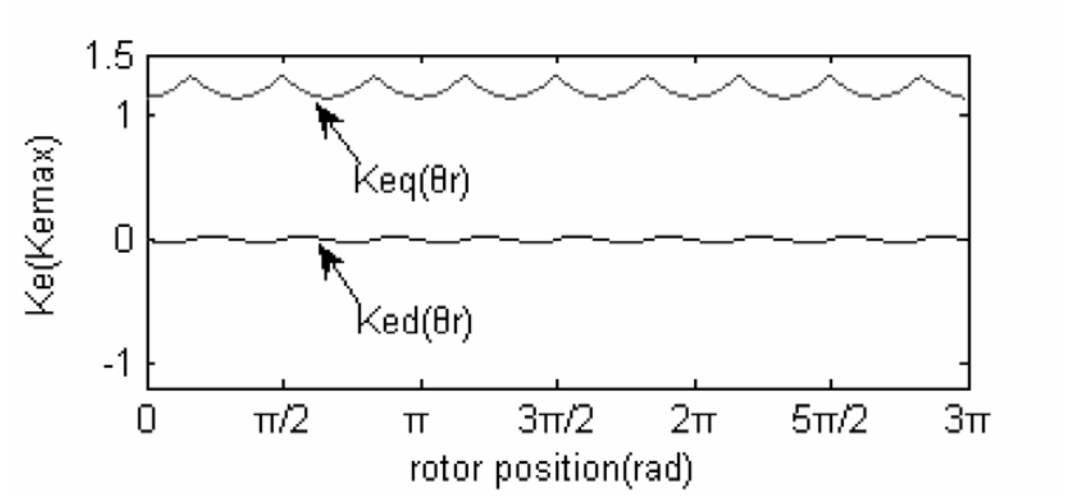


Figure 26: Torque ripple in a BLDC motor, due to a variation of $k_e \approx k_t$ along the electrical angle. Figure from [29].

In the Brushless-DC (*BLDC*) motors, the value of k_t is not constant along the rotation of the motor, and results in a *torque ripple* during the motor rotation that repeats at six-times the rate of rotation.

$$k_t(\theta_r) = k_e(\theta_r) = \frac{d\lambda_e(\theta_r)}{d\theta_r} \tag{69}$$

This phenomena is acceptable for many applications, where the motor works at high-rotation-speeds, because the load inertia and the friction has a low-pass effect on the torque transmission, resulting in a "sufficiently smooth" motion, at the endpoint of the transmission chain. As opposite, this torque-ripple can result unacceptable for many industrial processes, having a need of high-precision and regularity.

The design of a self-balancing vehicle is partially a low-cost transportation application, where the torque ripple is not a big matter, but at the same time, the torque ripple could strongly influence the on-board accelerometer reading, thus seriously disturb, or even compromise the reading of the pitch angle, especially at low-speed.

Another fact is that the torque-ripple might probably excite some vibration modes of the structure, thus it might affect not only the controller, but can even compromise the long-term mechanical structure of the product, with possible failures.

6.2.0.1 *Torque vs driving technique*

In our application (for a matter of cost and dimension) it was not feasible to think about the use of sinusoidal morots, so eliminating the pure-DC motor, the *BLDC* choice was almost obvious. In the *BLDC* world, the motor construction and quality, strongly influences the magnitude and form of the ripple torque, thus a first choice of low-torque-ripple motor was made.

Even with this, the problem of the ripple torque will always be, as long as a *BLDC* motor will be employed. Nevertheless, if the choice of a *BLDC* motor is almost compulsory, the choice of the driving technique might play a decisive improvement in the torque ripple. Several studies such as [18] and [17] have tested the possibility to apply some advanced control techniques like Field-Orient Control (*FOC*) in order to reduce the torque ripple in the motor.

The scheme of the *3-phase BLDC FOC* control is shown in figure 27

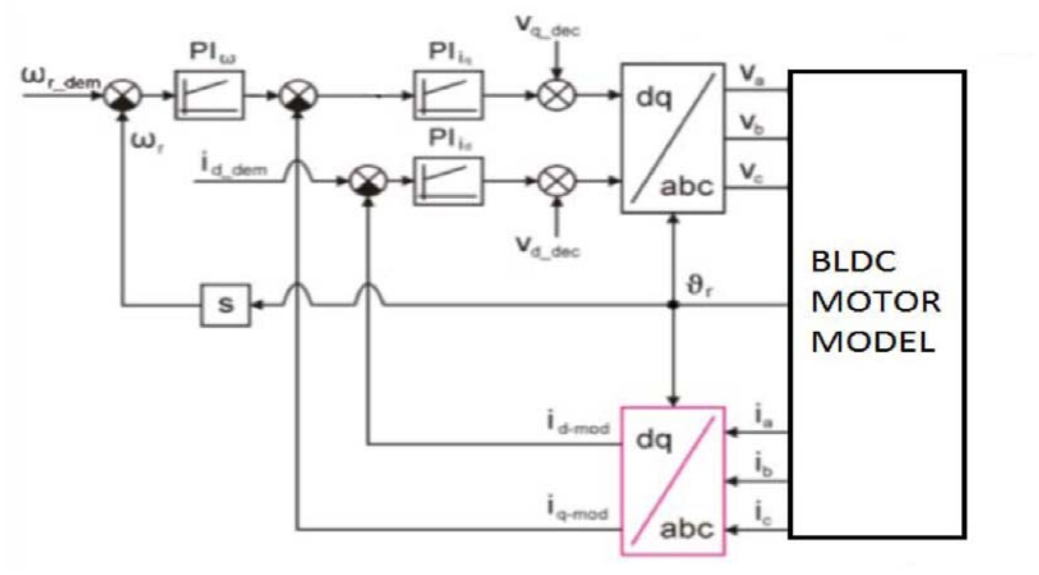


Figure 27: Block-diagram of a field-oriented control for a *3-phase BLDC motor*. Figure from [17].

This control technique, widely used in the motor control world is mainly based on the knowledge of the 3-phase motor's model (reported on figure) and it's parameters.

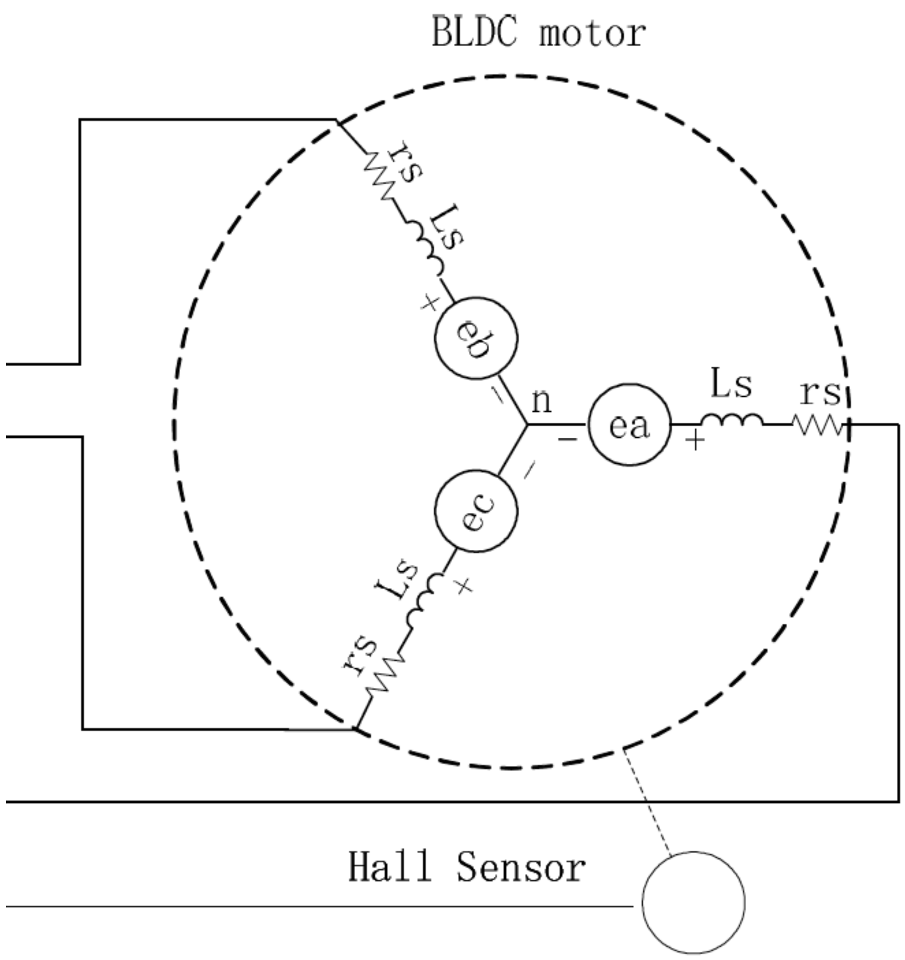


Figure 28: Electrical schematic (simplified) of a BLDC motor. Figure from [29].

Infact, by precisely orienting the magnetic field generated by the stator currents (named i_a , i_b and i_c), against the flux linkage, generated by the permanent magnet, it is possible to directly control the torque generated by the motor.

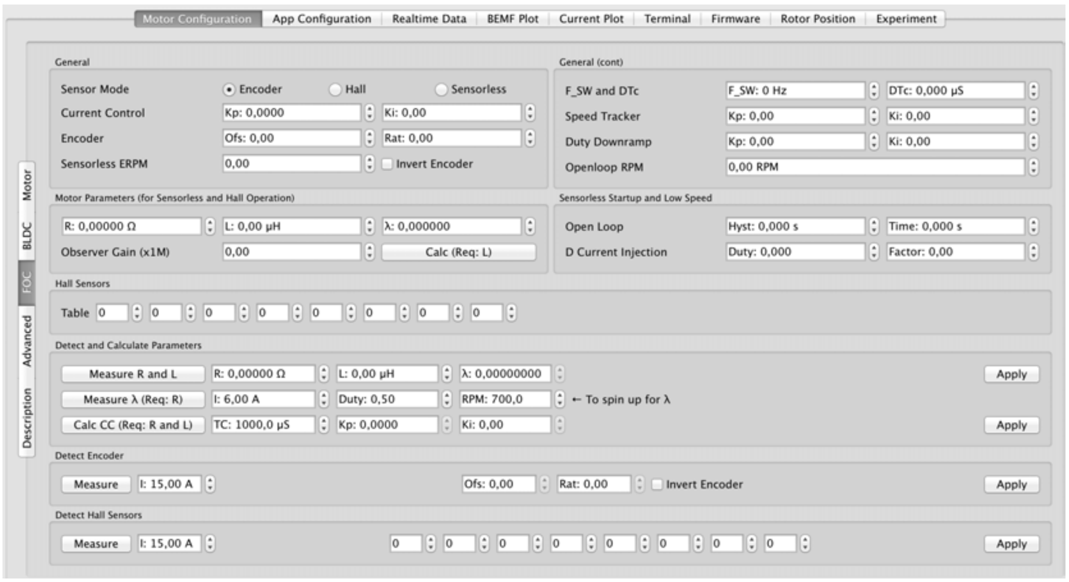
The application of the field-oriented control to the permanent-magnet motors, was originally on the sinusoidal *PMSM*, while an application to the *BLDC* was illustrated in [18].

The complete FOC theory will not be reported, but what is important is that for a proper control, the motor must be *IDENTIFIED*: the parameters shown on figure 28 must be measured or calculated.

In the board used in the project, a simple graphical interface (figure 29) made it possible to do the automatic identification process by measuring/calculating:

- winding resistance
- winding inductance
- flux linkage
- hall position
- current controller gain

This simple, but convenient GUI made it possible to "tune" the controller by simply connecting with an USB cord, the PC to the controller board.



General
Sensor Mode: (● Encoder) ○ Hall ○ Sensorless
Current Control: Kp: 0,0000 Ki: 0,00
Encoder: Ofs: 0,00 Rat: 0,00
Sensorless ERPM: 0,00 ☐ Invert Encoder

General (cont)
F_SW and DTc: F_SW: 0 Hz DTc: 0,000 µS
Speed Tracker: Kp: 0,00 Ki: 0,00
Duty Downramp: Kp: 0,00 Ki: 0,00
Openloop RPM: 0,00 RPM

Motor Parameters (for Sensorless and Hall Operation)
R: 0,00000 Ω L: 0,00 µH λ: 0,000000
Observer Gain (x1M): 0,00

Sensorless Startup and Low Speed
Open Loop: Hyst: 0,000 s Time: 0,000 s
D Current Injection: Duty: 0,000 Factor: 0,00

Hall Sensors
Table 0 0 0 0 0 0 0 0

Detect and Calculate Parameters
Measure R and L: R: 0,00000 Ω L: 0,00 µH λ: 0,00000000
Measure λ (Req: R): I: 6,00 A Duty: 0,50 RPM: 700,0 ← To spin up for λ
Calc CC (Req: R and L): TC: 1000,0 µS Kp: 0,0000 Ki: 0,00

Detect Encoder
Measure I: 15,00 A Ofs: 0,00 Rat: 0,00 ☐ Invert Encoder

Detect Hall Sensors
Measure I: 15,00 A 0 0 0 0 0 0 0 0

Figure 29: Graphical interface of the board controller: motor identification panel.

6.2.0.2 *FOC torque ripple benefits*

Unfortunately, during the measurements there were no instruments to measure the motor torque, in order to do a comparation between the FOC-driving, against a sector-wise constant-PWM driving, thus a "scientific measure" of the FOC benefit, against a constant-PWM driving was possible.

The only appreciable fact was a deep decrease in motor's noise and a higher-perceived regularity, during the movement.

In the theoretical model introduced in3, thanks to the adoption of the FOC strategy, it was not added any parasitic ripple-torque to the motors, (since it was verified that the BLDC motor was "naturally smooth"), and the further introduction of this control technique reduced the ripple-torque at the point that there was no need to consider the ripple in the model.

Concerning the FOC model, it was simply modeled as a *current controller* block, described in the following section.

Another note is that probably the terrain irregularities will have a higher disturbance effect, than the torque ripple, so in the model it was introduced only disturbance torques (one for each motor), representing this last phenomenon.

See figure 30.

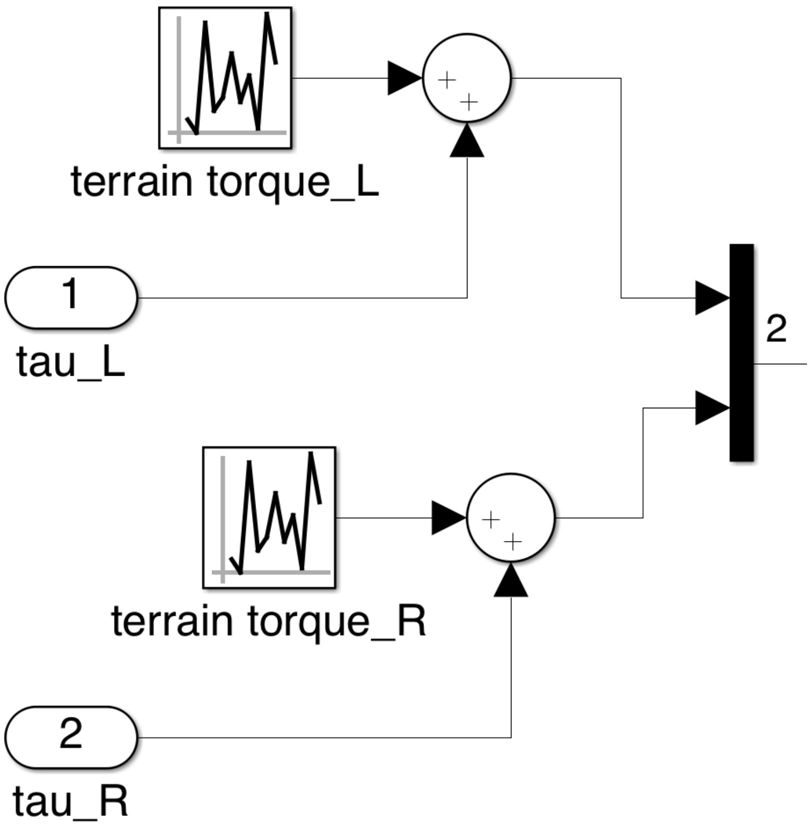


Figure 30: Noise torque applied to the motors: it represents mainly the terrain irregularities and other uncontrolled external torque disturbances.

6.3 CURRENT LOOP

Some studies of self-balancing vehicles in literature, includes in their dynamic model, a complete or approximated model of the motor(s) of the vehicle, resulting in a structure where the input signals are the motor's voltages and the outputs are the same as that in the model of chapter 3.

This is one possible approach, but might result in a complication of such model and the impossibility to separate it in a series of conceptually different blocks, thus the impossibility to separate the study of single blocks.

Another approach is that of the chapter 3, where it was assumed the presence of a *current controller*, which corresponds to a seamless direct control of the torque into the motors. In the simplest form infact, there is a linear correlation (k_t) between current and torque.

The exceptions about this linearity were all treated in the previous subsection, so in this section, it will be presented only a brief discussion about the non-idealities strictly related to the current control-loop, that influences the overall behavior of both the equilibrium and the steering controllers.

6.3.0.3 *current limit (saturation)*

In the ideal world, a motor supplies a torque proportional to the current flowing into it, but the problem is that the current is not *infinite*, but has natural or imposed limits.

symbol	value	unit
r_s	0.128	Ω
L_s	167	μH
k_t	0.694	V/(rad/s)
$V_{bus,MAX}$	36	V
P_{NOM}	350	W

Table 8: Parameters of the *Hub Motor* described in chapter 2.

A first current limit is the bus voltage: by the ohm's low, there is an intrinsic current limit by the resistance of the motor's winding, thus, referring to figure 28:

$$i_{a,peak,MAX} = \frac{V_{bus}}{2r_s} \tag{70}$$

obviously this is a *Maximum Value*, at steady speed: when the rotor starts spinning, the current limit will obviously be reduced because of the back-EMF generated by the e_a, e_b, e_c generators:

$$e_a = e_b = e_c = \omega_m \cdot k_e \approx \omega_m \cdot k_t \tag{71}$$

As an example, the *Hub Motor*, described in chapter 2 has the parameters listed in table 8.

Considering such parameters, it results a maximum resistance-limited current of:

$$i_{a,peak,MAX} = \frac{V_{bus}}{2r_s} = \frac{36}{2 \cdot 0.128} \approx 140 \text{ [A]} \tag{72}$$

Another current limit is imposed by the motor's constructor: the Joule's power dissipated by the winding resistance will heat-up the motor and make it reach a critical temperature. This limit is often stricter than the previous.

For many RC motors the current limit given by the producer has 2 values:

- *Maximum Continous Current* : tolerated for unspecified time, without cousing damages to motor
- *Maximum Pulse Current* : a current tolerated for a limited time (typically for one minute) that is almost the double of the previous

The *Hub Motor* had no explicit current limit, thus it was derived from the power rating of table 8. Having the mentioned nominal power, it can be calculated the nominal current as:

$$i_{mot,NOM} = \frac{P_{NOM}}{V_{bus}} = \frac{350}{36} \approx 10 \text{ [A]} \tag{73}$$

Thus it is admitted a "pulse current" of about double the previous, resulting in:

$$i_{mot,MAX} \approx i_{mot,NOM} * 2 \approx 20 \text{ [A]} \tag{74}$$

This limit was modeled as a "saturation block", visible in figure 31.

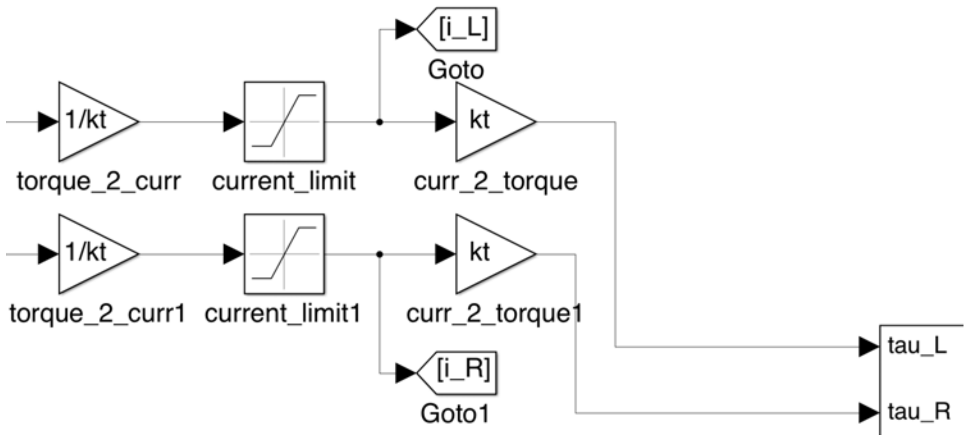


Figure 31: Current limit in the current controller. In the figure it is also noticeable the simplified linear relation (k_t) between current and torque.

6.3.0.4 *control loop delay*

In its simplest form, the *current-control loop*, is a simple *Proportional* controller which acts on a *PWM* signal that regulates the voltage (v_{L_s}) across the motor's stator inductance.
In this simplification, infact, it was considered only the high-frequency behavior (the most important effect of control loop), thus neglected the effect of the back-EMF and the resistance of the winding, because they act only at low-frequencies.

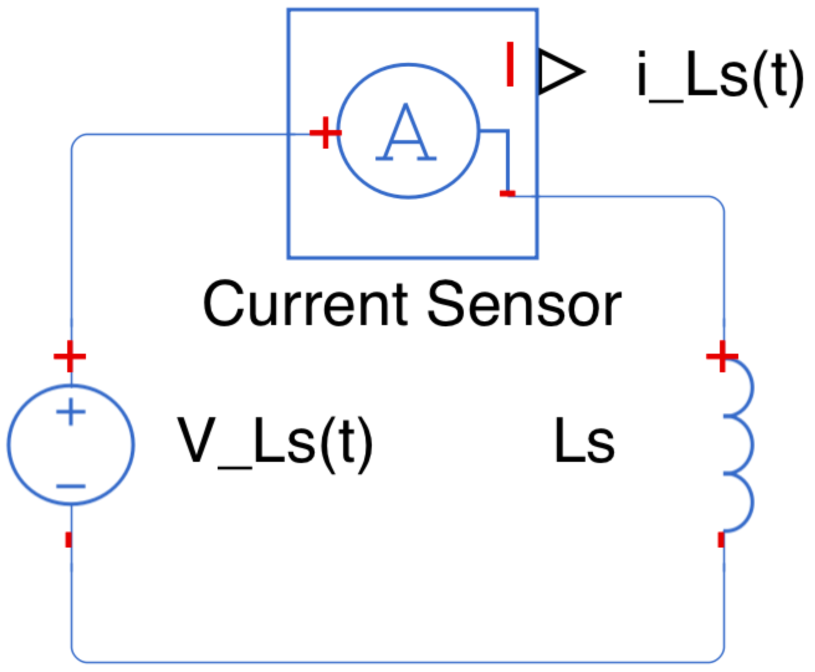


Figure 32: Variable voltage across inductor and corresponding current.

Referring to figure 32, the current through the L_s inductor is:

$$i_{L_s}(t) = \frac{1}{L}\int_0^t v_{L_s}(t)dt + i_0 \tag{75}$$

If we apply a simple proportional controller to the above circuit, the result is a scheme like that in figure 33.

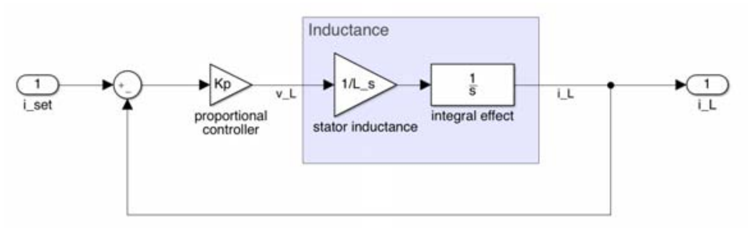


Figure 33: Block-model of a current proportional controller, acting on stator's inductor.

The effect of such a controller is a further high-frequency pole at a frequency of $2\pi \cdot (Kp/Ls)$, as visible in figure 34, which differs a lot from an ideal current actuator since it reduces significantly the gain margin of the controllers acting at the input of this block.

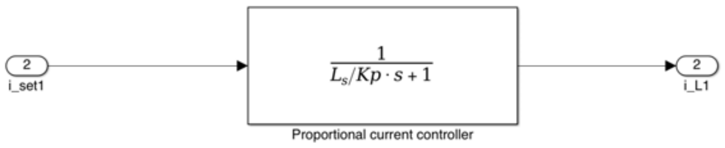


Figure 34: The control scheme represented in figure 33, is equivalent to a high-frequency pole at pulsation of $Kp/Ls\ rad/s$.

Any controller acting on this block's input must infact take into account of the presence of the further pole introduced by this block: the phase shift and the delay produced by it, might represent a stability risk for both the equilibrium and the steering controllers.

The final model for the current controller's and the saturation effect is visible on figure 35.

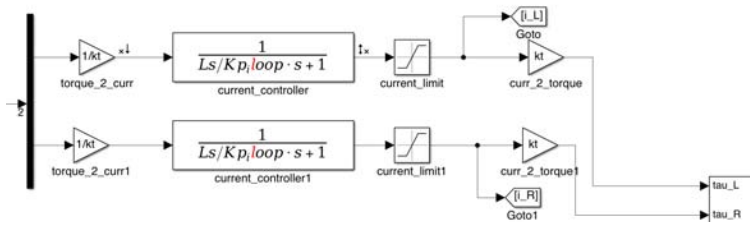
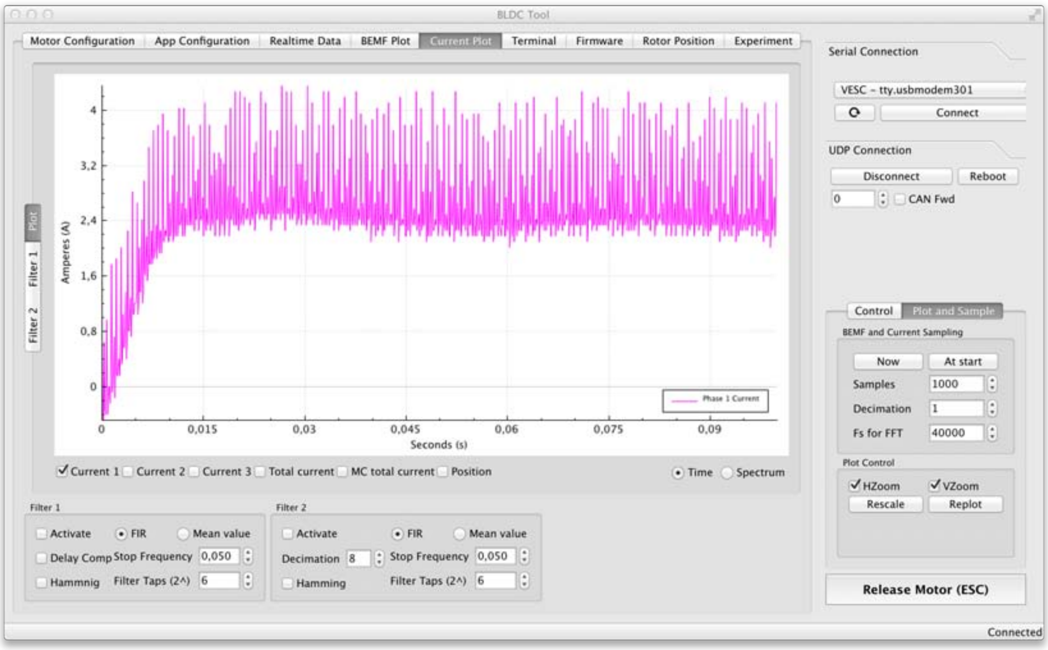


Figure 35: Final model for the current controller: the linear effect of a low-pass filter, plus the nonlinear effect of current saturation.

In the current loop of the vehicle designed, a *FOC* controller was employed, with a *PI* current controller instead of a simple *P* control. This is not a big deal since the final effect is only a first-order-like LPF transient, visible in figure 36. This transient makes it possible to calculate the τ time constant, used in the block of figure34 inserted on the overall device model.



VESC - tty.usbmodem301

UDP: 0 ☐ CAN Fwd

Samples: 1000
Decimation: 1
Fs for FFT: 40000
✓ HZoom ✓ VZoom

✓ Current 1 ☐ Current 2 ☐ Current 3 ☐ Total current ☐ MC total current ☐ Position
[● Time] ○ Spectrum

Filter 1: ☐ Activate [● FIR] ○ Mean value ☐ Delay Comp Stop Frequency 0,050 ☐ Hammnig Filter Taps (2^) 6

Filter 2: ☐ Activate [● FIR] ○ Mean value Decimation 8 Stop Frequency 0,050 ☐ Hamming Filter Taps (2^) 6

Figure 36: Current transient, with blocked rotor, measured at one of the motor-phases. Its effect is similar to that of a first-order LPF, thus it is possible to estimate its time-constant and put it into the model of figure 34.

6.4 angle and angle-rate reading

The equilibrium control of almost every kind of self-balancing vehicle is deeply based on the pitch angle (and angle-rate) reading. The most diffused method for the pitch-angle reading of mobile platforms, is the use of an Inertial Movement Unit (IMU).
IMU units are generally sensors that combines accelerometers and gyroscopes that senses one or more physical coordinates (generally they offer 2D or 3D sensing capabilities). The reading of the pitch angle with those devices is generally obtained by a sensor-fusion, described in the following subsections.

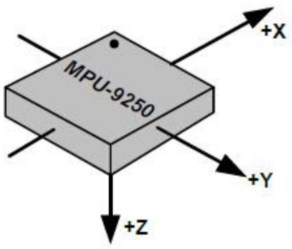


Figure 37: IMU magnetometer reference axes. Figure from [28]

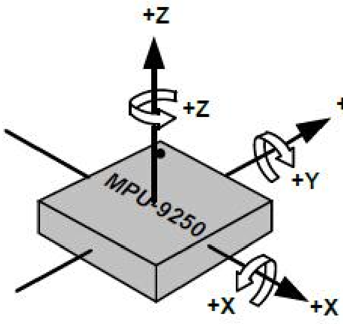


Figure 38: IMU accelerometer and gyroscope rotation directions. Figure from [28]

IMU-based angle reading is preferred because it permits the reading of the vehicle's base inclination, without the necessity to have any physical link to the ground and, most important, in the panorama of contact-less angle reading, it is one of the simplest methods (against ultrasound proximity sensors, visual sensors, laser etc.).

The mass-diffusion of such inertial sensors make them smaller and cheaper year-by-year. Another interesting feature is the deep-integration that those sensors offer: only few years ago, it was impossible to think about a 3-D accelerometer combined with a 3-D gyroscope without an expensive-price and huge dimension. Nowadays it is common to find not only 3-D accelerometer combined with a 3-D gyroscope, but even a 3-D magnetometer in the same (incredibly small) 3x3mm package (avoiding even the problem of misaligned sensor's origins).

Even with all those interesting features, those sensors' output can't be directly employed for the angle-reading task. As just reported, the signals must be "mixed" together, filtered (avoiding an excessive measurement noise) and the sensors must be calibrated, because they are non-ideal sensors.

In the following sections, the description of the sensor's models and all the signal conditioning for the angle and angle-rate reading, which is one of the challenging tasks in many control projects.

6.4.1 *IMU modelling and calibration*

The accelerometer and the gyroscope of an IMU are non-ideal sensors, whose output can suffer of several non-idealities, that can strongly compromise the angle-reading performances. The main non-idealities are listed:

1. Bias: almost always the accelerometer shows a non-null acceleration in the axis perpendicular to the graound, or even an acceleration different from "g" in that parallel: this is very common and requires an adjustment, in order

to supply a correct output. Even the gyro can output a non-null angle-rate, even with steady conditions.

2. Non-linearity: it is supposed that in the measurement range of the sensor, its output is always linear, but it is very common that at the extremes of the measurement range, a meaningful linearity distortion appears.

3. Gain: even having canceled the bias, it can appear that when flipping an accelerometer upside-down it shows a $+/-8.5m/s^2$ acceleration variation, instead of a $+/-9.81m/s^2$. This is the typical gain error.

4. Noise: as all "true" sensors, accelerometers and gyroscope suffers of noise presence, that must be conveniently managed.

5. Saturation: all the sensors can provide a measure in a limited range. When the true phenomena exceeds this, a saturation in the measurement might appear.

In this subsection it will be presented a model for both the accelerometer and the gyroscope, that can conveniently replicate the sensor's behavior in simulation, and a *calibration* procedure, that aims to minimize or eliminate some of those unidealities.

6.4.1.1 *Accelerometer model*

The accelerometer is one of the simplest ways for measuring the angle, with some limitations that will be soon explained. Basically the model of a 3-D accelerometer is described by:

$$A(t) = \begin{Bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{Bmatrix} = \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} + \begin{Bmatrix} n_x(t) \\ n_y(t) \\ n_z(t) \end{Bmatrix} + \begin{Bmatrix} acc_x(t) \\ acc_y(t) \\ acc_z(t) \end{Bmatrix} \tag{76}$$

Where $A(t)$ is the vector of raw accelerometer data, b_* are the accelerometer biases, n_* are the measurement noises (supposed as normal-distributed, zero-means noises $\mathcal{N}(0, \sigma^2)$) and acc_* are true accelerations.

Concerning the noises, the accelerometer's datasheet reports the value of $(300\mu g\sqrt{Hz})$, which is a constant-power, normally-distributed noise. It was modeled with a limited-bandwidth, white-noise block, visible in the models' scheme of figure 39.

In order to have a good measure, it remains the problem of bias cancellation, which is a part of the process called *calibration*.

The activity of IMU calibration is a very common engineering task and several different solutions have been proposed for this, such as [4], [32] and [26].

In the specific case of our system, the IMU (as stated in the datasheet) have been factory-calibrated in the gain value, for both accelerometer and gyro.

Even the non-linearity of the sensors were neglected by the fact that the device is used in the measurement of a very small angle range.

At the end, the only task remaining, concerning the accelerometer calibration is the accelerometer bias correction. A proposed method is the following.

Since the noise is supposed null-means, it won't appear in the average of measures. Giving this, simply averaging a sufficient number of accelero measures, it is possible to retrieve the sensor's bias in this way:

$$\begin{aligned}
avg(A(t)) &= avg\left(\begin{Bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{Bmatrix}\right) = \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} + avg\left(\begin{Bmatrix} acc_x(t) \\ acc_y(t) \\ acc_z(t) \end{Bmatrix}\right) \\
\text{thus}: & \\
\begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} &= avg\left(\begin{Bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{Bmatrix}\right) - avg\left(\begin{Bmatrix} acc_x(t) \\ acc_y(t) \\ acc_z(t) \end{Bmatrix}\right)
\end{aligned} \tag{77}$$

The bias removal was obtained simply by positioning the IMU parallel to the ground sequentially, in the three accelero coordinates and removing the value read in the axes parallel to the ground.

The bias-free measures are then:

$$\begin{Bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{Bmatrix} - \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} = \begin{Bmatrix} acc_x(t) \\ acc_y(t) \\ acc_z(t) \end{Bmatrix} + \begin{Bmatrix} n_x(t) \\ n_y(t) \\ n_z(t) \end{Bmatrix} \tag{78}$$

In this way, the accelerometer is *calibrated*, thus the values are unbiased and only affected by Gaussian noise. The model of the accelerometer is visible in figure 39, where only two coordinates were considered, since the equilibrium problem (for which the sensor is employed) is basically a bi-dimensional matter.

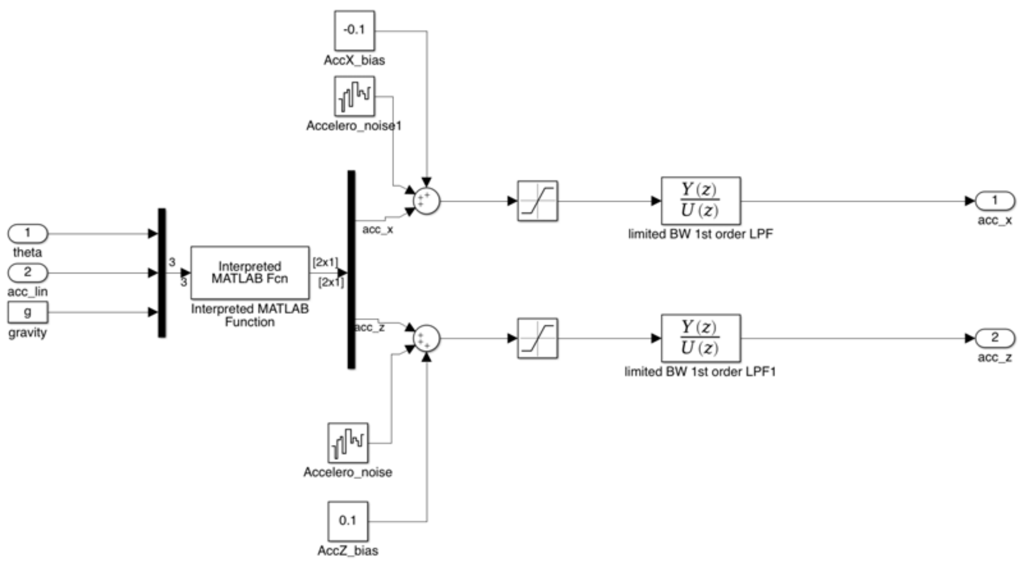


Figure 39: Block-model of the accelerometer unit.

The saturation block represents the full-scale of the accelerometer. Several values can be set. In our design it was chosen $\pm 2g$. The output filter is a configurable first-order LPF block, present inside the component that can be set at various values. In our case it was left to the default value of 250Hz.

The "Interpreted MATLAB Fcn" block of figure 39 represents the accelerometer's sensitivity for gravity and other accelerations, which are basically the linear acceleration of the base. The acceleration scheme is reported in figure 40.

The linear acceleration affects the measure, creating an *acceleration coupling* in the sense that there exist a coupling between linear acceleration and angle reading, thus the angle measure is no more precise as expected because affected by an acceleration other than the gravitational, that corrupts the angle measure. Other considerations about this problem are in the following sections.

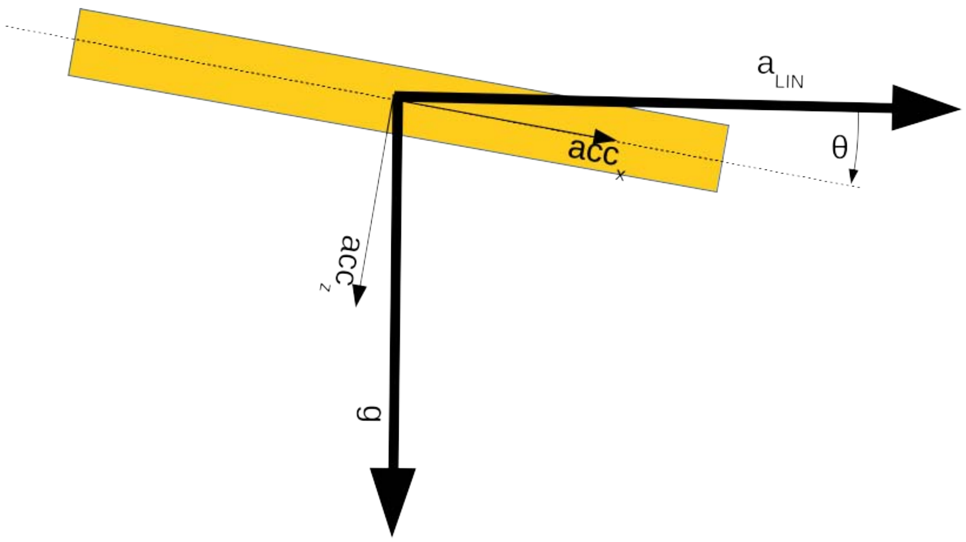


Figure 40: Accelerometer reading affected not only by the gravitational acceleration, but also by the movement-correlated linear acceleration

Longitudinal acceleration in fact adds to the x (and z) components of the accelerometer a quantity like:

$$\begin{aligned} acc_x &= g * \sin\theta + a_{LIN} * \cos\theta \\ acc_z &= g * \cos\theta - a_{LIN} * \sin\theta \end{aligned} \tag{79}$$

Considering equation 81, the corresponding accelerometer's angle will be deeply affected by any linear acceleration.

6.4.1.2 *Gyroscope model*

Concerning the gyroscope model, the equations governing it are:

$$G(t) = \begin{Bmatrix} g_x(t) \\ g_y(t) \\ g_z(t) \end{Bmatrix} = \begin{Bmatrix} b_x(t) \\ b_y(y) \\ b_z(b) \end{Bmatrix} + \begin{Bmatrix} n_x(t) \\ n_y(t) \\ n_z(t) \end{Bmatrix} + \begin{Bmatrix} \omega_x(t) \\ \omega_y(t) \\ \omega_z(t) \end{Bmatrix} \tag{80}$$

In this case, differently from the accelerometer, the bias is not simply a constant value (in fact it shows a *time-drift*) and its cancellation plays a key-role in the control strategy. Several models like such as [27] and [24] are proposed in literature, about this bias phenomena and strategies for its cancellation.

In our case, the bias of gyroscope was modeled with a constant-part and a random-walk one, representing the bias-drift, at low-frequency and low entity. The complete model is shown in figure 41.

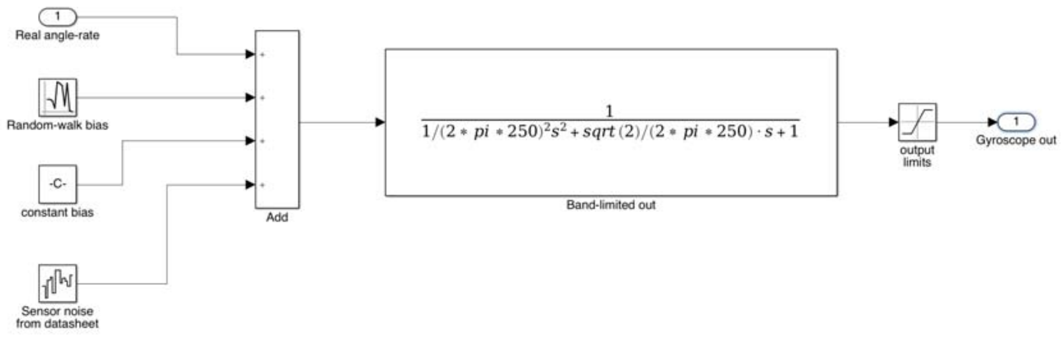


Figure 41: Block-model of the gyroscope unit.

As for the accelerometer, a saturation block represents the full-scale limit (in our case $250°/s$), and the output is limited in bandwidth (in our case it was left the default value of $256Hz$).

For both the accelerometer and gyro models, a validation process will be presented in chapter 8.

6.4.2 *angle measurement*

Looking at figure 37, under the assumption that the only acceleration acting on the IMU is the earth's g, it is possible to calculate the pitch angle by simply knowing the acceleration's x and z components, infact:

$$\theta = \arctan\left(\frac{acc_x}{acc_z}\right) \tag{81}$$

Since the angle reading must be performed many times per second, and the micro-controller has limited calculus power, the arctan operation might result in a heavy-weight for the micro-controller. In order to lighten the calculus load, it can be considered that equation (81) is a nonlinear function, however, we can approximate the θ angle through a straight line as it is showed in Figure 42, thus ($\sin\theta \approx \theta$) for small values of θ. In Figure 42, it can be observed that this linear approximation is acceptable in the interval $\pm 0.35rad$, that is $\pm 20°$. This range is large enough to contain the normal operation of the self-balancing vehicle. Therefore we can rewrite again the equation (81) in the approximate form:

$$\theta \approx \frac{acc_x}{acc_z} \tag{82}$$

The equation (82) is linear and it is meaningful as long as the system maintains the pitch angle in the mentioned range $\pm 20°$.

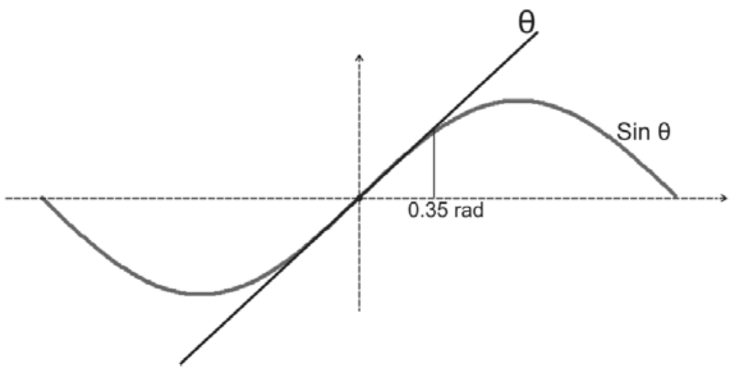


Figure 42: Linear approximation of angle: valid in the interval of ±20°. Figure from [25]

The presented technique is consistent as long as the only acceleration acting in the system is the gravitational one. Unfortunately, for the self-balancing vehicle, this assumption is almost never met, because of the presence of vibrations, longitudinal accelerations and side accelerations. This raises the problem of a very noisy reading, that can't be directly used for the purpose of angle-reading of self-balancing vehicle. The most typical procedure adopted for avoiding this problem is the *sensor fusion* of measures from accelerometer and gyroscope. Even by fusing accelerometer and gyroscope data, the angle measure is affected by some neglected accelerations (other than the gravitational) acting on the accelerometer. Some of them can be considered under the definition of "noises", uncorrelated with the movement, but some other not. As described in the accelerometer model, the vehicle acceleration represents an *acceleration coupling* that might alter measures.

Since the linear acceleration is known by the equilibrium algorithm (it is derived by double-derivation of wheels angle), the simplest thing to do, is to subtract it from the x-component of the accelerometer reading.

This simple solution is possible, under the assumption of working around small angle, where it is possible to linearize the equation(79), thus considering ($\cos\theta \approx 1$) and ($\sin\theta \approx \theta \approx 0$), thus neglecting the effect of linear acceleration on the z-component. If the angle is out of a small range, it is not possible to simply linearize it and subtract from the x-component: it is a recursive, hard to manage problem.

In the realization of the self-balancing vehicle, the condition of small-angle is always verified, so that the acceleration was simply removed only from the x-component. This subtraction was made "select-able", in order to test the effectiveness of the solution. This possibility, as long as the accelerometer's angle calculation is shown in figure 43.

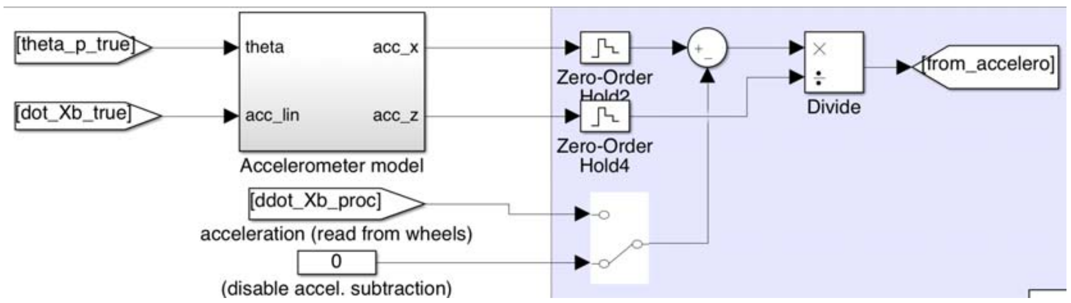


Figure 43: Model of angle reading from accelerometer: note the possibility to test the benefit of subtracting the acceleration calculated, from the x output of the accelerometer.

Since the accelerometer has many drawbacks about sensitivity of spurious accelerations, it might be obvious to measure the angle by integrating the gyroscope's output. This is obviously not feasible, since gyroscope has always a bias, thus its integration will result in an angle constantly increasing (or decreasing), or (since the bias has also a time-drift) a variable contribution to the angle. In this project, to overcome those problems, having different sensors, with different pros and cons, it is possible to use the *sensor fusion*.

6.4.3 *Sensor fusion*

This technique aims to take advantage of each (accelerometer and gyroscope) sensor, fusing measures into a unique high-quality output.

Several sensor-fusion techniques are possible, as proposed in [25], [31] and [21] .

A first simple one is the *Complementary Filter* and its structure is:

$$\begin{aligned} \theta_{filt} &= \alpha * (\theta_{filt} + gyro * T_s) + \beta * \theta_{acc} \\ & \quad where: \\ \alpha + \beta &= 1 \\ & \quad and \\ \alpha >> \beta & \end{aligned} \tag{83}$$

The gyroscope data is integrated every time-step with the current angle value. After this, it is combined with the angle read from the accelerometer.

This simple filter weights a lot the gyro integration, because in the rapid-movement it is very affordable, but avoids its drift because integration involves the accelerometer data that is more affordable in slow "long term" horizon, canceling the bias contribution given by the gyroscope's integration.

An example of such filter's behavior is shown in the Figure 44, where the accelerometer was read at a rate of 100Hz, and the complementary filter's coefficients were: $\alpha = 0.98$ and $\beta = 0.02$.

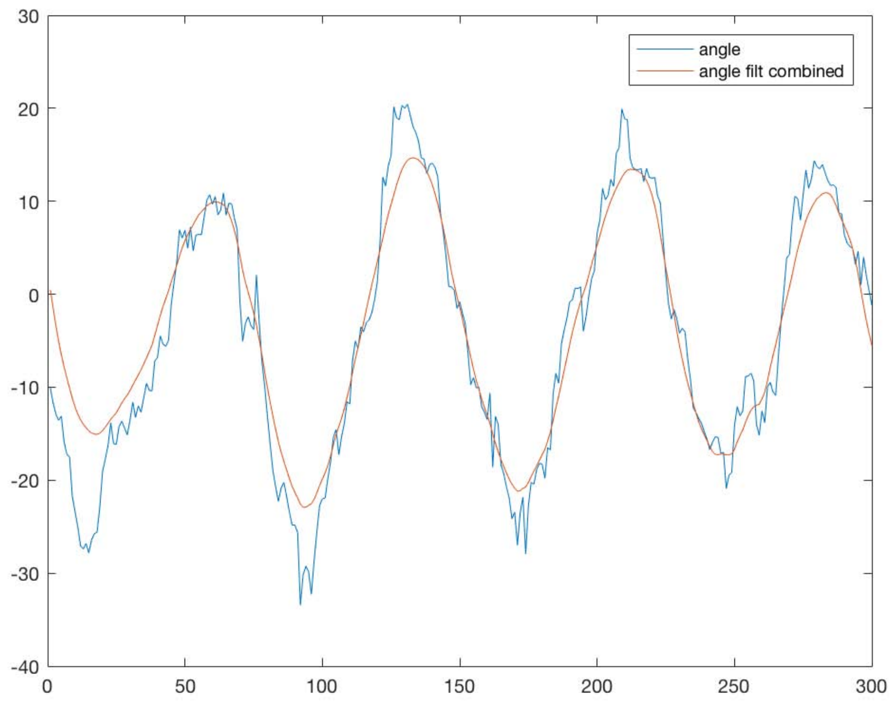


Figure 44: Comparison of angle reading with accelerometer and complementary filter. In this setup $\alpha = 0.98$ and $\beta = 0.02$

The result is good, but in some cases it might result "too noisy", or "too slow", if not properly calibrated. The complementary filter must then be tuned properly, in order to give the expected output. A model for this filter is presented in figure 45.

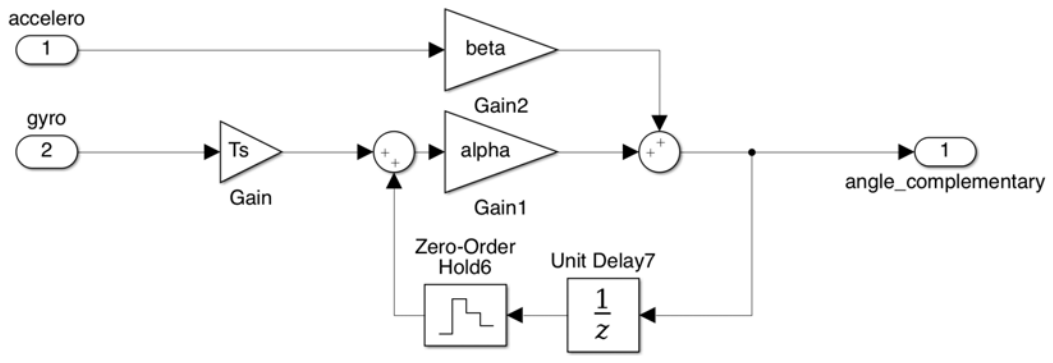


Figure 45: Complementary filter model: the simplest approach for IMU sensor fusion.

A validation of this model will be presented in chapter 8.

6.4.4 *Kalman filtering*

Even if the complementary-filter works fine in many applications, it has to be properly tuned in order to obtain a trade-off between responsiveness and noise removal. Many times, in order to reduce the noise in the measures it is necessary to deeply low-pass accelerometer's data, resulting in the introduction of heavy time-delays in measures, that sometimes might result unacceptable and might compromise device's performances.

An alternative to the simple complementary filter is the powerful and more-sophisticated *Kalman filter*.

The *Kalman filter* is a powerful algorithm used to identify (find a description that is the variance of) a disturbing signal acting on a system. It is a recursive algorithm based on the *innovation signal* of the process (which is the difference between the true and the estimated system output) to estimate the current state of a system. It is often used when the state of a system is not available, but necessary for the control purpose.
The correctness of the estimation must be verified: the residual noise of the *innovation signal* must be "white". For this scope it will be used a *Bartlett test* to verify this condition (explained in the following sections).
In our specific case, the process is the sensor's fusion model and the state is the "ideal" measure, without the additive noises.
The starting point is a discrete system in which acts noises, like the following:

$$\begin{aligned} x(k) &= Fx(k-1) + Gu(k-1) + m(k) \\ y(k) &= Hx(k) + w(k) \end{aligned} \tag{84}$$

The noises $m(k)$ acting on $x(k)$ can eventually be correlated. The important assumption that must be met, is that the noises $m(k)$ and $w(k)$ are *uncorrelated* (if they are correlated, they must be previously uncorrelated).
The correlation of the $m(k)$ signals is expressed by their *co-variance* (discrete) matrix obtained with:

$$Q_d = E[mm']$$

and the variance of the output error (for a scalar output) is:

$$R = E[ww'] = E[w^2]$$

The Kalman filter then executes a recursive prediction of the successive state, based on a *Minimum variance prediction* in the form:

$$\hat{x}(k) = F\hat{x}(k-1) + Gu(k-1) + K[y(k-1) - H\hat{x}(k-1)]$$

where:

$$K = PH^T(HPH^T + R)^{-1}$$

and P is the solution of Algebric Riccati Equation (ARE):

$$P = FPF^T + Q_{1d} - FPH^T(R + HPH^T)^{-1}HPF^T$$

Where the term $e(k-1) = [y(k-1) - H\hat{x}(k-1)]$ is the *Innovation* signal that will be investigated if "white".
The term Q_{1d} is the "true process co-variance", which describes the uncertainty of the noise acting on the process. It might be different form Q_d, because this one is the "nominal" value.

6.4.5 *Kalman tuning*

The process for discovering the correct Q_{1d}, called *the Kalman tuning*, consists in:

1. Calculate all the F, G, H, Q_d, R matrices and initializing:

   $P = 0$

   $Q_{1d} = Q_d * \kappa$

   where, initially it is posed:

   $\kappa = 1$

2. Run the system and evaluate the innovation $e(k) = [y(k) - H\hat{h}(k)]$ "whiteness". If it is "sufficiently white", the Q_{1d} is correct. If not:

3. Adjust the value of κ until the innovation results "sufficiently-white"[1] .

6.4.6 *The Bartlett's test*

This test gives a fast and intuitive way to verify the "whiteness" of a signal. Having the *FFT* of the signal, the values found are progressively integrated and plotted into a graph where the horizontal axis reports the number of FFT samples, while the vertical axis reports the value of the integration (whose absolute value is not important). What is important is to observe is the "straightness" of the line: if the signal is white, the FFT signal is almost "flat", thus the integration results in a straight ascending line. If the signal is not white, the FFT is not flat, and the integration results in a curved line that might be either over or below the graph's diagonal (see figure 46).

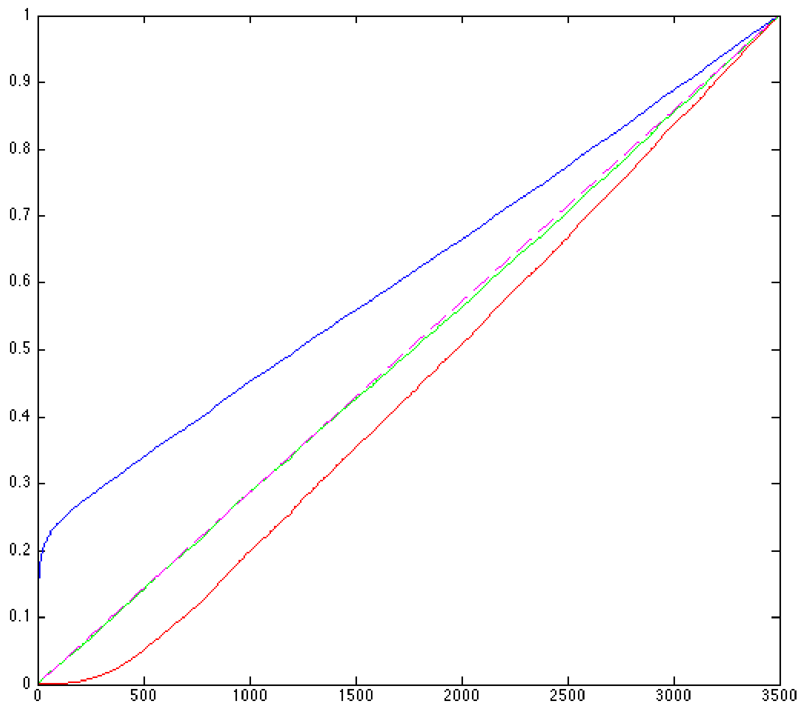


Figure 46: The *Bartlett whiteness test:* If a signal is white the resulting plotting is similar to the green line (almost coincident with te dashed diagonal), the red line indicates a "high-frequency-dominated" signal. The blue a "low-frequency-dominated".

1 note that Q_{1d} can even be arbitrary build (without relating it to its nominal value), but relating it to Q_d makes easier the tuning process

6.4.7 *Application of Kalman Filter to the system*

In order to use the Kalman filter in the IMU reading process, a proper model of gyroscope, accelerometer and their combination must be done.

6.4.7.1 *Gyroscope model*

Following the model proposed by [25], which corresponds to the single-dimension, discrete form of (80), the gyroscope output (considering only the "pitch" angle rate output coordinate) can be modeled as:

$$s_g(t) \quad = \quad \omega(t) + b_g(t) + n_g(t) = \dot{\theta}(t) + b_g(t) + n_g(t) \tag{85}$$

where:

- s_g is the gyroscope output
- ω is the angular velocity
- b_g is the gyro measure bias, supposed slightly variable
- n_g is the Gaussian noise added to the gyroscope measure, whose variance is unknown

The angle reading, starting from the gyro output, can be achieved by integration of the angle-rate:

$$\theta(t) \quad = \quad \theta(t_0) + \int_{t_0}^{t} \dot{\theta}(t) dt \tag{86}$$

Now, substituting (85) in (86) it results:

$$\theta(t) \quad = \quad \theta(t_0) + \tfrac{1}{s}\left(s_g(t) - b_g(t) - n_g(t)\right) \tag{87}$$

The bias is supposed to be affected by a process (white) noise n_b, thus its evolution model is:

$$\dot{b}_g(t) \quad = \quad b_g(t) + n_b(t) \tag{88}$$

6.4.7.2 *Accelerometer model*

Even for the accelerometer, a discrete-time model is build:

$$s_a(t) \quad = \quad \theta(t) + n_a(t) \tag{89}$$

where:

- s_a is the accelerometer output, already conditioned by subtracting bias and coupled acceleration, as described in section 6.4.1.1
- θ is the pitch angle
- n_a is the Gaussian noise added to the accelerometer measure, whose variance is unknown

In this case, the angle reading model for the accelerometer will be:

$$\theta(t) \quad = \quad s_a(t) - n_a(t) \tag{90}$$

Note that the signal $\theta(k)$ is not simply the accelerometer's output, but the result of the angle calculus explained in the previous section.

6.4.7.3 *Sensors combination*

The previous models of the sensors can be conveniently used to build a state-space system, affected by noise. Using the (87), we define the state of the system as the vector:

$$\dot{x}(t) = \begin{bmatrix} \dot{\theta}(t) \\ \dot{b}_g(t) \end{bmatrix} \quad \text{and} \quad u(t) = s_g(t) \tag{91}$$

The output definition is derived directly by the (90), thus the complete system, in the form (84) is obtained by substitution of all the previous, yielding to:

$$\begin{aligned} \dot{x}(t) = \begin{bmatrix} \dot{\theta}(t) \\ \dot{b}_g(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \theta(t) \\ b_g(t) \end{Bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} [s_g(t) + n_g(k-1)] + \begin{bmatrix} 0 \\ 1 \end{bmatrix} n_b \\ y(t) \quad = \quad \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + n_a(t) \end{aligned} \tag{92}$$

Where, as obvious:

$$\begin{gathered} x(t) = \begin{Bmatrix} \theta(t) \\ b_g(t) \end{Bmatrix} \quad u(t) = s_g(t) \quad A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad B_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad m_1(t) = n_g(t) \quad m_2(t) = n_b(t) \quad w(t) = n_a(t) \end{gathered} \tag{93}$$

Since the controller is discrete-time, the just-presented model must be discretized and it is necessary to find a model for the co-variance of the noise processes (to be applied to the kalman filter). Here the results:

$$\begin{aligned} F &= e^{AT_s} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \\ G &= \int_0^{T_s} e^{A\tau} B d\tau = \begin{bmatrix} T_s \\ 0 \end{bmatrix} \\ Q &= \sigma^2_{n_g} w^1_{0\to T_s} + \sigma^2_{n_b} w^2_{0\to T_s} \\ &\text{where:} \\ w^1_{0\to T_s} &= \int_0^{T_s} e^{A\tau} B_1 B_1^T (e^{A\tau})^T d\tau = \begin{bmatrix} T_s & 0 \\ 0 & 0 \end{bmatrix} \\ w^2_{0\to T_s} &= \int_0^{T_s} e^{A\tau} B_2 B_2^T (e^{A\tau})^T d\tau = \begin{bmatrix} T_s^3/3 & T_s^2/2 \\ T_s^2/2 & T_s \end{bmatrix} \\ R &= \sigma^2_{n_a} \end{aligned} \tag{94}$$

Note that the variance related to the accelerometer angle ($\sigma^2_{n_a}$) is not simply the sensor's noise variance, but since the angle coming from the accelerometer is a ratio between the two signals (see (82)), the corresponding variance is the product of the variances of the acc_x signal noise and the inverse variance of the acc_z signal noise (which is obtained by first-order Taylor approximation of the process).

The model of Kalman filter is already part of the Simulink environment. In figure 47, it is presented the blocks that makes possible to choose which sensor-fusion strategy to adopt.

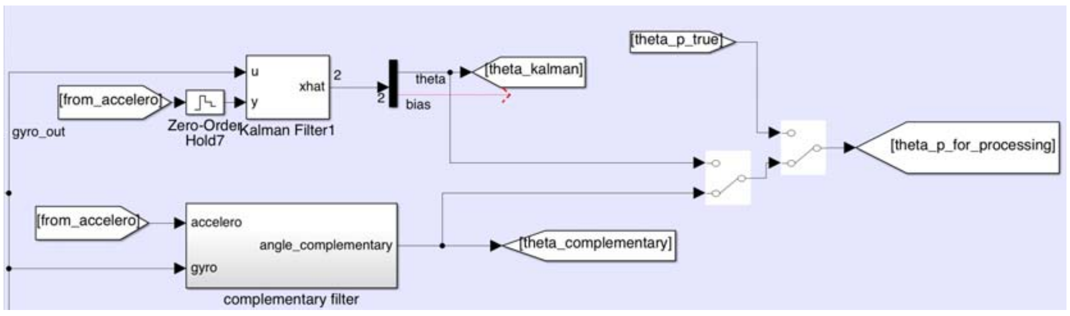


Figure 47: Possible solutions for the sensor-fusion algorithm to use, during simulations.

Unfortunately, for a matter of time, the validation of the kalman filter for the pitch angle calculation was not performed, thus the task is a point of interest for future works.

6.5 POSITION AND VELOCITY READING

In order to accomplish the controls designed in the chapter 5, it is necessary to have the description about the position and velocity of the base. An important assumption of slip-absence made in chapter 3 guarantees that the base position and velocity might be (linearly transforming) directly calculated by knowing the wheel's position and velocity.

The problem so, is the knowledge of the mentioned quantities for each wheel. Many times, if a previous good design was done, the problem reduces to the knowledge of motor's rotor position and velocity.

The rotor position and speed reading quality depends on several design issues such as:

- the presence or absence of sensors and their type
- the link between motor and wheel (elasticity, backlash)
- the measurements post-processing

The mentioned issues also drove the motor choice, in fact for this kind of application, not only the power and torque constraints are important, but also the motor chosen must meet the previous items.

6.5.0.4 *Sensorless vs sensored*

In the last years, the motor drivers have grown a lot their performances, making it possible to have amazing features even with cheap motors, even without any sensors (one of the first sensor-less applications was [14]). The limit of most sensor-less drivers is the low-speed operation, in fact the "traditional" sensor-less operation is based on the motor's Back-EMF reading, which is not easily measurable at low or steady speed.

Even for this last limit, some driving solutions like [30] and [11] have demonstrated the possibility of precise positioning of sensorless motors, even at low-speed, which is exactly the most important feature to achieve for correct balancing of the vehicle.

Unfortunately those techniques are relatively new and not available in many commercial or hobby products, infact their implementation sometimes needs the use of expensive components such as FPGAs and high-speed ADCs.

Giving that, for this project, a "more traditional" (sensor-ful) motor solution was chosen. In this context, it was necessary to determine the minimum motor angular resolution needed for the quasi-static, self-balancing task. It is obvious infact, that a poor resolution might result in an instable control, with seamlessly limit-cycles or even with control drift. On the other hand, the higher resolutions comes with a cost: a high-precision encoder is often very expensive. A trade-off between performance and cost must be found.

A good cost-reduction can be the direct use of hall-sensors, incorporated on many low-cost brushless motors, instead of the need for an expensive encoder. The big question is if the resolution is sufficient for the desired task. Unfortunately no literature was found about that matter, so the only thing possible was the study of this porblem or a try-and-error with models.

For a matter of time, the second choice was adopted, with the only starting-point constraint of looking for a combination of motor and wheel, with a minimum guaranteed resolution of $5°/\mathtt{hall}-\mathtt{step}$.

6.5.0.5 *Motor-wheel link*

The last important consideration about the wheel position reading is the quality of the movement transmission between the motor and the wheel. There exists a vast panorama of movement transmissions: The most common are:

- belts-pulleys
- gearboxes
- direct-drive motors

The belt-based solutions often has :

- Pro: low-weigth
- Pro: zero-backlash
- Con: considerable elasticity of the link.

Gearboxes often has exactly the opposite pros & cons of the belts.

Direct drive motors has:

- Pro: good positioning features and no elasticity or backlash
- Pro: are very silent
- Con: frequently is a big-sized motor.

The choice of the motor was already explained in chapter 2, and the final result was the choice of a hall-sensor-ed direct-drive hub-motor. It has many advantages, especially for the fact that the wheel's position coincides with the motor position. The other advantage is the possibility to ignore other non-idealities such as backlash or link elasticity, particularly critical for a self-balancing transporter.

Thanks to this choice of a direct-drive motor, from now on, the effect of transmission non-idealities will be neglected.

6.5.0.6 *Motor model and velocity reading*

Giving all the previous considerations, a convenient model for the motor position reading was provided: it simply considerate the position output as a quantized measure, as visible in the figure 48.

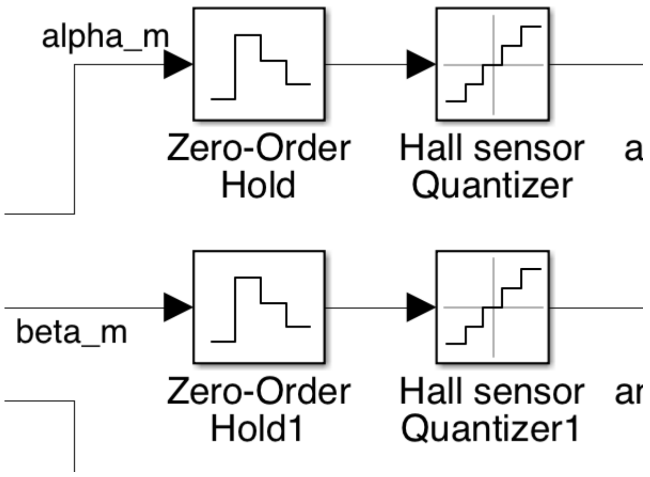


Figure 48: Model for the motor position reading: a quantized angle, in combination with a discrete-time reading.

The quantizers in the figure emulates the effect of the motor's hall sensors. The quantization step, here named *hall-step* is the number of degrees between two consecutive hall commutations. The total number of commutations in a motor revolution is:

$$n_{steps} = n_{poles} * n_{hall-sensors} \tag{95}$$

Thus the *hall-step* is:

$$h_S = \frac{2\pi}{n_{steps}} = \frac{2\pi}{n_{poles} * n_{hall-sensors}} \ [°/hall - step] \tag{96}$$

Where (as obvious), $n_{hall-sensors}$ in a motor is (almost always) 3 and n_{poles} is the number of magnetic poles of the motor. This is the value introduced in the "quantization step" of the blocks of figure 48.
By considering the presence of a motion reduction, with rate of ρ, the resolution in the reading of the wheel's angle is:

$$q_{wheel} = \rho * \frac{2\pi}{n_{steps}} \ [°/hall - step] \tag{97}$$

As explained in this chapter, the position reading is affected by an uncertainty, directly related to the quantization process. With the simulations and the practical experiments, this had demonstrated to be not a big problem, since (with a proper *hall step*), it won't determine any performance decrease.

Even if it doesn't compromise stability, after some simulations and experiments it was noted that the granularity in the position reading makes the velocity (and the acceleration) readings very noisy. That's why a second-order LPF (acting at 5 Hz) was introduced in the position reading path. As noticeable in figure 49, the "granularity" was deeply reduced, at the cost of a delay in the measure (lower graphs).

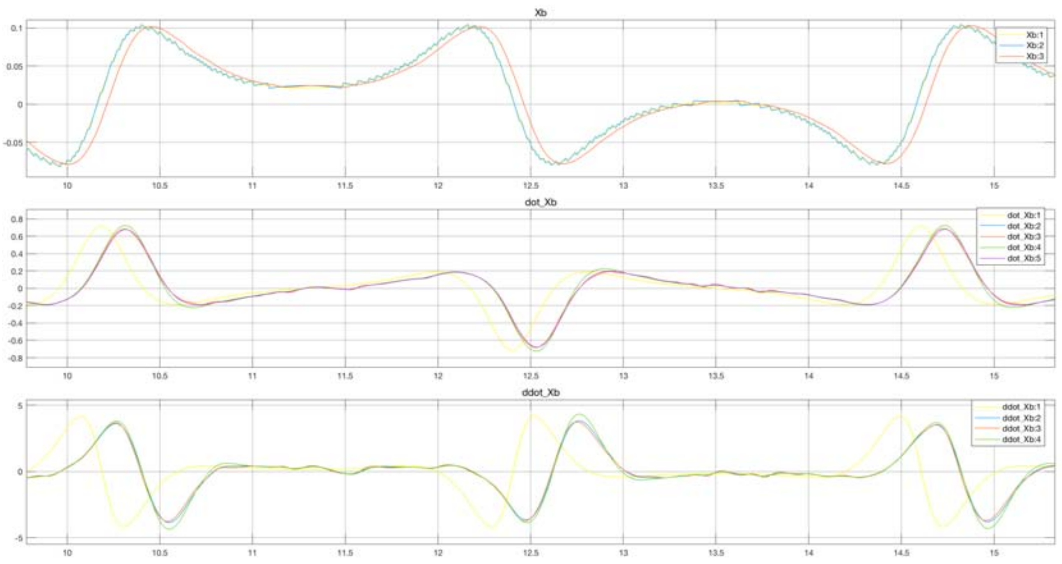


Figure 49: Position reading (higher) graph: after the introduction of a second-order, 5 Hz LPF, the "steps" (in the blue track) have been smoothed (red track), at the cost of a measure delay. This had a great benefit in velocity and acceleration reading.

At least, the *ZOH* block simulates the discrete-time reading of the angle. In fact the angle-reading happens at the rate of the control-loop.

The motor angle position is directly provided by the driver board, which translates the hall steps into an increasing/decreasing 32-bit counter, thus there was no need for translating the angle from a $0-360°$ interval, to a continuous one. The 32-bit counter also will seamlessly never saturate since one wheel represents (for the motor chosen) 90 counter steps. One wheel rotation corresponds to:

$$d_{rot} = 2\pi r_{wheel} = 2\pi 0.1 = 0.628 \text{ m} \tag{98}$$

Thus the counter (positive or negative) saturation will happen at:

$$d_{sat} = \frac{2^{31}}{90} d_{rot} = 14984.6 \text{ km} \tag{99}$$

whick is far more a the battery autonomy trip, thus it is reasonable to reset the counter at every battery charge and don't matter about counter saturation.

The model of position-reading, comprising the select-able LPF is visible in figure 50.

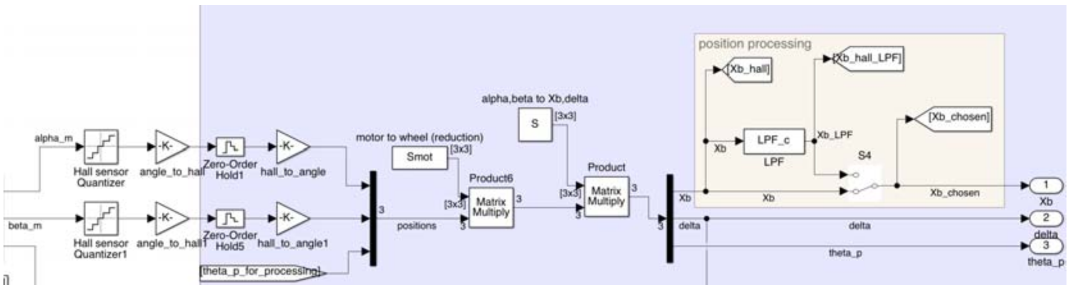


Figure 50: Position reading complete chain: hall quantization in the left, in the center the reduction and the coordinate transform. In the right, the selectable second-order LPF.

6.5.0.7 *Velocity reading*

In the system there is not a specific velocity sensor, so that the vehicle velocity must be derived from the (time-derivative of) position. This operation is risky in systems where a small superimposed noise might be amplified by the time-derivative. Even in the absence of noise, the quantization steps representing the hall process, if directly derived, will result in velocity-spikes that affects velocity as undesired noise.

Another fact is that the pure-time-derivative of a quantity is not possible in a real system, since it will result in a not-proper one. The challenge is then to find a convenient solution for the (reliable) velocity reading.

The problem of reading velocity from quantized-position is very common in motion control, thus literature provides a wide panorama of solutions such as:

- High-pass filter
- Average-derivative block
- PLL
- Kalman filter

A brief description of all the previous solutions, and their corresponding model, will be provided. The validation of such models will be discussed in chapter 8.

6.5.0.8 *High-pass filter*

In place of the derivation block, it might be adopted a (proper) *High-pass Filter*, as visible in figure 51.

This filter has the advantage of being exactly equivalent to a derivative block, but with the addition of a hgh-frequency pole, that makes it a proper system.

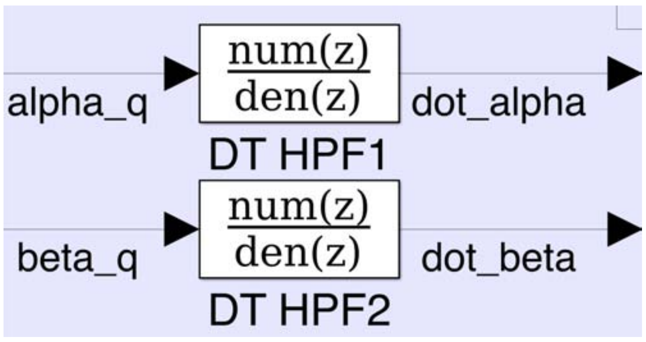


Figure 51: Wheel velocity retrieving: a high-pass filter is equivalent to a time derivative, plus a high-frequency pole, that makes it a proper system.

The Laplace form of the HPF is:

$$W_{\mathrm{HPF}} = \frac{s}{1+s/\omega_{\mathrm{HPF}}} \tag{100}$$

The previous filter was time-discretized at the rate of the controller (T_s) and the resulting frequency response is that on figure 52.

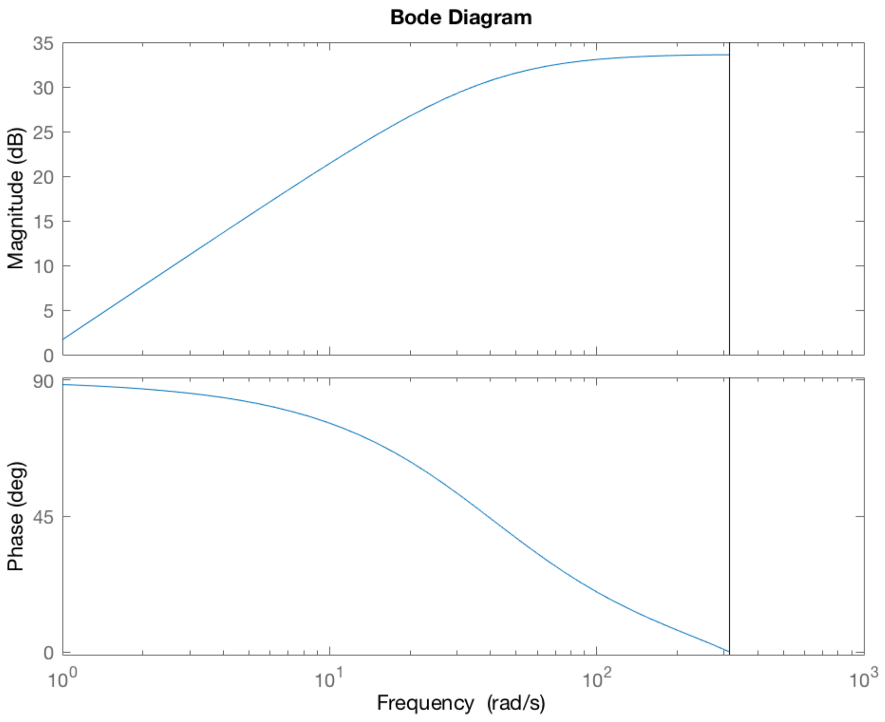


Figure 52: First-order, discrete high-pass filter, with a pole at $40 rad/s$.

This filter gave a first acceptable result for the velocity reading, but demonstrated a consistent high-frequency residual noise, discussed in chapter 8.

6.5.0.9 *Average-derivative block*

This technique, is essentially an interpolation of multiple time-derivatives of the position's readings, performed along successive sampling steps, to remove noise and even to limit "spikes".

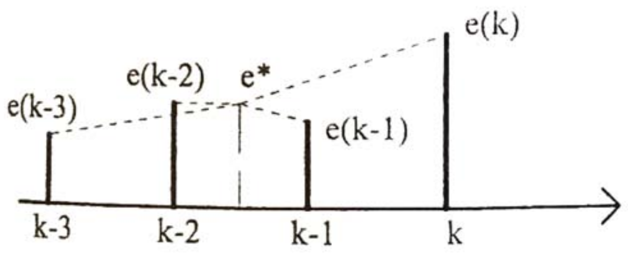


Figure 53: Average-derivative for velocity calculation.

Referring to figure 53, the value of e^* is the average of the input signal $e(k)$ and it is ideally placed in the middle of the time period.

The *average derivative* is defined as:

$$\frac{de(kT)}{dt} \approx \frac{1}{4}\left[\frac{e(k)-e^*}{1.5T} + \frac{e(k-1)-e^*}{0.5T} - \frac{e(k-2)-e^*}{0.5T} - \frac{e(k-3)-e^*}{1.5T}\right] \tag{101}$$

where:

$$e^* = \frac{1}{4}\left[e(k) + e(k-1) + e(k-2) + e(k-3)\right] \tag{102}$$

This corresponds to the following:

$$\frac{de(kT)}{dt} \approx \frac{1}{6T}\left[e(k) + 3e(k-1) + 3e(k-2) + e(k-3)\right] \qquad (103)$$

In case of an average of more than four values, the previous formula must be simply adapted.

The advantage of this technique is that it is not recursive and it is very computationally-light. The disadvantage is that the derivative value comes with a delay of $n/2$ steps, where n is the number of samples and the output might result "noisy" if compared with other techniques. A model of this filter was built with a 10-tap block, as shown in figure 54.

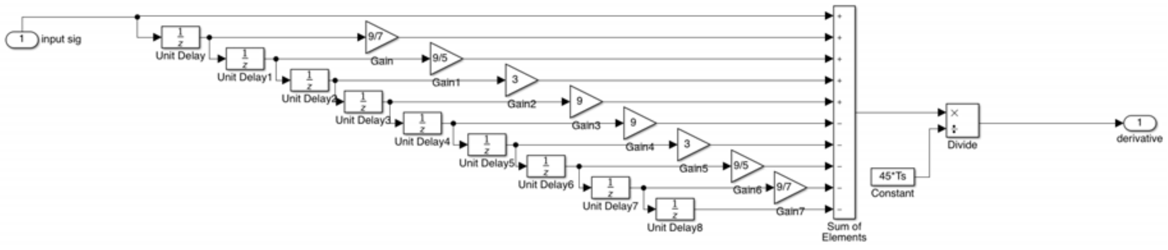


Figure 54: Simulink model of a 10-tap, average derivative block, suitable for velocity estimation.

6.5.0.10 *PLL*

PLL is another technique frequently used for the velocity derivation. Its scheme is visible in figure 55.

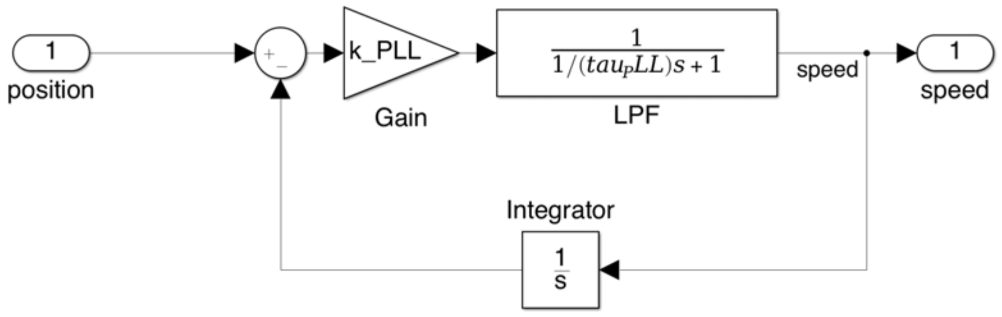


Figure 55: PLL scheme, used for the derivation of velocity, from position. For simplicity the controller is not a PI, but a simple proportional one.

In simple words the mentioned circuit is a "reference follower", that outputs the velocity derived from the position. In this scheme, the reference follower is a simple proportional controller. In a "true" PLL, an integral contribute to that controller is always present, to null the "regime error". In this section, the integral contribute was omitted in order to simplify the following considerations.

This powerful scheme gives good results in the speed reading, but it is not simple to properly "tune" the k_{PLL} and τ_{PLL} parameters. An improper tuning might in fact compromise the good reading of the velocity, together with a possible instability of the PLL circuit. Improper settings might result in a weak follower or -as opposite- in an under-damped one, with noise emphasis.

A proposed method to properly *tune* the PLL and avoid instability is to see the PLL as a *LTI* system and to derive its *closed-loop* transfer function:

$$W_{PLL} = \frac{k_{PLL}\frac{1}{1+s\tau_{PLL}}}{1+k_{PLL}\frac{1}{s}\frac{1}{1+s\tau_{PLL}}} = \frac{s}{s^2\tau_{PLL}/k_{PLL}+s/k_{PLL}+1} \tag{104}$$

Which is essentially a pure derivative, with the addition of two poles. In order to tune properly the PLL, it is proposed to choose a τ_{PLL} and then calculate the k_{PLL} gain in order to make the two poles complex-conjugate and with a dumping factor imposed by Butterworth's criteria. In this way it is guaranteed the absence of over-elongation in the output.

In the system model, an LTI (filter) block (named *VHPF+pole*) with a transfer function of (104) and a frequency response of figure 56 was made, in order to verify the coincidence of this block, with a PLL. The results of this comparative will be discussed in chapter 8.

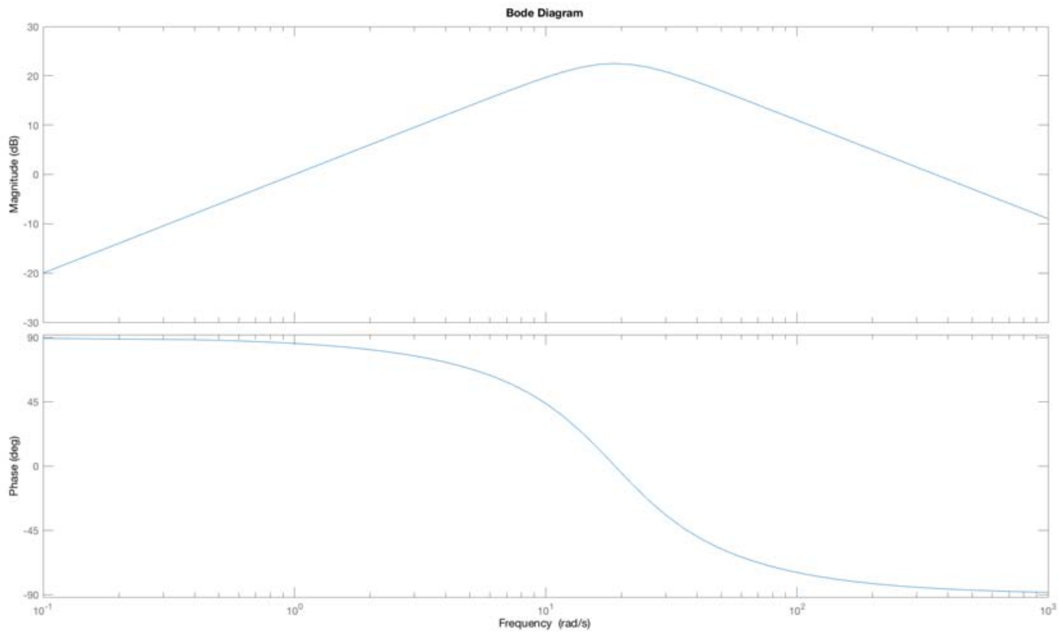


Figure 56: Bode plot of the transfer function (104). This emulates the PLL circuit with a Butterworth-tuned complex-conjugated poles.

6.5.0.11 *Kalman filter*

The kalman filter and its tuning were already discussed in section6.4.4. In this section it will be presented how the position and velocity processes were modeled, for the application of the Kalman filter.

A complete-order description of the motion of the base might be expressed as the following state-space system:

$$\begin{Bmatrix} X_b \\ v_b \\ a_b \end{Bmatrix}_{k+1} = \begin{bmatrix} 1 & T_s & T_s^2/2 \\ 0 & 1 & T_s \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} X_b \\ v_b \\ a_b \end{Bmatrix}_k + \begin{bmatrix} T_s^3/6 \\ T_s^2/2 \\ T_s \end{bmatrix} w_k \tag{105}$$

Where w_k is the noise applied to the acceleration process. This system is a third-order one, which means that it might be hard to manage, for Kalman routines running in a micro-controller. In order to make it simpler to manage, the system was reduced in dimension, thanks to the assumption that the contribution of a_b

is small, especially if considering a constant acceleration during the sample-time. Given this, the acceleration model was incorporated to the model of its noise $a_k \sim \mathcal{N}(0, \sigma_a^2)$.

Thanks to this, a simplified model was built, inspired to that of [36], where the state vector becomes:

$$\mathbf{x} = \begin{Bmatrix} X_b \\ v_b \end{Bmatrix}$$

and the system evolution description is:

$$\begin{aligned} x_{k+1} &= x_k + \dot{x}_k T_s + \frac{(T_s)^2}{2} a_k \\ \dot{x}_{k+1} &= \dot{x}_k + T_s a_k \end{aligned} \tag{106}$$

which corresponds to:

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} \frac{(T_s)^2}{2} \\ T_s \end{bmatrix} a_k \tag{107}$$

Obviously the output is the (noise affected) measure of the position: which corresponds to:

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{Bmatrix} X_b \\ v_b \end{Bmatrix} + v_k \tag{108}$$

where $v_k \sim \mathcal{N}(0, \sigma_h^2)$ is the noise related to the hall-steps, which was modeled as a normally-distributed noise with variance:

$$\sigma_{hall}^2 = \frac{q_{hall}^2}{12} = \frac{(\frac{2\pi}{n_h all_s teps})^2}{12}$$

The covariance matrix is defined as:

$$Q = \begin{bmatrix} \frac{(T_s)^4}{4} & \frac{(T_s)^3}{2} \\ \frac{(T_s)^3}{2} & T_s^2 \end{bmatrix} \sigma_a$$

This filter was only modeled and tuned in simulation, but not implemented in the real device, since its computational weight, compared with the reduced benefit against "lighter" filters made it not convenient.

6.5.0.12 *veleocity reading comparisons*

All the previous velocity-reading techniques were modeled with the corresponding simulink blocks and the model's outputs were select-able, in order to make different simulations and choose the best solution.

Souch a model is visible in figure 57.

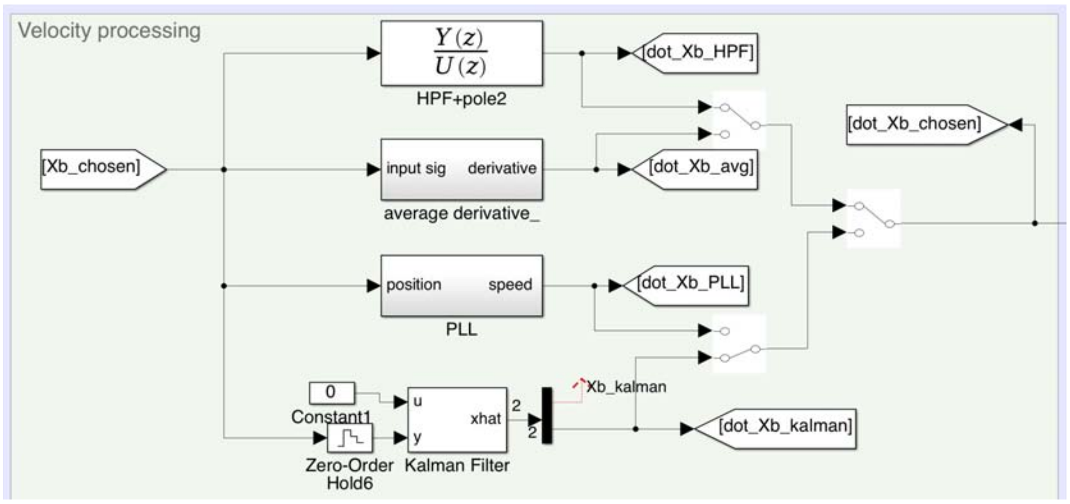


Figure 57: Different velocity-calculation blocks: the model's outputs were select-able, in order to make different simulations and choose the best solution.

The validation and the final choice of the velocity calculation method is discussed in 8.

6.6 CONCLUSIONS

In this chapter were illustrated the most important non-idealities that makes the difference between the theoretical model and the real one, how they were identified and put in the simulation model.

The good thing is that those non-idealities blocks are "additive" and affected only one of the blocks in the "block hierarchy", in fact the original block, representing the dynamic model was preserved (in a self-consistent Simulink block) and also the "high-level" block, representing the references and the controllers was not modified by these blocks.

Finally, in this chapter there were illustrated only the mathematical considerations and the related models because the validation of such models will be described in the following chapter.

7 ALGORITHMS IMPLEMENTATION

7.1 INTRODUCTION

Until now it was presented the system's components, the mathematical model and the system non-idealities models. In this chapter it will be presented how the models were traduced into code and how it "fits" into the system architecture and the practical issues that have been solved.

In particular, since the algorithm runs into an embedded microcontroller, a deep focus on the mechanisms lying into it and the motivation that drove some implementation choices.

7.2 OVERALL ARCHITECTURE AND BEHAVIOR

As introduced in chapter 2, the hardware architecture of the device consists of two identical motor-driver boards, where the one attached to the IMU acts as *MASTER* and the other as *SLAVE*, the -just mentioned- IMU board and a bluetooth module board. Differently from figure 11, which is a mere "schematic" representation, figure 58 will be used to represent the data flows, during the algorithm running.

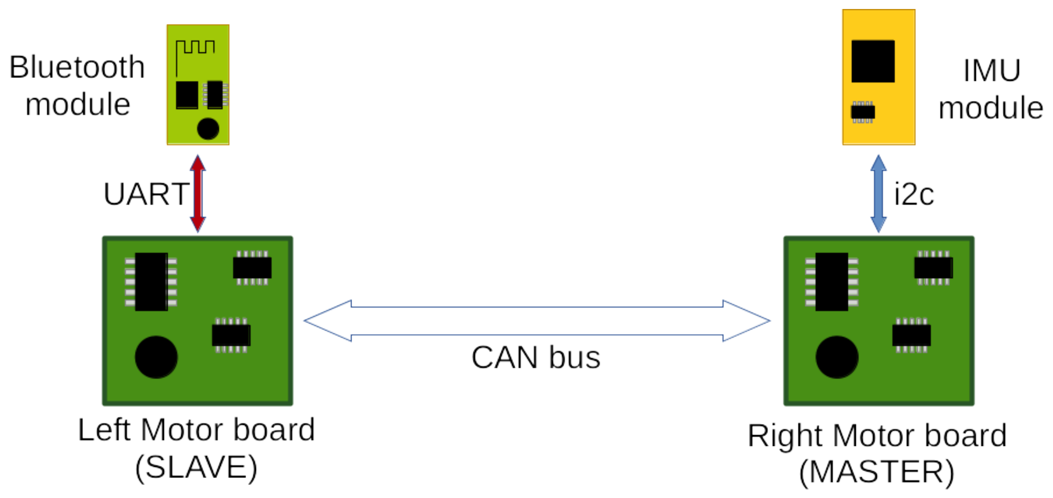


Figure 58: Logic architecture of the system

The two *Motor boards* are the components addicted to the generation of high-power signals that moves the motors, even making use of the motor's sensors. Even though, this is not the only work performed by those boards, since they are equipped with a powerful micro-controller with a real-time operating system (*RTOS*) that permits to perform several tasks in addition to the motor control. The original firmware that performs all this motor control and other tasks is well documented in [35].

The powerful platform and the operating system are a good starting point: this combination is the reason behind the idea of adding a further *TASK* into the *RTOS* which performs the control of both equilibrium and steering, without the need of an external board for this task.

The implementation and integration of the algorithm inside the micro-controller was successful, thanks also to the reliability of the high-speed CAN-bus communication channel that transfers data at a rate of ≈ 500 kbaud/s. Another advantage is the fact that part of the data for the algorithm is resident in the *MASTER* board and don't need to be remotely read.

The biggest problem in this implementation, is that the boards are identical, but must behave differently. Also there was the need to add some communication commands for data-exchange between the two boards (other commands in addition to those in the original firmware), and those commands must be symmetric between to the two boards.

The need was then to have:

- two hardware-identical boards
- a different behavior between the two boards
- a set of communication commands, shared between the boards

The solution adopted was to have a *unique firmware* in both the boards: in this way the communication protocol is automatically shared. The problem of the different behavior was solved by a non-volatile software parameter : the *CAN address*. In this way the *MASTER* board is that with *CAN address=0* and the slave has *CAN address=1*. This convention served also as definition of *LEFT=CAN address=1* and *RIGHT=CAN address=0*. At start-up, the micro-controller checks its address and consequently knows if it has to behave as *MASTER* or *SLAVE* inside the system.

Thanks to this solution also the firmware maintenance was very easy, with a unique firmware version and without the need to worry about cross-versions compatibility.

The *IMU* board is an i2c-SLAVE device, whose MASTER is the *MASTER* motor control board. The communication between the two is the fastest possible in the i2c standard 400kHz bit-clock. The micro-controller reads the IMU standard registers and doesn't use the special function (such as sensor-fusion or quaternion buil-up) provided by the manufacturer. In this way a possible replacement of the IMU model is possible, without the loss of specific algorithms.

The *bluetooth* board needed a minimum configuration since it is factory-set with a fixed UART-baudrate of 9600bps. Following the procedure described in [28], the baud-rate was changed with the fastest universally-supported bitrate (115200bps), that made it possible to exchange data with PC with low latencies and high data-throughput (particulary useful for data exchange between the self-balancing vehicle and the PC, during work). Note that the bluetooth module is simply an "interface bridge" between the serial ports of the PC and that of the micro-controller: there is infact no protocol overhead for the communication: both the PC and microcontrollers works like they were wire-connected by serial port.

7.3 THE MATLAB ENVIRONMENT

All the models described into chapters 3, 4 and 6 have already been implemented by block-modeling with Simulink software. What has not already been described is where the block-models picks the parameters.

In this section the description of mathematical implementations needed for the parameters calculus and how they are used before, during and after the simulation. In the MATLAB routines in fact it has been implemented the possibility of:

- calculation of simulation parameters and filters
- running the simulation with Simulink
- launching the experiment into the real-device (the same as the simulation)
- collecting the data from the real-device
- post-processing data and perform a comparison between simulation results and real-device behavior

Here is the code that performs the previous:

```
%     Two-wheels, inverse pendulum transporter model and controller
clc;
close all;
clear all;


% simulation parameters
s=tf('s');
f_PWM = 168e6/35000;            % [Hz] PWM frequency (see ST code)
Ts =  0.008;                     % [s] sampling time (loop executes at this
    frequency)
final_time=30;                  % [s] end time
% n_samples = t_simul/Ts;

run('vehicle_params.m');
run('overall_model.m');
run('equilibrium_model_and_controller.m');
run('steering_model_and_controller.m');
run('digital_filters');
run('header_file_control_params.m');  % write params into C header file

run('bluetooth_param_writer');  % write params via bluetooth radio

run('personal_transporter_modelo_discrete_PID.slx');
disp('ATTENTION: it is compulsory to run the simulation, in order to compare
    true and simulated data!')
disp('After simulation complete, come back here and press ENTER')
pause

run('state_readings_experim.m'); % start simulation and collect data
```

In the following subsections a brief description of the presented subroutines.

7.3.1 *Vehicle parameters*

This script sets all the parameters for the simulation. Very important are the motor and rider's parameters: they deeply influences the control behavior.

```
% % BLDC motor parameters (gearmotor)
PN = 500;                       % [W] Nominal power of each motor
iNom = 20;                        % [A] Approx nominal current .... See
    datasheet
n_poles = 30;                   % [/] number of magnetical poles on the motor
n_winds = 27;                   % [/] number of windings on the motor
r_gearbox = 1;                  % [/] Gearbox ratio
rho = 1/r_gearbox;
UN = 36;                        % [V] Nominal motor voltage
ke = 0.694;                     % [V*s/rad] this is the motor voltage constant
Rs = 0.128;                     % [Ohm] I don't know.... it has to be measured
     (i.e. with the 4 wires method)
Ls = 166.8e-6;                  % [H] motor ph-ph inductance
kt = ke;                        % [Nm/A] torque constant: number of Nm for
    every ampere of motor
n_hall = 3;                     % number of hall sensors

iMAX = 2*iNom; %it is assumed that the pulse current is approx. double the
    nominal current

% accelerometer params
n_bit_accelero=16;              % number of bits of accelerometer

m_P=30;                           %[kg] mass of the person
L=0.51;                            %[m] half the height of the person
m_b=5;                            %[kg]mass of the base
b_h=0.05;                         %[m]base heigth
b_l=0.2;                          %[m]base length
J_b=1/12*m_b*(b_h^2+b_l^2);       %[kg*m^2]momentum of inertia of the base
r=0.1;                            %[m] radius of the wheel
Jd=1/2*m_P*0.2^2;                 %[kg*m^2]momentum of inertia of the person,
    rotating around its vertical axis
D=0.5;%0.4;                       %[m] distance between wheels
m_w=2.5;                          %[kg] mass of a wheel
J_w=1/2*m_P*r^2;                  %[kg*m^2]momentum of inertia of the wheel
J_m=1/2*0.5*0.03^2;               %[kg*m^2]momentum of inertia of the motor
psi=5.7e-4;%1                       %[Nm/(rad/s)] viscous friction coefficient
     of the motor(derived with experimet)
g=9.81;                           %[m/s^2] gravity
JO=1/3*m_P*(2*L)^2;               %[kg*m^2]momentum of inertia of the person,
    rotating around wheel's axis
c_s=350;                          %[Nm/(rad/s)] viscous friction coefficient
    of physiological ankle model
k_s=1440;                         %[Nm/rad] spring coefficient of
    physiological ankle model
tacho_to_angle = (2*pi)/(n_poles*n_hall);
```

7.3.2 *overall model*

This file describes all the model-buildup as described in chapter 3, the coordinate-change matrix and the decoupling matrix:

```
%     Overall model: the "M","C","K" matrixes and the coordinate-change and
%     decoupling matrix
m11 = m_P*r^2/4+r^2*Jd/(D^2)+m_w*r^2+J_w+J_m/rho^2;
m12 = m_P*r^2/4-r^2*Jd/(D^2)+m_w*r^2+J_w;
m13 = m_P*L*r/2-J_m/rho^2;
m33 = m_P*L^2+2*J_m/rho^2+J0;
c11 = psi/rho^2;
k33 = -m_P*g*L;
%syms m11 m12 m13 m33 c11 k33

M = [m11 m12 m13;...
     m12 m11 m13;...
     m13 m13 m33];
C = [c11    0       -c11;...
     0      c11     -c11;...
     -c11   -c11    2*c11];
K = [0    0    0;...
     0    0    0;...
     0    0    k33];
 A = [ zeros(3) eye(3);...
      -inv(M)*K -inv(M)*C];
 U =[1/rho*eye(2);-1/rho -1/rho];
 B = [ zeros(3);inv(M)]*U;
% B_d = B*D;

% coordinates change, in order to decouple the system.
%  xb = r*(alpha + beta)/2
%  delta = r*(alpha-beta)/D
%  xb       r/2  r/2  0     alpha
%  delta = r/D  -r/D 0     beta
%  theta_P  0    0    1     theta_P
%syms r D

%matrix of base-change: S
S = [ r/2  r/2  0;...
      r/D  -r/D 0;...
      0     0   1];

% other coordinates change, to find the motor angle
%  alpha = theta_P+rho*alpha_mot
%  beta = theta_P+rho*beta_mot
%  theta_P = theta_P
Smot=[ rho  0   1;...
       0   rho  1;...
       0    0   1];

% now rewrite the whole system in the coordinates [xb delta theta_P]’
M_s = M*inv(S);
C_s = C*inv(S);
K_s = K*inv(S);

A_s = [ zeros(3) eye(3);...
     -inv(M_s)*K_s -inv(M_s)*C_s];
B_s = [ zeros(3);inv(M_s)]*U;

%decoupling matrix, built assuming:
```

```
%   tau_theta = 1  1   tau_L
%   tau_delta   1 -1   tau_R
Dec = inv([1 1;1 -1]);

%decoupled B_s matrix
B_sd = B_s*Dec;

%re-arrange the whole system in a more-convenient view
% the system's state vector were: [xb delta theta_P dot_xb dot_delta
    dot_theta_P]'
% now they becomes:[xb theta_P dot_xb dot_theta_P delta dot_delta]'
N=[ 1 0 0 0 0 0;...
    0 0 1 0 0 0;...
    0 0 0 1 0 0;...
    0 0 0 0 0 1;...
    0 1 0 0 0 0;...
    0 0 0 0 1 0];
A_sN=N*A_s*inv(N);
B_sdN=N*B_sd;
```

7.3.3 *Equilibrium model and controller*

The script about the equilibrium control, derives the equilibrium subsystem derived from the "overall model" and calculates the control parameters, used in the Simulink model.

In this script it is also defined the set of points used in the reference generator:

```
%% Balancing subsystem ("equilibrium")
A_e=A_sN(1:4,1:4);
B_e=B_sdN(1:4,1:1);

% % test of controllability with the only "theta_P" variables
C_e=[   0 1 0 0  ;...
        0 0 1 0 ];
D_e=[   0  ;...
        0  ];
states = {'Xb' 'theta_p' 'vb' 'theta_dot_p'};
inputs = {'tau_theta' };
outputs = {'theta_p' 'vb'};
sys_ssb = ss(A_e,B_e,C_e,D_e,'statename',states,'inputname',inputs,'outputname
    ',outputs);

% Check poles of the state-space (evaluate stability of the system)
poles_equilibrium = eig(A_e);
[V,P]= eig(A_e);


CTB=ctrb(sys_ssb);

if(rank(CTB) == length(A_e))
    fprintf('Ok MAC.... The system is controllable.\n')
else
    fprintf('Gosh MAC!... The system is NOT controllable!\n')
end

% observability of the system
```

```
obsv_rank = rank(obsv(A_e,C_e));

%% LQR parameters for the continuos time model
Q=diag([5;15;2;0.1]); %used for simulations
R=0.1;
%R=0.001;
K = lqr(A_e,B_e,Q,R);

% Controllable parameters
Ac = (A_e-B_e*K);
Bc = B_e;
Cc = C_e;
Dc = D_e;
sys_cl = ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',
    outputs);

%%  Model discretization (for the equilibrium subsystem)
sysPd = c2d(sys_ssb, Ts);        %   discrete-time model (LTI obj)
[Ad,Bd,Cd,Dd] = ssdata(sysPd);  %   discrete-time model (state-space matrices)

% DLQR parameters for the model: discrete-time variant of LQR
Kd = dlqr(Ad,Bd,Q,R)

% Controllable parameters
Acd = (Ad-Bd*K);
Bcd = Bd;
Ccd = Cd;
Dcd = Dd;

poles_rd = eig(Acd);

% reference vector, for the linear velocity
references_dot_Xb=[ 0, 0;...
                    1, 0.5;...
                    2, 0.5;...
                    4, -0.5;...
                    6, 0;...
                    final_time, 0];
```

7.3.4 *Steering model and controller*

Similar script, as that for the equilibrium: here the routines calculates parameters for the two proposed controllers: LQR and PI:

```
%% steering subsystem
states_steering = {'delta' 'dot_delta'};
A_t=A_sN(5:6,5:6);
B_t=B_sdN(5:6,2:2);
C_t=[0 1];
D_t=0;

sys_steer = ss(A_t,B_t,C_t,D_t,'statename',states_steering,'inputname','
    tau_delta','outputname','dot_delta');
sys_steer_d = c2d(sys_steer,Ts);
[A_td,B_td,C_td,D_td] = ssdata(sys_steer_d);
```

```
poles_steering=eig(A_t);
[tf_t_num tf_t_den] =ss2tf(A_t,B_t,C_t,D_t);

% design LQR for steering subsystem
%           delta;dot_delta
Q_steer=diag([1;0.01]);
R_steer=0.1;

K_steer_c = lqr(A_t,B_t,Q_steer,R_steer)
K_steer = dlqr(A_td,B_td,Q_steer,R_steer)

% lqr-controlled system
A_tc = (A_t-B_t*K_steer_c);
B_tc = B_t;
C_tc = C_t;
D_tc = D_t;
sys_steer_c = ss(A_tc,B_tc,C_tc,D_tc);
% Now evaluate the stability of the controlled system
poles_steer_controlled = eig(A_tc);

figure()
subplot(2,1,1);
pzplot(sys_steer);
title(’Poles and zeros of steering angle transfer function’)
subplot(2,1,2);
pzplot(sys_steer_c);
title(’Poles and zeros of the LQR controlled system’)

%% design of PI controller for the steering subsystem
% zeros_steering = roots(tf_d_num(1,:));
tf_t=tf(tf_t_num,tf_t_den);

% reduce the degree of the polynomials
if((tf_t_num(end)==0)&& (tf_t_den(end)==0))
   tf_t= tf(tf_t_num(1:length(tf_t_num)-1),tf_t_den(1:length(tf_t_den)-1));
end

KP_t=2% nearly unity gain, since the process is almost sufficiently fast
KI_t=omega_open_loop/10*KP_t
% then the controller becomes:
Wpi_t=KI_t/s+KP_t;
% tf_pi_t=tf(Wpi_t)
% discretization:
Wpi_td = c2d(Wpi_t, Ts);
[numPI, denPI] = tfdata(Wpi_td, ’v’);

%closed-loop controller and system: verification
Wloop_t=(tf_t*Wpi_t);
Wcl_t=(tf_t*Wpi_t)/(1+(tf_t*Wpi_t));

 figure()
 subplot(3,1,1);
 bode(tf_t);
 title(’Frequency plot of transfer function of steering subsystem’)
 subplot(3,1,2);
 margin(Wloop_t);
 title(’Frequency plot of loop–gain in the PI–controlled steering subsysyem’)
 subplot(3,1,3);
 margin(Wcl_t);
 title('PI-controlled steering subsysyem frequency response')

%% yaw-rate references
references_dot_delta=[  0 ,0;...
                        10, 0;...
                        12, 3;...
                        14, -3;...
                        16, 0;...
                        final_time, 0];
```

7.3.5 *Digital filters*

This script collects all the parameters concerning the digital filters. The generic name *digital filters* incorporates all the parameters for various function, as clearly noticeable directly in the code. Those parameters are useful for both the simulation environment and the real device (infact they are sent via bluetooth, as described in the following sections):

```
%% DISCRETE_TIME FILTERS

%% Define the DT high-pass filters (for velocity estimation)
%   continuous-time filter definition
% omega_HPF = 20;    % [rad/s] the pole is placed at that frequency
% omega_HFpole = 1000;    % [rad/s] another high-frequency pole
% numFc = s;
% denFc = (1+s/omega_HPF)*(1+s/(omega_HFpole));
% sysFc = numFc/denFc;%tf(numFc, denFc);       % s/(s/omega_HPF+1)
f_HPF = 2.5;
omega_HPF = 2*pi*f_HPF;    % [rad/s] the pole is placed at that frequency
psi_HPF=0.707;
numFc = s;
denFc = (s^2/omega_HPF^2+2*psi_HPF*s/omega_HPF+1);
sysFc = numFc/denFc;%tf(numFc, denFc);       % s/(s/omega_HPF+1)

%   filter discretization
sysHPF = c2d(sysFc, Ts);
[numF, denF] = tfdata(sysHPF, 'v');

%% discrete-time integrator
sysInt=tf([0 1],[1 0]); % = 1/s
sysIntD = c2d(sysInt, Ts);
[numInt, denInt] = tfdata(sysIntD, 'v');

%%   second-order low-pass filter for accelerometer
f_LPF=10; %[Hz]
omega_LPF=2*pi*f_LPF;
psi_LPF=0.707;
LPF_c=tf(1/(s^2/omega_LPF^2+2*psi_LPF*s/omega_LPF+1));
LPF_d = c2d(LPF_c, Ts);
[numLPF, denLPF] = tfdata(LPF_d, 'v');

%% PLL (alternative way for derivation): try to design it considering "omega
% and psi"
% f_PLL = 5;
% omega_PLL = 2*pi*f_PLL;
% k_PLL=10;                  %OPTIMUM VALUES
% psi_PLL=0.707;
% tau_PLL=omega_PLL

f_PLL = 3;
omega_PLL = 2*pi*f_PLL;
k_PLL=15;
psi_PLL=0.707;
tau_PLL=omega_PLL;

%% accelero and gyro internal bandwidths (internal first-order LPF)

f_IMU_BW = 5; %Hz
omega_IMU_BW =2*pi*f_IMU_BW;
lpfIMUinternal = c2d(tf(1/(s/omega_IMU_BW + 1)), Ts);
[numIMUint, denIMUint] = tfdata(lpfIMUinternal, 'v');


% IMU offsets
imu_accx_offset = 0.492-0.1-0.0517+0.024303;
imu_accz_offset = -0.2357-0.06;
imu_gyro_offset = -0.01489-0.0011;

%% Complementary filter
alpha_c = 0.999;
beta_c = 1-alpha_c;

%% Kalman Filter (for IMU)
Q_kalman_IMU=[0.000001, 0;0, 0.0001];
R_kalman_IMU=0.002;

%% current loop PI filter
Kp_i_loop=0.158;
Ki_i_loop=124;
mot_model=(1/(Rs+s*Ls));
curr_PI=(Kp_i_loop+Ki_i_loop/s);
iloop_PI =tf(curr_PI*mot_model/(1+curr_PI*mot_model));
iloop_PI_d=c2d(iloop_PI,Ts);
[numIloop, denIloop] = tfdata(iloop_PI_d, 'v');
```

7.3.6 *Simulink simulation*

The simulink simulation is used to collect the simulation data, exported to the MATLAB environment with the name of *statevars* as visible with the simulink blocks of figure 59.

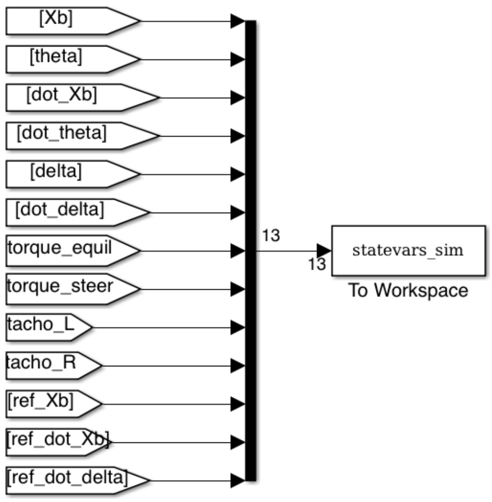


Figure 59: Simulink: blocks for exporting state variables to the MATLAB environment.

7.3.7 *bluetooth param writer*

After having set all the control parameters and the filters with the MATLAB software, in order to report al these settings to the board, a specific routine provides the *over-the-air* parameter send to the real-device control board via bluetooth. In addition to the parameters, also the simulation set-points are sent, so that the real-device experiment can execute the same work as the simulation.

This operation is based on a custom data-exchange serial protocol, implemented in MATLAB (for the PC-side) and in C for the embedded controller board. The protocol is the native implemented into the *Custom VESC* project, as described in [35], and the data flow is that shown in figure 60.

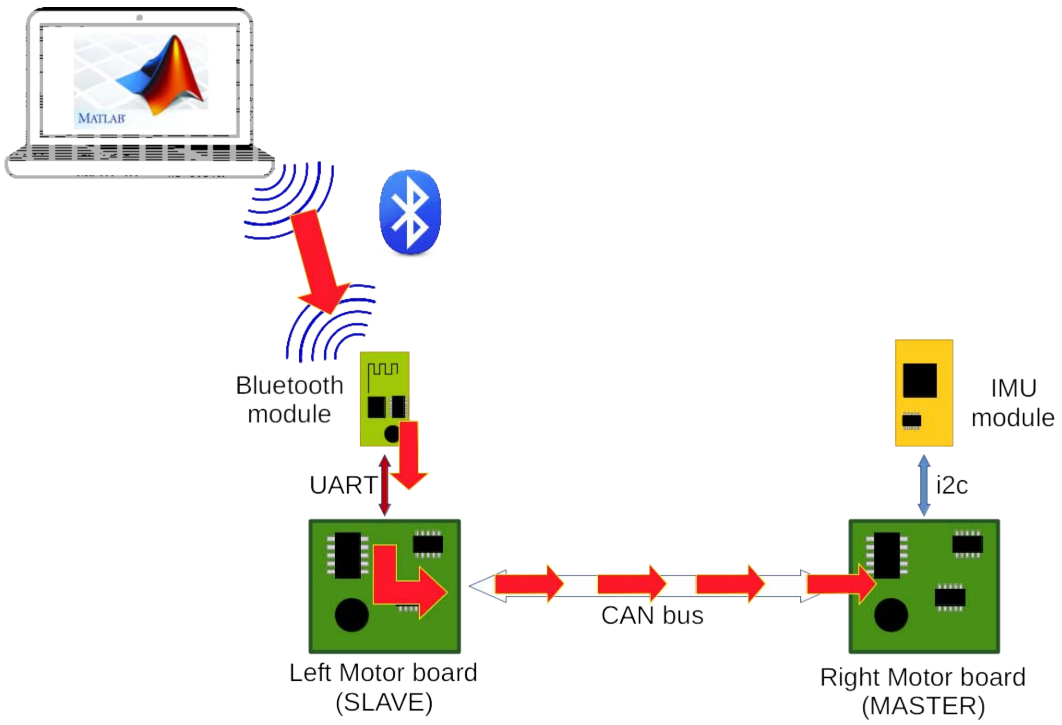


Figure 60: After having run the model parameters calculation, these data are sent from the PC, to the embedded board

7.3.8 *Data collection and post-processing*

The data collection and post-processing consists on sending via bluetooth the "start simulation" command and then collect via bluetooth all the data autonomously sent by the self-balancing vehicle.

After having sent the "start simulation" command, the PC "toggles its role", by "listening" the data coming from the device. In this situation, the data-flow is the opposite as that in figure 60.

```
close all;
n_samples = final_time/Ts;

ser = serial('/dev/tty.SELF_BALANCE_115200-SPP','BaudRate',115200);
fopen(ser);
ser.Terminator=3;

startSimulation(ser);

var_names=["Xb";"thetaP";"dotXb";"dotThetaP";"delta";"dotDelta";"torqueEq";"
    torqueSteer";"tachoL";"tachoR";"ref Xb";"ref dotXb";"ref dotDelta"];
num_vars=length(var_names);

timevector=zeros(1,n_samples);
statevars=zeros(num_vars,n_samples);
 %[...]

%% Reading from serial
% header = 0x02
% tail = 0x03
% format:
% header  len   payload CRC16 tail
%  (1B)   (1B)   (n*B)  (2B)  (1B)
ind=1;
buf_ind=1;

buffer='';
while ind<=n_samples
% [...collect samples]
fclose(ser);

run('state_reading_postproc.m');

function out = startSimulation(ser)
    COMM_START_SIM_SEQUENCE = 44;
    msg = MessageComposer;
    msg = msg.setCommand(COMM_START_SIM_SEQUENCE,true);
    msg = msg.addCrcHeadersAndSend(ser);
end
```

As visible near the end of the routine, a further step of "post-processing", provides the comparation between the collected samples and the simulation results: useful for model validation:

```
%% Collection finished: here is data-processing

timevector = timevector-timevector(1);
timevector = timevector*Ts;
```

```
figure();
subplot(2,1,1);
plotSimAndExperimVar("Xb",timevector,var_names,statevars_sim.Data,statevars,"m
    ");
subplot(2,1,2);
plotSimAndExperimVar("dotXb",timevector,var_names,statevars_sim.Data,statevars
    ,"m/s");

figure();
subplot(2,1,1);
plotSimAndExperimVar("thetaP",timevector,var_names,statevars_sim.Data,
    statevars,"rad");
subplot(2,1,2);
plotSimAndExperimVar("dotThetaP",timevector,var_names,statevars_sim.Data,
    statevars,"rad/s");

figure();
subplot(2,1,1);
plotSimAndExperimVar("delta",timevector,var_names,statevars_sim.Data,statevars
    ,"rad");
subplot(2,1,2);
plotSimAndExperimVar("dotDelta",timevector,var_names,statevars_sim.Data,
    statevars,"rad/s");

function graph = plotSimAndExperimVar(varname,timevector,var_names,sim_vars,
    exper_vars,y_unit_str)
    temp_string = 'Plot of simulated and experimental value of: ' + varname;
    ind =find(var_names==varname);
    temp_string = "ref "+varname;
    ind_ref=find(var_names==temp_string);

    if(ind_ref)
        reference=sim_vars(1:length(timevector),ind_ref);
        %reference=exper_vars(ind_ref,:)
        graph=plot(timevector,reference,timevector,sim_vars(1:length(
            timevector),ind),timevector,exper_vars(ind,:));
        title(temp_string);
        legendvars=["reference";"simulated";"Experimental"]
        legend(legendvars);
    else
        graph=plot(timevector,sim_vars(1:length(timevector),ind),timevector,
            exper_vars(ind,:));
        title(temp_string);
        legendvars=["simulated";"Experimental"]
        legend(legendvars);
    end
    xlabel('time [s]')
    temp_string = 'value of: ' + varname + ' [' +y_unit_str+']';
    ylabel(temp_string)
end
```

7.4 THE EMBEDDED ENVIRONMENT

This section describes the environment in which the control algorithm works. The focus is the firmware architecture and how the control algorithm uses it.

The microcontroller is the *STM32F405RGT6*, which is a Cortex-M4 microcontroller working at 168MHz, with hardware floating-point unit: this feature was very helpful for the algorithms performances.
The compilation toolchain chosen for the project was *gcc-arm-none-eabi 4.8 2014q2* and the operating system is a custom porting of *ChibiOs ver. 3.2* provided by [35].

The whole setup of the toolchain and the debugging tools (STM32F3discovery evaluation board with embedded ST-link) is also well-described in [35].
Here it will be shortly described the operating-system components involved for the (equilibrium and steering) control.

As visible in figure [], the control software makes use of the following peripherals, with the corresponding drivers named *Hardware Abstraction Layers (HAL)*:

- Timer/counter 5, with PWM HAL: used for the time base that triggers the control algorithm

- I2c2, with i2c HAL: used for the communication between microcontroller and IMU

- UART, with serial HAL: used for the communication with PC: the communication channel is bluetooth, but it is "transparent" for the serial communication to the PC

- CAN bus with CAN HAL: used for inter-board communication. In the following sections, a detailed description of the data exchanged for the two boards.

- Threads and signals: those are not peripherals, but it is important to note that the control algorithm works inside one of the running threads: this is a key feature offered by the operating system, that deeply simplifies the firmware development. Also the signals were very helpful for synchronization with external events. These operating system features were very helpful and simplified a lot both the firmware structure and the "code readability".

- time measurement: also this is not a peripheral, but a primitive offered by the operating system. The time measurement was used for the calculus of algorithm performance.

The embedded environment is a short description of the hardware and firmware environment where the algorithm works. In the following section a deeper description of the control algorithm.

7.5 THE CONTROL FIRMWARE

In this section a detailed description of the control algorithm: how it works and how the external components are involved. Also some short code-blocks describing the algorithm.

Even in this section, a short discussion of the board's data flow, during other system tasks.

7.5.1 *The control algorithm*

The control algorithm consists of a single routine, executing at fixed time interval. This routine executes in loop, inside an operating system's thread:

```
// create the thread of control loop. Attention: this function holds the event
     catchers, thus it
// is necessary to start it before the event signals
chThdCreateStatic(ctl_loop_thread_wa, sizeof(ctl_loop_thread_wa), HIGHPRIO ,
    ControlLoopMain, NULL);
```

Even if it has strong time constraints, it doesn't execute inside an interrupt service routine (*ISR*) because it has to perform wait-cycles of external data and this is incompatible with ISR. Also the motivation is that the micro-controller used to run this algorithm must perform many other activities such as motor control, that has even stronger time constraints, because it also has "safety issues", imposed by over-current control.

In this view, the following strategy was adopted:

1. The pwm timer issues an interrupt at regular and very precise time intervals

```
void init_PWM_loop_trigger(uint8_t loop_period_ms){
        pwmStop(&PWMD5);
        // initialize PWM: the timer that triggers the loop function.
        pwmStart(&PWMD5, &pwmcfg);
        pwmChangePeriod(&PWMD5,(LOOP_TIMER_FREQ*loop_period_ms)/1000);
        pwmEnablePeriodicNotification(&PWMD5);
        // Starts the PWM channel 0 using 11% duty cycle.
        pwmEnableChannel(&PWMD5, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD5, 1100)
            );
        pwmEnableChannelNotification(&PWMD5, 0);
}
```

2. The ISR issues a "soft" signal (event)

```
// PWM peripheral: at timer reload provides the "tick" for the control
    loop
static void pwmpcb(PWMDriver *pwmp) {
// PWM reset (period) routine
  (void)pwmp;
  chEvtSignal(ctl_loop_thread,CH_EVT_TMR_LOOP_TRIGGER);
}
```

3. The control loop was "freezed", waiting this event. Once it is read, the control routine executes until its end (that comes when the torque commands are posted to the actuators) and loops again, freezing while waiting another trigger event.

```
static THD_FUNCTION(ControlLoopMain, arg){
        //control loop initialization
        for(;;){
                chEvtWaitAny((eventmask_t) CH_EVT_TMR_LOOP_TRIGGER);//
                    wait the event of the timer routine
                //...
        }
}
```

The drawback of the solution is that there is not the guarantee of precise time execution of the routine, since it runs in a "normal thread" and it is interruptible any time any ISR. This is not a big defect since the routine execution time is affected even by external data provision, that might introduce an important variable-latency behavior. Also the other positive-side of this approach is that the "trigger" remains precise, because it is generated by a hardware-timer and so the loop doesn't accumulate any time-execution delay.

In brief, the key steps of the control algorithm are:

1. data collection
2. data filtering/processing
3. state reconstruction
4. control signals generation
5. errors management

In the following subsections, the description ot these steps.

7.5.1.1 *data collection*

The MASTER board reads:

- its motor's (RIGHT) own position

```
// the local motor position
// NOTE: the motors are mounted in opposite direction, so it is necessary
    to flip MOT_R sign!
state_vars.tacho_R = -(mc_interface_get_tachometer_value(false));
```

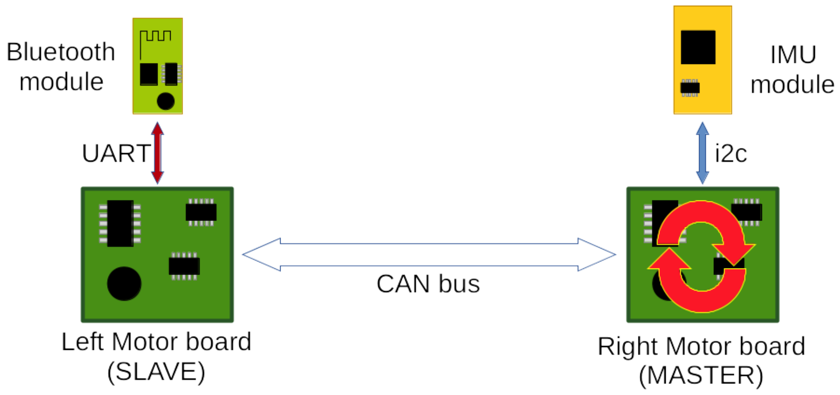


Figure 61: Master board reading its internal motor position (in RED the data flow).

Note: this value is an integer, 32-bit value that increments or decrements at every hall-sensor commutation. This is the simplest data-collection task since the data is resident in the board: no error is possible here .

- the IMU accelerometer and gyroscope registers

```
// read the IMU (has embedded timeout)
imu_ok=getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

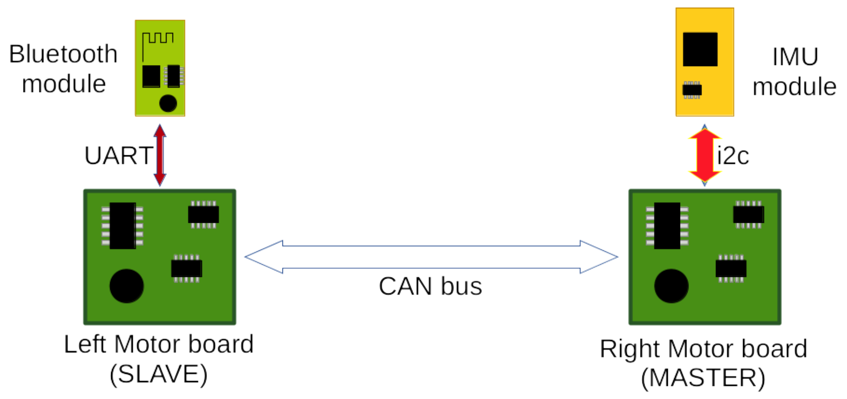


Figure 62: Master board reading the IMU registers (in RED the data flow).

In the routine it is noticeable a return-code for the operation: this sets to "false" in case of timeout (1ms) or reading error (i2c error). The error management will be discussed afterwards.
The values coming from the IMU are all 16-bit integers and the value represented depends on the full-scale set. Here the initialization values :

```
accel_full_scale = MPU9250_ACCEL_FS_2;  // [g]
gyro_full_scale = MPU9250_GYRO_FS_250;  // [deg/s]
imu_LPF_settings = MPU9250_DLPF_BW_256; // [Hz]
```

- the remote (LEFT) motor position The remote motor is read via the CAN bus:

```
//read the remote motor position
comm_can_get_tacho(MOT_L);

// wait the slave motor position to arrive (with a timeout, signaled as 0
     value return)
evt_received = chEvtWaitAllTimeout(CH_EVT_MOT_L_TACHO_RECV,LOOP_TIMEOUT);
```

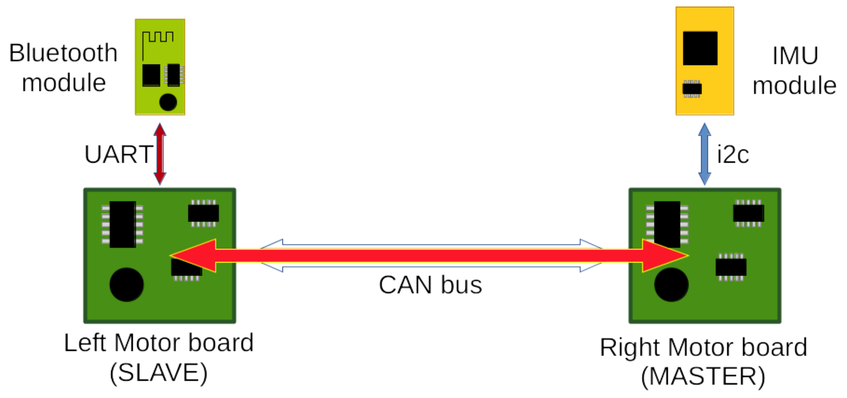


Figure 63: Master board reading the remote motor position (in RED the data flow).

Even here the data read is an integer, 32-bit value that increments or decrements at every hall-sensor commutation, and it is present an error detection (discussed afterwards).

After having collected all the data, it is time for the processing.

7.5.2 *data filtering/processing*

The main tasks in the data processing are those described in the chapter 6, relative to the:

1. angle reconstruction. This task is accomplished by the following routine, whose inputs are the IMU registers previously read:

```
// input: ax, ay, az, gx, gy, gz
// output: thetaP, dot_thetaP
void IMUSensorFusion(int16_t ax, int16_t ay, int16_t az, int16_t gx,
    int16_t gy, int16_t gz, float *thetaP, float *dot_thetaP){
        static float pitchAngleComplementary = 0;

        // avoid warnings
        ay++;
        gx++;
        gz++;

        float accX,accZ,gyroY;
        float pitchAngAccLin;

        // Accelerations: the output is a raw number, compared with full-
            scale, and the unit is "g".
        // Convert the measure in  m/s^2
        accX = (float)ax * accelero_coeff;
        accZ = (float)az * accelero_coeff;

        // angular velocities: in rad/s
        gyroY = (float)gy * gyro_coeff;

        // subtract the (already measured) offsets
        accX = accX -self_balancing_params.imu_accx_offset;
        accZ = accZ -self_balancing_params.imu_accz_offset;
        gyroY= gyroY+self_balancing_params.imu_gyro_offset;

        // variation: from ax, subtract the horizontal acceleration
        accX = accX-state_vars.ddot_Xb*cos(state_vars.theta_P);
        accZ = accZ+state_vars.ddot_Xb*sin(state_vars.theta_P);

        //float accel_norm = sqrt((Axyz[0]*Axyz[0])+(Axyz[1]*Axyz[1])+(
            Axyz[2]*Axyz[2]));
        float accel_norm = sqrt((accX*accX)+(accZ*accZ));// probably more
             correct
        float sin_pitch = -accX/accel_norm;
        float cos_pitch = sqrt(1.0-(sin_pitch*sin_pitch));

        float pitchAngAcc = atan2f(sin_pitch,cos_pitch);
        // avoid "divide-by-zero"
        if(accZ!=0.0)
                pitchAngAccLin = (accX/accZ);
        else
                pitchAngAccLin = (accX/0.0001);

        // low-pass filter the angle obtained by the noisy accelerometer
        float pitchAccFiltered = filter_second_order_process_sample(&
            self_balancing_params.accelero_LPF,pitchAngAccLin);
        // sensor fusion: in order to reduce the noise in the angle,
            introduced by accelerometers, use
        // the angle speed integral:
        *dot_thetaP = -gyroY;

        // the set_Xb_filt filter is a generic integrator, so we use its
            coefficients
        pitchAngleComplementary =  -self_balancing_params.set_Xb_filt.D1*
            pitchAngleComplementary + self_balancing_params.set_Xb_filt.
            N1*(*dot_thetaP);


        //pitchAngleComplementary = self_balancing_params.
            compl_filt_alpha*pitchAngleComplementary +
//      //
    self_balancing_params.compl_filt_beta*pitchAccFiltered;
//      pitchAngleComplementary =       self_balancing_params.
    compl_filt_alpha*pitchAngleComplementary +
//
    self_balancing_params.compl_filt_beta*pitchAngAccLin;
        pitchAngleComplementary =       self_balancing_params.
            compl_filt_alpha*pitchAngleComplementary +
            self_balancing_params.compl_filt_beta*pitchAngAcc;

        //kalman_Process_sample(k_sys,pitchAccFiltered,*dot_thetaP);
        //kalman_Process_sample(k_sys,pitchAngAccLin,*dot_thetaP);
        kalman_Process_sample(k_sys,pitchAngAcc,*dot_thetaP);

        float pitchAngleKalman = k_sys->angle;

        if(state_vars.send_angle_via_bluetooth){
                SendAngleFusionOverBluetooth(state_vars.loop_counter,
&accX,
&accZ,
&pitchAngAccLin,
&pitchAccFiltered,
dot_thetaP,
&pitchAngleComplementary,
&pitchAngleKalman,
&k_sys->err);
        }

        //*thetaP = pitchAngleComplementary;
        *thetaP = pitchAngleKalman;
}
```

As noticeable looking at the code, this routine has several variation possible, depending on what code rows are active and what are commented out. In fact the angle from the accelerometer can be calculated with linear approximation, or with "true" trigonometric value. Aldo the output of the accelerometer can be further filtered by a second-order filter (or not). Finally the pitch angle can be "fused" by complementary or by Kalman filter.

One important thing to note is the subtraction of the linear acceleration: since the control-loop has a fast dynamic, compared to that of the real device, it is assumed that the linear acceleration and the pitch angle are almost constant, during the control-step time. Thanks to this, it is possible to subtract the linear acceleration from the accelerometer reading, based on the previous-cycle calculated pitch angle.

2. velocity reconstruction

   The task of velocity reconstruction (for both the linear velocity, and the yaw-rate) consists of filtering the position data with the second-order filter described in 6.5.0.8. The discrete-time (second-order) filter, expressed in the z-domain has the following input-output form:

$$H(z^{-1}) = \frac{N0+N1*z^{-1}+N2*z^{-2}}{1+D1*z^{-1}+D2*z^{-2}} \tag{109}$$

   Then, the implementation of the previous discrete-time filter have been developed with the following C-code :

```
/**
 * Process one sample of a second-order filter
 * @param filter
 * The filter object that stores the memory of previous samples and
     coeffs
 * @param new_sample
 * The sample to process
 * @return
 * The current output, as a consequence of the new-sample processing
 */
float filter_second_order_process_sample(FILT_second_order *filt, float
    new_sample){
        float curr_out = -filt->D1*filt->prev_output-
                                        filt->D2*filt->preprev_output+
                                        filt->N0*new_sample+
                                        filt->N1*filt->prev_input+
                                        filt->N2*filt->preprev_input;
        filt->preprev_output=filt->prev_output;
        filt->prev_output=curr_out;
        filt->preprev_input=filt->prev_input;
        filt->prev_input =new_sample;
        return curr_out;
}
```

   Where the memories of the previous-states of both input and output are stored inside the "filter" object structure:

```
/** Second order digital filter. It is represented in the form:
* Y(z)     N0 + N1*z^(-1) + N2*z^(-2)
* ---- =  ----------------------------
* U(z)      1 + D1*z^(-1) + D2*z^(-2)
* Thus, the 5 coefficients must be specified
*/
typedef struct{
        float prev_input; // u(z^(-1))
        float preprev_input; // u(z^(-2))
        float prev_output;// y(z^(-1))
        float preprev_output;// y(z^(-2))
        float N0;
        float N1;
        float N2;
        float D1;
        float D2;
}FILT_second_order;
```

The velocity is then the desired output value coming with the output of the *filter second order process sample* function.

The velocity reconstruction is part of the following "state reconstruction" stage, described in the following subsection.

7.5.3 *state recontstruction*

The state reconstruction is (in this specific-case) the process of appliying all the necessary linear transforms that permits to obtain the state-variables, starting from de device's measures, as described by the blocks of figure 64:

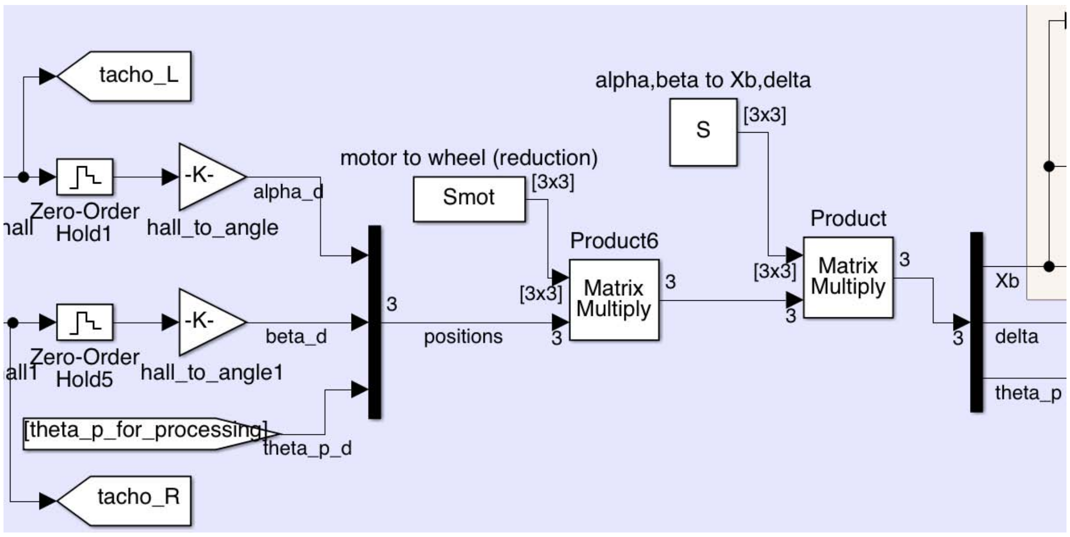


Figure 64: Retrieving the state variables from the quantized measures. These blocks represents the work inside the microcontroller routines.

The corresponding code is:

```
void UpdateStateVars(int32_t alpha_tacho, int32_t beta_tacho, float theta_P){

        state_vars.Xb = TachoToAngle()*theta_P + (alpha_tacho*
            self_balancing_params.r_wheel*self_balancing_params.rho*
            TachoToAngle())/2 + (beta_tacho*self_balancing_params.r_wheel*
            self_balancing_params.rho*TachoToAngle())/2;
        state_vars.delta = TachoToAngle()*theta_P + (alpha_tacho*
            self_balancing_params.r_wheel*self_balancing_params.rho*
            TachoToAngle())/self_balancing_params.d_interwheel - (beta_tacho*
            R_WHEEL*RHO*TachoToAngle())/self_balancing_params.d_interwheel;
        state_vars.theta_P=theta_P;

        // calculate velocities by HPF of positions
        state_vars.dot_Xb=filter_second_order_process_sample(&
            self_balancing_params.dot_Xb_filt,state_vars.Xb);
        state_vars.dot_delta=filter_second_order_process_sample(&
            self_balancing_params.dot_delta_filt,state_vars.delta);

        // calculate acceleration
        state_vars.ddot_Xb=filter_second_order_process_sample(&
            self_balancing_params.ddot_Xb_filt,state_vars.dot_Xb);
}
```

As noticeable, the routines retrieves also the velocities from the positions. The last routines calculates also the acceleration, used in the accelerometers' angle calculation.

7.5.4 *control signals generation*

The last task remaining is the generation of the control signals, using the controller designed and considering the references:

```
//  Schema:
// |Joy_FWD|*gain = set_ddot_Xb ->INTEGRATE-> set_dot_Xb ->INTEGRATE-> set_Xb
//
void ControlLoopBalance(void){
        if(state_vars.simulation_active){
                set_dot_Xb = reference_get_curr_value(&ref_seq_dot_Xb,((float)
                    state_vars.loop_counter*self_balancing_params.
                    loop_period_ms/1000.0));
        }else{
                // the joystick (forward direction) sets the acceleration
                    reference
                set_ddot_Xb = JOY_FWD_TORQUE_GAIN*JoystickGetForward();

                // find reference speed angle by integrating reference
                    acceleration
                set_dot_Xb = filter_first_order_process_sample(&
                    self_balancing_params.set_dot_Xb_filt,set_ddot_Xb);

                // by now, set all references to 0
                set_Xb = 0;
                set_dot_Xb = 0;
        }

        // find reference speed angle by integrating reference acceleration
        set_Xb = filter_first_order_process_sample(&self_balancing_params.
            set_Xb_filt,set_dot_Xb);

        float bal_torque=0.0;
        bal_torque=     self_balancing_params.gainsLQR[0]*(set_Xb-state_vars.
            Xb)+
                                self_balancing_params.gainsLQR[1]*(-state_vars
                                    .theta_P)+ // the setpoints of both thetaP
                                     and dot_thetaP will always be 0
                                self_balancing_params.gainsLQR[2]*(set_dot_Xb-
                                    state_vars.dot_Xb)+
                                self_balancing_params.gainsLQR[3]*(-state_vars
                                    .dot_theta_P);// the setpoints of both
                                    thetaP and dot_thetaP will always be 0

        // apply the gains to retrieve the balance torque:
        SetBalanceTorque(bal_torque);
}

void ControlLoopSteeringPID(void){
        if(state_vars.simulation_active){
                set_dot_delta = reference_get_curr_value(&ref_seq_dot_Delta,((
                    float)state_vars.loop_counter*self_balancing_params.
                    loop_period_ms/1000.0));
        }else{
                // the steering speed reference comes from joystick
                set_dot_delta = JOY_STEER_GAIN*JoystickGetSteering();
        }

        float dot_delta_error=(set_dot_delta-state_vars.dot_delta);
        // the PI controller is (at the end) a first-order filter, whose
            coefficients were calculated with
        // MATLAB
        float yaw_torque=filter_first_order_process_sample(&
            self_balancing_params.steering_PID,dot_delta_error);

        // apply the gains to retrieve the steering torque:
        SetYawTorque(yaw_torque);
}

void ControlLoopSteeringLQR(void){

        if(state_vars.simulation_active){
                set_dot_delta = reference_get_curr_value(&ref_seq_dot_Delta,((
                    float)state_vars.loop_counter*self_balancing_params.
                    loop_period_ms/1000.0));
        }else{
                // the steering speed reference comes from joystick
                set_dot_delta = JOY_STEER_GAIN*JoystickGetSteering();
        }

        // find reference yaw angle by integrating angle speed
        set_delta = filter_first_order_process_sample(&self_balancing_params.
            set_delta_filt,set_dot_delta);
        //set_delta = 0; // set delta to zero and act only on angular velocity
            : less reactive, but avoids error accumulation due to integration

        //self_balancing_params.set_dot_delta= 0;
        float yaw_torque=(self_balancing_params.gainsLQRsteering[0]*(set_delta
            -state_vars.delta)+
                        self_balancing_params.gainsLQRsteering[1]*(
                            set_dot_delta-state_vars.dot_delta));
        // apply the gains to retrieve the steering torque:
        SetYawTorque(yaw_torque);
}
```

As noticeable, the references for both the equilibrium and the steering controllers might come from two different sources:

- from the joystick, during the "normal" operation
- from the *reference builder*, during the simulation context (when the bluetooth has received a signal of "start simulation" and the device attempts to execute the same work of the Simulink's simulation).

The "reference builder" is used to simulate the reference generator ("repeating sequence" block) in the Simulink model, by permitting to store some set-points (in time and amplitude) and to return an interpolated value at every time-step in the sequence:

```
typedef struct{
        float time;
        float value;
}REF_POINT;

typedef struct{
        int8_t last_valid; // index of last valid setpoint
        REF_POINT refpoints[REF_MAX_REFERENCE_POINTS];
}REF_SEQUENCE;

/*
 * Add a setpoint to a reference sequence
 *
 * @param the sequence
 * @param the time of a setpoint
 * @param the value o set at the corresponding time
 * @return false if sequence is already full or if time requested is less than
      that of the last point
 */
bool reference_add_setpoint(REF_SEQUENCE* seq, float time_of_vlaue, float
    value){
        if(seq->last_valid==REF_MAX_REFERENCE_POINTS) return false;
        if(time_of_vlaue!=0 && time_of_vlaue <= seq->refpoints[seq->last_valid
            ].time) return false;

        // if none of the previous, everything OK
        seq->refpoints[++seq->last_valid] = (REF_POINT) {time_of_vlaue,value};
        return true;
}

/*
 * This updates the current value and the other parameters, according to the
     time passed as a parameter,
 * by interpolating the reference points. Any reference point whose time value
      is less than its previous
 * is considered as non-valid, like all the others succeding
 *
 * @param the sequence in which to read points
 *
 * @param the absolute time in the sequence.
 *
 * @return the value corresponding to the time specified. If the time is
     greater than the time of
 *                     the last reference point, it will be returned the last
      value
 */
float reference_get_curr_value(REF_SEQUENCE* seq, float time){
        if(time==0){
                if(seq->refpoints[0].time == 0){
                        return seq->refpoints[0].value;
                }else{
                        return 0;
                }
        }else{
                // if time is greater than last valid setpoint
                if(time>=seq->refpoints[seq->last_valid].time)
                        return seq->refpoints[seq->last_valid].value;

                uint8_t i=0;
                while(i<REF_MAX_REFERENCE_POINTS){ // find the imediately-next
                     point
                        if(seq->refpoints[i].time > time) break;
                        i++;
                }
                // interpolation
                if(i==0){
                        float ang_coeff = seq->refpoints[0].value/seq->
                            refpoints[0].time;
                        return time*ang_coeff;
                }else{
                        float ang_coeff = (seq->refpoints[i].value-seq->
                            refpoints[i-1].value)/
                                        (seq->refpoints[i].time-seq->refpoints
                                            [i-1].time);
                        return (seq->refpoints[i-1].value)+(time-seq->
                            refpoints[i-1].time)*ang_coeff;
                }
        }
}
```

7.5.5 *errors management and safety strategies*

The error management is more than necessary, because the system is distributed and many errors might compromise the correct algorithm work, especially those regarding the communication.
Those errors are admitted if-and-only-if they are very reduced in number: the control might tolerate the missing of one sample and "skip" one control-loop, but is the errors increase it results in hard compromise of filter's outputs and then controller's commands. The communication-fault management is this:

```
if(imu_ok && (evt_received!=0)){// if both motors received and even IMU
    received:
        retries_cnt=MAX_LOOP_RETRIES;// all OK: reset retries counter and exec
             routines only if no timeout

        // [...]

}else{ // if there is a communication failure
        sprintf((char*)buf,"IMU:%d,CAN:%d\n",imu_ok,(int)evt_received);
        comm_can_tunnel_bluetooth_buffer(buf,12);
        // a timeout has occurred: decrease retries counter
        if(retries_cnt!=0){
                retries_cnt--;
        }else{ // all retries expired
                comm_can_tunnel_bluetooth_buffer((unsigned char*)"RETRIES
                    EXPIRED!\n",17);
                EmergencyBrake();
                break;
        }
}
```

Note that the communication failure means that either there was an error in the data received, or there was a timeout in the reception of IMU data or the remote board's position answer.

In case of fault, even only one, a notification is sent via bluetooth and if the retries expires the controller can't perform any other correct work, thus calls the *EmergencyBrake* procedure, that simply sets 0 current to the motors (let them free-rotate). Those errors were frequent in the initial part of the project, but once the communications were all correctly tuned, no failure notifications have been registered during all the experiments.

Two conditions are checked for the purpose of *safety*:

1. The controller must work only if tilt angle doesn't exceeds the limit (set to ±20°), where the linearization assumptions doesn't hold anymore and the control action might result "risky":

```
// attention: apply the controls only if the tilt angle is within the
    limits!!!
if((state_vars.theta_P<CTL_MAX_TILT_ANGLE_RADS)&&(state_vars.theta_P>(-
    CTL_MAX_TILT_ANGLE_RADS))){
        ControlLoopSteeringPID();
        //ControlLoopSteeringLQR();
        ControlLoopBalance();
        eq_cnt=0;
}else{
        if((eq_cnt)<10){
                EmergencyBrake();
                eq_cnt++;
        }
}
```

   if not possible, stop any action to the motors and let them free-rotate.

2. At every control loop it is monitored the state of the *Safety button*: a normally closed contact that serves for the *Emergency Stop* of the vehicle:

```
// IMPORTANT safety guard: monitoring red button pressure
if(palReadPad(EMERGENCY_BUTT_PORT,EMERGENCY_BUTT_BIT)){ // if red button
    is pressed
        if(emgcy_cnt<10){
                EmergencyBrake();
                // reset the tachometers of both left and right wheels
                mc_interface_get_tachometer_value(true); // local motor's
                     tacho
                comm_can_reset_tacho(MOT_L);                    //
                    remote motor's tacho
                ResetStateVars();
                emgcy_cnt++;
        }

}else{
        emgcy_cnt=0;
        UpdateTorquesToVehicle();
}
```

   the emergency button was very useful during the experiments, even for stopping/restarting the device and to test its free-evolution behavior.

7.6 conclusions

In this chapter it was presented "all the software" in the project (all the source-code will become shortly open-source and published), both for the model buildup and for the real-time embedded controller.

This chapter have focused only on the key-routines, explaining some algorithms and underlying considerations. As stated before, the embedded code of the controller and the filters coexists in the micro-controller that drives the right motor. The correct work and the absence of communication errors and latency failures was considered a big success of integration of an evolved-controller code into a successful open-hardware project's device.

The possibility to perform many automated tasks (such as parameter update, start/stop simulations and collecting data) remotely via bluetooth is considered by the author a great result, useful for future investigations and/or improvements works.

Those are the motivations that drove the idea of sharing the work with an open-source project, fort the benefit of both hobbyists/passionate and for scientific analysis and community-provided project improvement contributions.

8 MODEL VALIDATION

8.1 Introduction

After having described the scope of the work, the components chosen and the models and algorithms applied, this chapter aims to *validate* all the models built and the solutions proposed. The main goal at the end of this work is to verify to have a good matching between the theoretical model built and the true device.

Having this, is a very important result, because it permits to test improved algorithms, new features and controllers tuning, all inside the simulator, which guarantees the result matching with the true system, without the need to waste much time with running the experiments with the real-system itself.

Another important result of the validation process is the proof of the effectiveness and stability of the designed controls, in order to govern the problem. A good simulation environment, in fact, is only a good starting point, but without a "good-controlled system" proof, many aspects might have been remained hidden.

This chapter thus aims to show the validation and to discuss this task's issues.

8.2 Velocity and acceleration reconstruction

As stated in chapter 6, several algorithms have been proposed for the derivation of velocity, starting from the position reading.
In this section a discussion about performances of the various algorithms and the final choice.

First of all, the different algorithms were tested during the free evolution of the self-balancing transporter, as visible in figure 65, that is useful both for testing the algorithms during low-speed and high-speed velocity variations.

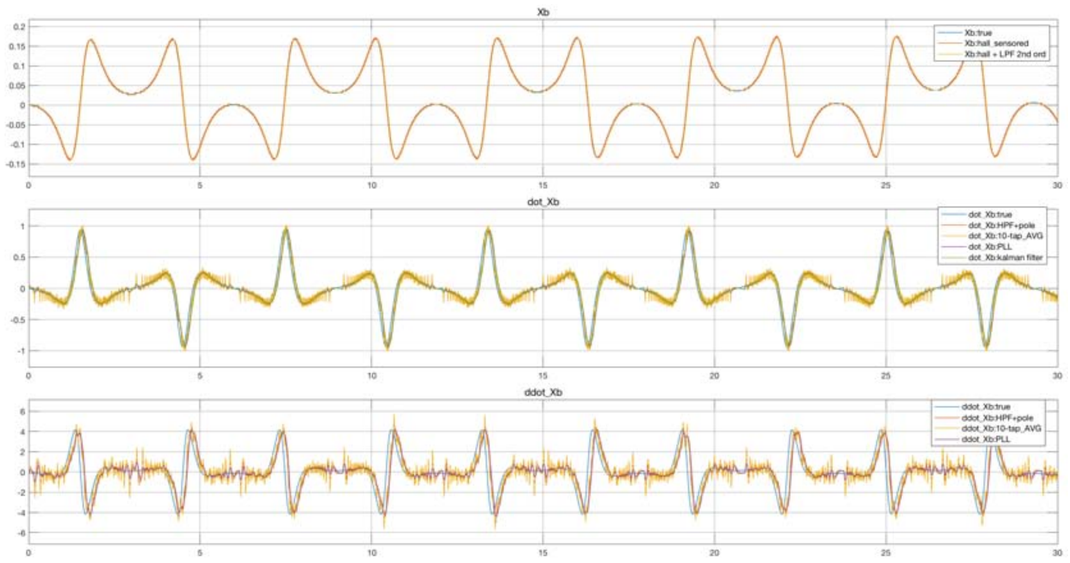


Figure 65: Position, velocity and acceleration reading: comparison of different algorithms.

As easily noticeable, figure 65 exhibits that the measure coming from the AVERAGE-derivative algorithm has a high "residual-noise", even having averaged 10 samples: this is due to the high sensitivity to the position "steps", generated by the hall sensors' output.

Even having this undesired behavior, it is interesting to zoom the figures before leaving at all this algorithm.

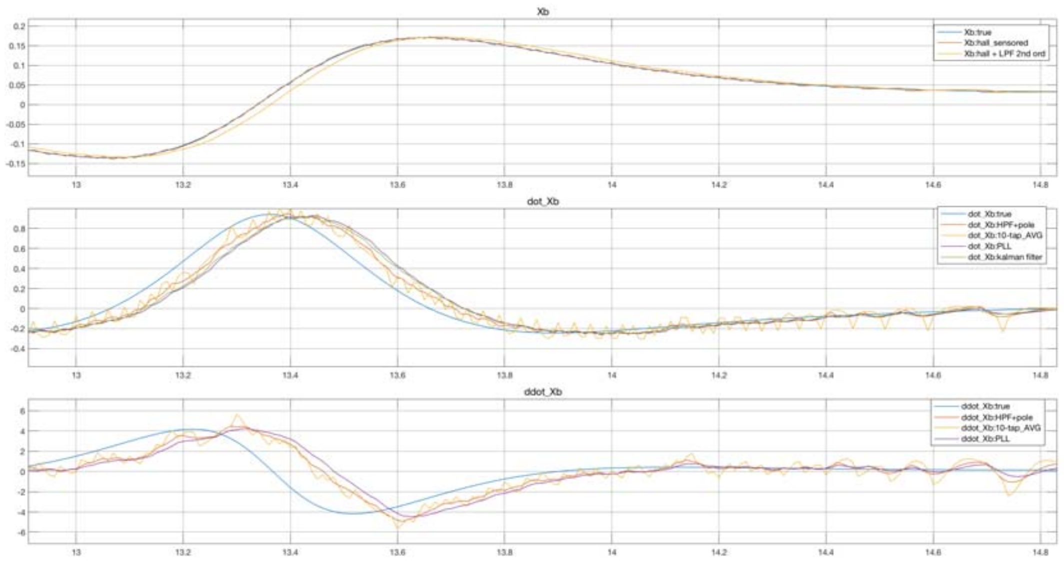


Figure 66: Position, velocity and acceleration reading: a zoom for investigation about algorithm's behavior during fast and slow transients.

Looking at figure 66, it is possible to note that the AVERAGE-derivative, even if noisier than the others has a lower delay in the velocity tracking, compared to the others.

A similar small delay is obtained with the filter "HPF + pole", that has additionally a lower residual noise. This is why this algorithm is the preferred and that implemented in the "real-life" device. Its computational "low-cost", in fact makes it very useful for the velocity-reconstruction task.

The other two algorithms, once correctly tuned, has similar behavior of the "HPF+pole" one, but has a higher computational weight. This is why the final solution was the "HPF+pole" algorithm, with the ω_{HPF} placed at the frequency of 5Hz.

8.2.1 *position and acceleration*

Looking again at figure 66, a brief consideration about position: even if the position filtered by the 10Hz-LPF filter is very smoother than that directly read from the hall-sensors, this has the disadvantage of a higher delay during the fast-slopes (that might eventually compromise stability) and a poor noise-rejection during the slow position-variations.
This is why in the true-device, the LPF was not used in the position reading.

Concerning the acceleration, the same algorithm chosen for the velocity, was adopted even for the acceleration. The result, in the last figure shows that the choice is a good compromise between responsiveness and low noise addition.

Figure 67 confirms the choice of filter "HPF+pole" as preferable even during a simulation of a controlled sequence.

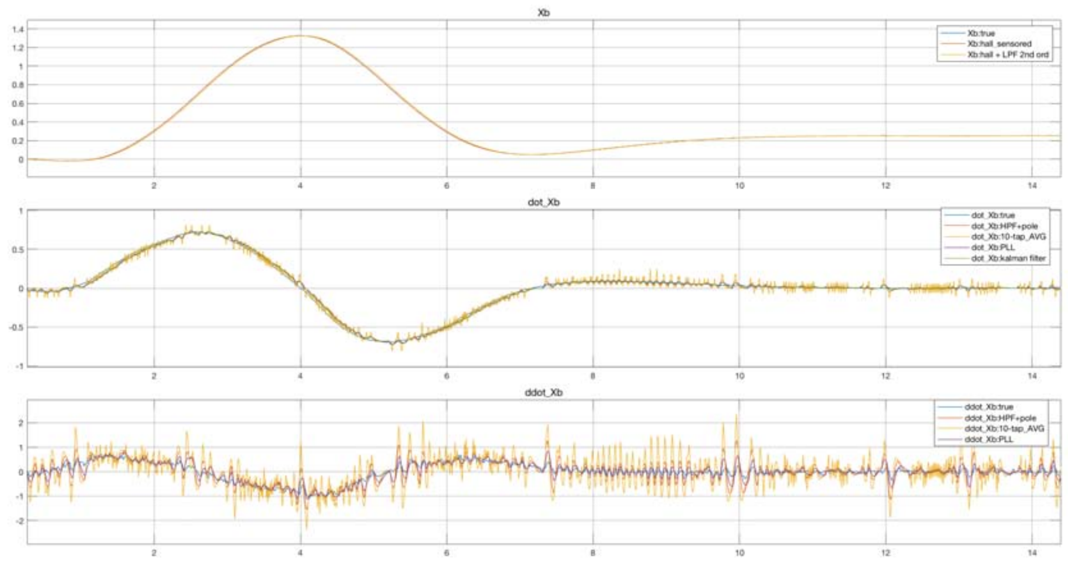


Figure 67: Position, velocity and acceleration reading during a simulated control sequence (not in free evolution).

The velocity and acceleration has low additional noises and an imperceptible delay.

Given the good results for the linear velocity reading, the filter was adopted even for the readout of the angle-rate. In this case the effect of the hall sensors has higher impact and the consequent superimposed noise is stronger, but even acceptable as visible in figure 68.

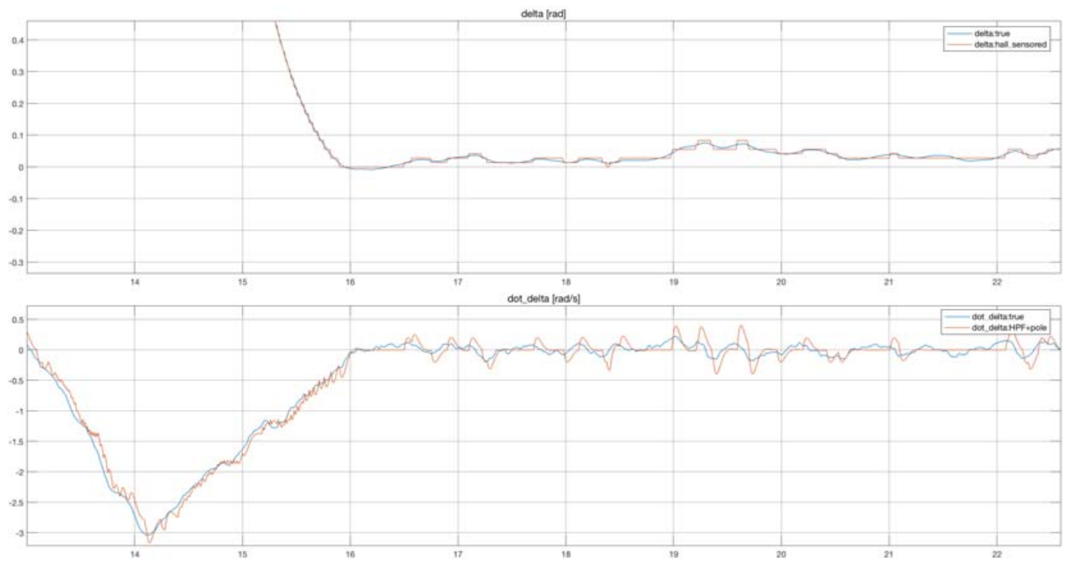


Figure 68: Yaw angle and angle-rate reading, during a simulated control sequence (not in free evolution) using the velocity-reading filter "HPF+pole" set with $\omega_{HPF} = 2\pi * 5\text{Hz}$.

The usage of the same filter both for the linear velocity and for the yaw angle-rate is not always acceptable, because these rates are different, and the difference deeply depends on the dynamics expected for the two phenomena.

At the end of the discussion about velocity reading, what is finally used is a single, second-order filter that acts (with different filter instances) in linear velocity,

acceleration and yaw rate. This means that the self-balancing device will implement three second-order-filters with identical parameters.
The definition of filters' parameters was automated by the MATLAB scripts inside the file "digital filters". Once defined there, the scripts provides the automatic parameters upgrade to the real-device, via bluetooth.

8.3 IMU MODEL VALIDATION AND TUNING

The IMU is one of the key components of the system. Its description and the corresponding model have been already described in chapter 6. Also in that chapter a discussion about the angle measurement by sensor-fusion.

In this section it will be described the validation procedures that aim to verify:

- the matching between datasheet's data and true measures
- the model scheme validation by measures
- the calibration of the IMU
- the tuning of the sensor-fusion algorithms

In this section it is described the analisys, valid for the specific component *Invensense MPU9250* [28].

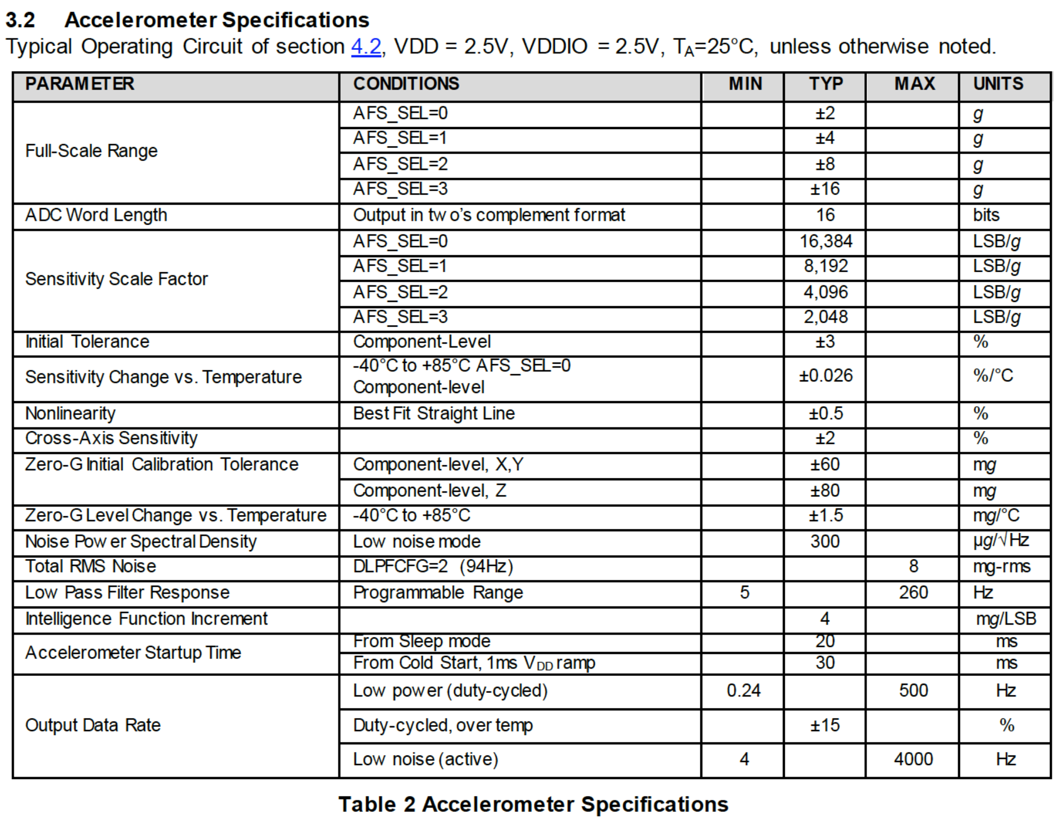
8.3.1 *datasheet verification*

One of the most important things reported in the datasheet is the noise entity and type. Noise affects both accelerometer and gyroscope measures: in order to build an affordable model, it must report the same type of noises of the real one.

For both sensors, the noise type is a *white noise*, with constant power along the frequency span. In the following subsections the verification of the datasheets data and the models' parameters data population.

8.3.1.1 *Accelerometer noise and bias*

The datasheet of the accelerometer reports the following data:



3.2 Accelerometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*
	AFS_SEL=1		8,192		LSB/*g*
	AFS_SEL=2		4,096		LSB/*g*
	AFS_SEL=3		2,048		LSB/*g*
Initial Tolerance	Component-Level		±3		%
Sensitivity Change vs. Temperature	-40°C to +85°C AFS_SEL=0 Component-level		±0.026		%/°C
Nonlinearity	Best Fit Straight Line		±0.5		%
Cross-Axis Sensitivity			±2		%
Zero-G Initial Calibration Tolerance	Component-level, X,Y		±60		mg
	Component-level, Z		±80		mg
Zero-G Level Change vs. Temperature	-40°C to +85°C		±1.5		mg/°C
Noise Power Spectral Density	Low noise mode		300		µg/√Hz
Total RMS Noise	DLPFCFG=2 (94Hz)			8	mg-rms
Low Pass Filter Response	Programmable Range	5		260	Hz
Intelligence Function Increment			4		mg/LSB
Accelerometer Startup Time	From Sleep mode		20		ms
	From Cold Start, 1ms V_{DD} ramp		30		ms
Output Data Rate	Low power (duty-cycled)	0.24		500	Hz
	Duty-cycled, over temp		±15		%
	Low noise (active)	4		4000	Hz

Table 2 Accelerometer Specifications

Figure 69: Section of the datasheet of MPU9250, regarding the accelerometer.

The parameter regarding the noise is that called "Noise Power Spectral Density", expressed in $\mu g/\sqrt{Hz}$. The Simulink block used to model it, is the "Band Limited White Noise", which expects the value of the "Noise Power", that is expressed in unit (in this case) of $(\mu g)^2/Hz = (\mu g/\sqrt{Hz})^2$. This means that the value to introduce in the model is the square of the "Noise Power Spectral Density" value thus: $(300 * 10^{-6} * g)^2$.

The validation of this noise model comes from the comparison of the output of the (unfiltered) accelerometer's simulated and real data. Obviously, during this measure, the IMU unit must be perfectly steady, thus the measure represents the sensor's own noise. In figure 70 the results.

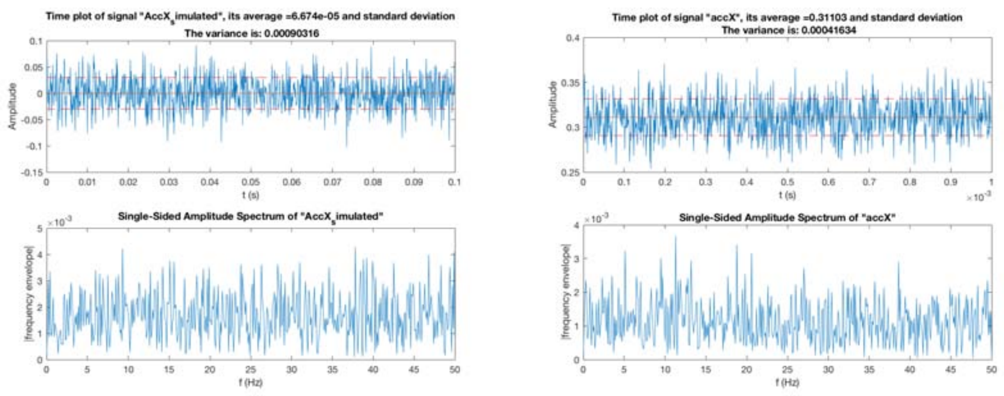


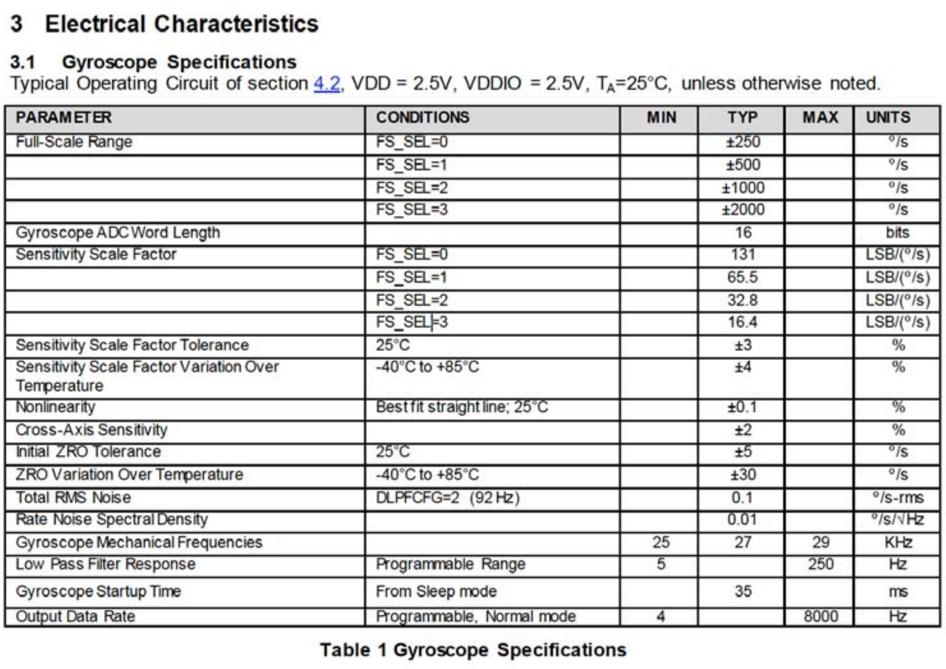
Figure 70: Noise at the "X" output of the accelerometer. On the left, the simulated block, modeled with a Power Spectral Density equal ot the value found in the datasheet. On the right the real-one. The outputs are very close each other and the noise variance is also similar.

The comparison suggests that the model is almost correct and the signal variance is similar. In case it is found a "real value" for the noise spectral density, heavily different from that of the datasheet, a correction to the model's parameters must be made.

An important thing to note in the figure is that the "real accelerometer" exhibits a non-zero mean output, even when perfectly horizontal. This is the *bias*, presented in the sensor modeling at chapter 6. This bias must be subtracted from the measures, as stated in the "calibration". The task was automated by a simple procedure that is by placing the value in the "digital filters" MATLAB file. This value is then sent to the device via bluetooth, together with the other filter's parameters and the calibration is made easy this way.

8.3.1.2 *Gyroscope noise and bias*

Concerning the gyroscope, the modeling procedure is similar to that of the accelerometer. The datasheet's section about gyro is that in figure 71.



3 Electrical Characteristics

3.1 Gyroscope Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)
Sensitivity Scale Factor Tolerance	25°C		±3		%
Sensitivity Scale Factor Variation Over Temperature	-40°C to +85°C		±4		%
Nonlinearity	Best fit straight line; 25°C		±0.1		%
Cross-Axis Sensitivity			±2		%
Initial ZRO Tolerance	25°C		±5		°/s
ZRO Variation Over Temperature	-40°C to +85°C		±30		°/s
Total RMS Noise	DLPFCFG=2 (92 Hz)		0.1		°/s-rms
Rate Noise Spectral Density			0.01		°/s/√Hz
Gyroscope Mechanical Frequencies		25	27	29	KHz
Low Pass Filter Response	Programmable Range	5		250	Hz
Gyroscope Startup Time	From Sleep mode		35		ms
Output Data Rate	Programmable, Normal mode	4		8000	Hz

Table 1 Gyroscope Specifications

Figure 71: Section of the datasheet of MPU9250, regarding the gyroscope.

The parameter regarding the noise is that called "Rate noise Spectral Density", expressed in $°/s/\sqrt{\text{Hz}}$. As for the accelerometer, the "Noise Power" value to put into the "Band Limited White Noise" is the square of the "Rate noise Spectral Density", with the difference that in the Simulink model, the gyroscope outputs $\mathtt{rad}/s$ instead of $°/s$, thus the final value inserted in the model is $(0.01 * 2 * \mathtt{pi}/180)^2$. Again, the validation by comparison is visible in figure 72.

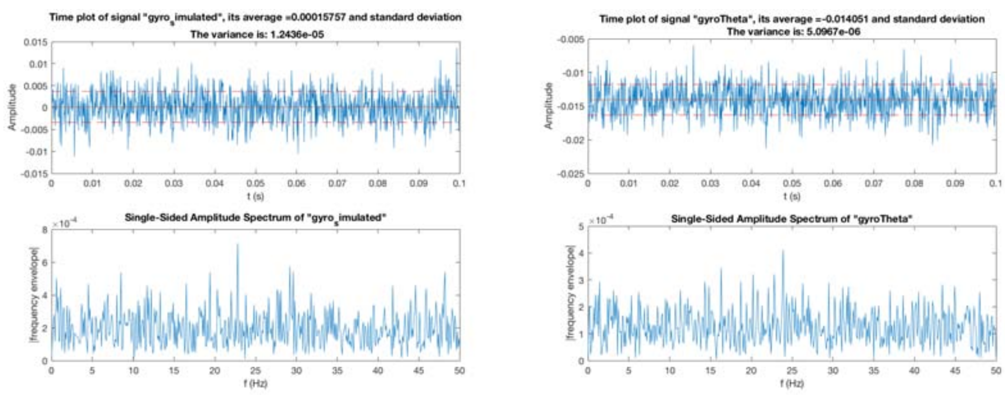


Figure 72: Noise at the output of the gyroscope. On the left, the simulated block, with the value found in the datasheet. On the right the real-one. The outputs are very close each other and the variance is also similar: the real-device variance is even smaller than that of the datasheet.

Even for the gyroscope, a good correspondence between the model and the true-life measures has been verified.

Concerning the *bias*, even for the gyroscope, the value of the average of the signal can be put into the "digital filters" file, so that the "parameters update" procedure provides the automatic bias removal (part of the IMU calibration task).

A note about this: in the case of the gyroscope, the bias has a higher impact on the measures, since the gyroscope's output is integrated during time, for retrieving the pitch angle (even if "fused" for with the accelerometer).

8.3.1.3 *IMU low-pass filter verification*

A further model verification was performed to proof the effect of the digital filter modeled inside the accelerometer and gyroscope. In figure 73 a comparison between the simulation-block and the true device behavior.

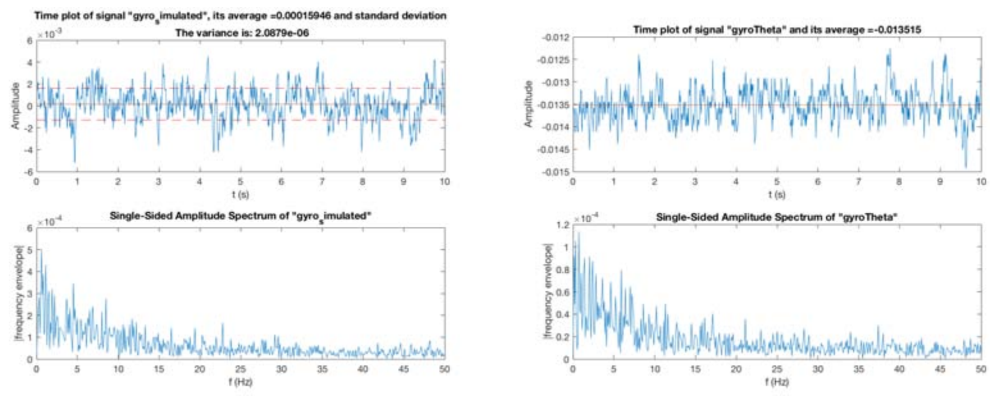


Figure 73: Noise at the output of the gyroscope, when filtered at 5Hz. On the left, the simulated block, on the right the real-one that has the embedded digital filter. The frequency behavior is very similar in the two situations.

The correct correspondence is useful because the simulation results is tight similar to the reality, making it possible to have affordable simulation and test benefits or drawbacks of filtering, directly by simulation, without the need to have a real device.

8.4 TUNING SENSOR FUSION ALGORITHM

After having validated the IMU sensors models and their calibration, this section discusses about the task of calibration the sensor fusion algorithm.

The main tasks about the IMU filters tuning are:

1. find the best coefficients for the complementary filter
2. verify the opportunity of linear-acceleration subtraction

Here the results of the experiments.

8.4.1 *Verification of linear acceleration subtraction*

In order to verify the opportunity for subtraction of linear acceleration from the accelerometer reading, it was performed an experiment of manual excitation of the inverse pendulum, making the self-balancing vehicle running back and forward, with simultaneous base inclination. In this experiment the base inclination changed in a sinusoidal-like way. The test was the verification of angle read from the accelerometer and the gyroscope output. Since the pitch angle and its rate's phases must differ of $90°$, the test aimed to verify if this is true or not, and if the acceleration subtraction has an influence.

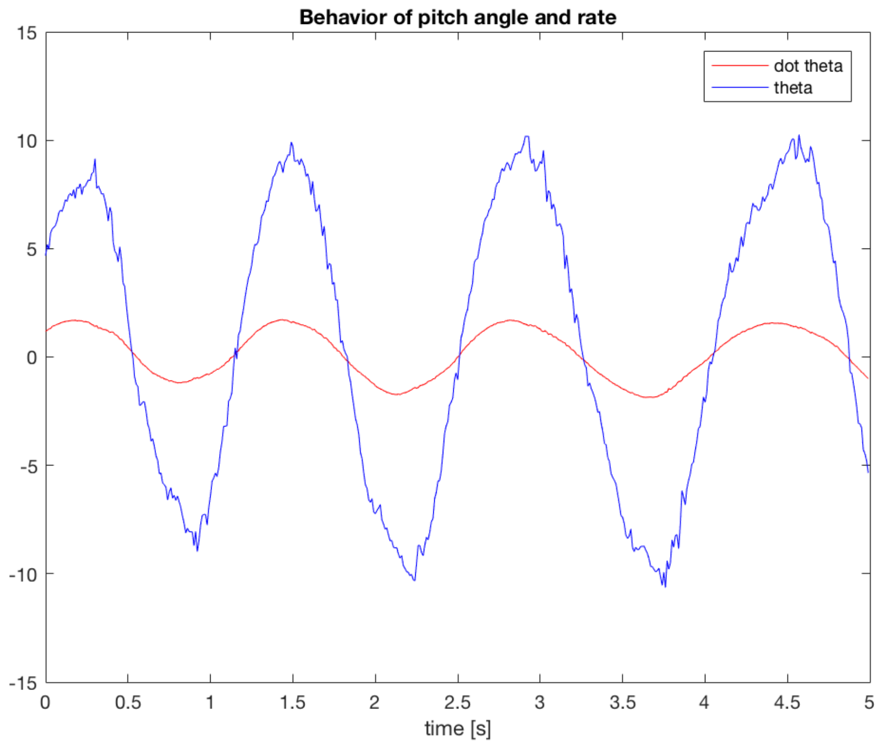


Figure 74: Investigation in linear acceleration subtraction: signals without linear acceleration subtraction.

By observing figure 74, when "fastly" perturbing the self-balancing device, it moves back and forward but the linear acceleration adds to the gyroscope reading and makes the pitch angle and its rate to differ by almost $0°$: it is considered big error.

After subtracting the linear acceleration from the accelerometer's x coordinete readings, the situation changes, like visible in figure 75.

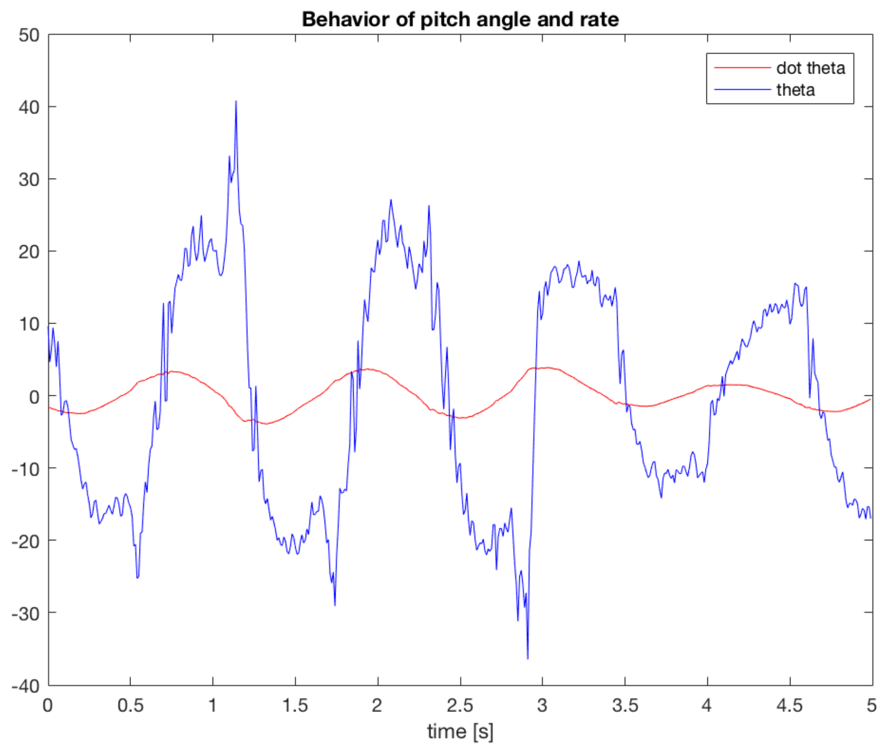


Figure 75: Investigation in linear acceleration subtraction: signals with linear acceleration subtraction.

This experiment confirms the opportunity for the subtraction of liner acceleration from the accelerometer reading, making it more affordable and coherent with the gyroscope.

The only drawback of this is the presence of spurious "spikes" in the acceleration readings that compromises the reading quality: this is not a big problem since the accelerometer reading is meaningful (see complementary filter) only in long-period (low frequencies), instead the spikes are fast and don't affect mush the complementary filter result.

8.4.2 *Finding the best coefficients for complementary filter*

The choice of the coefficients for α and β determines a different behavior of the device. As a "quantitative" consideration, we can tell that they they are "the indicators of what we trust more" between the output of the two IMU sensors. The high value of α indicates that "we trust more the gyroscope", a low value instead indicates that "we trust more the accelerometer", but it is not all here.

The values of α and β determines an "equivalent bandwidth" of work, for the sensors. In fact, since the gyroscope is affordable for "fast rotations", it is good for the management of "high frequencies". As opposite, the accelerometer is good for the management of long-term pitch angle reading (since differently from the gyroscope, it is bias-free). Even with this, it must be considered that a too-high value of α might determine a long time before any gyroscope bias error is canceled, as opposite a low-value might result in a loosely-stable equilibrium controller, since the accelerometer is very noisy.

The tuning activity is a trade-off between system stability and measure precision. In this case there were performed several experiments, simply by testing the reaction (angle and horizontal position deviation) of the TWSBT to external disturbances (kicks) with different values of α and "quantitatively" verify the effect on the equilibrium performances:

- $\alpha = 0.95$
- $\alpha = 0.99$
- $\alpha = 0.999$

Also the simulations have demonstrated the different behavior of the system with different settings of α. Their results are reported in figure 76.

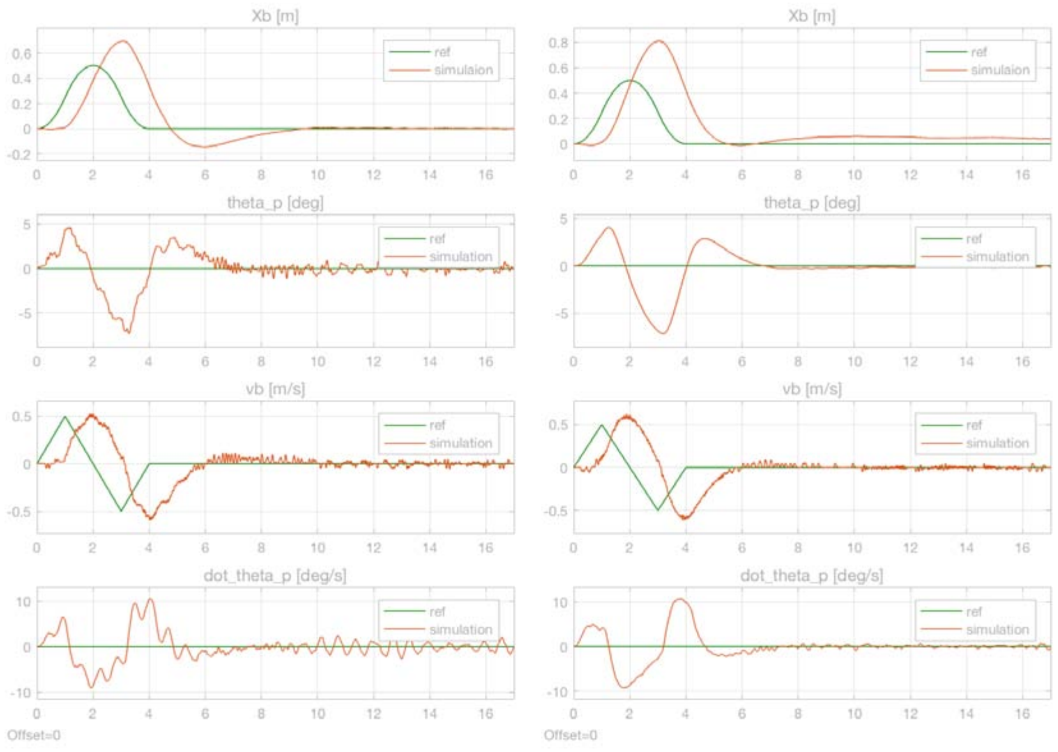


Figure 76: Simulation with different values of complementary filter coefficients (left $\alpha = 0.95$, right $\alpha = 0.999$).

As noticeable in figure 76, higher values of α guarantees less noise in the state variables and less elongations in the transients. Another key-consideration is the presence of strong oscillations in the device with $\alpha = 0.95$, confirming a loosely stability. This is also confirmed by the real-device, where a value of $\alpha < 0.95$ determines an unstable device.

At the end of the tests, it was chosen to adopt $\alpha = 0.999$, since the device has an intrinsic "slow dynamic", and the accelerometer output must be considered only for very-slow frequencies. The value chosen also guaranteed less deviation of the vehicle in case of external perturbations. It has also demonstrated to be a good choice to be employed for the transport of the rider, since it gives the (subjective) sensation of a "stable base", much more than with lower α values.

symbol	meaning	value	unit
$H = 2 * L$	Total height of the cylinder	1	[m]
d	Diameter of the cylinder	0.25	[m]
m_P	Mass of the cylinder	30	[kg]

Table 9: Parameters of the wood cylinder of figure.

8.5 VALIDATION OF THE FIRST MODEL

In chapter 7 it was illustrated the automated script that makes possible the comparison between the model simulation and the real-device behavior. In this section the results of this comparison, for different tasks of the control.

In order to make the real-device as similar as possible to the real one, a custom wood cylinder was built. Having a well-defined solid geometry, the wood cylinder is easy to model and its data were placed into the MATLAB script's data. The wood-cylinder, visible in figure 77 has the parameters described in table 9.

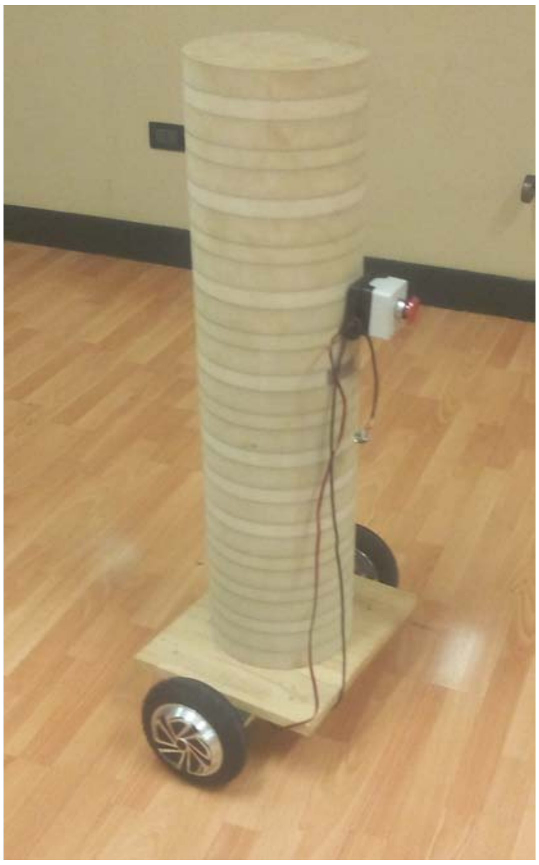


Figure 77: Wood cylinder anchored to the self-balancing transporter. The simple and regular geometry and the knowledge of weight and dimensions, makes it easy to model it into the simulator.

8.5.1 *Steady state*

Here a comparison of the steady-state behavior of the device. During the first experiment it is possible to notice that differently from the theoretical model, the "experimental" (which means the real-device) behavior can't stand in a fixed position, but has a residual oscillation due to a *limit cycle*. In figure 78 the focus on this behavior.

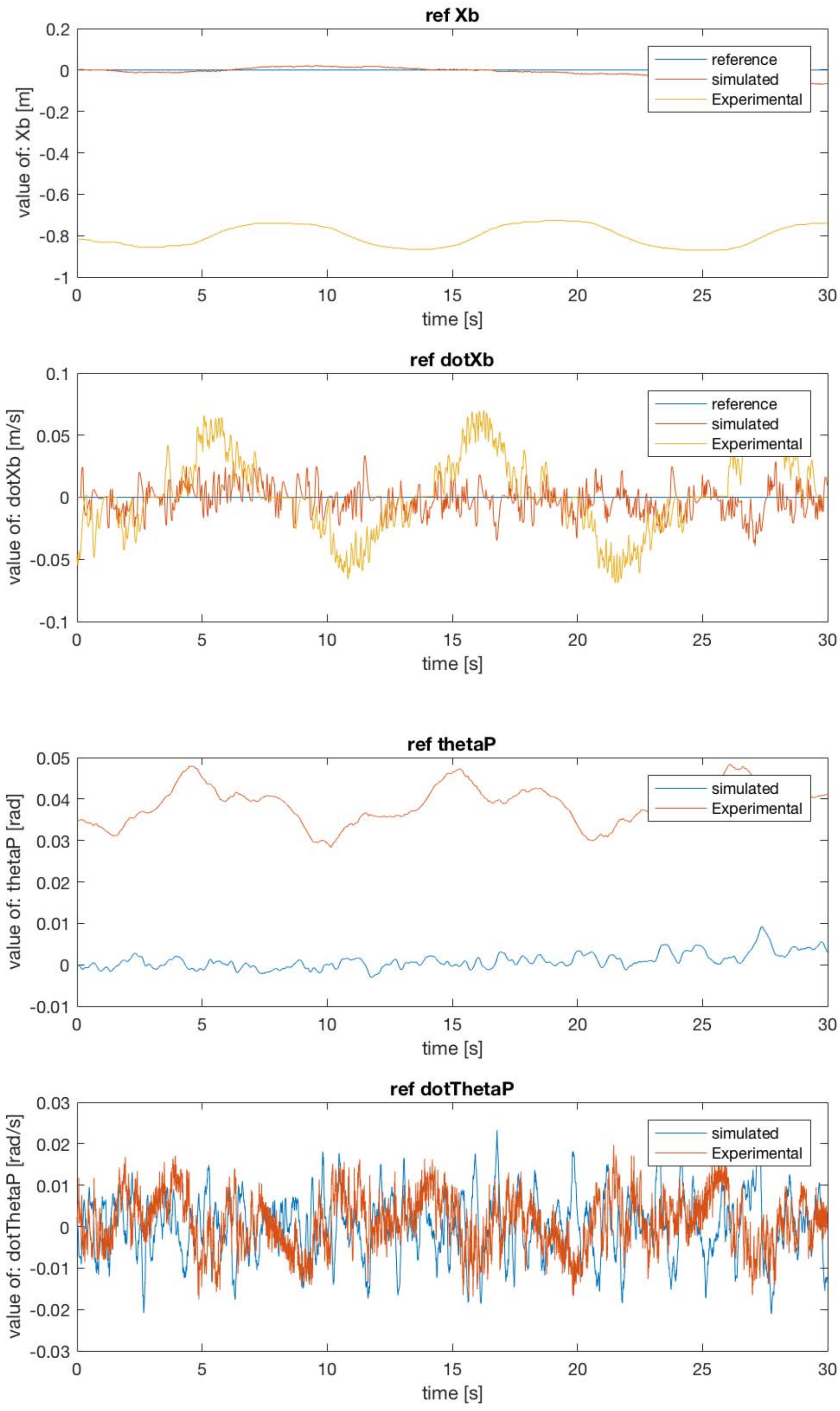


Figure 78: Steady state simulation: here the "real-device" shows a different behavior in both horizontal position and velocity, and in pitch angle: it is visible an oscillation around a value, that denotes the seamlessly presence of an unconsidered limit-cycle. Note that even if the reference is 0, the equilibrium point is not in $\theta_P = 0$ and $X_b = 0$, probably because of an imperfection in the angle reading, from the IMU.

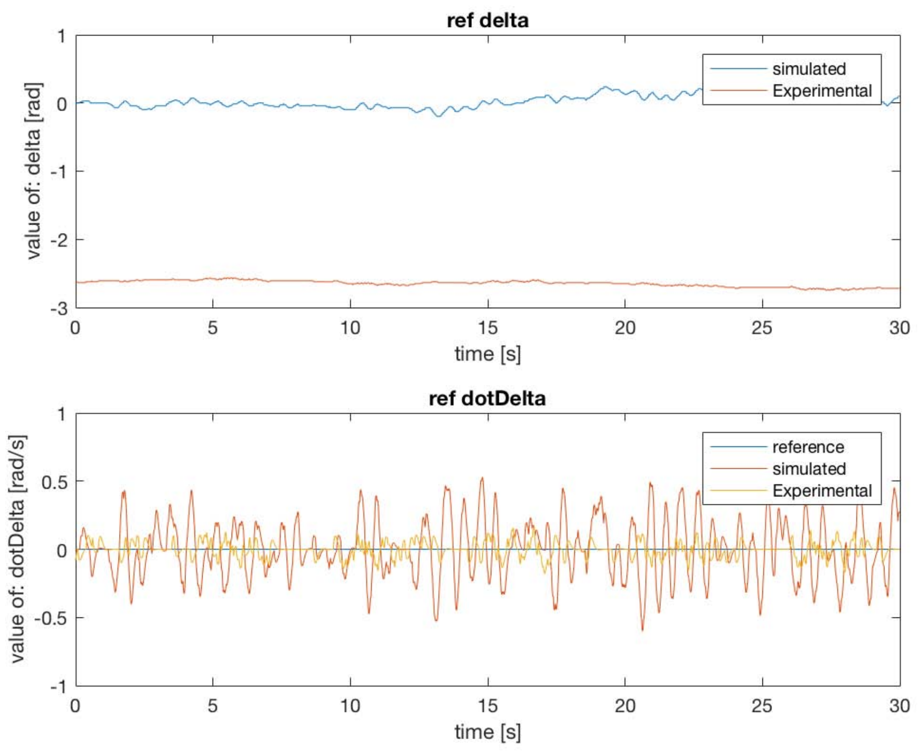


Figure 79: In the yaw angle the real-device behavior is very similar to the simulation, except by the presence of an offset in the yaw angle: it is due to the fact that here the controller is a PI and it doesn't care about yaw, but only about yaw-rate.

The presence of the limit-cycle is probably due to a Coulomb-friction not considered in the model, or even some backlash in the assembly of the cylinder with the base.

The entity of the limit-cycle changes, when changing the weights of the LQR equilibrium controller: when decreasing the weight related to the Xb control, as expected, the limit-cycle increases in amplitude. The same for the weight of θ_P angle.

No strategies have been adopted to try to eliminate the phenomena, since it is out of the scope of this work.

8.5.2 *Reference-following sequence*

In this subsection a verification of the matching between the simulation results and the real-device behavior. The scope of this verification is not to find a "good reference-follower" but even if the device dynamics are slower than the reference, the scope is to verify that the same behavior is present in the true device. In figure 81

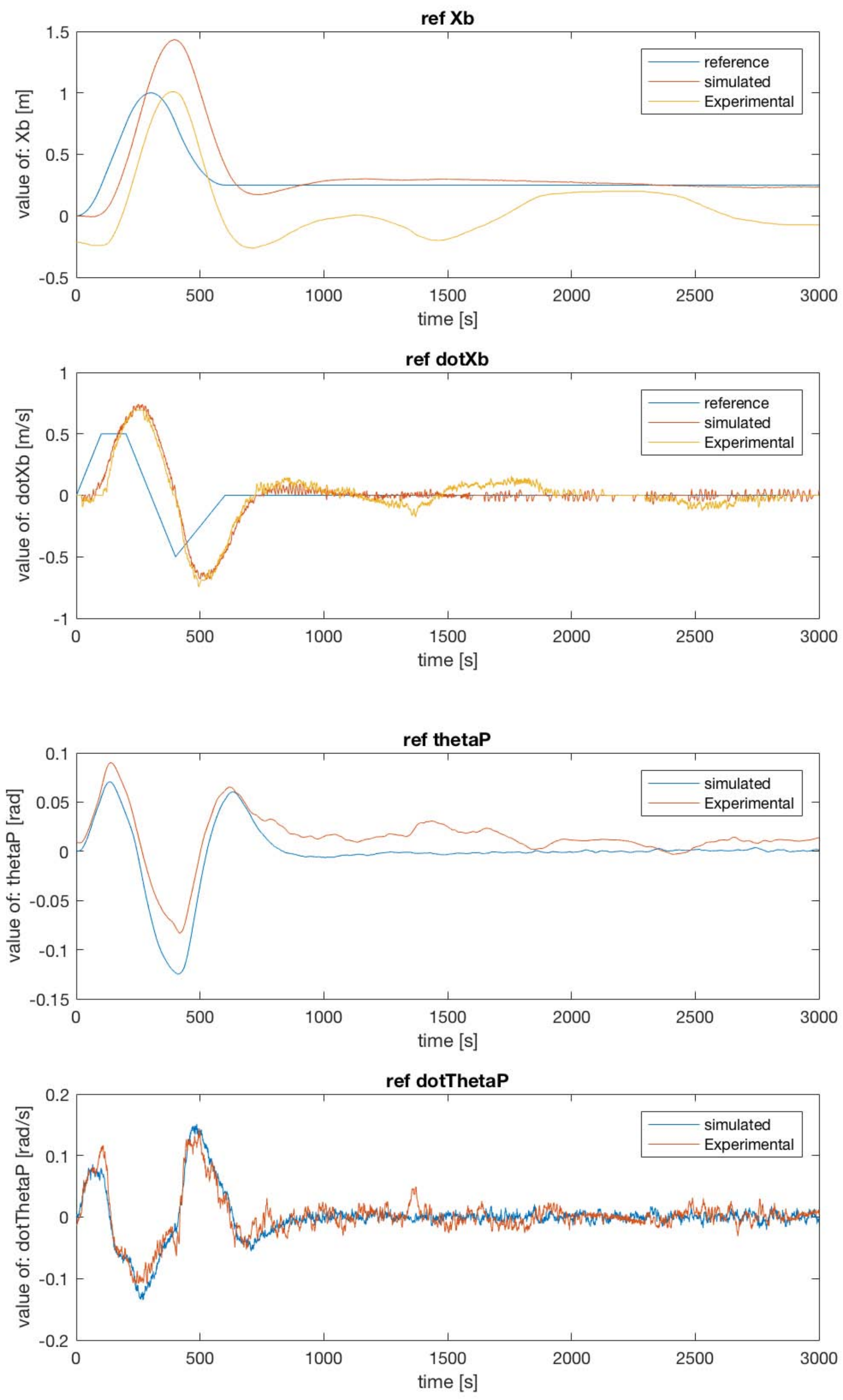


Figure 80: Reference-following simulation: during velocity ramps there is a good matching between the simulated and the real-device behavior, while when the reference turns to 0, it is visible the limit-cycle previously mentioned. Note that like the "steady" test, when the reference is 0, the equilibrium point is not in $\theta_P = 0$ and $X_b = 0$. This means that during the following rotation, caused by the following of yaw-angle reference, a deviation in the position and velocity happens.

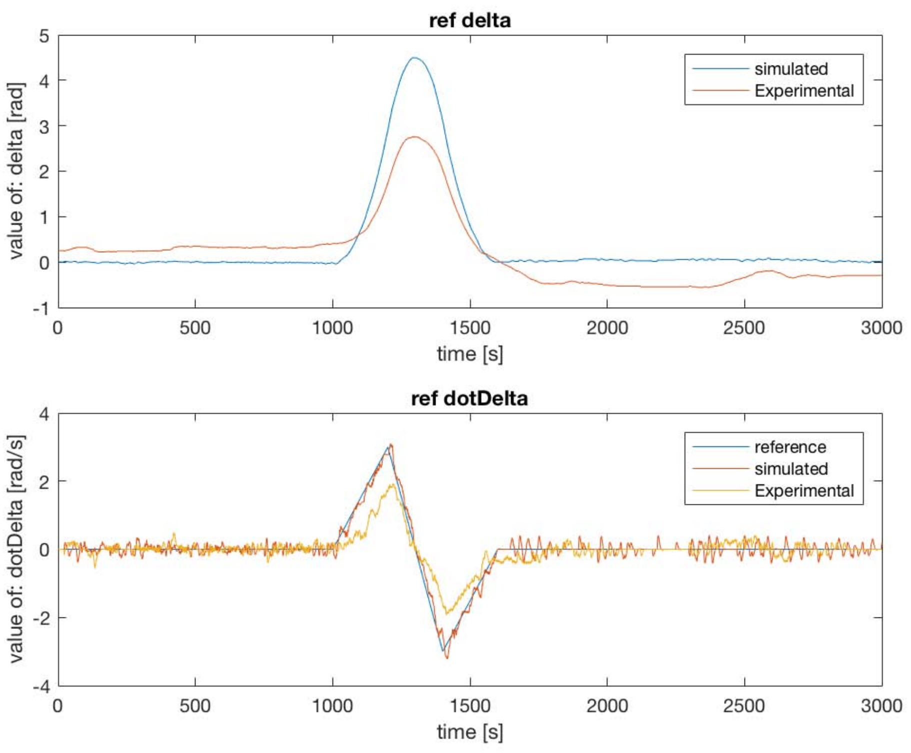


Figure 81: In the yaw angle the real-device behavior is different because the model has a low-value in the friction coefficient of the wheels.

8.5.2.1 *Parameters adjust*

Since the simulation of yaw angle was not good, it was repeated the test increasing the ψ friction coefficient in the model. Figure 82 shows the results when setting the coefficient to $\psi = 0.1$ (instead of the previous $\psi = 0.0005$).

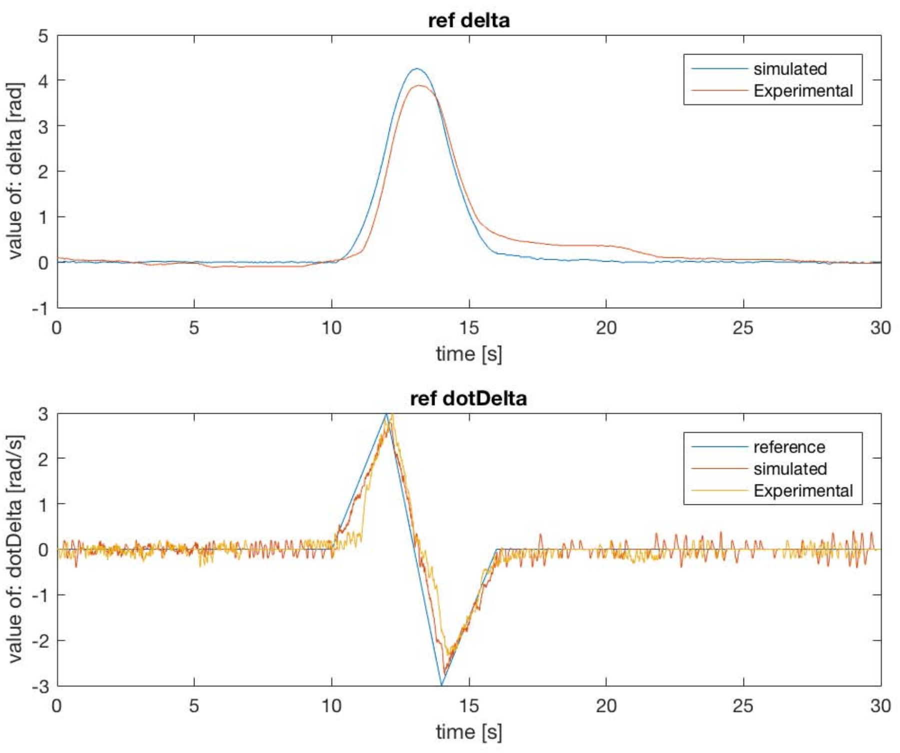


Figure 82: Reference-following of yaw-angle: by setting the wheel's friction coefficient to $\psi = 0.1$, it is visible a better matching between the simulation and the real-device experiment.

In figure 82 it is visible a better matching between the simulation and the real-device experiment. The residual yaw-angle-error is due to the fact that the controller is a PI and doesn't care about position tracking: only cares about yaw-rate error tracking. In the following sections a discussion with LQR control even in the steering.

8.6 STEERING MODEL AND CONTROLLER TUNING

In figure 82 it was shown the behavior of the yaw-angle reference follower when the steering controller was a PI with $K_P = 2$ and $K_I = 1.6878$. It was good, but as mentioned, the controller doesn't track regime-yaw errors. In figure 84 the results of a simulation using the LQR for the steering subsystem.

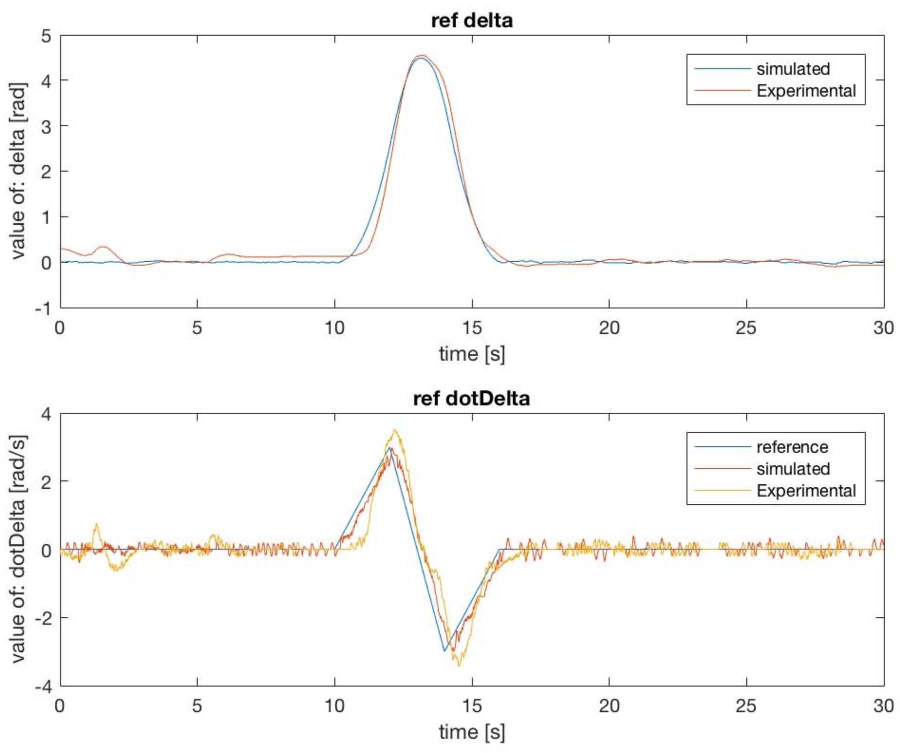


Figure 83: Reference-following of yaw-angle: the LQR controller tracks not only the yaw-rate, but also the yaw-angle: this is considered a great improvement in the overall performances.

Using an LQR controller even for the steering subsystem makes it possible to have a better tracking of the steering angle. This is not immediately visible by the figures, but results in a more-stable device that even if encountering terrain discontinuities, takes back the steering to the correct angle. This was not achieved with the previous PI controller.

This is why the LQR was the preferred controller and its parameters are $Q = diag([1, 0.1]), R = 0.1$;

8.7 EQUILIBRIUM MODEL AND CONTROLLER TUNING

During all the experiments there was not a real need of performances, since the device to control was a simple wood-cylinder. The only need was that the device turns back to the original position (in order to verify if it attempts to it). The final choice for the coefficients of Q and R matrices was: $Q = diag([10, 3, 0.1, 0.1])$ and $R = 0.1$, which means an attention to the horizontal displacement and to the vertical position.

A non-null value in the velocities is to "mitigate" the behavior against disturbances: since the regime reference is 0 for all the state-variable-references, the device attempts to "gracefully reduce the velocities". The final result is visible in figure 84.

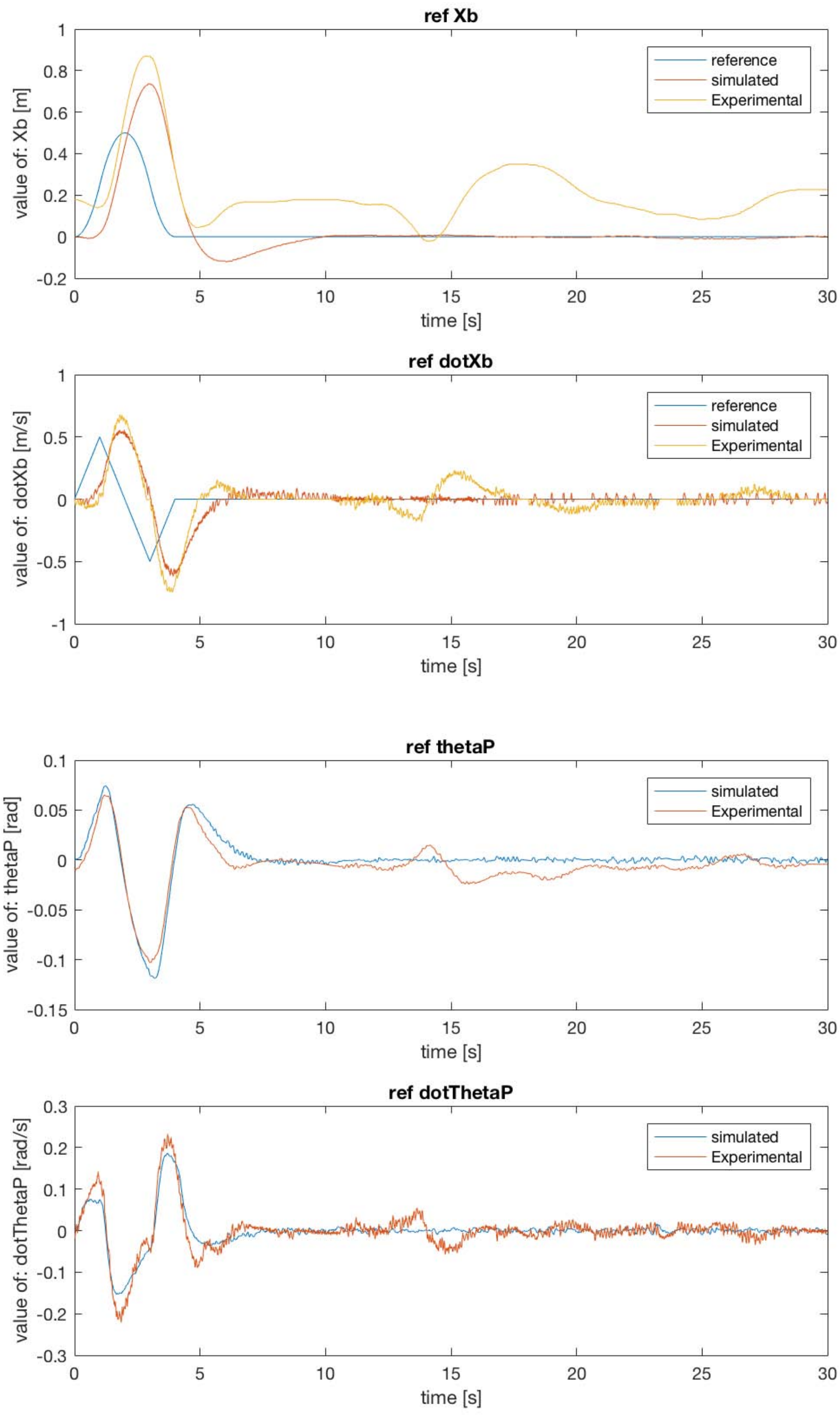


Figure 84: Horizontal displacement and angle tracking with definitive weights of $Q = \mathit{diag}([10, 3, 1, 0.1])$ and $R = 0.1$.

8.8 TUNING AND VALIDATION OF THE EVOLVED (RIDER'S) MODEL

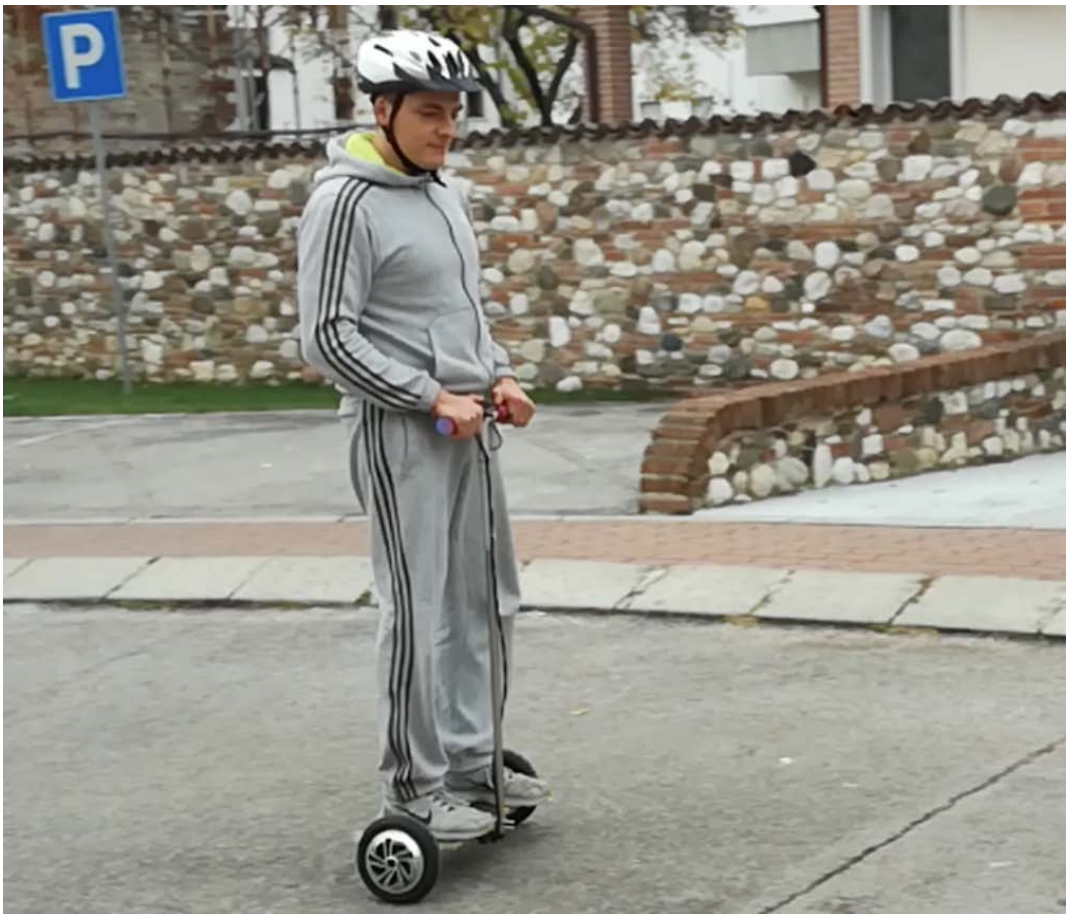


Figure 85: Last tests with a human rider, driving the TWSBT.

This section describes the process of validation of the model proposed in chapter 4, where the self-balancing vehicle was enriched by the presence of a rider that can lean backward and forward, achieving the movement of the vehicle, as desired. This is in fact what happens in the real-device: the rider does exactly this.

An important thing to keep in mind is that (as already stated) the system proposed in chapter 4 is not fully observable thus the pitch angle of the rider cannot be calculated with the only knowledge of the base's pitch angle.
The positive fact is that the non-observable poles of the system are stable thus the overall system can be stabilized (as expected).

The resulting thing is that since the only known pitch coordinates are that of the base, the equilibrium controller will remain the same as that proposed for the model with the solid cylinder linked with the base, as visible in figure 86.

symbol	meaning	value	unit
$H = 2 * L$	Total height of the rider	1.8	$[m]$
d	Diameter of the equivalent cylinder	0.5	$[m]$
m_P	Mass of the rider	80	$[kg]$
k_s	Elastic constant of the ankle's spring	850	$[Nm/rad]$
c_s	Viscous friction coefficient of the ankle damper	350	$[Nm/(rad/s)]$

Table 10: Parameters of the rider governing the TWSBT.

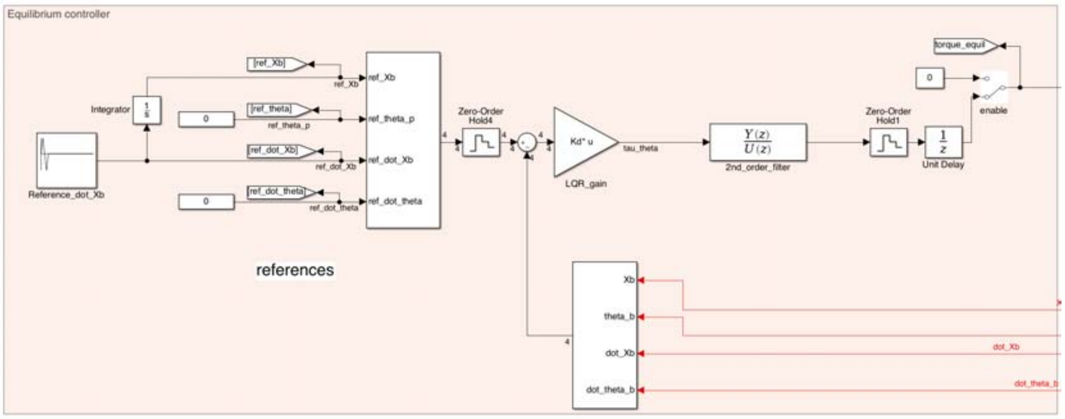


Figure 86: Controller for the model of chapter 4, with rider's pitch angle different from that of the base.

The tuning consists in finding the adequate controller's parameters for the equilibrium and steering subsystems.

A first attempt was made by placing the parameters of a man (like me) into the model and test the effects. The parameters are in table 10.

Putting these parameters into the model and and letting MATLAB solve the DRE for the calculation of the LQR gains coefficients, results in the system behavior of figure 87.

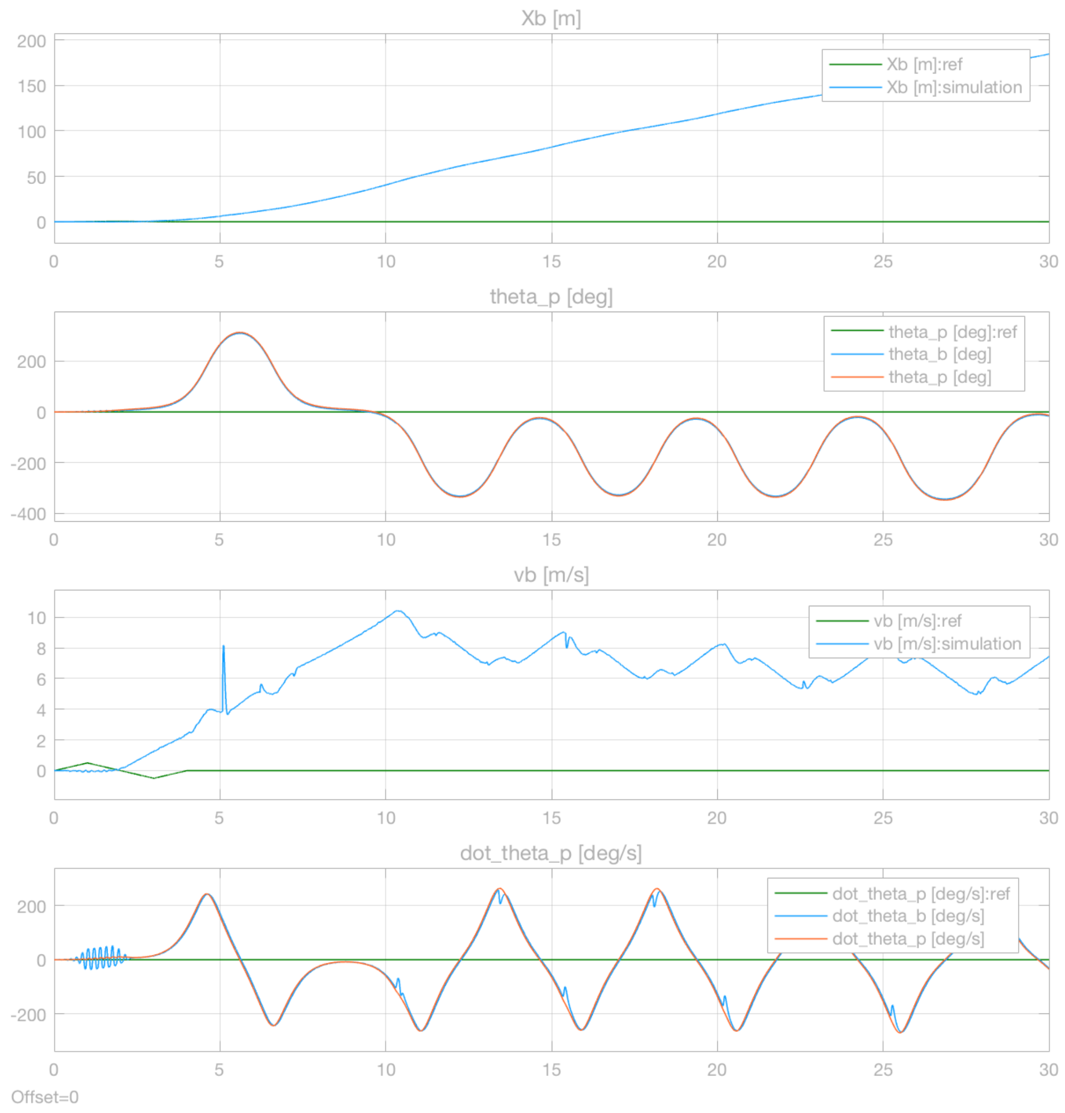


Figure 87: First simulation of rider with controller tuned with parameters of table 10: the controller is unstable.

As noticeable in figure 87 the controller attempts to take the rider vertical, but suddenly a strong vibration in the pitch angle makes it impossible and the controller diverges.

This behavior was verified also in the real-device: a strong base vibration made it impossible to ride it. It is a negative thing since the controller can't achieve stable equilibrium, but is also positive since the simulated behavior is similar to the real one.

In the following subsection the actions performed to achieve the self-balancing feature.

8.8.1 *Controller tuning, for the rider*

The actions taken to try to stabilize are a first strategy to make the vehicle suitable for a rider's transport. The result obtained is not fully satisfactory, but for a matter of time it was not possible to perform further actions in that sense.

In brief, the actions that solved the problem of equilibrium for a rider, were two:

1. *Band-limitation of torque signal*: having verified the presence of a high-frequency vibration in the simulation, it was introduced a second-order low-pass filter in the torque actuators. This made possible the control of phenomena at frequencies similar to that of the system dynamics' equilibrium, but avoids the presence of high-frequency vibrations in the device's base, especially during the rider's step-up and step-down. The filter is a second-order LPF at 7Hz over-damped with $\xi = 3$(since it further reduces the presence of vibrations).

2. *Reduction of rider's height*: the controller was calculated by assuming a reduced height of the rider. The simulations demonstrated that the stability is achieved when the height is reduced to about $1/9$ of the original height. In the case of the rider with parameters of table 10, the height was reduced to $H = 0.2m$. All other parameters remained unvaried.

Thanks to those modifications, the device became "ride-able", with a sufficient perceived stability.

8.8.2 *Tuning the parameters of "Q" matrix*

This activity was to define good settings for the Q matrix,in order to give a good subjective riding experience. In this case, the scope is to drive the vehicle by leaning forward or back and not to "follow a reference", then the weights of Q were very different from that of the solid-linked wood cylinder. In particular:

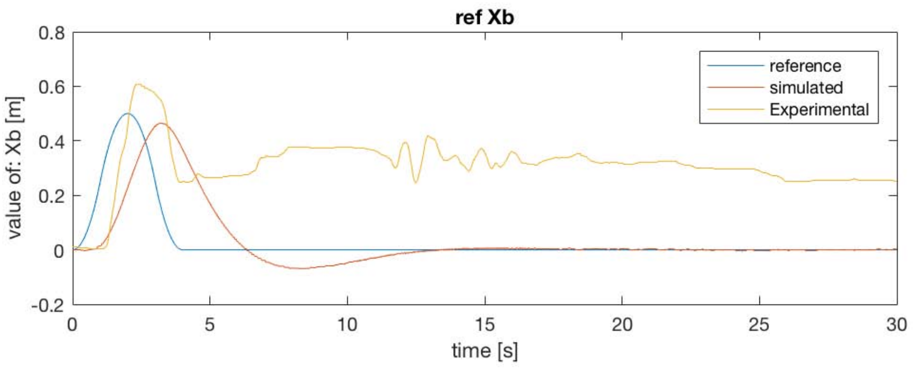
- The weight associated to X_b must be diverse from 0, since if 0 the controller will become unstable, but very low since the scope is to maintain the rider vertical, without care of its Horizontal position.
- The weight associated to θ_P must be high enough to give the rider the sensation of a "solid base", but low enough to avoid the growth of undesired "nervous-like" behavior
- The weights associated to the velocities are low, but not null because a null value contributes to a "too-nervous" behavior, instead a small value has a positive contribute in reducing some induced self-oscillations

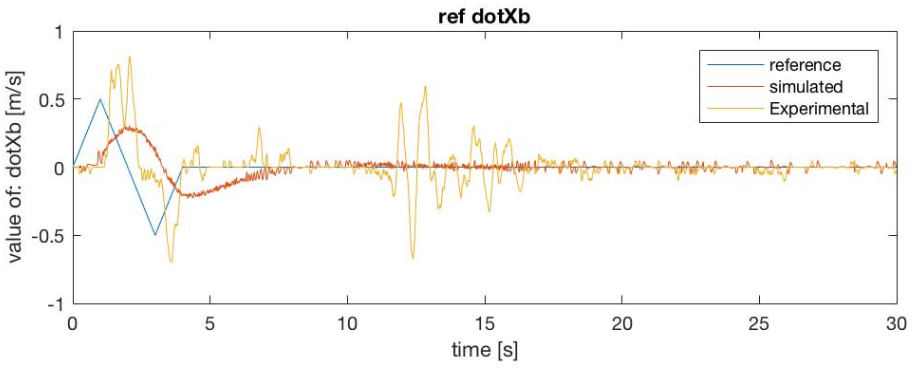
The final weights chosen, after several experiments are that who results in a better subjective riding experience:

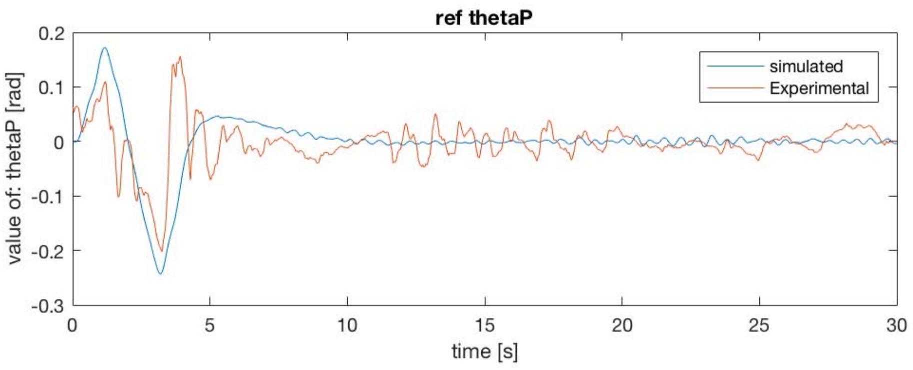
$$Q = diag([1e - 10; 5; 0.1; 0.1]); \tag{110}$$

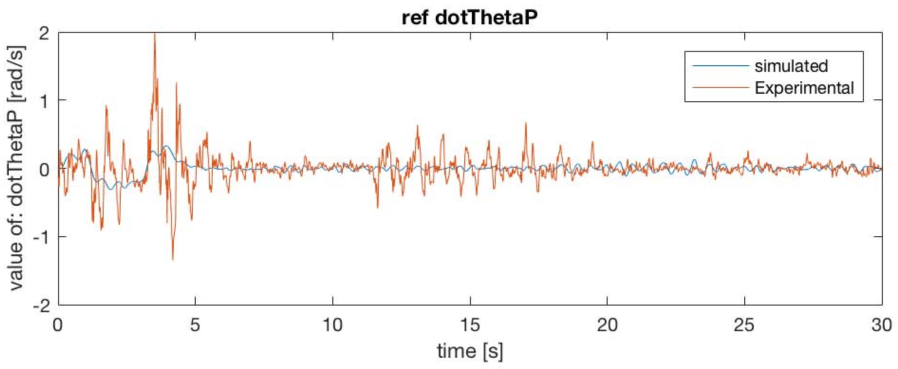
8.8.2.1 *Validation of rider's control*

Even if the control in presence of a rider has different Q-settings against those used for the wood-cylinder, it was replicated the reference-following experiments in presence of a rider, in order to validate the rider's model presented in chapter 4. The results are visible in figure 88.


ref Xb
value of: Xb [m]
time [s]
reference
simulated
Experimental


ref dotXb
value of: dotXb [m/s]
time [s]
reference
simulated
Experimental


ref thetaP
value of: thetaP [rad]
time [s]
simulated
Experimental


ref dotThetaP
value of: dotThetaP [rad/s]
time [s]
simulated
Experimental

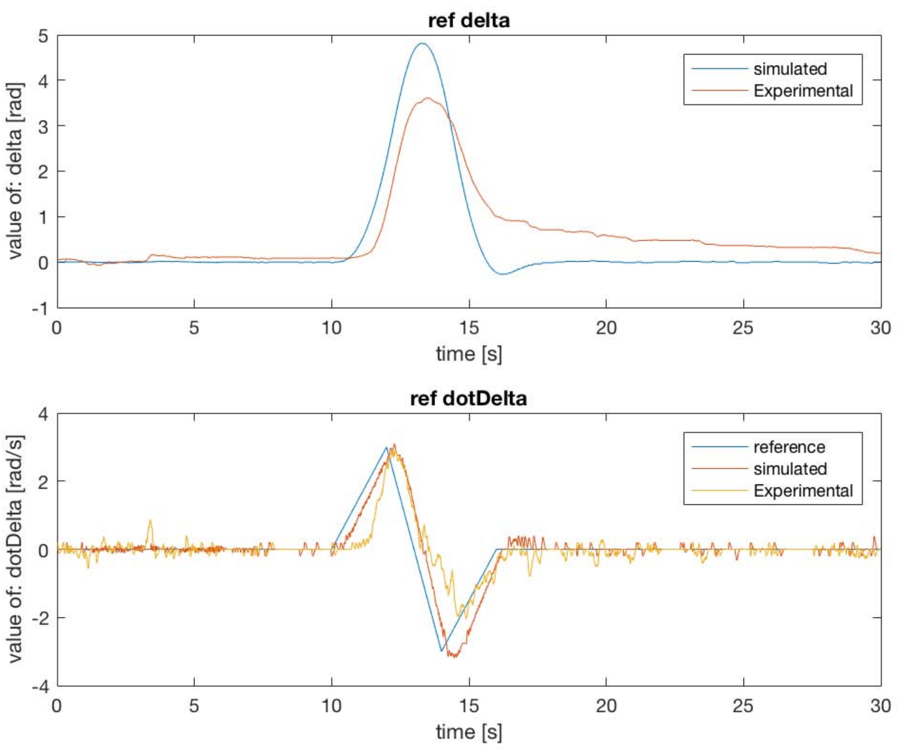


Figure 88: Comparison of the results of simulation and real-device, while it controls the rider.

Figure 88 denotes that the TWSBT is capable of taking vertical the rider while it follows the reference, but the results of the experiment are quite different from the data of the simulation.

This is almost obvious because of the facts that:

- the controller must be designed for the model of chapter 3, and then applied for the evolved model, thus it is not really "optimum"
- the controller used was designed by thinking a rider with very different geometry: if using the true geometry the system is not stable
- in the simulation it was not modeled the possible reactions of the rider, that in the experiment instead were present

Due to a matter of time the analysis have stopped here and no other experiments were carried. Further analysis might take substantial improvement in the model, thus the simulation might be more affordable and useful for control improvement.

8.9 CONCLUSIONS

In this chapter there were discussed the tasks related to the parameters identification, model validation and the controller tuning. The experiments has confirmed that concerning the case of a solid body standing over the vehicle there was a good matching between the simulation and the experimental results. This confirms the good modeling and the correct parameters identification, thus the affordability of the simulation environment.

Concerning the case of a rider, there were poor performances in simulation, demonstrating that the model has a good dynamic description, but lacks of other phenomena not already investigated (such as human reaction models) that might be studied in successive investigations.

The positive thing about the rider configuration is that a first set of configurations made it possible to have a good riding experience, that is (after all) one of the most important goals for that kind of vehicle.

CONCLUSIONS

The scope of this work was to design a controller capable to drive a vehicle based on the (inherently unstable) *inverse pendulum* system.

This result was achieved by following a rigorous engineering process that consisted on:

- collection of the product requirements
- build-up of the system model, by solving the dynamic equations
- build-up of the simulation environment with introduction of system non-idealities, for better simulation results
- parameters identification of the real device
- control design with stability verification and simulation
- practical implementation into a distributed embedded system
- verification of simulation environment by comparison with real-device experiments

The result at the end of the work is a stable device, capable of carrying both solid-attached loads and persons, with a good riding experience.

Also important is a high-quality simulation environment, that has good simulation-experiment matching when carrying solid bodies linked with the TWSBT, useful for further investigation, control tuning or control model evolution.

The limits of the system are the starting-points for possible future works, such as:

- elimination of limit-cycle found in the device
- further improvements in tuning of Kalman filter applied to the IMU
- further tuning of rider's model and controller, with better matching between simulation and real-device experiments
- design of an adaptive or self-tuning controller, for better rider-experience, according to the rider's weight and geometry

The work has confirmed to be very interesting and challenging. It demonstrated to be "complete" both for the theoretical content and the practical issues that needed to be solved. At the end of the work, an excellent simulator and data-collection system was built: this work will be soon published as an open-source project, for the benefit of other researchers.

APPENDIX

APPENDIX-A: LAGRANGIAN DERIVATION STEP-BY-STEP

A

Even having the possibility to derive the expression of motion equations with automated tools, such as Wolfram Mathematica, in this appendix, the mathematical steps of the derivation of the motion equations. The final equations were verified also with the mentioned software.

A.1 LAGRANGIAN: DERIVATION FOR THE FIRST MODEL

Here the extended expression of the Lagrangian for the system

$$\begin{aligned} L &= T - U = \\ &= \frac{1}{2}m_P\left[r^2\frac{\dot{\alpha}^2}{4} + r^2\frac{\dot{\beta}^2}{4} + r^2\frac{\dot{\alpha}\dot{\beta}}{2} + L^2\dot{\theta}_P^2 + L\dot{\theta}_P r(\dot{\alpha}+\dot{\beta})\cos\theta_P\right] + \\ &\frac{1}{2}J_{\theta P}\dot{\theta}_P^2 + \frac{1}{2}\left(J_{\delta P} + m_P L^2\sin^2\theta_P\right)\left[r^2\frac{\dot{\alpha}^2}{D^2} + r^2\frac{\dot{\beta}^2}{D^2} - 2r^2\frac{\dot{\alpha}\dot{\beta}}{D^2}\right] + \\ &\frac{1}{2}(m_w r^2 + J_w)(\dot{\alpha}^2 + \dot{\beta}^2 + 2\dot{\alpha}\dot{\beta}) + \\ &\frac{1}{2}\frac{J_m}{\rho^2}(\dot{\alpha}^2 + \dot{\beta}^2 + 2\dot{\theta}_P^2 - 2\dot{\alpha}\dot{\theta}_P - 2\dot{\beta}\dot{\theta}_P) + \\ &-m_P g L\cos\theta_P \end{aligned} \tag{111}$$

Having this, the Lagrange equations becomes:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}}\right) - \frac{\partial L}{\partial \alpha} &= C_L \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}}\right) - \frac{\partial L}{\partial \beta} &= C_R \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_P}\right) - \frac{\partial L}{\partial \theta_P} &= -(C_L + C_R) \end{aligned} \tag{112}$$

The right-terms of the Lagrange equations represents the active contributes to the dynamics. The first and second equations' terms are quite obvious, while for the third equation, the active contribute is the sum of the torques of the two motors, as easily noticeable observing figure 13.

Executing the partial derivatives just presented, the equations' becomes this non-linear system:

$$
\begin{aligned}
-\frac{\partial L}{\partial \alpha} &= 0\\
-\frac{\partial L}{\partial \beta} &= 0\\
-\frac{\partial L}{\partial \theta_P} &= m_P L\dot{\theta}_P r\frac{\dot{\alpha}+\dot{\beta}}{2}\sin\theta_P - m_P g L\sin\theta_P +\\
&\quad -m_P L^2\sin\theta_P\cos\theta_P\left[r^2\frac{\dot{\alpha}^2}{D^2}+r^2\frac{\dot{\beta}^2}{D^2}-2r^2\frac{\dot{\alpha}\dot{\beta}}{D^2}\right]\\
\\
\frac{\partial L}{\partial \dot{\alpha}} &= \frac{1}{2}m_P\left[r^2\frac{\dot{\alpha}}{2}+r^2\frac{\dot{\beta}}{2}+L\dot{\theta}_P r\cos\theta_P\right]+\\
&\quad \left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\frac{\dot{\alpha}}{D^2}-r^2\frac{\dot{\beta}}{D^2}\right]+\\
&\quad (m_w r^2+J_w)[\dot{\alpha}+\dot{\beta}]+\frac{J_m}{\rho^2}(\dot{\alpha}-\dot{\theta}_P)\\
\frac{\partial L}{\partial \dot{\beta}} &= \frac{1}{2}m_P\left[r^2\frac{\dot{\alpha}}{2}+r^2\frac{\dot{\beta}}{2}+L\dot{\theta}_P r\cos\theta_P\right]+\\
&\quad \left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\frac{\dot{\beta}}{D^2}-r^2\frac{\dot{\alpha}}{D^2}\right]+\\
&\quad (m_w r^2+J_w)[\dot{\alpha}+\dot{\beta}]+\frac{J_m}{\rho^2}(\dot{\beta}-\dot{\theta}_P)\\
\frac{\partial L}{\partial \dot{\theta}_P} &= m_P\left[L^2\dot{\theta}_P+Lr\frac{\dot{\alpha}+\dot{\beta}}{2}\cos\theta_P\right]+J_{\theta P}\dot{\theta}_P+\frac{J_m}{\rho^2}(2\dot{\theta}_P-\dot{\alpha}-\dot{\beta})\\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}}\right) &= \frac{1}{2}m_P\left[r^2\frac{\ddot{\alpha}}{2}+r^2\frac{\ddot{\beta}}{2}+L\ddot{\theta}_P r\cos\theta_P-L\dot{\theta}_P^2 r\sin\theta_P\right]+\\
&\quad \left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\frac{\ddot{\alpha}}{D^2}-r^2\frac{\ddot{\beta}}{D^2}\right]+\\
&\quad 2m_P L^2\dot{\theta}_P\sin\theta_P\cos\theta_P\left[r^2\frac{\dot{\alpha}}{D^2}-r^2\frac{\dot{\beta}}{D^2}\right]+\\
&\quad (m_w r^2+J_w)[\ddot{\alpha}+\ddot{\beta}]+\frac{J_m}{\rho^2}(\ddot{\alpha}-\ddot{\theta}_P)\\
\\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\beta}}\right) &= \frac{1}{2}m_P\left[r^2\frac{\ddot{\alpha}}{2}+r^2\frac{\ddot{\beta}}{2}+L\ddot{\theta}_P r\cos\theta_P-L\dot{\theta}_P^2 r\sin\theta_P\right]+\\
&\quad \left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\frac{\ddot{\beta}}{D^2}-r^2\frac{\ddot{\alpha}}{D^2}\right]+\\
&\quad 2m_P L^2\dot{\theta}_P\sin\theta_P\cos\theta_P\left[r^2\frac{\dot{\beta}}{D^2}-r^2\frac{\dot{\alpha}}{D^2}\right]+\\
&\quad (m_w r^2+J_w)[\ddot{\alpha}+\ddot{\beta}]+\frac{J_m}{\rho^2}(\ddot{\beta}-\ddot{\theta}_P)\\
\\
\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_P}\right) &= m_P\left[L^2\ddot{\theta}_P+Lr\frac{\ddot{\alpha}+\ddot{\beta}}{2}\cos\theta_P-Lr\frac{\dot{\alpha}+\dot{\beta}}{2}\dot{\theta}_P\sin\theta_P\right]+\\
&\quad J_{\theta P}\ddot{\theta}_P+\frac{J_m}{\rho^2}(2\ddot{\theta}_P-\ddot{\alpha}-\ddot{\beta})
\end{aligned}
\tag{113}
$$

Which permits to write the complete form of the equations (112) this way:

$$\begin{aligned}
\tfrac{1}{2}m_P\left[r^2\tfrac{\ddot{\alpha}}{2}+r^2\tfrac{\ddot{\beta}}{2}+L\ddot{\theta}_P r\cos\theta_P-L\dot{\theta}_P^2 r\sin\theta_P\right]+&\\
\left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\tfrac{\ddot{\alpha}}{D^2}-r^2\tfrac{\ddot{\beta}}{D^2}\right]+&\\
2m_P L^2\dot{\theta}_P\sin\theta_P\cos\theta_P\left[r^2\tfrac{\dot{\alpha}}{D^2}-r^2\tfrac{\dot{\beta}}{D^2}\right]+&\\
(m_w r^2+J_w)[\ddot{\alpha}+\ddot{\beta}]+\tfrac{J_m}{\rho^2}(\ddot{\alpha}-\ddot{\theta}_P) &= \tfrac{\tau_L}{\rho}-\tfrac{\psi}{\rho^2}(\dot{\alpha}-\dot{\theta}_P)\\
\\
\tfrac{1}{2}m_P\left[r^2\tfrac{\ddot{\alpha}}{2}+r^2\tfrac{\ddot{\beta}}{2}+L\ddot{\theta}_P r\cos\theta_P-L\dot{\theta}_P^2 r\sin\theta_P\right]+&\\
\left(J_{\delta P}+m_P L^2\sin^2\theta_P\right)\left[r^2\tfrac{\ddot{\beta}}{D^2}-r^2\tfrac{\ddot{\alpha}}{D^2}\right]+&\\
2m_P L^2\dot{\theta}_P\sin\theta_P\cos\theta_P\left[r^2\tfrac{\dot{\beta}}{D^2}-r^2\tfrac{\dot{\alpha}}{D^2}\right]+&\\
(m_w r^2+J_w)[\ddot{\alpha}+\ddot{\beta}]+\tfrac{J_m}{\rho^2}(\ddot{\beta}-\ddot{\theta}_P) &= \tfrac{\tau_R}{\rho}-\tfrac{\psi}{\rho^2}(\dot{\beta}-\dot{\theta}_P)\\
\\
m_P\left[L^2\ddot{\theta}_P+Lr\tfrac{\ddot{\alpha}+\ddot{\beta}}{2}\cos\theta_P\right]+&\\
J_{\theta P}\ddot{\theta}_P+\tfrac{J_m}{\rho^2}(2\ddot{\theta}_P-\ddot{\alpha}-\ddot{\beta})+&\\
-m_P gL\sin\theta_P+&\\
-m_P L^2\sin\theta_P\cos\theta_P\left[r^2\tfrac{\dot{\alpha}^2}{D^2}+r^2\tfrac{\dot{\beta}^2}{D^2}-2r^2\tfrac{\dot{\alpha}\dot{\beta}}{D^2}\right] &= -\tfrac{\tau_L}{\rho}-\tfrac{\tau_R}{\rho}-\tfrac{2\psi}{\rho^2}\dot{\theta}_P+\tfrac{\psi}{\rho^2}\dot{\alpha}+\tfrac{\psi}{\rho^2}\dot{\beta}
\end{aligned} \tag{114}$$

By re-arranging the terms, it is easy to come back to the form of (43).

BIBLIOGRAPHY

[1] alienpowersystems. alien-5065-sensored-outrunner-brushless-motor, 2017. URL http://tinyurl.com/zdl27ww.

[2] Alps. Alps joystick, 2017. URL http://www.alps.com/prod/info/E/HTML/MultiControl/Potentiometer/RKJXK/RKJXK1210002.html.

[3] W. An and Y. Li. Simulation and control of a two-wheeled self-balancing robot. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 456–461, Dec 2013. doi: 10.1109/ROBIO.2013.6739501.

[4] R. Antonello, I. Nogarole, and R. Oboe. Motion reconstruction with a low-cost mems imu for the automation of human operated specimen manipulation. In *2011 IEEE International Symposium on Industrial Electronics*, pages 2189–2194, June 2011. doi: 10.1109/ISIE.2011.5984500.

[5] Arduino. Hc-06 uart-bluetooth module, 2017. URL http://tinyurl.com/yd3w3ayh.

[6] Zippy batteries. Zippy 6s 8000mah, 2017. URL http://tinyurl.com/ybs4uddc.

[7] J. R. Cao, C. P. Huang, and J. C. Hung. Stabilizing controller design using fuzzy t-s model on two wheeled self-balancing vehicle. In *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, pages 520–523, Nov 2016. doi: 10.1109/ICAMSE.2016.7840187.

[8] C. H. Chiu and Y. F. Peng. Electric unicycle control. In *2017 International Conference on Applied System Innovation (ICASI)*, pages 1546–1549, May 2017. doi: 10.1109/ICASI.2017.7988222.

[9] M. Ciezkowski and E. Pawluszewicz. Determination of interactions between two-wheeled self-balancing vehicle and its rider. In *2015 20th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 851–855, Aug 2015. doi: 10.1109/MMAR.2015.7283988.

[10] funefunshop. 8 inch hub motor, 2017. URL https://www.fonefunshop.com/8-Inch-Electric-Scooter-Balance-Board-Wheel-With-Motor.html.

[11] F. Gabriel, F. De Belie, and X. Neyt. Inductance-based position self-sensing of a brushless dc-machine using high-frequency signal injection. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 3682–3687, Oct 2012. doi: 10.1109/IECON.2012.6389306.

[12] Y. Gong, X. Wu, and H. Ma. Research on control strategy of two-wheeled self-balancing robot. In *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, pages 281–284, Oct 2015. doi: 10.1109/CSMA.2015.63.

[13] Y. Gong, X. Wu, and H. Ma. Research on control strategy of two-wheeled self-balancing robot. In *2015 International Conference on Computer Science and*

Mechanical Automation (CSMA), pages 281–284, Oct 2015. doi: 10.1109/CSMA.2015.63.

[14] K. Iizuka, H. Uzuhashi, M. Kano, T. Endo, and K. Mohri. Microcomputer control for sensorless brushless motor. *IEEE Transactions on Industry Applications*, IA-21(3):595–601, May 1985. ISSN 0093-9994. doi: 10.1109/TIA.1985.349715.

[15] L. Jiang, H. Qiu, Z. Wu, and J. He. Active disturbance rejection control based on adaptive differential evolution for two-wheeled self-balancing robot. In *2016 Chinese Control and Decision Conference (CCDC)*, pages 6761–6766, May 2016. doi: 10.1109/CCDC.2016.7532214.

[16] W. Junfeng and Z. Wanying. Research on control method of two-wheeled self-balancing robot. In *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, volume 1, pages 476–479, March 2011. doi: 10.1109/ICICTA.2011.132.

[17] B. P. Kumar and Krishnan C. M. C. Comparative study of different control algorithms on brushless dc motors. In *2016 Biennial International Conference on Power and Energy Systems: Towards Sustainable Energy (PESTSE)*, pages 1–5, Jan 2016. doi: 10.1109/PESTSE.2016.7516444.

[18] Yong Liu, Z. Q. Zhu, and D. Howe. Direct torque control of brushless dc drives with reduced torque ripple. *IEEE Transactions on Industry Applications*, 41(2):599–608, March 2005. ISSN 0093-9994. doi: 10.1109/TIA.2005.844853.

[19] Loram I. D. Kelly S. M. and M. Lakie. Human balancing of an inverted pendulum: is sway size controlled by ankle impedance. *Journal of Physiology*, 532 (3):879–891, 2001.

[20] Winter D. A. Patla A. E. Prince F. Ishac M. and Gielo-Perczak K. Stiffness control of balance in quiet standing. *Journal of Neurophysiology*, 80(3):1211–1221, 1998.

[21] K. Madhira, A. Gandhi, and A. Gujral. Self balancing robot using complementary filter: Implementation and analysis of complementary filter on sbr. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 2950–2954, March 2016. doi: 10.1109/ICEEOT.2016.7755240.

[22] H. Mansoor, I. U. H. shaikh, and S. Habib. Genetic algorithm based lqr control of hovercraft. In *2016 International Conference on Intelligent Systems Engineering (ICISE)*, pages 335–339, Jan 2016. doi: 10.1109/INTELSE.2016.7475145.

[23] S. Noguchi and S. Jeong. Evaluation of rapid rider weight shifting behavior for a stand-riding-type self-balancing personal mobility vehicle: Pilot study. In *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 1474–1479, Oct 2013. doi: 10.1109/ICCAS.2013.6704119.

[24] N. O-larnnithipong and A. Barreto. Gyroscope drift correction algorithm for inertial measurement unit used in hand motion tracking. In *2016 IEEE SENSORS*, pages 1–3, Oct 2016. doi: 10.1109/ICSENS.2016.7808525.

[25] J. Juan Rincon Pasaye, J. Alberto Bonales Valencia, and F. Jimenez Perez. Tilt measurement based on an accelerometer, a gyro and a kalman filter to control

a self-balancing vehicle. In *2013 IEEE International Autumn Meeting on Power Electronics and Computing (ROPEC)*, pages 1–5, Nov 2013. doi: 10.1109/ROPEC.2013.6702711.

[26] U. Qureshi and F. Golnaraghi. An algorithm for the in-field calibration of a mems imu. *IEEE Sensors Journal*, PP(99):1–1, 2017. ISSN 1530-437X. doi: 10.1109/JSEN.2017.2751572.

[27] M. I. Rashed, S. Lim, and H. Bang. Numerical modeling, testing and bias drift analysis of mems based three-axis gyroscope for accurate angular rate estimation for attitude determination of nano-satellites. In *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 376–381, Oct 2013. doi: 10.1109/ICCAS.2013.6703928.

[28] seedstudio. Grove - imu 9dof, 2017. URL http://wiki.seeedstudio.com/wiki/Grove_-_IMU_9DOF.

[29] Fairchild Semiconductor. Bldc-ripple-torque-reduction-via-modified-sinusoidal-pwm, 2008. URL https://www.fairchildsemi.com/technical-articles/BLDC-Ripple-Torque-Reduction-via-Modified-Sinusoidal-PWM.pdf.

[30] M. K. L. Siu and K. T. Woo. A high-frequency signal injection based sensorless drive method for brushless dc motor. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 367–372, July 2017. doi: 10.1109/ICAR.2017.8023634.

[31] F. Sun, Z. Yu, and H. Yang. A design for two-wheeled self-balancing robot based on kalman filter and lqr. In *2014 International Conference on Mechatronics and Control (ICMC)*, pages 612–616, July 2014. doi: 10.1109/ICMC.2014.7231628.

[32] D. Tedaldi, A. Pretto, and E. Menegatti. A robust and easy to implement method for imu calibration without external equipments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3042–3049, May 2014. doi: 10.1109/ICRA.2014.6907297.

[33] Wikipedia the hoverboard. Self balancing scooter (hoverboard). URL https://en.wikipedia.org/wiki/Self-balancing_scooter.

[34] C. C. Tsai, H. C. Huang, and S. C. Lin. Adaptive neural network control of a self-balancing two-wheeled scooter. *IEEE Transactions on Industrial Electronics*, 57(4):1420–1428, April 2010. ISSN 0278-0046. doi: 10.1109/TIE.2009.2039452.

[35] Benjamin Vedder. A custom vesc, 2015. URL http://vedder.se/2015/01/vesc-open-source-esc/.

[36] Wikipedia. Kalman filter. URL https://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical.

[37] Wikipedia-segwayPT. Segway personal transporter. URL https://en.wikipedia.org/wiki/Segway_PT.

[38] Zanichelli. attrito sul piano inclinato, 2017. URL http://tinyurl.com/y8fjk2on.

[39] Z. Zheng and M. Teng. Modeling and decoupling control for two-wheeled self-balancing robot. In *2016 Chinese Control and Decision Conference (CCDC)*, pages 5263–5267, May 2016. doi: 10.1109/CCDC.2016.7531939.