UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea magistrale Ingegneria Informatica e dell'Automazione

"Controllo sensorless di un motore sincrono a magneti permanenti: validazione sperimentale su un elettrodomestico"

"Sensorless control of a permanent magnet synchronous motor: experimental validation on an appliance"

Relatore:

Prof. **Gianluca Ippoliti**

Tesi di Laurea di:

Laura Moretti

Correlatori:

Prof. **Giuseppe Orlando**

Luigi Fagnano

A.A. **2022 / 2023**

Sommario

La presente tesi ha come obiettivo la validazione sperimentale di un controllo sensorless e model based per un motore sincrono a magneti permanenti. La tecnica utilizzata è lo Sliding Mode Control (SMC) e verrà confrontata con le attuali tecniche di controllo usate, in questo caso con i PI standard. Il motore a cui facciamo riferimento è quello di una pompa di scarico a magneti permanenti di una lavastoviglie.
Questa tesi rappresenta uno dei possibili sviluppi futuri di un altro lavoro di tesi; quest'ultimo, infatti, tratta l'implementazione dei regolatori di velocità e delle correnti sviluppati con logica Sliding Mode (SM) all'interno di un simulatore realizzato e fornito dalla Whirlpool di Fabriano. Avendo ottenuto degli ottimi risultati nell'ambiente simulato, si è pensato ad un'implementazione reale di tale modalità di controllo. Si riporta una descrizione del simulatore e di come sono stati realizzati i regolatori SM in $MATLAB\&Simulink$. Partendo da questo, si effettuano gli step necessari per ottenere il codice C da poter inserire all'interno del microcontrollore della scheda di una lavastoviglie. Il codice C, per l'appunto, è stato ottenuto dai blocchi Simulink che sono stati predisposti all'autogenerazione del codice. Dopo una prima verifica del funzionamento all'interno del simulatore, tali codici vengono inseriti all'interno del microcontrollore. Il sistema utilizzato per la validazione sperimentale è formato da componenti hardware, quali: scheda della lavastoviglie, programmatore per scaricare il firmware nel microcontrollore, due componenti utilizzate per la comunicazione tra il PC e la scheda della lavastoviglie ed il PC; mentre a livello software sono stati utilizzato: il programma $MAC\ Dish$, il tool $FreeMaster$ e il programma $MATLAB\&Simulink$.
Gli esperimenti sono stati effettuati caricando una quantità d'acqua all'interno della lavastoviglie ed eseguendo lo scaricamento sia con controllore SM sia con controllore PI. Dai dati raccolti è possibile fare un confronto tra le due modalità di controllo e si può notare che hanno delle prestazioni simili. In particolare, si osserva elevata prontezza, transitori privi di oscillazioni ed eccellente inseguimento dei segnali di riferimento nei risultati ottenuti con i regolatori Sliding Mode.
Successivamente, sono stati calcolati due indici di prestazione: l'Integral Absolute Error (IAE) e il Mean Squared Error (MSE). Questi ultimi sono stati calcolati variando i parametri in simulazione. Dai valori di questi indici si può affermare che lo SMC gode di robustezza. Successivamente, questi due indici sono stati calcolati anche sui dati sperimentali. Si nota che i valori ottenuti sono simili a quelli che si ottengono in simulazione in condizioni nominali. Grazie a questi esiti si può considerare lo Sliding Mode Control un'alternativa valida e altamente performante

rispetto al controllo PI. Riguardo possibili sviluppi futuri, si ritiene che i risultati finora ottenuti possano migliorare introducendo un osservatore implementato con logica Sliding Mode (SMO), sostituendo l'attuale osservatore di Luenberger.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis aims to showcase the effectiveness of Sliding Mode Control in comparison to standard regulators currently used in the industry, for controlling a synchronous motor with permanent magnets in sensorless mode. Specifically, the thesis will compare Sliding Mode Control with the PI control standard.
Sliding Mode Control (SMC) is a robust and non-linear control technique that provides theoretical assurance of invariance to system uncertainties. Therefore, this thesis aims to demonstrate how a theoretical approach can be applied in a real-world context, resulting in significant advantages.
Based on the findings of thesis [1], the use of Sliding Mode logic for speed and current regulators produces better results than the standard PI implemented in the Whirlpool simulator used in the study. This paper will describe the tools used in this research, including the simulator, the design of the SM regulators, and the steps taken to obtain the C codes for use in Software in the loop. These codes were then inserted into the dishwasher board microcontroller to allow for experimental validation. The results obtained from both SMC and PI Control will be presented and discussed.
Additionally, this paper will examine the system's behavior in the event of electrical or mechanical parametric variations to verify its robustness. Finally, suggestions for future improvements will be proposed.

Chapter 2

Thesis Problem and Objective

This thesis aims to design and develop a model based and sensorless control system for a dishwasher permanent magnet drain pump (figure 2.1). To improve the performance and efficiency of the drive system, Sliding Mode Control (SMC) is used to implement speed and current regulators. The final objective is to demonstrates that the considered solution, compared with a PID control policy, achieves effective speed trajectory tracking and maintains robustness in the presence of system disturbances. Thanks to the Software-in-the-Loop (SIL) methodology, the control software to be verified can directly interact with the emulation of the system implemented by Whirlpool. The Whirlpool company of Fabriano has, in fact, designed a simulator that consist of a MATLAB/Simulink project containing the PI controllers. The PI controllers are S-functions executed starting from the C code inside them, which is the same imported on the dishwasher's microcontroller. The company has also set up the simulator for the C code's auto-generation of the sliding mode controllers subsystems: this will allow to import these controllers C code onto the actual system and make a comparison outside the simulated environment.

Figure 2.1: Dishwasher permanent magnet drain pump

Chapter 3

The Whirlpool Simulator

The objective of this chapter is to introduce the tool through which it was possible to simulate the behavior of the process that concerns the discussion. The Whirlpool simulator permits the observation of the control action of the regulators in a simulated environment. Therefore the chapter describes the general structure of the simulator by outlining the context in which the sliding mode controllers are inserted.

3.1 Whirlpool Simulator structure

The Whirlpool simulator is a Simulink model consisting of 3 blocks shown in figure 3.1.

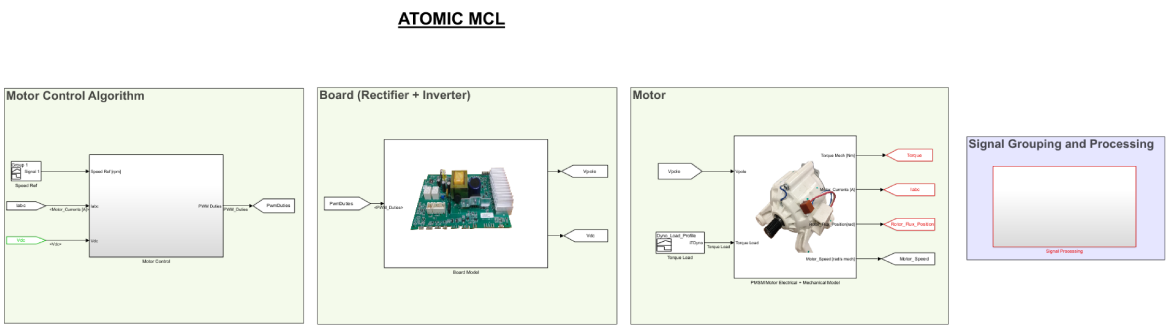

Figure 3.1: Whirlpool Simulator

The electronic board and mechanical model of the electric motor, represented by *Board* and *Motor* blocks respectively, are outside the scope of this thesis work. The block that is of interest for this work is the one depicted in the figure 3.2:

These are the necessary steps for engine control, made with Simulink blocks containing s-functions, which are protected by industrial secrecy.
Simulation can be performed in two modes :

1. Sensorless

2. Sensored

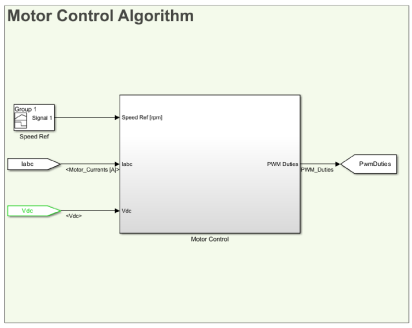

Figure 3.2: Whirlpool Motor Control

To select the desired mode, change the value in the block to Fig. 3.3. This will activate the relevant Simulink blocks. For instance, if you are in *Sensored* mode, you will use the measurements obtained from the sensors. Conversely, if you are in *Sensorless* mode, you will consider the measurements obtained by the observer.

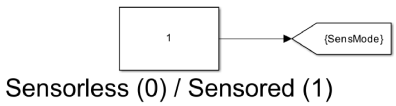

Figure 3.3: SensMode

The diagram of blocks used for implementing control from Whirlpool is presented below:

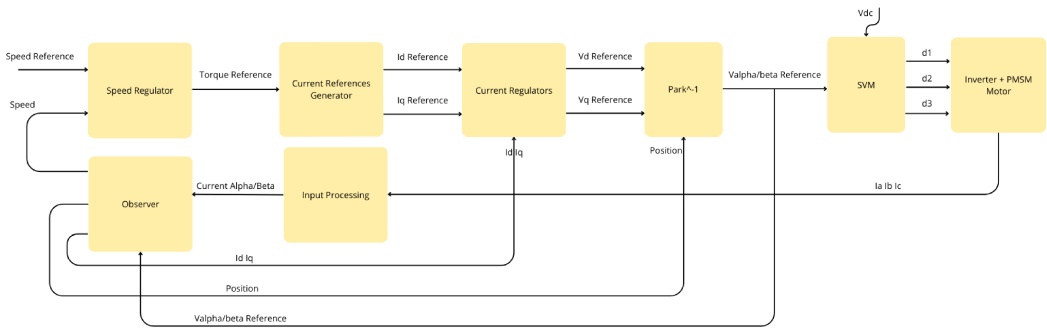

Figure 3.4: Control scheme

3.1.1 Speed Regulator

Whirlpool uses a speed controller that is based on a standard PI controller. This controller receives the error of speed as input, which is the difference between the reference speed and the measured speed, and outputs the reference torque.
The measured speed can either be estimated by the observer or obtained by the sensor. Both speeds are expressed in mechanical radians per second. The parameters

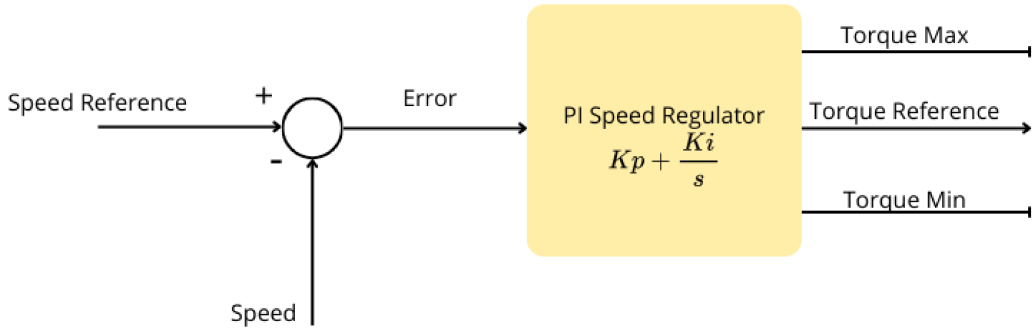

Figure 3.5: Whirlpool Speed Controller

used are:

1. Kp: proportional gain
2. Ki: integrative gain

In addition, the output of the PI regulator is limited above and below to avoid exceeding the physical limits of the motor and board.

3.1.2 Reference Currents generator

The reference's torque is divided by K_t, which represents the torque constant, to obtain the reference current Iq to be supplied as input to the current regulator (3.6).

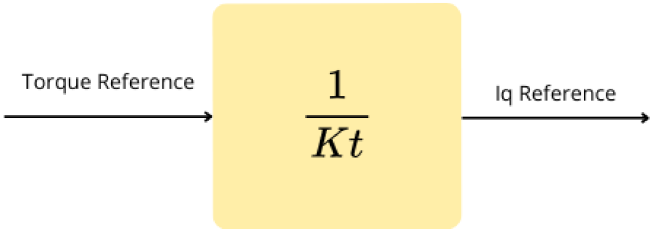

Figure 3.6: Whirlpool Current References Generator

3.1.3 Current Regulator

The current regulator made by Whirlpool is implemented by a standard PI regulator. This regulator takes as input the error obtained from the difference between the

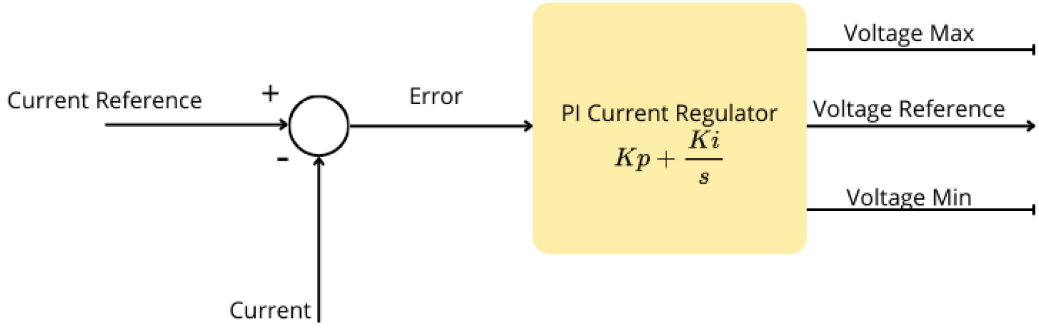

Figure 3.7: Whirlpool Currents Controller

reference current and the measured current and provides as output the reference voltage in the d and q coordinates (fig.3.7). The parameters used are:

1. Kp: proportional gain
2. Ki: integrative gain

In addition, it is important to limit the output of the PI regulator above and below the available voltage on the board.

3.1.4 Transformations

After acquiring the reference voltages in the d-q coordinates, they are transformed into the alpha-beta coordinates by performing an inverse Park transform. These voltages are then used for the Space Vector Modulation, which takes the input voltages in Alpha-Beta coordinates, along with the voltage of the DC bus and generates the PWM pulses to be supplied as input to the inverter. Finally, there is an inverter connected to a synchronous permanent magnet motor. (fig.3.8)

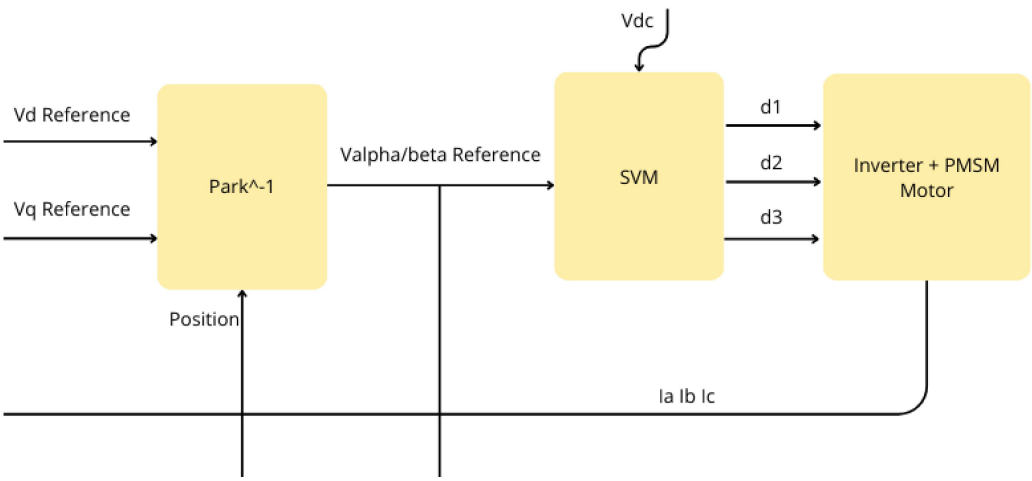

Figure 3.8: Inverse Park Transformation

These steps are essential for obtaining the currents in the ABC coordinates.

3.1.5 Input Processing

This module converts currents from ABC coordinates to Alpha-Beta coordinates using a direct Clarke transform.

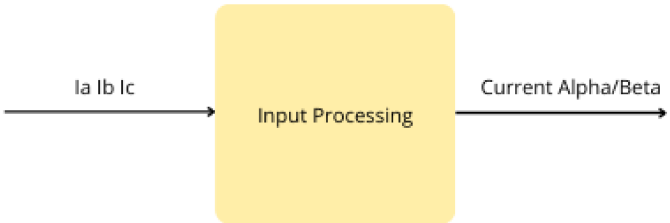

Figure 3.9: Input Processing

3.1.6 Observer

The Whirlpool observer utilizes a standard Luenberger observer, where the error between the estimated and measured quantity is multiplied by a gain, G. (3.10)

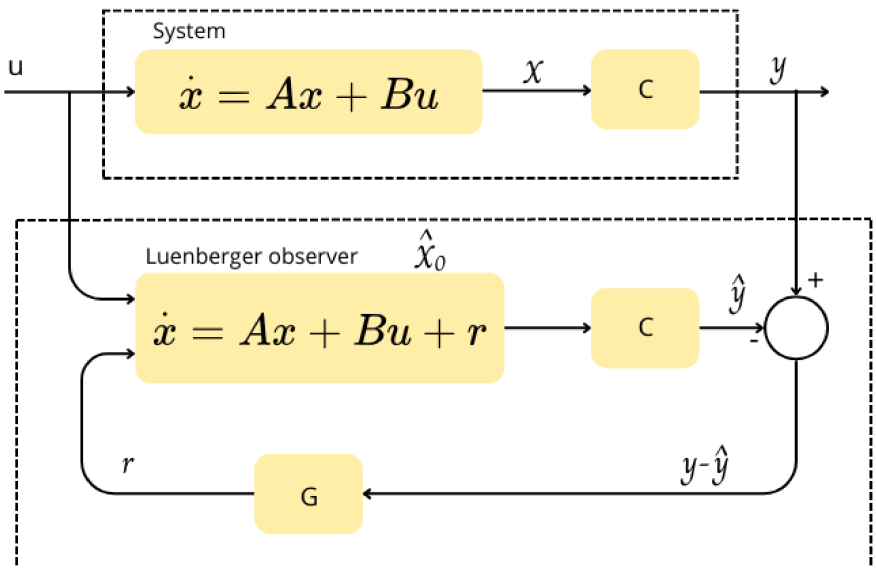

Figure 3.10: Whirlpool Observer

The input includes the motor's currents and voltages. The output provides the estimated speed, position, and force against the electromotive. The parameters used are:

1. Inductance: Ld e Lq
2. Magnetic flux
3. Phase resistance
4. Gain G

3.2 Motor Control

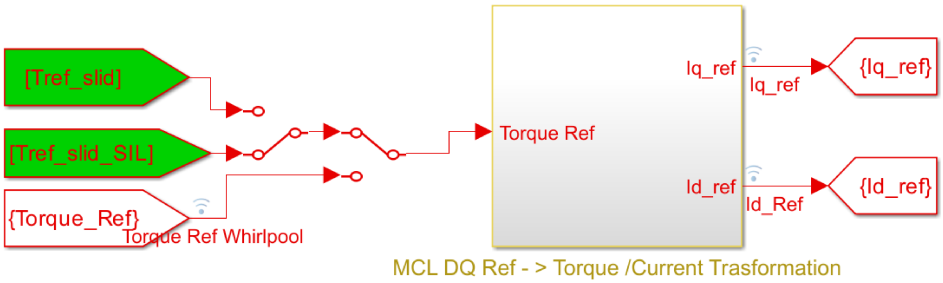

Figure 3.11: Selection of the torque value for calculating the reference currents

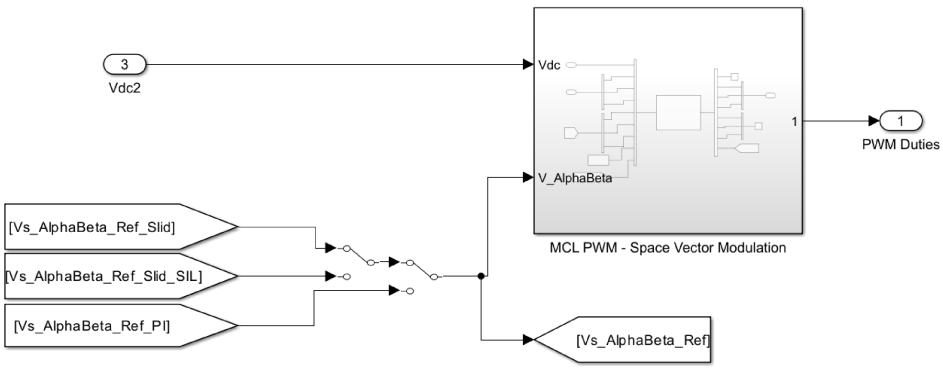

Figure 3.12: Selection of the value of the α-β voltages for the SVM

As anticipated in the first section of the chapter, the "Motor Control" block contains the speed controller and the current controllers. In version 22 of the software, there are:

1. the regulators implemented with Sliding Mode technique;
2. needed block for the software in the loop (SIL) of the regulator's autogenerated code (chapter 6);
3. the regulators implemented by Whirlpool.

The simulation is designed to switch from one controller to another for both the speed and current regulator (figures 3.11 and 3.12). The controllers implemented

with the *Sliding Mode* technique, which concern the discussion, are contained in two subsystems whose description is given below.

3.3 Design of the speed controller subsystem in Simulink

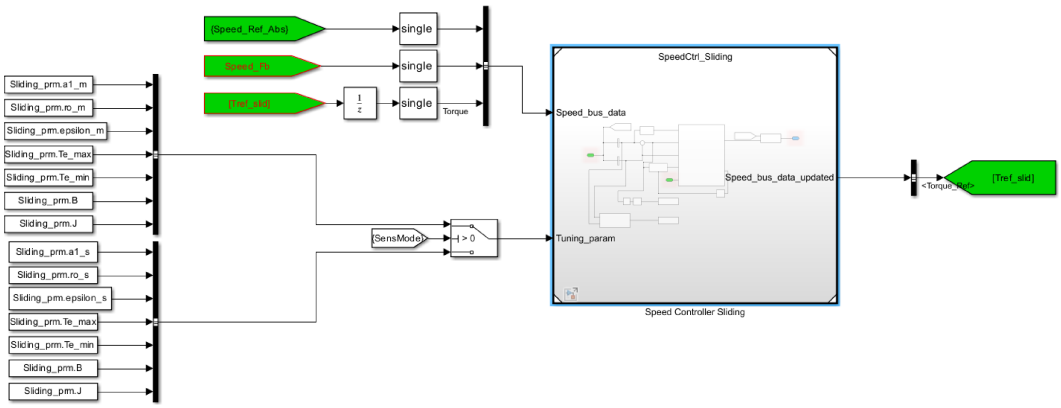

Figure 3.13: Speed controller subsystem

The speed regulator is contained within the "Speed Controller Sliding" subsystem (figure 3.13) which has the following inputs:

- **"Speed_bus_data"**: it is a bus consisting of $Speed\_Ref\_Abs$, $Speed\_Fb$ and $Tref\_slid$ that are respectively reference speed, measured or estimated speed and the reference torque calculated with the Sliding Mode technique;
- **"Tuning_param"**: it is the result of selection between two buses which contain speed control law's tuning parameters (a_1,ρ e ε), the reference torque upper bound Te_max, its lower bound Te_min, the coefficient of viscous friction B and the mechanical inertia of the motor and load J. The first bus starting from the top contains sensored mode tuning parameters, while the second bus the sensorless mode ones; these buses are identified and selected by the switch before the subsystem. The switch selects the top bus when $SensMode$ is equal to 1 and vice versa if it is equal to 0.

"Speed_bus_data_updated" is the output of the subsystem; the reference torque calculated can be extracted from this bus through a Bus Selector.
In the subsystem it is easy to find the MATLAB Function "Speed Controller" which contains the sliding mode control law; the needed mathematical steps for its synthesizing are in chapter 4.

3.4 Design of the current regulator subsystem in Simulink

Current I_q and I_d controllers are inside the subsystem "Current Controller Sliding (figura 3.14) and it has the following inputs:

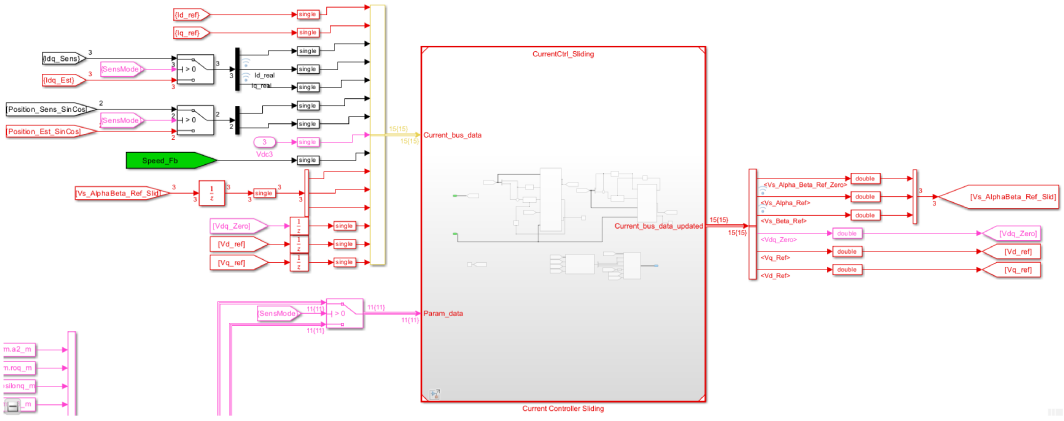

Figure 3.14: Current regulator subsystem

- **"Current_bus_data"**: it is a bus composed by the reference currents Id_ref and Iq_ref, the measured or estimated currents I_d and I_q, measured or estimated rotor position ($Position\_Sens\_SinCos$ and $Position\_Est\_SinCos$), the voltage Vdc, the measured or estimated speed $Speed\_Fb$, the reference voltages α-β, the sequence zero voltage Vdq_Zero and lastly the reference voltages Vd_ref and Vq_ref.

- **"Param_data"**: it is the result of the selection between two buses which contain both currents control laws tuning parameters ($a2$, $a3$, roq, rod, $espilonq$, $epsilond$) and control electrical parameters ($lambda0$, R, Lq, Ld). The bus is shown in figure 3.15.

"Current_bus_data_updated" is the output of the subsystem; the reference voltages α-β, Vd_ref, Vq_ref and their sequence zero voltages can be extracted from the above-mentioned bus.
In the subsystem it is easy to find the MATLAB Functions "Iq Controller" and "Id Controller" which contain the sliding mode control laws; the needed mathematical steps for their synthesizing are in chapter 5.

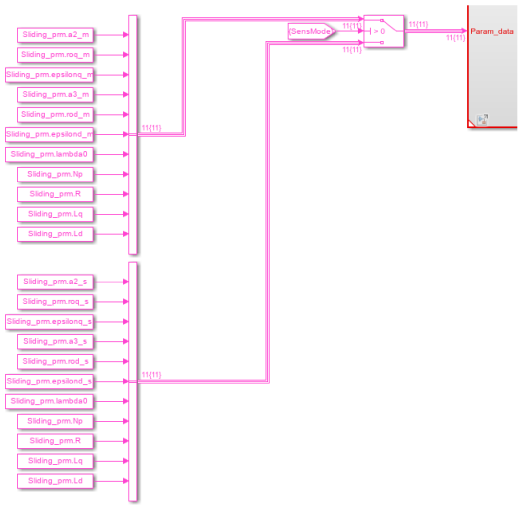

Figure 3.15: *Param_data* buses with switch

3.5 MATLAB Files

Before proceeding to the discussion, it is useful to briefly describe the MATLAB scripts that allowed the simulator to be adapted to control needs using the *Sliding Mode* technique. The scripts that will be reported in this section provide the structure of the aforementioned buses, as well as their initialization through *tuning* parameters and the electrical and mechanical motor control parameters. The simulator consists of a group of folders, in particular, the "main_model" folder contains all Simulink files and the scripts of interest (figure 3.16).

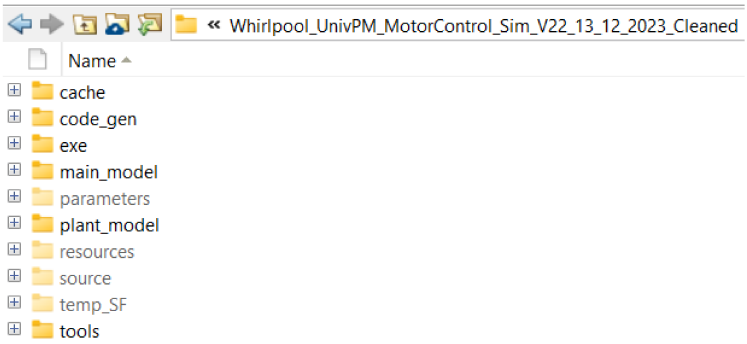

Figure 3.16: Project directory

3.5.1 BusInit_CodeGen.m

This script defines the structure of the buses used for the speed regulator and the current regulators. Once the structure of the buses is defined, the script "parameters_sliding_init.m" initializes them as shown in the next subsection. The speed controller has two buses:

- "MCL_SPEED_SMC_IO_F_TYPE": it contains the reference speed $Speed\_Rot\_Ref$, the measured or estimated speed $Speed\_Rot$ and the reference torque $Torque\_Ref$;
- "MCL_SPEED_SMC_PARAMS_TYPE": it contains the tuning parameters $a1$, ro and $epsilon$, the reference torque saturation bounds Te_max and Te_min, the the coefficient of viscous friction B and the mechanical inertia of the motor and load J.

These buses are both inputs of the subsystem "Speed_Controller_Sliding" described previously; the first one has three signals and the second one has seven signals. The current controller has also two buses:

- "MCL_CURRENT_SMC_IO_F_TYPE": it contains the reference currents Id_ref and Iq_ref, the zero sequence current Idq_zero, Id and Iq currents, sine and cosine values to apply the Park transform, the voltage Vdc, the measured or estimated speed, the triple of voltages obtained from the inverse Park transform ($Vs\_Alpha\_Beta\_Ref\_Zero$, Vs_Alpha_Ref, Vs_Beta_Ref) and the triple of voltages in Park domain (Vdq_Zero, Vd_Ref, Vq_Ref);
- "MCL_CURRENT_SMC_PARAMS_TYPE": it contains the tuning parameters of both current controllers (respectively $a2$, roq, $epsilonq$ and $a3$, rod, $epsilond$), the electrical parameters as the flux linkage of the permanent magnet $lambda0$, the number of pole pairs Np, the winding resistance R and the winding inductances Ld and Lq.

The first bus has fifteen signals, while the second one has eleven of them. These buses are the inputs of the subsystem "Current_Controller_Sliding".

```
%%%%%%%%%%%%%%%%%%%%%%%% Speed COntroller %%%%%%%%%%%%%%%%%%%%
speed_ctrl_io.Speed_Rot_Ref = single(0);
speed_ctrl_io.Speed_Rot =single(0);
speed_ctrl_io.Torque_Ref = single(0);

Simulink.Bus.createObject(speed_ctrl_io)
MCL_SPEED_SMC_IO_F_TYPE = slBus1;
clear slBus1
clear speed_ctrl_io

speed_ctrl_prm.a1= single(0);
speed_ctrl_prm.ro=single(0);
speed_ctrl_prm.epsilon =single(0);
speed_ctrl_prm.Te_max =single(0);
speed_ctrl_prm.Te_min = single(0);
speed_ctrl_prm.B = single(0);
speed_ctrl_prm.J = single(0);

Simulink.Bus.createObject(speed_ctrl_prm)
```

```
MCL_SPEED_SMC_PARAMS_TYPE = slBus1;
clear slBus1
clear speed_ctrl_prm

%%%%%%%%%%%%%%%% Current Controller %%%%%%%%%%%%%%%%%%

current_ctrl_io.Id_ref = single(0);
current_ctrl_io.Iq_ref = single(0);
current_ctrl_io.Idq_zero =single(0);
current_ctrl_io.Id =single(0);
current_ctrl_io.Iq =single(0);
current_ctrl_io.Sin =single(0);
current_ctrl_io.Cos = single(0);
current_ctrl_io.Vdc = single(0);
current_ctrl_io.Speed = single(0);

current_ctrl_io.Vs_Alpha_Beta_Ref_Zero = single(0);
current_ctrl_io.Vs_Alpha_Ref = single(0);
current_ctrl_io.Vs_Beta_Ref = single(0);
current_ctrl_io.Vdq_Zero= single(0);
current_ctrl_io.Vd_Ref= single(0);
current_ctrl_io.Vq_Ref=single(0);

Simulink.Bus.createObject(current_ctrl_io)
MCL_CURRENT_SMC_IO_F_TYPE = slBus1;
clear slBus1
clear current_ctrl_io


current_ctrl_prm.a2= single(0);
current_ctrl_prm.roq =single(0);
current_ctrl_prm.epsilonq =single(0);
current_ctrl_prm.a3 =single(0);
current_ctrl_prm.rod =single(0);
current_ctrl_prm.epsilond =single(0);
current_ctrl_prm.lambda0 =single(0);
current_ctrl_prm.Np =single(0);
current_ctrl_prm.R =single(0);
current_ctrl_prm.Lq =single(0);
current_ctrl_prm.Ld =single(0);

Simulink.Bus.createObject(current_ctrl_prm)
MCL_CURRENT_SMC_PARAMS_TYPE = slBus1;
clear slBus1
clear current_ctrl_prm
```

3.5.2 parameters_sliding_init.m

This script's role is to initialize all the parameters contained in the buses previously described. Tuning parameters are empirically selected to obtain satisfying performances of the speed and currents controllers. Signals behavior is also very sensitive to the flux linkage and the torque constant's values in fact, the values assigned to them are slightly different from those obtained by applying the formula that links them. The last part of the script can be uncommented for the robustness testing; this topic will be discussed in detail in the chapter 7.

```
%%%%%%%%%%%%%%%%% Speed Parameters %%%%%%%%%%%%%%%%%%%
% tuning parameters SENSORLESS
Sliding_prm.a1_s= 90;
Sliding_prm.ro_s = 0.08;
Sliding_prm.epsilon_s = 400;

% tuning parameters SENSORED
Sliding_prm.a1_m= 10;
Sliding_prm.ro_m = 0.5;
Sliding_prm.epsilon_m = 200;

% mechanical parameters
Sliding_prm.Te_max = 0.070000000298023;
Sliding_prm.Te_min = -0.009999999776483;
Sliding_prm.B=7.40e-05;
Sliding_prm.J=2.130e-06;

%%%%%%%%%%%%%%%% Current Parameters%%%%%%%%%%%%%%%%%%%%
% tuning parameters SENSORLESS
%Iq
Sliding_prm.a2_s = 500;
Sliding_prm.roq_s = 5;
Sliding_prm.epsilonq_s = 10;
%Id
Sliding_prm.a3_s = 500; %500
Sliding_prm.rod_s = 5; %5
Sliding_prm.epsilond_s = 3000; %3000

% tuning parameters SENSORED
%Iq
Sliding_prm.a2_m = 100;
Sliding_prm.roq_m= 20;
Sliding_prm.epsilonq_m = 10;
%Id
Sliding_prm.a3_m = 1000;
Sliding_prm.rod_m = 136;
Sliding_prm.epsilond_m = 3000;
```

```

% electrical parameters
Sliding_prm.lambda0 = 0.0857;
Sliding_prm.Np=1;
Sliding_prm.R=45.5;
Sliding_prm.Lq=0.120;
Sliding_prm.Ld=0.120;
Motorparams.K_Torque = 0.128;

% Motorparams.K_Torque = (3/2)*(Sliding_prm.lambda0)*(Sliding_prm.Np);


Sliding_prm.prm0 = 0;
Sliding_prm.prm1 = 0;

%%%%%%%%%%%% Automatic Robustness Tests %%%%%%%%%%%%%%%%%
%tableData = table('Size', [0, 2],'VariableTypes', {'double', 'double'}, 'VariableNames', {'IAE', 'MSE'});

%for i = 1:32
%    [Rs,Ld,Lq,Phim,J,B] = robustness_test(i);
%    Motor_prm.Electrical.Rs_0= Rs;
%    Motor_prm.Electrical.Ld_0= Ld;
%    Motor_prm.Electrical.Lq_0= Lq;
%    Motor_prm.Electrical.PM_flux_h1_0= Phim;
%    Motor_prm.Mechanical.B= B;
%    Motor_prm.Mechanical.Jrotor= J;
% sim('MCU_Simulation_Architecture_Sliding_CodeGen_13_12_2023.slx',10);

%    newRow = table(IAE, MSE, 'VariableNames', {'IAE', 'MSE'});
%    tableData = [tableData; newRow];
%end
% excelFileName = 'IAE_MSE.xlsx';
%writetable(tableData, excelFileName, 'Sheet', 'Sheet1');
```

Chapter 4

The speed controller design

In the first part of this chapter, the focus is on the calculation of the speed regulator's control law. The control law synthesis adheres to the theory presented in chapter 4 of the thesis [1]. The second part of the chapter will cover the design of the controller in Simulink. Lastly, the third part provides the MATLAB Function's code which carries out the speed control action.

4.1 The speed control law

Considering the following system

$$\begin{cases} \dot{x}_1(t) = x_2 = \omega_{ref} - \omega_{mis} \\ \dot{x}_2(t) = f(x) + g(x)u = \dot{\omega}_{ref} - \dot{\omega}_{mis} \end{cases} \tag{4.1}$$

while the sliding surface is defined as

$$s = e + a_1 \int e \Longrightarrow s = x_2 + a_1 x_1 \tag{4.2}$$

where $e = \omega_{ref} - \omega_{mis}$ is the speed error and its integration is the position error. The control law represents the electrical torque.

$$u = \tau_e = J\dot{\omega}_{mis} + B\omega_{mis} + \tau_L \quad [2] \tag{4.3}$$

Explicating $\dot{\omega}_{mis}$

$$\dot{\omega}_{mis} = \frac{u - B\omega_{mis} - \tau_L}{J} + \delta(\omega) \tag{4.4}$$

the system is

$$\begin{cases} \dot{x}_1(t) = \omega_{ref} - \omega_{mis} \\ \dot{x}_2(t) = \dot{\omega}_{ref} - \frac{u}{J} + \frac{B}{J}\omega_{mis} + \frac{\tau_L}{J} - \delta(\omega) \end{cases} \tag{4.5}$$

so the functions $f(x)$ and $g(x)$ can be determined.

$$
\begin{aligned}
f(x) &= \dot{\omega}_{ref} + \frac{B}{J}\omega_{mis} + \frac{\tau_L}{J} - \delta(\omega) \\
g(x) &= -\frac{1}{J}
\end{aligned}
\tag{4.6}
$$

Consequently this is the Lyapunov function derivative (from theory in the thesis [1])

$$
\dot{V}(s) = s(-\frac{1}{J})[u - J\dot{\omega}_{ref} - B\omega_{mis} - \tau_L + J\delta(\omega) - Ja_1(\omega_{ref} - \omega_{mis})] \tag{4.7}
$$

and the control law is obtained through a few steps:

$$
u = \underbrace{B\omega_{mis} + J[\dot{\omega}_{ref} + a_1(\omega_{ref} - \omega_{mis})]}_{u_{eq}} - \underbrace{\rho \cdot sat(s)}_{v} \tag{4.8}
$$

It removes all the known dynamic from the Lyapunov function derivative.

$$
\dot{V}(s) = s(-\frac{1}{J})(v - \tau_L + J\delta(\omega)) \tag{4.9}
$$

By adjusting the values of ε, a_1, and ρ, a tuning can be carried out.

4.1.1 The speed sliding surface

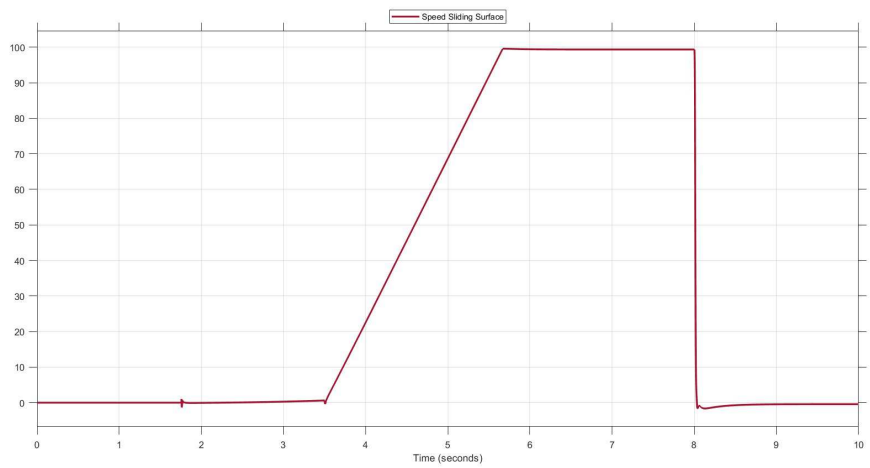

Figure 4.1: The speed sliding surface

The speed sliding surface is convergent: this means that a sliding mode motion of the speed error is correctly occurring (figure 4.1). In general, the sliding surface behavior depends on the chosen tuning and can improve the surface convergence's speed.

4.2 Speed Controller Sliding

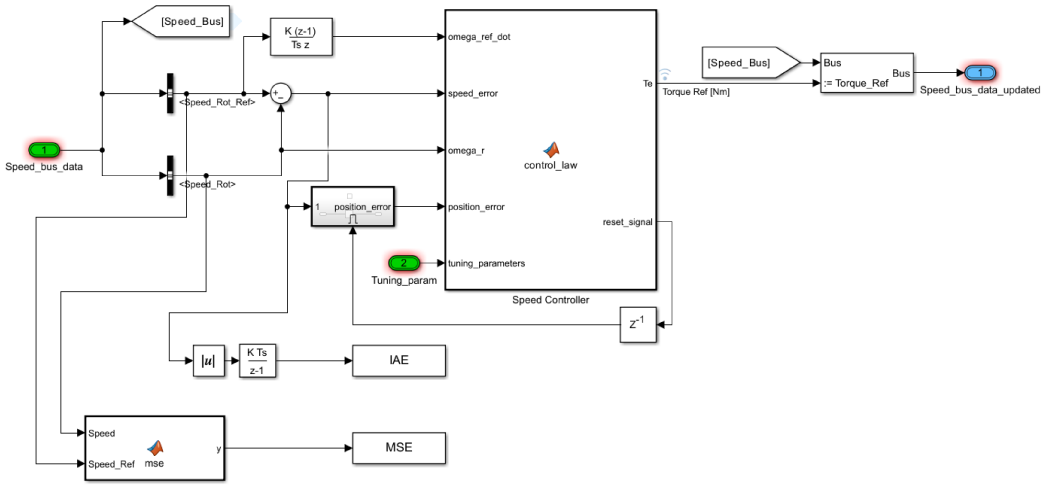

Figure 4.2: Speed regulator Simulink scheme

As anticipated in chapter 3, there is a MATLAB Function inside the subsystem "Speed Controller Sliding"; Inside the MATLAB Function, there is the control law previously implemented (figura 4.2). These are the block's inputs:

- *omega_ref_dot*: the motor's reference angular acceleration, calculated as the reference angular speed derivative provided by the "Speed_Bus";
- *speed_error*: the speed error, calculated as the difference between the reference speed and the measured/estimated speed (both extracted from the "Speed_Bus");
- *omega_r*: the motor's measured or estimated angular speed;
- *position_error*: the position error calculated as the integration of the speed error inside an Enabled Subsystem;
- *tuning_parameters*: a bus that contains the tuning parameters.

The position error is calculated inside an Enabled Subsystem, through which the integration is reset when the block receives the reset signal called *reset_signal* as input; it notifies the saturation of the torque preventing the windup of the integral action (see the next subsection 4.2.1). The *reset_signal* and the reference torque *Torque_Ref* are the outputs of the Speed Controller block. The reference torque value updates the torque value inside the "Speed_Bus" through the Bus Assignment block, then the updated bus is returned as output by the subsystem. Furthermore, within the subsystem, the **IAE** and **MSE** indices were calculated to carry out robustness tests. The robustness tests' methods and results are reported in chapter 7.

4.2.1 Integrator Windup

The controlled variable of every regulator always has upper and lower bounds. During the permanent scheme, the controlled variable's value is significantly distant from saturation's limits, but it could reach them during wide and rapid transients. When the controlled variable is saturated, the process evolves with constant input, without a proper regulation as if the process was not in a closed loop. The regulator also evolves as it was in an open loop, without a feedback effect between the output and the error. Furthermore, the integrator is a dynamic system not asymptotically stable; thus when the controlled variable's saturation occurs, the range of values the integrator takes could be uneffective as control action. It takes time before the integrator's value ceases to increase; this phenomena is known as the integrator windup. The conditional integration is an effective method to prevent the wind-up: the integration is switched off when the control is far from steady state. Integral action is thus only used when certain conditions are fullfilled, otherwise the integral term is kept constant [5].

4.3 The Speed Controller's code

"Speed Controller" contains the code in this section. The notation used in the code respects the one adopted for the mathematical calculation of the control law. Among the inputs of the MATLAB Function, there is the bus that contains the struct *tuning_parameters*, necessary for the initialization of the tuning parameters. Subsequently, the sliding surface s is defined and the saturation is calculated so that v can be set. It is useful to remember that v is one of the two components of the control law and note that this component depends on the sliding surface itself and the parameters ρ and ε. Immediately afterward the component u_{eq} and the control law u itself are defined. Finally, the saturation limits of the reference torque Te are established. The reference torque is the controller's output. Whenever the torque saturates, *reset_signal* takes the value -1 to disable the integral's action, otherwise it takes the value 1.

```
function [Te,reset_signal] = control_law(omega_ref_dot,
    speed_error,omega_r,position_error,tuning_parameters)

a1 = tuning_parameters.a1;
ro = tuning_parameters.ro;
epsilon = tuning_parameters.epsilon;
Te_max=tuning_parameters.Te_max;
Te_min=tuning_parameters.Te_min;
B=tuning_parameters.B;
J=tuning_parameters.J;

%sliding surface
```

```
s = speed_error + a1*position_error;

%sat(s)
if abs(s) > epsilon
    v = ro*sign(s);
else
    v = ro*(s/epsilon);
end

%control law
u_eq = B*omega_r + J*(omega_ref_dot+a1*(speed_error));
u = u_eq + v;

%sat(Te)
if u >= Te_max
    u = Te_max;
    reset_signal=-1;
elseif u <= Te_min
    u = Te_min;
    reset_signal=-1;
else
    reset_signal=1;
end
Te=u;
```

Chapter 5

The currents controller design

The first part of this chapter is dedicated to calculating the control law of current regulator I_d and current regulator I_q, the synthesis of which follows the theory reported in the thesis [1]. The second part describes the design of the controllers in Simulink. Finally, the third part of the chapter provides detailed comments and reports on the MATLAB Functions code that executes the control action on the currents.

5.1 The current I_q control law

Considering the following system

$$\begin{cases} \dot{x}_1(t) = x_2 = I_{q_{ref}} - I_{q_{mis}} \\ \dot{x}_2(t) = f(x) + g(x)u = \dot{I}_{q_{ref}} - \dot{I}_{q_{mis}} \end{cases} \tag{5.1}$$

while the sliding surface is defined as follows

$$s = e + a_2 \int e \Longrightarrow s = x_2 + a_2 x_1 \tag{5.2}$$

where $e = I_{q_{ref}} - I_{q_{mis}}$. The control law coincides with the reference voltage $V_{q_{ref}}$.

$$u = V_{q_{ref}} = RI_{q_{mis}} + L\dot{I}_{q_{mis}} + \omega_e \varphi + \omega_e L I_{d_{mis}} \quad [2] \tag{5.3}$$

Explicating $\dot{I}_{q_{mis}}$

$$\dot{I}_{q_{mis}} = \frac{V_{q_{ref}}}{L} - \frac{RI_{q_{mis}}}{L} - \frac{\omega_e \varphi}{L} - \omega_e I_{d_{mis}} + \delta(I_q) \tag{5.4}$$

the system is

$$\begin{cases} \dot{x}_1(t) = I_{q_{ref}} - I_{q_{mis}} \\ \dot{x}_2(t) = \dot{I}_{q_{ref}} - \frac{V_{q_{ref}}}{L} + \frac{RI_{q_{mis}}}{L} + \frac{\omega_e \varphi}{L} + \omega_e I_{d_{mis}} - \delta(I_q) \end{cases} \tag{5.5}$$

so the functions $f(x)$ and $g(x)$ can be determined.

$$
\begin{aligned}
f(x) &= \dot{I}_{q_{ref}} + \frac{RI_{q_{mis}}}{L} - \omega_e I_{d_{mis}} - \delta(I_d) \\
g(x) &= -\frac{1}{L}
\end{aligned}
\tag{5.6}
$$

Consequently this is the Lyapunov function derivative (from theory in the thesis [1]).

$$
\begin{aligned}
\dot{V}(s) = s(-\frac{1}{L})[V_{q_{ref}} - L\dot{I}_{q_{ref}} - \omega_e\varphi - L\omega_e I_{d_{mis}} + \\
-RI_{q_{mis}} - a_2 L(I_{q_{ref}} - I_{q-mis}) + L\delta(I_q)]
\end{aligned}
\tag{5.7}
$$

and the control law is obtained through a few steps:

$$
u = \underbrace{L\dot{I}_{q_{ref}} + \omega_e\varphi + \omega_e I_{d_{mis}} L + RI_{q_{mis}} + a_2 L(I_{q_{ref}} - I_{q_{mis}})}_{} + \underbrace{\rho \cdot sat(s)}_{v}
\tag{5.8}
$$

By adjusting the values of ϵ, a_3 and ρ, a tuning can be carried out.

5.2 The current I_d control law

Considering the following system

$$
\begin{cases}
\dot{x}_1(t) = x_2 = I_{d_{ref}} - I_{d_{mis}} \\
\dot{x}_2(t) = f(x) + g(x)u = \dot{I}_{d_{ref}} - \dot{I}_{d_{mis}}
\end{cases}
\tag{5.9}
$$

while the sliding surface is defined as follows

$$
s = e + a_3 \int e \Longrightarrow s = x_2 + a_3 x_1
\tag{5.10}
$$

where $e = I_{d_{ref}} - I_{d_{mis}}$. The control law coincides with the reference voltage $V_{d_{ref}}$.

$$
u = V_{d_{ref}} = RI_{d_{mis}} + L\dot{I}_{d_{mis}} - \omega_e L I_{q_{mis}} \quad [2]
\tag{5.11}
$$

Explicating $\dot{I}_{d_{mis}}$

$$
\dot{I}_{d_{mis}} = \frac{V_{d_{ref}}}{L} - \frac{RI_{d_{mis}}}{L} + \omega_e I_{q_{mis}} + \delta(I_d)
\tag{5.12}
$$

the system is

$$
\begin{cases}
\dot{x}_1(t) = I_{d_{ref}} - I_{d_{mis}} \\
\dot{x}_2(t) = \dot{I}_{d_{ref}} - \frac{V_{d_{ref}}}{L} + \frac{RI_{d_{mis}}}{L} - \omega_e I_{q_{mis}} - \delta(I_d)
\end{cases}
\tag{5.13}
$$

so the functions $f(x)$ and $g(x)$ can be determined.

$$
\begin{aligned}
f(x) &= \dot{I}_{d_{ref}} + \frac{RI_{d_{mis}}}{L} - \omega_e I_{q_{mis}} - \delta(I_d) \\
g(x) &= -\frac{1}{L}
\end{aligned}
\tag{5.14}
$$

Consequently this is the Lyapunov function derivative (from theory in the thesis [1]).

$$
\dot{V}(s) = s(-\frac{1}{L})[V_{d_{ref}} - L\dot{I}_{d_{ref}} + I_{d_{mis}}(a_3L - R) + L\omega_e I_{q_{mis}} - a_3 L I_{d_{ref}} + L\delta(I_d)] \tag{5.15}
$$

and the control law is obtained through a few steps:

$$
u = \underbrace{L\dot{I}_{d_{ref}} - I_{d_{mis}}(a_3L - R) - L\omega_e I_{q_{mis}} + a_3 L I_{d_{ref}}}_{u_{eq}} + \underbrace{\rho \cdot sat(s)}_{v} \tag{5.16}
$$

5.2.1 The currents sliding surfaces

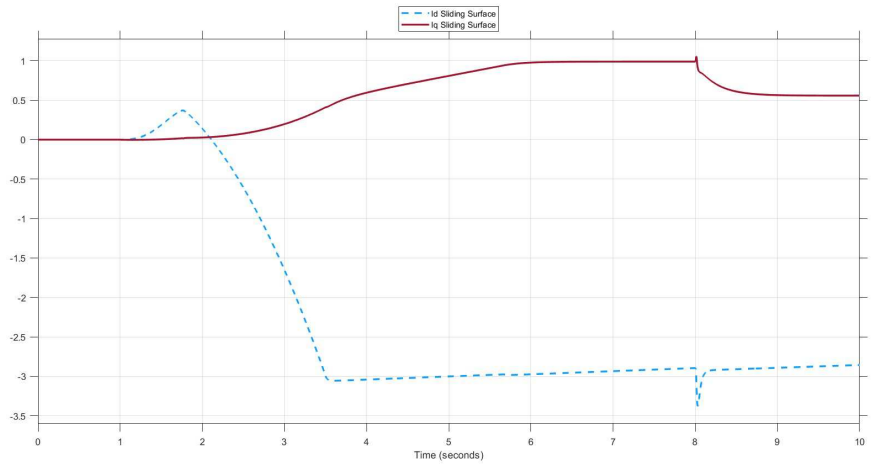

Figure 5.1: The currents sliding surfaces

The sliding surface for the current I_q is currently converging, which indicates that the sliding mode motion of the I_q current error is happening correctly. On the other hand, the sliding surface for the current I_d is increasing so slowly that it can be considered constant. Therefore, by extending the simulation time, the drifting settles down, and the sliding motion is guaranteed by the I_d current error (figure 5.1). The behavior of the sliding surfaces, in general, depends on the chosen tuning and can improve the speed of surface's convergence.

5.3 Current Controller Sliding

As anticipated in chapter 3, there are two MATLAB Functions inside the subsystem "Current Controller Sliding"; Inside the MATLAB Functions "Iq Controller" and "Id Controller", there are the control laws previously implemented (figures 5.2 e 5.3).

5.3.1 Iq Controller

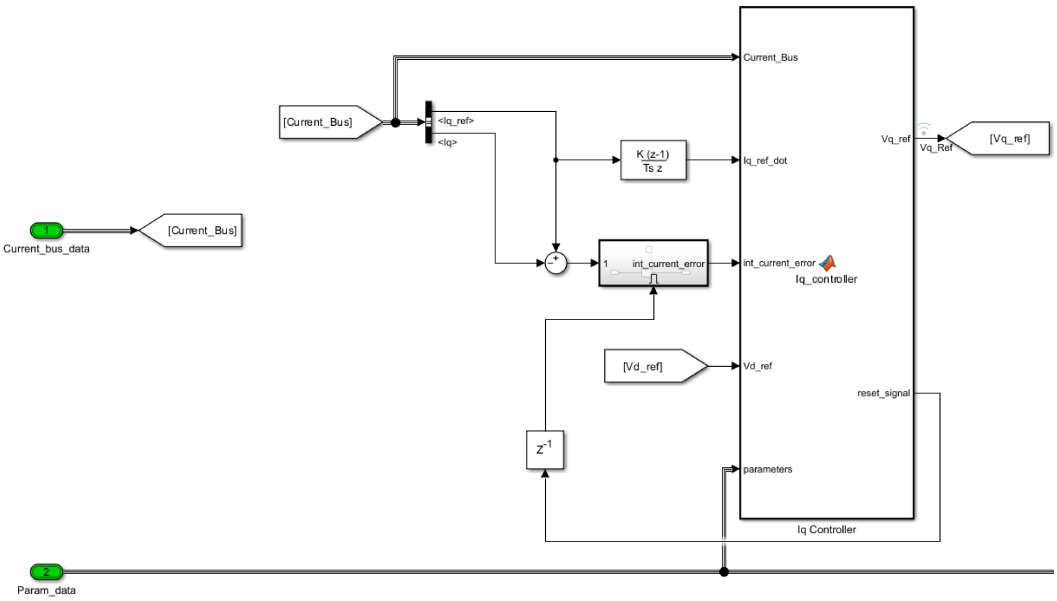

Figure 5.2: The current I_q regulator Simulink scheme

The MATLAB Function "Iq Controller" (figure 5.2) has the following inputs:

- $Current\_Bus$: a bus containing useful signals for the control law calculation;
- Iq_ref_dot: reference current I_q derivative;
- $int\_current\_error$: current error integration, that is calculated as the difference between the reference current $I_{q_{ref}}$ and the measured or estimated I_q current. The Enabled Subsystem that realizes the anti windup logic returns $int\_current\_error$ as output;
- Vd_ref: reference V_d voltage;
- $parameters$: contains the parameters provided by the bus "Param_data", described in chapter 3.

The reference voltage Vq_ref and $reset\_signal$ are the outputs.

5.3.2 Id Controller

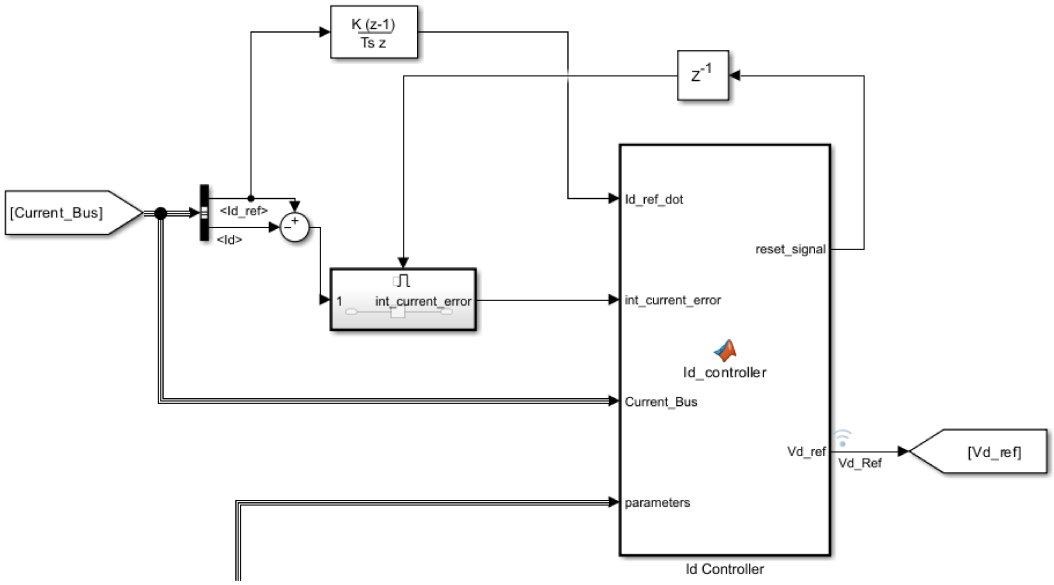

Figure 5.3: The current I_d regulator Simulink scheme

The MATLAB Function "Iq Controller" (figure 5.3) has the following inputs:

- Id_ref_dot: reference current I_d derivative;
- $int\_current\_error$: current error integration, that is calculated as the difference between the reference current $I_{d_{ref}}$ and the measured or estimated current I_d. There is an anti windup logic for this current regulator too;
- $Current\_Bus$ and $parameters$: they are the same input bus of the current I_q controller;

The reference voltage Vd_ref and $reset\_signal$ are the outputs.

5.3.3 Inverse Park Transformation

The "Current_bus_data_updated" bus is populated through a Bus Assignment: the calculated voltage triad Vdq_Zero, Vd_ref, Vq_ref and their inverse Park transform; the latter is calculated by the "InversePark" block (figure 5.4) whose code is shown.

5.4 Iq Controller and Id Controller's code

This section contains the code inside the "Iq Controller" and "Id Controller". They have a structure similar to the "Speed Controller", except for the control laws relating to the currents I_q and I_d. For both controllers:

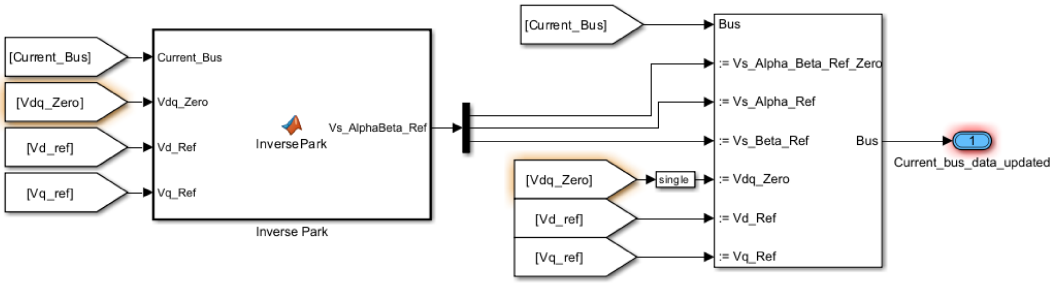

Figure 5.4: Inverse Park Transformation's MATLAB Function

1. the parameters initialization for tuning and electrical parameters happens through, respectively, the struct *parameters* and the bus *Current_Bus*;
2. the respective sliding surface is defined;
3. the control laws are created by combining the components of v and u_{eq}.
4. the saturation limits of the reference voltages are defined. The reference voltages are the regulators outputs;
5. whenever the voltage saturates, *reset_signal* takes on the value -1, otherwise 1.

```
function [Vq_ref, reset_signal] = Iq_controller(Current_Bus,
    Iq_ref_dot,int_current_error,Vd_ref,parameters)

%Iq_ref=Current_Bus.Iq_ref;
omega_r=Current_Bus.Speed;
Iq=Current_Bus.Iq;
Id=Current_Bus.Id;
Vdc=Current_Bus.Vdc;
Iq_ref=Current_Bus.Iq_ref;

lambda0 = parameters.lambda0;
Np = parameters.Np;
omega_e=Np*omega_r;
R=parameters.R;
Lq=parameters.Lq;

a2=parameters.a2;
ro=parameters.roq;
epsilon=parameters.epsilonq;

%sliding surface
s = (Iq_ref - Iq) + a2*int_current_error;

%sat(s)
```

```
if abs(s) > epsilon
    v = ro*sign(s);
else
    v = ro*(s/epsilon);
end

%control law
u_eq = (omega_e*lambda0)+(R*Iq)+Lq*(Iq_ref_dot+(omega_e*Id)+(a2
    *(Iq_ref-Iq)));
u = u_eq + v;

Upper_Limit=sqrt(((Vdc/sqrt(3))^2)-(Vd_ref^2));
Lower_Limit=-sqrt(((Vdc/sqrt(3))^2)-(Vd_ref^2));

%sat(Vq_ref)
if u >= Upper_Limit
    u = Upper_Limit;
    reset_signal=-1;
elseif u <= Lower_Limit
    u = Lower_Limit;
    reset_signal=-1;
else
    reset_signal=1;
end

Vq_ref= u;
```

```
function [reset_signal,Vd_ref] = Id_controller(Id_ref_dot,
    int_current_error,Current_Bus,parameters)

Id=Current_Bus.Id;
Id_ref=Current_Bus.Id_ref;
Iq=Current_Bus.Iq;
omega_r=Current_Bus.Speed;
Vdc=Current_Bus.Vdc;

Np = parameters.Np;
omega_e=Np*omega_r;
R=parameters.R;
Ld=parameters.Ld;

a3=parameters.a3;
ro=parameters.rod;
epsilon=parameters.epsilond;

%sliding surface
s = (Id_ref-Id)+(a3*int_current_error);
```

```

%sat(s)
if abs(s) > epsilon
    v = ro*sign(s);
else
    v = ro*(s/epsilon);
end

%control law
u_eq = (Ld*Id_ref_dot)-(Id*(-R+a3*Ld))-(Ld*omega_e*Iq)+(a3*Ld*
    Id_ref);
u = u_eq + v;

Upper_Limit = Vdc/sqrt(3);
Lower_Limit = -Vdc/sqrt(3);

%sat(Vd_ref)
if u >= Upper_Limit
    u = Upper_Limit;
    reset_signal=-1;
elseif u <= Lower_Limit
    u = Lower_Limit;
    reset_signal=-1;
else
    reset_signal=1;
end

Vd_ref=u;
```

5.5 InversePark code

The "InversePark" code defines the inverse Park transformation matrix and pre-multiplies it to the voltage triad Vdq_Zero, Vd_Ref, Vq_Ref. The sine and cosine of the angle used for the Park transform are extracted from $Current\_Bus$. Finally, the MATLAB Function outputs the new triple, enclosed in the vector $Vs\_AlphaBeta\_Ref$ [3].

```
function Vs_AlphaBeta_Ref  = InversePark(Current_Bus,Vdq_Zero,
    Vd_Ref,Vq_Ref)
sinTheta=Current_Bus.Sin;
cosTheta=Current_Bus.Cos;
T=[cosTheta sinTheta;
    -sinTheta cosTheta];
Vs_AlphaBeta=(T')*[Vd_Ref;Vq_Ref];
Vs_AlphaBeta_Ref=[Vdq_Zero;Vs_AlphaBeta];
end
```

Chapter 6

Code Generation, Implementation and Machine Control

This chapter will illustrate the steps needed for actual implementation. It will describe how to generate the code, how to use it within the simulator and finally the implementation on the machine will be shown.

6.1 Code Generation

As mentioned in chapter 3, the *Motor Control Algorithm* block contains all the necessary blocks for control, including those required for simulating with the generated code. Figures 3.13 and 3.14 illustrate two model references that are designed for the autogeneration of code. These references call two files - *"SpeedCtrl_Sliding"* and *"CurrentCtrl_Sliding"* - in Simulink.

To generate the C code for these blocks, it needs to open the reference model and run the *Slidingparam_and_codegen_init* shorcut. Next, press $Ctrl + B$ to generate the code, which will be compiled through a *compile_CodeGen* shorcut for use as an s-function in the blocks required for this operation. Once you've completed these steps, it's possible to switch to SIL (Software In Loop) mode in the simulation.

The simulation tool allows to switch between different controllers using manual switches. To choose a controller, select the tags that are abbreviated as "SIL". When you use the generated code, all simulations are performed in "*Sensorless* mode.

In the figure 6.1, there is the block representing the *software in the loop* (SIL) with the code generated by the speed regulator subsystem. On the other hand, the figure 6.2 represents the block with the code generated by the current regulator subsystem.

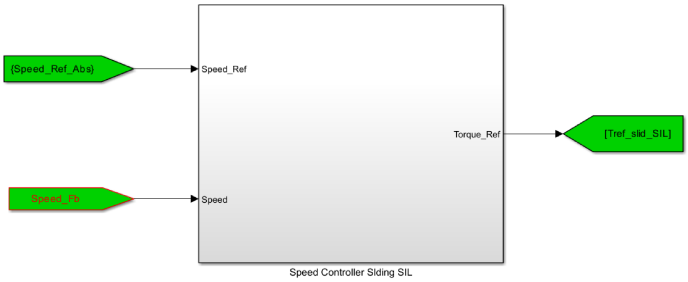

Figure 6.1: Speed Controller Sliding SIL

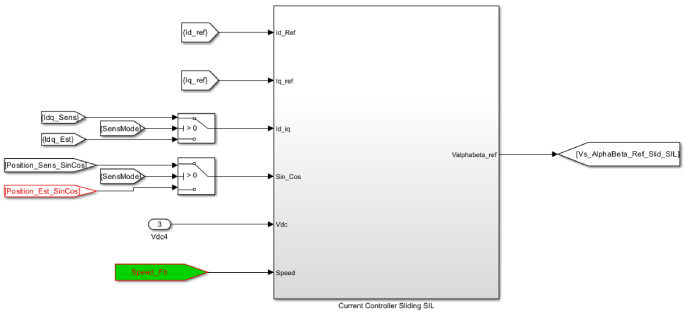

Figure 6.2: Current Controller Sliding SIL

The subsystem represented in Figure 6.1 includes two variables: $Speed\_Ref\_Abs$ and $Speed\_Fb$. These variables respectively refer to the reference speed and the actual speed, depending on the mode selected. The output of this subsystem is $Tref\_slid\_SIL$, which is a reference pair. Figure 6.3 provides a visual representation of the contents of the subsystem:

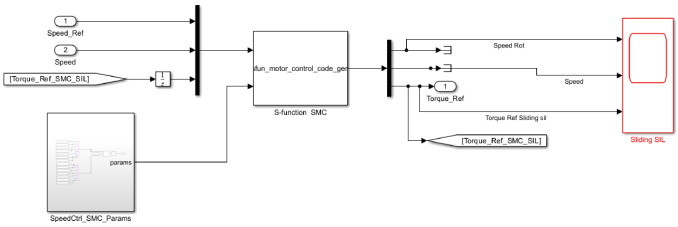

Figure 6.3: Speed Controller Sliding SIL - Inside

the s-function obtained from the compiled code is inside the $S-function\ SMC$. The latter has as input:

- $Speed\_Ref$: reference speed
- $Speed$: actual speed
- $Torque\_Ref\_SMC\_SIL$: reference torque delayed in feedback
- $SpeedCtrl\_SMC\_Params$: block 6.4 containing all the required parameters, described in paragraph 3.3.

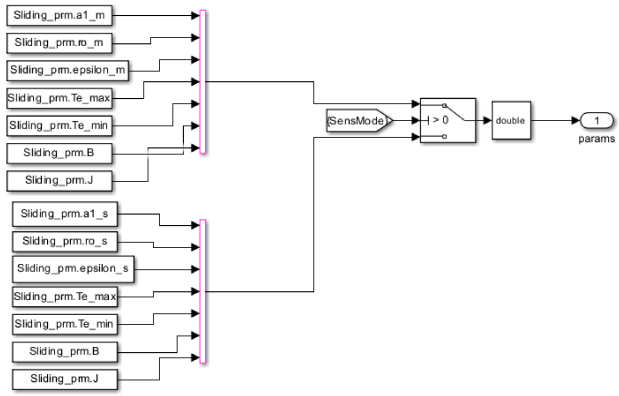

Figure 6.4: Speed Control SMC Parameters

Instead of output, returns:

- *Speed Rot*
- *Speed*
- $Torque\_Ref\_SMC\_SIL$

The *Sliding SIL* block enables the user to visualize graphs.

By manually adjusting the switch, as shown in figure 6.5, you can obtain the reference currents for SIL Id_Ref and Iq_Ref currents represent two inputs required

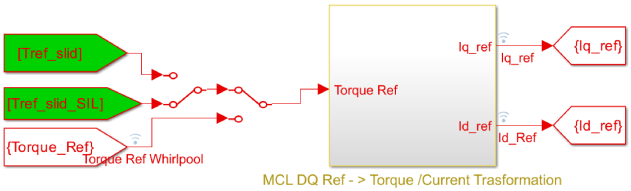

Figure 6.5: Selection of the torque value for calculating the reference current

for subsystem 6.2. In addition to them, the inputs are:

- Id e Iq currents;
- $Position\_Sens\_SinCos$ $Position\_Est\_SinCos$: rotor position;
- Vdc voltage;
- $Speed\_Fb$ speed

The outputs are V_{alpha} e V_{beta} reference voltage.

In figure 6.6 it is shown the subsystem's content:

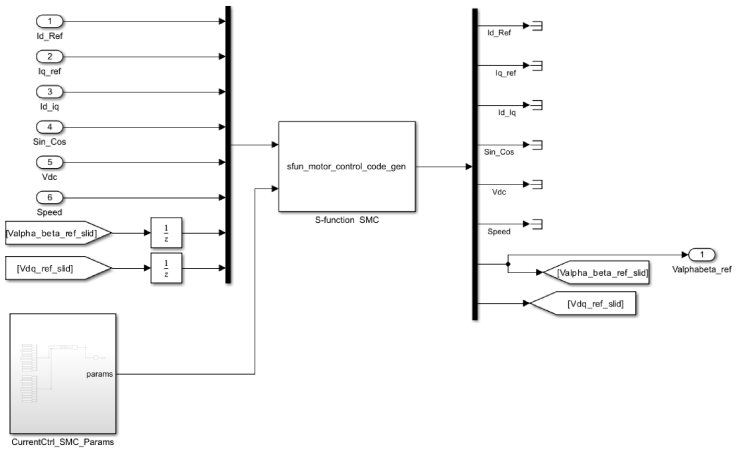

Figure 6.6: Current Control Sliding SIL - inside

The S-function that has been obtained from the compiled code can be found within the block $S - function\ SMC$. The latter has as inputs:

- Id_Ref e Iq_Ref: reference currents
- Id e Iq: currents
- Sin_Cos: rotor's position
- Vdc: voltage
- $Speed$: actual speed
- $Valpha\_beta\_ref\_slid$: α and β reference voltages delayed
- Vdq_ref_slid: d and q reference voltages delayed
- $CurrentCtrl\_SMC\_Params$: block 6.7 containing all the required parameters, described in paragraph 3.4.

$Valpha\_beta\_ref\_slid$ and Vdq_ref_slid are the outputs.

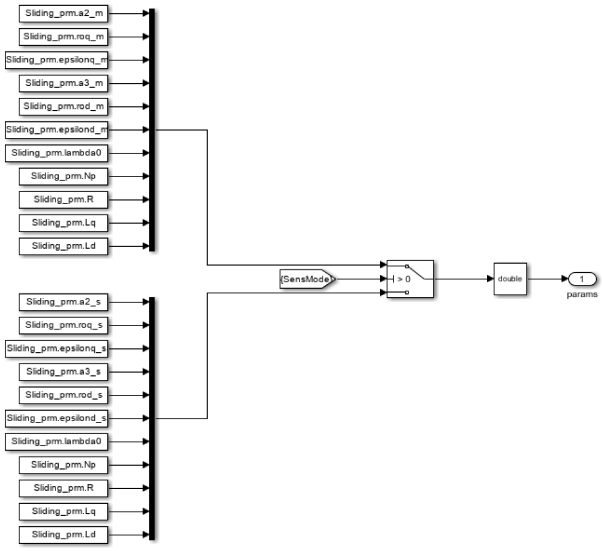

Figure 6.7: Currents Control SMC Parameters

6.2 Graphical Results in nominal conditions

Since the SIL controllers replicate the behavior of the controllers implemented in Simulink, but in the form of C code, the obtained graphs are identical to those from the sliding mode controller in the sensorless case (chapter 7 in thesis [1]). The Sliding Mode regulator can be directly compared to a PI controller since both are realized with s-functions.

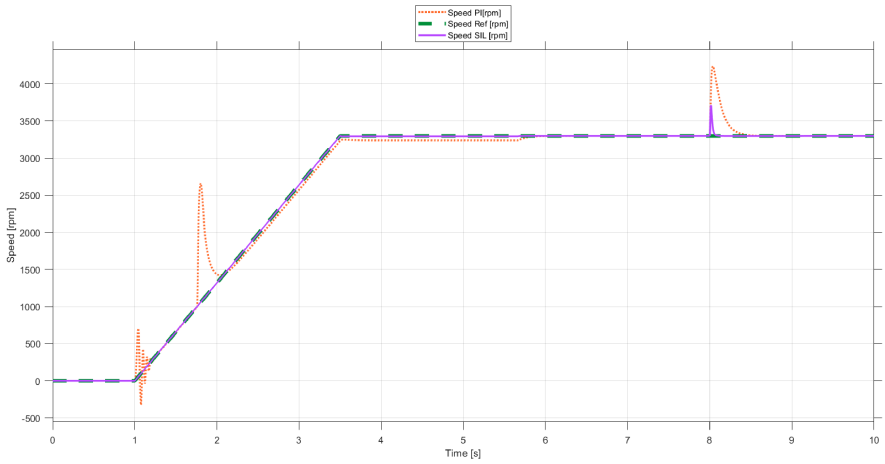

Figure 6.8: Speed SIL and PI

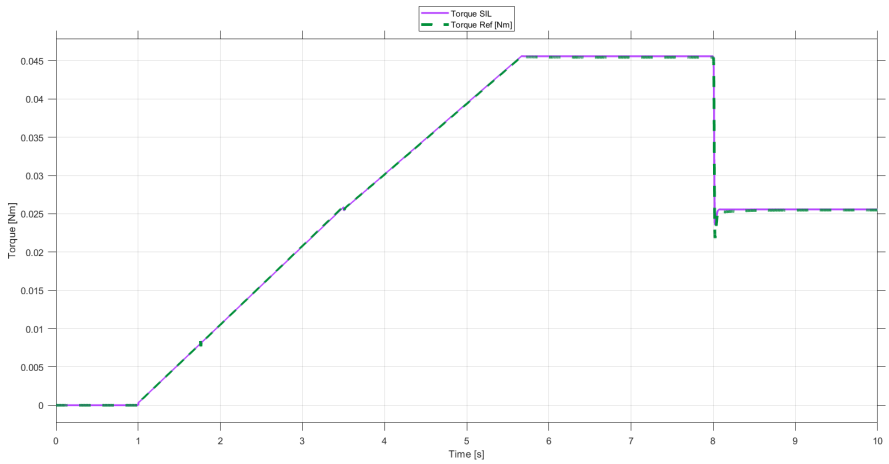

Figure 6.9: Torque SIL

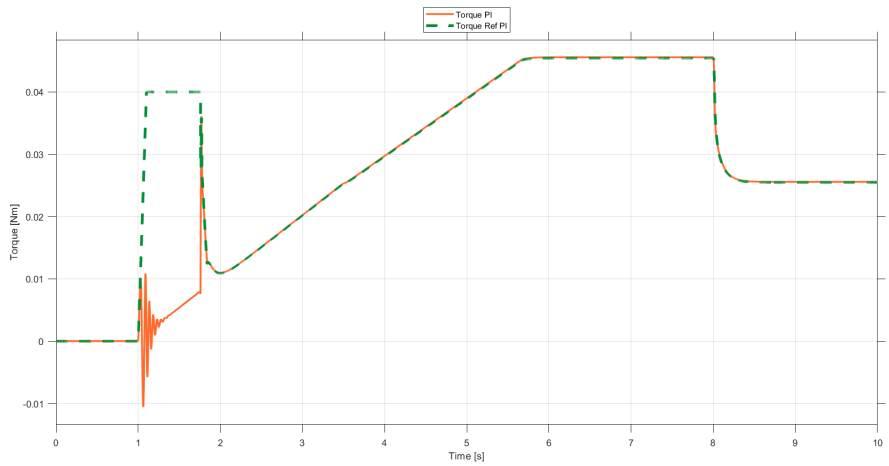

Figure 6.10: Torque PI

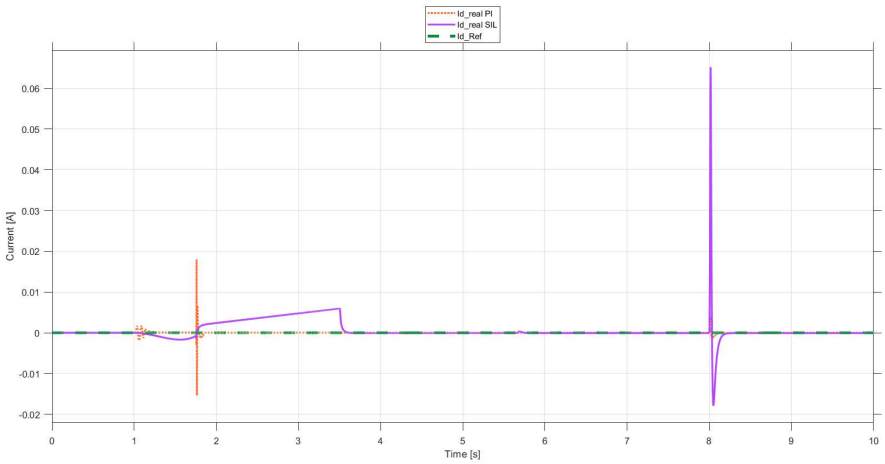

Figure 6.11: Id Current SIL and PI

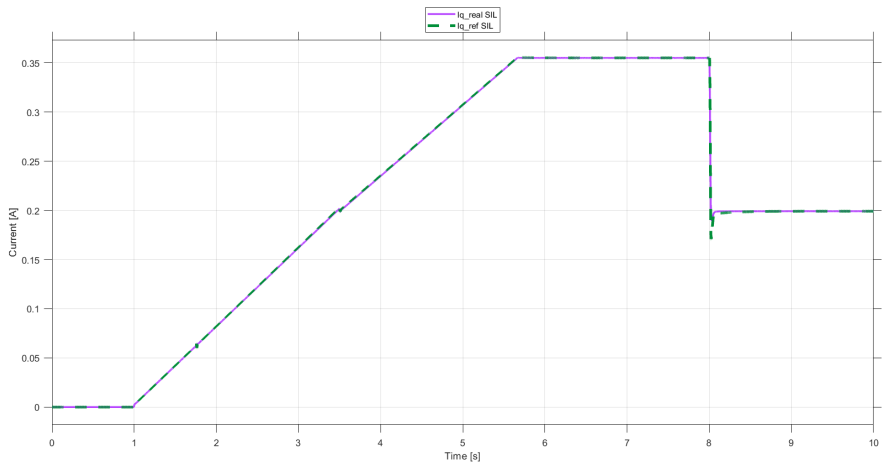

Figure 6.12: Iq Current SIL

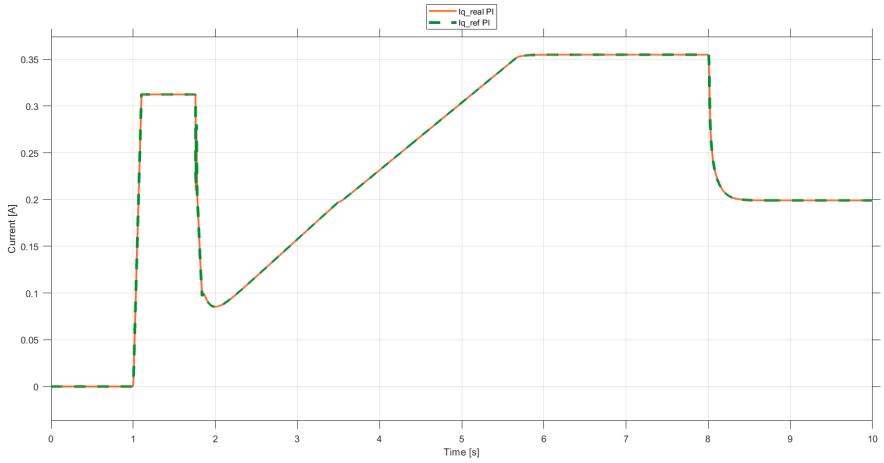

Figure 6.13: Iq Current PI

6.3 Graphical Results in presence of saturation

For completeness, graphical results are also reported in case of saturation.

Voltage Saturation

In order to test the behavior of the regulators when the variable DC_Bus is saturated, it was decided to insert a signal. Initially, the variable is set to 325 V. After 3 seconds, it drops to 60 V, and then after 2 seconds, it returns to 325 V. Below are the graphical results of the test. As expected, these graphic results are identical to those obtained in chapter 7 in the thesis [1].

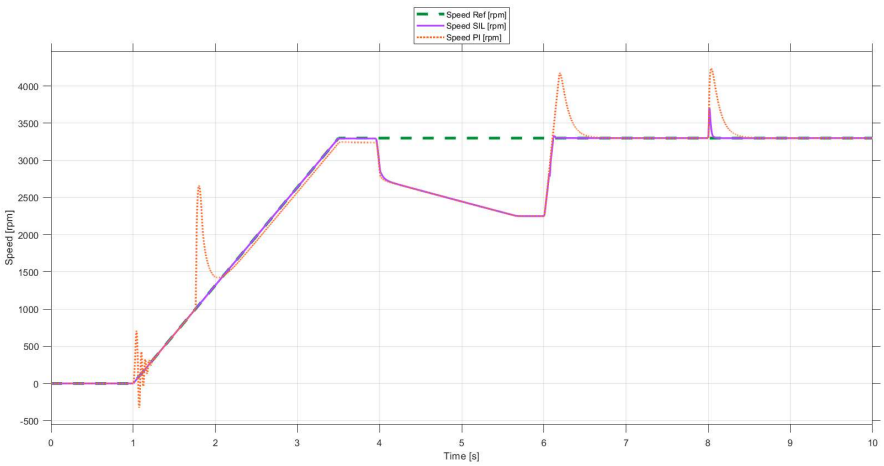

Figure 6.14: Speed SIL and PI with DC Bus signal

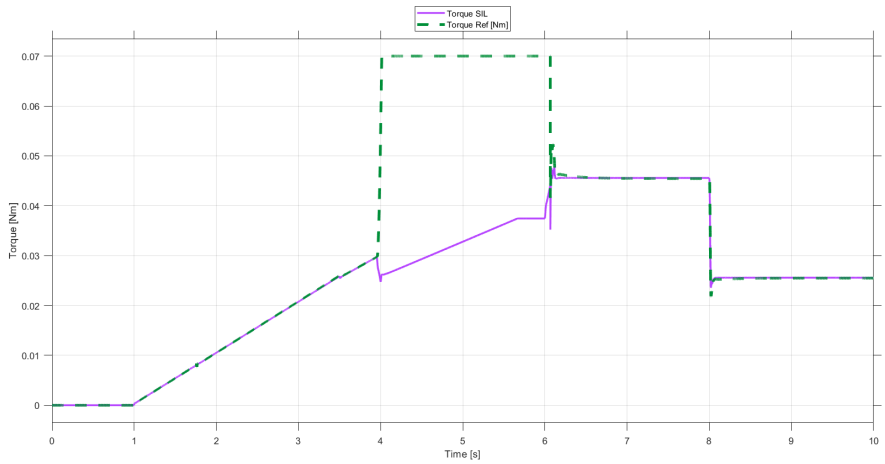

Figure 6.15: Torque SIL with DC Bus signal

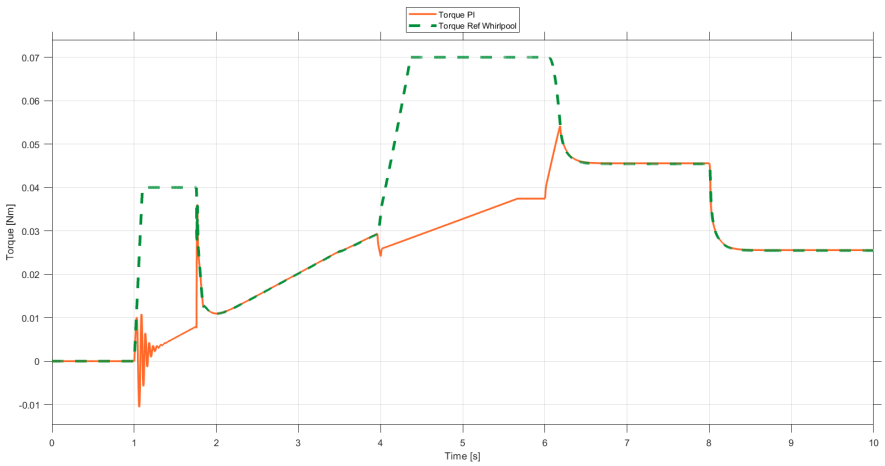

Figure 6.16: Torque PI with DC Bus signal

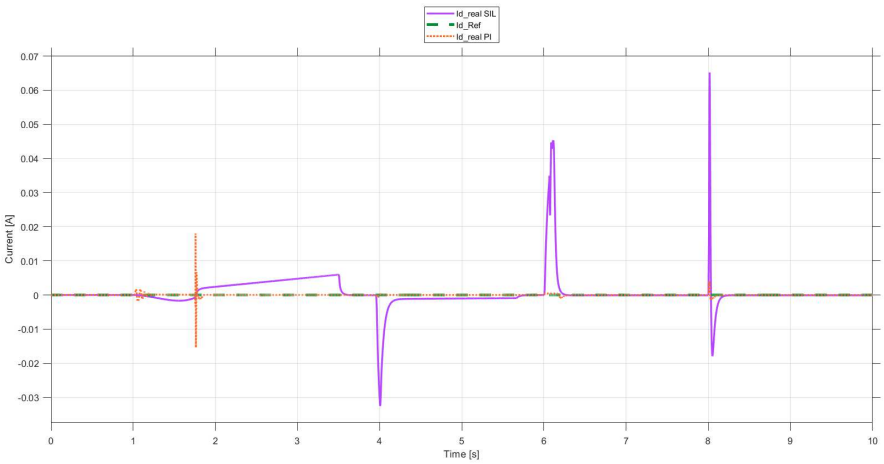

Figure 6.17: Id Current SIL and PI with DC Bus signal

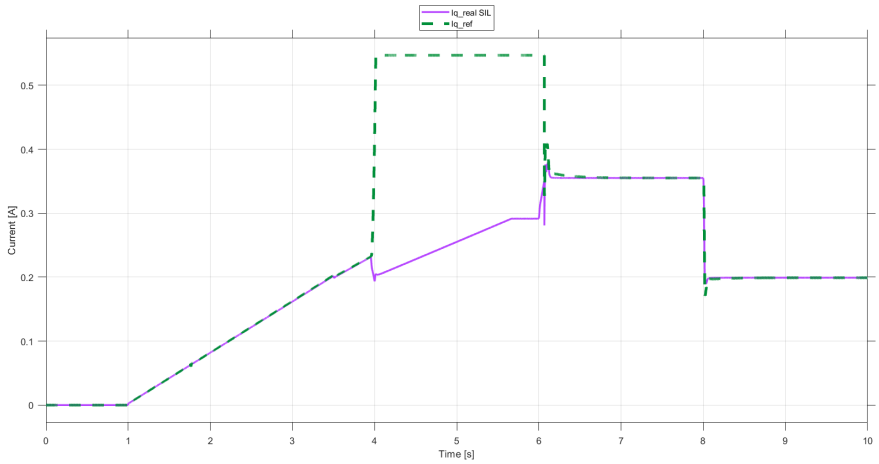

Figure 6.18: Iq Current SIL with DC Bus signal

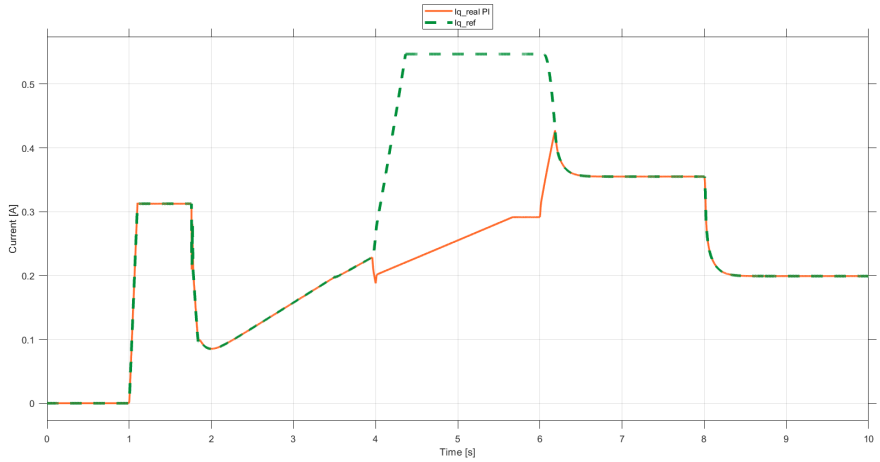

Figure 6.19: Iq Current PI with DC Bus signal

Torque Saturation

Torque saturation is achieved by setting the upper bound of the reference torque to 0.035 Nm instead of 0.07 Nm. As expected, these graphic results are identical to those obtained in chapter 7 in the thesis [1].

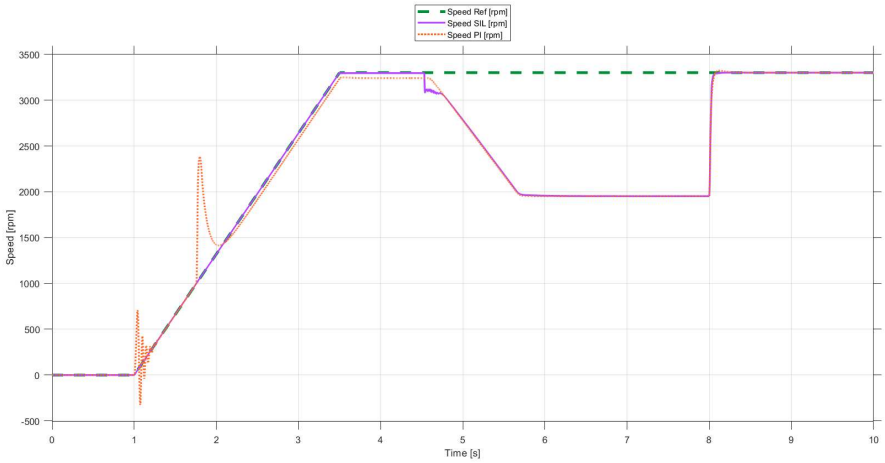

Figure 6.20: Speed SIL and PI with Te max=0.035 Nm

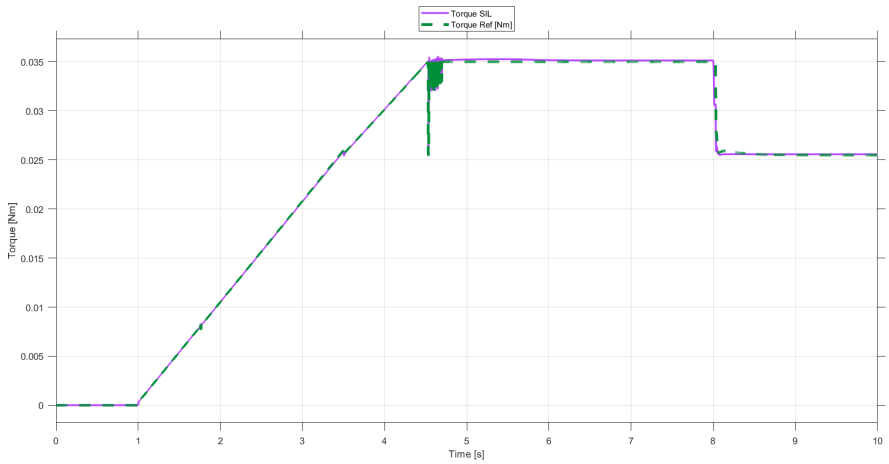

Figure 6.21: Torque SIL with Te max=0.035 Nm

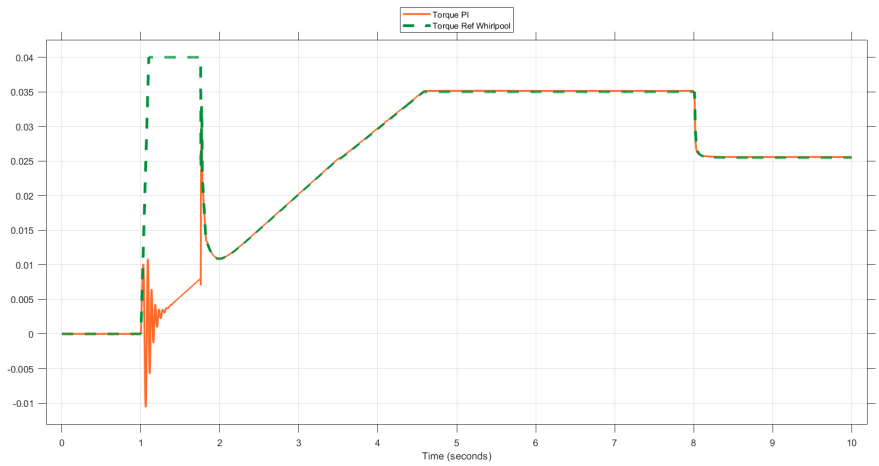

Figure 6.22: Torque PI with Te max=0.035 Nm

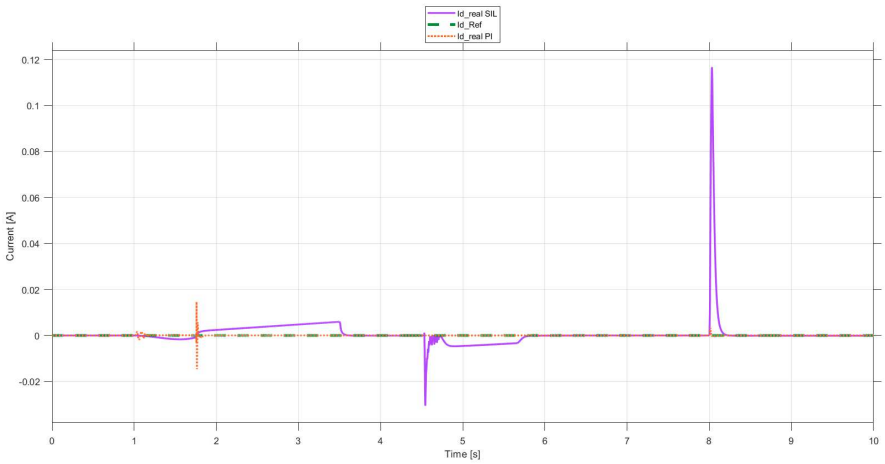

Figure 6.23: Current Id SIL and PI with Te max=0.035 Nm

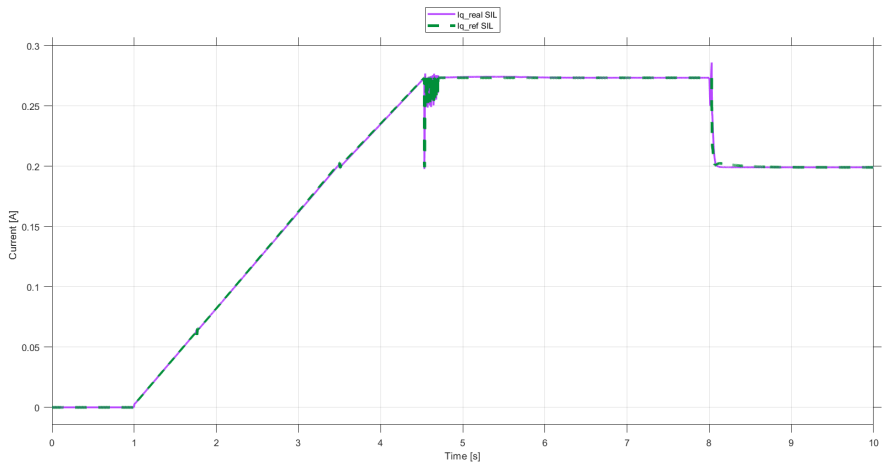

Figure 6.24: Current Iq SIL with Te max=0.035 Nm

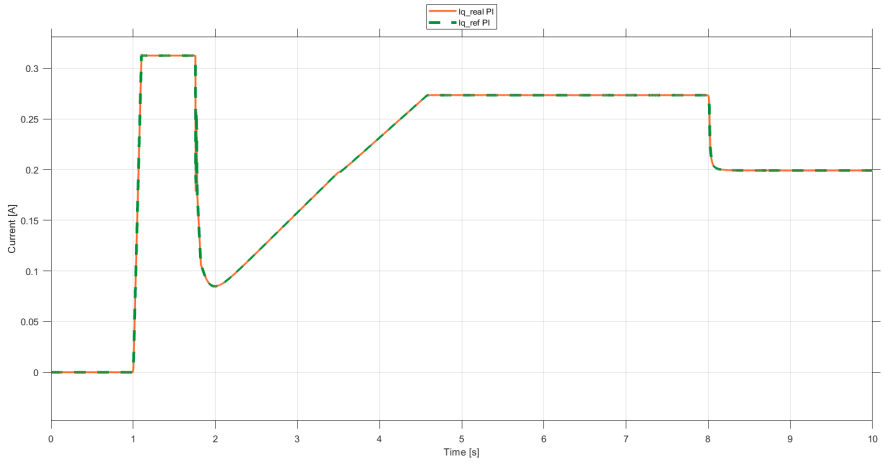

Figure 6.25: Current Iq PI with Te max=0.035 Nm

Torque and Voltage Saturation

This subsection contains the graphics obtained saturating both torque and voltage. As expected, these graphic results are identical to those obtained in chapter 7 in the thesis [1].

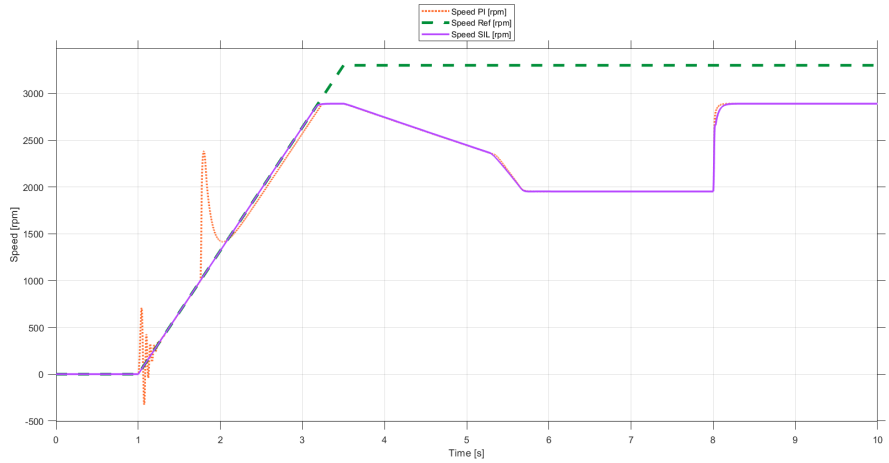

Figure 6.26: Speed SIL and PI with DC Bus=60 V and Te max=0.035 Nm

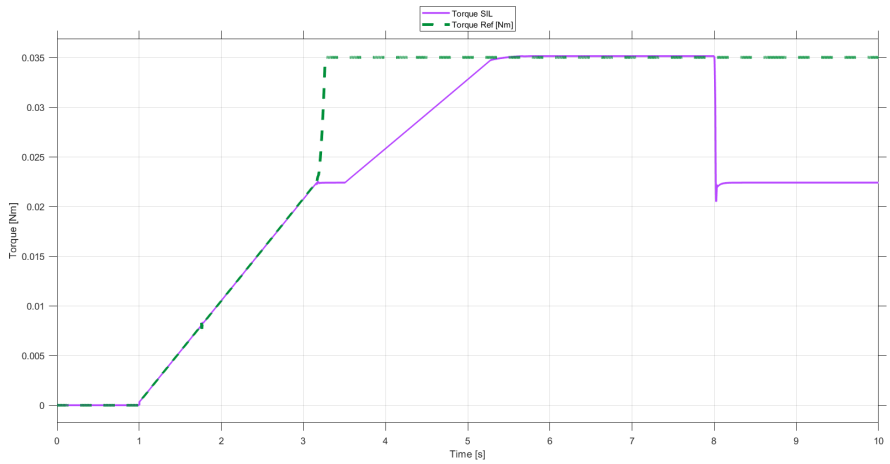

Figure 6.27: Torque SIL with DC Bus=60 V and Te max=0.035 Nm

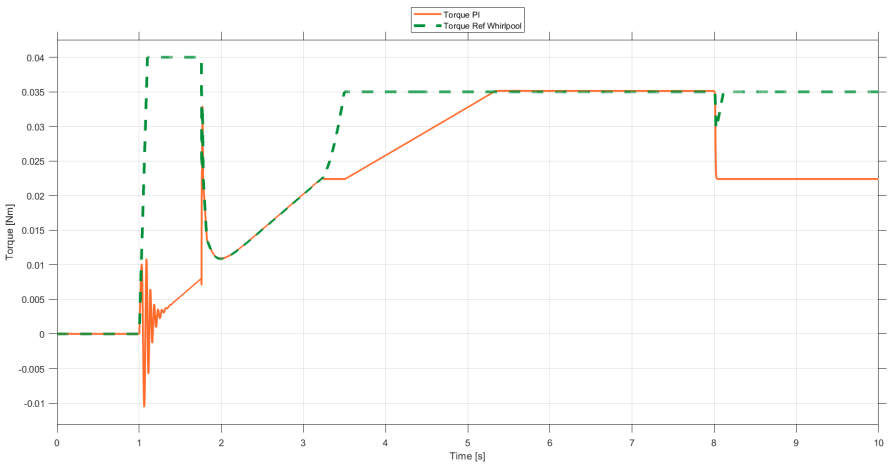

Figure 6.28: Torque PI with DC Bus=60 V and Te max=0.035 Nm

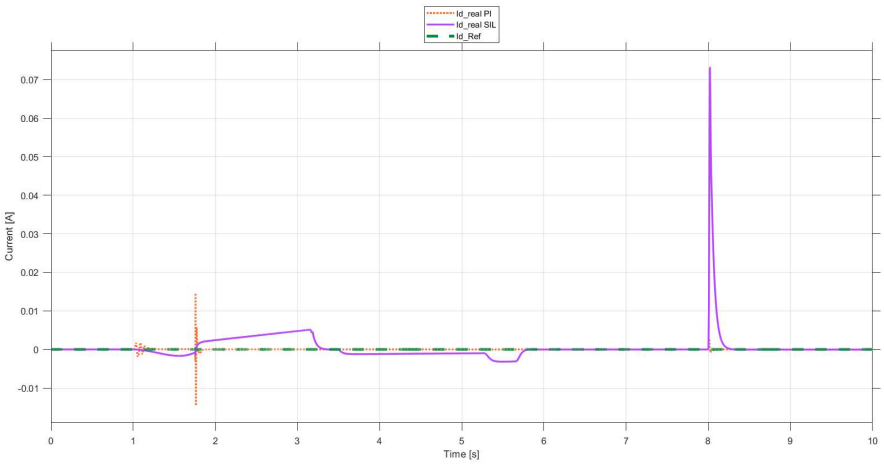

Figure 6.29: Current Id SIL and PI with DC Bus=60 V and Te max=0.035 Nm

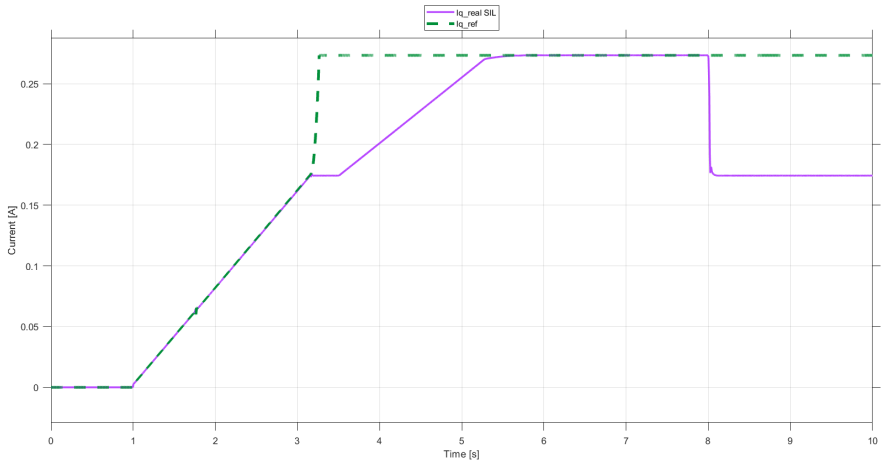

Figure 6.30: Current Iq SIL with DC Bus=60 V and Te max=0.035 Nm

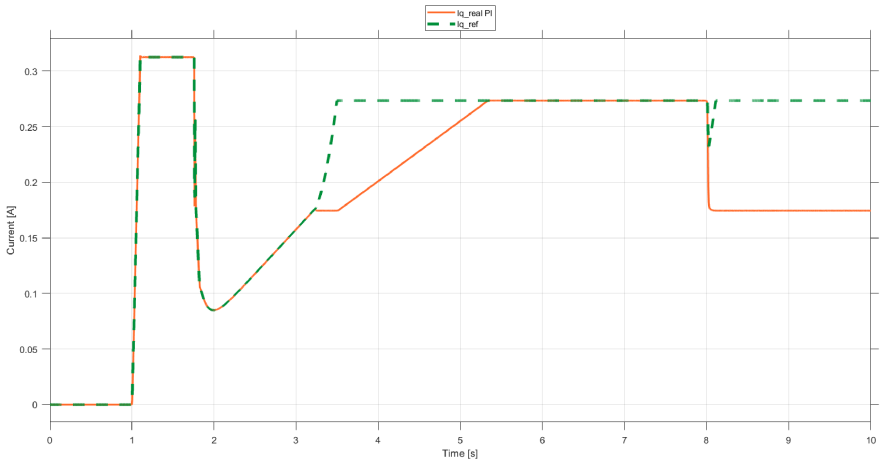

Figure 6.31: Current Iq PI with DC Bus=60 V and Te max=0.035 Nm

Thanks to the accuracy of the Whirlpool' Simulator, simulated performances are really close to the real ones. As the control implemented proved to be effective in simulation, the same performance can be guaranteed on a real system. Therefore, sliding mode can be considered as a potential alternative control solution to PI control.

6.4 Dishwasher' Implementation

The aim of this section is to present the actual implementation of the dishwasher. Firstly, it will describe the experimental setup, then it will demonstrate the graphical results obtained, and finally, it will compare the two control modes.

Figure 6.32: Dishwasher example

Figure 6.33: Dishwasher PM drain pump

Figure 6.32 shows the model of the dishwasher used for the experimental validation. Inside is mounted the dishwasher PM drain pump that interests this study, which is shown in figure 6.33. The dishwasher drains water with the control developed by Whirlpool and the work presented in this thesis.
In order to carry out various tests, it was necessary for the PC to connect to the dishwasher board shown in figure 6.34. This required the use of several devices. (6.35 and 6.36)

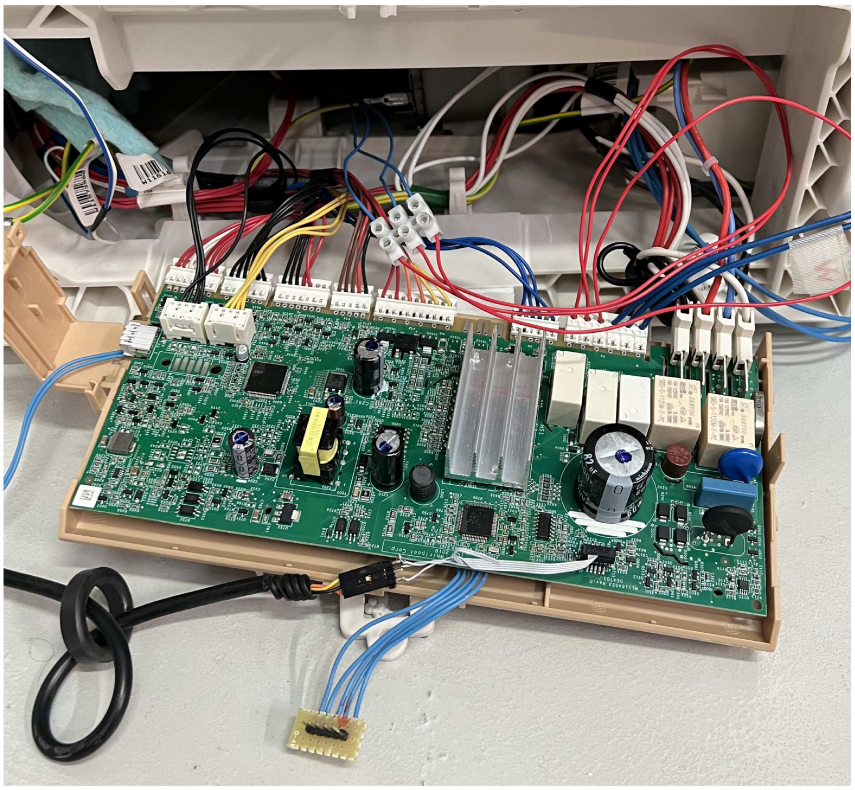

Figure 6.34: Dishwasher's board

Figure 6.35: Setup experimental

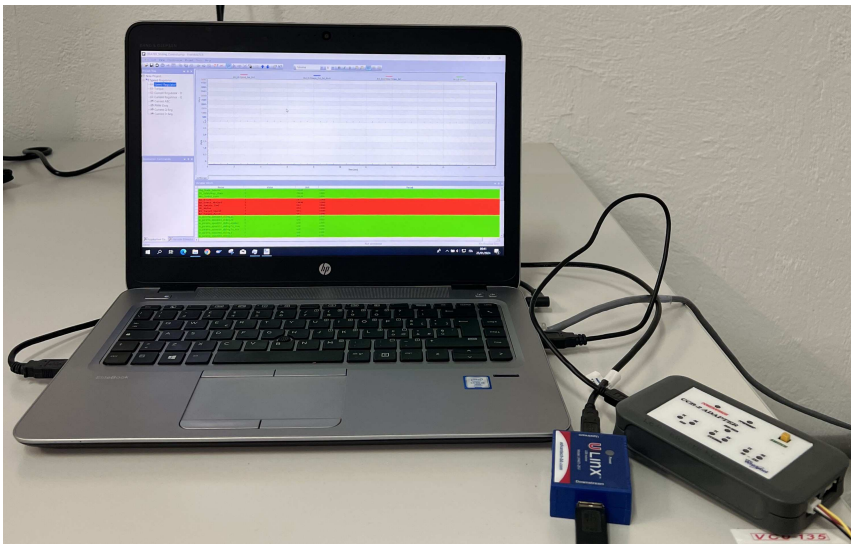

Figure 6.36: Setup experimental - close up

The first device used was the programmer (6.37), which was utilized to download the firmware onto the microcontroller of the board via a JTAG connection. The microcontroller is highlighted in the figure 6.38.

Figure 6.37: Debugger J-Link Arm

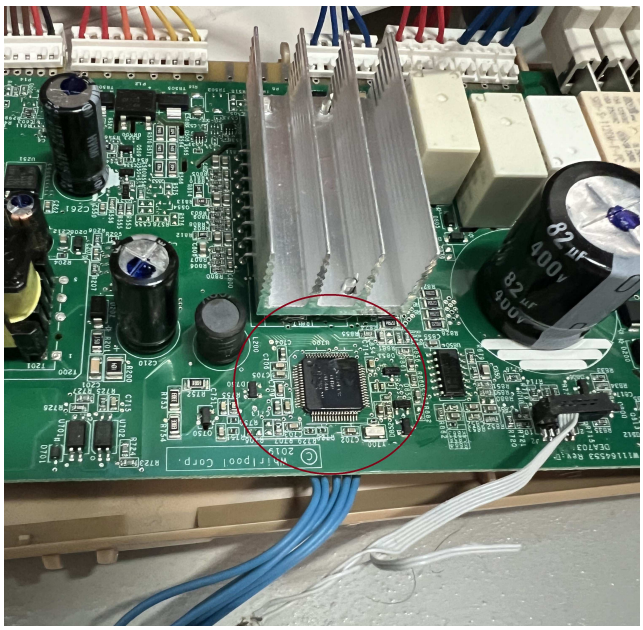

Figure 6.38: Dishwasher's Microcontroller

In addition, two other devices were used. The $U - Link$ device enabled communication between the PC and the microcontroller through the UART serial interface, allowing for the reading and writing of variables within the firmware.

The second device is entirely developed by Whirlpool and is used to manually drive the loads of the appliance, as well as acquire data from it. In this particular case, it was used to manually activate the solenoid valve for charging water.(6.39)

Figure 6.39: Serial trasmitter and Whirlpool's device

Two programs were installed on the PC to control the dishwasher board. The $MAC\ Dish$ software is used to operate the various dishwasher components through switches. In this case, the variable of interest is the $fill\_valve$, which activates or deactivates the solenoid valve to regulate the water flow inside the dishwasher. To ensure consistent measurements, a stopwatch was used to activate the solenoid valve for 20 seconds each time, ensuring a similar amount of water. The $FreeMaster$ tool was used to display and modify control parameters during this process. Through the panel displayed in the figure 6.40, it is possible to:

1. Change the type of controller (in variable $Dbg\_Mode\_Selector$ there is a drop-down menu);

2. Start the drain pump motor at a specific speed and acceleration;

3. Modify control and process parameters.

Variable Watch

Name	Value	Unit	
Dbg_Mode_Selector	DBG_CONTROLLER_SLIDING_FULL	ENUM	1000
SR_SafetyMngr_State	SAFETY_MNGR_RUNNING	ENUM	1000
Mci_Control_State	MCI_RUNNING	ENUM	1000
BD_Reset	**255**	**DEC**	**1000**
BD_Select_Method	**SELECT_SPEED**	**ENUM**	**1000**
BD_Update_Cmd	**0**	**DEC**	**1000**
BD_Motor	**1**	**DEC**	**1000**
BD_Target_Speed	**3300**	**DEC**	**1000**
BD_Target_Accel	**2000**	**DEC**	**1000**
io_params_speedctrl_sliding.a1	90	unit	1000
io_params_speedctrl_sliding.ro	0.08	unit	1000
io_params_speedctrl_sliding.epsilon	400	unit	1000
io_params_speedctrl_sliding.Te_max	0.07	unit	1000
io_params_speedctrl_sliding.Te_min	-0.01	unit	1000

Figure 6.40: FreeMaster Variable Watch

Please note that any changes or actions made within the *FreeMaster* panel must be followed by setting the *BD_Update_Cmd* value. To start the engine, you need to set the value of the variable *BD_Target_Speed* to "3300" and then give the command to set the above variable to 1. On the other hand, to stop the engine it is necessary to set the variable *BD_Target_Speed* to zero and give the command.

Steps needed to acquire measurements:

1. Charging water in the dishwasher with the *MAC Dish*;
2. Enable the *FreeMaster* data log;
3. Start the engine by operating the *FreeMaster* comand;
4. Once the water has been discharged, turn off the engine and deactivate the data log.

For completeness are reported in the next figures both the graphs obtained by FreeMaster and those obtained using the data collected with FreeMaster, plotted through a Matlab script. Each figure has two squares, one for each variable being considered.
In figure 6.41 displays the speed and torque trends along with their references in the case of the SIL Sliding Mode regulator. It can immediately observe how the speed's trend closely follows the reference and stabilizes at the set speed without any over-elongation.
Regarding the torque, there are three distinct situations: Firstly, the graph shows a trend similar to a parabolic arc, followed by a stabilization phase. Finally, there is a decline and recovery phase, which is related to the complete discharge of water.
Both graphs show a final phase where the constant value drops to zero using a step, indicating that the engine has been turned off.

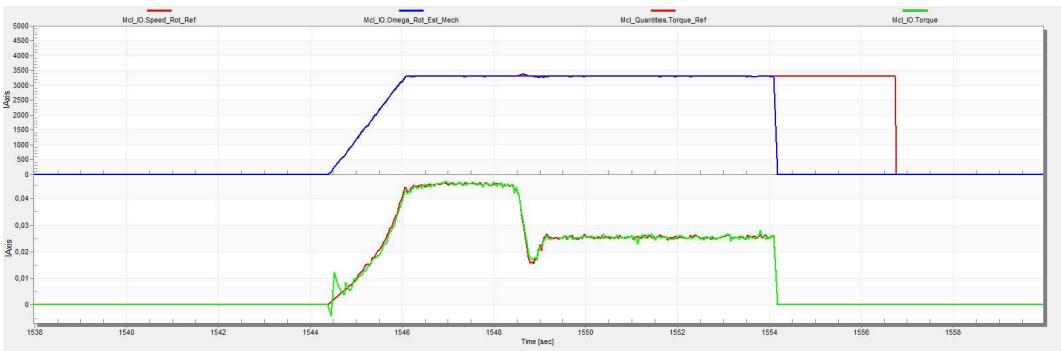

Figure 6.41: Torque and Speed SIL - FreeMaster

When comparing the figure described earlier with the one obtained using a PI controller (6.42), there are some noticeable differences. Although the speed follows

the reference throughout, there are two noticeable overshoots. However, the torque does not overshoot and maintains a concave trend, similar to the trend observed in the SIL case due to the action of water. Following this, there is a decrease linked to the final discharge of water, and it eventually stabilizes at a constant value.

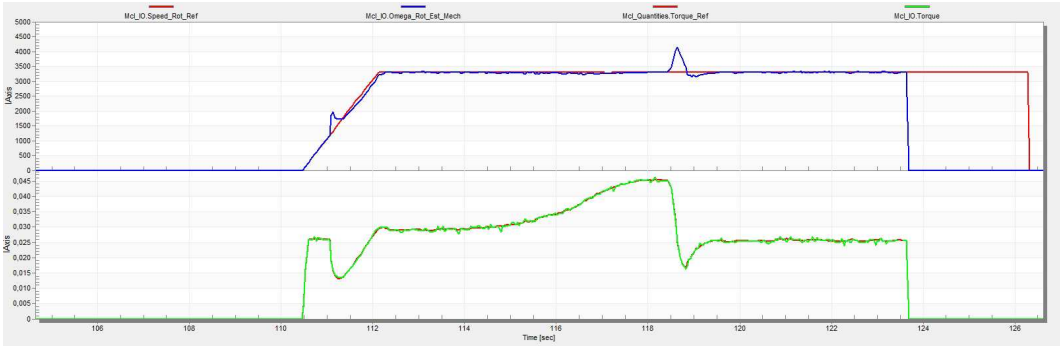

Figure 6.42: Torque and Speed PI - FreeMaster

Figures 6.43 and 6.44 show the trends of Iq currents and Vq voltages obtained respectively with the two types of control. It is visible how the graphs of the Iq are identical to those of the torque, as obtained proportionally.

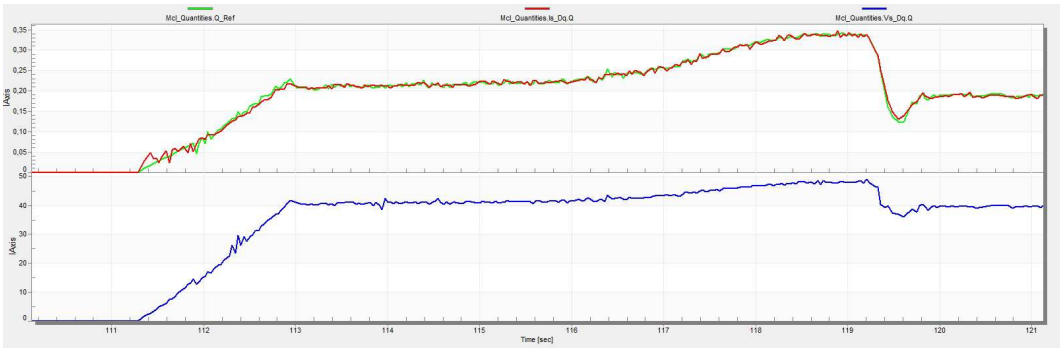

Figure 6.43: Current Iq and Voltage Vq SIL - FreeMaster

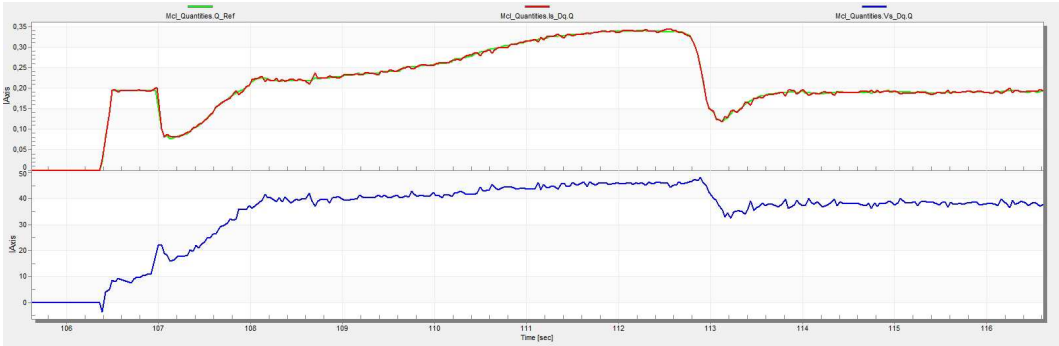

Figure 6.44: Current Iq and Voltage Vq PI - FreeMaster

Regarding the current Id, the PI regulator exhibits oscillations, but they are centered around zero which makes them tolerable. However, the SIL regulator signal shows oscillations with a deviation from zero. (6.46 and 6.45)

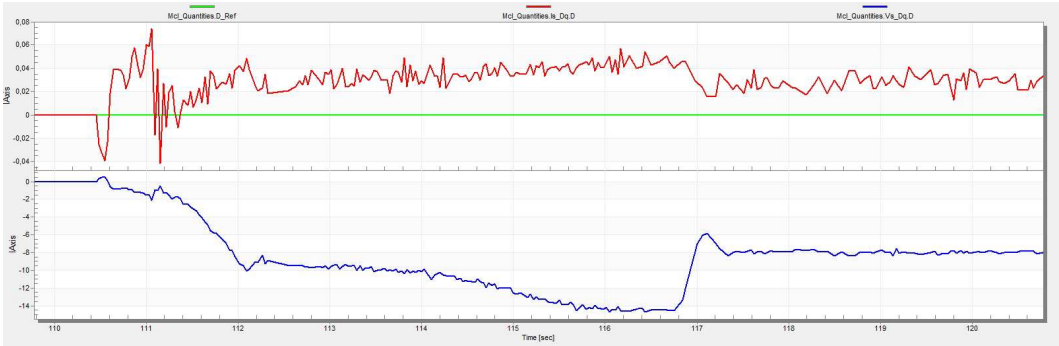

Figure 6.45: Current Id and Voltage Vd SIL - FreeMaster

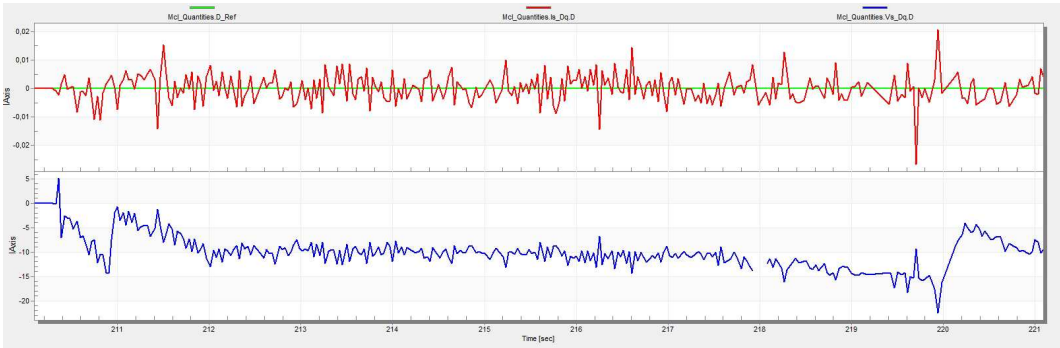

Figure 6.46: Current Id and Voltage Vd PI - FreeMaster

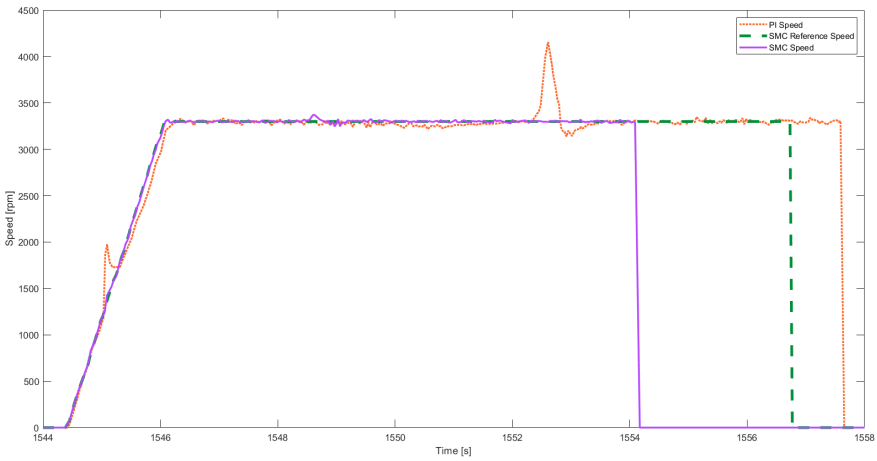

Figure 6.47: Speed SIL and PI - Log

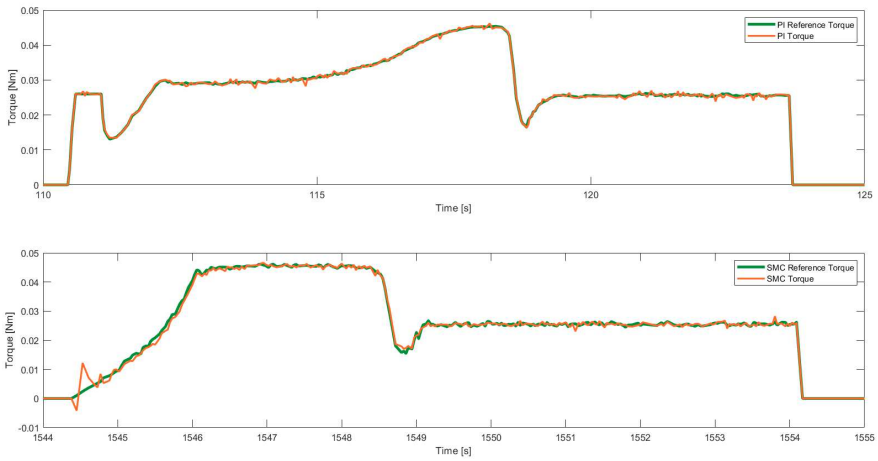

Figure 6.48: Torque SIL and PI - Log

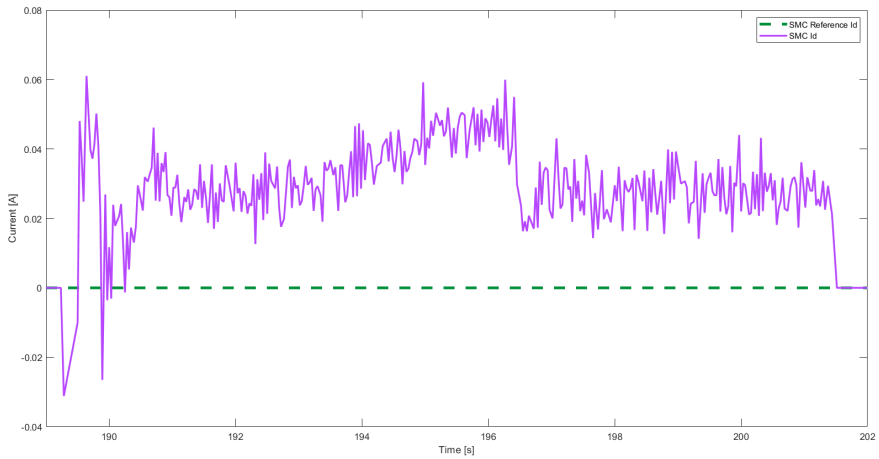

Figure 6.49: Current Id SIL - Log

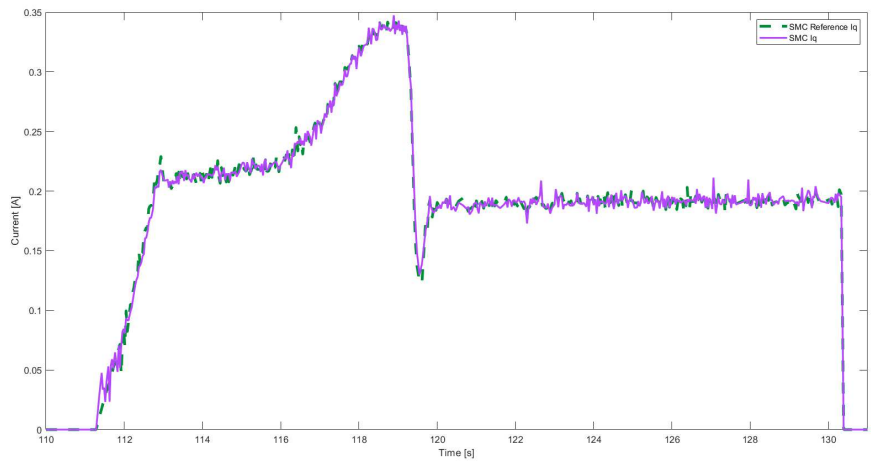

Figure 6.50: Current Iq SIL - Log

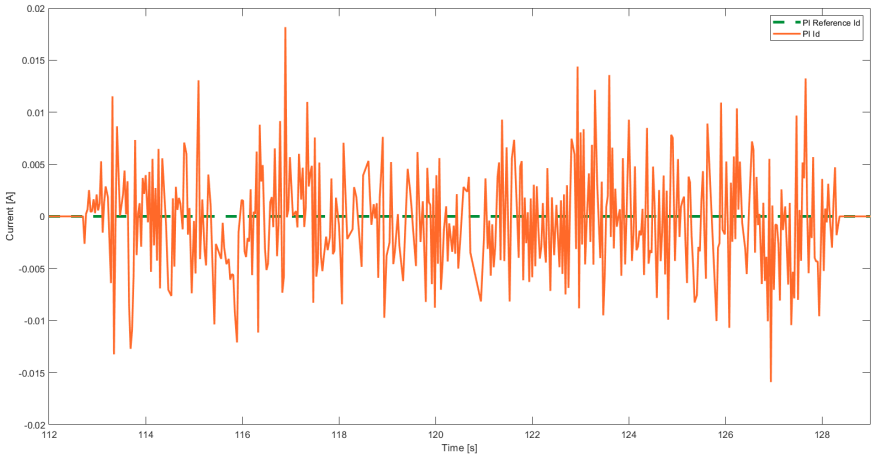

Figure 6.51: Current Id PI - Log

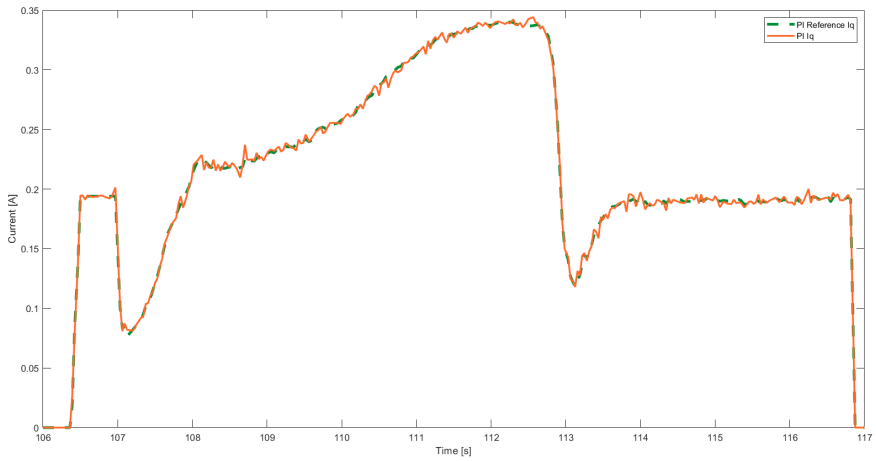

Figure 6.52: Current Iq PI - Log

6.5 Conclusion

The real implementation of both types of control - SMC and PI - has produced some important results.
Firstly, both types of control have maintained their characteristics and performance. The graphs generated through the simulation code are identical to those obtained in the experimental case.

Certainly, the SMC controller has shown better readiness than the PI control. Additionally, it is worth noting that, even when charging the same amount of water, the PI controller requires more time to drain than the SMC controller.
To evaluate the control quality, two performance indexes have been calculated in chapter 7. These indexes are based on the graphs representing both the simulated and real processes.

Chapter 7

Robustness Test

7.1 robustness_test.m

To automatically perform robustness tests, the last part of the script outlined in chapter 3 needs to be uncommented. It creates a table that contains the IAE and MSE values obtained. It then makes the thirty-two tests, populating the table at each iteration, which then is written into an Excel file. The valorization of the parameters for each test is performed by the "robutness_test.m" function: it assigns maximum and minimum values for each parameter by reading from a given matrix.

```
%%%%%%%%%%%%%% Automatic Robustness Tests %%%%%%%%%%%%%%%%%%
tableData = table('Size', [0, 2],'VariableTypes', {'double', '
    double'}, 'VariableNames', {'IAE', 'MSE'});

for i = 1:32
    [Rs,Ld,Lq,Phim,J,B] = robustness_test(i);
                                                % -1       % 1
    Motor_prm.Electrical.Rs_0= Rs;              %40.2686   61.5965
    Motor_prm.Electrical.Ld_0= Ld;              %0.1116    0.1284
    Motor_prm.Electrical.Lq_0= Lq;              %0.1116    0.1284
    Motor_prm.Electrical.PM_flux_h1_0= Phim;    %0.0673    0.0939
    Motor_prm.Mechanical.B= B;                  %7.03      7.77
    Motor_prm.Mechanical.Jrotor= J;             %2.02      2.24

sim('MCU_Simulation_Architecture_Sliding_CodeGen_13_12_2023.slx
    ',10);

    newRow = table(IAE, MSE, 'VariableNames', {'IAE', 'MSE'});
    tableData = [tableData; newRow];
end
 excelFileName = 'IAE_MSE.xlsx';
writetable(tableData, excelFileName, 'Sheet', 'Sheet1');
```

```
function [Rs,Ld,Lq,Phim,J,B] = robustness_test(i)

%matrice
range = 'A2:E33';
```

```
[matrice]=xlsread('RobustnessTest.xlsx',range);
%disp(matrice);

a=matrice(i,1);
b=matrice(i,2);
c=matrice(i,3);
d=matrice(i,4);
e=matrice(i,5);

if a==1
    Rs= 61.5965;
else
    Rs= 40.2686;
end

if b==1
    Ld= 0.1284;
    Lq= 0.1284;
else
    Ld= 0.1116;
    Lq= 0.1116;
end

if c==1
    Phim=0.0939;
else
    Phim=0.0673;
end

if d==1
    J= 2.24e-06;
else
    J= 2.02e-06;
end

if e==1
    B= 7.77e-05;
else
    B= 7.03e-05;
end

end
```

7.2 Test Parameters

Robustness means "the degree to which a system or component can work properly in the presence of invalid or stressful inputs and environmental conditions".[4] Robustness testing is a type of testing that is used to solicit a system in exceptional situations and to understand how it can react. The performed tests have impacted the variation of certain parameters, which are outlined in the table 7.1. It is important to note that all parameters are tested at their boundary values in a combination called Design of Experiments (DOE). (se serve definizione DOE: DOE is a method for designing and executing experiments using statistical analysis to find relationships between input variables and output variables.) This means that the robustness of the regulator is tested in extreme scenarios where it is unlikely that all values will be at the limit in a real-world situation.

Parameter	Nominal Value	Min (10°C)	Max (100°C)	Unit Measure
Rs	45.5	40.2686	61.5965	Ohm
Ld	120	111.6	128.4	mH
Lq	120	111.6	128.4	mH
Phim	0.0857	0.0673	0.0939	Vpk/rad/s
J	2.13E-06	2.02E-06	2.24E-06	Kg*m$\hat{2}$
B	7.40E-05	7.03E-05	7.77E-05	Nm/ rad/s

Table 7.1: Parameters

- Rs is the stator resistance;
- Ld and Lq are th inductance;
- Phim is the electromagnetic flux;
- J is the inertia rotor;
- B is the friction coefficient;

Rs and *Phim* values are related to physics and temperature, not just the process; while others are only tied to the process. Temperature, as showed in the table 7.1, can fluctuate from 10°C to 100°C.

The table 7.2 contains values of either -1 or 1. These values indicate whether the corresponding parameter is at its minimum or maximum value, respectively. Since there are five parameters that vary, there will be 32 tests to conduct, equal to the number of rows in the table.

To quantify the goodness of the tests, two performance indices have been considered: the Integral of the Absolute Error (IAE) and the Mean Squared Error (MSE). these indices allow to compare PI robustness to SMC one.

Rs	Ls	Phim	J	B
1	1	-1	1	1
1	1	1	-1	1
-1	-1	-1	-1	1
1	-1	1	1	-1
-1	-1	-1	1	-1
1	1	-1	-1	1
-1	1	-1	1	1
-1	1	1	1	-1
1	-1	1	1	1
-1	1	-1	1	-1
-1	1	-1	-1	1
-1	-1	-1	1	1
-1	1	-1	-1	-1
-1	1	1	1	1
1	-1	-1	1	1
1	-1	1	-1	1
1	-1	1	-1	-1
1	1	1	1	1
1	-1	-1	-1	-1
1	1	1	1	-1
-1	-1	1	1	-1
-1	1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	1
-1	-1	-1	-1	-1
-1	-1	1	1	1
1	-1	-1	-1	1
-1	1	1	-1	1
-1	-1	1	-1	-1
1	1	-1	1	-1
1	-1	-1	1	-1
1	1	-1	-1	-1

Table 7.2: DOE - Design of Experiment

The tests were performed inside the simulator in sensorless mode.

7.3 Results

IAE Nominal S	MSE Nominal S	IAE Sensorless	MSE Sensorless
2.269	1.49E-08	4351	2.75E+05
2.269	1.49E-08	4.318	1.13E-07
2.269	1.49E-08	4424	2.91E+05
2.269	1.49E-08	3.835	9.31E-08
2.269	1.49E-08	3.18	2.38E-07
2.269	1.49E-08	4348	2.75E+05
2.269	1.49E-08	5.039	2.38E-07
2.269	1.49E-08	2.583	3.73E-09
2.269	1.49E-08	5.276	5.96E-08
2.269	1.49E-08	3.18	2.38E-07
2.269	1.49E-08	4424	2.91E+05
2.269	1.49E-08	5.039	2.38E-07
2.269	1.49E-08	3.087	3.02E-07
2.269	1.49E-08	3.015	3.36E-07
2.269	1.49E-08	4351	2.75E+05
2.269	1.49E-08	4.318	1.13E-07
2.269	1.49E-08	4.386	9.31E-08
2.269	1.49E-08	5.276	5.96E-08
2.269	1.49E-08	6.146	1.13E-07
2.269	1.49E-08	3.835	9.31E-08
2.269	1.49E-08	2.583	3.73E-09
2.269	1.49E-08	2.608	9.31E-08
2.269	1.49E-08	4.386	9.31E-08
2.269	1.49E-08	4.28	4.51E-07
2.269	1.49E-08	3.087	3.02E-07
2.269	1.49E-08	3.015	3.36E-07
2.269	1.49E-08	4348	2.75E+05
2.269	1.49E-08	4.28	4.51E-07
2.269	1.49E-08	2.608	9.31E-08
2.269	1.49E-08	8.069	7.54E-08
2.269	1.49E-08	8.069	7.54E-08
2.269	1.49E-08	6.146	1.13E-07

Table 7.3: Test results Sliding Mode robustness

The table 7.3 reports the results of the experiment, which indicate 6 KO.

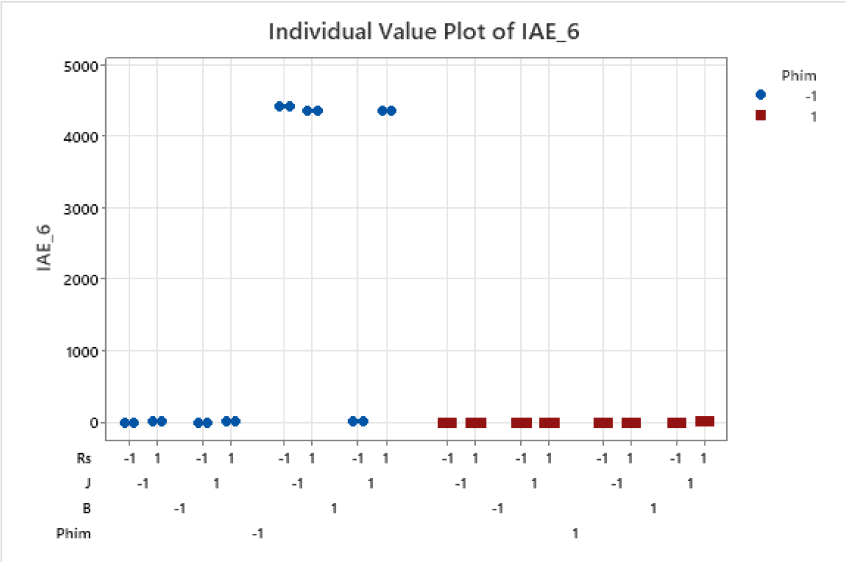

Figure 7.1: IAE - 6 KO

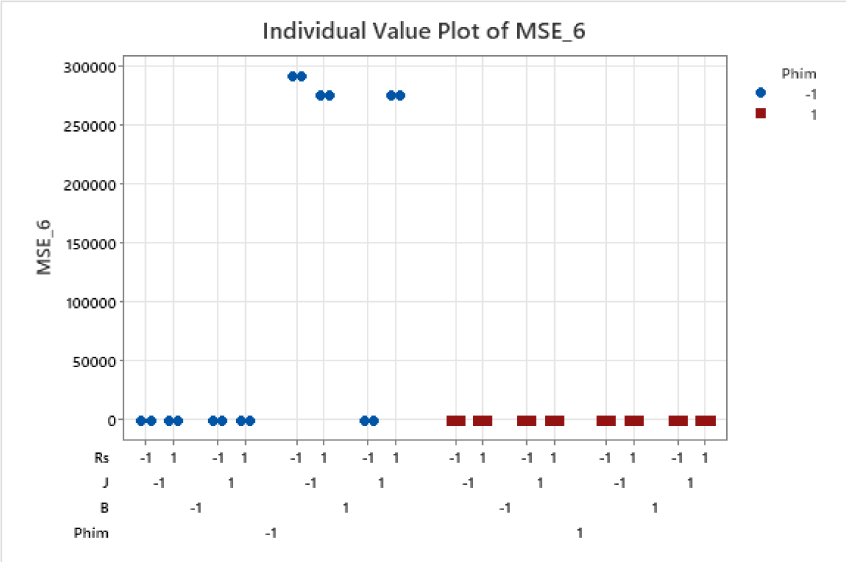

Figure 7.2: MSE - 6 KO

Graphs 7.2 and 7.1 reveals that these knockouts share a common pattern of parameter variation. Specifically, it appears that the regulator is most sensitive to three parameters when they are set to their extremes:

- Phim set to the minimum;
- Rs set to the maximum (in 4/6 KOs);
- B set to the maximum.

On the other hand, the variation of the inductance has no effects overall. When analyzing different negative tests, it was noticed that there was a similarity between them. To address this issue and decrease the likelihood of robustness failure, the parameters Rs and B were set to their maximum value, while the parameter $Phim$ was set to the minimum value, within the regulators Sliding Mode. This led to a new tuning of the regulator parameters, in order to achieve a balance between control performance and robustness.

IAE Nominal	MSE Nominal	IAE Sensorless	MSE Sensorless
2.295	4.56E-08	6.215	7.21E-04
2.295	4.56E-08	5.891	2.33E-08
2.295	4.56E-08	3.588	2.38E-07
2.295	4.56E-08	4.473	9.31E-08
2.295	4.56E-08	3.819	1.34E-07
2.295	4.56E-08	4.797	6.46E-04
2.295	4.56E-08	3.558	1.83E-07
2.295	4.56E-08	2.309	1.23E-04
2.295	4.56E-08	4619	3.16E+05
2.295	4.56E-08	3.819	1.34E-07
2.295	4.56E-08	3.588	2.38E-07
2.295	4.56E-08	3.558	1.83E-07
2.295	4.56E-08	3.834	7.54E-08
2.295	4.56E-08	2.435	1.30E-04
2.295	4.56E-08	6.215	7.21E-04
2.295	4.56E-08	5.891	2.33E-08
2.295	4.56E-08	3.18	9.31E-08
2.295	4.56E-08	4619	3.16E+05
2.295	4.56E-08	4813	3.29E+05
2.295	4.56E-08	4.473	9.31E-08
2.295	4.56E-08	2.309	1.23E-04
2.295	4.56E-08	2.35	1.19E-04
2.295	4.56E-08	3.18	9.31E-08
2.295	4.56E-08	4681	3.29E+05
2.295	4.56E-08	3.834	7.54E-08
2.295	4.56E-08	2.435	1.30E-04
2.295	4.56E-08	4.797	6.46E-04
2.295	4.56E-08	4681	3.29E+05
2.295	4.56E-08	2.35	1.19E-04
2.295	4.56E-08	5.137	7.61E-04
2.295	4.56E-08	5.137	7.61E-04
2.295	4.56E-08	4813	3.29E+05

Table 7.4: Tests obtained by fixing Rs max, B max and Phim min with new tuning

Table 7.4 indicates that, even after aforementioned implementing changes, the number of KO's remains at 6. It's worth noting that the negative test results obtained are different from those obtained with the previous configuration (7.3).
Graphs 7.3 and 7.4 reveal that the "new" KOs exhibit a pattern of parameter variation that is no longer consistent.

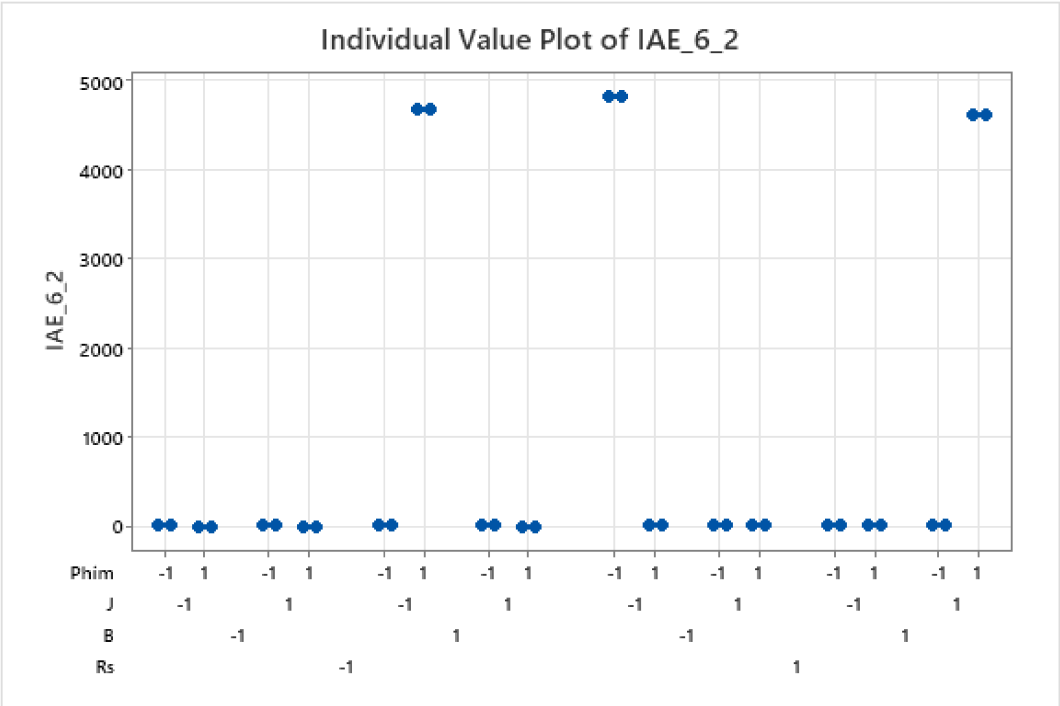

Figure 7.3: IAE - 6 KO

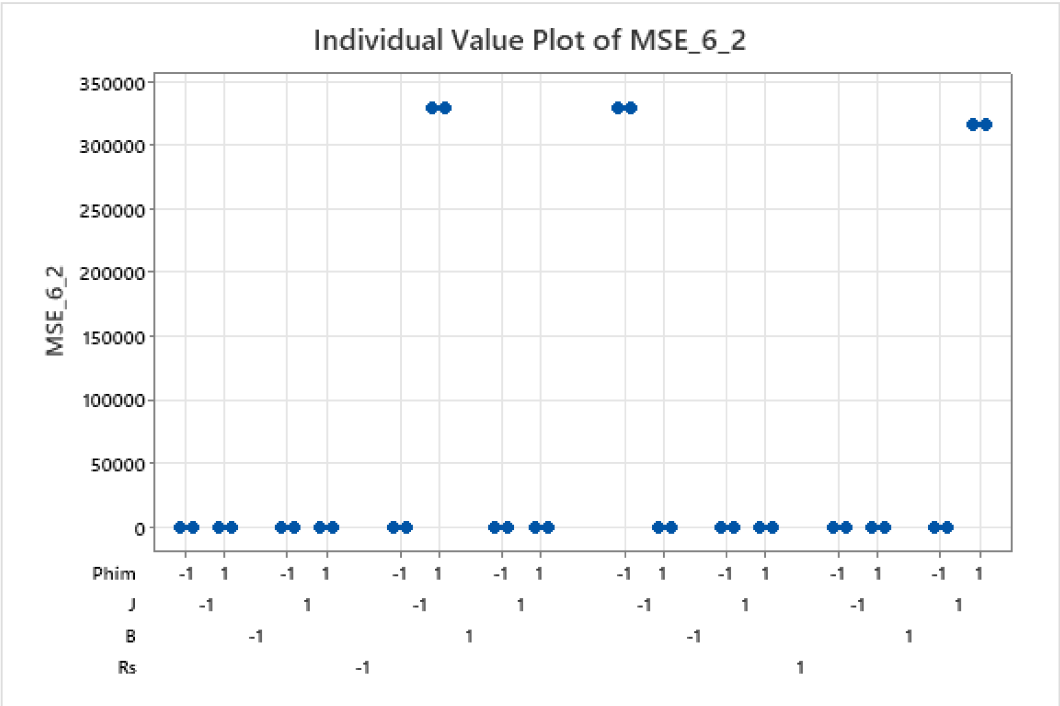

Figure 7.4: MSE - 6 KO

As the previous optimization has not shown any improvement, the first parameters' tuning has been restored. The control variable *Phim* was set to a lower value than the minimum, therefore the *K_Torque* value changes proportionally to *Phim*. In particular, the *Phim* value is fixed at 0.0580 Vpk/rad/s and a *K_Torque* value of 0.087 Nm/A is obtained. Robustness tests will be conducted again by varying the process parameters.

IAE Nominal	MSE Nominal	IAE Sensorless	MSE Sensorless
3.000912	3.36E-07	4.74E+03	3.23E+05
3.000912	3.36E-07	4.65E+00	1.57E-07
3.000912	3.36E-07	2.92E+00	6.79E-07
3.000912	3.36E-07	4.60E+03	3.15E+05
3.000912	3.36E-07	3.10E+00	7.54E-08
3.000912	3.36E-07	5.50E+00	9.31E-08
3.000912	3.36E-07	2.79E+00	1.57E-07
3.000912	3.36E-07	3.59E+00	4.93E-07
3.000912	3.36E-07	2.90E+00	1.34E-07
3.000912	3.36E-07	3.10E+00	7.54E-08
3.000912	3.36E-07	2.92E+00	6.79E-07
3.000912	3.36E-07	2.79E+00	1.57E-07
3.000912	3.36E-07	3.24E+00	1.83E-07
3.000912	3.36E-07	3.49E+00	6.30E-07
3.000912	3.36E-07	4.74E+03	3.23E+05
3.000912	3.36E-07	4.65E+00	1.57E-07
3.000912	3.36E-07	4.25E+00	1.57E-07
3.000912	3.36E-07	2.90E+00	1.34E-07
3.000912	3.36E-07	5.57E+00	7.54E-08
3.000912	3.36E-07	4.60E+03	3.15E+05
3.000912	3.36E-07	3.59E+00	4.93E-07
3.000912	3.36E-07	4.43E+00	4.93E-07
3.000912	3.36E-07	4.25E+00	1.57E-07
3.000912	3.36E-07	4.23E+00	9.54E-07
3.000912	3.36E-07	3.24E+00	1.83E-07
3.000912	3.36E-07	3.49E+00	6.30E-07
3.000912	3.36E-07	5.50E+00	9.31E-08
3.000912	3.36E-07	4.23E+00	9.54E-07
3.000912	3.36E-07	4.43E+00	4.93E-07
3.000912	3.36E-07	4.39E+00	1.83E-07
3.000912	3.36E-07	4.39E+00	1.83E-07
3.000912	3.36E-07	5.57E+00	7.54E-08

Table 7.5: Tests obtained by fixing Phim value inside the regulator

Applied changes produced the results in table 7.5. As displayed KO's number is reduced to 4. In figure 7.5 and 7.6, it is evident that the regulator is sensitive to maximum values of Rs and J.

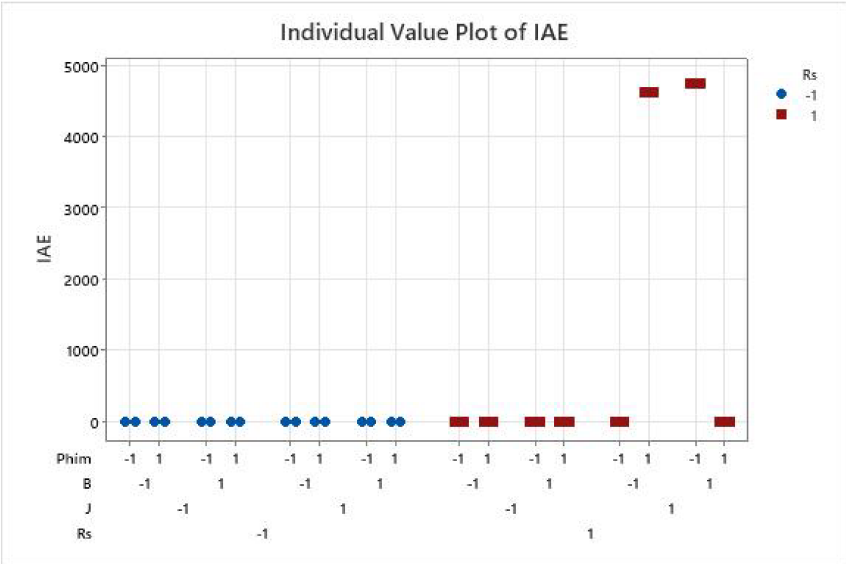

Figure 7.5: IAE - 4 KO

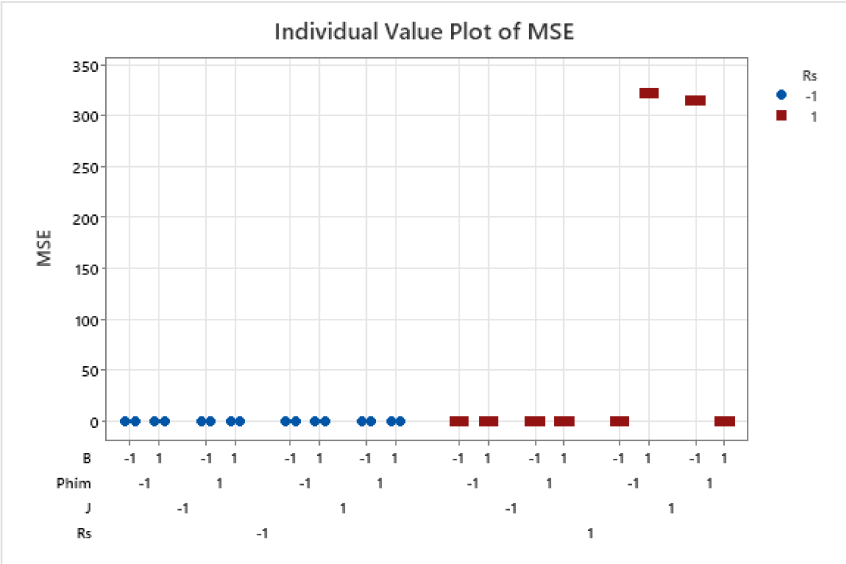

Figure 7.6: MSE - 4 KO

IAE Nominal	MSE Nominal	IAE	MSE
13.1342268	2.38E-07	13.7096	8.38E-09
13.1342268	2.38E-07	12.3994	2.10E-07
13.1342268	2.38E-07	13.7732	1.49E-08
13.1342268	2.38E-07	12.9721	4.51E-07
13.1342268	2.38E-07	14.6297	9.31E-08
13.1342268	2.38E-07	13.7289	9.31E-10
13.1342268	2.38E-07	13.7498	2.33E-08
13.1342268	2.38E-07	13.128	1.64E-06
13.1342268	2.38E-07	12.3974	1.83E-07
13.1342268	2.38E-07	14.6297	9.31E-08
13.1342268	2.38E-07	13.7732	1.49E-08
13.1342268	2.38E-07	13.7498	2.33E-08
13.1342268	2.38E-07	14.6503	1.57E-07
13.1342268	2.38E-07	12.5269	8.95E-07
13.1342268	2.38E-07	13.7096	8.38E-09
13.1342268	2.38E-07	12.3994	2.10E-07
13.1342268	2.38E-07	12.9722	6.79E-07
13.1342268	2.38E-07	12.3974	1.83E-07
13.1342268	2.38E-07	14.5792	0
13.1342268	2.38E-07	12.9721	4.51E-07
13.1342268	2.38E-07	13.128	1.64E-06
13.1342268	2.38E-07	13.1373	1.57E-06
13.1342268	2.38E-07	12.9722	6.79E-07
13.1342268	2.38E-07	12.5353	8.38E-07
13.1342268	2.38E-07	14.6503	1.57E-07
13.1342268	2.38E-07	12.5269	8.95E-07
13.1342268	2.38E-07	13.7289	9.31E-10
13.1342268	2.38E-07	12.5353	8.38E-07
13.1342268	2.38E-07	13.1373	1.57E-06
13.1342268	2.38E-07	14.5661	8.38E-09
13.1342268	2.38E-07	14.5661	8.38E-09
13.1342268	2.38E-07	14.5792	0

Table 7.6: Tests Robustness passed

The results displayed in table 7.6 indicate that the controller sliding mode has successfully passed all 32 robustness tests. To achieve this, certain control parameters were adjusted, including those of the speed and current regulators, as well as the K_Torque and $Phim$ variables. These adjustments were made through the tuning of:

- Speed regulator
  - a1=20;
  - ρ=0.018;
  - ε=130;

- Current Iq regulator
  - a2=500;
  - ρ_q=100;
  - ε_q=10;
- Current Id regulator
  - a3=500;
  - ρ_d=100;
  - ε_d=3000;
- K_Torque=0.09;
- $Phim$=0.06.

Table 7.7 shows the results of the robustness tests for PI regulators.

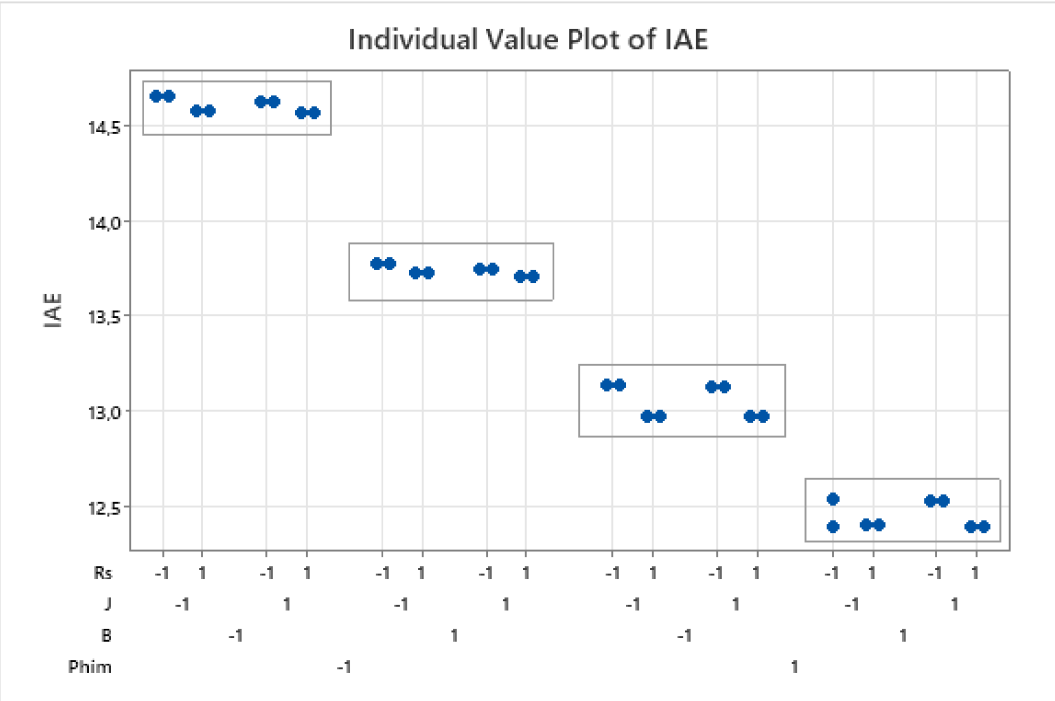

Figure 7.7: IAE - OK

IAE Nominal	MSE Nominal	IAE	MSE
56.4062	2.10E-07	64.4127	7.54E-08
56.4062	2.10E-07	53.8872	2.33E-08
56.4062	2.10E-07	64.4751	2.69E-07
56.4062	2.10E-07	53.7957	1.13E-07
56.4062	2.10E-07	64.2792	1.34E-07
56.4062	2.10E-07	64.3975	4.56E-08
56.4062	2.10E-07	64.5113	2.33E-08
56.4062	2.10E-07	53.8351	1.83E-07
56.4062	2.10E-07	53.9064	9.31E-10
56.4062	2.10E-07	64.2792	1.34E-07
56.4062	2.10E-07	64.4751	2.69E-07
56.4062	2.10E-07	64.5113	2.33E-08
56.4062	2.10E-07	64.2402	1.13E-07
56.4062	2.10E-07	53.9458	8.38E-09
56.4062	2.10E-07	64.4127	7.54E-08
56.4062	2.10E-07	53.8877	2.33E-08
56.4062	2.10E-07	53.7716	1.13E-07
56.4062	2.10E-07	53.9064	9.31E-10
56.4062	2.10E-07	64.1643	1.13E-07
56.4062	2.10E-07	53.7957	1.13E-07
56.4062	2.10E-07	53.8351	1.83E-07
56.4062	2.10E-07	53.8111	1.83E-07
56.4062	2.10E-07	53.7716	1.13E-07
56.4062	2.10E-07	53.9276	9.31E-10
56.4062	2.10E-07	64.2402	1.13E-07
56.4062	2.10E-07	53.9458	8.38E-09
56.4062	2.10E-07	64.3975	4.56E-08
56.4062	2.10E-07	53.9276	9.31E-10
56.4062	2.10E-07	53.8111	1.83E-07
56.4062	2.10E-07	64.1839	1.13E-07
56.4062	2.10E-07	64.1839	1.13E-07
56.4062	2.10E-07	64.1643	1.13E-07

Table 7.7: Tests Robustness PI

7.4 Real implementation

To quantify the goodness of the regulators, the IAE and MSE indices were calculated even for real-world implementation. Based on the data collected during the experiment described in chapter 6, IAE and MSE values were calculated for both SMC and PI controllers, as reported in the table 7.8.

	IAE	MSE
Sliding Mode Control	9.0893	2.0394
PI	64.8658	1.38E+02

Table 7.8: IAE and MSE in real implementation

7.5 Qualitative and quantitative evaluation of the conducted experiments

Generally, in most of KOs, Rs is set to its maximum value. In both tests with 6 KOs, Rs is set to its maximum in 4/6 cases. Changing control K_Torque according to control $Phim$ to lower values then their minimum, robustness failures decreased. The latest tests were conducted by setting $Phim$ to a value smaller than the minimum and adjusting the parameters tuning. While the speed does not seem to be influenced by the new tuning, there is a small offset between the Reference Torque and the real Torque. Nevertheless, it is evident that the control preserves its readiness (figure 7.8).

The difference between IAE and MSE nominal values and their value in the presence of parametric variation is negligible, both using PI and SMC controllers: anyway, it is worth highlighting that this difference is ±1 using SMC and is -3 or +8 using PI. These results demonstrate appreciable SMC robustness.

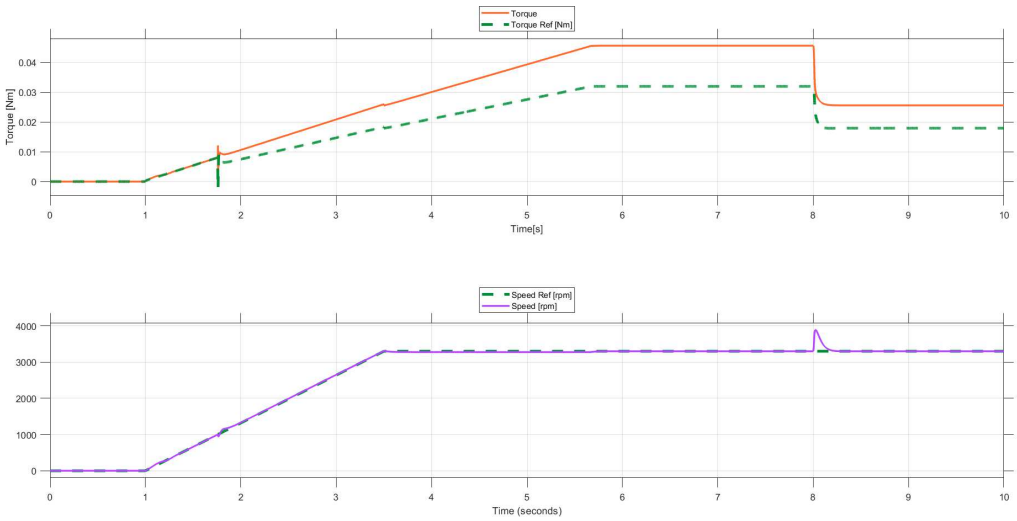

Figure 7.8: Torque and Speed SMC with the new tuning

Chapter 8

Future Developments

The results obtained are satisfactory, indicating the effectiveness of sliding mode control. However, there is still room for improvement in control performance. In fact, some undesired effects can be reduced by further adjusting the tuning preserving the overall robustness.
The analysis for this thesis was conducted using Whirlpool's Observer, which is essentially a Luenberger's Observer. However, it could be worthwhile to consider replacing it with a Sliding Mode Observer (SMO). Several studies have shown that an SMO can provide effective estimates of rotor position and speed, and can achieve good static and dynamic performance. The SMO is popular due to its simple algorithm and robustness, which reduces the observer's dependence on the model [6]. This possibility suggests that a full Sliding Mode approach could be a feasible solution with unexpected but excellent results.

Chapter 9

Conclusions

The main goal of this thesis was to demonstrate how Sliding Mode Control, starting from a simulated environment, could be a viable alternative to current control techniques even in real-world scenarios. Following the outstanding results achieved in thesis [1], the *SIL* (*Software in the Loop*) approach was pursued. This involved generating C code and running it in a simulation, to obtain the same results as observed in the model based approach. Afterward, the code was implemented in the micro controller of the dishwasher board to test in the actual environment.
The test results were fully comparable to those obtained in the simulation. This was made possible thanks to the excellent simulator developed by Whirlpool and the effective implementation of regulators in Sliding Mode.
Additionally, it is worth noting that some oscillations present in the signals obtained in simulation, such as those of currents and voltages, were absent in the signals obtained in a real environment.
The discharge times of two controllers were compared and it was found that using Sliding Mode regulators took lesser time than using PI regulators. Moreover, the Sliding Mode Control was more efficient, as it showed no over-elongation.

A quantitative comparison between the two regulators showed that both were robust to parametric variations. However, the values of IAE and MSE of the Sliding Mode Control varied by only ± 1 with respect to the nominal value, while for PI, the range of values was from -3 to +8 concerning the nominal value. In reality, the IAE and MSE values were found to be very close to those obtained in simulation for both regulators.

This paper has demonstrated that a theoretical concept can be applied in the industrial field and represents a valid starting point for possible future developments. In conclusion, Sliding Mode Control is a valid and highly-performing alternative to PI control.

Bibliography

[1] R. N. Gulesin, *Design and development of a model based and sensorless control system for a permanent magnet synchronous motor*, February 2024

[2] Maria Letizia Corradini, *Senior Member, IEEE*, Gianluca Ippoliti, Sauro Longhi, *Senior Member, IEEE*, and Giuseppe Orlando: *A Quasi-Sliding Mode Approach for Robust Control and Speed Estimation of PM Synchronous Motors*, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 59, NO. 2, FEBRUARY, 2012.

[3] A. E. Fitzgerald, C. Kingsley Jr., A. Kusko, *Macchine elettriche*, FRANCO ANGELI EDITORE, 2006.

[4] IEEE Standard 61012-1990 *IEEE Standard Glossary of Software Engineering Terminology*, Dec. 1990, pp. 1–84.

[5] K.Astrom, T.Hagglund: "Integrator Windup" in *PID Controllers: Theory, Design and Tuning - Second Edition*, INSTRUMENTS SOCIETY OF AMERICA, 1995

[6] Zhaowei Qiao, Tingna Shi, Yindong Wang, Yan Yan, Changliang Xia, *Senior Member, IEEE* and Xiangning He, *Fellow, IEEE*, *New Sliding-Mode Observer for Position Sensorless Control of Permanent-Magnet Synchronous Motor*, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 60, NO. 2, FEBRUARY 2013