ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

INGEGNERIA DELL'ENERGIA ELETTRICA
E DELL'INFORMAZIONE "GUGLIELMO MARCONI" - DEI

AUTOMATION ENGINEERING

TESI DI LAUREA
in
DIAGNOSIS AND CONTROL M

Effective procedure for the optimization of the linear controller in an industrial drive for a nonlinear mechanism

CANDIDATO:
Massimiliano Semati

RELATORE:
Prof. Ing. Andrea Tilli

CORRELATORE:
Ing. Matteo Degli Esposti

Anno Accademico 2017/2018
Sessione II

Abstract

Il progetto di tesi è incentrato sull'ottimizzazione del procedimento di taratura dei regolatori lineari degli anelli di controllo di posizione e velocità presenti negli azionamenti usati industrialmente su macchine automatiche.

La taratura del controllo risulta difficile specialmente quando il carico è ad inerzia variabile in dipendenza dalla posizione, dunque non lineare, in casi di cinematiche complesse come ad esempio un quadrilatero articolato.

Il lavoro è stato svolto in collaborazione con l'azienda G.D S.p.A. utilizzando un gruppo meccanico standalone realmente utilizzato nelle macchine automatiche per il packaging di sigarette.

L'ottimizzazione si basa sulla simulazione in ambiente Matlab/Simulink dell'intero sistema di controllo, comprensivo

- *del modello Simulink degli anelli di controllo del drive, inclusa la dinamica elettrica del motore, e*
- *del modello Simscape del meccanismo.*

L'attività è stata divisa in diverse fasi:

La prima fase del lavoro è stata la validazione di entrambi i modelli affinché fossero sufficientemente fedeli al comportamento reale.

Il secondo passo è stato fornire una prima taratura di tentativo che fungesse da punto di partenza per l'algoritmo di ottimizzazione. In questa fase abbiamo linearizzato il modello meccanico con l'inerzia minima e utilizzato il metodo delle formule di inversione per determinare i parametri di controllo. Già questa taratura, seppur conservativa, ha portato ad un miglioramento delle performance del sistema rispetto alla taratura empirica comunemente fatta in ambito industriale.

Infine, abbiamo lanciato l'algoritmo di ottimizzazione definendo opportunamente la funzione di costo, ed il risultato è stato decisamente positivo, portando ad un ulteriore miglioramento medio del massimo errore di inseguimento di circa il 25%, ma anche oltre il 30% in alcuni casi.

Table of Contents

1 Introduction

Nowadays, the concept of "digital twin" has become very popular within the wider topic of Industry 4.0 in the automation field. It refers to the digital model of a physical system or process, such a model allows engineers to speed up the design process and make tests and simulations avoiding expensive and time-consuming experiments on real systems. In particular, it becomes easier and faster to try different solutions, different parameters, different structures; all of that before implementing such solutions in practice.

Within this context, the work of the thesis was developed during a collaboration project with G.D S.p.A. with the aim of, firstly, validating the Simulink model of a drive controller on a Simscape model of a nonlinear mechanical system used in automatic machines produced by the company, and, secondly, creating a procedure to optimize the tuning of the linear regulators in the drive using a linearization of the plant and a global optimization algorithm.

The model of the control scheme was built throughout a previous internship experience in G.D, while the Simscape model of the mechanical system was created by mechanical engineers of the company.
The control scheme includes, as usual, position, velocity and current loops; however, the current dynamics of a motor can usually be neglected, then the first step of the workflow was to connect the mechanical system directly to the output of the velocity control loop, assuming the current loop as a unitary gain. In a second step we added the current loop, which includes the current regulator and the electric model of the motor, to be able to isolate the source of problems, in case they would occur, since the electric model introduces inevitably some uncertainties. Another feature we decided to add was the PWM modulation and a simplified model of the inverter to better understand the weight of all the phenomena affecting the real system.

Several tests were performed on the mechanical setup changing both the cam and the speed of the cam and the experimental data were collected. Those data were later compared to the simulation data to estimate the deviation of the simulation with respect to reality.

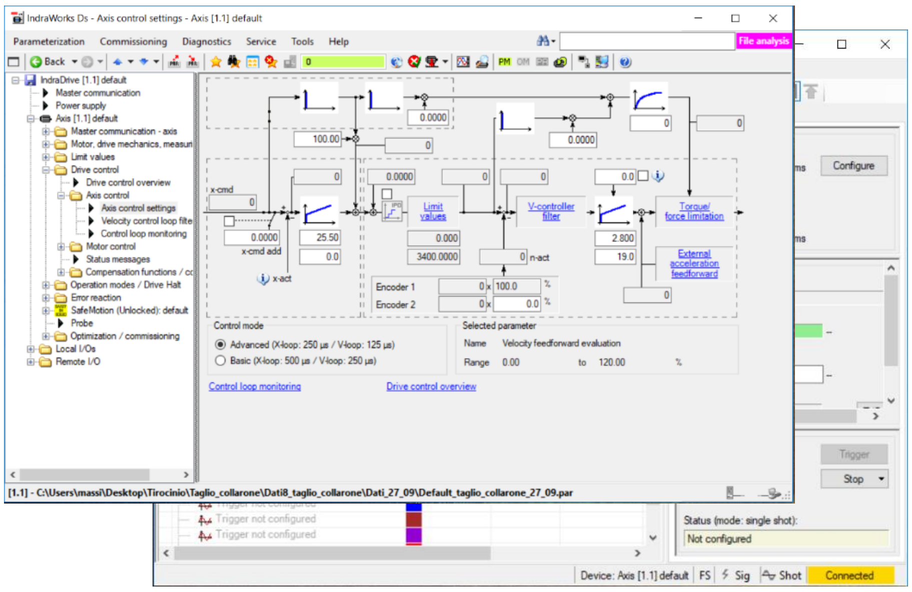
After the validation of the control scheme was concluded, the most important step of the project was to approach the problem of tuning of the controller, because very frequently it is performed manually on the real system following some empirical rules, thus not relying on a simulation environment or any knowledge of control theory. This method is really time-consuming, especially as the number of electric axes in an automatic machine increases, and does not guarantee the best possible performances. Hence, we established a fast procedure for an initial tuning, based on a linearized model of the mechanism and on the inversion formulae, then ran an optimization that uses it as starting point and launches the simulation at each iteration to evaluate the performances of the system in terms of following error and frequency content of the speed.

The next chapter describes the experimental setup and simulation models, chapters 3, 4, 5 show the results of the simulations with the mechanical system only, with the current loop, with current loop and PWM, respectively. Chapters 6 and 7 describes the attempts to model friction and cogging torque disturbance. Chapter 8 presents the first optimization problem setup for the control parameters, without exploiting any knowledge of the controlled system, and the related results obtained with the real plant. In chapter 9 it is explained how the system is linearized and the controller is tuned following theorical criteria, and how this preliminary stage is useful for a more effective optimization.

2 Testbed, Data and Simulation Models

The system to be controlled is a mechanism, called inner frame cutter, used in automatic machines for the packaging of cigarettes and it can be thought of as a four-bar linkage. The motor and the drive are provided by Bosch Rexroth.

In order to collect the data, we connected the laptop to the drive through Ethernet cable, then ran the motor and recorded all the signals of interest with the oscilloscope available in IndraWorks software, by Bosch. Moreover, the software allows to save the parameters of the drive, such as cycle times, and of the controlled motor in a file that can be imported to Matlab before running the simulations.



Control mode
[✓ Advanced (X-loop: 250 µs / V-loop: 125 µs)]
Basic (X-loop: 500 µs / V-loop: 250 µs)

Selected parameter
Name: Velocity feedforward evaluation
Range: 0.00 to 120.00 %

Encoder 1: 0 x 100.0 %
Encoder 2: 0 x 0.0 %

Control loop monitoring
Drive control overview

[1.1] - C:\Users\massi\Desktop\Tirocinio\Taglio_collarone\Dati8_taglio_collarone\Dati_27_09\Default_taglio_collarone_27_09.par

Status (mode: single shot): Not configured

Device: Axis [1.1] default | FS | Sig | Shot

Figure 2-1 IndraWorks software interface

Initially, we collected series of data by selecting different cams for the motor at different velocities (80 ppm, 200 ppm, 400 ppm, 600 ppm). The nominal speed of the automatic machine this mechanism belongs to is 600 ppm, therefore the following figures shows the considered cams at that speed.

- Cam 1

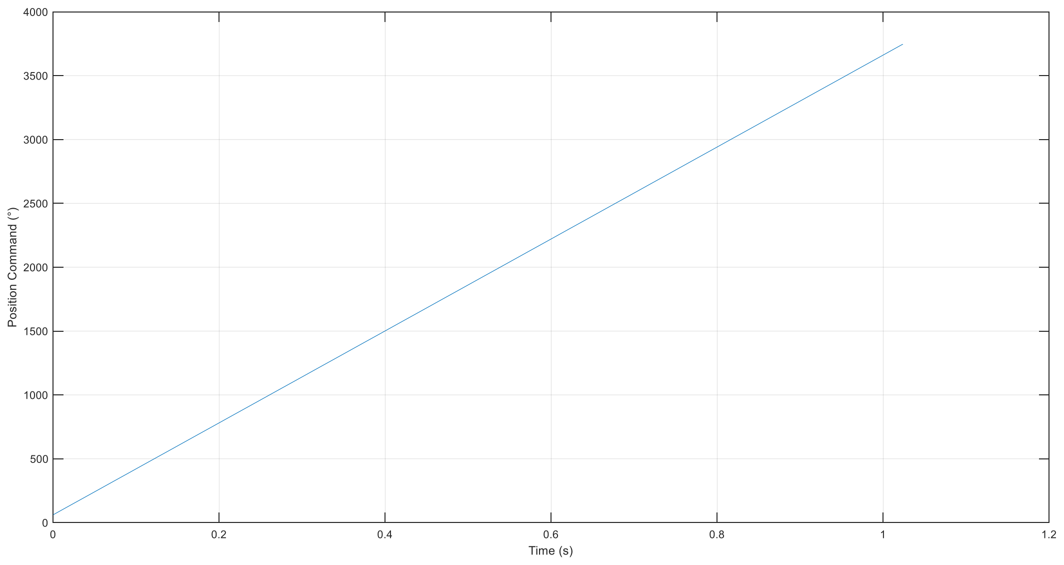


Figure 2-2 Position Reference (degrees)

- Cam 2

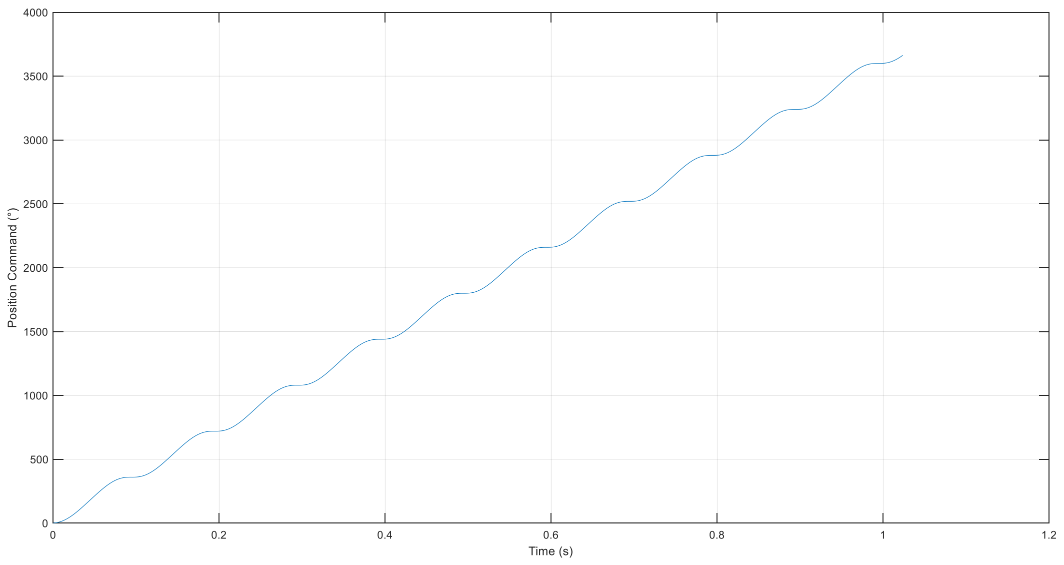


Figure 2-3 Position Reference (degrees)

- Cam 3

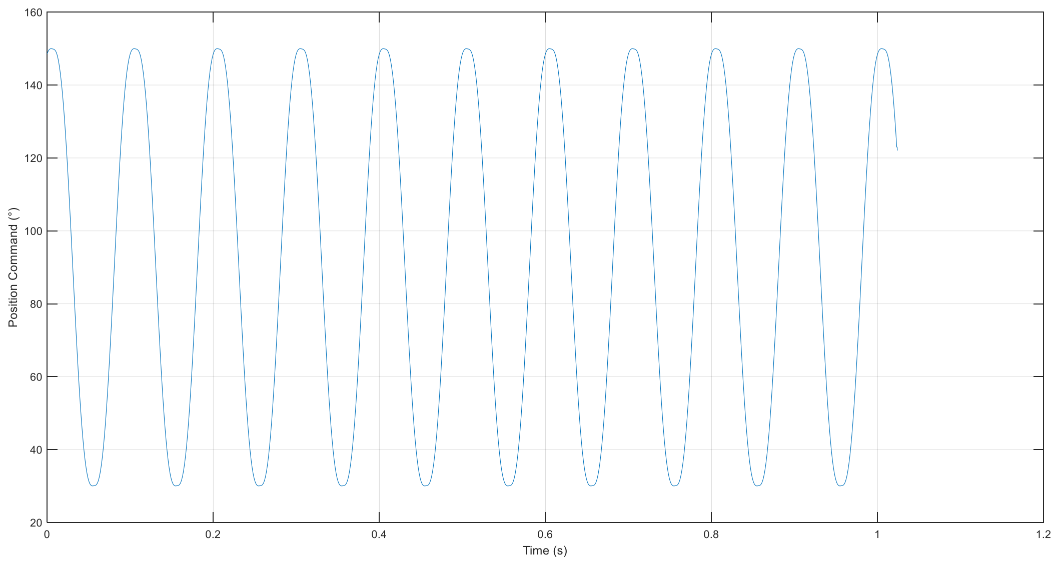


Figure 2-4 Position Reference (degrees)

- Cam 4

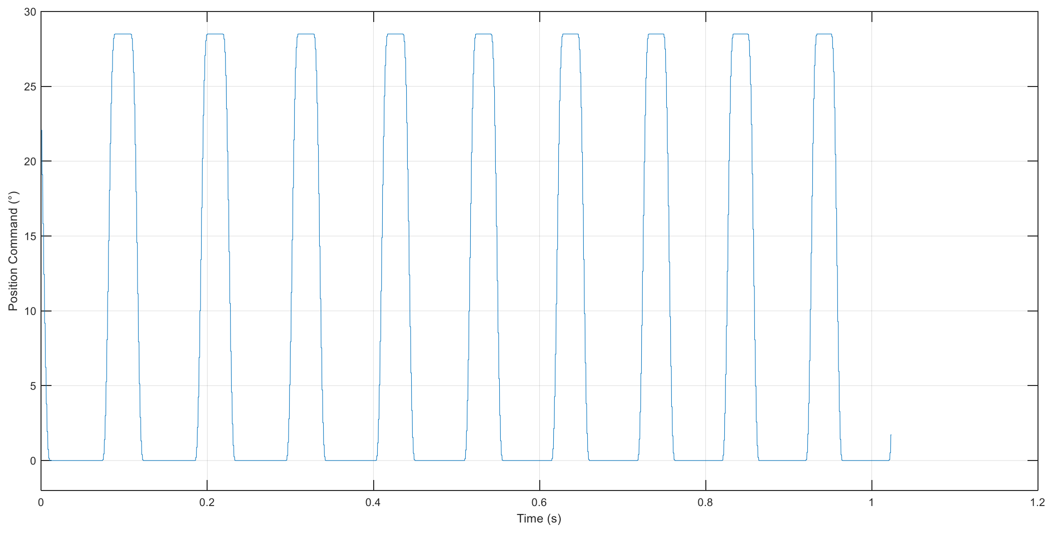


Figure 2-5 Position Reference (degrees)

- Cam 5

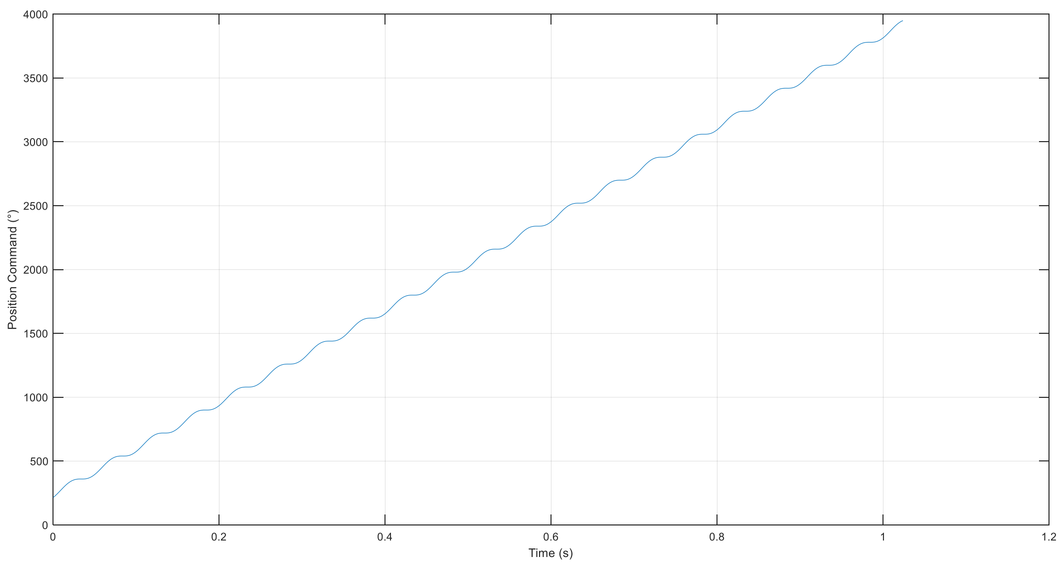


Figure 2-6 Position Reference (degrees)

- Cam 6

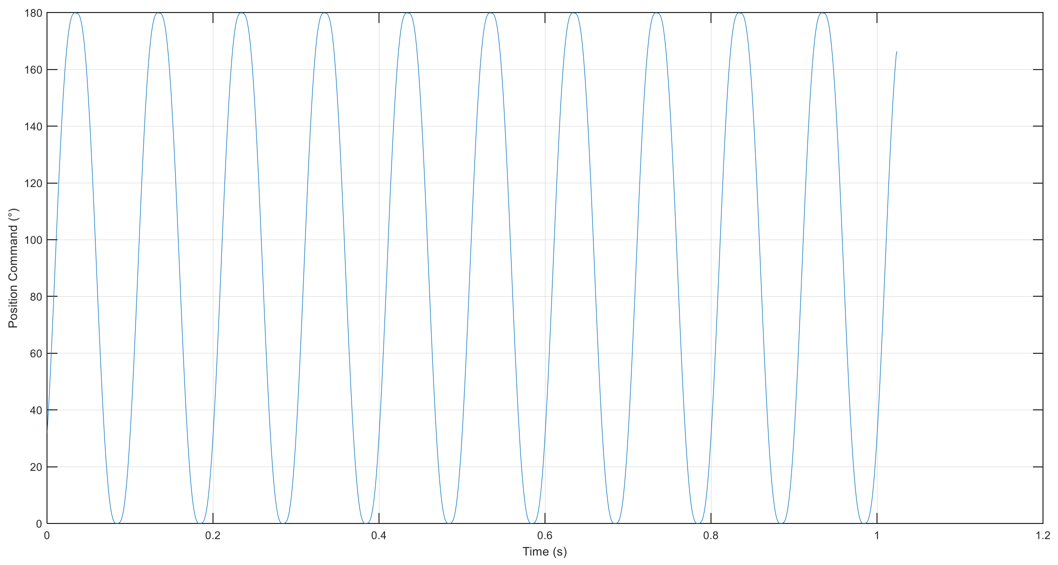


Figure 2-7 Position Reference (degrees)

Each set of data contained the position command values from the PLC, which are fed to the interpolator in the simulation environment, and other signals used to make comparisons with their respective simulated counterparts: position feedback, speed reference, speed feedback, current reference, current feedback, output voltage. We recorded the integral action of the PI speed regulator, too, which allowed us to initialize the integral in our PI regulator properly. That was necessary as the data were collected at steady-state, not at null initial conditions.

The integral action of current regulator couldn't be read, then we initialized it knowing input/output data, in those cases we included also the current dynamics:

$$\begin{aligned} &Integral\ initial\ cond \\ &\quad = V_{out}(1) - \underbrace{K_p * \left(I_{ref}(1) - I_{act}(1)\right)}_{\text{Proportional Action}} - \underbrace{p * \phi_e * motor\_speed(1)}_{\text{Back-emf compensation}} \end{aligned}$$

K_p : proportional gain;

p : number of poles;

ϕ_e : excitation flux.

2.1 Simscape Model of the Plant

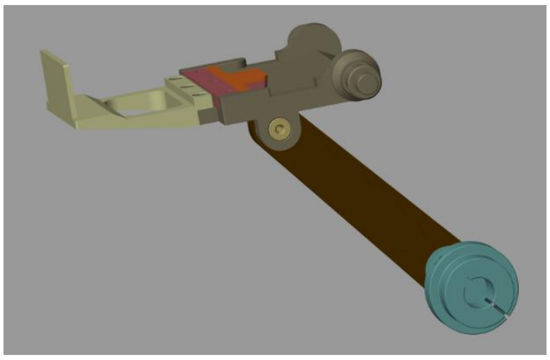
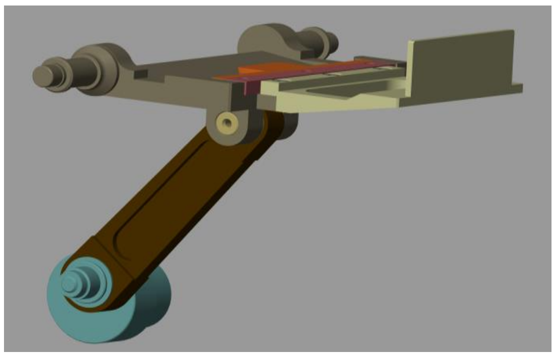



Figure 2-8 Images of the 3D Simscape model

The 3D model of the mechanism was created by mechanical engineers of the company and imported into Simscape with some simplifications. Neither elasticity, i.e. all bodies are considered infinitely rigid, nor bearings are taken into account. The Simscape block diagram is shown below.

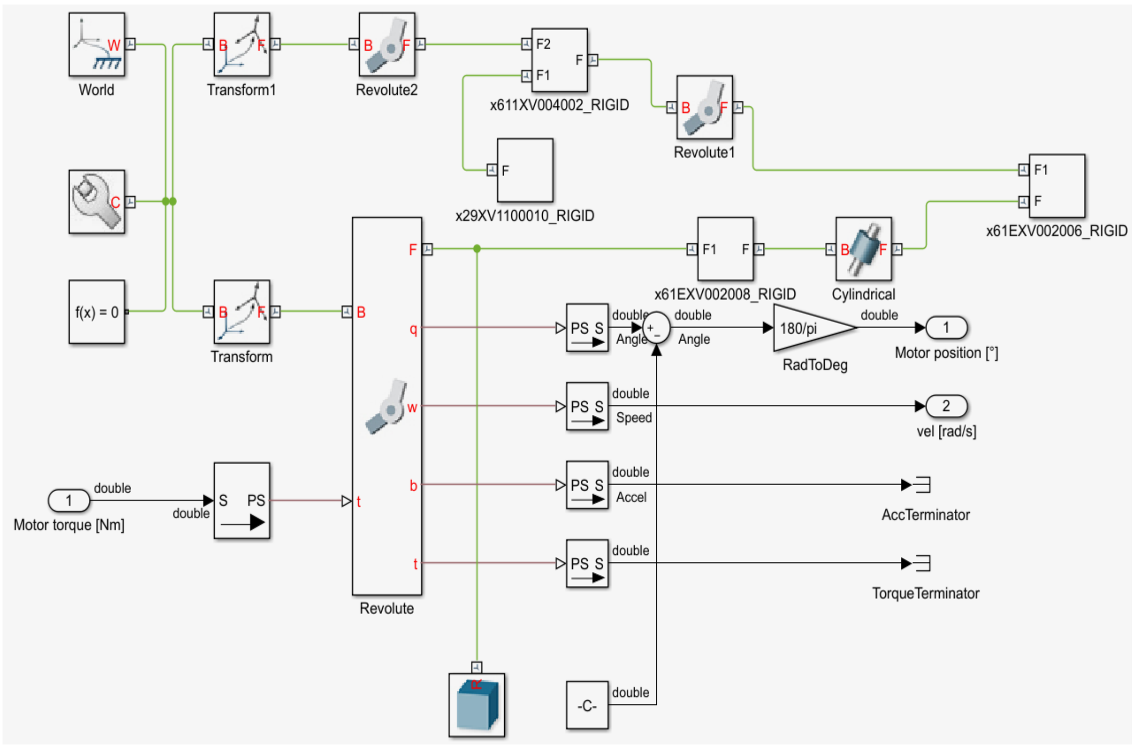


Figure 2-9 Simscape scheme

The zero position of the system is such that the plate is horizontal. The effect of gravity was included.

2.2 Simulink Model of the Controller

The model of the controller is composed by the following subsystems:

1. Position control loop: pure proportional regulator plus velocity feedforward;
2. Velocity control loop: discrete PI regulator plus some optional filters acting on the velocity error (not used in this work);
3. Current control loop: discrete PI regulator plus compensation of the back-emf.
4. Interpolator of the position command values.

A global view of the controller is shown in the next figure.

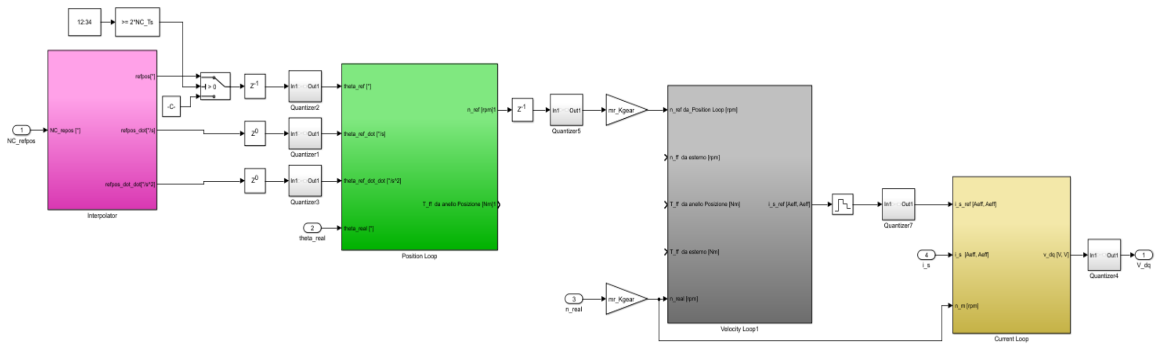


Figure 2-10 Full control scheme

This is a discrete controller, then the input signals must be sampled and the integrators in the velocity and current loop are discretized as well, with the forward Euler discretization method. Also, the signals must be quantized according to the quantization in the real drive: 0.0001° for position, 0.0001 rpm for velocities, 0.001 A for currents, 0.1 V for the output voltage.

Each control loop has its own cycle time, during the tests performed by us we kept a cycle time of 250 µs for the position loop, 125 µs for the velocity loop, 250 µs for the current loop. It may seem odd that the current regulator has a lower frequency than that of the velocity one, but it is set so because 250 µs is also the switching period for the PWM modulation.

The PLC connected to the drive changes the position command value every 1ms, four times slower than the position loop, therefore an interpolator is placed before that regulator to have a better approximation of the desired continuous position waveform. The interpolator let us select between linear interpolation and cubic interpolation, however only the cubic one is adopted usually.

During the previous internship in G.D, described in a report called "Realizzazione e validazione del modello Matlab/Simulink di un drive Bosch-Rexroth" [1], the performances of this model with respect to reality were studied and we observed that the position and velocity controllers were very good, while the electric behavior had noticeable deviation from the real signals.
Nevertheless, since the electric dynamics of a motor are often much faster than the mechanical transients, the current loop is neglected, typically; that's why we decided to start the work simulating a model where the electric part of the motor

and the current regulator were absent, so that, if some troubles had occurred, we would be able to isolate better their source.

3 Simulation of the mechanical system only

In these simulations the mechanical load, i.e. the Simscape model of the mechanism, is fed directly by the output of the velocity regulator, that is the torque command.

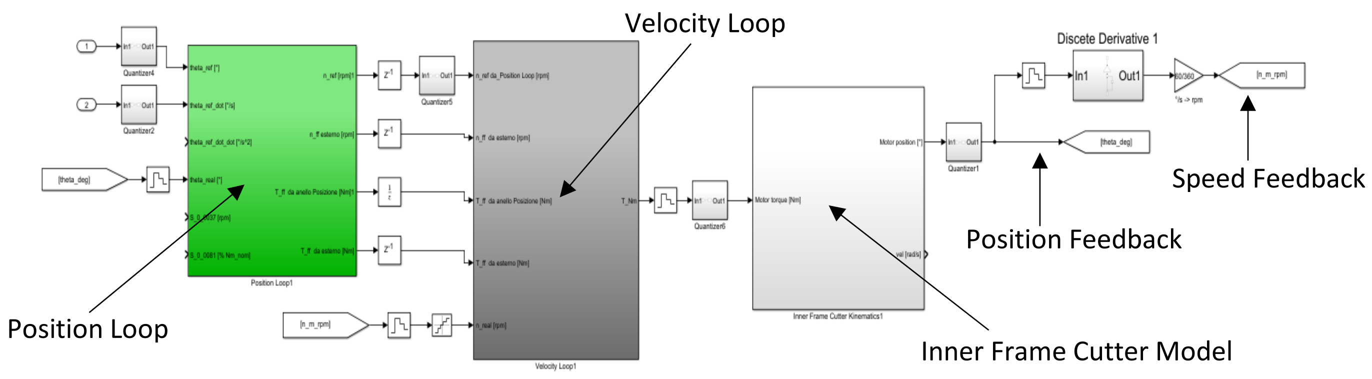


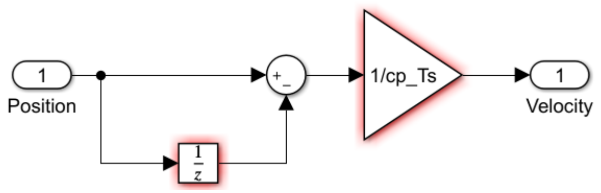
Figure 3-1 Mechanical plant and controller without current loop

3.1 Initialization

As we said before, the data were not collected at the start of the motor, but at steady state, so we had to be careful about the initialization of the simulation, especially the proper functioning of the interpolator.
Its working principle is that every time the PLC sends a new position value, the cubic interpolation is computed again, then at the time we started recording the data, the interpolator knew some previous samples which it could use to interpolate, but we didn't know them. The solution we adopted was to enable the subsystem containing the controller and the plant just after some simulation time, in such a way that, from that time on, the simulated interpolated position signal was equal to the real one.

About the mechanism, we had to set properly the initial position and velocity of the revolute joint corresponding to the motor. Positions and velocities of the other joints are automatically determined, since a four-bar linkage has only one degree of freedom.

The velocity feedback is not taken directly from the mechanical model, instead it is computed by the discrete derivative of the position, which is sampled every 250 μs, since that is how it is actually computed in the drive.



cp_Ts: cycle time of the position loop

3.2 Simulation Results

Now we will have a look at some results of the simulations.

1. Cam 1, 80 ppm

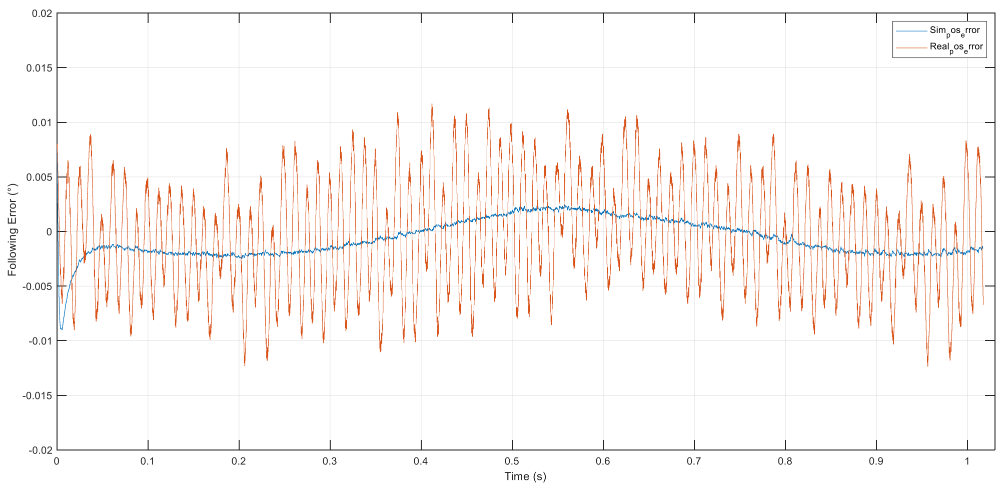


Figure 3-2 Real position error (red) and simulated one (blue)

2. Cam 3, 80 ppm

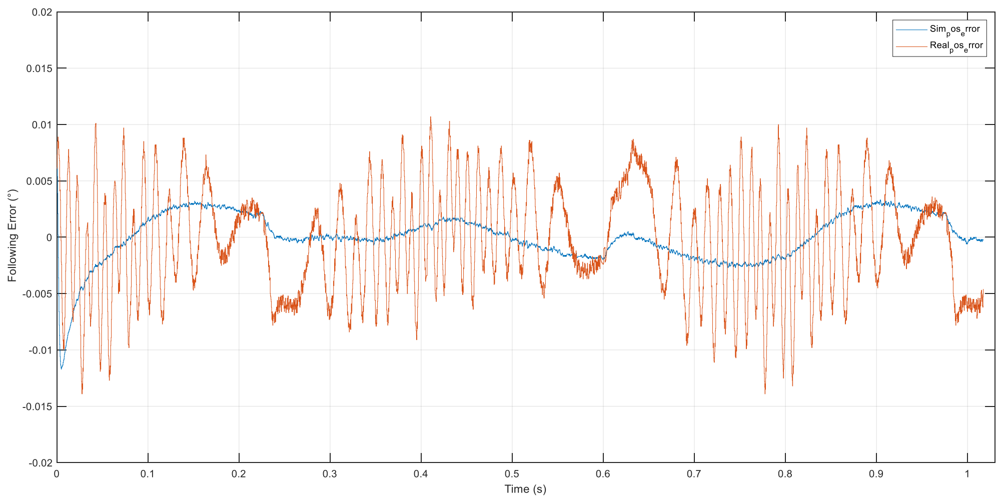


Figure 3-3 Real position error (red) and simulated one (blue)

3. Cam 6, 80 ppm

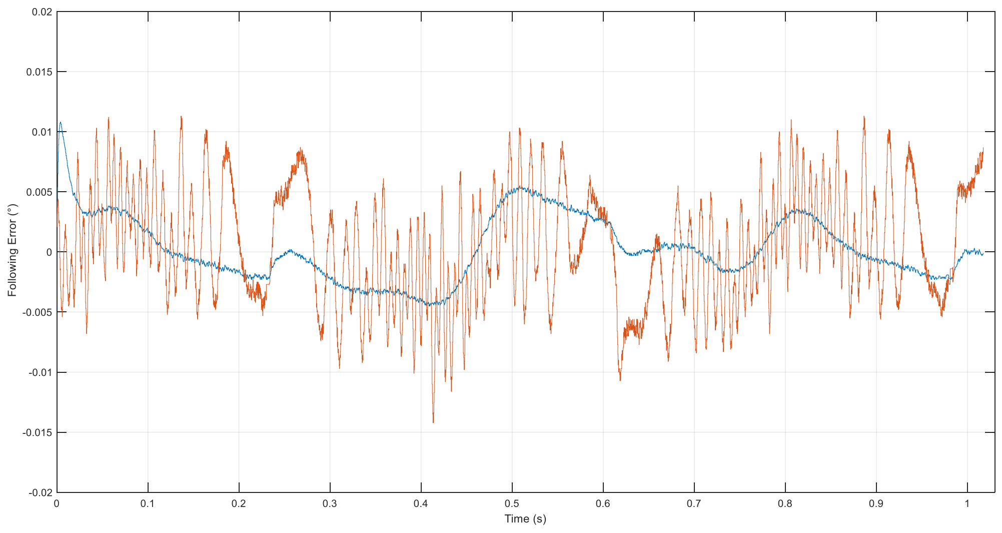


Figure 3-4 Real position error (red) and simulated one (blue)

4. Cam 2, 200 rpm

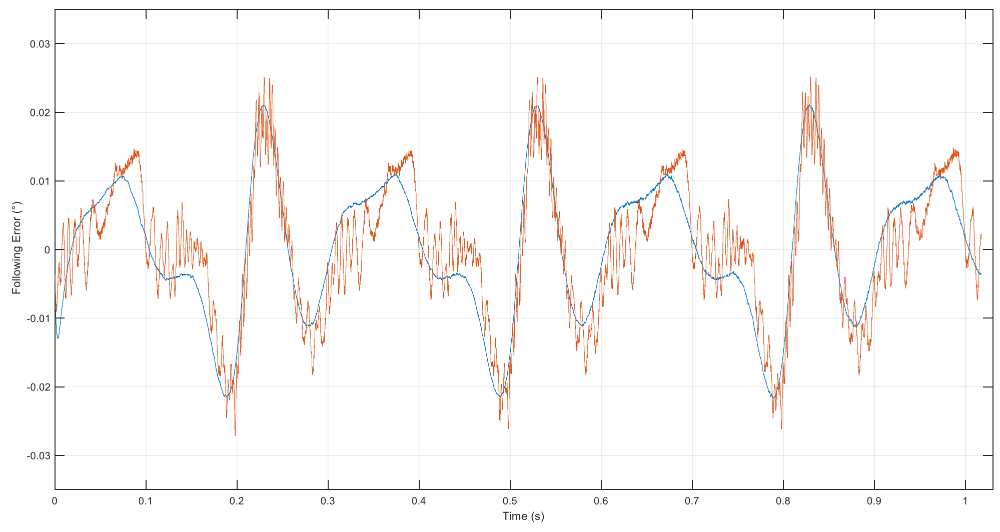


Figure 3-5 Real position error (red) and simulated one (blue)

5. Cam 4, 200 rpm

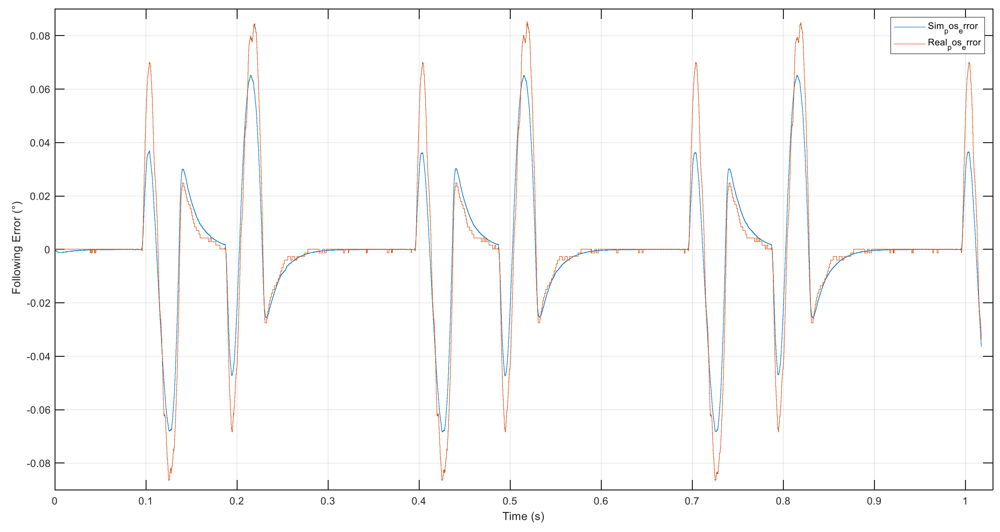


Figure 3-6 Real position error (red) and simulated one (blue)

6. Cam 1, 400 rpm

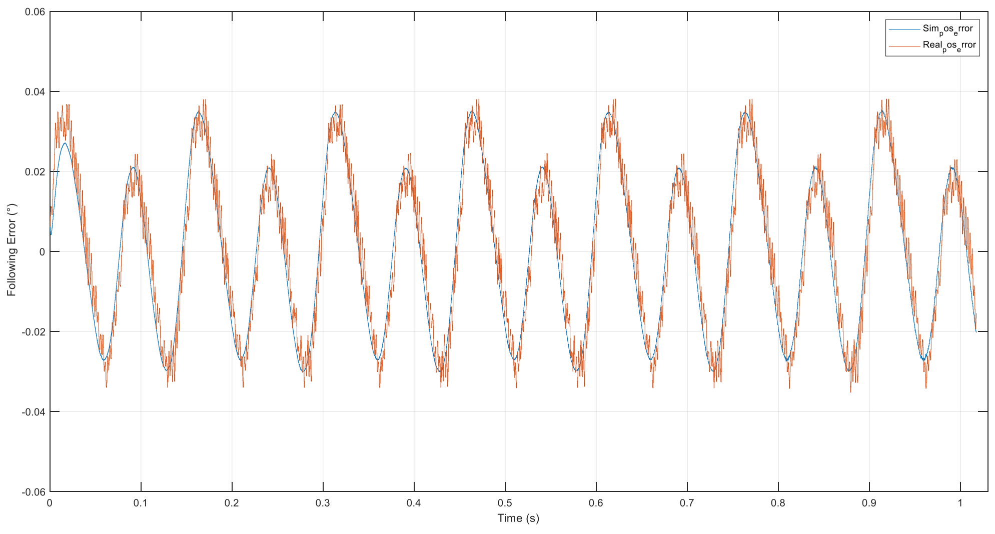


Figure 3-7 Real position error (red) and simulated one (blue)

7. Cam 2, 400 rpm

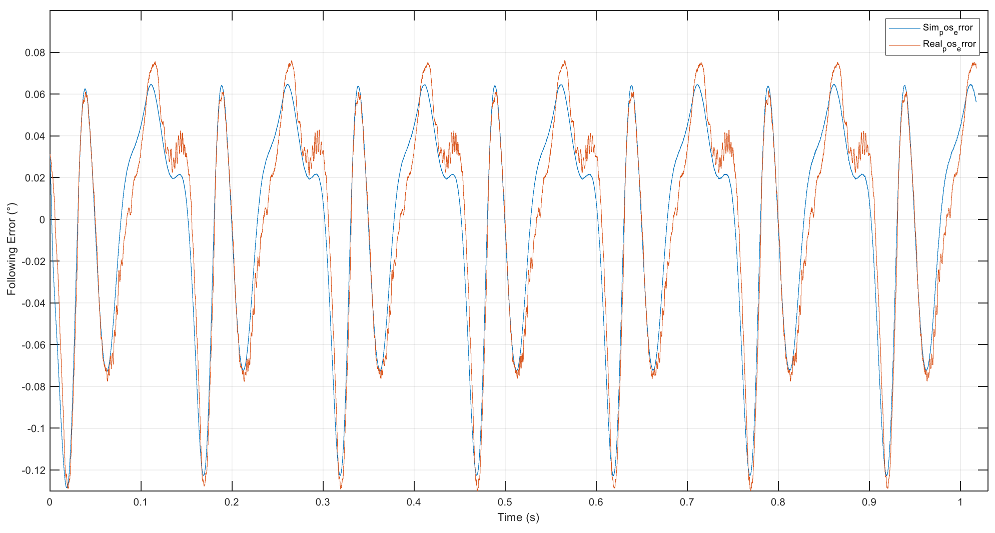


Figure 3-8 Real position error (red) and simulated one (blue)

8. Cam 5, 400 rpm

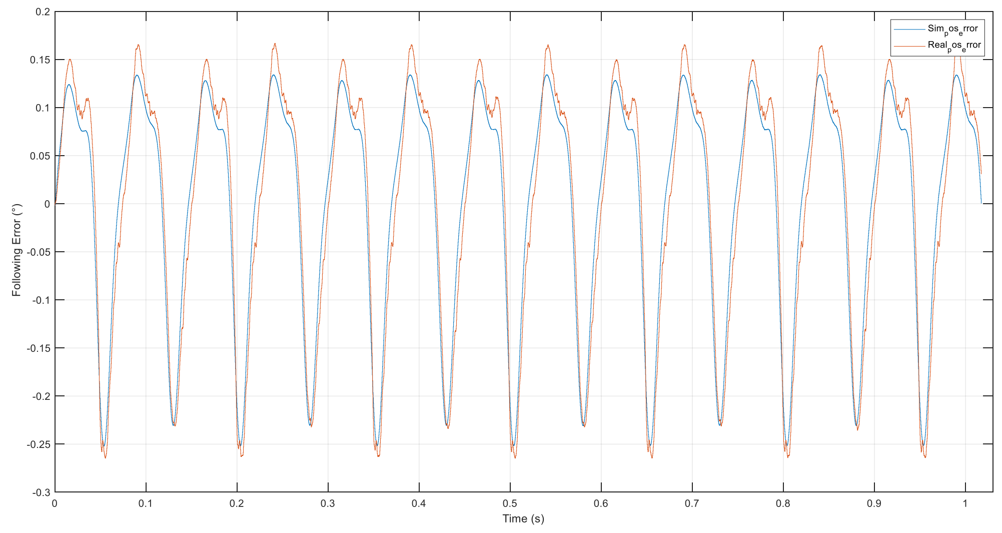


Figure 3-9 Real position error (red) and simulated one (blue)

9. Cam 3, 600 ppm

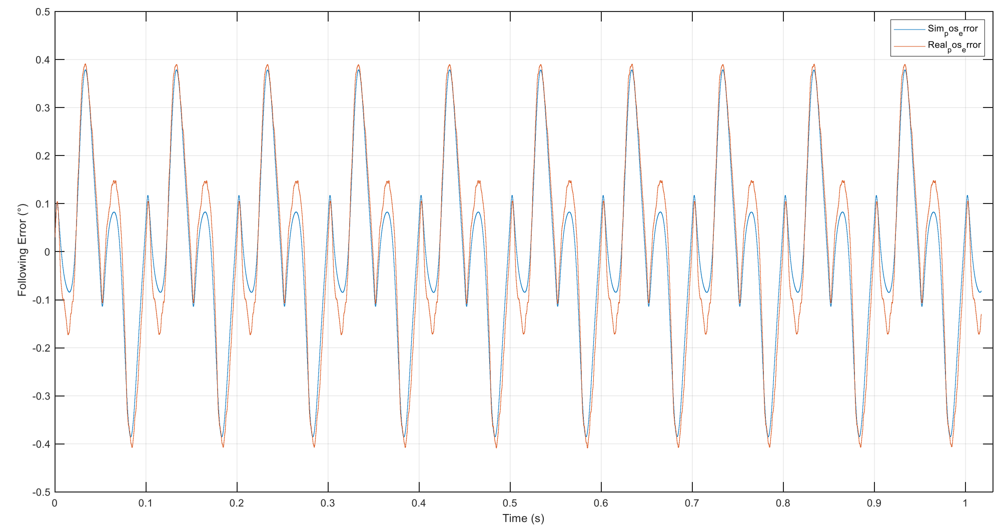


Figure 3-10 Real position error (red) and simulated one (blue)

10. Cam 4, 600 ppm

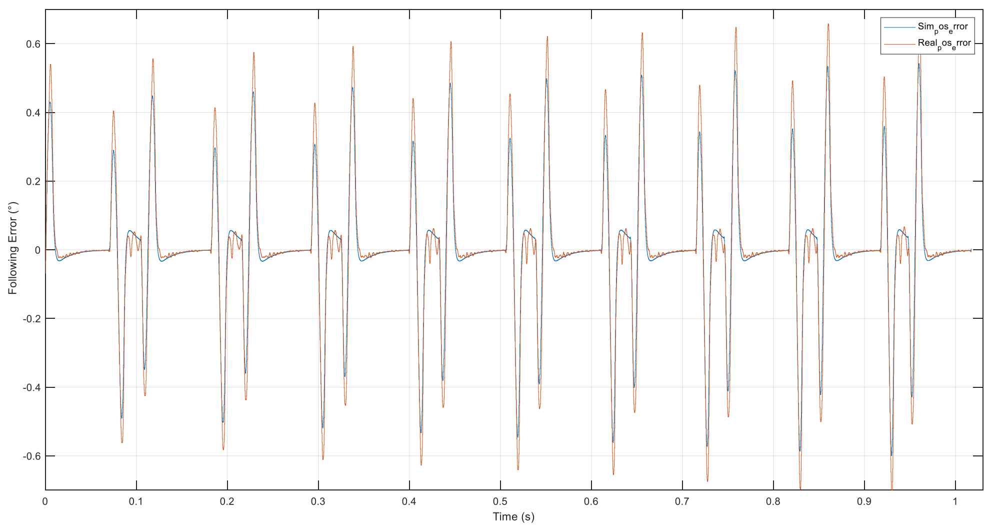


Figure 3-11 Real position error (red) and simulated one (blue)

11. Cam 5, 600 ppm

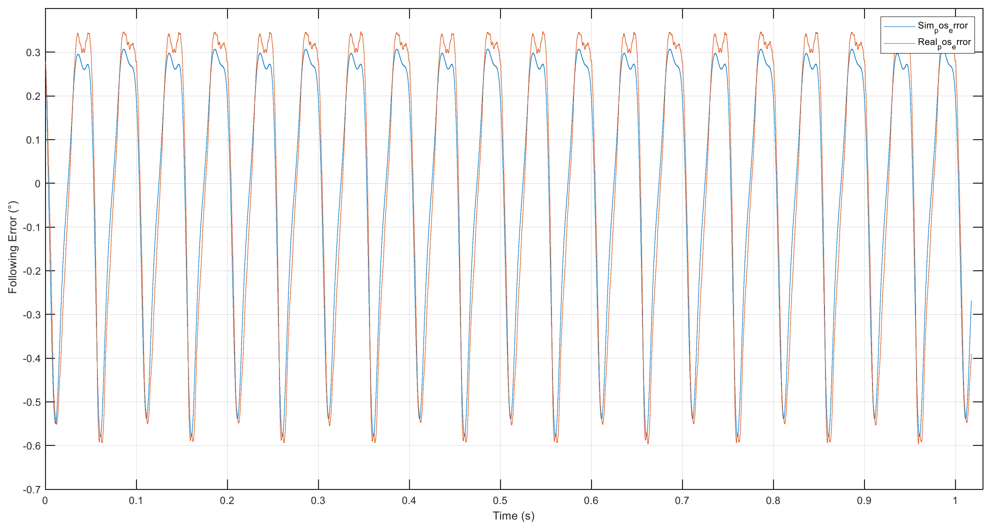


Figure 3-12 Real position error (red) and simulated one (blue)

From the previous plots we can notice that the simulation error, the difference between the real following error end the simulated one, is much better at high speed, because at low speed the real following error has considerable oscillations

and the simulated one almost seems its mean value; those oscillations tend to get smaller, and at last disappear, as speed increases. This may suggest that they are due to some torque disturbance that is filtered at high speed thanks to the inertia of the load.
However, the cams at high speed show that the real system tends to have slightly higher position overshoots with respect to the model: the peaks of the following error are a bit higher.
In general, the simulation error is not greater than 0.1° with high-speed cams.
Now, we will add also the current loop and the electric model of the motor, to understand if its dynamics has significant effect on the following error.

4 Simulation including electric dynamics

The electric model of the permanent magnet synchronous motor is made considering the space vectors of the electromagnetic quantities in the rotating DQ reference system, which the Field-Oriented Control is based on.
This model was developed by one engineer in GD.

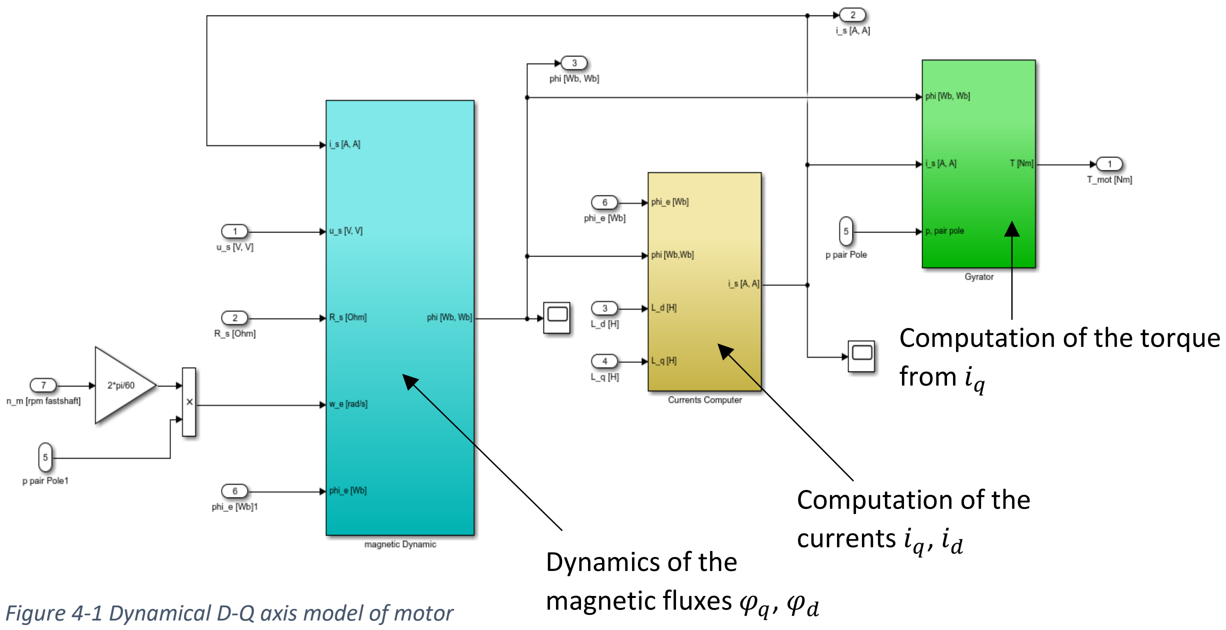


Figure 4-1 Dynamical D-Q axis model of motor

In automatic machines, motors are almost always operated below the nominal speed, i.e. in the range on velocity where the maximum available torque is constant. Therefore, the reference for the current i_d is permanently set to 0.

For the next simulations, this model is fed by the voltages V_d, V_q produced by the current loop, the PWM modulation and the inverter are not taken into account.

4.1 Simulation Results

The following plots show the comparison between the following error with the electric part in the model and the following error without it (as in the previous case).

1. Cam 1, 80 ppm

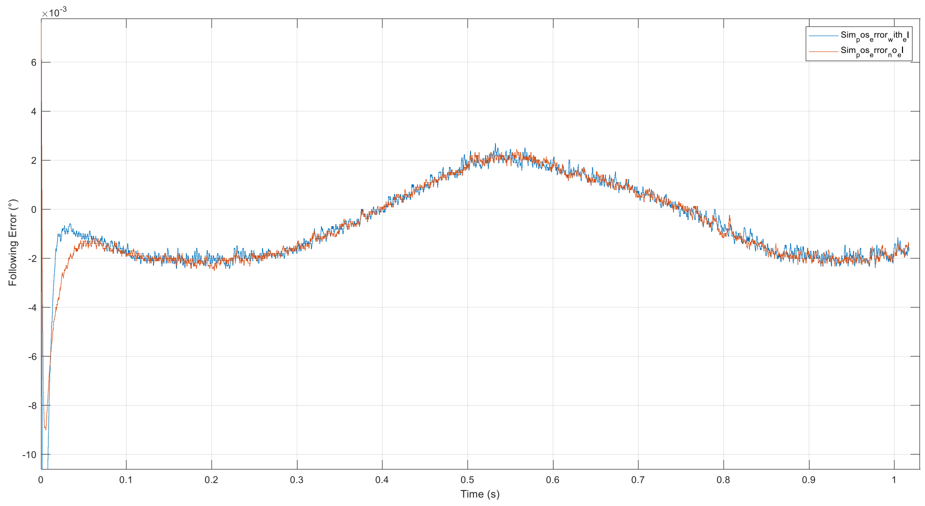


Figure 4-2 Following error with electric part (blue) and without it (red)

2. Cam 5, 80 ppm

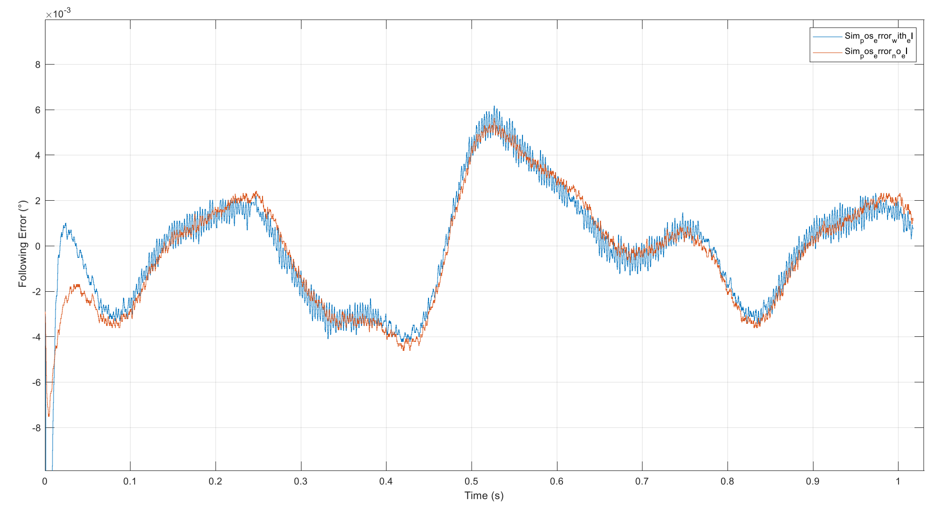


Figure 4-3 Following error with electric part (blue) and without it (red)

3. Cam 3, 200 ppm

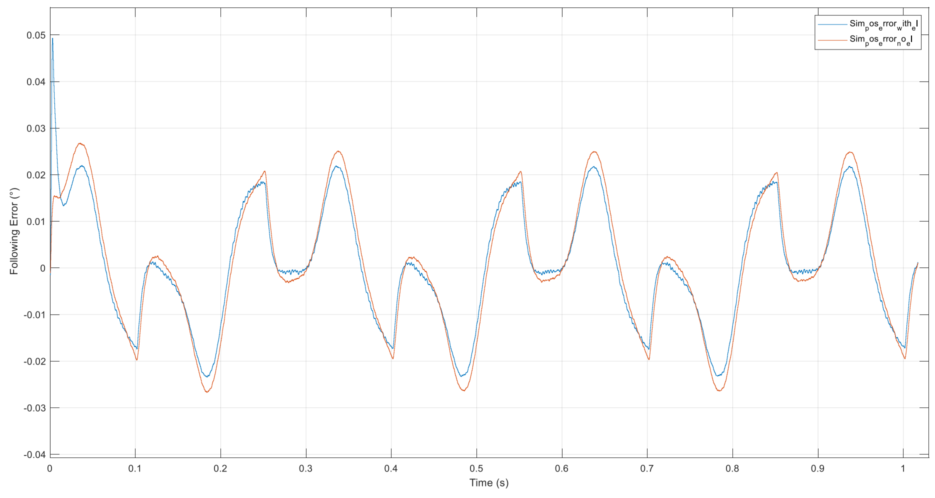


Figure 4-4 Following error with electric part (blue) and without it (red)

4. Cam 6, 200 ppm

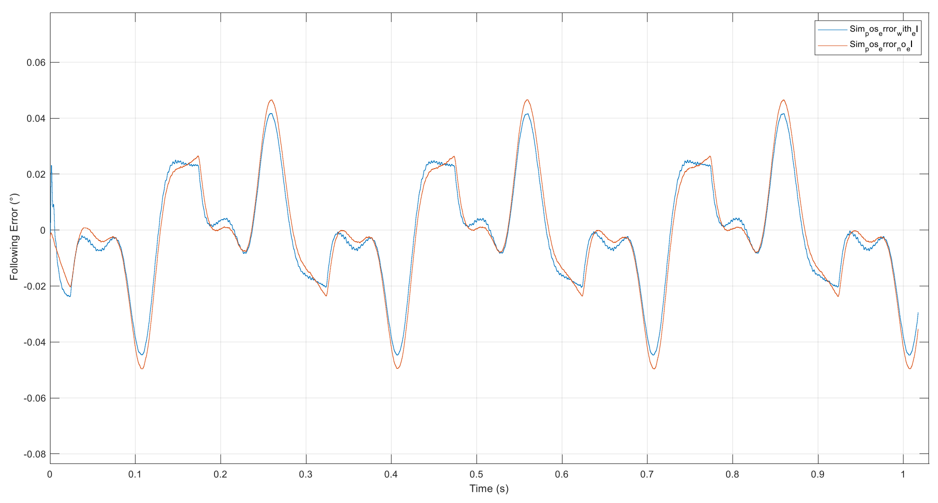


Figure 4-5 Following error with electric part (blue) and without it (red)

5. Cam 2, 400 ppm

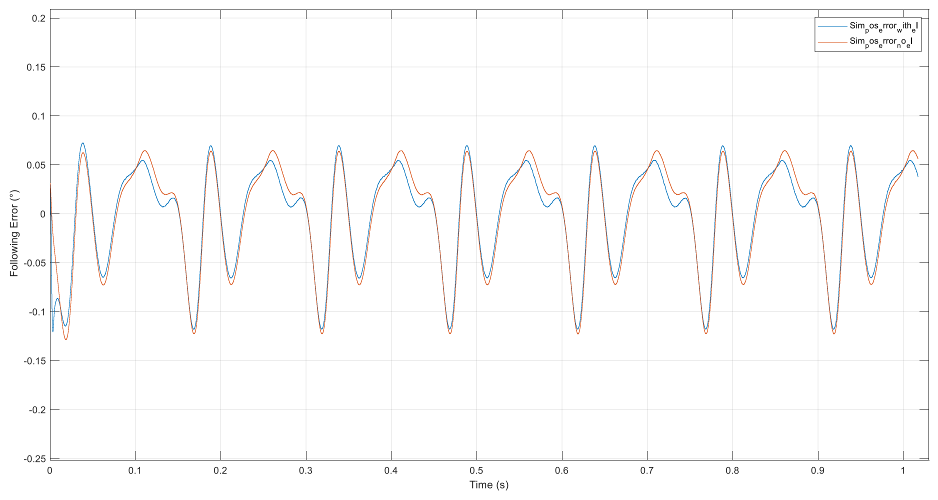


Figure 4-6 Following error with electric part (blue) and without it (red)

6. Cam 5, 400 ppm

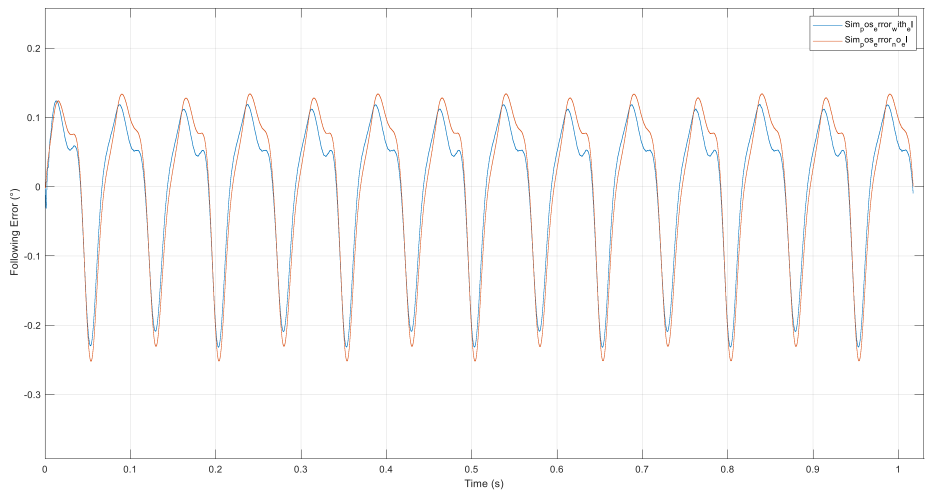


Figure 4-7 Following error with electric part (blue) and without it (red)

7. Cam 1, 600 ppm

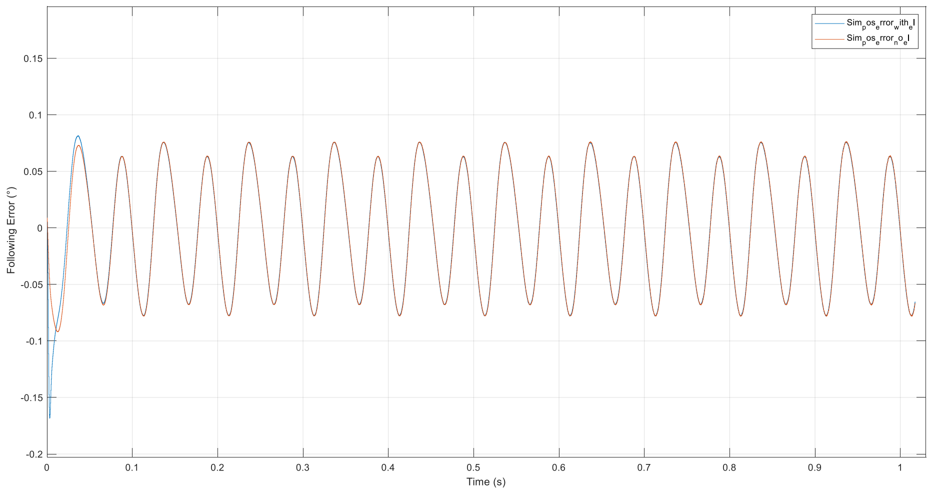


Figure 4-8 Following error with electric part (blue) and without it (red)

8. Cam 4, 600 ppm

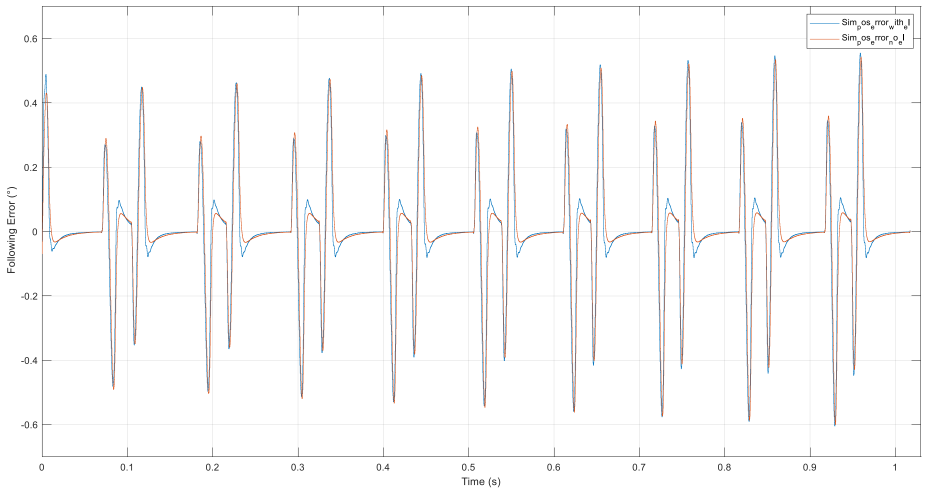
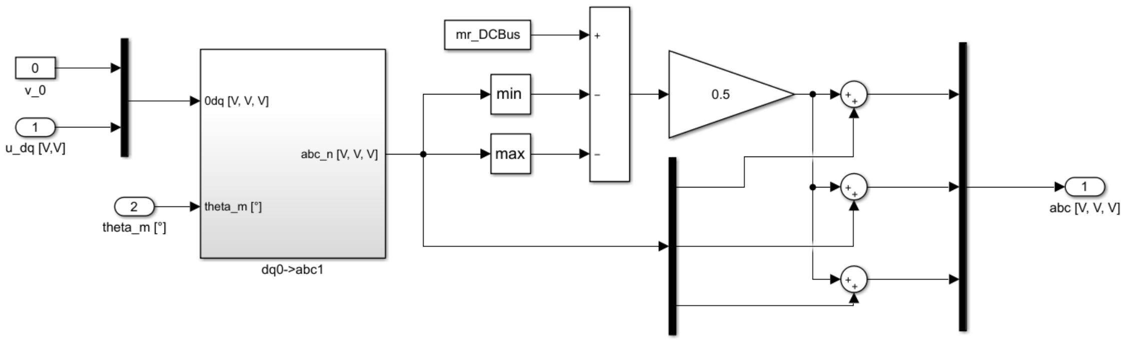


Figure 4-9 Following error with electric part (blue) and without it (red)

All figures tell us that the two following errors are almost equal, being the one including the electrical part just slightly lower than the other one, which means that it is worse in term of simulation error, since we have already seen that the following error of the model without electric dynamics is a bit lower than the real one.

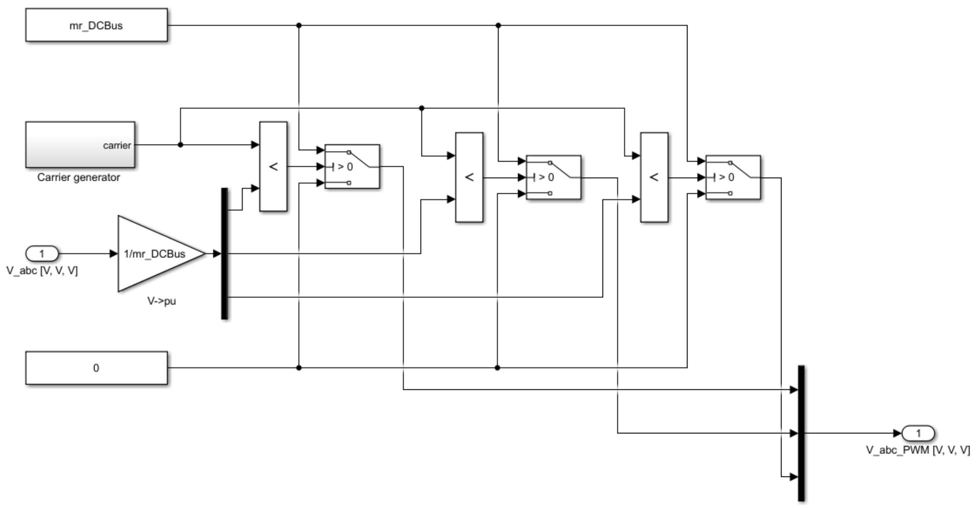
5 Simulation with electric part and PWM

At last, we included in the model the modulation and a simplified model of the inverter, too. The documentation of the drive doesn't give clear information about the chosen type of modulation strategy, then we decided for the Space Vector PWM (SVPWM) to be the most likely one.



The above subsystem computes firstly the desired phase voltages starting from the output voltages V_d, V_q and applying the inverse Park and Clarke transforms, then the desired pole voltages of the inverter according to SVPWM, given the value of the DC Bus voltage, which is assumed constant.

The reference pole voltages enter the inverter subsystem, whose legs are just ideal switches from 0 V to the DC Bus voltage, and are compared to a triangular carrier. The outputs are pulsed pole voltages.



The pulsed pole voltages are, then, transformed to the direct and quadrature voltages applied to the motor using the direct Clarke and Park transforms.

5.1 Simulation Results

The results of the simulations are shown in the following pictures in comparison to those of the model without modulation.

- Cam 3, 80 ppm

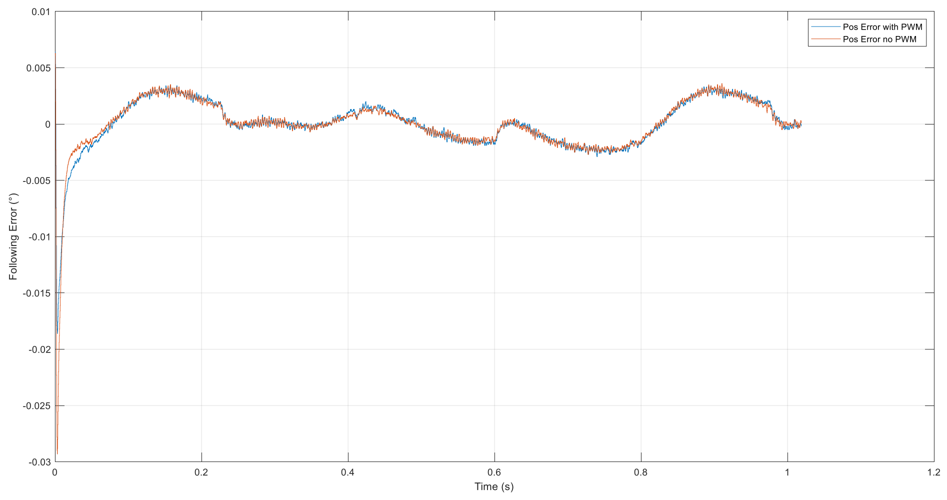


Figure 5-1 Following Error with PWM (blue) and without it (red)

- Cam 4, 80 ppm

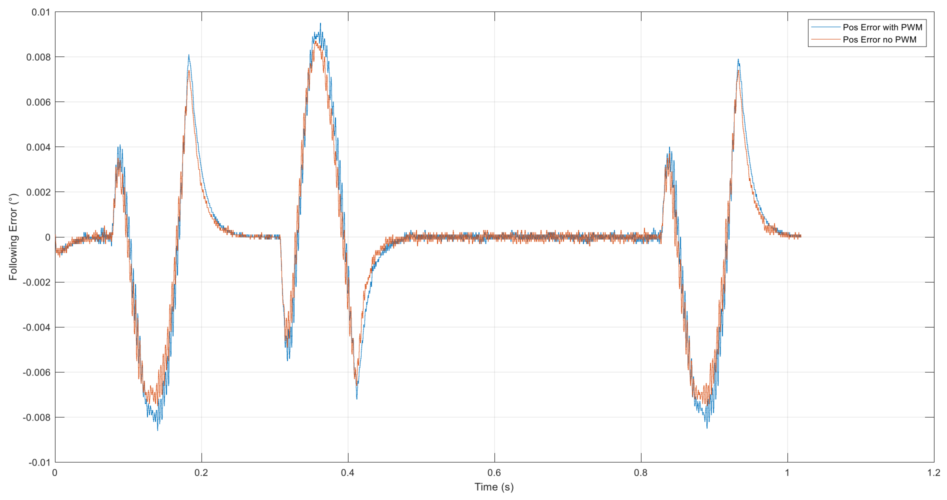


Figure 5-2 Following Error with PWM (blue) and without it (red)

- Cam 1, 200 ppm

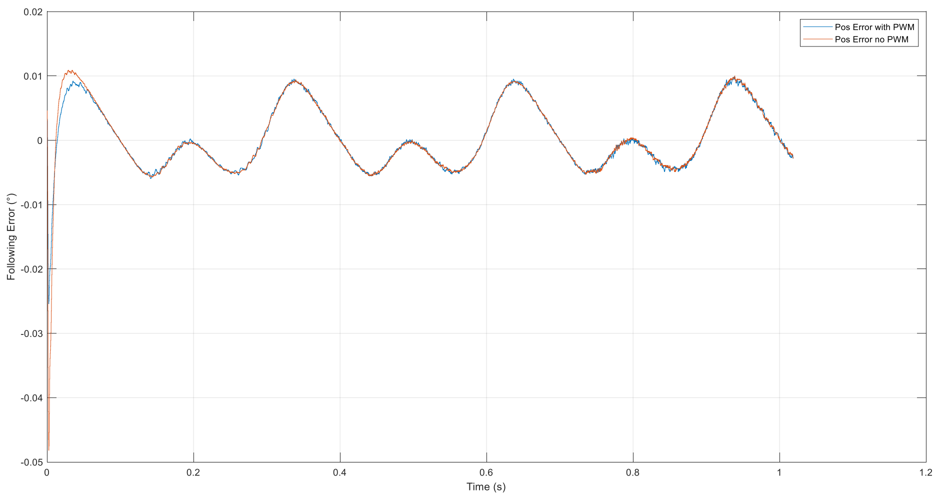


Figure 5-3 Following Error with PWM (blue) and without it (red)

- Cam 5, 200 ppm

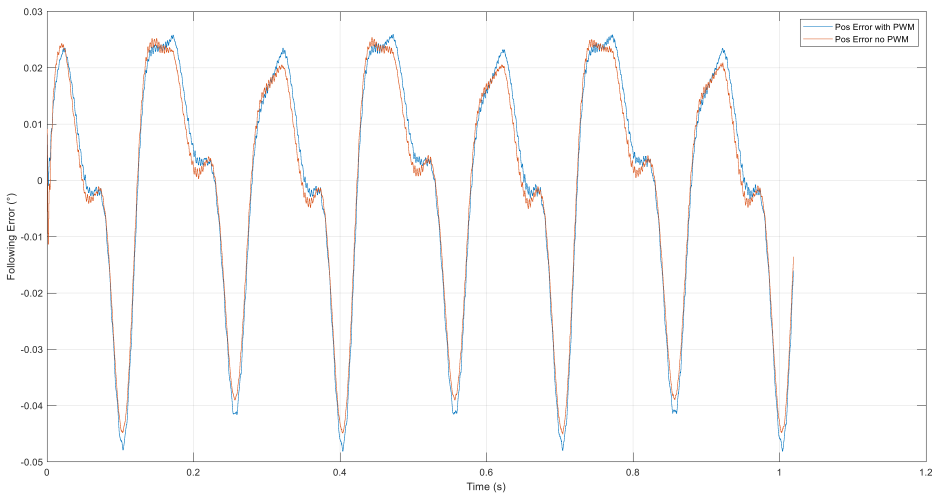


Figure 5-4 Following Error with PWM (blue) and without it (red)

- Cam 2, 400 ppm

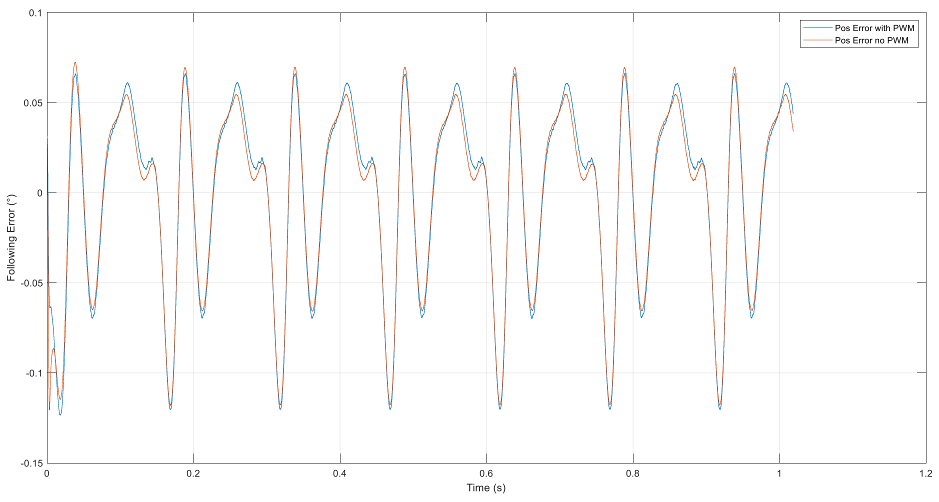


Figure 5-5 Following Error with PWM (blue) and without it (red)

- Cam 6, 400 ppm

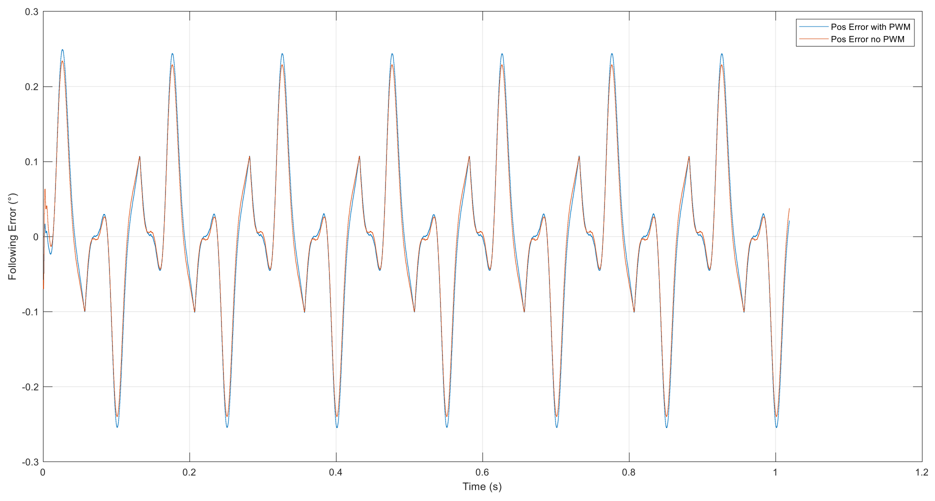


Figure 5-6 Following Error with PWM (blue) and without it (red)

- Cam 4, 600 ppm

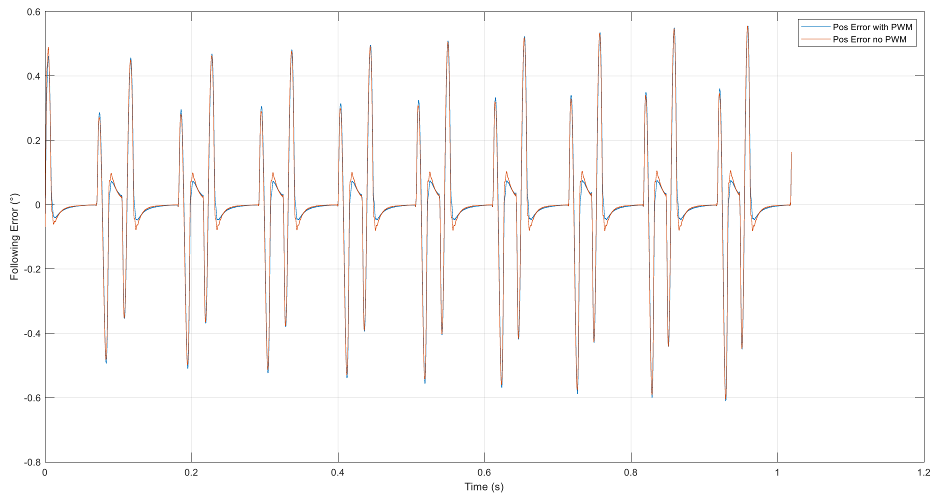


Figure 5-7 Following Error with PWM (blue) and without it (red)

- Cam 5, 600 ppm

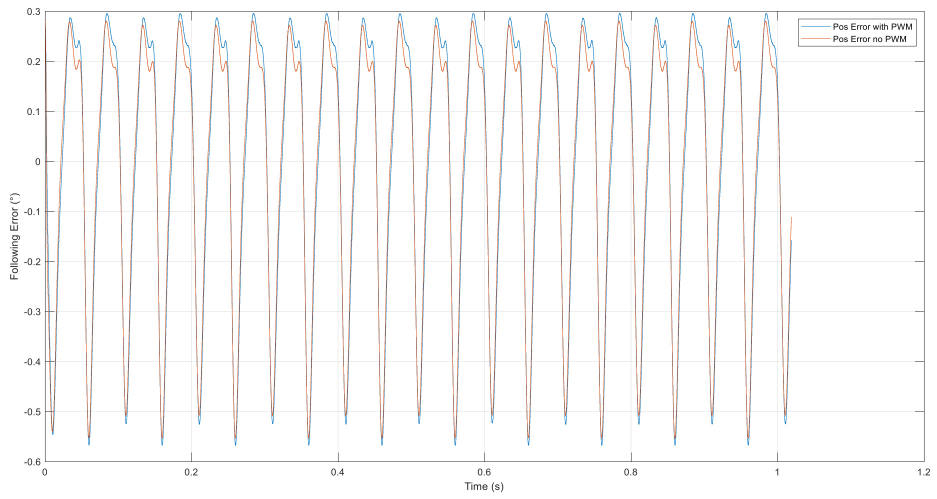


Figure 5-8 Following Error with PWM (blue) and without it (red)

We can see that the presence of modulation and inverter has very little effect on the following error of the system, as a matter of fact the difference is appreciable mostly at the peaks, where the error in the case with PWM is a little bit higher, being the difference about 2-2.5 % of the error without PWM.

After these considerations, for the sake of simplicity and simulation speed, we developed the next work using the model without the electrical part at all, since the simulation error is better in this case, as we have previously seen.

6 Friction model and parametrization

With the aim of reducing the simulation error, we tried to implement a simplified model of friction, which acts as a disturbance torque, taking into account both dry friction and viscous friction.

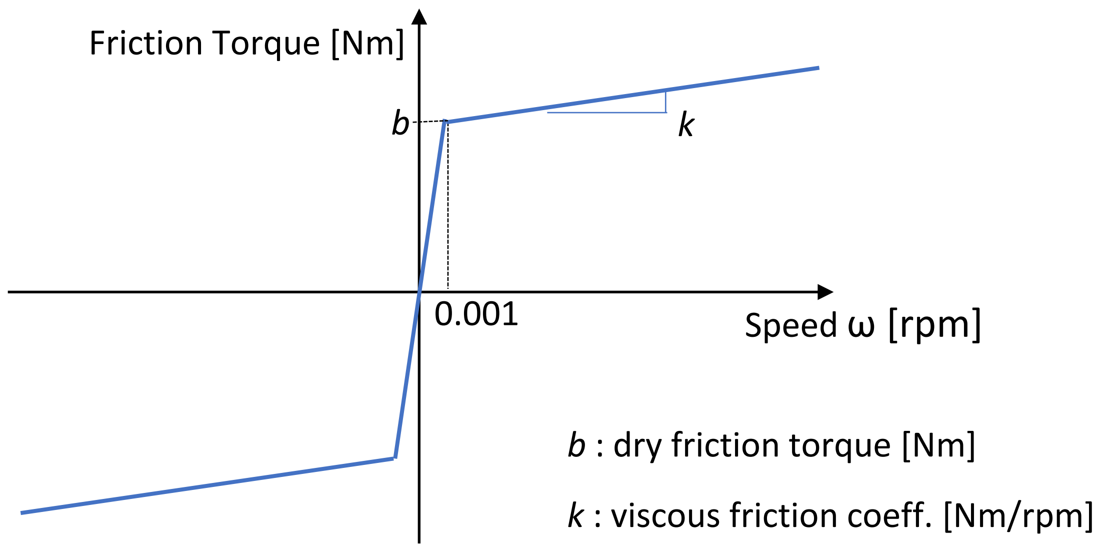
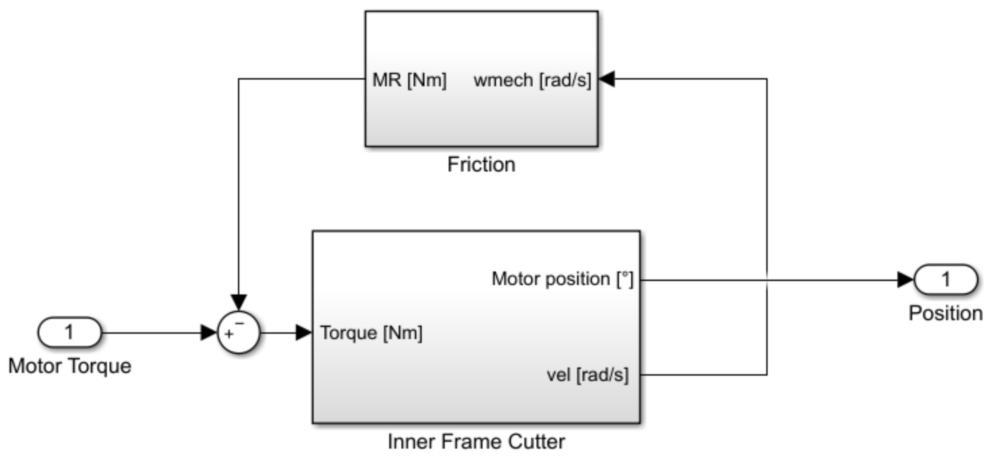


Figure 6-1 Friction function



The real system has got bearings in all joints, then we actually didn't expect it to be affected by a significant amount of friction.

The parameters *b* and *k* were unknown, of course, then they had to be estimated somehow, thus we followed this approach: running an optimization algorithm to minimize a cost function related to the simulation error changing the friction parameters, then, if the values obtained with different cams and velocities had been similar, then we would have considered them valid.

To do that, we used some functions of the Optimization Toolbox of Matlab, in particular we have first defined the cost function, then defined the optimization problem (initial condition and constraints) and finally ran the global search algorithm. Obviously, the algorithm has, in practice, a finite number of iterations, which implies that it is possible that it ends up finding a local minimum of the cost function instead of the global one.

The cost function is the following Matlab function:

```
function J_driveBoshOtt = J_driveBoshOtt_Friction(par)


        assignin('base','Sliding_friction_coeff’, par(1));
        assignin('base','Stiction', par(2));

        sim_out= sim('Controllo_senza_loop_di_corrente.slx',...
                    'ReturnWorkspaceOutputs','on');

        J_driveBoshOtt = rms(sim_out.actpos_real-sim_out.actpos_sim) +
        rms(sim_out.n_real-sim_out.n_sim);
end
```

It updates the friction parameters, runs the simulation, and computes the function to be minimized as the sum of the rms value of the position simulation error and the rms value of the velocity simulation error.
The optimization problem has the point (0,0) as starting point (no friction at all) and the bounds [0, 1e-3] for parameter *k* and [0, 0.2] for parameter *b*, those are values chosen after some manual test. We added bounds since a search over all the positive real numbers would be meaningless.

```
problem = createOptimProblem('fmincon',...
   'x0',[0;0],...
   'lb',[0;0],...
   'ub',[5e-3;0.4],...
   'objective',@J_driveBoshOtt_Friction);
```

At last, the global search is launched, imposing some limits on the number of iterations of the algorithm:

```
gs= GlobalSearch( 'FunctionTolerance',1e-4,'NumStageOnePoints', 200,
'NumTrialPoints', 500, ...
'StartPointsToRun', 'bounds');

[par, residual, exitflag] = run(gs, problem);
```

6.1 Simulation Results

The results, related to the different cams, are shown below.

- Cam 4, 600 ppm

  k = 6.33e-4 Nm/rpm   b = 0.0109 Nm

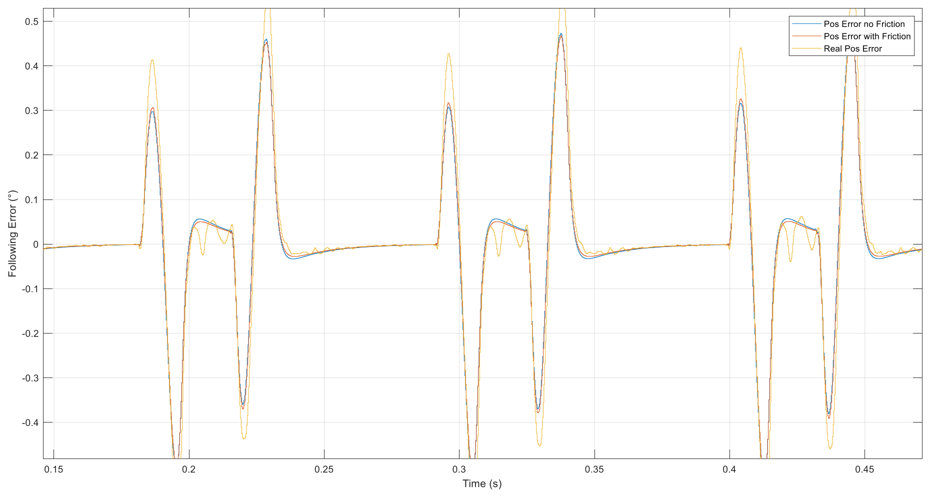


Figure 6-2 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 5, 600 ppm

  k = 3.97e-5 Nm/rpm   b = 0.1548 Nm

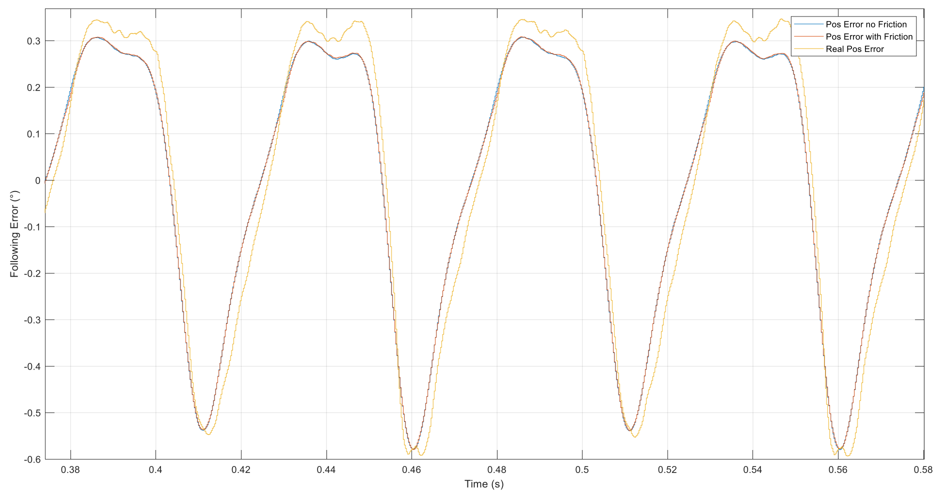


Figure 6-3 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 3, 400 ppm

  k = 2.53e-4 Nm/rpm   b = 0.1733 Nm

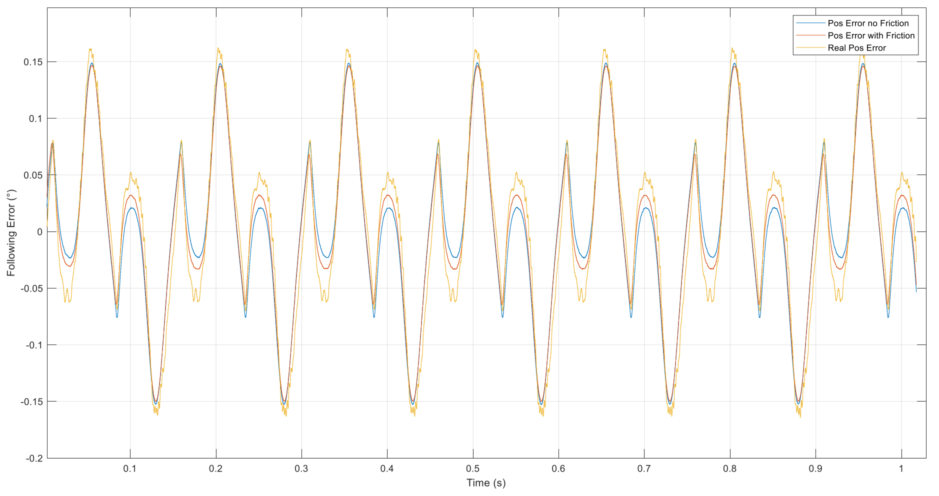


Figure 6-4 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 6, 400 ppm

  k = 4.14e-4 Nm/rpm   b = 0.1035 Nm

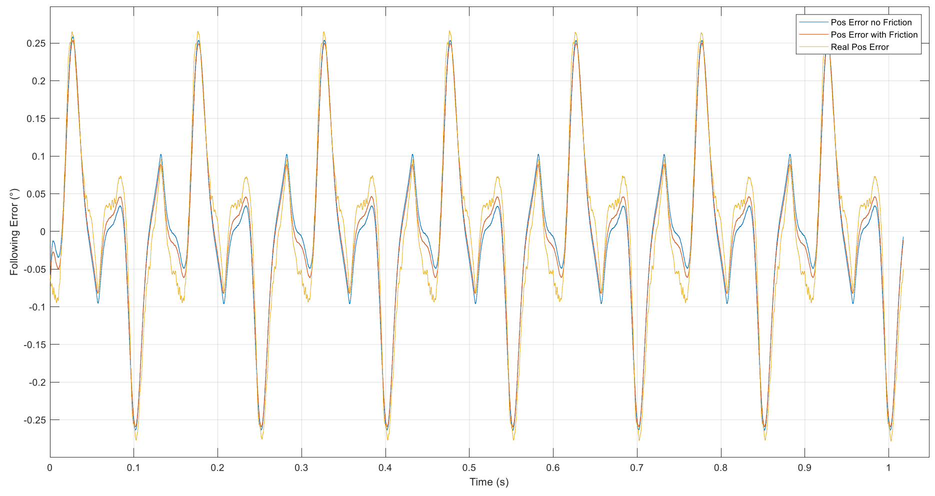


Figure 6-5 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 2, 200 ppm

  k = 0.0022 Nm/rpm   b = 0.18 Nm

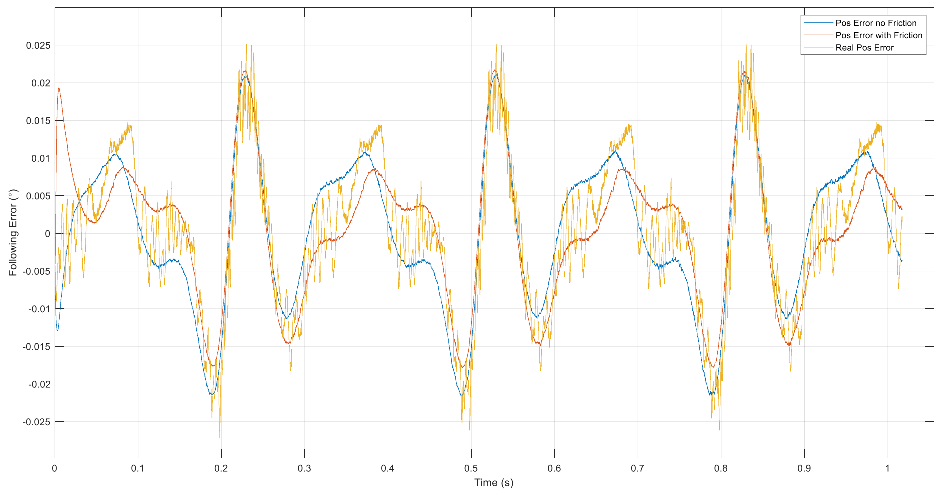


Figure 6-6 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 4, 200 ppm

  k = 0.0014 Nm/rpm   b = 0.1324 Nm

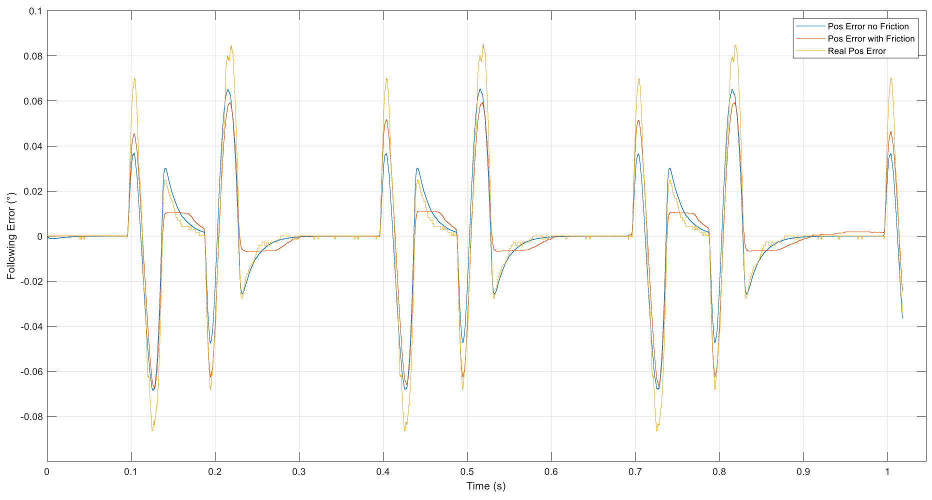


Figure 6-7 Real following error (green), simulated one with friction (red) and without it (blue)

- Cam 5, 80 ppm

  k = 0.0017 Nm/rpm   b = 0.0279 Nm

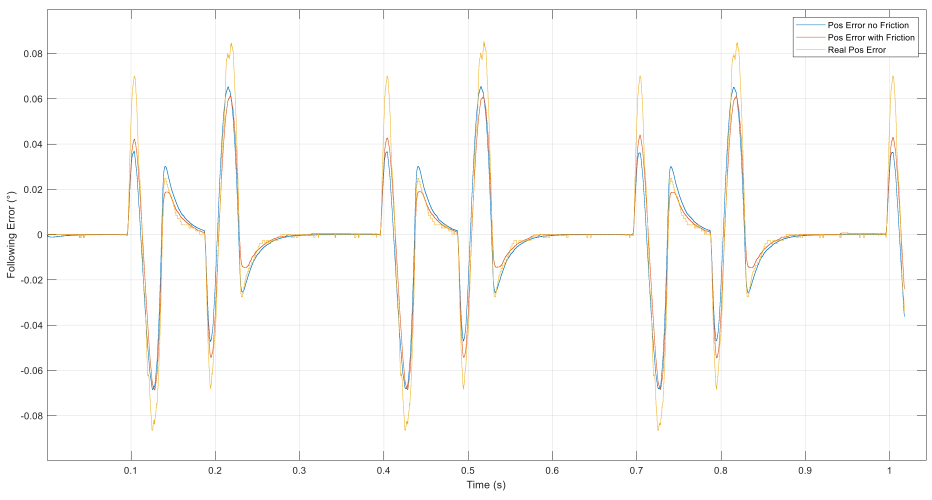


Figure 6-8 Real following error (green), simulated one with friction (red) and without it (blue)

The values of the parameters show a wide variability and the simulation error does not improve significantly, we can easily notice that the difference between the following error with friction and the one without friction is indeed negligible. Certainly, the oscillation of the following error with the cams at low speed is due to other mechanical phenomena.

For these reasons we decided not to study further this part, neglecting the effect of friction in the real system.

7 Cogging Torque Disturbance

As we said before, the quality of the simulation with respect to reality was low at low speed but high at high speed, this means that we had not modelled something that disturbs the system at low speed but loses effect as velocity increases. Then we thought that the source of disturbance could be the cogging torque acting on the motor.

Cogging arises from the interaction of the rotor magnets with the steel teeth on the stator, causes torque ripples and thus speed ripples in permanent magnet motor systems. It usually has greater effect at low speed, because at high speed it is filtered by the inertia of the mechanical load attached to the motor.
The cogging torque can be modelled in a simplified way as a sinusoidal function depending on the position of the rotor:

$$T = A * \sin(n * \theta + \varphi)$$

where θ is the rotor position and A, n, φ are parameters to be identified.
To do that, we proceeded in a similar manner as for the previous estimation of friction.
Simulating cam 1, 80 ppm, we did some manual trials to get an initial set of values, that could work as a first rough estimate that could be used in the following optimization process.

7.1 Identification through optimization of the cogging parameters

The cost function is the same as before, trying to minimize both the position and the velocity simulation error:

```
function J_driveBoshOtt = J_driveBoshOtt_Cogging(par)


        assignin('base','Ampl', par(1));
        assignin('base','freq_factor',par(2));
        assignin('base','phase_bias', - par(3));
        warning('off','all');
        sim_out= sim('Controllo_senza_loop_di_corrente.slx',...
                        'ReturnWorkspaceOutputs','on');

        J_driveBoshOtt = rms(sim_out.actpos_real-sim_out.actpos_sim) +
        rms(sim_out.n_real-sim_out.n_sim)
end
```

Also, the optimization problem has a similar structure:

```
Gs = GlobalSearch( 'FunctionTolerance',1e-4,'NumStageOnePoints', 200,
'NumTrialPoints', 400, ...
 'StartPointsToRun', 'bounds');

problem = createOptimProblem('fmincon',...
   'x0',[ 0.1; 59; 863.4 ],...
   'lb',[ 0.08; 40; 800 ],...
   'ub',[ 0.18; 70; 900 ],...
   'objective',@J_driveBoshOtt_Cogging);
[par, residual ,exitflag] = run(gs, problem);
```

We used mostly low-speed cams for this task, i.e. 80 ppm and 200 ppm, because the others are almost unaffected by cogging, then an optimization applied to them wouldn't have given meaningful results.

7.2 Simulation Results

- Cam 1, 80 ppm

  Ampl = 0.12 freq_factor = 59.8968 phase_bias = -870.0625

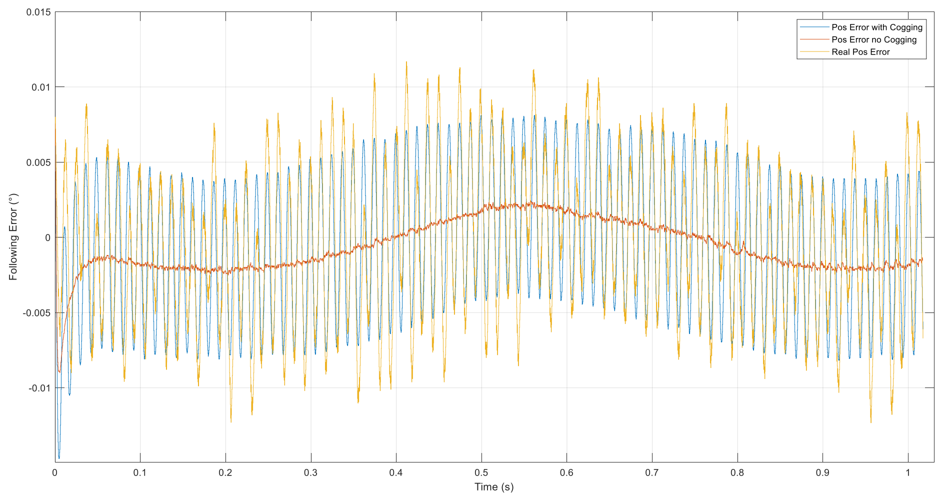


Figure 7-1 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 3, 80 ppm

  Ampl = 0.1323 freq_factor = 59.4272 phase_bias = -880.8459

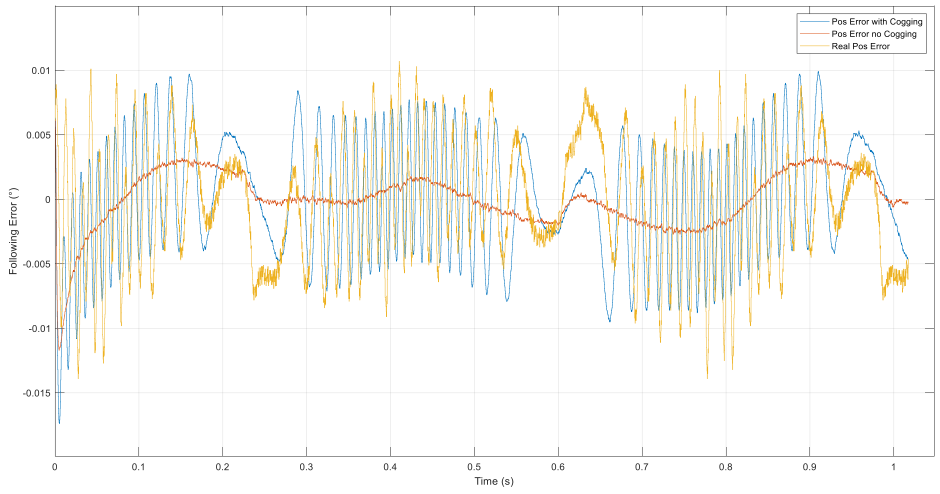


Figure 7-2 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 6, 80 ppm

  Ampl = 0.0963 freq_factor = 59.2686 phase_bias = -863.3811

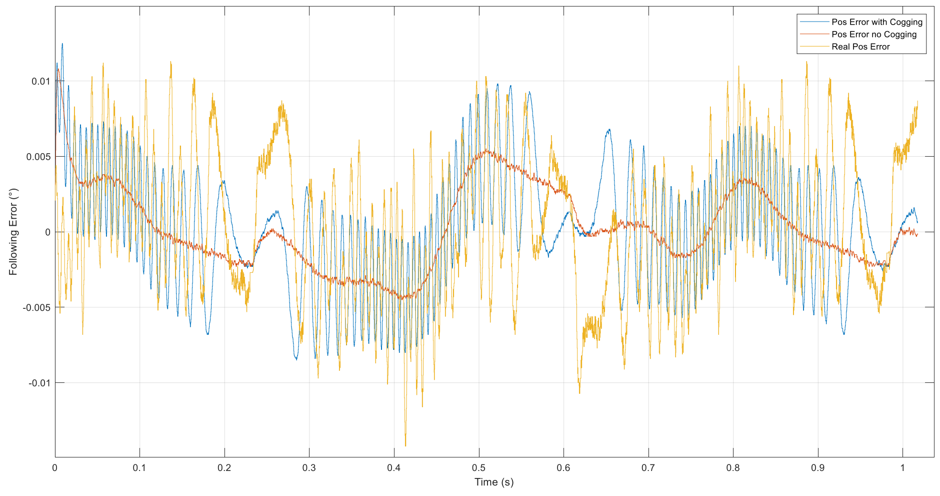


Figure 7-3 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 1, 200 ppm

  Ampl = 0.0971 freq_factor = 61.9314 phase_bias = -878.8221

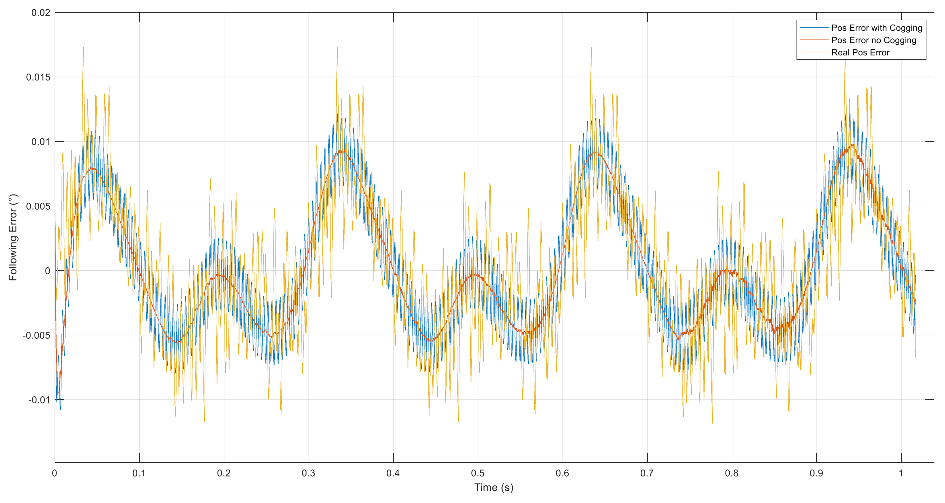


Figure 7-4 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 2, 200 ppm

  Ampl = 0.0971 freq_factor = 61.9318 phase_bias = -878.8221

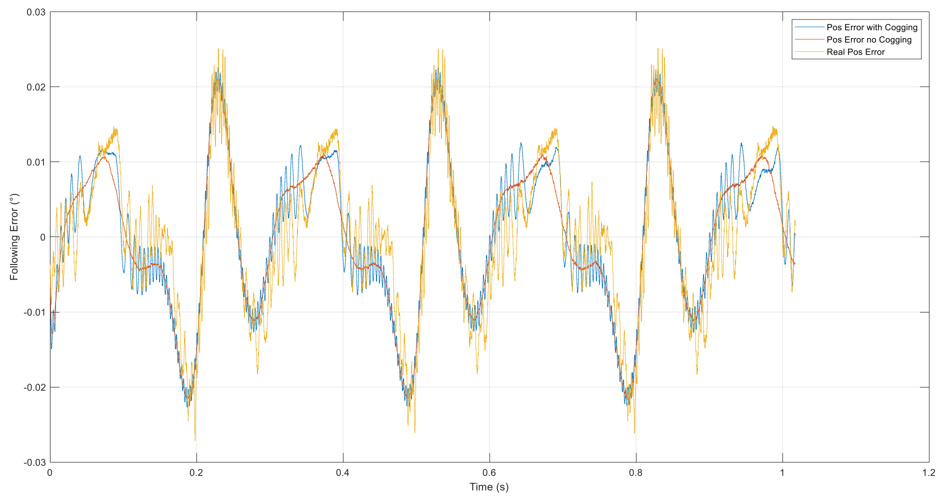


Figure 7-5 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 5, 200 ppm

  Ampl = 0.0931 freq_factor = 61.89 phase_bias = -811.8

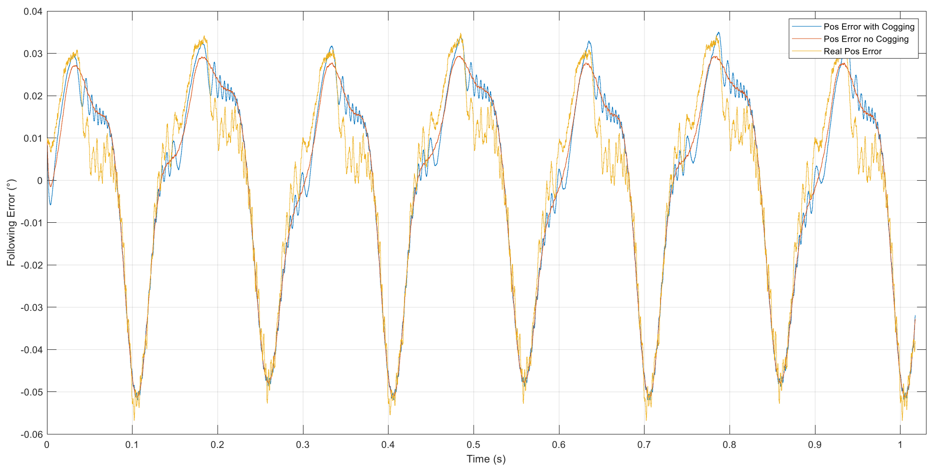


Figure 7-6 Real following error (green), simulated one with cogging (blue) and without it (red)

- Cam 1, 400 ppm

  Ampl = 0.098 freq_factor = 61.90 phase_bias = -879.4

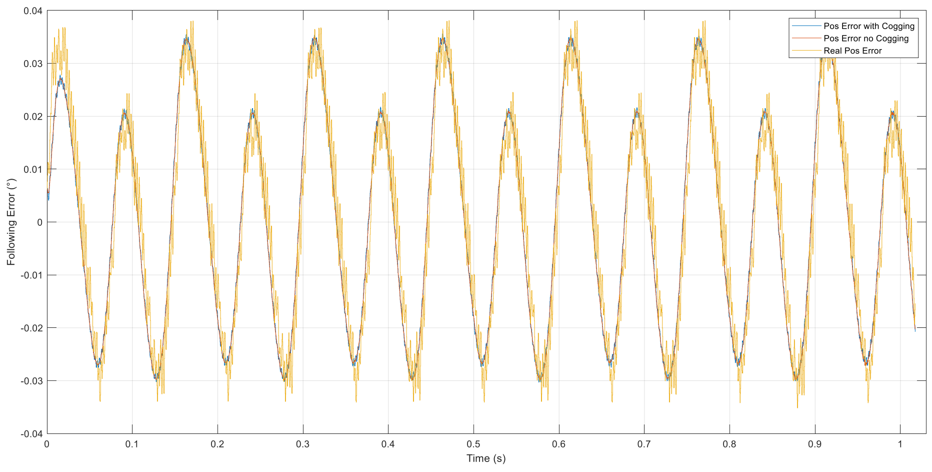


Figure 7-7 Real following error (green), simulated one with cogging (blue) and without it (red)

Adding the model of cogging, we were able to obtain a simulated following error that has the same oscillation frequency of the real one and is in phase with it.

However, the amplitudes do not match exactly, this is likely due to the fact that we assumed a constant amplitude, while, if we look at the figures, we can guess that it is a function of the position of the rotor. Identifying such function wouldn't prove easy, and it wouldn't be so useful, because low-speed operation is not, in general, of much interest, then we accepted such an approximation.
The final set of data we used thereafter was *Ampl = 0.098, freq_factor = 60.7, phase_bias = -878.*

8 Optimization of the controller parameters

Once that the model of the system was sufficiently accurate, we could use it to for the next step of the project: running an optimization process to find, through simulation, a better tuning of the controller parameters, the proportional gain of the position loop and the gains of the PI regulator of the velocity loop. The current controller is tuned automatically by the drive and, commonly, is not modified.
The final aim of this part of the work was to achieve a method to speed up the tuning of electrical axes in an automatic machine, because the manual tuning, although effective indeed, of a single axis requires quite a lot of time without guarantee to reach the maximum achievable performance.
However, optimization algorithms should be used with care, because the goodness of their results depend on user-made choices, like the definition of the cost function, which should contain some proper performance criteria for the specific task, and the limit on the algorithm iterations.

Theoretically, with an unbound number of iterations, a global optimization algorithm achieves for sure the global minimum of the cost function, but, in practice, we can't allow it to run for too long, then it may fail the search for the global minimum and find a local one.
For this reason, the starting point matters a lot, too. For if it is already close to the global minimum the algorithm is likely to succeed, while if it is an almost random starting point, the algorithm may return just an unsatisfactory local minimum. The degree of goodness of a local minimum depends, of course, on our goal.
Moreover, if we give some bounds to the variables of interest, we are in fact limiting the domain of the function, thus the algorithm will find the minimum of the function within such domain, which may not be the minimum on a broader domain.
So, some trade-offs must be chosen and accepted.

8.1 Optimization problem definition

Here, you can see the Matlab code for the global constrained optimization problem:

```
%% GLOBAL Optimiziation

gs= GlobalSearch( 'FunctionTolerance',1e-4,'NumStageOnePoints', 100,
'NumTrialPoints', 300, ...
 'StartPointsToRun', 'bounds');


problem = createOptimProblem('fmincon',...
    'x0',[ 416.67; 2.1; 140],...
    'lb',[   360; 1.1;  100],...
    'ub',[   470; 3.1; 180],...
    'objective',@J_driveBoshOtt_PID);
[par, residual ,exitflag] = run(gs, problem);
```

The values are *cp_P, cv_P, cv_I,* in this order. The starting point is the set of controller parameters found manually by technicians of the company. Since these parameters are already quite good, we decided at this stage to assign not so wide bounds.

About the cost function, we decided to use multiple cams, at 600 ppm, and give specifications on the following error both in the time domain and in the frequency domain. Let us see an example in the following script.

```
function J_driveBoshOtt = J_driveBoshOtt_PID(par)

    assignin('base','cp_P',par(1));
    assignin('base','cv_P',par(2));
    assignin('base','cv_I',par(3));

    warning('off','all');

    for i=3:5

          assignin('base','file_name',strcat('Dati8_600_rpm_camma',num2str(i),'.txt'
          ));
          evalin('base','ImportData_8_Signals_06_07; Initial_conditions;
          Disturbo_Coppia_cogging;');
      sim_out= sim('Controllo_senza_loop_di_corrente.slx',...
                      'ReturnWorkspaceOutputs','on');
                    freq_content_pos_error=FFT(sim_out.sim_pos_error(1:4074));
      max_freq_pos_error(i)=max(freq_content_pos_error);
      rms_freq_pos_error(i)=rms(freq_content_pos_error);
      abs_err_max(i)=max(abs(sim_out.sim_pos_error(1:4074)));
      error_rms(i)=rms(sim_out.sim_pos_error(1:4074));
    end
    a = rms(max_freq_pos_error);
    b = rms(rms_freq_pos_error);
    c = rms(abs_err_max);
    d = rms(error_rms);

    J_driveBoshOtt = rms([a,b,c,d])
end
```

At each iteration, this function writes the new controller parameters, then runs the simulation, with the model without the current loop, for three cams (3, 4 and 5 in this example). For each cam, it finds the maximum and the rms value of the position error, it computes the Fast Fourier Transform of it and again finds the maximum and the rms value of the spectrum.
After the three simulations are done, we compute the rms values of the arrays of the previous variables of interest and, finally, the cost function is the rms of those rms values. So, in brief, we want to minimize the amplitude of the vector whose elements are the amplitudes of the vectors of the measurements mentioned before.
We performed the optimization three times, with different sets of cams.

8.2 Simulation Results

1. Cams 1-2-3, 600 ppm
   Results: *cp_P = 406.8961 [1/s], cv_P = 3.0930 [Nm/rad/s], cv_I = 155.9560 [Nm/rad]*

- *Cam 1*

  The new following error is about 28% lower than the old one.

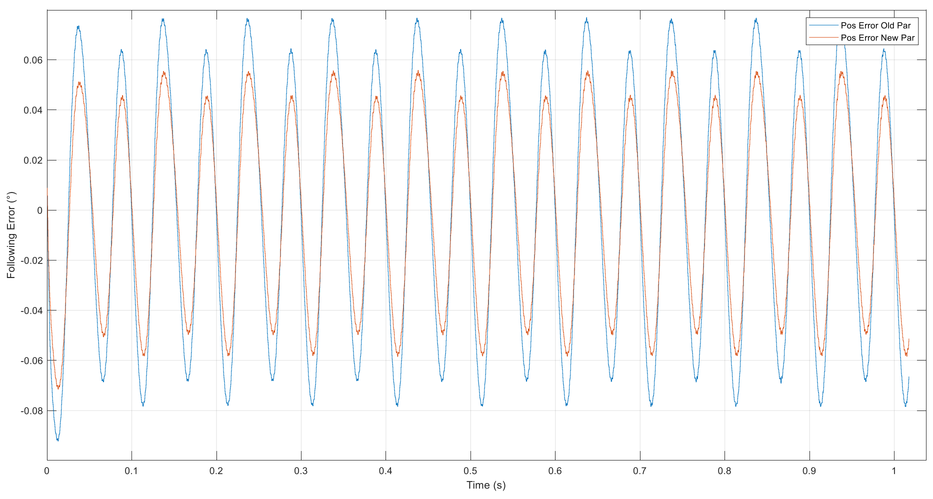


Figure 8-1 Following error with initial parameters (blue) and new ones (red)

- *Cam 2*

  The new following error is about 34% lower than the old one.

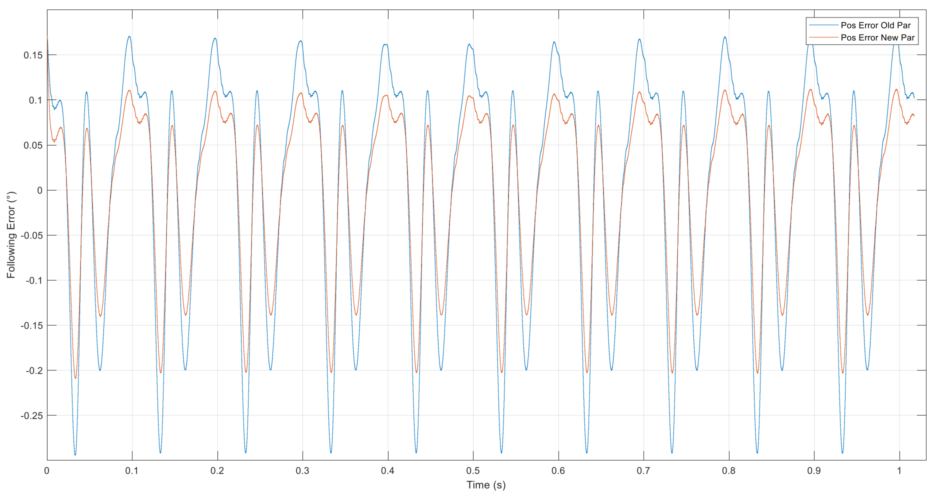


Figure 8-2 Following error with initial parameters (blue) and new ones (red)

- *Cam 3*

  The new following error is about 33% lower than the old one.

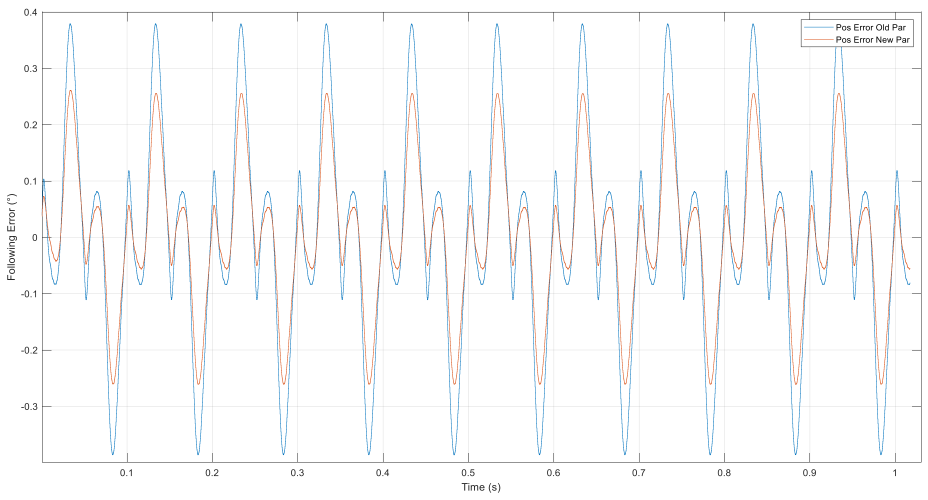


Figure 8-3 Following error with initial parameters (blue) and new ones (red)

2. Cams 3-4-5, 600 ppm

   Results: *cp_P = 418.8950 [1/s], cv_P = 2.7882 [Nm/rad/s], cv_I = 159.8563 [Nm/rad]*

- *Cam 3*

  The new following error is about 27% lower than the old one.

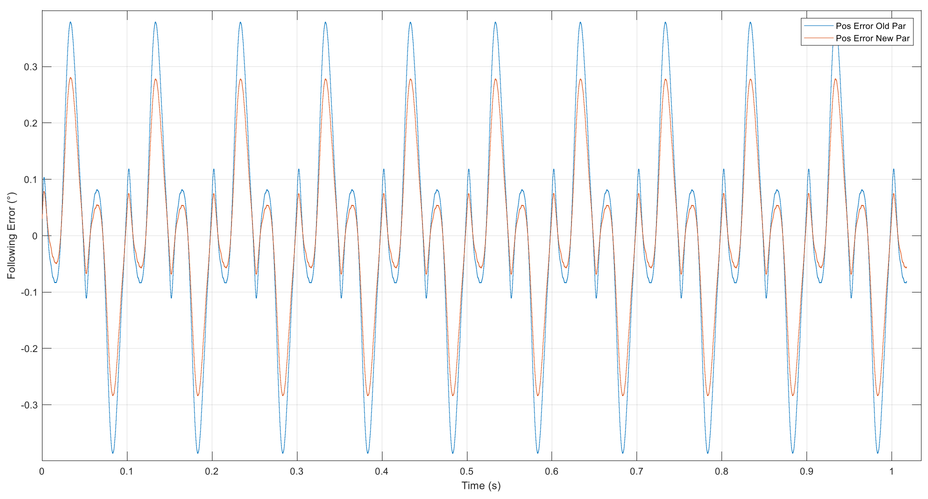


Figure 8-4 Following error with initial parameters (blue) and new ones (red)

- *Cam 4*

  The new following error is about 35% lower than the old one.

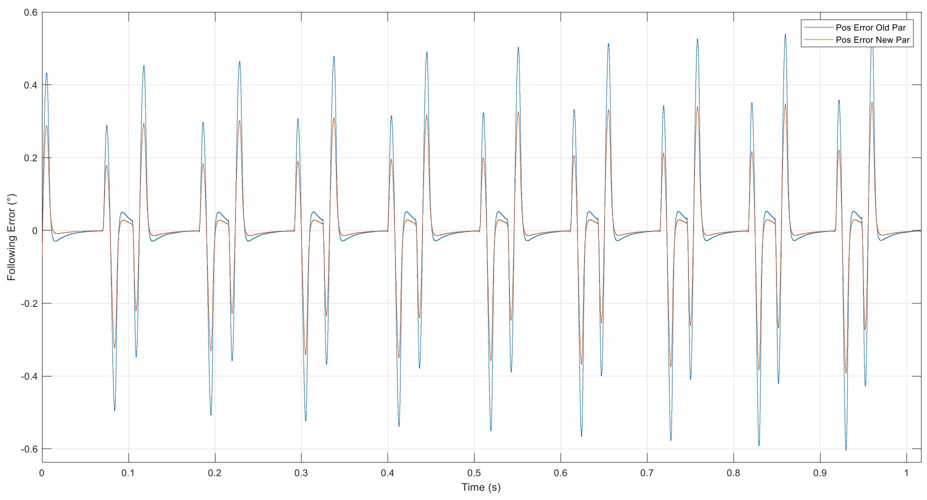


Figure 8-5 Following error with initial parameters (blue) and new ones (red)

- *Cam 5*

  The new following error is about 30% lower than the old one.

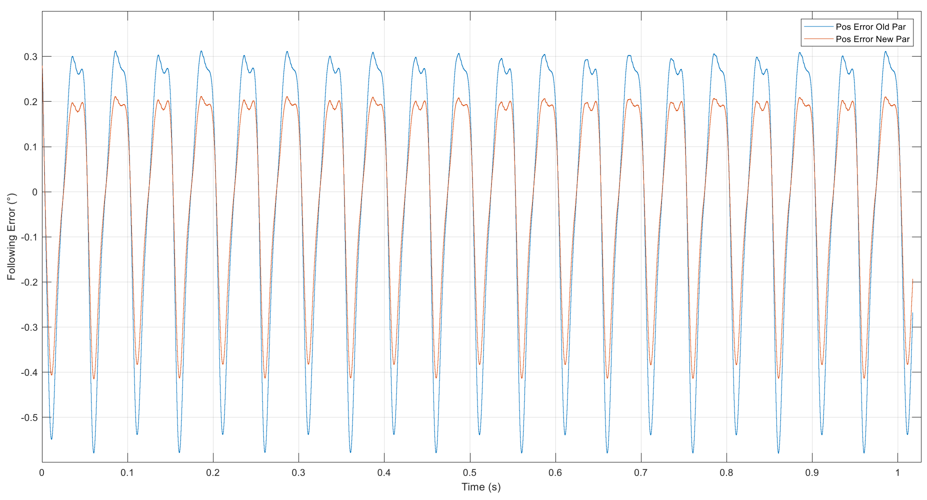


Figure 8-6 Following error with initial parameters (blue) and new ones (red)

3. Cams 4-5-6, 600 ppm

   Results: *cp_P = 425.6201 [1/s], cv_P = 2.8357 [Nm/rad/s],*

   cv_I = 146.8367 [Nm/rad]

- *Cam 4*

  The new following error is about 38% lower than the old one.

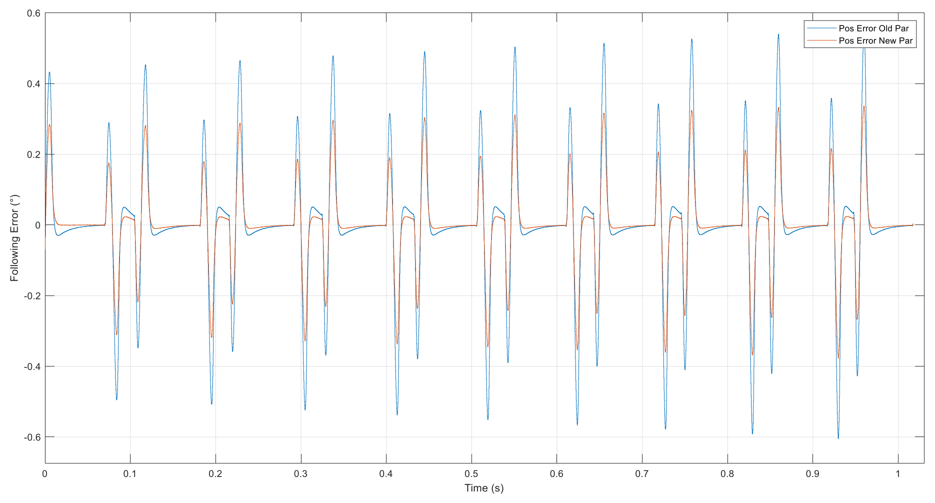


Figure 8-7 Following error with initial parameters (blue) and new ones (red)

- *Cam 5*

  The new following error is about 32% lower than the old one.

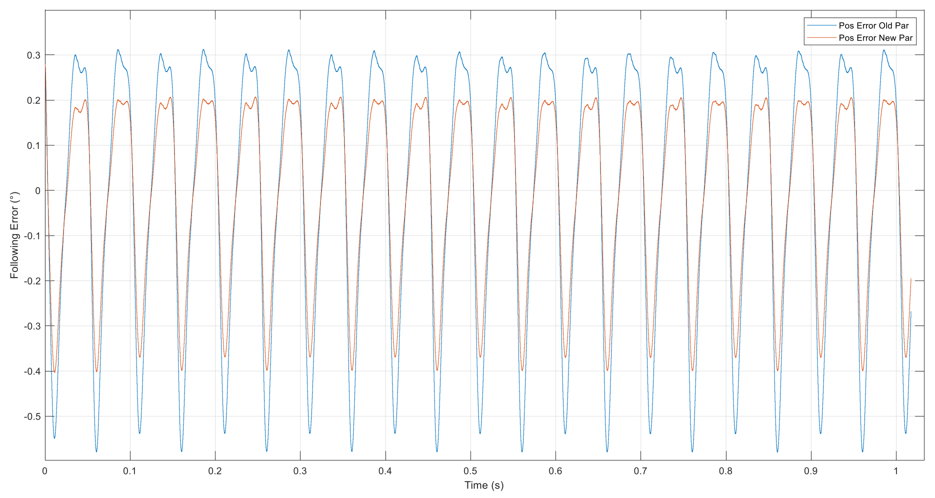


Figure 8-8 Following error with initial parameters (blue) and new ones (red)

- *Cam 6*

  The new following error is about 29% lower than the old one.

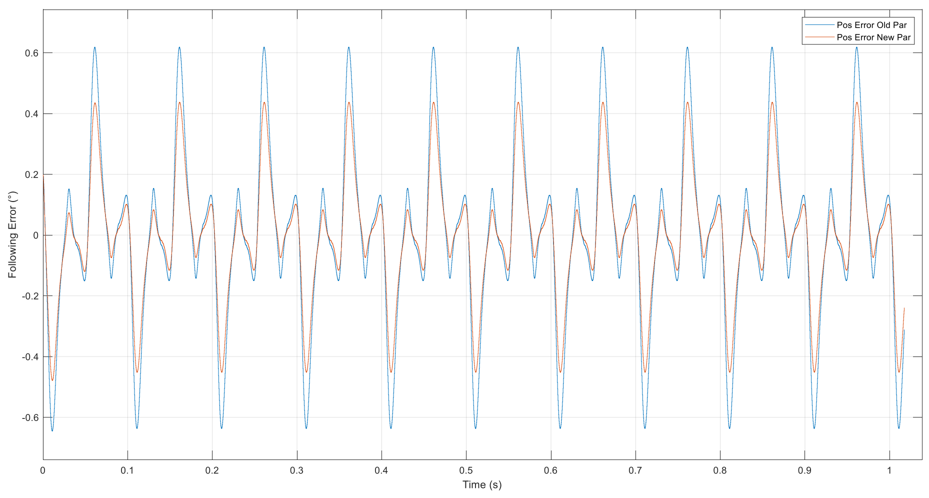


Figure 8-9 Following error with initial parameters (blue) and new ones (red)

We can see that the optimization process led to an improvement of the simulated position error by 30 % on average, in spite of the quite strict boundaries on the controller parameters assigned in the problem definition.
Then, we set the new values of the parameters on the real drive and read the data to understand if the improvement could take place in the real system, too. In particular, we used the parameters got from the set of cams 4-5-6.

8.3 Application of the optimized parameters to the real system

In the following figures, the new data and the old ones are not phased because, obviously, they were taken through different experiments, anyway looking at the peaks we can understand how much we gained in terms of performance.

1. *Cam 1, 600 ppm*

   The new following error is about 19% lower than the old one.

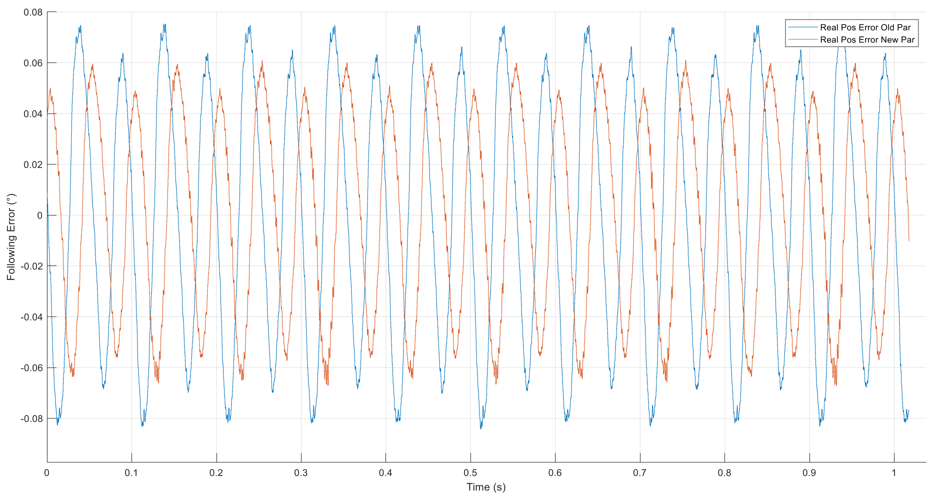


Figure 8-10 Following error with initial parameters (blue) and new ones (red)

2. *Cam 2, 600 ppm*

   The new following error is about 25% lower than the old one.

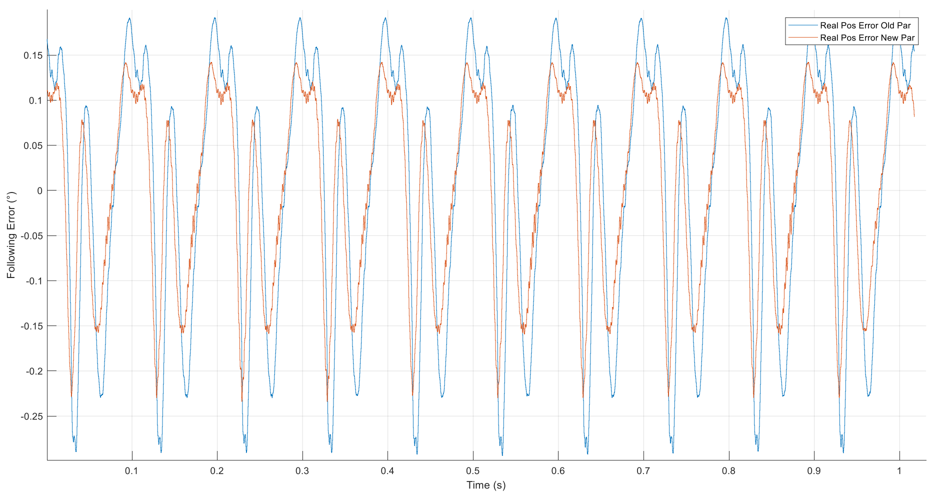


Figure 8-11 Following error with initial parameters (blue) and new ones (red)

3. *Cam 3, 600 ppm*

   In this case, the new parameters produce some speed ripple, leading to high frequency oscillations of the following error. Yet, the low-frequency content is lower than the old one. The simulation with the model including the electric part produces a non-oscillating following error, then we can say that this ripple is caused by some mechanical phenomenon not modelled yet.

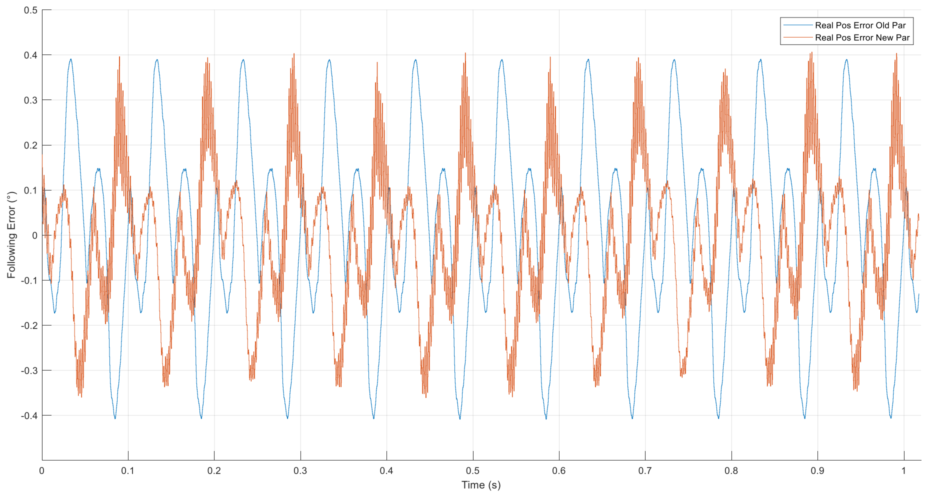


Figure 8-12 Following error with initial parameters (blue) and new ones (red)

4. *Cam 4, 600 ppm*

   The new following error is about 29% lower than the old one.

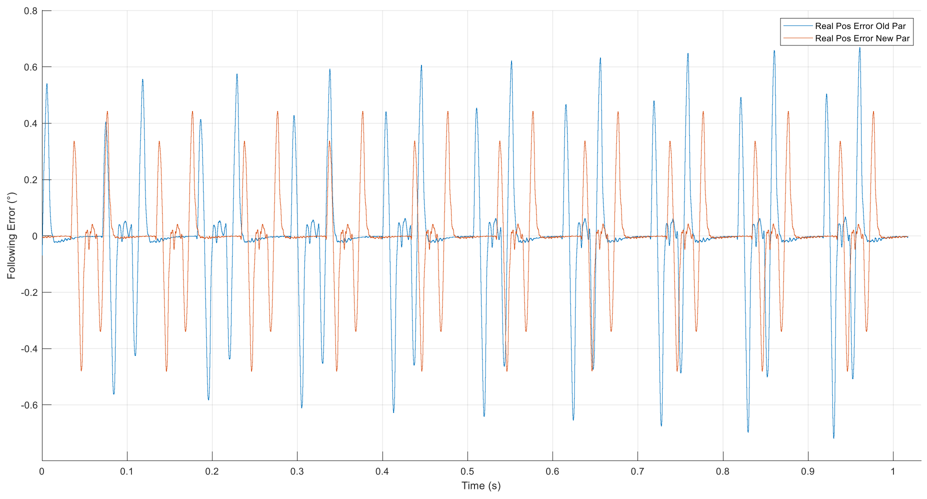


Figure 8-13 Following error with initial parameters (blue) and new ones (red)

5. *Cam 5, 600 ppm*

   The new following error is about 24% lower than the old one.

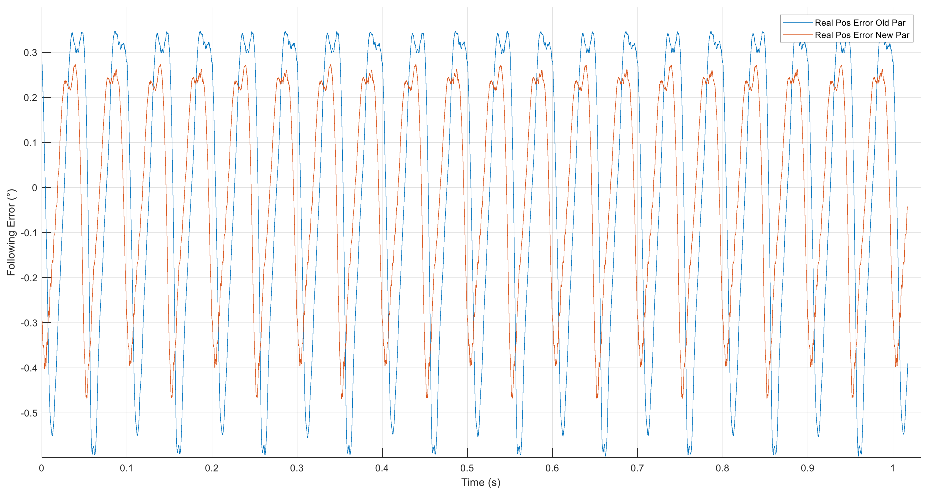


Figure 8-14 Following error with initial parameters (blue) and new ones (red)

6. *Cam 6, 600 ppm*

   The new following error is about 25% lower than the old one.

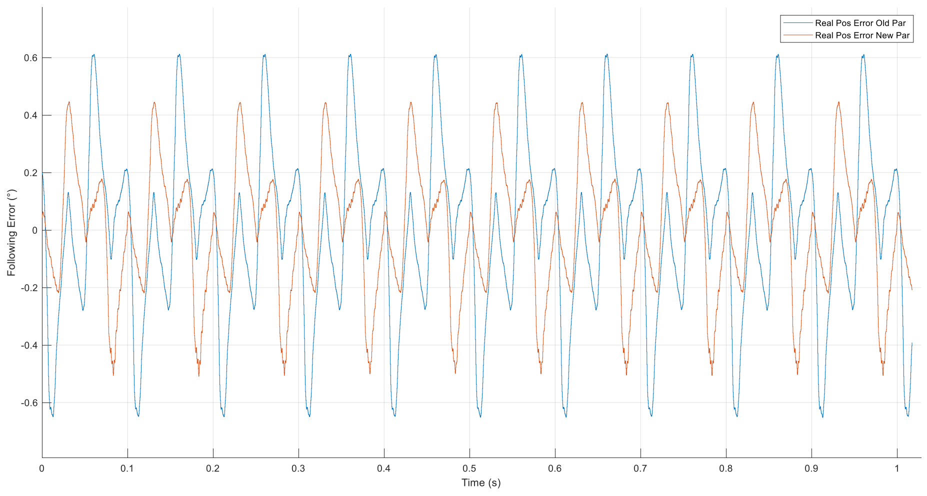


Figure 8-15 Following error with initial parameters (blue) and new ones (red)

The improvement in the real system with those new parameters is 25% on average, a few percentage points lower than the one in the simulation environment because of the ever-present uncertainties of the model.
A decrease by one-fourth of the following error is, anyway, not negligible.
If it is worth the hours needed by the optimization algorithm to search the set of parameters, that is up to the company and depends on the degree of accuracy required by the task.

That previous optimization could be seen as a refinement of an already good set of controller parameters, but we took into consideration the possibility that the real optimum may be quite different from the parameters found manually, then we ran a second optimization, this time widening the ranges for the parameters.

```
problem = createOptimProblem('fmincon',...
   'x0',[418.9;2.8;160],...
   'lb',[300;0.5;80],...
   'ub',[550;5.1;240],...
   'objective',@J_driveBoshOtt_PID);
```

The resulting parameters, applied to the real drive, produced a highly noisy behavior, because speed showed high frequency oscillations, as we can see in the figures below.

- Cam 1, 600 ppm

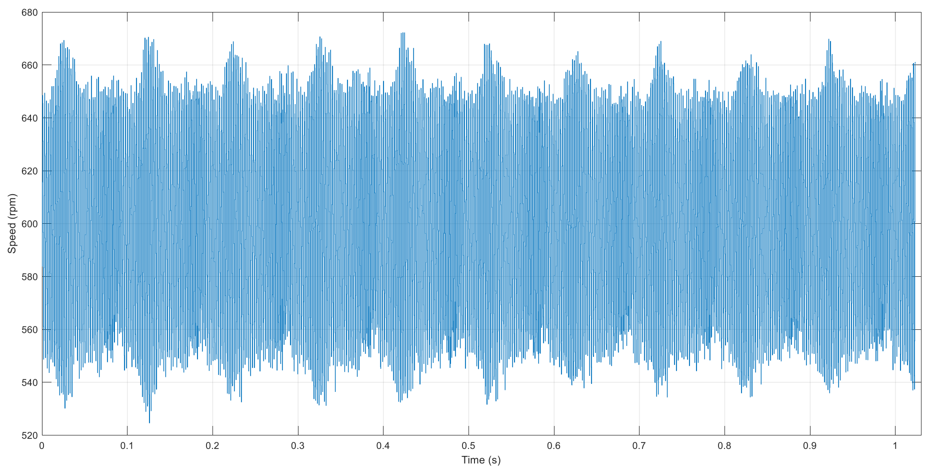


Figure 8-16 Real motor speed (rpm)

- Cam 3, 600 ppm

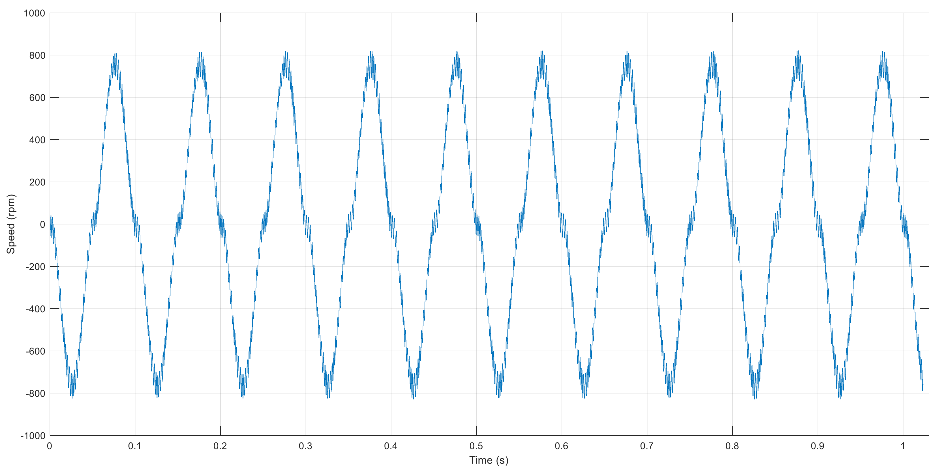


Figure 8-17 Real motor speed (rpm)

- Cam 2, 600 ppm

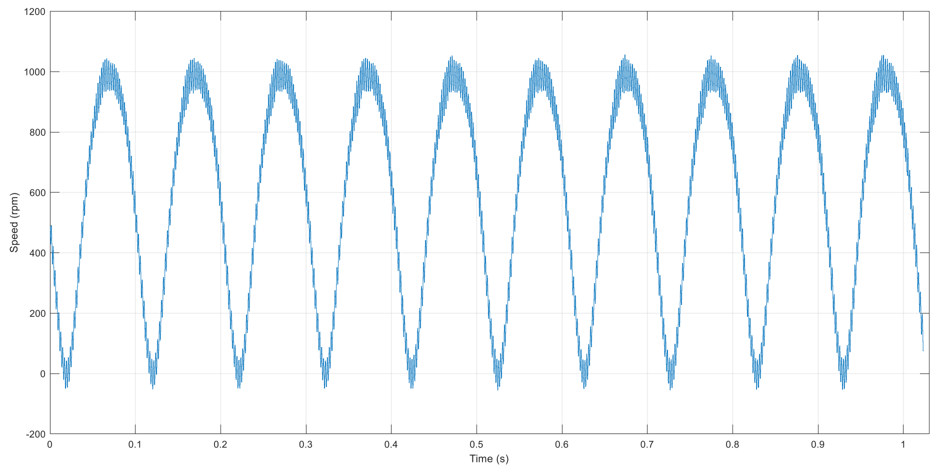


Figure 8-18 Real motor speed (rpm)

Such ripple was not present in the simulated data with the model without current loop, we initially supposed that it might be due to some mechanical resonances that could not occur in the simulation environment since the mechanical model is perfectly rigid; but later we ran the simulation with the model including the current loop and the speed oscillations showed up, then we understood that the actual reason for this undesired operation was that the new parameters had pushed the

bandwidth of the velocity loop too high in the frequency domain, getting too close to the bandwidth of the current loop.
Consequently, we had to find a new cost function that could avoid high gains at high frequencies.

8.4 Redefinition of the cost function

We ended up with the following cost function:

$$J = k_1 * \max|Pos\ Err| + k_2 * frequency\_domain\_term$$

where k_1 and k_2 are constant weights.
The second term aims at minimizing the variance between the spectrum of the position reference signal and the spectrum of the actual position, and it is computed as described now:

1. Compute the Fast Fourier Transform of the two signals;
2. Make the difference between the two spectra;
3. Slide a window of 100 Hz over the frequency axis computing the variance of that difference in each window;
4. Sum all the variances with an increasing weight as the frequency increases. So, we are saying that a possible peak at high frequency is worse than a peak at low frequency.

This sum is the term in the cost function.

Such procedure is implemented with few lines of code:

```
DIFF = FFT(sim_out.refpos_sim) - FFT(sim_out.actpos_sim);
SUM = 0;
for i = 1:1900
   VAR(i) = var(DIFF(i:i+99));

   SUM = SUM + (i/10)*VAR(i);
end
```

Since the reference signal and the feedback signal are sampled at a frequency of 4 kHz, the FFT generates the spectra of the signals from 0 to 2 kHz.

Unfortunately, the correct selection for the weights k_1 and k_2 is not so easy, the two terms of the cost function should have the same order of magnitude, with a slightly greater importance to the frequency domain term, as a behavior with a small error but a speed ripple is not an acceptable solution at all. The variable *SUM* is in the order of thousands, approximately from five to ten thousand, while the maximum error is approximately from 0.3° to 1°, then, at first, we used k_1= 2 and k_2 = 1/1000.

8.5 Simulation Results

The resulting parameters, put in the real drive, led to a position error very similar to the initial one, just a little bit lower.

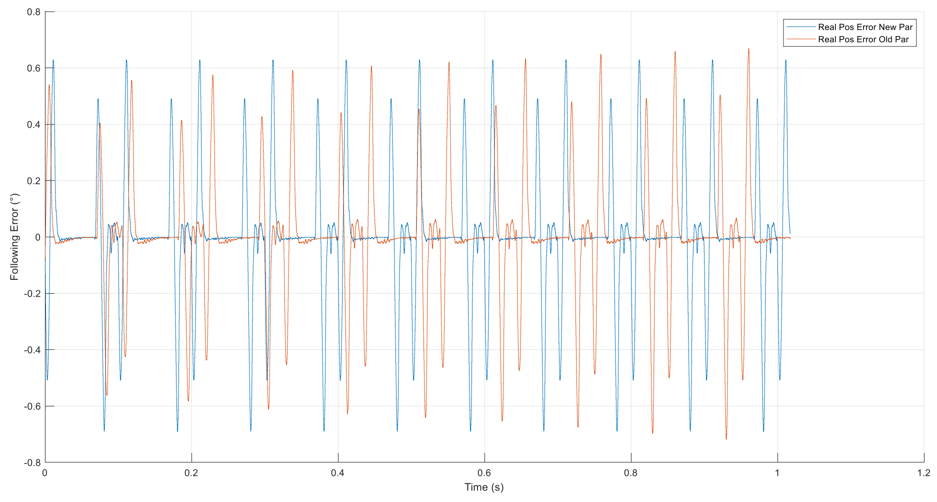


Figure 8-19 Following error with initial parameters (red) and with new ones (blue)

Clearly, such a small improvement is not worth the time required by the optimization algorithm; that is likely due to an excessive weight of the frequency term, so we tried to increase the weight of the maximum error setting k_1= 10. This time the new parameters produced effectively a smaller following error but again the operation was noisy, because of some speed ripple. A fine tuning of the weights requires a time-consuming trial-and-error procedure that nullifies any gains in terms of time spent that optimization may provide.

Consequently, we decided to try another type of approach, relying on the classic theoretical knowledge for the tuning of PI regulators.

9 Control tuning based on system inertia and loop bandwidth

The mechanism to be controlled is obviously a nonlinear system, but we may linearize it identifying its overall inertia around the different angles. Once a linear model is obtained, we can design the controller to fulfill the common requirements used for SISO linear control systems, such as loop bandwidth, which determines how fast the system responds to a stimulus, and phase margin, which somehow states how much robust the system is with respect to disturbances and uncertainties on the controlled parameters.

9.1 Inertia identification

The first step was to find the values of the inertia, referred to the motor shaft, for the different motor positions. We know that

$$T(t) = J(t) * \ddot{\theta}(t) + \frac{dJ(t)}{dt} * \dot{\theta}(t)$$

where T is the torque, J is the inertia, $\ddot{\theta}(t)$ and $\dot{\theta}(t)$ are acceleration and speed, respectively. Actually, we know the inertia to be a function of the motor angle, since the mechanism has just one degree of freedom, then we can write

$$T(t) = J(t) * \ddot{\theta}(t) + \frac{dJ(\theta)}{d\theta} * \dot{\theta}^2(t)$$

A table with the values for $\frac{dJ(\theta)}{d\theta}$ for the angles from 0° to 360°, at steps of 1°, is commonly provided by mechanical engineers of the company.
Using the Simscape model of the system and imposing a certain motion profile, we could read the actuation torque and, then, compute $J(t)$.

In particular, we imposed a constant acceleration of 0.01 rad/s^2.

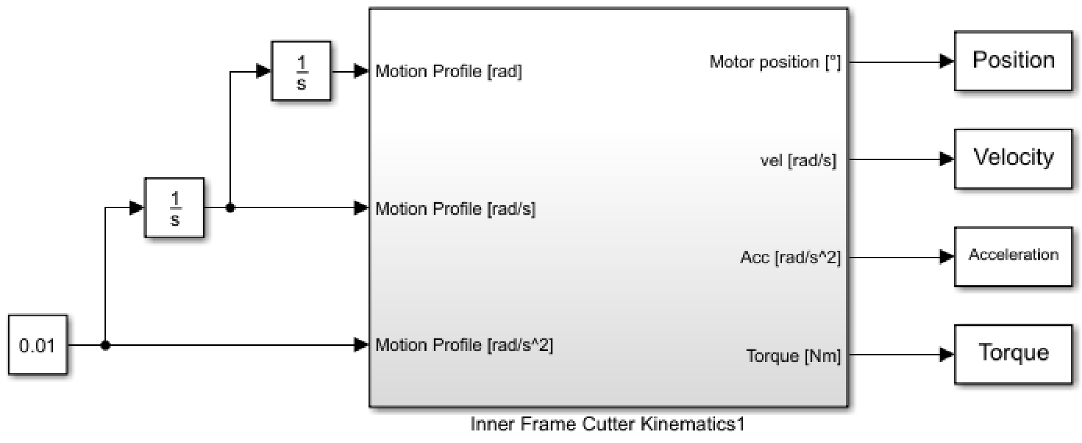


Figure 9-1 Simscape model of the mechanism

The inertia varies over 360 degrees as the next plot shows:

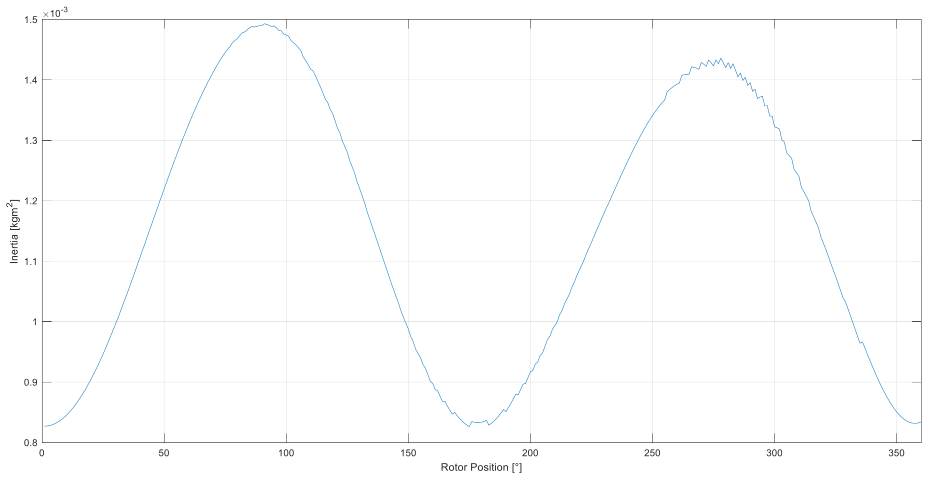


Figure 9-2 Inertia as function of the rotor angle

As we could expect, it is almost a sinusoidal function of the angle of the motor.

Minimum value : *8.2626e-04 kgm^2*   Maximum value : 0.0015 kgm^2

9.2 Linearization and automatic tuning

At that point, we made a conservative choice selecting the minimum of the inertia as the value used for the linearization. So, we were sure enough that the bandwidth of the velocity loop wouldn't have overcome the limit imposed by us, accepting, however, that at higher values of the inertia the system could have been slower.

The linear model of the mechanical system, neglecting friction, becomes simply $\frac{1}{J_{min}*s}$, then the overall scheme of the control loops (in continuous time) is:

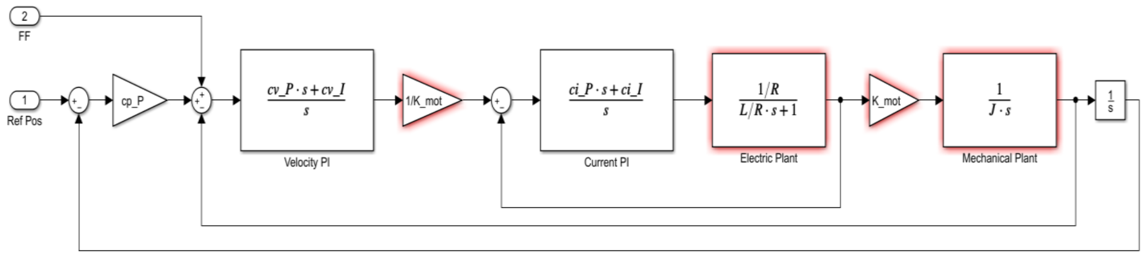


Figure 9-3 Control Scheme

The parameters of the current controller are set automatically by the drive and should not be modified, usually. Therefore, our degrees of freedom are the position and the velocity regulators.
We took advantage of the Control System Toolbox of Matlab to write a code that automatically generates the values of *cp_P, cv_P, cv_I* based on a specification on the phase margin and the crossover frequency of the velocity open-loop transfer function and the crossover frequency of the position open-loop transfer function.

```
%%
Electric_Plant = tf(1/mr_R_s,[mr_L_q/mr_R_s 1]);

Current_Controller = tf([ci_P ci_I],[1 0]);

Current_Closed_Loop = Electric_Plant*Current_Controller/(1 +
Electric_Plant*Current_Controller);

Current_Bandwidth = bandwidth(Current_Closed_Loop);
%%
Min_Inertia = 8.2626e-04;

Mechanical_Plant = tf(1,[Min_Inertia 0]);

opt = pidtuneOptions('PhaseMargin',75,'DesignFocus','reference-tracking');

Wc = Current_Bandwidth/1.2; %Crossover frequency [rad/s]

Speed_Controller =
pidtune(Current_Closed_Loop*Mechanical_Plant,'PI',Wc,opt)
```

```
Speed_Closed_Loop =
Speed_Controller*Current_Closed_Loop*Mechanical_Plant/(1+Speed_Controll
er*Current_Closed_Loop*Mechanical_Plant);

Integrator = tf(1,[1 0]);

Position_Controller = pidtune(Speed_Closed_Loop*Integrator,'P',Wc/5)
%%
cp_P = Position_Controller.Kp;
cv_P = Speed_Controller.Kp;
cv_I = Speed_Controller.Ki;
```

We selected quite a high phase margin for robustness and for avoiding overshoots. In order to get high performances, similar to those obtained by tuning by hand, we had to push the velocity loop bandwidth close to the current loop bandwidth, even the position loop has a high bandwidth, almost one fifth of the velocity loop one.

The frequential separation of the loops is not as theory usually suggests; the PWM frequency is only 4 kHz, then the current loop has a bandwidth of 400 Hz only, consequently the velocity loop bandwidth cannot be much lower than that if we want to achieve the desired performances.

Here are the parameters obtained running the script:

cp_P = 431.7 [1/s]; cv_P = 2.38 [Nm/rad/s]; cv_I = 92.5 [Nm/rad].

Now we will compare the real following error with these values and with the old ones (those tuned by hand).

- Cam 1, 600 ppm

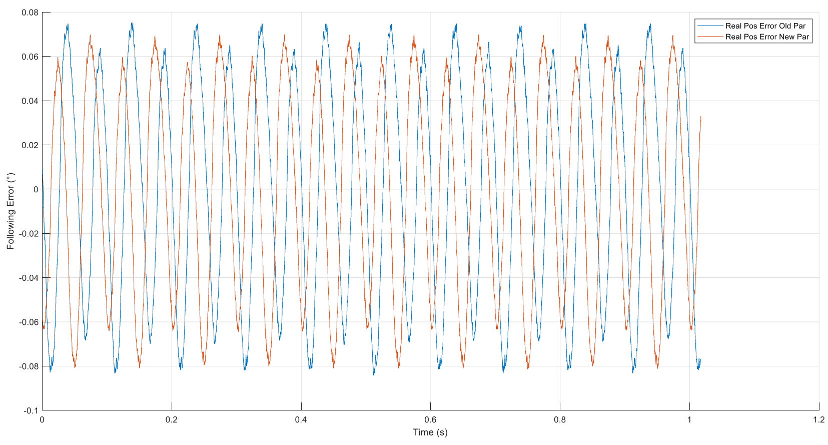


Figure 9-4 Following error with initial parameters (blue) and new ones (red)

- Cam 4, 600 ppm

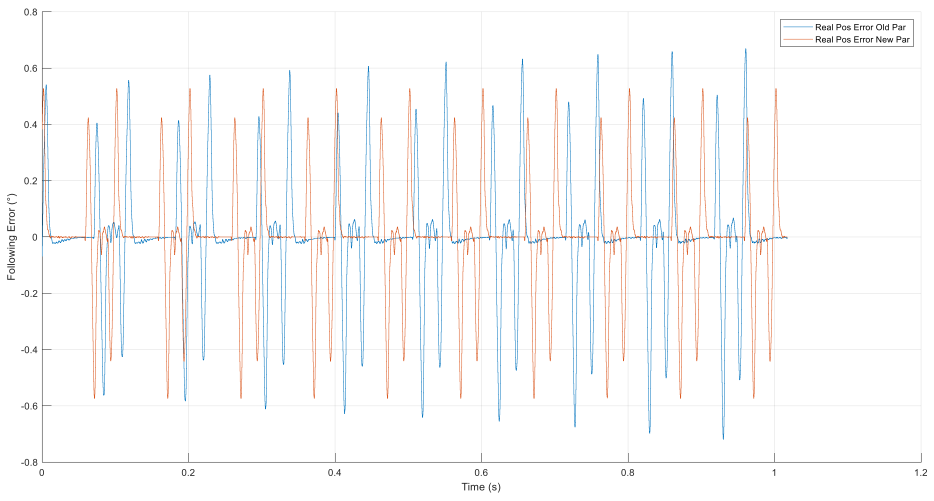


Figure 9-5 Following error with initial parameters (blue) and new ones (red)

- Cam 6, 600 ppm

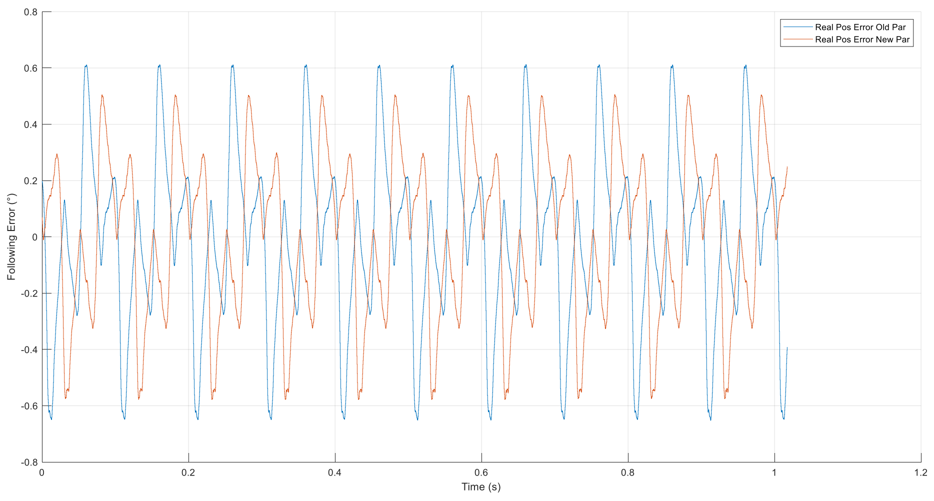


Figure 9-6 Following error with initial parameters (blue) and new ones (red)

We see that even with the conservative choice of Jmin for the inertia, the new following error is better than the previous one. The improvement is about 16 %. This is an important result, because we proved that with just some theoretical considerations and few lines of code we can get in few seconds better results than those achieved after a long procedure of tuning by hand.

Later, we also tried to run the script using Jmax instead of Jmin, but the new parameters (*cp_P = 431.7 [1/s]; cv_P = 4.33 [Nm/rad/s]; cv_I = 168 [Nm/rad])* pushed the velocity loop bandwidth too high with respect to the current loop bandwidth whenever the inertia was lower than the maximum value, then a noisy behavior occurred in the real experiment.

9.3 Optimization algorithm and results

So, we decided to run an optimization using the values from Jmin and Jmax as bounds for the ranges to be explored. Actually, the gain of the position regulator is the same for both cases, only the velocity regular should be optimized.
The cost function considers the maximum error and the variance of the difference between the FFT of the velocity reference and the FFT of the velocity feedback, in a similar manner as we discussed before.
The weights were decided such that the cost function had a lower value for the parameters corresponding to Jmin than for the parameters corresponding to Jmax.
The optimization led to the following parameters: *cp_P = 431.7 [1/s]; cv_P = 2.91 [Nm/rad/s]; cv_I = 148.7 [Nm/rad]*. Such controller managed to achieve smaller maximum error without experiencing velocity ripple.
The next plots compare the following error from parameters obtained with Jmin and the error from optimized parameters, showing that the second is indeed lower than the first one, so the optimization was successful.

- Cam 1, 600 ppm

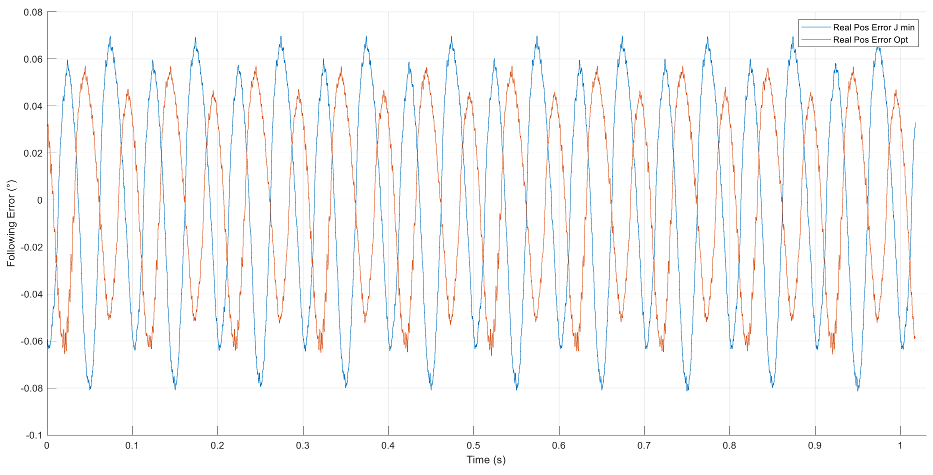


Figure 9-7 Following error with parameters from Jmin (blue) and from optimization (red)

- Cam 4, 600 ppm

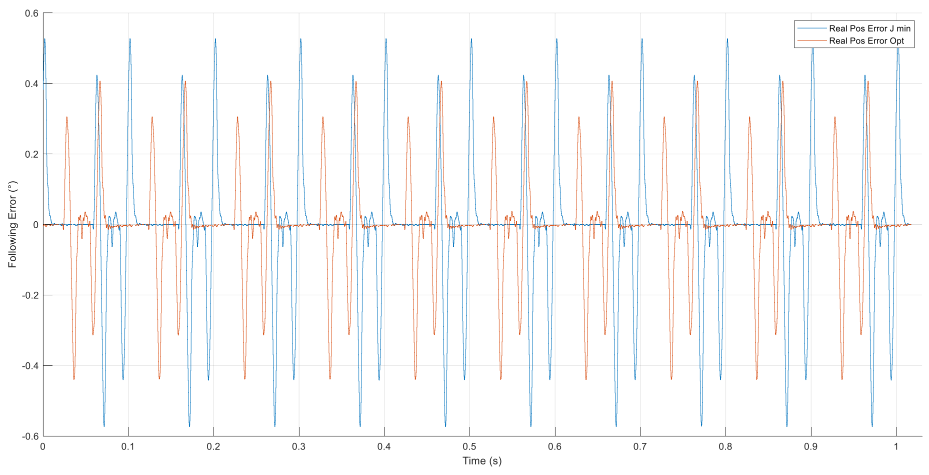


Figure 9-8 Following error with parameters from Jmin (blue) and from optimization (red)

- Cam 6, 600 ppm

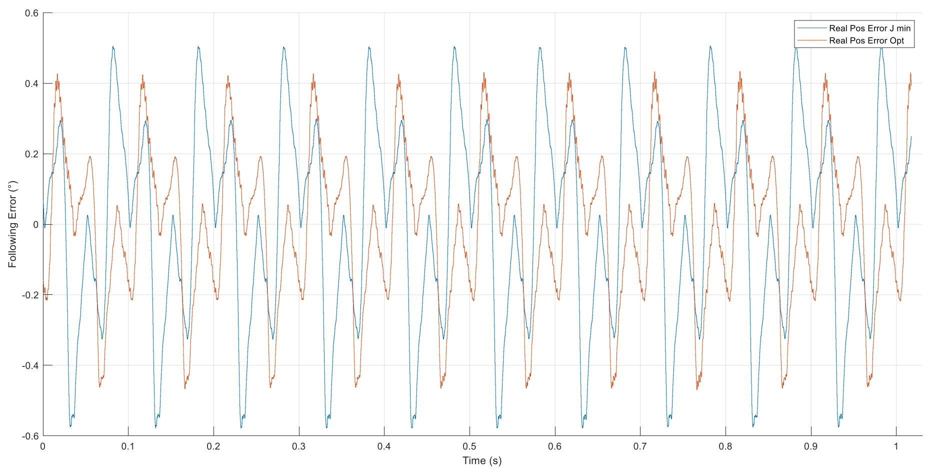


Figure 9-9 Following error with parameters from Jmin (blue) and from optimization (red)

In conclusion, at the end of this optimization we achieved a following error that is, on average, 30% lower than the one with the initial parameters tuned by hand. So far, we let Matlab perform the tuning based on our requirements of bandwidths and phase margin, but it would be better to write explicitly the algorithm ourselves in order to reach better understanding and transparency of the whole process.

9.4 Implementation of the tuning algorithm with inversion formulae

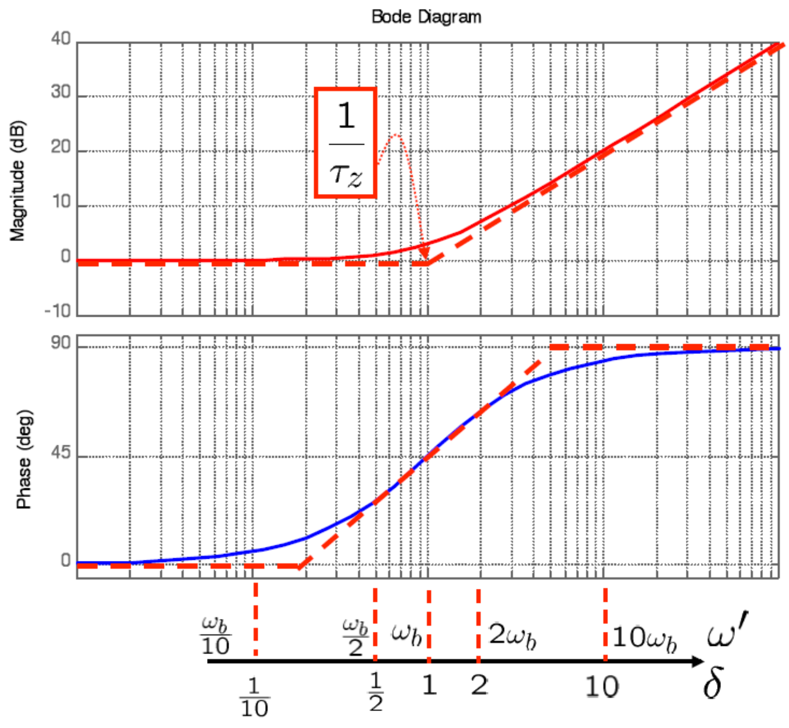
The tuning algorithm is based on the well-known inversion formulae design method. Generally speaking, let's call ***G(s)*** the system to be controlled and write the PI compensator in the following form:

$$R(s) = \mu * \frac{1 + \tau_z s}{s}$$

We want to adjust the action of the zero to impose a certain phase margin and a certain crossover frequency for the extended system

$$G_e(s) = G(s) * \frac{1}{s}$$

Let us call $\omega_b = \frac{1}{\tau_z}$, all frequencies can be written as the product of ω_b and a factor $\delta : \omega' = \omega_b * \delta$. The effect of the zero in the frequency domain is characterized by its Bode diagram:



$$R_z(j\omega') = 1 + \frac{1}{\omega_b} * j\omega' =$$
$$= 1 + j\delta$$

At a given frequency ω', the amplification is $|R_z(j\omega')| = \sqrt{1+\delta^2}$ and the positive phase displacement is $argR_z(j\omega') = \tan^{-1}\delta$.

Our goal is to select the proper time constant τ_z and the gain μ, such that the overall open-loop transfer function has the desired crossover frequency ω_c^* and the desired phase margin P_m^*.

1. The necessary phase lead $\varphi^* > 0$ to achieve P_m^* is
$$\varphi^* = P_m^* - (180° + argG_e(j\omega_c^*))$$
2. From φ^* we can compute the corresponding value of δ:
$\delta^* = \tan(\varphi^*)$ and we choose $\tau_z = \frac{\delta^*}{\omega_c^*}$, so that we have the phase lead φ^* at the frequency ω_c^*.

The amplification introduced by the zero for $\delta = \delta^*$ is $A^* = \sqrt{1+{\delta^*}^2}$, then the crossover frequency of the open-loop transfer function is ω_c^* only if

$$\mu * A^* * |G_e(j\omega_c^*)| = 1$$

So, the static gain should be $\mu = \frac{1}{A^* * |G_e(j\omega_c^*)|}$.

As we can see from the Bode diagram of the zero, the maximum positive phase displacement that it can offer is 90°, therefore it may be impossible to satisfy the desired phase margin if it is quite high. The solution we adopted is to decrease cyclically by 1° the phase margin until the necessary phase lead becomes feasible, i.e. lower than 90°.

```
while Phase_Lead > 89
    Phase_Margin = Phase_Margin-1;
    Phase_Lead = Phase_Margin-(180+Sys_Phase);
end
```

In our specific case, the function $G(s)$ is the product of the low-pass filter representing the current closed-loop and the mechanical inertia (its minimum or maximum value):

$$G(s) = \frac{1}{\tau_c s + 1} * \frac{1}{Js}$$

where τ_c was $\frac{1}{2662}$ [s], since the bandwidth of the current loop is 2662 [rad/s]. Thanks to this algorithm we computed the proportional and integral gain of the speed PI compensator:

$$R(s) = \mu * \frac{1 + \tau_z s}{s} = \mu * \tau_z + \frac{\mu}{s}$$

then $P = \mu * \tau_z \, , I = \mu$.

The proportional gain of the position regulator can be found imposing the desired crossover frequency for the position loop:

$$P_{pos} * G_{pos}\left(j\omega_{c_{pos}}^*\right) = 1 \; \rightarrow \; P_{pos} = 1/G_{pos}\left(j\omega_{c_{pos}}^*\right)$$

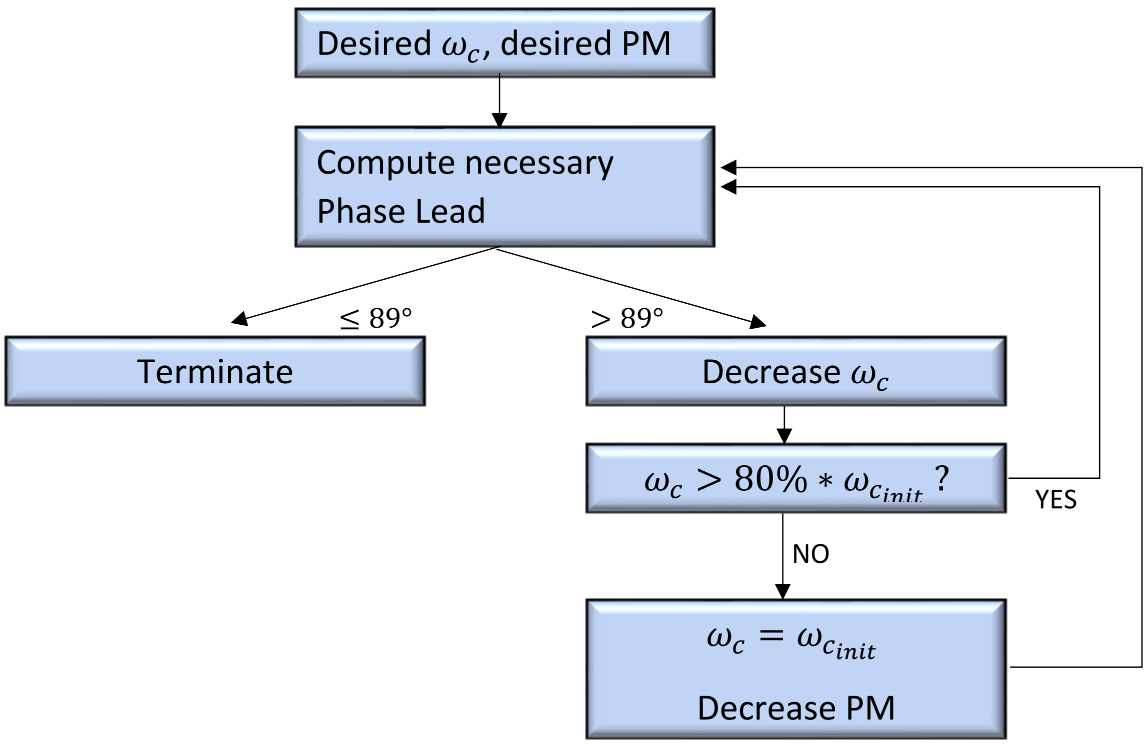
$G_{pos}(s)$ is the product of the speed closed-loop transfer function and the integrator that integrates the velocity to obtain the position.

In comparison to the tuning performed by Matlab, if the requested phase margin is feasible, then the two solutions produce the same parameters for the controller;

otherwise, the way Matlab finds a feasible phase lead is probably different from our choice, consequently the results are a little bit different.

Here is an example: we tried to ask a phase margin of 75°, which was not feasible, the results of the Matlab algorithm were cp_P = 431.7, cv_P = 2.38, cv_I = 92.5 and a phase margin of the speed open-loop transfer function equal to 49.2°; the results of our algorithm were cp_P = 430.7, cv_P = 2.39, cv_I = 110.4 and a phase margin equal to 49°.

We considered, also, another way to deal with a not feasible problem, letting the user choose the preferred one. The new method tries to find a good compromise between speed crossover frequency and phase margin, while the previous one fully penalizes the phase margin, thus the robustness, in favor of the desired bandwidth.
The procedure of this approach is explained through the following diagram:


Desired ω_c, desired PM
Compute necessary Phase Lead
$\leq 89°$
$> 89°$
Terminate
Decrease ω_c
$\omega_c > 80\% * \omega_{c_{init}}$?
YES
NO
$\omega_c = \omega_{c_{init}}$
Decrease PM

In summary, given a certain phase margin the algorithm decreases the crossover frequency until the problem becomes feasible or it is too lower than the initial desired crossover frequency, in that case the phase margin is decreased and the range of allowed frequencies is explored again. The degrees of freedom are the minimum accepted percentage of the desired crossover frequency and how much the frequency and the phase margin are decreased at each step, we decided for 80% for the first and 1 for the second (for both the variables).

All of this is implemented within a function that requires as inputs the parameters of the system: inertia, motor phase resistance and quadrature inductance; the parameters of the current PI regulator; the requirements for the crossover frequency and phase margin of the velocity loop, the desired crossover frequency of the position loop, the solving mode in case of unfeasible problem. The function returns the three controller parameters, the actual phase margin and the actual speed crossover frequency, so that it is possible to see if the requested ones were acceptable or not.

```
function
[vel_P,vel_I,pos_P,Resulting_Phase_Margin,Resulting_Speed_Crossover_Freq
uency] =
TuningFunction(Inertia,Factor_Speed_Crossover_Frequency,Phase_Margin,Fa
ctor_Position_Crossover_Frequency,Resistance,Inductance,P_Current,I_Curre
nt,Mode)
%
% Inertia : inertia of the mechanical system in kgm^2;
%
% Factor_Speed_Crossover_Frequency : value, greater than 1, that divides the
%                             current loop bandwidth to determine the crossover
%                             frequency of the speed open-loop tf;
%
% Phase_Margin : desired value of the phase margin, in degrees, of the speed
open-loop
%              tf. Such value may not be feasible, the function
%              guarantees the highest feasible phase margin and can be read
through
%              the last output of the function;
%
% Factor_Position_Crossover_Frequency : value, greater than 1, that divides
the
%                               speed crossover frequency to determine the
%                                          crossover
%                               frequency of the position open-loop tf;
%
% Resistance : motor phase resistance in Ohm;
%
% Inductance : motor quadrature inductance in H;
%
```

```
% P_Current : proportional gain of the current regulator;
%
% I_Current : integral gain of the current regulator;
%
% Mode : 'reference tracking' keeps the desired speed crossover frequency
fixed
%        and reduces the phase margin,
%       'balanced' finds a good compromise between crossover frequency and
%        phase margin

%% Electrical part
Electric_Plant = tf(1/Resistance,[Inductance/Resistance 1]);

Current_Controller = tf([P_Current I_Current],[1 0]);

Current_Closed_Loop = Electric_Plant*Current_Controller/(1 +
Electric_Plant*Current_Controller);

Fast_Pole=min(pole(Current_Closed_Loop));

FirstOrder_LPf = tf(-Fast_Pole,[1 -Fast_Pole]);

%% Mechanical part
Mechanical_Plant = tf(1,[Inertia 0]);

G = Mechanical_Plant*FirstOrder_LPf;                 % Controlled System

%% Velocity Loop tuning
W_c = -Fast_Pole/Factor_Speed_Crossover_Frequency;    % Desired
Crossover frequency for
speed loop [rad/s]

G_e = G*tf(1,[1 0]);                              % Extended System

Sys_Phase = rad2deg(angle(evalfr(G_e,1i*W_c)));       % Phase [°] of the
extended system
                        at frequency W_c

if Sys_Phase > 0
    Sys_Phase = -360+Sys_Phase;
end

Phase_Lead = -180+Phase_Margin-Sys_Phase;            % Necessary phase
lead for the                                                desired phase
margin

if strcmp(Mode,'reference tracking')
    while Phase_Lead > 89
          Phase_Margin = Phase_Margin-1;
          Phase_Lead = -180+Phase_Margin-Sys_Phase;
```

```
   end
else
    W_c_init = W_c;

    while Phase_Lead > 89
      W_c = W_c - 1;

      if W_c < 0.8*W_c_init
         Phase_Margin = Phase_Margin-1;
         W_c = W_c_init;
      end

      Sys_Phase = rad2deg(angle(evalfr(G_e,1i*W_c)));      % Phase [°] of the
                                                           extended system at
frequency W_c

      if Sys_Phase > 0
         Sys_Phase = -360+Sys_Phase;
      end

      Phase_Lead = -180+Phase_Margin-Sys_Phase;            % Necessary
phase lead for                                                      the
desired phase margin
    end
end

Resulting_Speed_Crossover_Frequency = W_c;

Resulting_Phase_Margin = Phase_Margin;

delta = tand(Phase_Lead);

Zero_time_const = delta/W_c;                  % Zero placement such that it
                                               guarantees the necessary phase
lead                                                at the desired crossover
frequency

A = sqrt(1+delta^2);                        % Amplification introduced by the
                                                  zero for delta

Gain = 1/(abs(evalfr(G_e,1i*W_c))*A);           % Necessary attenuation to
have                                                   the desired crossover
frequency

vel_I = Gain;
vel_P = Zero_time_const*Gain;

Speed_Controller=tf([vel_P vel_I],[1 0]);

Speed_Closed_Loop = Speed_Controller*G/(1+Speed_Controller*G);
```

```
%% Position Loop tuning

W_c_pos = W_c/Factor_Position_Crossover_Frequency;    % Desired
crossover frequency for
position loop [rad/s]

G2 = Speed_Closed_Loop*tf(1,[1 0]);

Sys_Gain = abs(evalfr(G2,1i*W_c_pos));              % Gain of the system at
desired                                                         crossover
frequency

pos_P = 1/Sys_Gain;                             % Necessary gain to cross at
                                                W_c_pos
end
```

10 Conclusion

The main purpose of the project was to develop a methodology to optimize the tuning of the linear controllers (PI regulators) present in current industrial drives in order to save time and get better performances with respect to the common manual tuning based on empirical rules.

In particular, we explored a way to set the controller parameters when a 1 DOF nonlinear mechanism is attached to the motor. The mechanism used as benchmark is adopted in automatic machines for packaging of cigarettes and resembles a four-bar linkage.

The optimization method is based on a Simscape model of the mechanical plant and a Simulink model of the drive, which includes the discrete-time regulators for position, velocity and current loops and the electric d-q axis model of the motor.

First of all, we validated the model of the drive by comparing the simulation data with the data obtained experimentally on the real system, we tried three different configurations of the drive model: position and velocity loops only; the current loop, too; all loops and the PWM modulation with a simplified model of the inverter. The difference among them in terms of simulation error is almost negligible when the current-loop bandwidth is significantly higher than the bandwidth of the speed loop; however, later we observed that, when the parameters of the speed PI regulator push the bandwidth too high, the model with current loop shows high-frequency speed ripple which are absent in the model without it, then the current loop is needed when running the optimization algorithm for tuning.

We ran one first global optimization for the parameters of the position and speed regulators assuming as starting point the set of values found manually, which were good, and imposing quite strict ranges for the search. The function cost considered the maximum of the following error and the frequency content of it. The resulting parameters led to a significant improvement, reducing the maximum following error for the different cams we tried by 20% on average.

Finding a good starting point and reasonable ranges is fundamental for the success of the algorithm in an acceptable time, then we adopted a theoretical method to provide them. We found the minimum and maximum values of the inertia of the mechanism and tuned the controller linearizing the plant with these values, specifying the desired phase margin and crossover frequencies and using the well-known inversion formulae. The parameters obtained with the minimum inertia produced a maximum error 15% lower, on average, with respect to the one with the parameters tuned by hand. This is a considerably good result because just running a piece of code of the tuning function we have not only minimized the time spent but also achieved a very small following error.

The parameters obtained with the maximum inertia produced velocity ripple, thus were not usable; nonetheless, we used those values and those for the minimum inertia as bounds for the ranges to be searched by the optimization algorithm, the latter were also held as starting point. The parameters resulting from the optimization led to a maximum following error lower by 25% than that for manually tuned parameters, on average. This proves the validity of our approach, that merges theoretical knowledge and considerations on the controlled plant with a global optimization search.

Bibliography

[1] Massimiliano Semati, “Realizzazione e validazione del modello Matlab/ Simulink di un drive Bosch-Rexroth”, 2018