UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Sensorless-Field-Oriented-Control for PMSM Air Ventilating Motor

Autor: Diego Colás Arnaiz

Trabajo realizado en la Universidad: FH JOANNEUM University of Applied Sciences

País: Austria

Profesor Tutor: Raúl Estrada Vázquez

FIRMAS

Secretario de la escuela: D. Antonio José de Vicente Rodríguez

Coordinador internacional: D. Juan José Blanco

Tutor Académico: D. Juan Carlos García García

CALIFICACIÓN:

FECHA:

ÍNDICE GENERAL

I. RESUMEN

Este proyecto, como su nombre indica tiene como objetivo el control de un motor de imanes permanentes (PMSM) que actuará integrado en un ventilador de un coche convencional. Para hacer frente a este problema, el algoritmo Field Oriented Control (FOC) se utiliza en primer lugar en combinación con técnicas de control en tiempo discreto y después con control adaptativo predictivo.

En la primera parte en la que se estudia el control del motor mediante FOC con controladores PID, el objetivo perseguido es un control robusto que pudiese accionar el motor de una manera fiable en presencia de cualquier perturbación y/o cambio en la referencia. De acuerdo con la parte en la que se trató el control Adaptativo Predictivo, acercarse o incluso mejorar el comportamiento del motor controlado con PID discretos es el objetivo principal. Por desgracia, la prueba de este último en la plataforma real no fue posible, debido a razones de diferencias entre modelo calculado del motor y motor real que se explicarán con más detalle en la tesis.

Para el desarrollo de todo el trabajo, todos los materiales necesarios fueron proporcionados por el FH JOANNEUM. Estos van desde un sensor de corriente de efecto Hall simple a una herramienta sofisticada llamada MicroAutoBox actuando como la CPU del sistema o el "cerebro" de la aplicación.

Resumen en inglés

This project as its name indicates aims at to control a Permanent Magnet Synchronous Machine (PMSM) integrated in the fan of the air ventilation system in a conventional car. To tackle this problem the Field Oriented Control (FOC) algorithm is used in combination with control techniques in discrete state, at first, and afterwards with predictive adaptive control.

In the first part where FOC with discrete Proportional Integral Derivative (PID) controllers are studied, the objective was to achieve robust control capable of driving the motor in a reliable way in presence of any disturbance. Related to the part where Adaptive Predictive control is tried, approaching or even enhancing the behavior of the motor controlled with discrete PID's has been the main objective. Unfortunately the final test with the real system was not possible due to reasons that will be explained in more detail within the thesis.

For the development of the whole work, all necessary system components were provided by the FH JOANNEUM. These go from a simple Hall Effect current sensor to a sophisticated Rapid Control Prototyping Unit called MicroAutoBox acting as the CPU or "brain" of the application.

II. RESUMEN AMPLIADO

A continuación les presento un resumen ampliado en el que describiré a grandes rasgos todos los pasos desarrollados para la realización de este proyecto, así como las facilidades en el centro donde se ha realizado y por parte de sus profesores.

El proyecto trata del control de un motor de imanes permanentes (PMSM) para realizar la función de un ventilador dentro de un automóvil. Esta función hoy en día se realiza utilizando un motor de corriente continua, sin embargo se quería probar la posibilidad de usar un PMSM debido a sus ventajas en cuanto a consumo y peso.

El drawback de los motores de corriente alterna es precisamente el control, ya que este es mucho más complicado que el de un motor simple de corriente continua. Para esta tarea se ha utilizado el conocido algoritmo FOC (Field Oriented Control) y así poder controlar el motor de corriente alterna de forma fiable y precisa. Con la programación en bloques de Simulink del algoritmo FOC (Field Oriented Control), obtuve gran ayuda por parte de de mi tutor del proyecto, Raúl Estrada Vázquez, y de otro profesor, Alfred Steinhuber, que ya había indagado anteriormente en el mismo tema. Ese hecho de no empezar este trabajo desde cero y las ayudas recibidas fue bastante importante para la realización del proyecto final.

Una vez tenido el algoritmo listo para funcionar y controlar el motor, el siguiente paso fue implementar un control en cascada de corrientes y velocidad. Primero era necesario el control de las dos corrientes resultantes del FOC, iq y id, estas corrientes son derivadas de las corrientes de fase del motor pero en un diferente sistema de coordenadas. Para su control se usó un PID como controlador discreto, aunque realmente carece de parte derivativa, utilizando la técnica de control Internal Model Control (IMC), que sostiene que a través de una buena identificación de la planta de nuestro sistema se puede alcanzar un control ideal sin errores. Esto, como ya es predecible, en la práctica un control ideal es completamente inviable por lo que los valores dados por esta técnica no fueron los definitivos y empíricamente se hallaron los valores de la constantes proporcional e integral de ambos controladores.

Esta prueba del sistema y posterior cálculo de los valores adecuados, se realizó en una plataforma real con un motor PMSM real. La herramienta utilizada para ejecutar el programa en tiempo real es el MicroAutoBox y el software para la comunicación en tiempo real con ella, el ControlDesk. Estas caras y poderosas herramientas fueron proporcionadas por la FH JOANNEUM para la realización del trabajo así como varias placas de comunicaciones para la conducción de las señales I/O y un inversor. Este tema del material se trata más a fondo en la memoria del proyecto.

Al terminar esta parte del proyecto y observar los resultados obtenidos en el motor, se pudieron observar claramente las no linealidades del motor PMSM lo que hace difícil su control. Para intentar contrarrestar este problema se utilizó una nueva técnica de control que es capaz de adaptar el modelo previamente obtenido del motor, por tanto las no linealidades no se harán tan notables.

Hablamos de control Adaptativo Predictivo, una técnica que no está en uso hoy en día en las industrias debido a su no suficiente investigación pero que puede proporcionar unos resultados bastante más interesantes que un control con un PID convencional.

Para su desarrollo, se hizo uso del mismo modelo en Simulink que el usado para la aplicación con el PID, cambiando únicamente el bloque que controla los parámetros del

motor. El software para el control adaptativo es implementado en un bloque Matlab function, que permite la compilación y conexión del código para su transferencia al MicroAutoBox. Aunque posteriormente la prueba en la plataforma real no será exitosa, ya que el modelo del motor utilizado en el código no es suficientemente preciso y la respuesta final tiene grandes fluctuaciones debido a estas diferencias y a que el proceso de adaptación no es suficientemente rápido. Por lo tanto los resultados de este apartado únicamente se obtendrán en simulación y no con la plataforma real.

Una vez finalizado el modelo en Matlab, se prueba su ejecución para comprobar que los valores del código son correctos. Los valores en esta tipo de control que se pueden cambiar según el tipo de respuesta requerida son el Periodo de Control, que hace referencia al número de muestras que la aplicación deja pasar entre ejecuciones del control; y el Horizonte de Control, que es el número de ejecuciones del control que se quieren para estabilizar la respuesta final con respecto a la referencia.

Este proceso de definición de los valores para los controladores se realizó de forma análoga al modelo con los PID, primeramente se fijan valores para el control de corriente para luego, una vez alcanzado un buen y fiable control, hacer lo mismo con el control de velocidad. Los resultados de esta última prueba realizada en simulación fueron bastante satisfactorios, dejando a un lado las fluctuaciones en la respuesta al principio de activar el control.

En conclusión, tras la realización de todo este trabajo fuimos capaces de obtener un control robusto del motor resistente a perturbaciones y a cambios en la referencia, contrarrestando las no linealidades del sistema real, como se puede ver más detenidamente a continuación, en la memoria completa del proyecto.

III. MEMORIA

Trabajo Fin de Grado

Título:

Sensorless-Field-Oriented-Control for PMSM Air Ventilating Motor

Idioma: Inglés

1. ACKNOWLEDGEMENTS

I would like to write this part of my thesis in order to thank one person in particular, Raúl Estrada Vázquez who was my project supervisor in the FH JOANNEUM in Austria. Remarking the difficulty of making the bachelor thesis abroad, something that from my point of view has much importance in someone's career; Raul was in every moment supportive and helpful, surely without his advice and knowledge about the field, I would not have been able to finish successfully this work.

I also want to mention Alfred another professor from Austria for the same reason, with whom I also have a good relationship.

Personally, I feel very pleased to have had the opportunity of working in this project for the FH JOANNEUM; the personal treatment I got from them could not be better. I am also proud of all the knowledge achieved in my home university, the University of Alcalá de Henares, which made possible the realisation of this work from my part.

To conclude I would like to dedicate all the work done to my family and girlfriend María, since with their support from Spain made it possible.

2. CONTENT

3. INTRODUCTION

The Sensorless-Field-Oriented-Control for PMSM air ventilating Motor [2] was implemented and tested in Matlab/Simulink® environment, using Mathworks property programing toolboxes. In this sense, the complete FOC strategy was implemented by exclusively using Simulink blocks in order to allow code generation for an embedded system implementation.

The whole Field Oriented Control (FOC) strategy consists of the following parts [2]:

- FOC
- Position Estimation
- Signal conversions, Clarke/Park

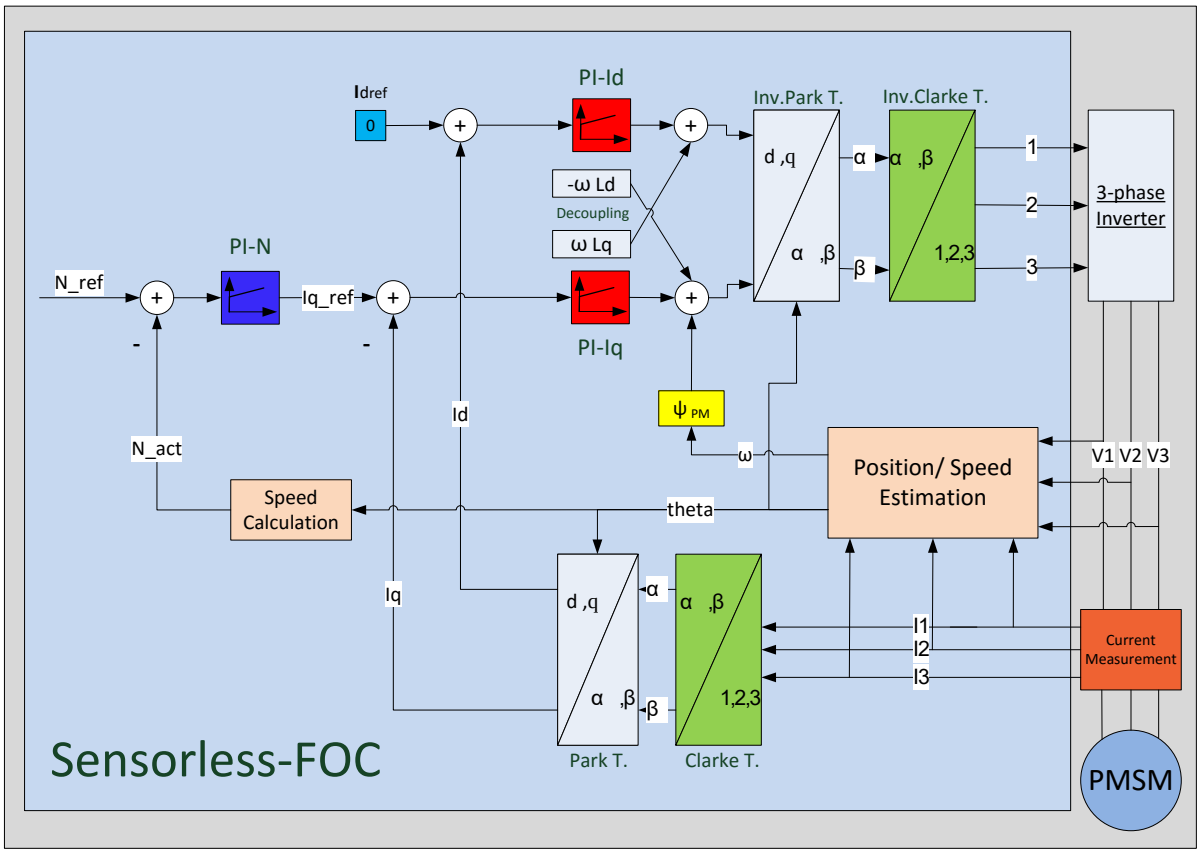


Figure 1: Schematic of the Sensorless FOC

The above shown figure depicts the setup for the Sensorless-Field-Oriented-Control (within blue frame). With Field Oriented Control of the PMSM [2] the stator current usually will be controlled to obtain a right angle between stator current and pole flux, Due to the permanent magnets at the rotor, the reference value for the field exciting current Id is set to zero. As a result the current does not contribute to magnetization, but only to torque production. Basically the Controller structure is a two layer control structure, where the outer layer

controls the actual speed and the inner layer controls the stator current, which is directly proportional to the torque. Critical in this point is the correct estimation of the rotor position, as this information is the backbone for a correct working FOC.

The purpose of such a system [2] is to work as a fan in a car, making the approach more challenging by using PMSM as the driving machine. Normally in all cars this function is carried out by simple DC motors since the control of these is much easier. However in this approach it is tried to develop the same principle but with an AC motor having as main advantage the power-weight ratio and the low power consumption when using FOC technique, so that the efficiency within the car can be increased.

The most important drawback of AC motors and precisely in this case PMSM [2] is that their control techniques are far more complicated than those to control DC motors. Nevertheless the purpose of this project is to manage to achieve an acceptable control of the PMSM.

4. OBJECTIVES

The main goal of the project is to challenge the difficulties of controlling an AC motor and find a way to control it precisely enough.

This way will be through Field Oriented Control (FOC), which is going to be the heart of the program, and, as a first approach, the control algorithm will be carried out by PID's controllers. The results will be proved on hardware with a real motor in real time using a tool, or should I say a "brain", called **MicroAutoBox** and its related software **ControlDesk** which allows the user to interact with it.

Once the PID control is developed, the idea is to move to Adaptive Predictive Control. This sophisticated control technique was developed in order to know beforehand or predict what the system is probably going to do or how it is going to react, thus a model of the controlled system is needed (the PMSM in this case) or a good procedure of online adaptation if the model is not accurate enough.

The implementation of this software in real time will also be proved with the MicroAutoBox if its building process turns successful.

5. SENSORLESS FOC FOR PMSM

a. Theory of Permanent Magnet Synchronous Motors (PMSM)

Permanent-magnet Synchronous Motors [3] came first on the market in the early 1990s. Their main application fields are dynamic Servodrives in manufacturing processes within the industry. Due to the achievable high power density and low weight they are also becoming a choice for the drive component for hybrid and electric vehicles.

The main advantages are:

1. Highest power density and thus very compact and relatively light-weighted
2. High efficiency as a result of the passive field excitation
3. High torque and speed dynamics
4. Low noise emissions

The magnetic field is supplied by extremely strong magnets built in the rotor and made of:

- Samarium-Cobalt (Sm Co5)
- Neodymium-Iron-Boron (Nd2 Fe14 B)

Figure 2: Permanent Magnet Synchronous Machine (figure from directindustry.com)

Operating as a motor or as a generator

The torque and thus the transmitted power [3] are a result of the rotor angle, which is the angle between the rotor position at no load, which correspond to the direction or position of the rotary magnetic field of the stator, and the rotor position at a certain load. With a leading rotor the machine is operated as a generator, whereas it is operated as a motor when the rotor is lagging.

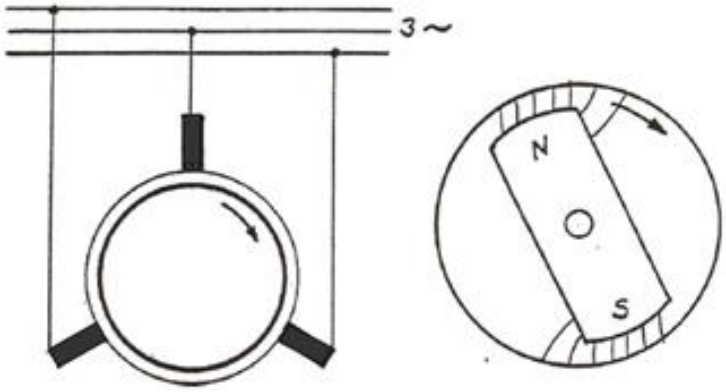


Figure 3: Sketch of the PMSM stator and rotor

The rotor can be seen as a freely rotating magnet, which is influenced by the stator rotary field and which tries to align with this mentioned field. That would be the stable position. When the magnetic field starts to turn, the rotor will follow.

Attaching an external braking torque at the rotor shaft, the flux of the stator magnetic field will increase in magnitude as a result of the high torque and the motor will maintain its synchronous speed.

Varying torque only changes the rotor angle but not the speed, just if the motor has the power enough to deal with the torque.

Only when the maximum torque limit of this magnetic spring is finally exceeded, the interconnection of the fields will break off. This process is called out-of-step operation.

Inverter-fed PMSM

PMSMs are fed by PWM inverters [3], which are providing a symmetric three-phase voltage system (the same is made to induction motor applications).

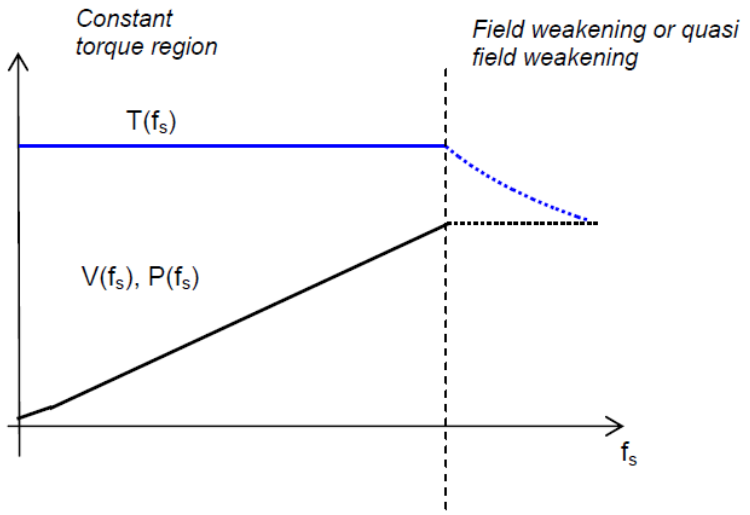


Figure 4: Voltage / frequency characteristics

In a PMSM it is very difficult to influence the magnitude of flux via the stator current (because of high effective air-gap-length) that is why the control of this type of motors is quite more difficult compared to dc motors. Only in case of a rotor with buried magnets the possibility of distorting the field exists. This allows the rotor field to be influenced by external signals.

PMSM-Motor model

In order to test the FOC algorithm by means of simulation [2] it is required to generate a mathematical model of a PMSM. To make the motor model easier the three phase voltages at the input of the motor are transformed to a space vector form, which will be also the same procedure when developing an FOC control. This results into the voltage equation of the PMSM in d-q frame.

$$\overrightarrow{u_s^{dq}} = R_s \overrightarrow{i_s^{dq}} + \frac{d}{dt}\left(\overrightarrow{\psi_s^{dq}}\right) + j\omega_m \overrightarrow{\psi_s^{dq}}$$

$$u_d = R\, i_d + \frac{d\psi_d}{dt} - \omega_e \psi_q$$

$$u_q = R\, i_q + \frac{d\psi_q}{dt} + \omega_e \psi_d$$

R_S…Stator resistance; L_d…Direct inductance; L_q…Quadrature Inductance; ω_e…Electrical speed

The magnetic flux linkage in the rotor reference frame of the PMSM are given with the equations

$$\overrightarrow{\psi_s^{dq}} = L_s \cdot \overrightarrow{i_s^{dq}} + \overrightarrow{\psi_{PM}^{dq}}$$

$$\psi_d = i_d L_d + \psi_{PM}$$

$$\psi_q = i_q L_q$$

ψ_{PM}…Flux of permanent magnet

Combining the previous equations results into the PMSM voltage equation of following form

$$u_d = R\, i_d + L_d \frac{d\psi_d}{dt} - \omega_e \psi_q = R\, i_d + \frac{d(L_d i_d + \psi_{PM})}{dt} - \omega_e L_q i_q$$

$$u_d = R\, i_d + L_d \frac{di_d}{dt} - \omega_e L_q i_q$$

$$u_q = R\, i_q + \frac{d\psi_q}{dt} + \omega_e \psi_{PM} = R\, i_q + L_q \frac{di_q}{dt} + \omega_e L_d i_d + \omega_e\, \psi_{PM}$$

$$u_q = R\, i_q + L_q \frac{di_q}{dt} + \omega_e (L_d i_d + \psi_{PM})$$

The motion of the motor can be related to

$$\frac{d\omega_m}{dt} = \frac{1}{J}(T_e - T_L)$$

T_e… Torque produced by Motor; T_L…Load Torque; ω_m…Mechanical speed; J…Moment of Inertia

The Torque produced by the motor

$$T = \frac{3}{2}\frac{P}{2}\left(\overrightarrow{\psi_s} \wedge \vec{i}_s\right) = \frac{3}{2}\frac{P}{2}\left(\psi_d\, i_q - \psi_q\, i_d\right) = \frac{3}{2}\frac{P}{2}\left[\psi_{PM}\, i_q + i_d\, i_q \left(L_d - L_q\right)\right]$$

P… Poles of Motor;

Assuming

$$L_d = L_q$$

The Torque equation can be written as follows

$$T = \frac{3}{2}\frac{P}{2}\psi_{PM} i_{sq}$$

Electrical speed can be related to

$$\omega_e = \omega_m * P/2$$

In these equations above, which completely describe the behavior of a PMSM, u_s stands for the stator voltage which is in relation with the stator resistance R_s and the stator current i_s as well as the stator flux ψ_s and the electrical angular frequency ω_m. In turn, the flux created by the stator depends on the stator self-inductance L_s the stator current i_s and the flux ψ_{PM} resulting from the permanent magnet.

The motor model is implemented into the Matlab Simulink environment transforming those expressions into blocks in Simulink. Important in this case is the management of the d-q frame within the model, since the model has to handle three phase voltages as an input. The conversion to the d-q frame [10] thereby is achieved by means of transformations (Clark and Park). As an output the model delivers the three phase currents, the produced torque, the

angular position and the speed of the motor. Figure 5 depicts the structure of the implemented motor model. According to the equations the model is split into a current producing part and torque generating part. Each part is briefly explained in the next sections.

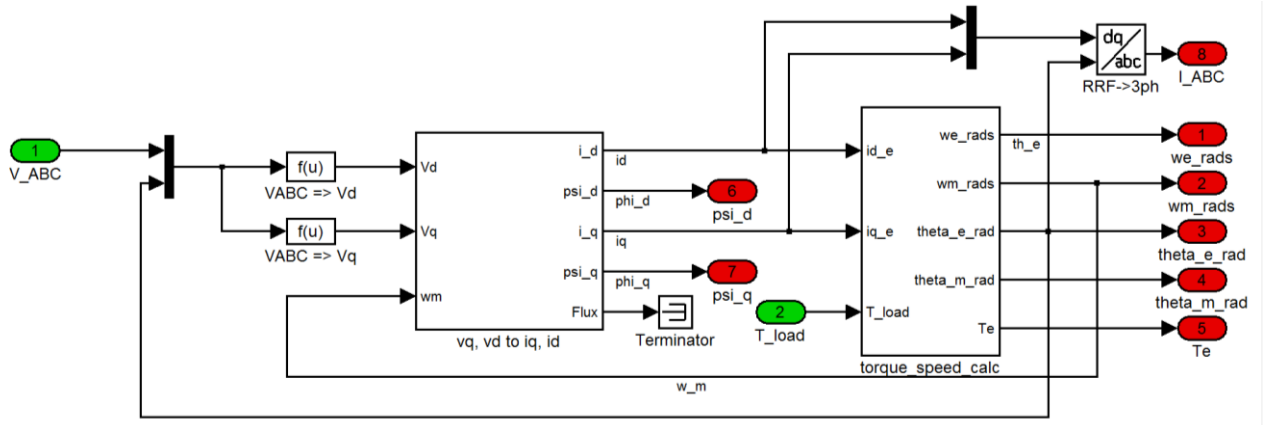


Figure 5: PMSM motor model

Current generating part

In order to achieve the current demanded by the motor [2], the voltage equations have to be modified. This process results into the following formulas depicting the motor current in the dq-Frame

$$i_d = \int \frac{u_d - R_s i_d + \omega_m L_q i_q}{L_d} dt$$

$$i_q = \int \frac{u_q - R_s i_q - \omega_m L_d i_d - \omega_m \psi_{PM}}{L_q} dt$$

Based on this equation the current generating part of the Motor is implemented depicted in Figure 6.

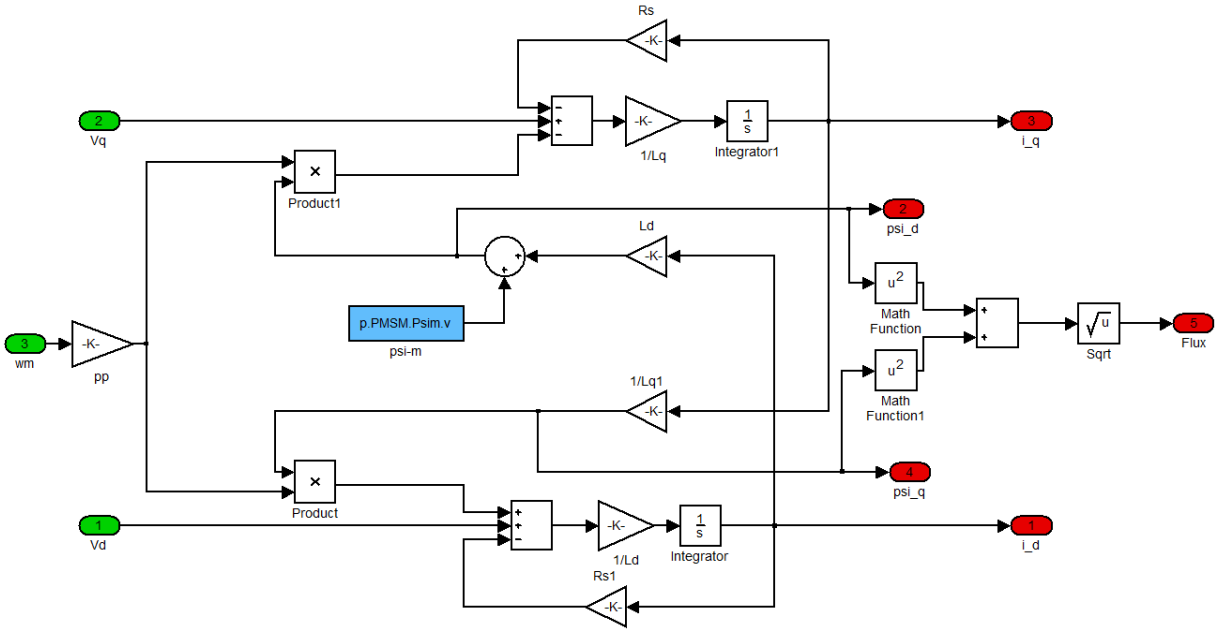


Figure 6: Current generating part of PMSM-Model

Torque generating part

The torque and speed of the Motor are connected by following relation

$$T = \frac{3}{2}\frac{P}{2}\left[\psi_{PM}\, i_{sq} + i_{sd}\, i_{sq}\left(L_d - L_q\right)\right]$$

$$\frac{d\,\omega_m}{dt} = \frac{1}{J}\left(T_e - T_{load}\right)$$

$$\omega_m = \frac{P}{2}\omega_m$$

The implementation of this equation can be found in Figure.

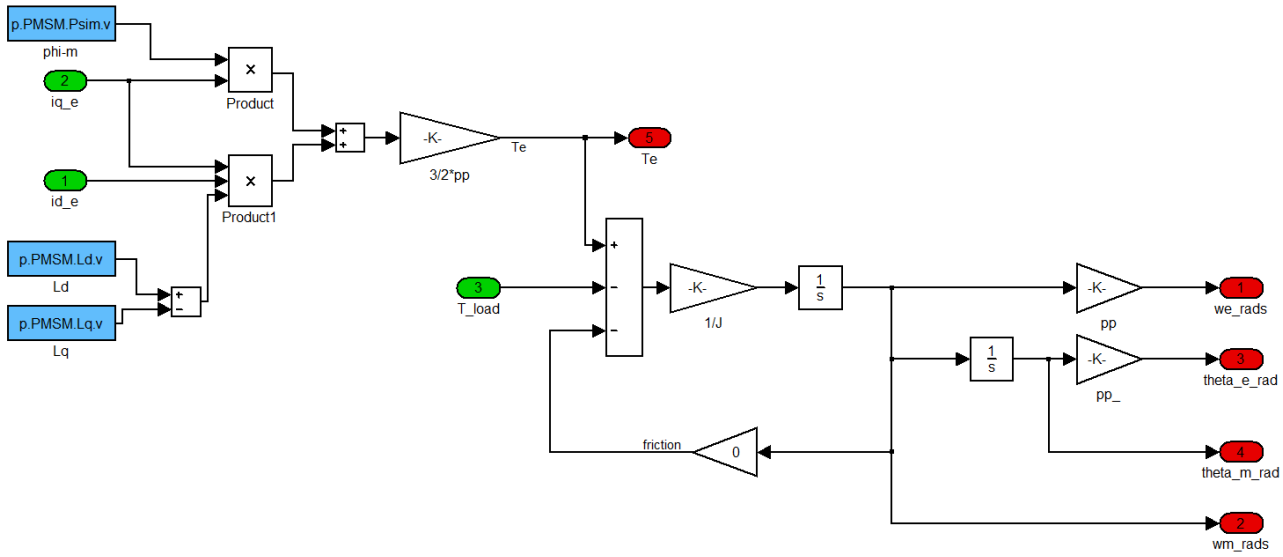


Figure 7: Torque generating part of PMSM-Model

b. How FOC works. Theory of space vector control

FOC of permanent-magnet synchronous motors [5] starts by measuring two of three phase currents, ia and ib. Only two current sensors are needed as third phase current is calculated from ia + ib + ic = 0. The Clarke Transform converts three-phase currents onto a two-axis plot to create variables Iα and Iβ. As viewed from the perspective of the stator, Iα and Iβ are time-varying quadrature-current values. These variables that form the two-axis plot create a plane which is deviated from the rotor flux plane.

The two-axis plot is rotated to align with the rotor flux using transformation angle θ. θ is calculated during the last iteration of the control loop. With this last step id and iq variables are provided from Iα and Iβ. The Park Transform aligns quadrature currents id and iq with the rotating plot in each iteration. Both id and iq remain constant during steady-state conditions, they just change at the moment of every iteration.

Measured Id and iq signals with the reference values for each generate error signals. The id reference controls rotormagnetizing flux while the iq reference governs motor torque output. These error signals are fed normally to proportional-integral (PI) or to proportional-integral-derivative (PID) controllers whose outputs are sent to the motor as voltage vectors Vd and Vq. One note that sould be introduced here is that, by saying “the motor” in the last sentence it is meant the hardware system which includes inverter and motor. These controller outputs are sent to the inverter, but previously modified

Vd and Vq are rotated back into the stationary reference frame using the transformation angle to obtain quadrature voltage values, Vα and Vβ. Vα and Vβ are then mathematically transformed back into three phase-voltages Va, Vb, and Vc, which determine the new PWM duty-cycle that will drive the inverter.

The new coordinate transformation angle is then estimated for the next iteration using Vα, Vβ, Iα, and Iβ as inputs. The new commutation angle guides the FOC algorithm in placing the next voltage vector.

Of course, to determine the required phase voltages it's necessary to know the current position of the rotor relative to the three phase windings. An encoder or resolver could supply that information, but at added cost and complexity. However, you can estimate the rotor position using motor currents and voltages.

Application steps

1. Phase currents are measured, obtaining the values or vectors in a,b,c coordinate system.

2. These vectors are converted to (α, β) coordinate system, which is a system that rotates in the rotor reference frame.

3. Current vector is converted to (d,q) coordinate system. This is a static frame and does not change between iterations.

4. The d-axis component of the stator current vector is used to control the flux in the rotor and the imaginary q-axis component is used to control the motor torque.

5. PI controllers included in the system provide (d,q) coordinate voltage components. A decoupling term is sometimes added to the controller output to improve control performance mitigating big and rapid changes in speed, current and flux.

6. Once the voltage components are obtained, they are transformed from (d,q) coordinate system to (α, β) coordinate system.

9. Voltage components are again transformed from (α, β) , to (a,b,c) coordinate system for signaling to the PWM driven power inverter section.

Significant aspects of Field Oriented Control application:

- Speed or position measurement or some sort of estimation is needed.
- Torque and flux can be changed reasonably fast when changing the references.
- The step response has some overshoot if PI control is used.
- The switching frequency of the transistors is usually constant and set by the modulator.
- The accuracy of the torque depends on the accuracy of the motor parameters used in the control. Big errors due to the rotor temperature for example are often encountered.

- Reasonable processor performance is required; typically the control algorithm has to be calculated at least every millisecond.

c. Estimation of rotor position. Definition of sensorless

To determine the required phase voltages it's necessary to know the current position of the rotor relative to the three phase windings. An encoder or resolver could supply that information, but at added cost and complexity. However, you can estimate the rotor position using motor currents and voltages.

The position estimator [6] starts with a mathematical model to measure motor position indirectly with an observer. Motor position is estimated by assuming the PMSM model is the same as that of a dc motor. The model consists of a series circuit containing the winding resistance, *R*, winding inductance, *L*, and the back-EMF.

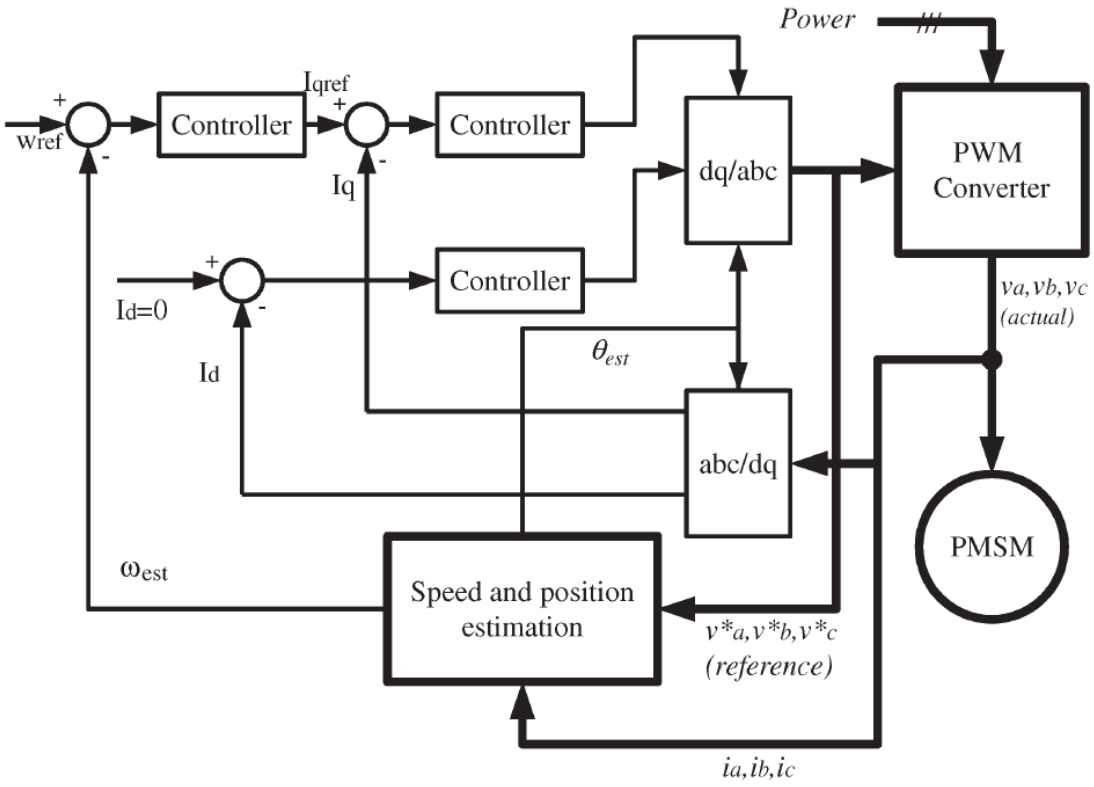


Figure 8: Sensorless algorithm structure. Estimator function

Note that the term sensorless referred here does not mean that no sensor is used; means that no position or velocity sensor is needed. In the hardware there are, however, current and voltage sensors.

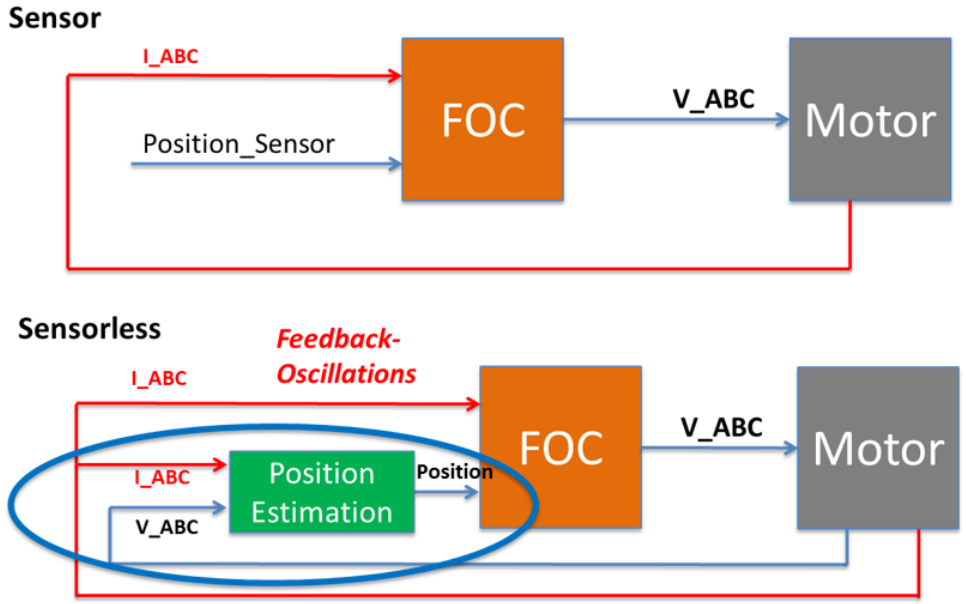


Figure 9: Sensor vs Sensorless FOC

There are several different Estimators known by literature [2]. The algorithms taken into consideration for the application of the air ventilating fan can be dedicated to the category of model based estimators.

Simple Back Electromagnetic Force Estimators (BEMF):

This type of estimator is based on the model of the PMSM [2] and the measured electrical variables in order to determine within each step of the operation the rotor position and speed. The stator flux linkage in stationary frame can be calculated out of the voltage equations in the αβ-frame by solving the derivatives

$$u_\alpha = R\, i_\alpha + \frac{d\psi_\alpha}{dt}$$
$$u_\beta = R\, i_\beta + \frac{d\psi_\beta}{dt}$$

Solving the derivative results in

$$\psi_{s\alpha} = \int (u_{s\alpha} - R \cdot i_\alpha) dt$$
$$\psi_{s\beta} = \int (u_{s\beta} - R \cdot i_\beta) dt$$

The flux of the stator contains the inductance Ls and the flux of the permanent magnet

$$\psi_{PM,\alpha} = \psi_{s\alpha} - L_s \cdot i_\alpha$$
$$\psi_{PM,\beta} = \psi_{s\beta} - L_s \cdot i_\beta$$

Substituting the equation in the above formula results in

$$\psi_{PM,\alpha} = \int (u_{s\alpha} - R \cdot i_\alpha) - L_s \cdot i_\alpha$$
$$\psi_{PM,\beta} = \int (u_{s\beta} - R \cdot i_\beta) - L_s \cdot i_\beta$$

The angle corresponds to

$$\theta_m = \arctan\left(\frac{\psi_{PM,\beta}}{\psi_{PM,\alpha}}\right)$$

Drawbacks of model based estimators (observers):

- At low speed currents and voltages are not providing enough information for achieving a correct or meaningful position
- Steady state errors, as the estimation is dependent on the provided parameters (R_S …Stator resistance; L_d …Direct inductance; L_q …Quadrature Inductance; ψ_{PM}…Flux of permanent magnet)
- Important is the range of inaccuracy of these values compared to the real value.

This type of estimator is a good approximation of the actual position in open loop condition. Therefore the estimation can be used for monitoring the functionality of the position sensor. Or also vice versa, a calibrated sensor may be used to know whether the estimator is precise enough or does not meet a minimum level of accuracy

BEMF estimator with PLL in stationary ref frame (alpha, beta)

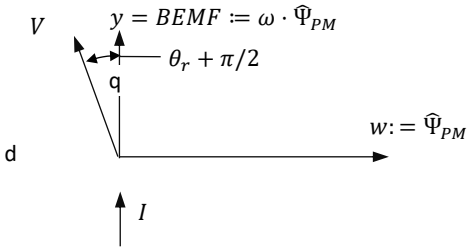
This type of estimator [2] is based on the voltage equation of the PMSM which is converted to the stationary reference frame (αβ) (assumption Ld=Lq=L)

$$\underline{u}^s = R_s\, \underline{i}^s + \frac{d\underline{\psi}^s}{dt} = R_s\, \underline{i}^s + L\frac{d\underline{i}^s}{dt} + j\omega_r \psi_{PM} e^{j\theta_r}$$

Solving this equation to ω_r results in

$$\hat{\omega}_r = \frac{d\theta_r}{dt} = \underbrace{(\underline{u}^s{}_{ref} - \hat{R}\,\underline{i}^s - L\frac{d\underline{i}^s}{dt})}_{\underline{y}} \underbrace{e^{j(\theta_r+\pi/2)}}_{\underline{w}} \frac{1}{\hat{\psi}_{PM}}$$

Being y the BEMF phasor whereas the length of the phasor is directly related to the speed. The unit phasor w leads the estimated rotor position by $\pi/2$.



The goal to pursue in this algorithm is the calculation of the angle correction based on the cross product between the two quantities y x w. If the cross product is zero which means that the estimation is correct (y is aligned with w), the error signal sent to the PI Controller (PLL) is set to zero.

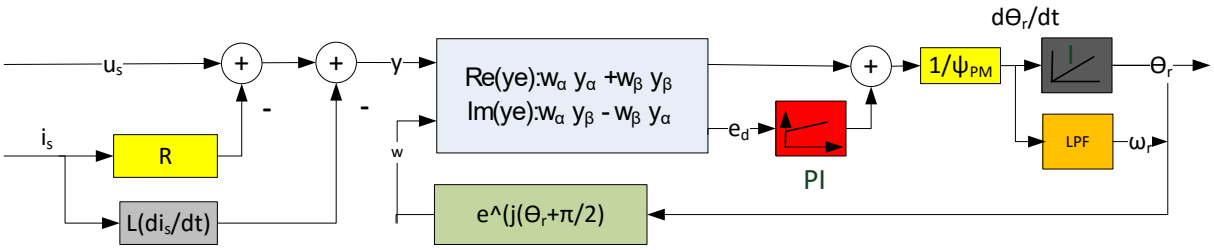


Figure 20: Schematic of the BEMF-PLL Estimator

Consider the Euler formula for the exponent

$$e^{j(\theta_r+\pi/2)} = \cos(\theta_r + \pi/2) + j\sin(\theta_r + \pi/2)$$

This implementation in Matlab Simulink (Figure 10) is based on the equations shown above. According to the problems explained before, the estimator is not capable to provide a position from standstill on, since the y and w variables are aligned, thus the cross product is set to 0. However there is a Block added named Startup function, such function is just imposing a ramp for the starting position [2].

d. Simulink model for the application

The Simulink model of the software wanted to be implemented is depicted in the following figures.

First the model for the real time application is shown, it consist of 5 main blocks or parts:

- Hardware interface IN (green block): where the measured voltage and current signals are read from the I/O ports and processed.
- Position monitoring (grey): position estimator avoiding the position sensor.
- FOC_1ms (blue): where the FOC strategy and control strategies are programmed.
- Duty generator (grey): adapt the results of the control to correct duty cycles to be introduced in the inverter.
- Hardware interface OUT (red): where the appropriate signals are sent to the I/O ports in order to be afterwards the signals which drive the inverter.

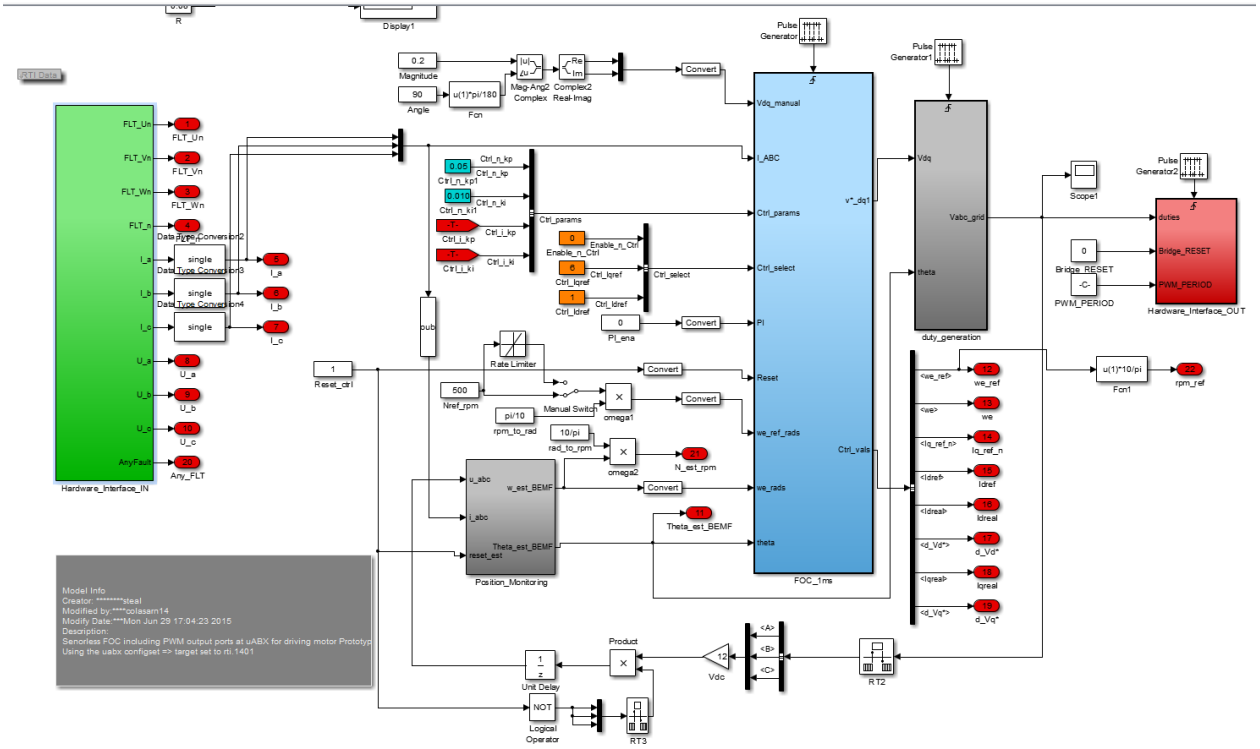


Figure 31: Simulink model for real time application

As follows, the model just for simulation can be also seen. This model is just created to try the software in the Simulink environment before transferring and running it in the real motor. It has little difference, speaking of control or FOC algorithm, but still the model is quite different.

Here the different blocks and its functions are presented:

- Sensor behavior (dark green): represents through transference functions the effect of the current and voltage sensors, therefore it is introduced in the simulation model.
- Scale code to sim (orange): values escalation
- PMW reference 200 micro sec (black): where the pulses to drive the inverter are calculated from the reference voltage value of the vector and the angle.

- PMSM motor model (orange): mathematical model of the PMSM motor.

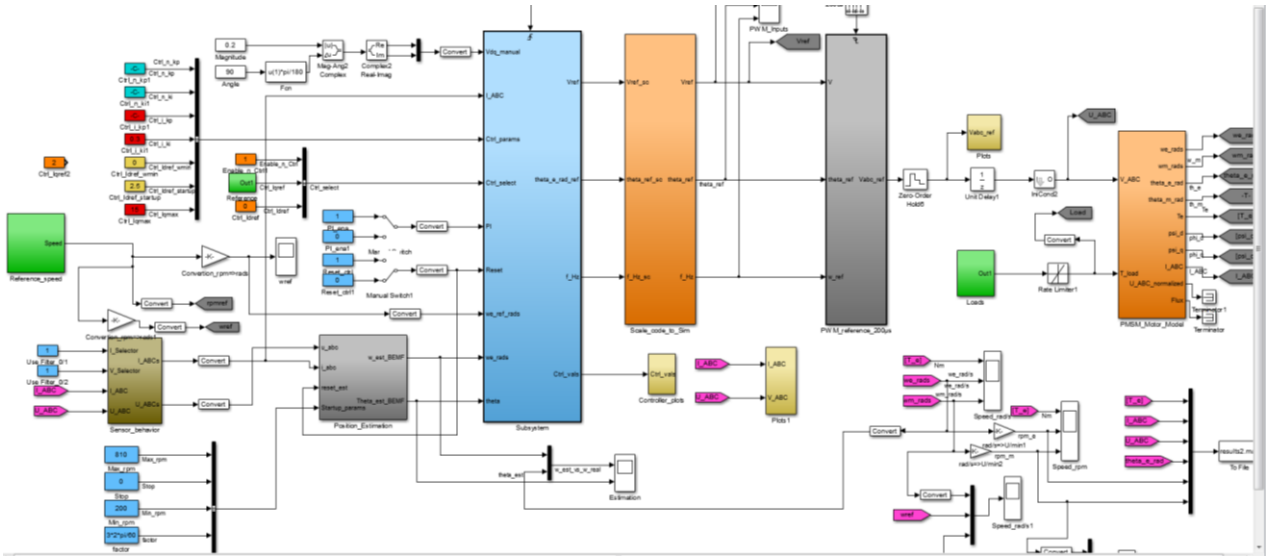


Figure 42: Simulink model for simulation

6. CONTROL OF A PMSM WITH DISCRETE BASED CONTROL TECHNIQUES

a. IMC

For the current and speed control of the PMSM the Internal Model Control [7] (IMC) technique is used.

This control philosophy relies on the Internal Model Principle, which states that control can be achieved only if the control system encapsulates, either implicitly or explicitly, some representation of the process to be controlled. In particular the control scheme has been developed in simulation based on an exact model of the process; therefore "perfect" control is theoretically possible to be achieved.

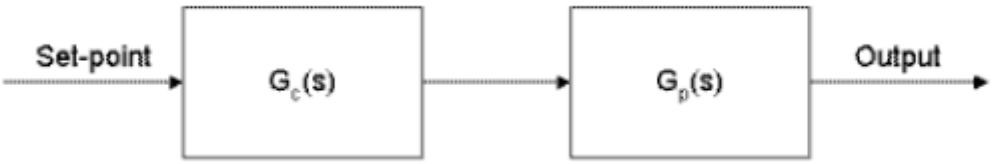


Figure 13: Example of IMC

A controller, Gc(s), is used to control the process, Gp(s). Suppose $G_1p(s)$ is a model of Gp(s). Setting Gc(s) to be the inverse of the model of the process.

$Gc(s) = Gp_1(s)^{-1}$,

And if $Gp(s) = Gp_1(s)$, the model is an exact representation of the process.

Then it is clear that the output will always be equal to the set-point. Notice that this ideal control performance is achieved without feedback. What this tells us is that if we have complete knowledge about the process (as encapsulated in the process model) being controlled, we can achieve perfect control. It also tells us that feedback control is necessary only when knowledge about the process is inaccurate or incomplete.

The IMC strategy

In practice, however, process model mismatch is common; the process may not be invertible [7] and the system is often affected by unknown disturbances. Thus the above open loop control arrangement will not be able to maintain output at set-point. Nevertheless, it forms the basis for the development of a control strategy that has the potential to achieve perfect

control. This mentioned strategy of Internal Model Control (IMC) has its general structure depicted in Figure 14.

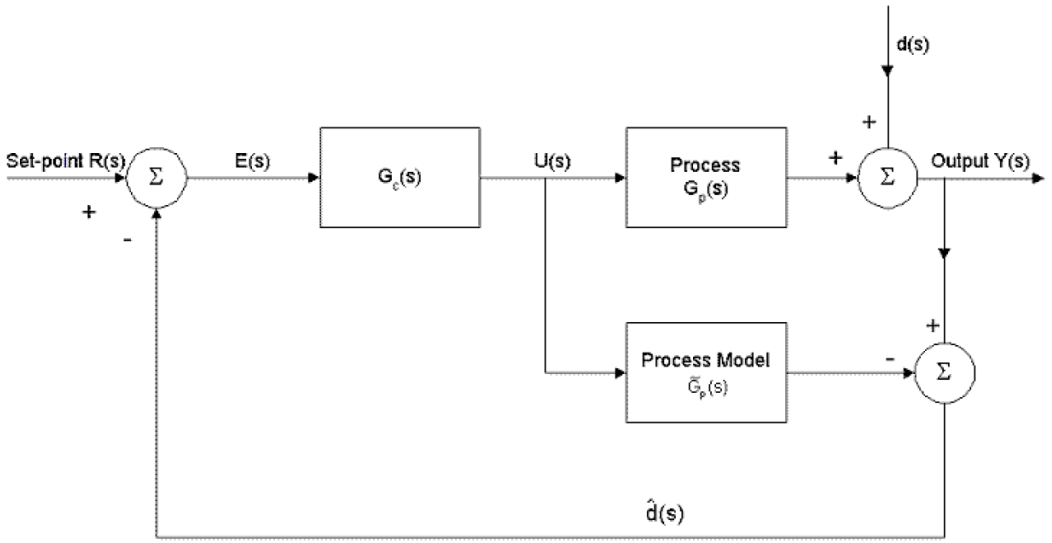


Figure 14: Structure of IMC

In case of the PMSM motor, the system to be controlled is far away from linearity therefore it could not be inverted in a very precise way. The IMC control technique is used just for an approximation of a robust controller and then from that point empirically being able to achieve an acceptable control of the system.

More relevant it is the identification of the system to find the best values of the controllers gains kp for the proportional part and ki for the integral one. It is intended to identify this PMSM as a first-order system, therefore it is necessary to simplify the motor behaviour since the PMSM structure is more complicated.

b. Calculation of the current controller parameters

To control the motor, the controller structure is a two layer control structure, where the outer layer controls the actual speed and the inner layer controls the stator current. So that it is possible to limit the stator current optimizing the energy used in the running of the system by controlling id and iq within the field oriented control software.

For each layer, it is taken as a system plant the part of the system which they are able to influence directly. That means it is not taken the whole motor model to calculate the controllers parameters but a simplification of it.

Simplified Plant and calculations for the Current controller Gp1

The current control, as its name indicates, controls the stator currents and therefore the torque. Its output is voltage, so it actuates on the motor by voltage and what directly affect this voltage too, is represented in the following figure. Being V_{in} the output voltage of the controller and the R-L circuit represents the stator of the motor for each phase.

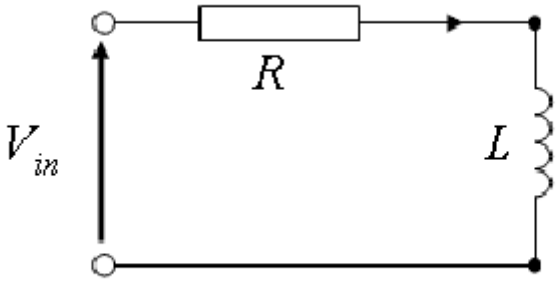


Figure 15: Motor's stator circuit

Knowing this it is now proceeded to calculate the values for the controller following the IMC method of inverting the plant model.

$$Gp_1(s) = 1/(R + Ls)$$

$$G_{IC}(s) = \frac{R + Ls}{1}; G_{IC}(s) = \frac{R + Ls}{1} * \frac{X_1}{s} = L * X_1 + R * X_1/s$$

Being Gp_1 the plant model, G_{IC} the equation for the current controller, X_1 the factor that takes into account the sample time and δ the tuning variable of the controller:

$$X_1 = \frac{2 * \pi * fs}{\delta}$$

fs: sampling frequency

δ: variable used to tune the controller

Enhance the values empirically

After the best value (gamma) for the controller is found, the system with the current control is tested. This process of having just one changing constant in order to control two parameters in the controller is not good enough when reliability and accuracy are wanted. Important it is to note that these tests were made in the real motor with the code charged in the MicroAutoBox, managing it from the computer through dSpace ControlDesk. This implementation in hardware will later be detailed more precisely.

So when gamma is well defined according to the necessities we have then the values around which the correct controller constants are likely to be. Afterwards the empiric tuning of the

controller is done now being able to change both controller constants (kp and ki) watching its behaviour with different references.

c. Results enabling current control

Finally after several test in the motor the values for the controller´s constants are fixed. These values are the following.

Kp = 0.5

Ki = 0.200

They assure a correct and stable current control as it can be seen in the following figures.

The desirable achievement is a reliable current control more than a quick one, because on it depends the correct running of the system when operating with the speed control as one. Therefore as well as houses are built from solid basements, the goal here is to build a robust controller, which will always have an acceptable response together with the plant when sudden changes appear. For this purpose it is better to have a slow but accurate response of the whole system (as it can be seen in the speed and current graphics of Figure 16) than a fast one, it adds robustness when afterwards designing the speed PI controller. This response is also desirable due to the noise that current sensors introduce and the over-dimension of the inverter used. These facts add instability to the system and to its behaviour, therefore a slow response as mentioned before will decrease their effect.

This however does not mean that those values will be the final values when the controller is tried enabling both of its layers. Depending on the speed control the current control values are likely to change since the goal here is to build a robust controller for then adapting the two layers of the controller.

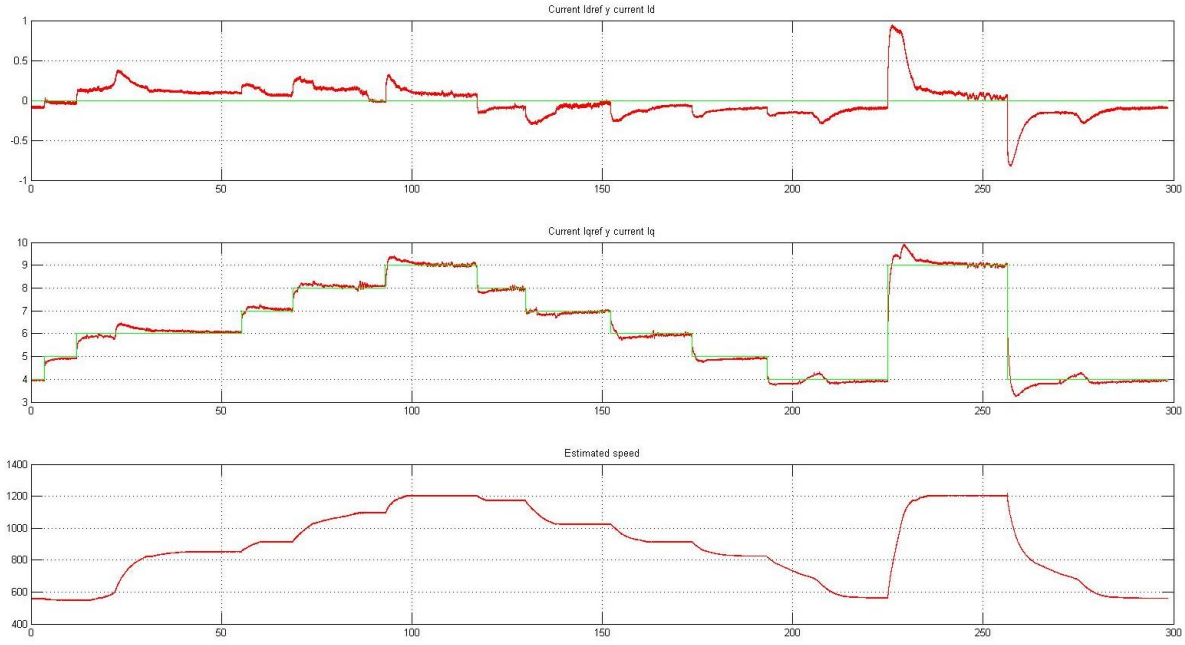


Figure 16: Responses in closed loop with current control

These tests were done setting the id reference to 0 since the iq, related to the torque, is the important component of the current in this case. So to avoid extra power delivered, id is forced to be near 0.

d. Calculation of the speed controller parameters

This second layer of the controller manages the angular speed of the motor modifying the current iq that reaches the current controller, so that now the two layers of the whole controller are created. Therefore the system circuit affected by this outer layer can be represented as the following.

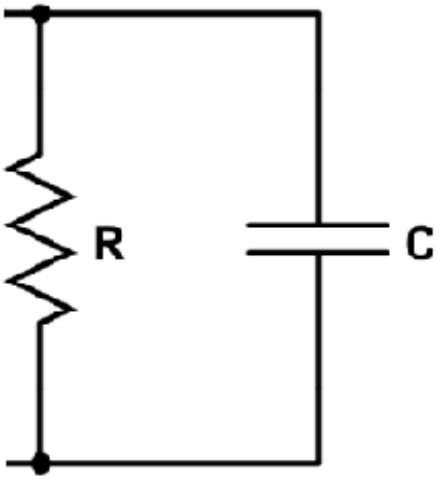


Figure 17: Simplified circuit seen by the speed controller

Representing the dc part and showing how in reality the change of current that the speed controller will make affects the system.

And using the IMC technique as before.

$$Gp_2(s) = \frac{R}{1 + \tau * s};\ \tau = R * C$$

$$G_{VC}(s) = \frac{1 + R * C * s}{R}; G_{VC}(s) = \frac{1 + R * C * s}{R} * \frac{X_2}{s} = C * X_2 + X_2/(R * s)$$

Being Gp_2 the plant model for the speed controller case, G_{VC} the equation for the voltage controller, X_2 and δ as well as before but they take different values as they are part of a different controller:

$$X_2 = \frac{2 * \pi * fs}{\delta}$$

fs: sampling frequency

δ: variable used to tune the controller

Empirical election of the suitable values

Same as with the current control, it is continued with the procedure of the tuning of the speed controller, knowing that in this controller the goal for it is different. The best response as possible of the complete system already with the current control working will be now searched. Saying best response it is meant, in this case, with a slight or no overshoot and as quick as possible within that fact.

Some problems appeared in the tuning of this controller because of the non-linearity of the motor, its behaviour was not the same while changing the reference through all its possible range. To give an example, the response of the motor is different when a deceleration of 300rpm is required than when the same change but accelerating is done.

This increases the difficulty to control the system since the correct running is not assured when some valid controller values for a certain behaviour of the motor are calculated. Then, as the motor is not linear those constants are not suitable enough when it is running at different speed than the previous situation.

e. Results enabling the two layers of the control

With this unavoidable fact in mind, the values for the cascade loops which can in an accurate way control the PMSM as it is shown in the next figure are:

Current control	Speed control
Kp = 0.337	Kp = 0.05
Ki = 0.500	Ki = 0.010

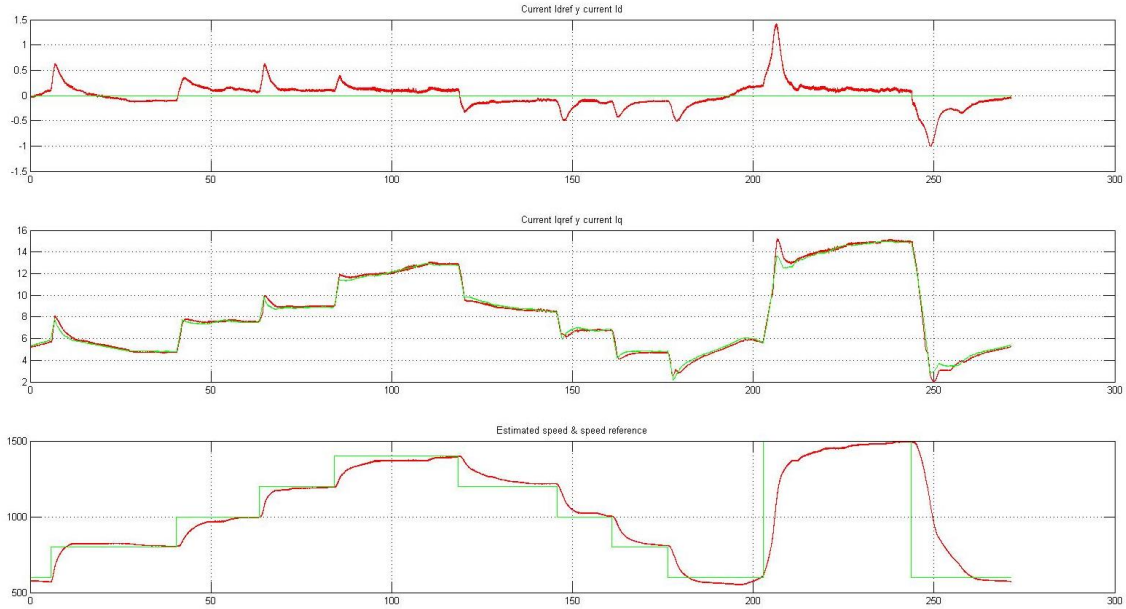


Figure 18: System responses in closed loop with speed and current control

As shown in the figure the non-linearity is depicted in the velocity graph, the motor's way of reacting to changes differs in relation to the magnitude of the change.

These tests were done fixing the value of id to 0 or near 0 and changing the speed reference afterwards.

f. Reaction against disturbances and solutions for the non-linearity

After the linearly correct tuning of the controller, the system was tested to see its reaction when a perturbation is being applied.

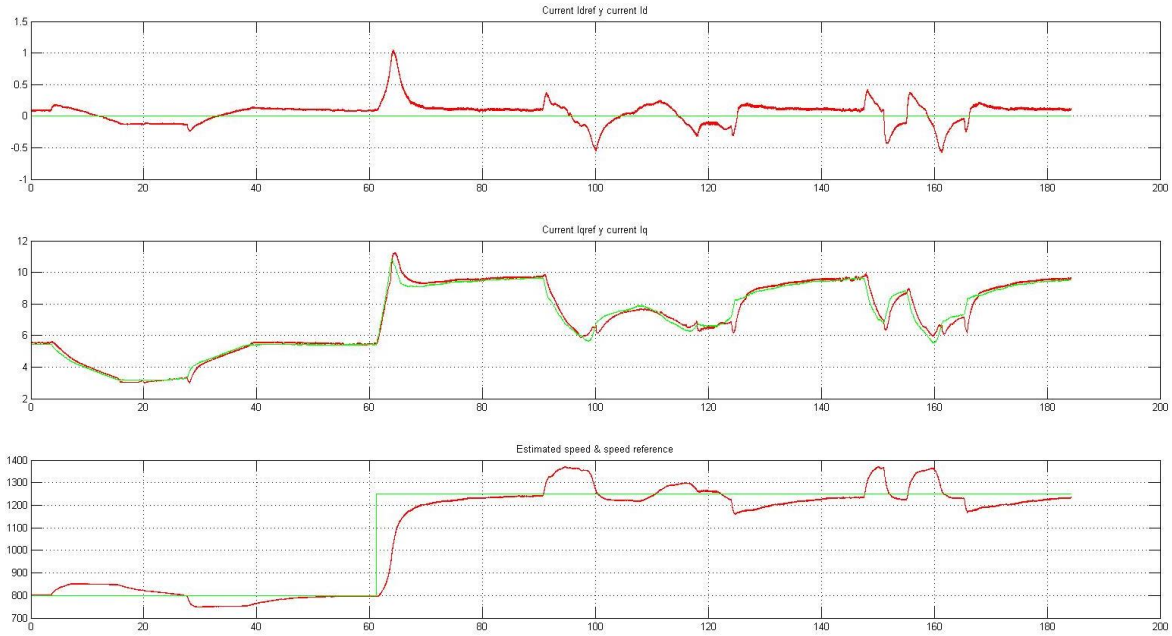


Figure 19: System responses in closed loop with speed and current control applying an external perturbation

It can be seen how it reacts to the external events, when the perturbation is applied the system velocity increases at first as it diminishes the wind resistance, then it is equalized decreasing the power delivered in the motor. On the other hand, when the external force is removed the speed decreases, and increasing slowly the power delivered, velocity is again following the reference.

It can be concluded that this system is roughly robust against sudden changes in the reference as well as in the load, as can be seen in the above figures. The only disadvantage of the system is the speed of its response after any change, but as it has been pointed out before the motor behaviour is far away from linearity and therefore the goal of the system tuning was not to achieve fast responses but accurate ones that can provide good performance at any reference value and before any perturbation.

Future works on this field are oriented to a better control of a non-linear system with an adaptive control. In other words, control a non-linear system with a non-linear controller in which the goal is to linearize the plant behaviour adapting the controller to any working point.

This is indeed the next step of the project, the study and posterior development of the same cascade control system done however with predictive adaptive controllers.

7. CONTROL OF A PMSM WITH ADAPTIVE PREDICTIVE CONTROL

a. Adaptive predictive control

Introduction to predictive control

Here a new way of control is presented enhancing the use of the PID controllers. For the approach predictive controllers are used to control the behaviour of the system.

The difference between these two types of control [1] techniques are related to time response and offline training. Predictive control attempts at anticipating the error by using a model of the plant and giving the necessary controller output for reaching the target. Meanwhile, with the state of the art PID controllers, we must wait for the error to occur in order to react and correct it. Therefore the time until the system reaches the reference should be longer with the pid structure if a good offline training is done or , with another words, if the model is precise enough.

This is developed knowing beforehand the model of the system which is to be controlled, data is extracted from the real plant and, before running the control loop (offline), an accurate model is built [9]. This model will be used in the predictive control in order to know beforehand how the system is going to response.

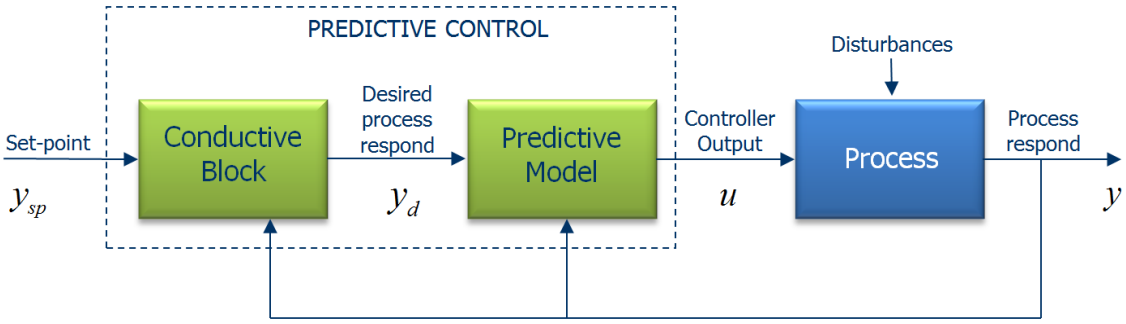


Figure 20: Structure of a predictive system

For the process of building the control it is taken into account that the predictive controller reacts in a better way when smooth changes of the reference happens. Therefore a block called Conductive Block after the reference and before the real controller (Predictive model) is included, being its purpose to soften the reference input to the controller.

Adaptive control

Predictive control is related ideally to a unenhanceable offline training in order to achieve good results, there is another way of controlling which does not need such an exhaustive work offline. The adaptation machanism [2] is used mainly when we have little knowledge of the process that is being controlled or when the model of the system is not quite accurate.

Its goal is to adapt the parameters of the model while the system is running, so in this case we do not need this exact offline training but a process which is developed online. For this it is supposed that a powerful control unit is needed for this online training, nowadays with the so powerful computers and microcontrollers this issue is not a problem.

Although this predictive adaptive control is not yet into the industry, the reason is surely its complexity when developing any system.

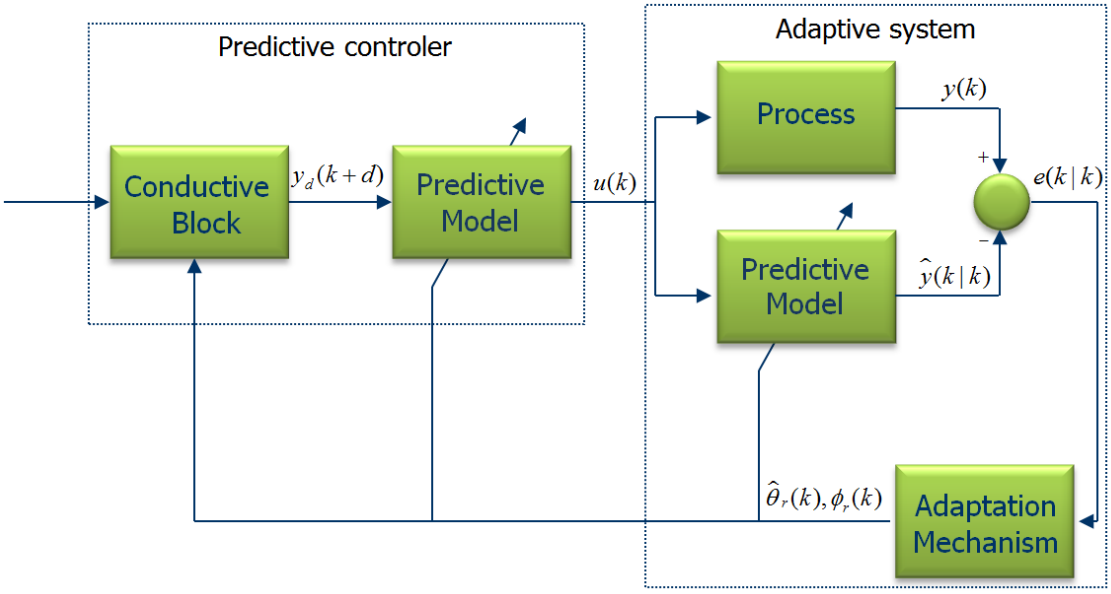


Figure 21: Structure of a predictive adaptive system

In this figure the basic ideas for a predictive adaptive control are depicted. Basically it consists of creating a simple predictive model in order to know beforehand what the process is. After each iteration a predictive error will be generated because of the difference between the real process and the model calculated offline. This error is read by the adaptation mechanism block which adapts the predictive model to enhance it and lower the predictive error.

When the matlab code is finished with all the parameters calculated, there are two values of high importance. They act as if they were the Kp and Ki in case of the pid controller. This parameters are two: the Control Period (CP) and the Prediction Horizon (Lambda).

The *Control Period (CP)* is the time between controller actions. The controller will not do any action before the running time gets higher than the CP.

The predictive model also allows *measureable disturbances*, i.e. additional input variables affecting the process respond which the controller cannot influence; but it does take them into account for anticipating and minimizing their impact, just by computing the model parameters for measureable input disturbance over the prediction horizon *Lambda*. This Lambda is the number of steps after which the input signal is considered steady when predicting the future response of the system. In other words, it gives in how many steps the controller will make the system output stable. In order to achieve that the computer or microcontroller used to compute those calculations will have more or less work to do in each iteration.

b. Example of adaptive predictive control

In this example a second order model of a system is simulated, the values of the system have been chosen randomly and also the disturbance included which will be shown afterwards. With the example we will be able to comprehend how an adaptive predictive control works, watching the graphics, starting from a model calculated offline which is not accurate enough to work correctly with no adaptive control.

It can be seen that the block named Adaptive Predictive Controller is placed as if it was a normal pid controller with two inputs, one is the system reference and the other the feedback value which comes from the output with or without applying disturbances.

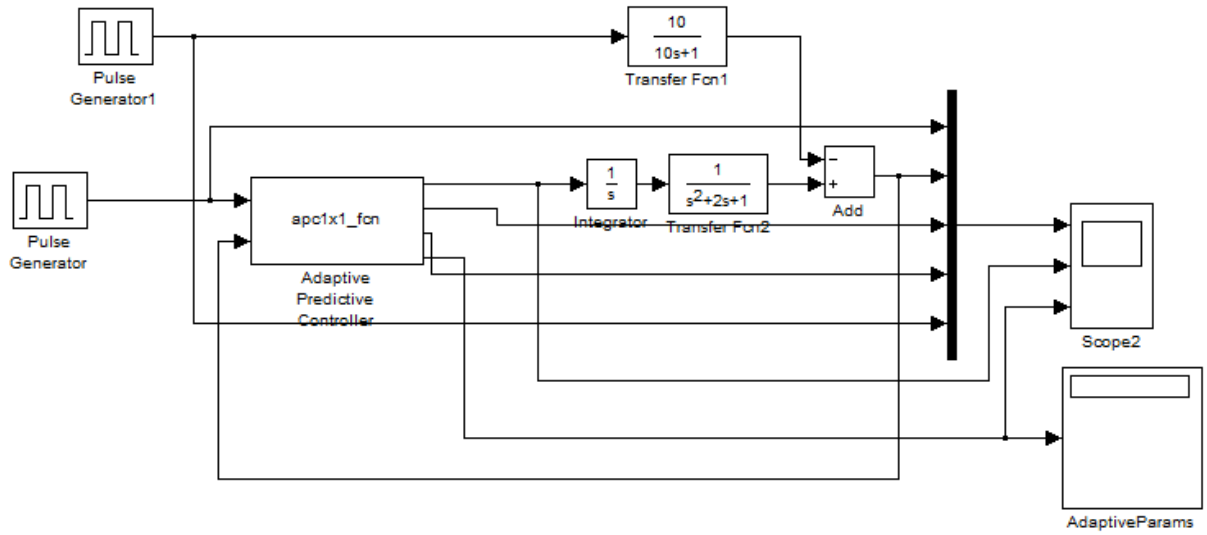
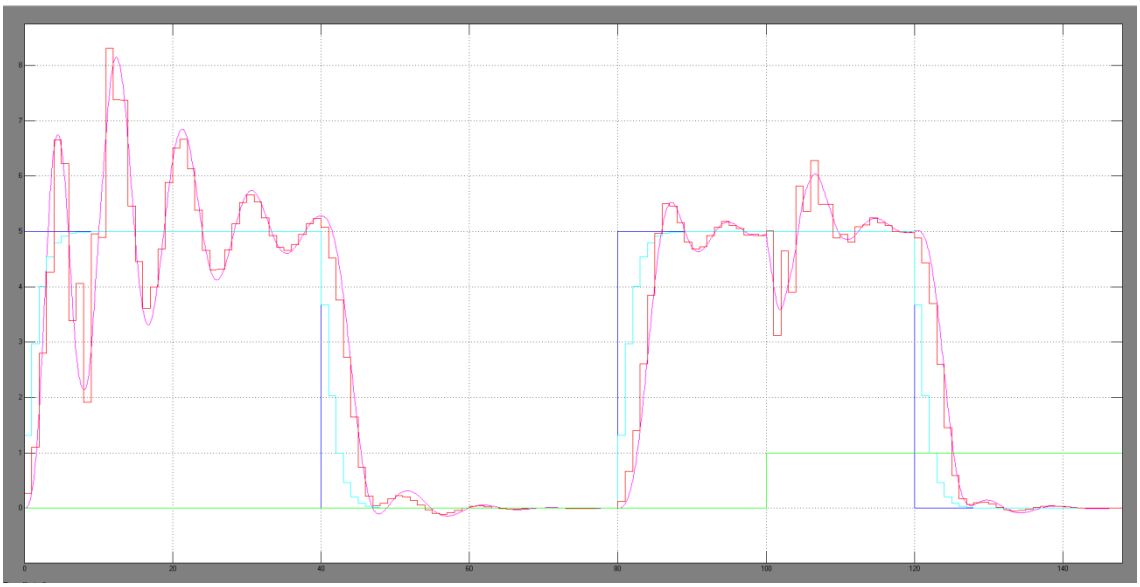


Figure 22: Simulink model of a system with predictive adaptive control

In these plots it is shown the input of the system as well as the output, the predicted output and the desired output in order to see the evolution of the prediction starting with a roughly calculated model of the system and then adapting the system parameters online as it is running.



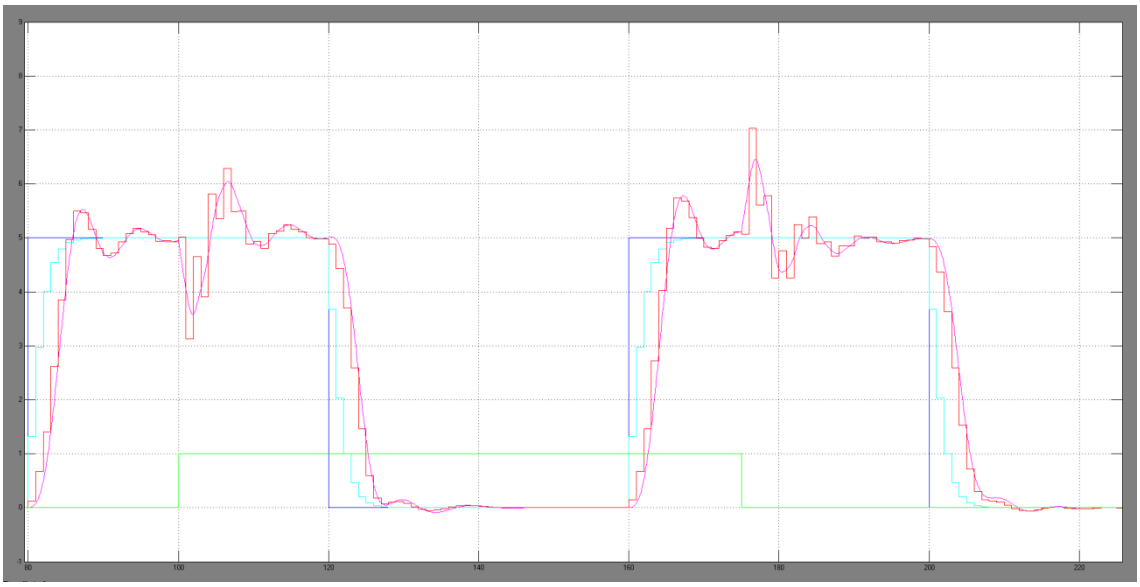


Figure 23 and 24: Outputs of the model with adaptive predictive control

The whole controlled system reacts exatcly how it was described above. In the beginning its behaviour is not accurate enough in order to meet any requirement in speed response, time of stablishment or maximum overshoot. But then it can be seen how the system learns from the previous iterations and when the next change in the reference comes, its reaction has been totally enhanced by the adaptive process of the controller. In the last graphic the evolution of the system parameters is depicted, there we realise how the controller is trying to adapt the model of the system by changing these values and making a more accurate and reliable system.

c. Our approach with the two-layer structure

Previous approach

First of all I would like to remind how was made our two-layer structure controller and what was its structure. It has been described in previous pages in more detail

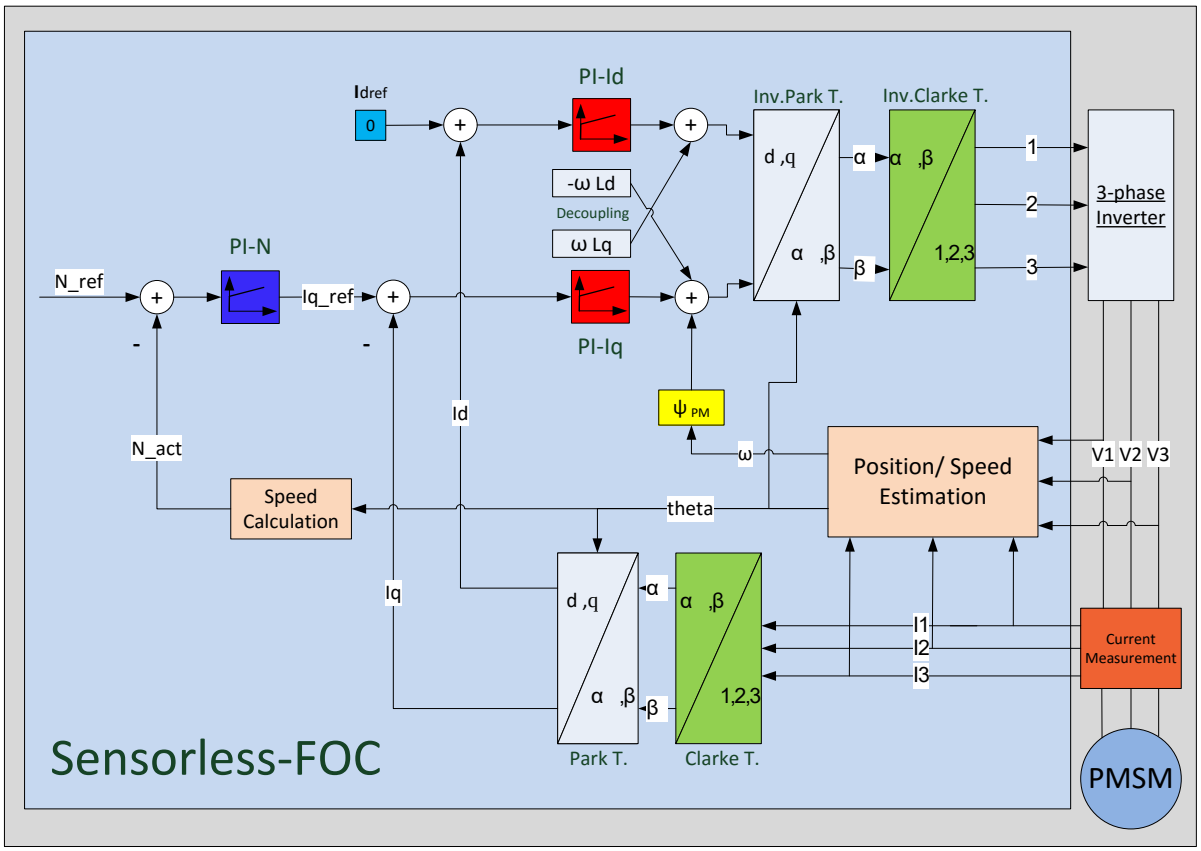


Figure 25: Global structure of the application

The PID controllers following this structure are organized in 2 levels, one level is for the current control in which 2 controllers are needed, and the other level for the speed control which only can influence the iq part of the current because it is the part related to the torque and therefore the speed can be changed. With this kind of structure one level of controllers has to be the so called master and the other the slave. This means that if the master controller layer is deactivated the whole control system is not able to work even though the slave could be enabled. In our case the inner layer makes this function, current controllers have to be on otherwise we will have no control over the motor

Idea of converting the previous PIDs to adaptive predictive controllers but same structure

After refreshing all these facts again, the new approach is to proceed to change each controller, instead of using pid technique adaptive predictive controllers will be used to

control the three-phase motor. Note that this approach will first be tested in simulation and then, when its correct behaviour is checked, it will be implemented in the motor.

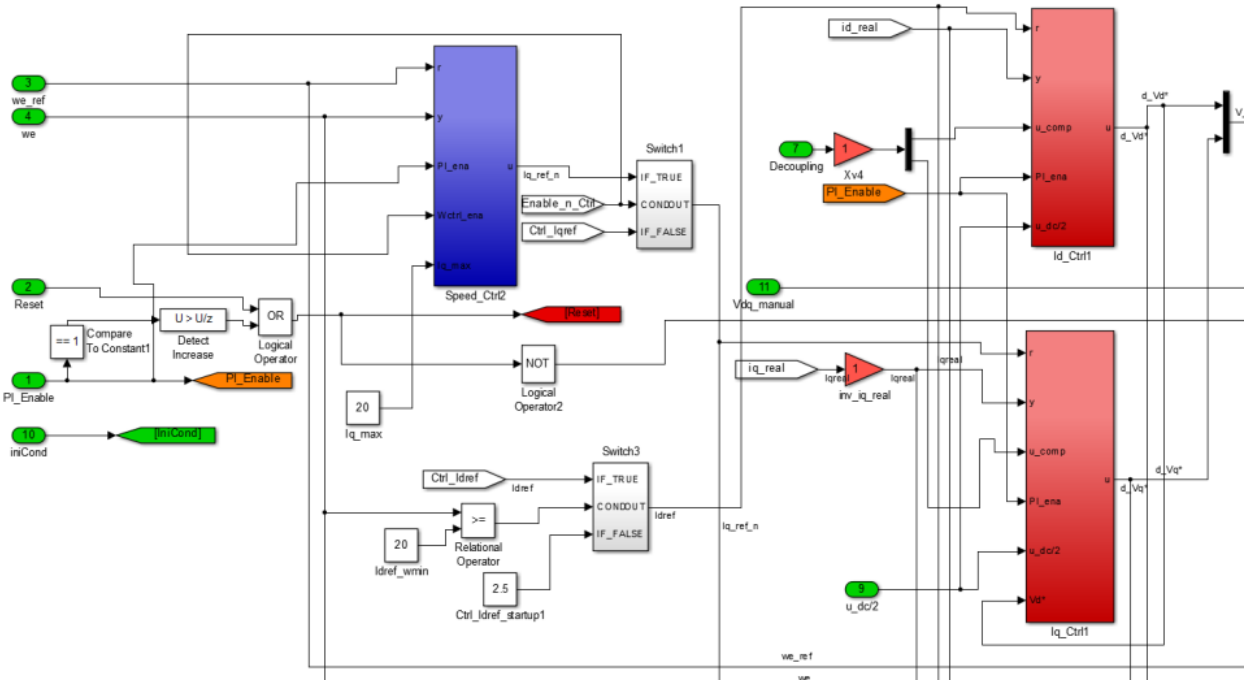


Figure 26: Model of the two-layer predictive adaptive controller

Slight changes will be made in the control software to adapt it to the new control technique. Once the structure is defined, the next step is calculate the model parameters which will simulate the behaviour of the controlled motor. As we explained before these parameters are not scrutiniously calculated offline, even though they play an important role in the future control. A simple model of the motor is settled in order to make the online adaptive process less time consuming.

d. Calculation of the parameters (how)

Taking the previous pid controllers, we know how was the model of the system they were controlling, because the IMC technique was developed; that means that if information about the controller is known it is also possible to guess how the model of the plant looked like.

As an approximation of the model a transfer function in the discrete space is used. Beforehand the system model was calculated in the Laplace domain in order to make the two layer pid controller system using the IMC control tecnique.

Now two ways could be followed, one is to calculate theoretically the model of the motor in Laplace, as done before, and afterwards transform it to the discrete space. The other method, which is the chosen one due to its simplicity, consists of taking the pid parameters already calculated for the transfer functions of the cascade controller system, and from them the discrete motor model can be determined.

As the discrete parameters had been settled for an IMC control, which worked in a reliable way, we used these same values in the discrete space to calculate the model of the motor and the parameters needed for the adaptive predictive controller. These parameters will of course be different for the current control and for the speed control.

The method to compute them is the following:

The model of the plant should be of the form:

$$model = \frac{b0 - b1 * z^{-1} - b2 * z^{-2} - b3 * z^{-3} \ldots}{a0 - a1 * z^{-1} - a2 * z^{-2} - a3 * z^{-3} \ldots}$$

Then beginning from de PI controller, the plant model is reached inverting the transference function.

$$PIi = kpi + \frac{kii * z}{z - 1}$$

Kpi (proportional value of the current control) = 0.337

Kii (integral value of the current control) = 0.500

$$sys = \frac{\frac{1}{kpi + kii} - \frac{1}{kpi + kii} * z^{-1}}{1 - \frac{kpi}{kpi + kii} * z^{-1}}$$

The same process is done with the speed PI controller.

$$PIn = kpn + \frac{kin * z}{z - 1}$$

Kpn (proportional value of the speed control) = 0.050

Kin (integral value of the speed control) = 0.0105

$$sys = \frac{\frac{1}{kpn + kin} - \frac{1}{kpn + kin} * z^{-1}}{1 - \frac{kpn}{kpn + kin} * z^{-1}}$$

Already with both of the models of the systems seen by the controllers obtained, the parameters for the code have to be picked. To accomplish this task 4 parameters will be selected for the code in matlab: [b1 b2] as numerator values and [a1 a2] as denominator; being b2 = b1/10. The reason why this value is because b2 must have the least influence possible in the placement of the closed loop zeros, so that it does not affect much in the response. The same with a2 but with the closed loop poles.

But then, why choosing a b2 or a2 value for the model in the code and ignoring b0 and a0? It is wanted that the parameters chosen have influence in the zeros and poles location, therefore also in the response; this values are the constants which are multiplying the z in the discrete space. Moreover, as two parameters or more are needed to make the model more accurate, the second constant b2 or a2 is settled to a tenth of b1 or a1. As a result of that the

model has 4 parameters, two of them at first do not influence much the model previously found. However, when the program runs and the adaptive process is enabled these second parameters will be of more importance since the controller has more possibilities to change its response according to the situation.

e. Tuning of the three controllers (structure and differences in the code, find correct values for CP and Lambda)

Once the parameters are known the code is ready to be executed. Then the task is to set first the current controllers, the inner layer so to say, before the whole control is proved. In the simulink model it is needed just to change the pid controller structure by our code programmed as an level-2 matlab s-function, setting the needed inputs and the required outputs for each of the two currents to be controlled.

In order to make the controller to work properly two values or inputs to the controller need to be studied according to the behaviour of the system. These parameters settled as inputs are the control period (CP) and the prediction horizon (Lambda). These two correspond to different control actions which have to be taken into account when selecting the values offline.

There is also another value of much importance for the adaption process which is the adaption coefficent, called AM in the implementation code. This parameter can just be any value between 0 and 1 and weights the adaption matrix when calculating the adaptive parameters for the next iteration, being 1 the most adaptive value and 0 no adaptive process.

Make the current controllers work

When the inner layer is adapted to the new controller, the simulink model with the controller code can be proved. It is also important to say that firstly the program will be run on simulation and if good results are achieved then it can be transferred to the MicroAutobox and check whether it is also working correctly or not.

However before trying to transfer all data to the microautobox the correct functioning of the whole system must be assured in simulation, so that its behaviour can be studied and future problems when running the motor can be avoided.

In order to tune the controller, as it was said before, two important parameters need to be settled as the system requires. According to the speed and accuracy of the motor response the best values for these can be selected.

The following graphics shows how the system reacts to the inputs with different values of CP and Lambda.

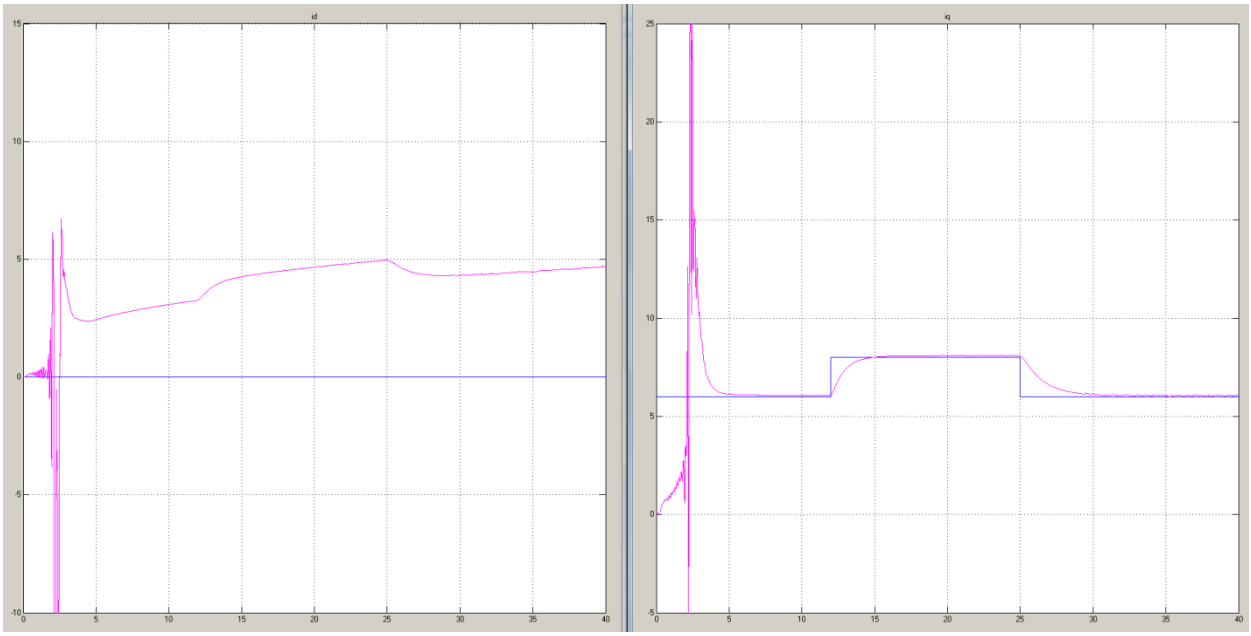


Figure 27: iq and id response just with current control

At first the best response in terms of fluctuations and accuracy, mostly in relation to iq (right side) the one related to the torque, is chosen. For this example of current control the following values of CP and Lambda were chosen:

CP: 50 and Lambda: 15

Even though, when most likely later on these values are going to be changed, because of the tuning of the speed control. The values are first chosen in order to have the best response in terms of current, however that does not mean that the speed response later will also be the best possible.

Make the two layer controller work reliably

Once the current control is set and tuned, the following step is to achieve the same with both layers of the controller on. As it was said before the values that the current controllers have may not be definitive, since making changes in the current control loop can affect positively to the speed control which in the end a good speed control is what we are willing to have.

When tuning the system with the two layer controller enabled, many acceptable responses with different CP and lambda values were obtained. These will be shown and discussed in the next section.

f. Different tests

The different values for the control period and prediction horizon were tested in the system which also includes a disturbance in the torque. Graphics and numeric results are showed as follows.

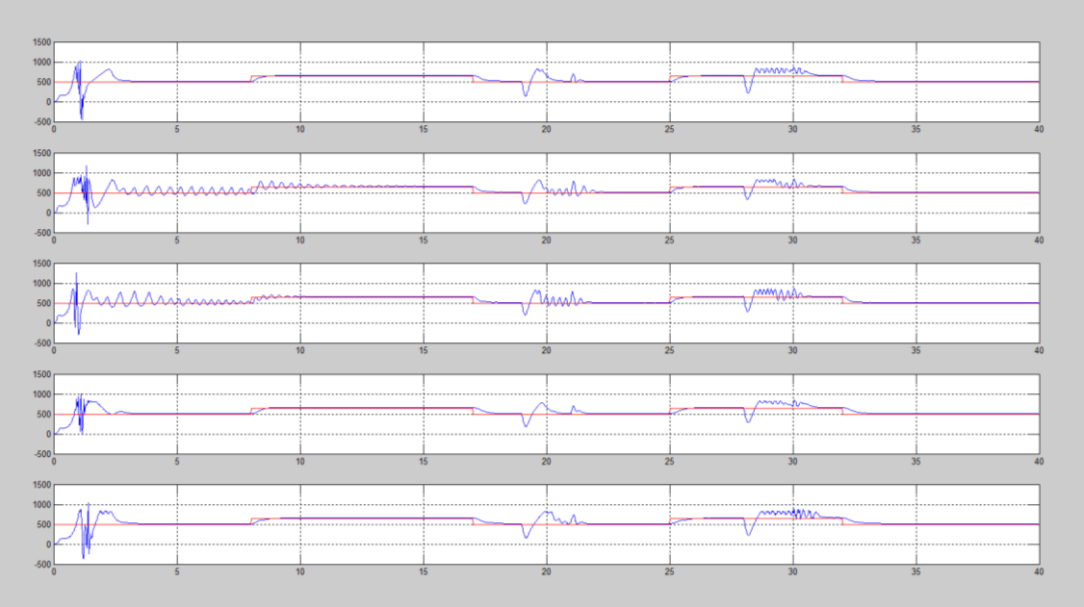


Figure 28 Graphics of model with speed control with different values CP and Lambda

```
>> apcs_comparator
Speed control:50 10; Current control:30 10
Avg_error: 50.028164
Max positive deviation: 35.90%
Max negative deviation: 167.74%

Speed control:40 10; Current control:28 10
Avg_error: 42.165840
Max positive deviation: 36.26%
Max negative deviation: 149.01%

Speed control:35 10; Current control:25 10
Avg_error: 41.563438
Max positive deviation: 33.62%
Max negative deviation: 157.70%

Speed control:55 10; Current control:35 10
Avg_error: 48.665016
Max positive deviation: 44.60%
Max negative deviation: 155.75%

Speed control:60 10; Current control:40 10
Avg_error: 58.020073
Max positive deviation: 36.35%
Max negative deviation: 165.76%
```

Figure 29: Error values of the system using speed control with different values CP and Lambda

From this parameters in Figure 29, which have been computed from the graphics of Figure 28 in a determined time interval, exatcly from second 12 to 35, the best controllers in terms of accuracy and maximum deviation can be chosen. According to the numerical values, the second and third controller are the most reliable ones and are here depicted in more detail.

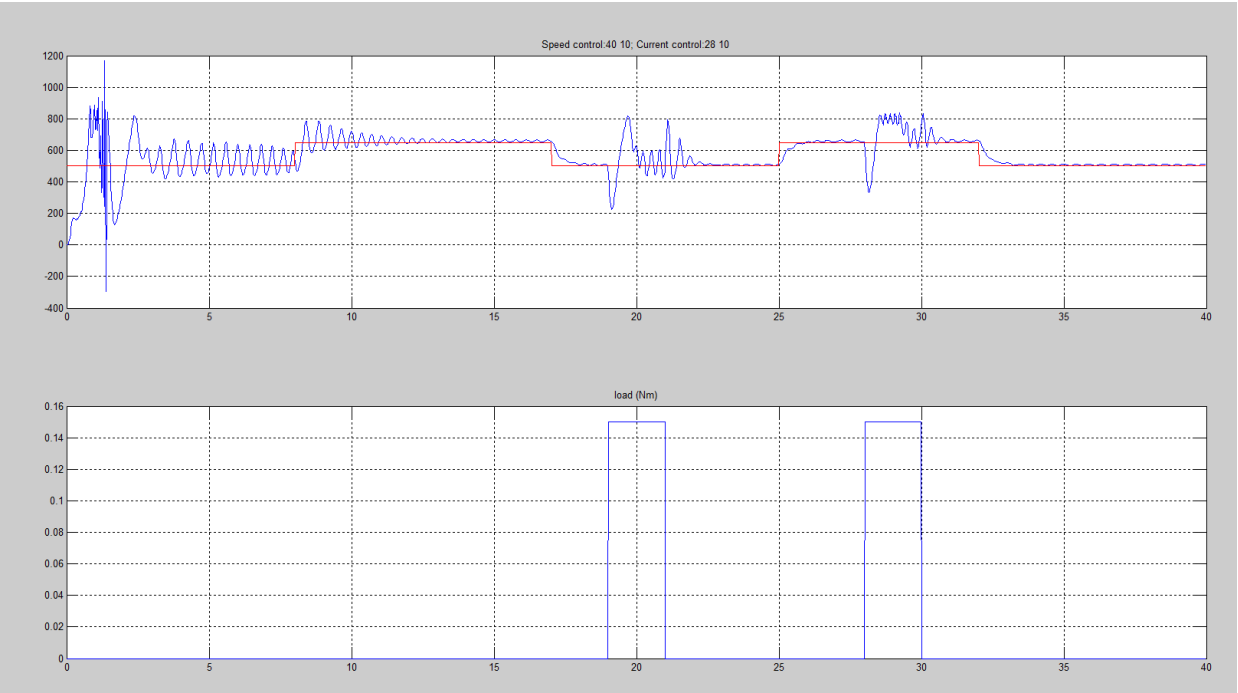


Figure 30: Response from the second controller with its correspondent values

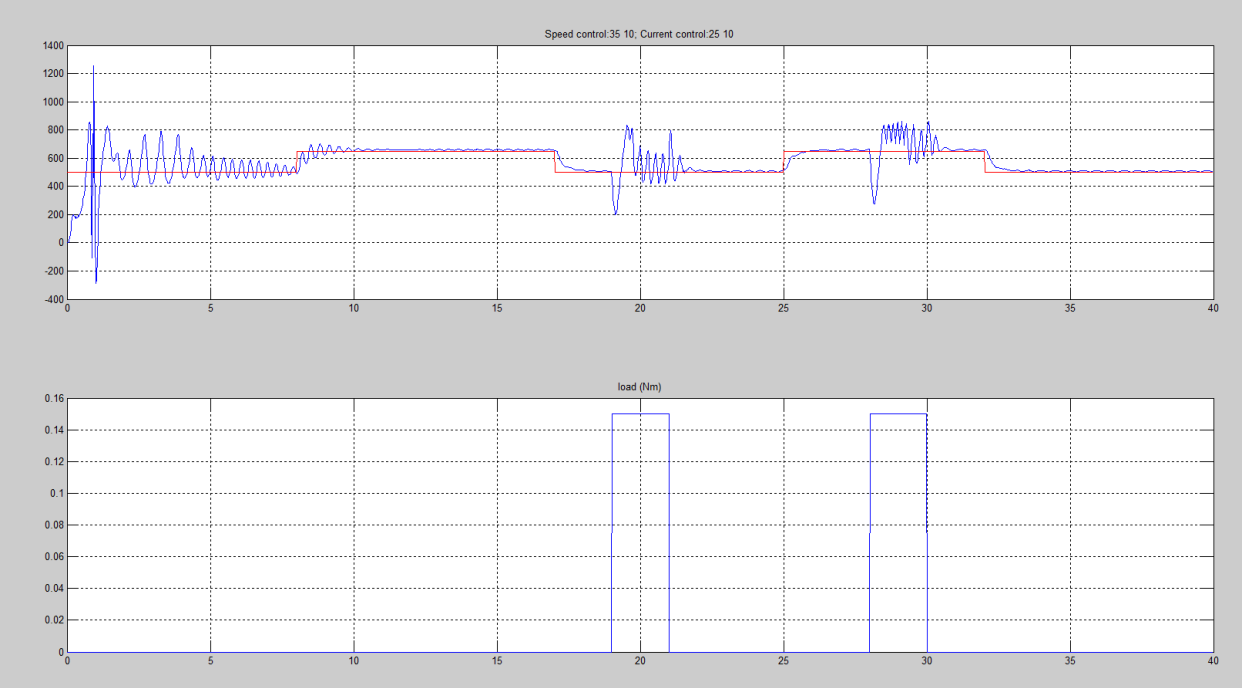


Figure 31: Response from the third controller with its correspondent values

Surprisingly the best controller responses are the most oscillating in the beginning but when they are “adapted” the error and deviation are the lowest. The best solution between those is searched, forgetting about the fluctuations at the beginning when the change in the reference is quite high, could match the graphic number 3. Even though the response of number 2 seems to be faster when changes appear, the third controller is slightly more accurate and its adaptive processes achieve good results as it can be seen.

g. Adapt the adaptive parameters to enhance transitory response

To try to prevent the high oscillations when running the model we adapted again the adaptive parameters. This redundance mean that after a first simulation with the current parameters, its initial value will be changed with the one they have at the end of the simulation. The main reason for doing this is that the parameters stabilize as the program runs to fixed values at the end of the simulation, when the model has had enough time to adapt them. Therefore they then represent better the motor when the process of adaption is finished, so that the system, theoretically, should work in a better way.

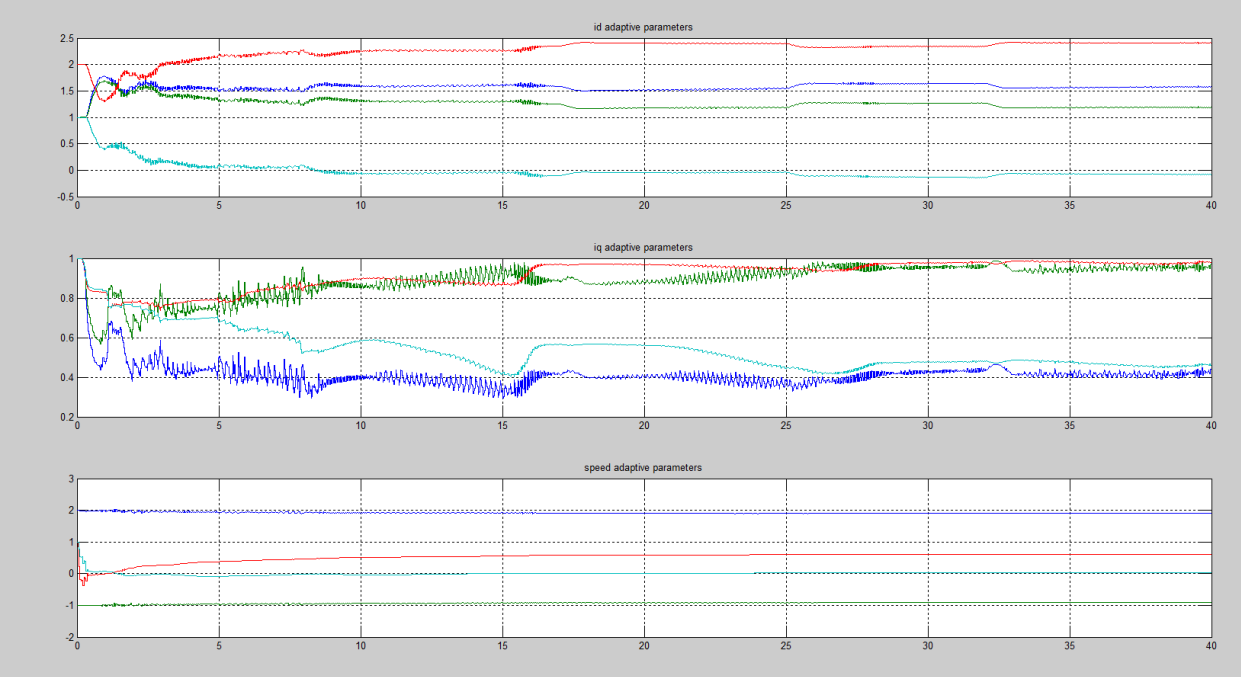


Figure 32: Adaptive parameters’ evolution

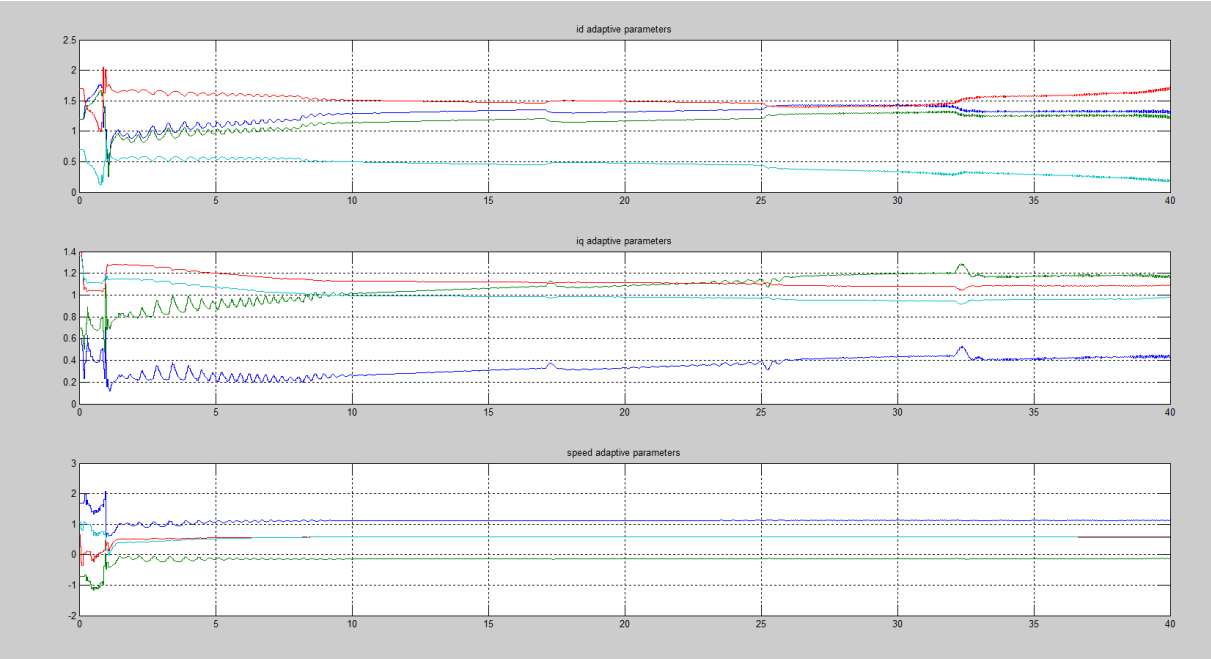


Figure 33: Adaptive parameters' evolution adapting initial conditions

The figures represents how the adaptive parameters change at first from the values we calculate theoretically and then by changing them after a first simulation of the system. It can be observed that the parameters do not change in the same way, being these changes in the second graphic slightly lighter after the offline empirical work.

h. Linking the project to ControlDesk and microautobox and translation of the code

The main goal of this project was to implement the control of the motor in a real 3 phase PMSM and for that purpose the transference of data and code from the simulation to the hardware is needed. As we know from the pid application before, a microAutoBox is used as the CPU of the whole system and the program dSpace to communicate in real time with the computer to the hardware. The section about hardware implementation later on in this work is documented with all important information about dSpace and MicroAutoBox.

With this second part of the desired control for the system, the process of transferring the program will be more complicated than it was before with the pid application. In this case level 2 s-functions are used for the predictive adaptive process, and these will give some problems when trying to compile and create a target in the RTW matlab.

Problems with the level 2 s-functions

The problems that we met when trying to compile the code and creating a target were basically the s-functions. After analysing the code and checking that there were not such an error, information about matlab through MathWorks and about dSpace was searched. This

lead to the knowledge that these level 2 s-functions were not readable or recognisable neither by dSpace and then nor by the microAutoBox and therefore in order for the code to work there must be no 2 level s-functions in the program.

Translating the code to basic matlab funtions

To overcome this issue and assuring then its correct operation, basic Matlab functions instead of level 2 s-functions are used. These Matlab functions are far simpler than the others for the compiler and linker to work with them since they do not have intrinsic functions or already made structures inside, they also cannot save any number of values in order to be used in the next iteration. Unfortunately the reasons that makes the Matlab function easier to use in the RTW are also the ones that makes it difficult for the programmer who implements the code; since there is no internal structure to save the data or no intrinsic functions to deal with it.

To try to transalte the code and make it work the same way as with the s-functions, a model is created using as a reference the same system as in the adaptive predictive control example; so that it can be compared with the level 2 s-function simulink model since they perform the same task with exatcly the same values and reference.

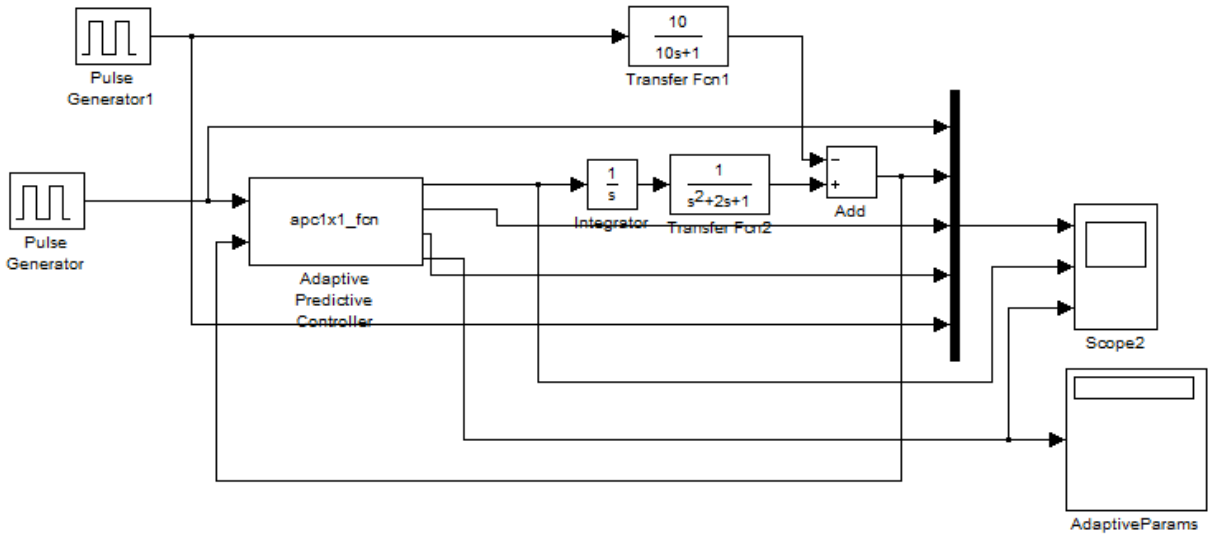


Figure 34: Random model using level 2 S-Function

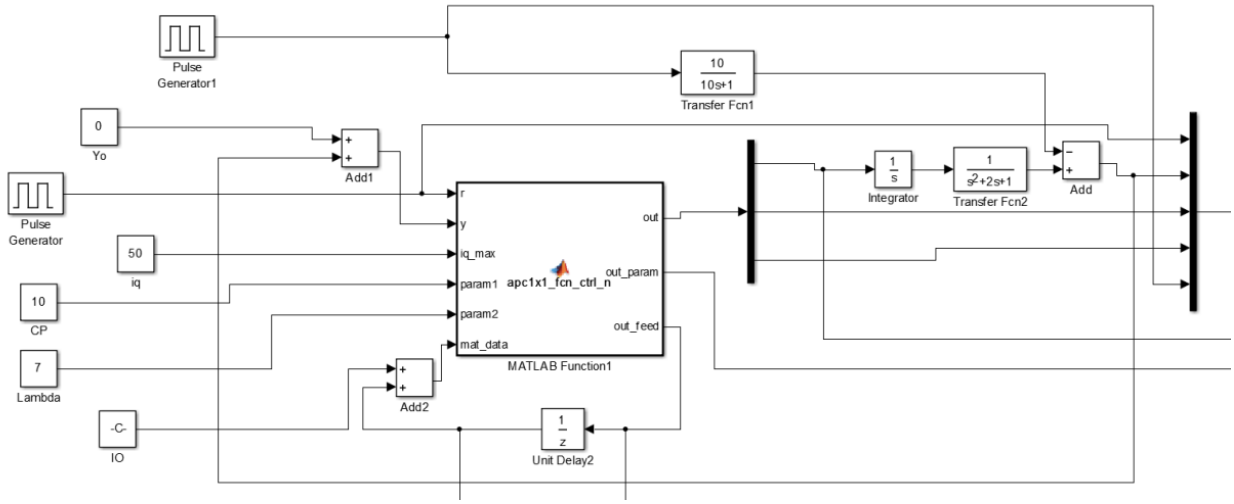


Figure 35: Random model using Matlab function

One of the most important differences, which was mentioned before, can be seen from the appearance of the figures above; it is the necessity to feedback the values from the last iteration which are needed for the next one in the predictive and adaptive process. For that purpose a unit delay of one step of time is used in the feedback loop so that the past values are available in current time.

Here a comparative graph between the implementation of both models using the same input is presented. The output curves look quite similar to each other, being the only difference the fluctuations of the output at the starting of the simulation. The maximum overshoot is slightly higher when implemented with matlab function, this could be due to the different solvers used for each model, although the time step time of the simulation is the same in both cases.

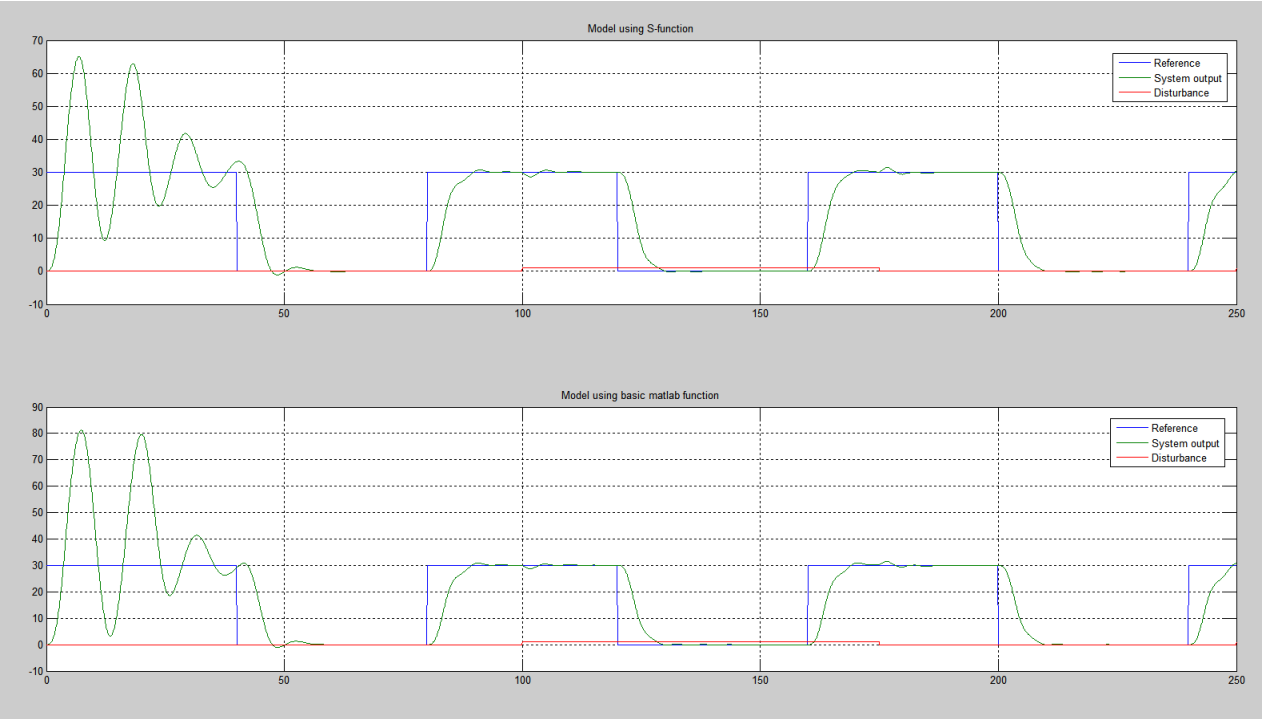


Figure 36: Comparison of outputs (s-function vs created matlab function model)

These curves prove the accuraccy of the created model in relation to the one existing before. Therefore, with the matlab function block, now is posible to implement the control of the PMSM motor model with this same function used in this part

Implementation of the controllers in our model with the basic matlab function

As it is depicted before with a random model whose purpose was just to try to determine the algorithm for a basic matlab function that works in the same way as that from the 2 level s-function. Now that its behaviour is checked, it is proceeded to introduce and implement the blocks into the two-layer controller.

Here there is only showed one layer of the controller, the speed controller to be precise, since the others will have the same or very similar implementation.

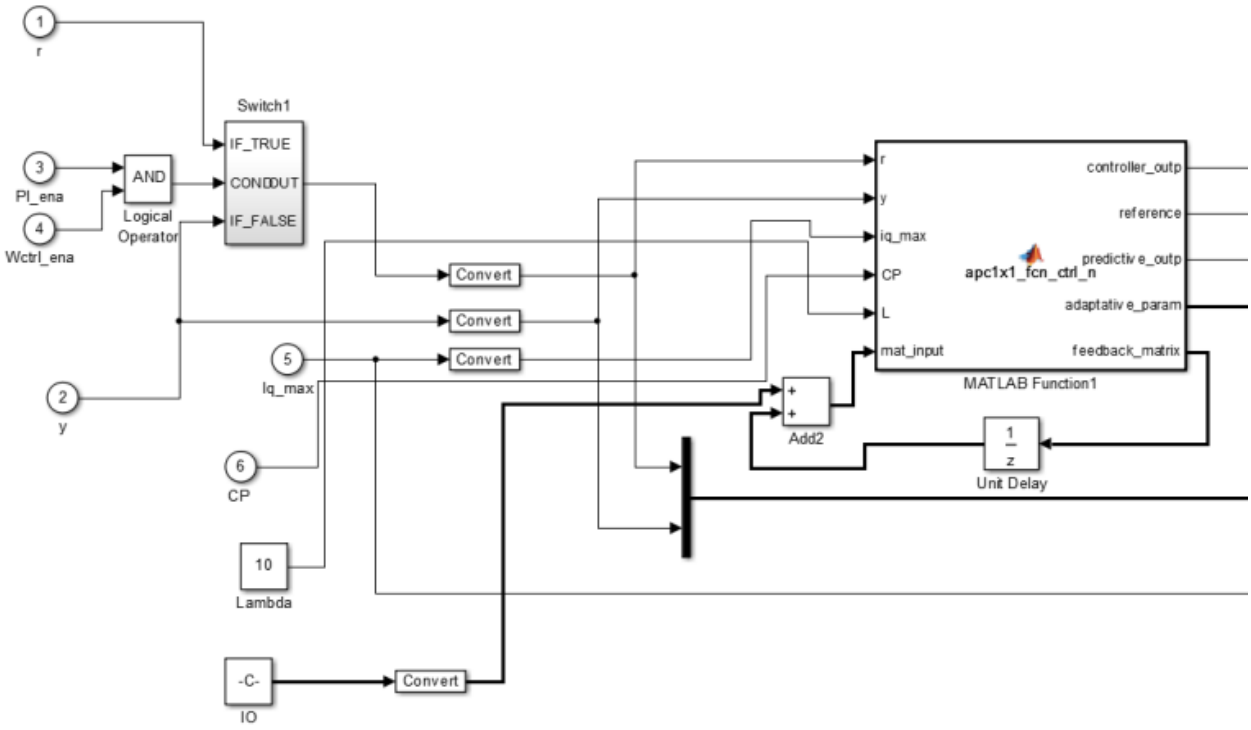


Figure 37: Speed controller part

i. Proof of the code in an example with the basic matlab functions

This are the results of the model in simulation replacing the 2 level S-functions for basic matlab functions. In the following graphs a mechannical torque is applied as a disturbance to the motor. This is showed in all figures with both components (id and iq) of the current and with the speed in order to see how they change when the unexpected torque appears.

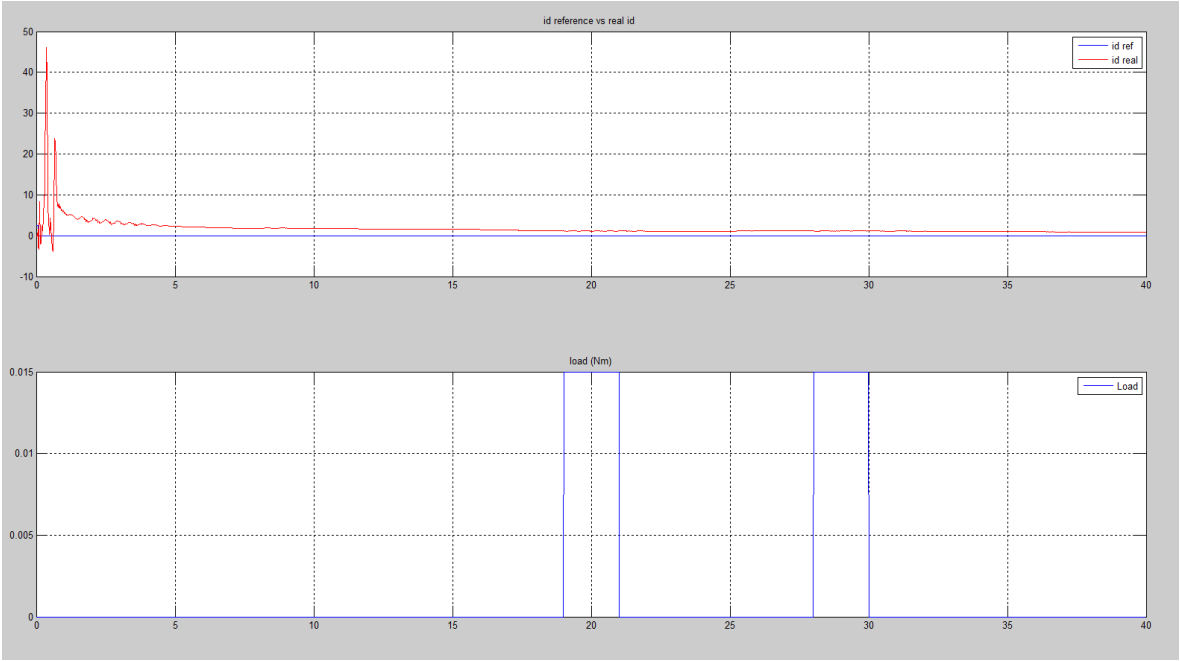


Figure 38: Real id with its reference and applying load

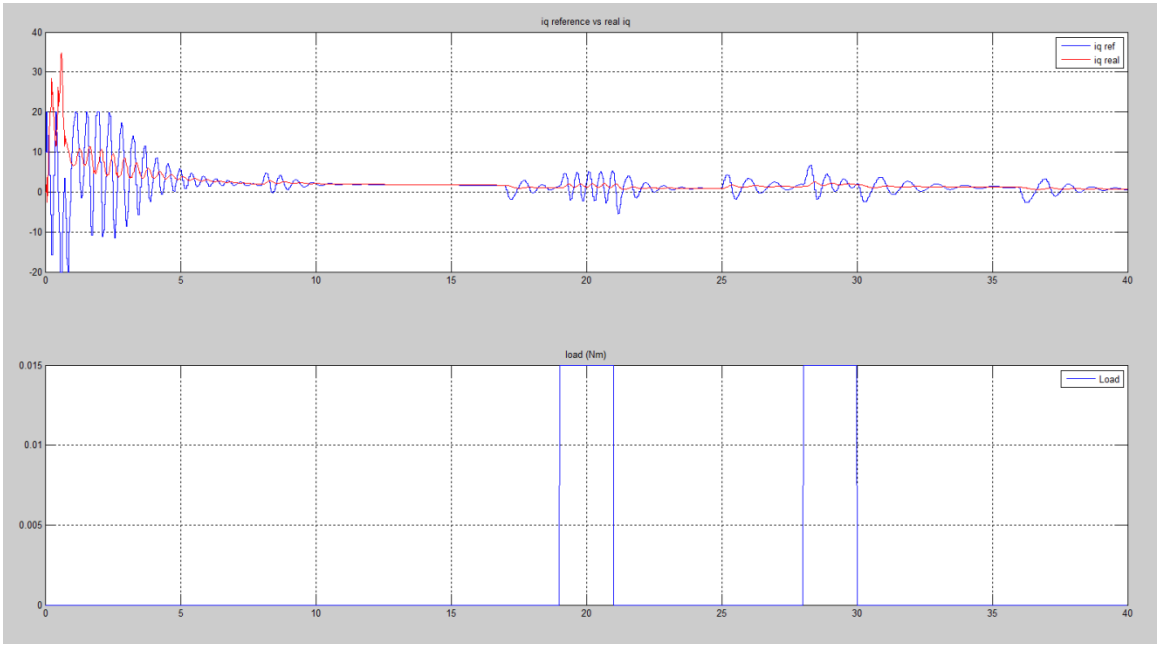


Figure 39: Real iq with its reference and applying load

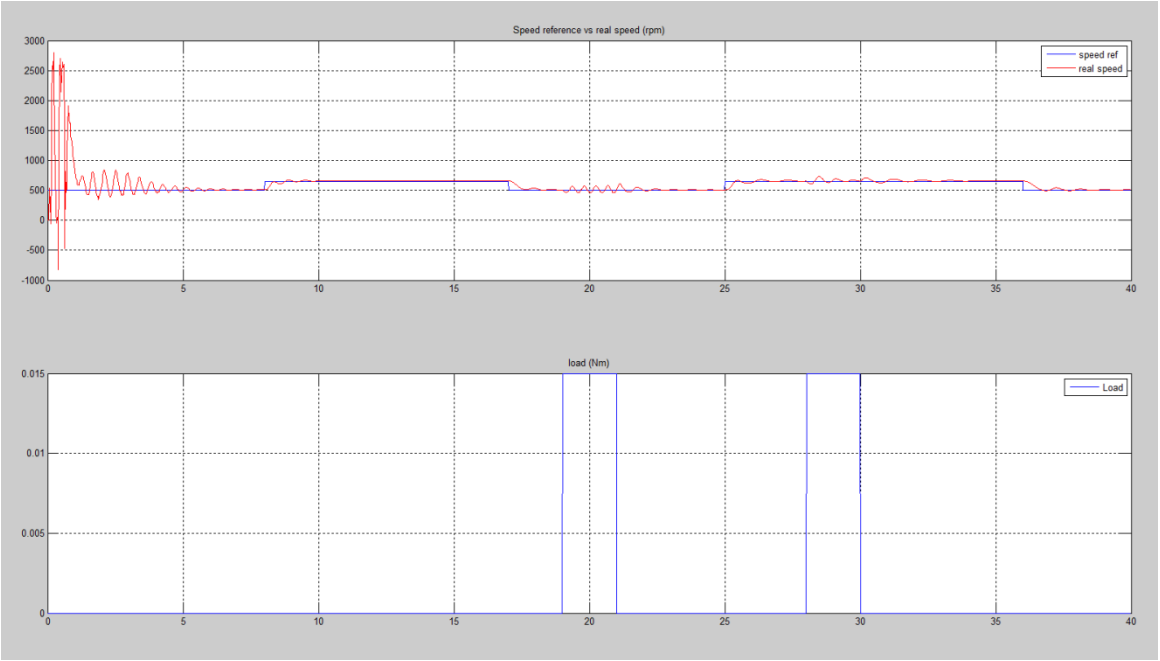


Figure 40: Real speed with its reference and applying load

For these figures the beforehand calculated values for the parameters CP and Lamba are used to drive the controllers. To refresh this information, these values were settled before after a series of proves simulating the two layer controller driving the FOC and the motor model. They were 25 and 10 for CP and Lambda respectively in the current control layer, also 35 and 10 for CP and Lambda as well for the speed control.

j. Implementation of the model in the real motor

From the figures above shown can be observed that there are relatively high fluctuations of current and speed at the beginning of the simulation. These when running the program code in the hardware can be a problem introducing this fluctuating current in the motor.

To try to damp those fluctuations with some strategies in the simulation model is the best solution. The fluctuations occur because of the mismatches between the real motor and the simple motor model created for the algorithm; this is not accurate enough for not having fluctuations when the control is enabled. These strategies could consist of adapting the model before the control begins to run, so that the motor model will have similar behaviour as the real one when the controller is set.

Another possible solution could be a very precise off-line training, therefore the behaviour of the real motor would be better known and the differences between model and real motor would be far less than those shown in this work.

8. HARDWARE IMPLEMENTATION

a. Application in real time with a real motor

In this part all the hardware used for the project work will be presented, as well as the tools needed for the implementation of the code.

The hardware part, as it is depicted below, consists of:

- A PMSM motor
- An inverter to drive the motor in which the pulse signals for the semiconductors are introduced
- A board to manage al the analog and digital signals which go to the inverter and also those which come from the current measurement
- This current measurement is done by Hall effect sensors, integrated in the board.
- A handmade board which is used to drive the cables coming from the last board to the I/O ports of the MicroAutoBox.
- The MicroAutoBox [4] acting as the “brain” of the operation. It is a very powerful tool which can achieve high speeds and data management at a very high rate. This is powered with 12V and linked to the computed through an ethernet cable in order to be able to communicate in real time with the user.
- And also two power supplies, one acting as the DC link of 12V in a car for the application and other also of 12V to supply power to the boards and to the MicroAutoBox.

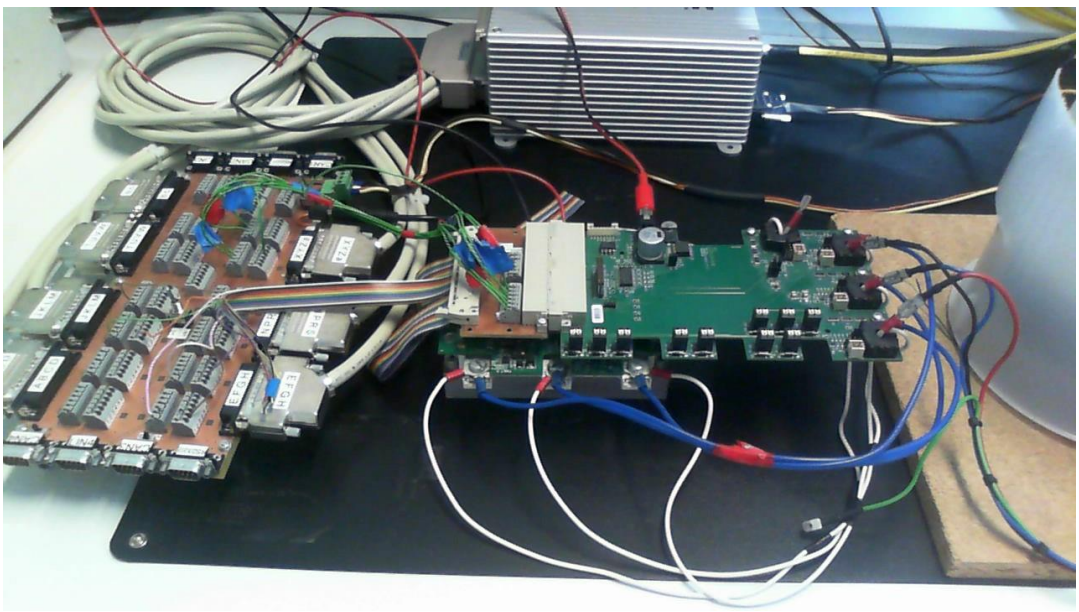


Figure 41: Communication I/O board and signals and current measurements board

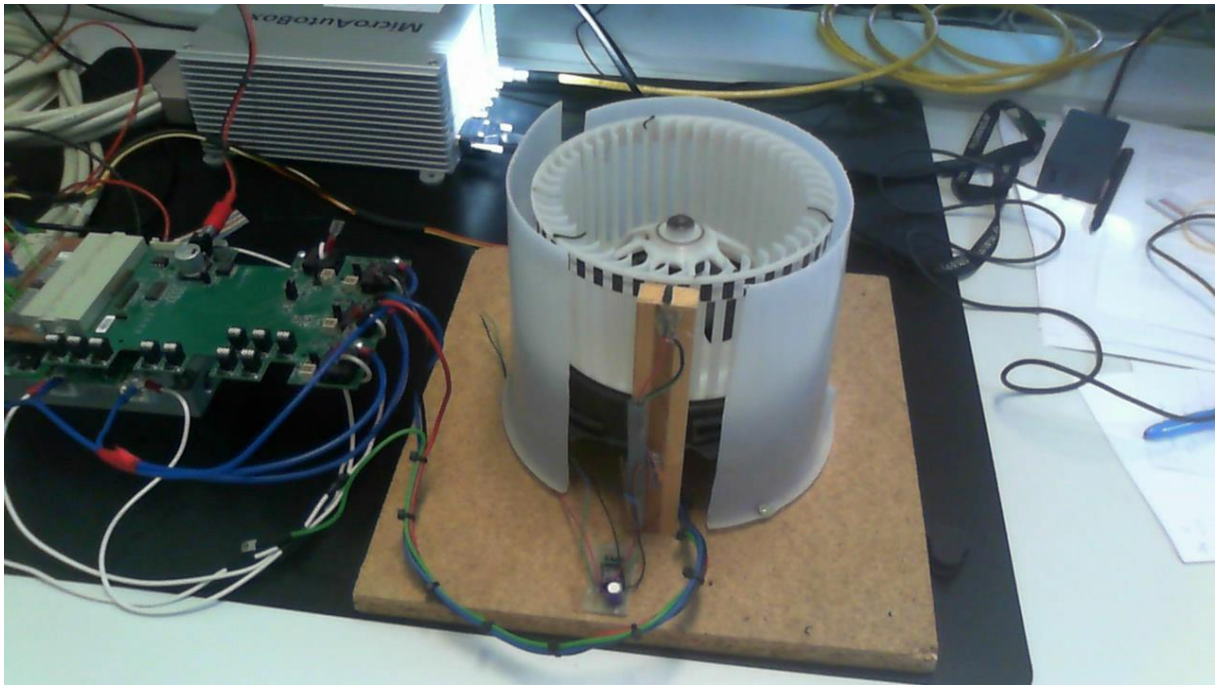


Figure 42: PMSM acting as a fan

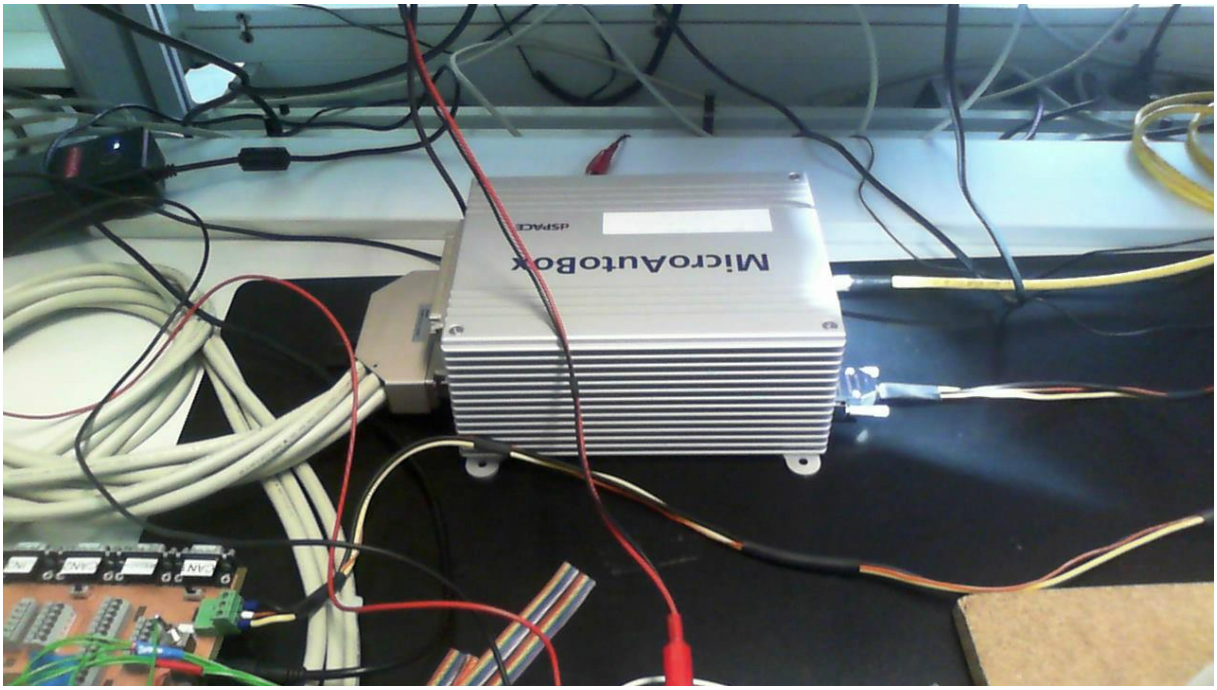


Figure 43: The CPU of the application, the MicroAutoBox

b. Introduction to dSpace and the MicroAutobox

One of the most important issues of this work is the real time communication between the user and the MicroAutoBox, this link will allow the user to change parameters of the code while running the program and get the measurements from the motor in real time.

Here it will be now presented the specific software of the MicroAutoBox [4], which is called dSpace. It is an excellent environment to interact with the MicroAutoBox. It recognizes the hardware through the ethernet cable and once the IP is set the connection is quite reliable and fast as well. This tool has also the possibility to connect with the matlab environment, so that in matlab the user has access to dSpace libraries which, in this work, are used to write and read data from the I/O ports.

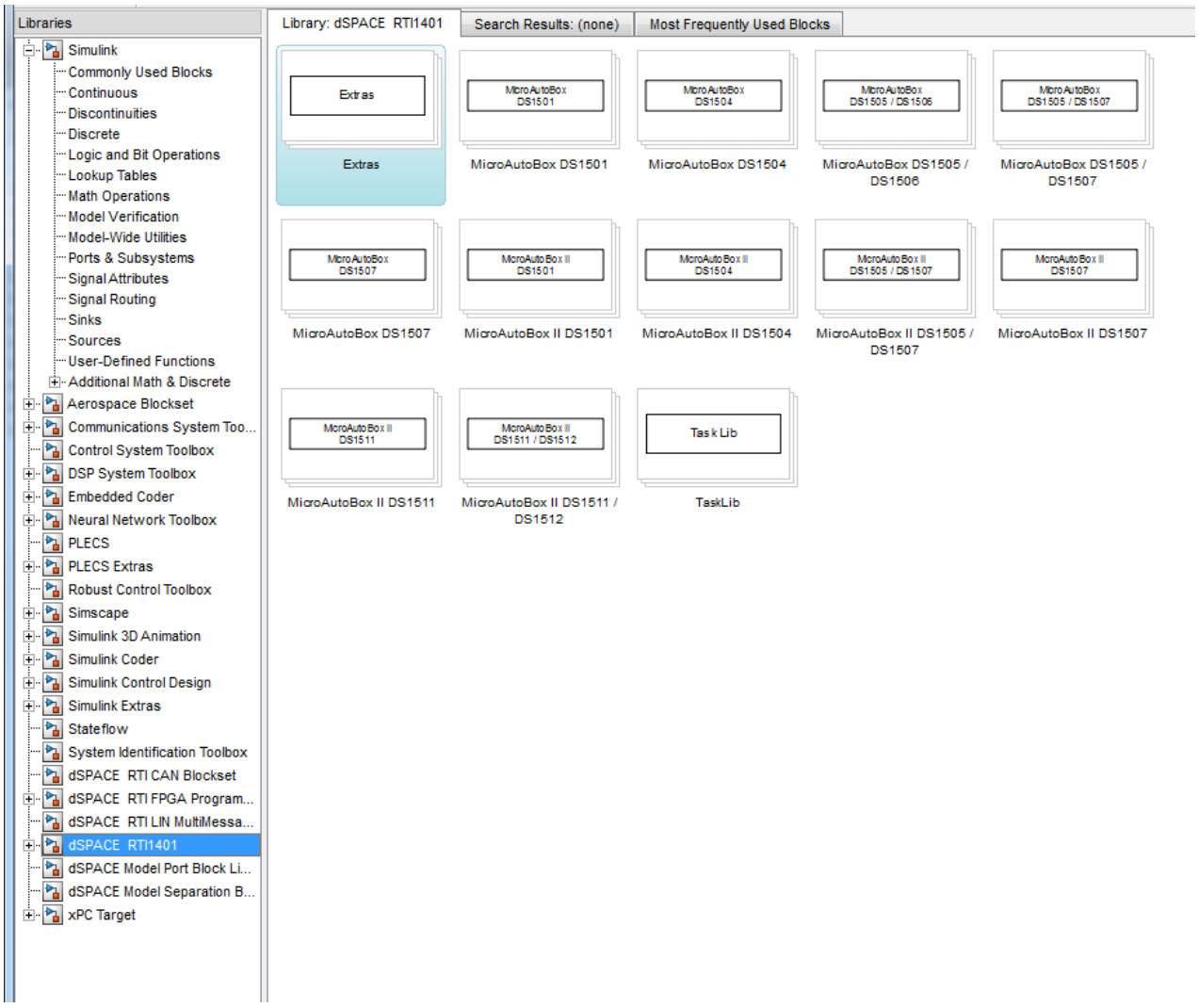


Figure 44: Libraries of dSPACE to be used in Matlab

Here an example of how the library blocks are used, in this case to read information from the current and voltages measurements.

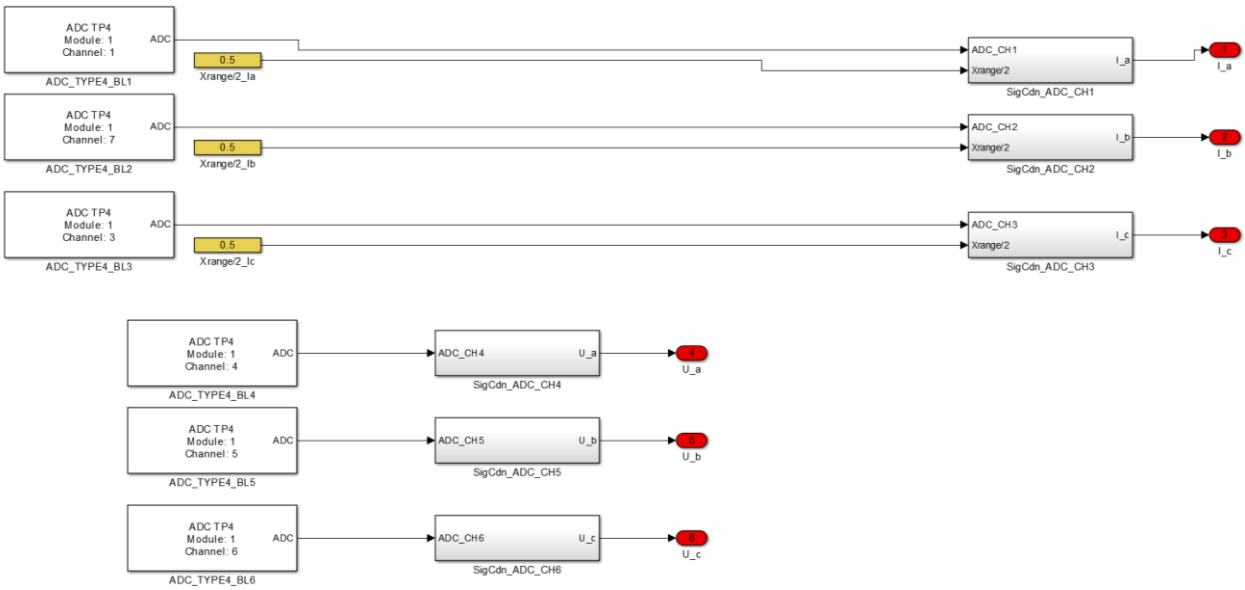


Figure 45: Current and voltage measurement from the hardware in the model

After a short presentation of the tools, now the process of charging the code into the MicroAutoBox is described.

The code, as it is already known, is implemented in matlab and for the transference of code the Real Time Workshop is needed.

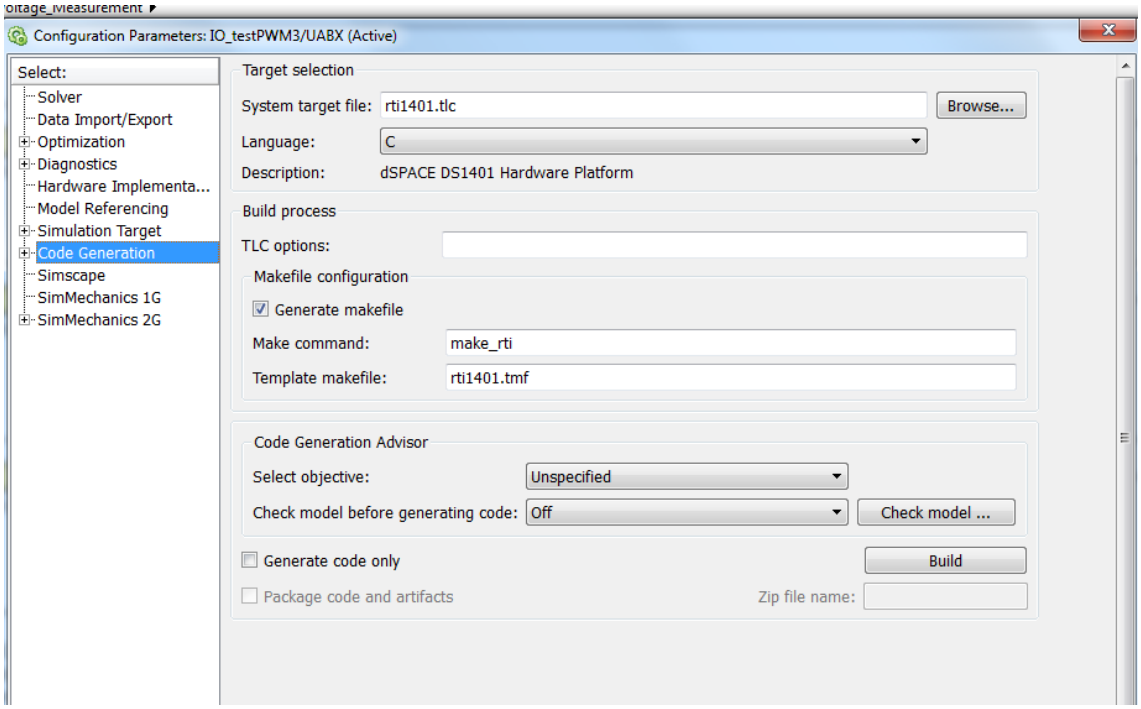


Figure 46: Settings of the RTW in the matlab model

It can be observed that the used files belong to the model of the MicroAutoBox which is used for the application, these files are basically for the translation of the code from the matlab environment to a language that the “brain” can understand. In order to use these files and to ensure a correct work of the RTW, matlab has to be linked with dSpace.

Once the code or model in Simulink is ready to be executed, as first task for transferring the data is building the model. In other words complation and linking processes are carried out when the model is built. For this the matlab RTW is used. When the code is correctly written with no mistakes and the linking with the MicroAutoBox is successful, a green light will appear in the tool showing its status.

From this point on the program is already running in the MicroAutoBox and is just dSpace the one which can interact with it and change parameters in the code.

Here some figures are shown in order to observe how does the dSpace environment looks like.

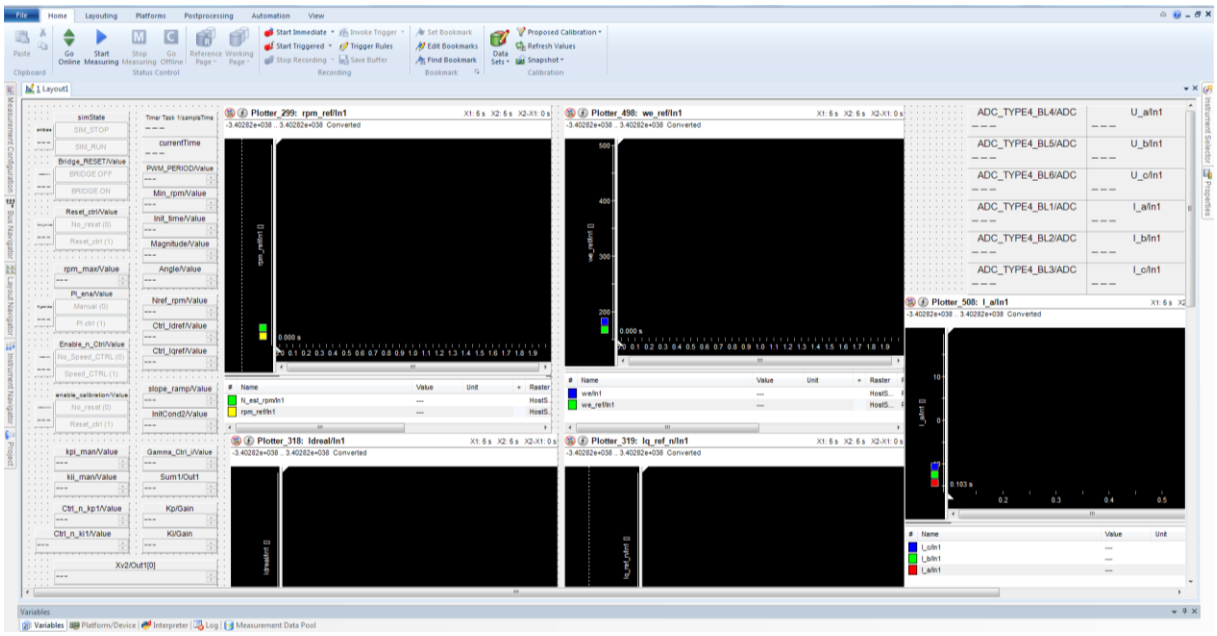


Figure 47: dSpace in one of the projects created

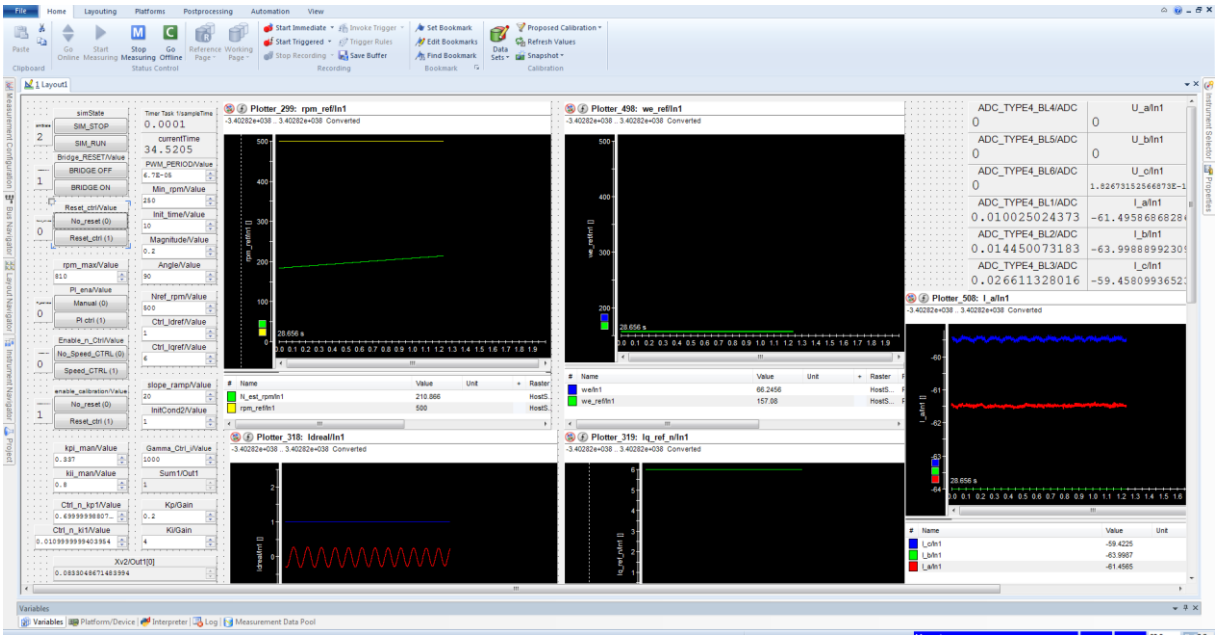


Figure 48: dSpace online with the MicroAutoBox

The only action which needs to be done to be fully connected and to receive data from the MicroAutoBox is to set the dSpace online (connected with the MicroAutoBox) and to allow data measurement; these options are on the above part in the dSpace screen.

Once the status is online, the user can fully interact with the hardware accesing variables from the code programmed and being able to change them in every moment. In order to a have access to all variables which were used in the program there is a .sdf file created im the

building process in which all these are stored in order to be able to access them afterwards in the dSpace environment.

c. Projects

A project in dSpace has to be created in order to run an application and several controls and plots for all the variables and constants were set so that with all the information needed in the screen such as currents, rotor position or speed, the user is able to see how the system is operating and can react according to the situation.

9. BUDGET

In order to present what the dimensions of this project would have been, economically speaking, the total money hypothetically spent within all the different parts of this work is presented as follows.

Hardware		
- PMSM 400W with fan component		320 €
- Inverter		125 €
- Power supply		55 €
- Boards, cables and sensors		25 €
- MicroAutoBox (software included in the tool)		12.000 €
- Desk computer. Intel Dual Core. For 8 months		600 € for 5 years
	Total hardware	**12.605 €**
Software		
- Matlab 2012b individual license		2.000 €
- PLECS individual license		7.000 €
	Total software	**9.000 €**
Human resources		
- Programmer 3h./day during 8 months		2,30 € per hour
- Computer expert. Setting up the computer. 5 days 3 hours a day		30 € per hour
	Total human resources	**1.830 €**
	Total project	**23.435 €**

10. CONCLUSIONS

This project was for me at first a really exciting challenge for two different reasons: one because it has taken place in a foreign country and two because it was about the topic that I liked most therefore I am pleased that I gained much knowledge for the future and also for myself.

When I first started the thesis I thought to myself that I knew much about electric motors and had also good experience with Matlab. After fighting with the first steps of the project for a period of time I realise that the reality was not as I had figured. Therefore having the opportunity to work in this topic has given me a great experience overall and also a huge push into modelling and motor control.

Another issue to remark is the language used to write and develop the project. I have never done such big work in a foreign language and I am now that I did manage to finish it.

To conclude I feel like I achieved much knowledge during the development of this work and that I have learnt many issues for my still academic or working life.

11. BIBLIOGRAPHY

[1] MSc Raul Estrada Vázquez, professor in FH Joanneum, ***"Online Energy Management Strategy for a Conventional Vehicle with Dual Power Net.*** *An adaptive predictive approach"*.

[2] Document from Raul Estrada Vázquez and Alfred Steinhuber, professors in FH Joanneum. Presentation of ***"Sensorless-Field-Oriented-Control for PMSM Air Ventilating Motor"***.

[3] Dr. Hubert Berger from FH Joanneum, Notes about **Automotive Electric Drives**

[4] Manuals provided for the management of ControlDesk and the MicroAutoBox

[5] Zambada, Jorge (Nov 8, 2007). ***"Field-oriented control for motors"***. MachineDesign.com.

[6] Fabio Genduso, Rosario Miceli, Member IEEE, Cosimo Rando, and Giuseppe Ricco Galluzzo, **"*Back EMF Sensorless-Control Algorithm for High-Dynamic Performance PMSM*"**, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 57, NO. 6, JUNE 2010

[7] Mung T. Tham (2002), ***"Introduction to Robust Control"***

[8] Pavel Vaclavek and Petr Blaha. ***"PMSM Model Discretization for Model Predictive Control Algorithms"*.** Proceedings of the 2013 IEEE/SICE International Symposium on System Integration, Kobe International Conference Center, Kobe, Japan, December 15-17.

[9] Saverio Bolognani, Silverio Bolognani, Luca Peretti and Mauro Zigliotto, ***"Design and Implementation of Model Predictive Control for Electrical Motor Drives"***, IEEE Transactions on Industrial Electronics, VOL. 56, NO. 6, JUNE 2009.

[10] Park, Inverse Park and Clarke, Inverse Clarke Transformations MSS Software Implementations

12. ANEXES

S-function code. Source: Raul Estrada Vázquez, MSc from FH Joanneum

```
function apc1x1_fcn(block)
% Adaptive predictive controller
%   Copyright 2011 FH JOANNEUM GmbH
%   Author: Raul Estrada Vazquez
%   $Revision: 1 $

  setup(block);

%endfunction

function setup(block)

  block.NumDialogPrms  = 2;
  block.DialogPrmsTunable = {'Tunable','Tunable'};

  %% Register number of input and output ports
  block.NumInputPorts  = 2;
  block.NumOutputPorts = 4;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

  block.InputPort(1).Complexity   = 'Real';
  block.InputPort(1).DataTypeId   = 0;
  block.InputPort(1).SamplingMode = 'Sample';
  block.InputPort(1).Dimensions   = 1;

  block.InputPort(2).Complexity   = 'Real';
  block.InputPort(2).DataTypeId   = 0;
  block.InputPort(2).SamplingMode = 'Sample';
  block.InputPort(2).Dimensions   = 1;

  block.OutputPort(1).Complexity   = 'Real';
  block.OutputPort(1).DataTypeId   = 0;
  block.OutputPort(1).SamplingMode = 'Sample';
  block.OutputPort(1).Dimensions   = 1;

  block.OutputPort(2).Complexity   = 'Real';
  block.OutputPort(2).DataTypeId   = 0;
  block.OutputPort(2).SamplingMode = 'Sample';
  block.OutputPort(2).Dimensions   = 1;

  block.OutputPort(3).Complexity   = 'Real';
  block.OutputPort(3).DataTypeId   = 0;
  block.OutputPort(3).SamplingMode = 'Sample';
```

```
  block.OutputPort(3).Dimensions   = 1;

  block.OutputPort(4).Complexity   = 'Real';
  block.OutputPort(4).DataTypeId   = 0;
  block.OutputPort(4).SamplingMode = 'Sample';
  block.OutputPort(4).Dimensions   = 4;


  %% Set block sample time to inherited
  block.SampleTimes = [0.1 0];

  %% Register methods
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('Outputs',                 @Output);
  %block.RegBlockMethod('Update',                  @Update);

%endfunction


function DoPostPropSetup(block)

  %% Setup Dwork
  block.NumDworks = 8;
  block.Dwork(1).Name = 'OP';
  block.Dwork(1).Dimensions      = 3;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;

  block.Dwork(2).Name = 'PV';
  block.Dwork(2).Dimensions      = 3;
  block.Dwork(2).DatatypeID      = 0;
  block.Dwork(2).Complexity      = 'Real';
  block.Dwork(2).UsedAsDiscState = true;

  block.Dwork(3).Name = 'SP';
  block.Dwork(3).Dimensions      = 2;
  block.Dwork(3).DatatypeID      = 0;
  block.Dwork(3).Complexity      = 'Real';
  block.Dwork(3).UsedAsDiscState = true;

  block.Dwork(4).Name = 'Yp';
  block.Dwork(4).Dimensions      = 1;
  block.Dwork(4).DatatypeID      = 0;
  block.Dwork(4).Complexity      = 'Real';
  block.Dwork(4).UsedAsDiscState = true;

  block.Dwork(5).Name = 'Yd';
  block.Dwork(5).Dimensions      = 2;
  block.Dwork(5).DatatypeID      = 0;
  block.Dwork(5).Complexity      = 'Real';
  block.Dwork(5).UsedAsDiscState = true;

  block.Dwork(6).Name = 'Ap';
  block.Dwork(6).Dimensions      = 4;
  block.Dwork(6).DatatypeID      = 0;
  block.Dwork(6).Complexity      = 'Real';
  block.Dwork(6).UsedAsDiscState = true;
```

```
  block.Dwork(7).Name = 'DO';
  block.Dwork(7).Dimensions      = 3;
  block.Dwork(7).DatatypeID      = 0;
  block.Dwork(7).Complexity      = 'Real';
  block.Dwork(7).UsedAsDiscState = true;

  block.Dwork(8).Name = 'counter';
  block.Dwork(8).Dimensions      = 1;
  block.Dwork(8).DatatypeID      = 0;
  block.Dwork(8).Complexity      = 'Real';
  block.Dwork(8).UsedAsDiscState = true;

  %endfunction

function InitConditions(block)

  %% Initialize Dwork
    block.Dwork(1).Data = [0 0 0];
    block.Dwork(2).Data = [0 0 0];
    block.Dwork(3).Data = [0 0];
    block.Dwork(4).Data = 0;
    block.Dwork(5).Data = [0 0];
    block.Dwork(6).Data = [1 -0.2 0.1 0.1];%[1 -2 1 1];
    block.Dwork(7).Data = [0 0 0];
    block.Dwork(8).Data = block.DialogPrm(1).Data;
%endfunction

function Output(block)
    %% Decreasing control period counter
    block.Dwork(8).Data = block.Dwork(8).Data - 1;

    %% New controller output is calculated only at the control period or
    %%  when set-point changes
    if           block.Dwork(8).Data           ==           0           ||
block.InputPort(1).Data~=block.Dwork(3).Data(2)
        d.Lambda=block.DialogPrm(2).Data;       % Prediction horizon
        d.AM = 1;                               % Adaptive coeffient
        d.dp = 0;                                 % Number of delay periods
for b-parameters
        d.an = 2;                                  % Number of a-parameters
(denominator-->poles)
        d.bn = 2;                                  % Number of b-parameters
(numerator-->zeros)
        d.Ad  =  [0.7358  -0.1353  0.2642  0.1353];    %  2nd  order  reference
trajectory (approx 9 CPs to get SP)
        d.opl   = -50;                                    % Lower  limit  for
controller output
        d.opu   =   50;                                   % Upper  limit  for
controller output
        d.opil =  10;                                  % Incremental limit for
controller output

    %% Reading data of previous iteraction
        d.OP = block.Dwork(1).Data;
        d.PV = block.Dwork(2).Data;
        d.SP = block.Dwork(3).Data;
        d.Yp = block.Dwork(4).Data;
        d.Yd = block.Dwork(5).Data;
        d.Ap = block.Dwork(6).Data;
```

```
        d.DO = block.Dwork(7).Data;

    %% Reading input ports
        d.SP(1) = block.InputPort(1).Data;
        d.PV(1) = block.InputPort(2).Data;

    %% Preparation for incremental form
        ET=zeros(d.Lambda,d.an+1);
        for ren=1:d.Lambda
            for col=1:d.an
                if ren==1
                    ET(ren,col)=d.Ap(col);
                else
                    ET(ren,col)=ET(ren-1,1)*d.Ap(col)+ET(ren-1,col+1);
                end
            end
        end
        E=ET(:,1:d.an);

        GT=zeros(d.Lambda,d.bn+1);
        for ren=1:d.Lambda
            for col=1:d.bn
                if ren==1
                    GT(ren,col)=d.Ap(d.an+col);
                else
                    GT(ren,col)=ET(ren-1,1)*d.Ap(d.an+col)+GT(ren-1,col+1);
                end
            end
        end
        G=GT(:,2:d.bn);
        hs=sum(GT);
        h=hs(1);

        if d.Lambda==1
            sumE=E;
            sumG=G;
        else
            sumE=sum(E);
            sumG=sum(G);
        end

        for col=1:d.an
            DYk(col)=d.PV(1+col-1)-d.PV(1+col);
        end
        for col=1:d.bn-1
            d.DOk(col)=d.DO(1+col);
        end

    %%  Reference trajectory
        R=d.Ad(1)*d.Yd(1)+d.Ad(2)*d.Yd(2)+d.Ad(3)*d.SP(1)+d.Ad(4)*d.SP(2);

    %%  Control law
        d.DO(1)=(R-d.PV(1)-sumE*DYk'-sumG*d.DOk')/h;
        d.OP(1)=d.OP(2)+d.DO(1);

    %%  Apply limits to controller output
        if abs(d.DO(1))>d.opil
            d.OP(1)=d.OP(2)+sign(d.DO(1))*d.opil;    %Applying    incremental
limit
        end
```

```
        if d.OP(1)>d.opu
            d.OP(1)=d.opu;          % Applying upper limit
        end
        if d.OP(1)<d.opl
            d.OP(1)=d.opl;          %Applying lower limit
        end

        d.DO(1)=d.OP(1)-d.OP(2);     % Recalculating increment of controller
output

    %% Adaptive mechanism
        err=d.PV(1)-d.Yp;           % Prediction error
        Xp=[d.PV(2);d.PV(3);d.OP(2+d.dp);d.OP(3+d.dp)];  % Allocating state
vector
        B=eye(d.an+d.bn)*d.AM;           %  B  matrix  weighed  by  adaptive
coefficient

        %if abs(err)>0 && abs(err)<=100
           d.Ap=(B*Xp/(1+Xp'*B*Xp))*err+d.Ap;    %    Adapting    predictive
parameters
        %end
    %% Calculate predictive process value for next iteraction
        Xp=[d.PV(1);d.PV(2);d.OP(1+d.dp);d.OP(2+d.dp)];
        d.Yp=d.Ap'*Xp;

    %% Store data for next iteraction
        block.Dwork(1).Data = [0 d.OP(1) block.Dwork(1).Data(2)];
        block.Dwork(2).Data       =       [0       block.InputPort(2).Data
block.Dwork(2).Data(2)];
        block.Dwork(3).Data = [0 block.InputPort(1).Data];
        block.Dwork(4).Data = d.Yp;
        block.Dwork(5).Data = [R block.Dwork(5).Data(1)];
        block.Dwork(6).Data = d.Ap;
        block.Dwork(7).Data = [0 d.DO(1) block.Dwork(7).Data(2)];
        block.Dwork(8).Data = block.DialogPrm(1).Data;


    %% Write actual values to output ports
        block.OutputPort(1).Data = d.OP(1);                    %Controller
output
        block.OutputPort(2).Data = R;                           %Reference
process value
        block.OutputPort(3).Data = d.Yp;                       %Predictive
process value
        block.OutputPort(4).Data = d.Ap;                           %Actual
adaptive parameters

    else
    %% Keep previous values at output ports
        block.OutputPort(1).Data  =  block.Dwork(1).Data(2);    %Controller
output
        block.OutputPort(2).Data  =  block.Dwork(5).Data(1);     %Reference
process value
        block.OutputPort(3).Data  =  block.Dwork(4).Data;       %Predictive
process value
        block.OutputPort(4).Data  =  block.Dwork(6).Data;           %Actual
adaptive parameters

    end
```

```
%endfunction
```

Matlab function code.

```
function
[controller_outp,reference,predictive_outp,adaptative_param,feedback_matrix
] = apc1x1_fcn_ctrl_n(r,y,iq_max,CP,L,mat_input)

%% internal variables


OP = mat_input(1,1:3);
PV = mat_input(2,1:3);
SP = mat_input(3,1:2);
Yp = mat_input(4,1);
Yd = mat_input(5,1:2);
Ap = mat_input(6,1:4);
DO = mat_input(7,1:3);
counter = mat_input(8,1);



%% initializing the outputs of the function
Outp = 0; %controller output
ref = 0; %desired input
pred_out = 0; %predictive output
ad_param = zeros(1,4); %adaptive parameters
mat_past_data = zeros(8,4); %matrix got from feedback containing tha
variables from last iteration

%% function which calculates the outputs
% [OP,ref,pred_out,ad_param,mat_past_data]=output(strg);
%
[OP,ref,pred_out,ad_param,mat_past_data]=output(inputs,mat_input,int_variab
les);
[Outp,ref,pred_out,ad_param,mat_past_data]=output(r,y,iq_max,CP,L,OP,PV,SP,
Yp,Yd,Ap,DO,counter,mat_input);

%% outputs

controller_outp = Outp;
reference = ref;
predictive_outp = pred_out;
adaptative_param = ad_param;
feedback_matrix = mat_past_data;




%% funtion to calculate the output of the controller
function [Outp,ref,pred_out,ad_param,mat_past_data] = output
(r,y,iq_max,CP,L,OP,PV,SP,Yp,Yd,Ap,DO,counter,mat_input)
% Lambda = L
Lambda = 3; % when avoiding this line it produces an unexpected error
initial_ad_params = [1 -0.2 0.1 0.1]; % initial adaptive parameters
```

```
%% Establishing initial conditions for counter and adapt parameters
if Ap(1) == 0 && Ap(2) == 0 && Ap(3) == 0 && Ap(4) == 0 % if its all 0 we
suppose we are in the first iteration so
    Ap = initial_ad_params; % initial conditions are settled
end
% initialize the counter to CP when it reaches 0
if counter == 0
    counter = CP;
end
counter = counter - 1;


%% process of adaption and prediction
if counter == 0 || r~=SP(2)

    AM = 1;
    dp = 0;
    an = 2;
    bn = 2;
    Ad = [0.7358 -0.1353 0.2642 0.1353];
    opl  = -iq_max;                              % Lower limit for controller
output
    opu  =  iq_max;                              % Upper limit for controller
output
    opil =  10;



    %% Reading input ports
    SP(1) = r;
    PV(1) = y;

    %% Preparation for incremental form
    ET=zeros(Lambda,an+1);
    for ren=1:Lambda
        for col=1:an
            if ren==1
                ET(ren,col)=Ap(col);
            else
                ET(ren,col)=ET(ren-1,1)*Ap(col)+ET(ren-1,col+1);
            end
        end
    end
    E=ET(:,1:an);

    GT=zeros(Lambda,bn+1);
    for ren=1:Lambda
        for col=1:bn
            if ren==1
                GT(ren,col)=Ap(an+col);
            else
                GT(ren,col)=ET(ren-1,1)*Ap(an+col)+GT(ren-1,col+1);
            end
        end
    end
    G=GT(:,2:bn);
```

```
hs=sum(GT);
h=hs(1);

if Lambda==1
    sumE=E;
    sumG=G;
else
    sumE=sum(E);
    sumG=sum(G);
end

DYk = zeros(1,an);
DOk = zeros(1,bn-1);

for col=1:an
    DYk(col)=PV(1+col-1)-PV(1+col);
end
for col=1:bn-1
    DOk(col)=DO(1+col);
end


%%  Reference trajectory
R=Ad(1)*Yd(1)+Ad(2)*Yd(2)+Ad(3)*SP(1)+Ad(4)*SP(2);

%%  Control law
DO(1)=(R-PV(1)-sumE*DYk'-sumG*DOk')/h;
OP(1)=OP(2)+DO(1);

%%  Apply limits to controller output
if abs(DO(1))>opil
    OP(1)=OP(2)+sign(DO(1))*opil; %Applying incremental limit
end
if OP(1)>opu
    OP(1)=opu;          % Applying upper limit
end
if OP(1)<opl
    OP(1)=opl;          %Applying lower limit
end

DO(1)=OP(1)-OP(2);    % Recalculating increment of controller output

%% Adaptive mechanism
err=PV(1)-Yp;           % Prediction error
Xp=[PV(2);PV(3);OP(2+dp);OP(3+dp)]; % Allocating state vector
B=eye(an+bn)*AM;      % B matrix weighed by adaptive coefficient

Ap_aux=(B*Xp/(1+Xp'*B*Xp))*err+Ap'; % Adapting predictive parameters
Ap=Ap_aux';

%% Calculate predictive process value for next iteraction
Xp=[PV(1);PV(2);OP(1+dp);OP(2+dp)];
Yp=Ap*Xp;


%% Write actual values to output ports
Outp=OP(1);           %Controller output
ref=R;                %Reference process value
pred_out=Yp;          %Predictive process value
```

```
    ad_param=Ap;          %Actual adaptive parameters

    %% Store data for next iteraction
    mat_past_data = zeros(8,4);

    mat_past_data(1,:) = [0 OP(1) OP(2) 0];
    mat_past_data(2,:) = [0 y PV(2) 0];
    mat_past_data(3,:) = [0 r 0 0];
    mat_past_data(4,:) = [Yp 0 0 0];
    mat_past_data(5,:) = [R Yd(1) 0 0];
    mat_past_data(6,:) = Ap;
    mat_past_data(7,:) = [0 DO(1) DO(2) 0];
    mat_past_data(8,:) = [counter 0 0 0];
    %matrix to save last values

else
    %% Keep previous values at output ports
    Outp = OP(2);  %Controller output
    ref = Yd(1);  %Reference process value
    pred_out = Yp;      %Predictive process value
    ad_param = Ap;      %Actual adaptive parameters
        %% Store data for next iteraction
    mat_past_data = zeros(8,4);
    mat_past_data = mat_input;
    mat_past_data(8,1) = counter; %actualize the counter in each iteration


end
```