POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Meccanica

Tesi di Laurea Magistrale

Development and comparison of prognostic methodologies applied to electromechanical servosystems (EMA) for aerospace purposes

Relatori

prof. Paolo Maggiore

Ing. Matteo Davide Lorenzo Dalla Vedova

Candidato

Stefano Re

Ottobre 2018

Thanks to…

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore od imprecisione contenuto in questa tesi.

Anzitutto ringrazio sentitamente l'ing. Dalla Vedova e il prof. Maggiore, relatori, i quali hanno mi hanno guidato con esperienza e simpatia lungo il tortuoso percorso di ricerca e speso parte del loro tempo a revisionare le bozze: senza il loro supporto e la loro guida questa tesi non avrebbe visto la luce.

Ringrazio anche Pier Carlo Berri per il supporto che mi ha dato con Matlab e per gli innumerevoli consigli di cui spero di aver fatto tesoro in modo efficace.

Vorrei dedicare questo lavoro ad alcune persone a me molto care:

A papà Alberto e mamma Giovanna, i cui sacrifici spero di aver ripagato nel modo migliore possibile, sempre pronti a sostenermi nei momenti complicati.

A tutti gli amici, vicini e lontani, nuovi o di lunga data, che in questi anni mi hanno dimostrato il loro affetto. In particolare, ci tengo a ringraziare Andrea e Lorenzo, fantastici compagni di avventura e amici leali con i quali ho diviso esperienze fantastiche, e Giorgia, il cui affetto mi ha permesso di superare le avversità dell'esperienza torinese.

Ma soprattutto dedico questo lavoro a me stesso, che ho lavorato duramente per ottenere ciò che mi ero prefissato, senza cercare scusanti ai fallimenti che ottenevo.

Abstract

Over the past few decades, mostly in commercial and defence markets, there has been the will to implement always new approaches and strategies to carry out prognosis on complex systems, in order to achieve benefits in terms of reliability and safety of the product, planning maintenance and logistic costs. In this work, a model-based prognostics approach is proposed: it has been applied to the study of fault appearance in flight controls moved by an electromechanical servosystem (EMA). The faults taken into consideration are friction, backlash, short circuit, rotor eccentricity and gain: they are deeply described and implemented in two different models. The first model is the high-fidelity one and it represents the motor to analyse; the second one is the monitor model, lighter and quicker then the previous one, which has to approximate as best as it can the parameters of the first one. To simulate the reference model, four different optimization algorithms are applied to the monitor: Genetic Algorithm (GA), Differential Evolution (DE), Particle Swarm Optimization (PSO) and Greywolf Optimization (GWO). Their features and their capability to solve this problem are deeply investigated and compared each other; particular attention is paid to percentual error between the optimizations and the reference values and to computational time.

CONTENTS

1. Introduction and overview

1.1. Introduction to prognostics

Over the past few decades, mostly in commercial and defence markets, there has been the will to implement always new approaches and strategies to carry out prognosis on complex systems, in order to achieve benefits in terms of reliability and safety of the product, planning maintenance and logistic costs. Prognostics is the discipline which evaluates the current state of a system or single component and estimates the Remaining Useful Life (RUL), namely how much time will pass until the object of the study will no longer able to operate within its stated specification [2]. Prognostics is exploited in a huge range of applications including automotive, robotics, automation and naval purposes for its extremely reliable behaviour: in this work it's applied to an electromechanical actuator which commands a secondary flight control. This discipline is composed by several mandatory steps:

1. *Data collection*: the system or the machinery studied is equipped with sensors which acquires the temporary trend of a variable as displacement, speed, torque or current. Next, is important to exclude the sensors which gave back unreliable trends caused by noise or excessive vibrations.
2. *Clean up of the trends*: in every sensor's records is stored also the noise and vibrations which could affect the real behaviour of the system; in this phase, using analogic or digital filters, signals are cleaned up.
3. *Thresholds*: with a filtered signal, it's mandatory to determine thresholds directly from data already acquired (history) or from requirements.
4. *Prediction of RUL*: this is the last phase and the most "aleatoric" one. Taking advantage from algorithms, neural networks, mathematical regressions and statistics, a prediction of RUL is carried out; in this way it's possible to plan next maintenance stops. This prediction could be made studying the future operating working conditions as input commands, environment or loads, analysing the failure modes with FMEA or FMECA models, detecting every indication of failure in our

system as wear, cracks or aging and correlating them with the experience or mathematical expressions.

The increasing of reliability and safety of the products, and decreasing the logistic time, costs and unnecessary services are the main aims of prognostics. This discipline could be divided into some different categories [5]:

- Type I, reliability data-based. These models are based on statistical considerations and exploit historical fault data to estimate the Remaining Useful Life (RUL) mainly for parts used in nominal conditions;
- Type II, stress-based. Learning from accumulated knowledge, here a fault growth model considers all the environmental stresses as external loads, temperature, noise, vibrations etc;
- Type III, condition-based. These models take care also about specific conditions in which a component is working and hence its failure conditions, which can be used to plan a maintenance.

Nowadays the implementation of prognostic methods is managed by an approach which compares the complexity of the system and the benefits in terms of availability, cost savings, maintenance scheduling. For components not essential in the working-life of a machine or for the cheapest ones is not convenient to do such a study.

Prognostics is different from the diagnosis, because the latter indicates the condition of the system after a break, it represents something already happened (e.g. a break-down of the turning screw); the former is nothing but a *prediction* of the possible future working behaviour of the system based on the actual conditions.

In this work a model-based prognostics approach is proposed: it will be applied to the study of fault appearance in flight controls. These are the most critical system mounted on aircraft, so they are usually designed with the “safe life” philosophy: every part or component of these systems must be replaced with a new one after a determined amount of flight hours. The aim of this thesis is to suggest a reliable option to plan maintenance actions just in the moment where the system needs it, comparing some different algorithms and their response.

1.2. Flight controls

In last decades, flight control systems have shown a great evolution thanks to improvements done in aircraft technologies [4]. In the first planes there was a fly-by-wire technology, in which pilot's commands in the cockpit were connected directly to the control surfaces by means of thick wires. This control method remained very used for many years; when the technologies was approaching the supersonic region of flight, the necessity of more complex and reliable control systems returned back at the top of the designers' priority list [7]. Furthermore, the increasing load capacity of planes makes inadequate a system moved only by pilot's strength: in this period hydraulically-powered and pneumatically-powered actuators become widespread. These systems saw a brilliant evolution throughout the years, until they become one of the most studied parts of the aircraft; nevertheless, their increasing complexity and their interactions during flight may reduce the reliability and efficiency of the whole plane. A small leakage in pneumatic or hydraulic hardware could provoke wrong functionality of all networks, causing issues, delays or dramatic consequences. This leads to a concept introduced years ago, during World War II: the "all-electric" aircraft [6].

In the figure 1.1 below (taken from [21]), the flight controls are represented and named.

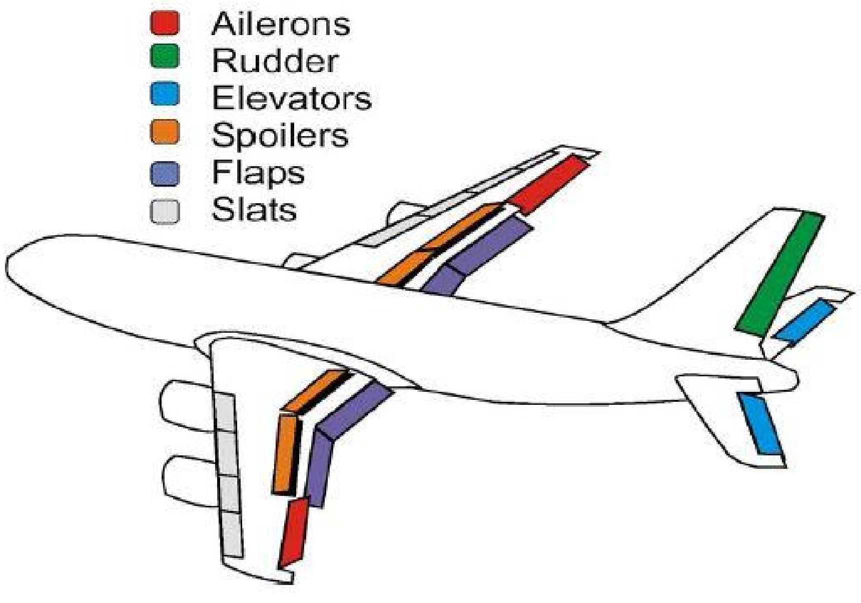

Figure 1. 1 - Name of the main flight controls

All aircrafts, from the simplest to the most complex one, use quite the same principles of flight controls. In order to modify its position in the space, an aircraft needs to move some specific control surfaces dislocated in different places (wings, fuselage, stabilizers…), which allow to exploit the force of the air to make the plane turn around one of its three main axis or to modify its altitude.

All surfaces represented above allow the aircraft to do rotational or translational movements around a set of pre-determined axis. To modify the altitude, an aircraft exploits the lift force, generated by the air passing through the wings, to overcome the weight of the structure, people and cargo; the other translational force (directed in the main direction of the flight) is generated by the propulsion system, it's called thrust and it needs to be greater than the drag force, which is simply the resistance of the air acting against the motion of the plane [8]. In the figure 1.2 taken from [8] these four forces are clearly shown.

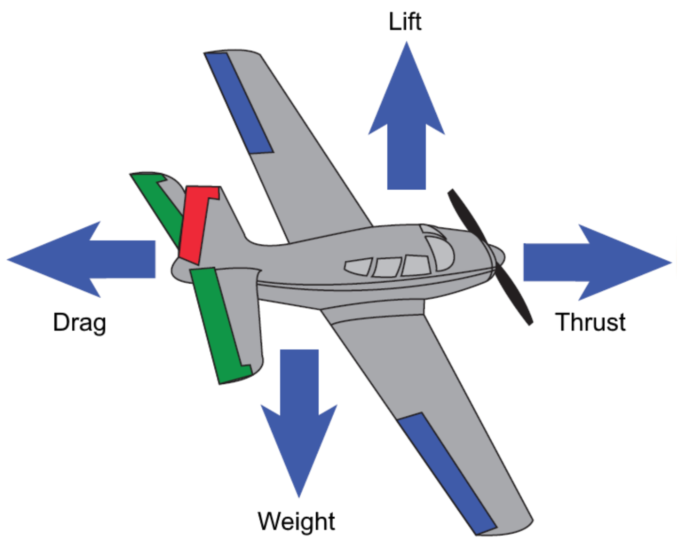

Figure 1. 2 - Forces acting on the plane

Flight controls could be divided into two categories: primary and secondary flight controls.

1.2.1. Primary flight controls

Primary flight controls are systems which could carry out rotational displacements around the three main body axis: the motions are called *pitch*, *roll* and *yaw*, and they are represented in figure 1.3 [8].

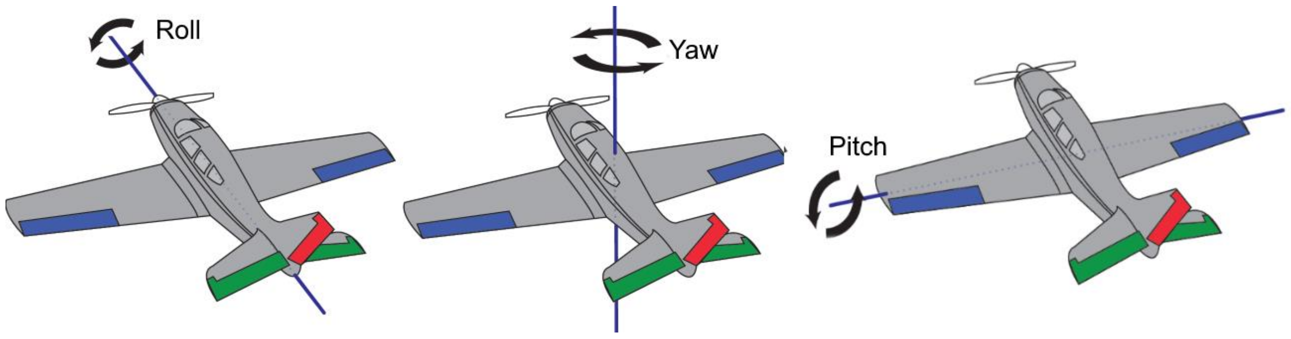

Figure 1. 3 - Name of the rotational displacements

The three surfaces which could execute these manoeuvres are the ailerons (represented in the figure 3 in blue), the rudder (shown in red) and the elevators (the green ones). The movement of any of these three primary flight control surfaces varies the pressure distribution around and over the airfoil. During a normal flight, actuators connected with them are continuously powered and checked: these systems are usually full of redundancies and high-frequency feedback rings (force feedback is the most common used), in order to have always under control their functionality. Furthermore, they need to compensate the external disturbances, so their action is vital.

To modify the aircraft heading and altitude, primary control surfaces generate unbalanced torques and forces which are able to modify aircraft position. To change plane's heading, usually the pilot needs to execute a roll modifying the *ailerons*' inclination. These surfaces are normally two, they are situated at the rear of each wing and work in opposition to each other: when one is raised, the other is lowered, in order to decrease the lift force on one wing (aileron raised) and increase it on the other (aileron lowered). They are mainly used in fixed-wing aircraft to turn.

The yaw is carried out by the *rudder*, situated on the tail of the aircraft, precisely on the vertical stabilizer. Both roll and yaw allow the aircraft to change heading in the same way: the main difference is that the former involves the coordination of two axis whilst the latter only one. During the turning operation made by roll, a component of the lift is directed toward the centre of rotation, so the real effect of this force decreases; this situation provokes the loss of altitude of the aircraft. The pilot needs to help the ailerons with the rudder, and at the same time he needs to raise the nose of the aircraft. To do that, two surfaces called *elevators* are installed at the end of the aircraft, on the horizontal stabilizers. The alteration of their angular position acts like a lifter (here is the origin of their name) and allows to increase or decrease the altitude of the aircraft. Furthermore, they have also a “support” purpose, because usually the cockpit or generally the front side of a plane is heavier than the rest: elevators generates a downward force on the tail which compensates this unbalanced situation.

Primary flight controls could be reversible or powered. The former relies on a direct linkage made up by rod or steel cables between the movable surfaces and commands in the cockpit, hence the pilot need to act on the levers or pedals with a force able to counteract the air flow. The latter exploit hydraulic and electric actuation systems to compensate a great amount (or, in some cases, almost entire effort) of air force: they are widely used when the loads or the speed of the aircraft do not allow the first solution. In this last case, the pilot needs only to give the input signal and the command is transferred to the surfaces with a fly-by-wire or fly-by-light system.

Some of the main control parts have been generally described above, but in a flight control system there are a lot of more surfaces which enable the pilot to have a suitable control over all aircraft, also during landing and take-off. Their design takes care about advanced aerodynamics and systems dynamics and it could sharply vary from an aircraft to another, depending on the purpose (commercial or military), performance requirements (maximum speed and payload, agility, etc.) and dimensions. These other control surfaces are the secondary flight controls.

1.2.2. Secondary flight controls

Secondary flight controls include several added surfaces to make easier and more efficient the control of the aircraft for the pilot. For further clearness, it’s necessary to introduce deeply this type of control surfaces.

Commonly used secondary flight control surfaces are [9]:

- *Flaps*, which are the most common lift devices installed on an aircraft: their role is to induce drag and lift for any angle of attack of the wing. These surfaces are located alongside the wings and could be raised or retracted inside the wing's structure depending on the need. There are four types of flaps, represented in figure 1.4 (in grey is the section of the wing, in orange the flap), taken from [9].

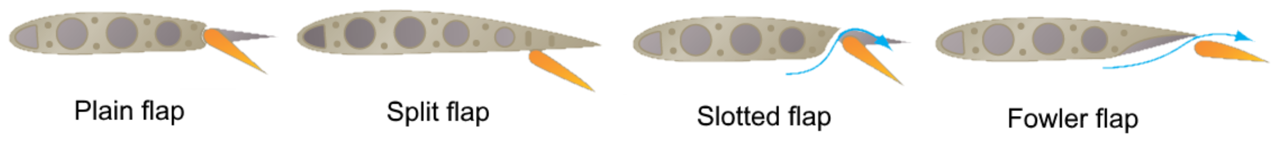

Figure 1. 4 - Main types of flaps used nowadays

  Plain flap is the oldest and the simplest between the four shown above: it changes the camber of the wing, generating a quick increase of the lift coefficient (usually represented with C_L notation), and also introduces a drag component and a movement of the centre of pressure back towards the tail, which allow the aircraft to perform a pitch rotation (nose downward).

  The second type of flap is the *split* one: it is installed under the wing, allowing a great increase of the drag component due to turbulent flow of the air and a slightly lower increase of lift compared with the plain flap.

  Surely, the most spread and used secondary flight control surface on aircraft nowadays is the *slotted* flap, which increases the lift coefficient C_L sharply and more than the previous two types of surfaces discussed. When this flap is lowered, a small conduit widens between the flap's and wing's edges: this opening admits the passage of highly-energized air, which could act on larger surfaces compared with plain and split flap, generating a superior lift force. Furthermore, the duct allows the increase of drag force reducing strongly the creation of harmful vortices.

  A particular type of slotted flap is the *Fowler* flap, which does not rotate on a hinge, but slides on tracks. This surface has the possibility to generate a very harmful pitching moment (nose up or down), which could be counteracted with a trim adjustment.

- *Slats* are the leading-edge device located on the front side of each wing. They could be classified as fixed slots, movable slats, leading-edge flaps and leading-edge cuffs.

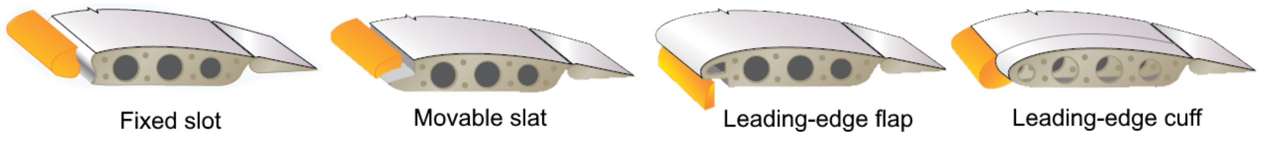

Figure 1. 5 - Types of slats

The main aim of these surfaces is to delay stall until the wing reaches a right angle of attack: directing the airflow to the upper wing surface, the separation of the airflow is strongly retarded. Usually these devices are pilot-operated and can be used at any angle of attack.

Fixed and *movable* slots are very similar, but formers are usually hinged and cannot be dismounted, latters move on tracks and could be replaced.

Leading edge flaps are commonly used with flaps because they can counteract the nose-down pitching moment generated those surfaces. A small rising movement of leading edge flaps increases sharply the lift force applied to the aircraft; when this device is extended, drag component become bigger quicker than the lift.

Leading edge cuffs increase the maximum value of the lift coefficient C_L like leading edge flaps, but the formers are a fixed device. These surfaces could move forward and down the leading edge, decreasing aircraft's stall speed (airflow could adhere better to the upper surface of the wing).

- Other control surfaces are the *spoilers,* whose aim is to decrease lift and to increase drag force. They are located alongside the wing like the flaps, but have a different use: they allow the pilot to have a perfect control over the aircraft during landing. Special application of the spoiler could be in roll manoeuvre to counteract the adverse yaw torque. Deploying the entire spoiler structures permits to decrease altitude without gaining speed.

With flaps, slats and spoilers, other types of secondary flight control surfaces are trim systems, balance tabs, ground adjustable tabs, servo and anti-servo tabs and adjustable stabilizers.

An ON/OFF actuator usually powers secondary flight surfaces, and their control law does not require an excessively high frequency rate: a position or at least a speed control ring are installed. Actuators are located in the centre of the plane and the motion is transmitted to the surfaces in order to achieve the best symmetry possible; only devices which could work asymmetrically are spoilers.

1.3. Actuation systems

To drive the control surfaces, usually several servomechanisms are exploited. They receive the input commands from the cockpit, compare them with the actual situation of the system (position, speed, force, torque, temperature, pressure or electrical magnitude) and consequently act. Actuators requirements are robustness, reliability and safety and main solutions are described below.

1.3.1. Hydromechanical actuation

The simplest hydromechanical system is composed by a reservoir, a pump, a filter to maintain the fluid clean, a selector valve to control the direction of the flow, an actuator and a relief valve to drain the excessive pressure in the circuit. They could be classified [11]:

- Position hydraulic servo-system (with or without mechanical feedback)
- Speed hydraulic servo-system
- Force hydraulic servo-system

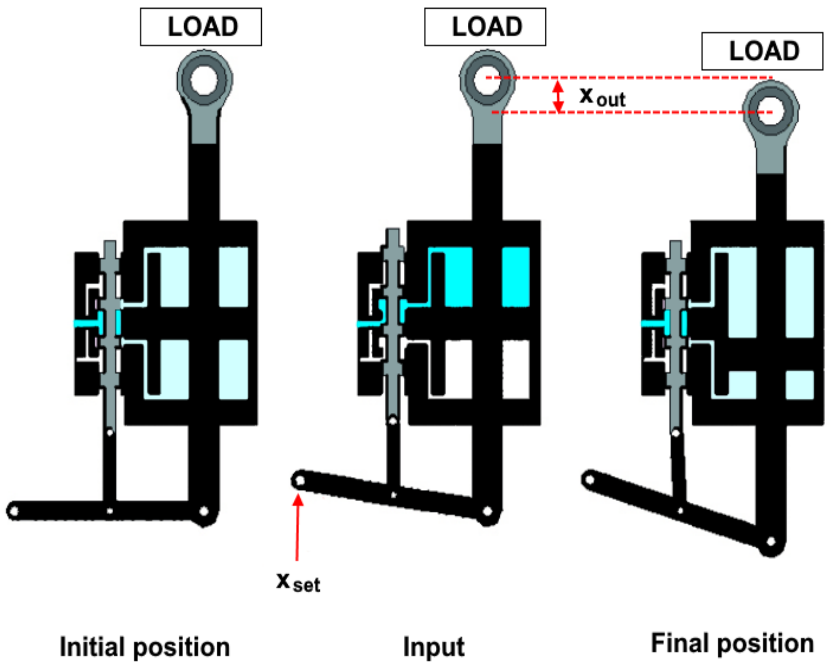

Figure 1. 6 - Typical hydromechanical actuator

An effective type of hydromechanical actuator with mechanical feedback is shown in figure 1.6. This system is composed by a hydraulic cylinder and a proportional control valve. A perturbation on the three-pivot control lever x_{set} causes the movement of the valve's spool: in this way one chamber of the cylinder and a tank (or generically the supplier) are connected. Fluid flows in the superior chamber of the cylinder, moving downwards the actuator's shaft and generating the displacement x_{out}. This solution is reversible: changing the direction of the input, hence the direction of the spool motion, actuator's shaft will move upwards. At the initial and final position, when the input and output variable are steady, the valve closes all possible holes and maintains blocked the servosystem.

A system is really reliable if is also insensible to an eventual disturbance; for a position control the disturbance is represented by a force. Hence, if the shaft has to sustain a heavier load, the valve's spool opens a duct and supplies the needed pressurized flow to bear it: in this way the system has also a disturbance rejection.

This type of servomechanism is widely used due to its simplicity and reliability, it can be found on several operational aircraft such as F-15 Eagle of Boeing 737.

1.3.2. Electrohydraulic actuation

Nowadays, the most used actuation system on aircraft is the electrohydraulic one. It's very similar to hydromechanical, but the mechanical feedback is replaced by a fly-by-wire structure. Here the simple valve shown in figure 1.6 is replaced by a more complex servovalve: this could be flapper-nozzle type (figure 1.7) or jet pipe type (figure 1.8).

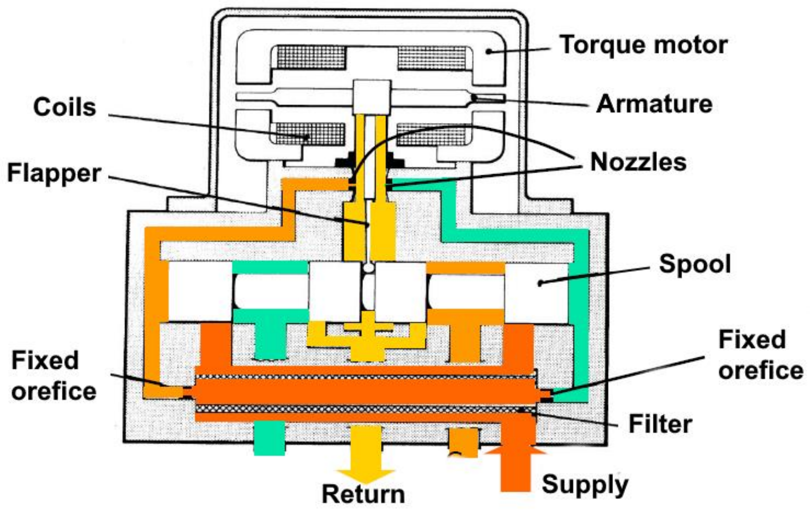

Figure 1. 7 - Flapper-nozzle servovalve

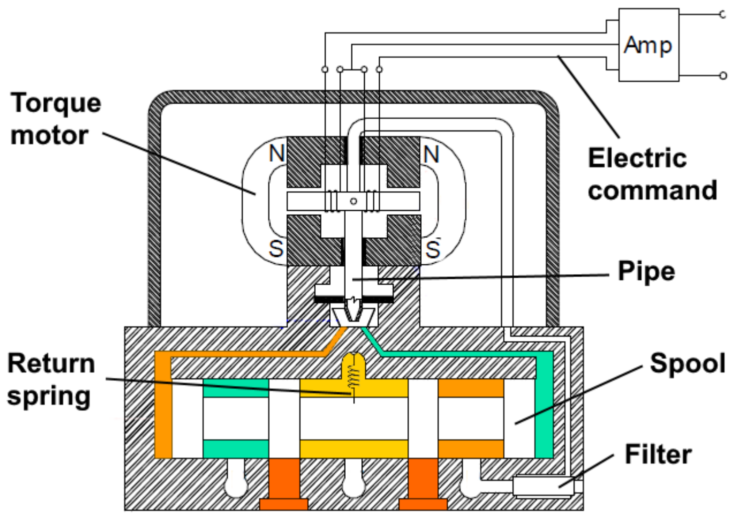

Figure 1. 8 - Jet-pipe servovalve

A servovalve is different from the proportional valve seen before because in this case there is an electrical component, called torque motor, which transmits the command instead of a mechanical lever. In the flapper-nozzle servovalve the torque motor induces a magnetic field that could rotate a "T" shaped component, bonded at the end of the long shaft to the valve's spool. Hence this structure approaches a nozzle and leaves from the other: in this way the amount of fluid that flows from the ducts, so also the pressure, changes. In the jet-pipe servovalve the functioning is the same, but here the torque motor modifies the heading of an elastic spout from which the pressurized fluid flows [11].

In these two types of electrohydraulic systems there is not a direct feedback between the control valve and the actuator: the absolute position of the piston is surveyed by a Linear Variable Differential Transducer (LVDT) and sent to the control electronics.

1.3.3. Electro-hydrostatic actuation (EHA)

In recent years, the will to make flights cheaper and greener have driven the aerospace industry to develop more electrical actuation system to implement in Power-by-wire networks. Two examples of this philosophy are electro-hydrostatic (EHAs) and electromechanical actuators (EMAs). Nowadays, in civil transportation, EHAs are widely used by Airbus (A380/A400M/A350 mainly), while EMAs are more utilized by Boeing (B787), both for first and secondary control systems.

Electro-hydrostatic actuators [12] are a power-by-wire type of motors that execute the movement of the control surfaces exploiting a localized hydraulic power obtained by the electrical power. These actuators could be divided in several groups according to their difference in control modes:

- Fixed Pump displacement and Variable Motor speed EHA (FPVM-EHA);
- Variable Pump displacement and Fixed Motor speed EHA (VPFM-EHA);
- Variable Pump displacement and Variable Motor speed EHA (VPVM-EHA).

In this paragraph are described only the FPVM-EHA because are the most popular thanks to their structural simplicity and high efficiency. A typical structure is shown in figure 1.9.

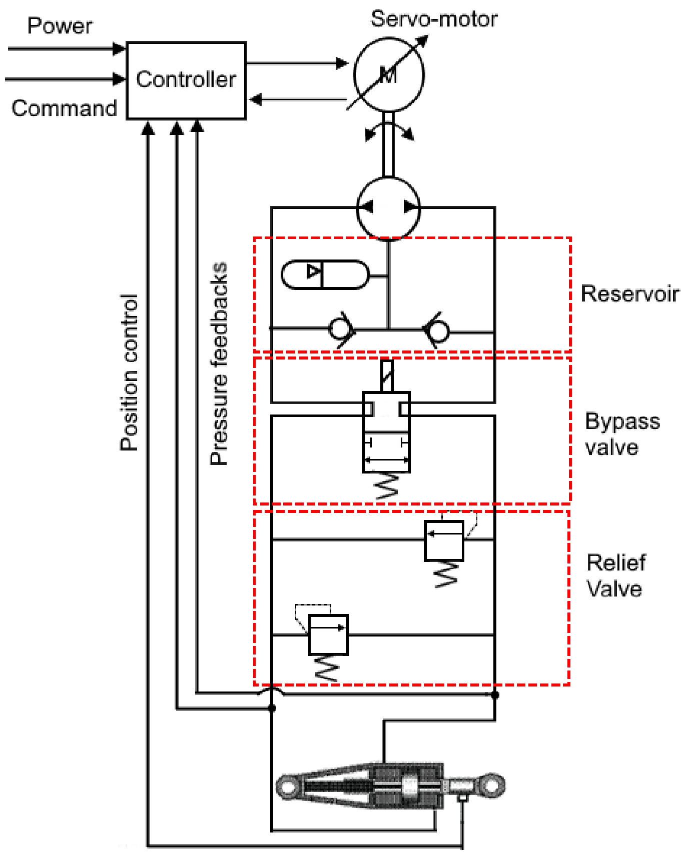

Figure 1. 9 - Typical FPVM-EHA simplified structure

A servomotor controlled by electronics drives a bi-directional pump with variable speed. The fluid in the circuit is stored in a proper reservoir, composed by a low-pressure tank and two check valves able to maintain the minimum pressure required by the system. Close to reservoir there is a bypass valve electronically controlled and two relief valves arranged for safety of the system. The pressurized oil in the circuit it's not provided by the on-board main hydraulic system, but the pump takes the needed fluid from the tank. For this type of actuator too, the feedback ring needs a position sensor placed on the shaft.

This is not the best structure if the aim is to decrease aircraft's weight, but it's very effective to reduce the critical issue related to a centralized hydraulic system. Furthermore, it allows maintenance and fuel cost savings while it assures the same precision and reliability of a traditional hydraulic system. The main issue of this solution is the low frequency width (about 5 Hz) which makes EHA not suitable for military purposes.

1.3.4. Electromechanical actuation

Electromechanical actuation (EMA) moves aircraft design toward the philosophy of "More Electric Aircraft", with great level of safety, efficiency, cost saving and a strong reduction of pollution. In EMAs the hydraulic circuit is entirely replaced by mechanical solutions, usually reducers (both gearbox or nut-screw types), which exclude the possibility of leakages: those issues are often difficult to detect and hard to fix due the complexity and the scarce accessibility of the fluid circuit.

A possible simplified EMA structure for aerospace purposes is represented in the figure below.

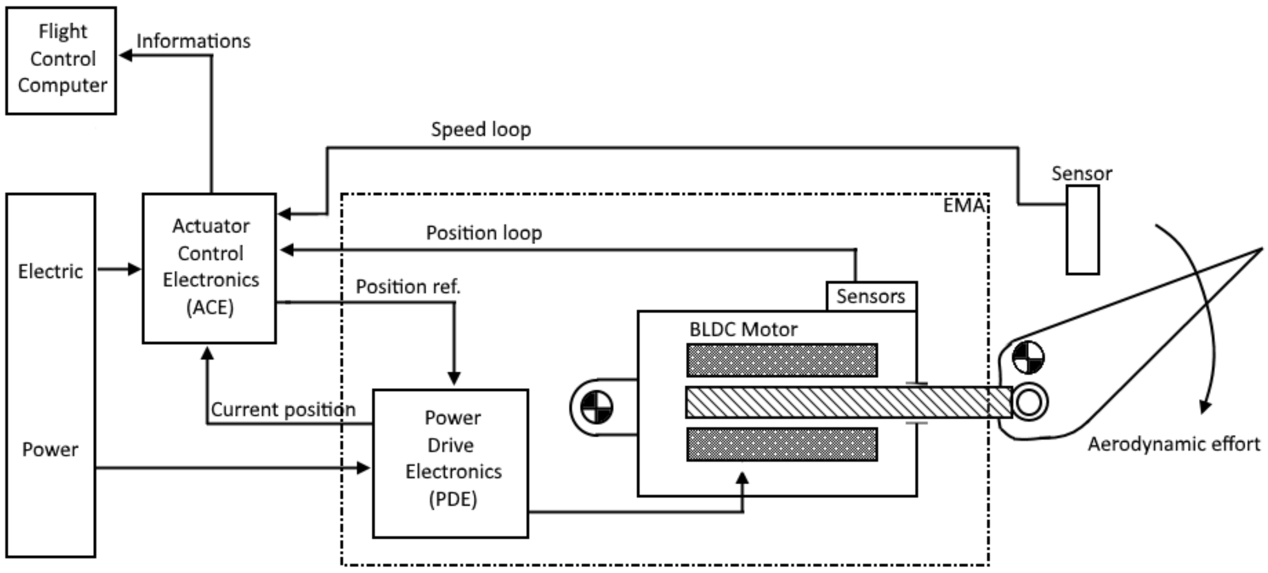

Figure 1. 10 - Essential scheme of EMA structure

The Actuator Control Electronics (ACE) is the main controller of the system and executes all calculations needed to maintain the error closest to zero; in this device arrive all the feedbacks collected by the sensors. Once calculated the current state of the end-effector through position and speed loops, ACE sends the reference position to the Power Drive Electronics (PDE). PDE it’s usually composed by a three-phases inverter bridge and its goal is to provide the correct power flow to electric motor in order to reach the reference position. The motor is usually a BLDC motor (Brushless powered by Direct Current): it transforms electrical input into mechanical rotational output for the reducer’s gears. For this task, in EMAs structure motors are usually very small, to reduce overall weight, and quick. To increase the torque and to reduce the speed transmitted a gears reducer is used, followed pretty always by a ball or roller screw (figure 1.11) which are able to transform a rotary motion into a translational one.

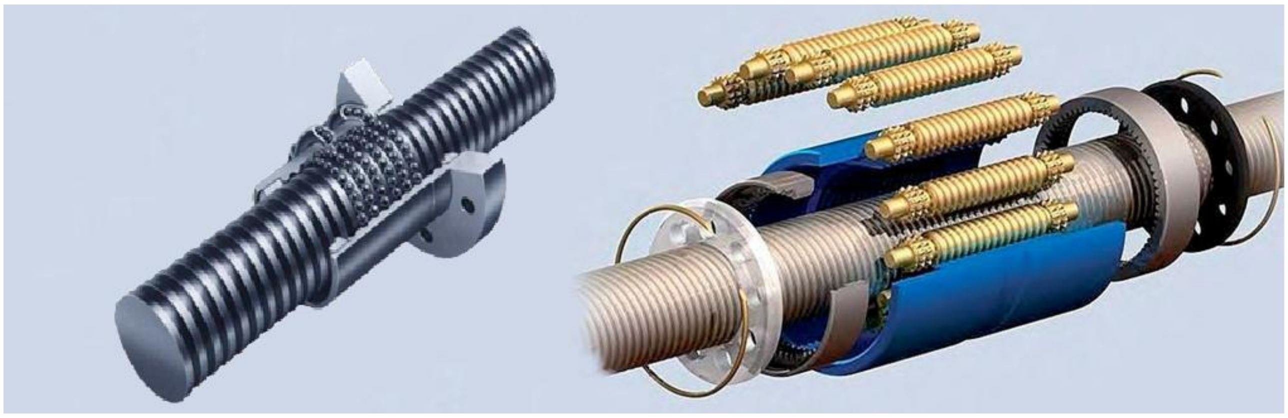
Figure 1. 11 - Ball screw (left) and roller screw (right)

These devices are composed by a rolling body (spheres for the ball screw and shaped cylinders for the roller screw) located between the rotating shaft and a nut. The friction is clearly very small, the loads could be very heavy, and it has a great resistance to wear; however there are some issues regarding the plays. The translation of these mechanisms allows the movement of the end-effector: in this simple example it is represented by a flight control surface on which is applied an aerodynamical effort. Another important part of the system are sensors: they are mandatory to detect the actual state of the system (position, speed, torque, force etc.).

Nowadays, electromechanical actuators are already used in military aircraft or in secondary flight controls; their application to primary flight control needs the improvement of some critical issues. Firstly, they have a very complex structure, which requires a deep study of kinematics and redundancy design; furthermore, to increase efficiency and avoid jam problems, a different BLDC motor is required for every surface, but in this way the possibility to cause a critical failure is increased. Another important problem is the possibility to maintain a determined position after the application of a heavier load: if the motor’s speed is near to zero and the torque is high, all the current given to stator is dissipated by joule effect. This issue could be overcome with an irreversible transmission.

2. Brushless Motor

Brushless Direct Current motors (also called BLDC motor or synchronous DC motors) are synchronous, direct current-powered and electronically-commutated actuators [14]. This type of motor is gaining popularity very quickly; nowadays is used in a wide range of applications such as automotive, aerospace, medical and industrial automation. In electromechanical system are the most used due to their long list of positive characteristics.

BLDC motors, as the name suggest, do not exploit the brushes to control the current inside, but they are electronically controlled. This feature allows these actuators to not suffer from wearing and particle inclusions issues; furthermore, during the normal activity the noise is strongly reduced. Other advantages over the brushed DC motor are the higher dynamic response and efficiency, longer operating life, superior speed and better ratio between the torque and the weight of the system: this last characteristic made BLDC motor very suitable for application in which space and weight are critical issues. This type of actuator is shown in figure 2.1.

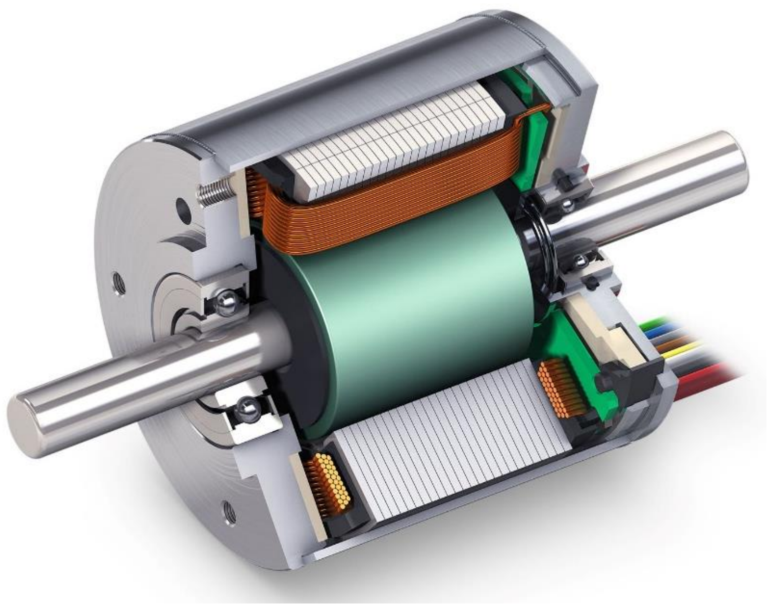

Figure 2. 1 - Cross section of a BLDC motor

Generally, brushless direct current motors are composed of an external stator in which is wounded a number of winding equal to the motor phases and an internal permanent magnet rotor; usually, the number of phases is three and the windings are star-connected. This type of actuator is synchronous, which means that the rotation frequency of the magnetic field of the stator and of the rotor is the same.

2.1. Stator

BLDC motor stator is made out of laminated steel stacked up to carry the windings and traditionally it resembles that of an induction AC motor; however, the distribution and the position of the windings in these two motors are different. Phase windings in a stator could be arranged in two different ways, star pattern (Y) or delta pattern (Δ): the main difference between these two configurations is the phase voltage. In fact, in delta configuration the phase voltage is equal to the line voltage, in star configuration is equal to $\frac{1}{\sqrt{3}}$ of the line voltage. Hence, it's clear that when the same voltage is applied, the delta pattern sees a higher current flow thus the torque (directly dependant to the current) is higher. However, in delta pattern all the windings must be always powered and the commutation order is different; for this reason, this last solution is used only in special applications.

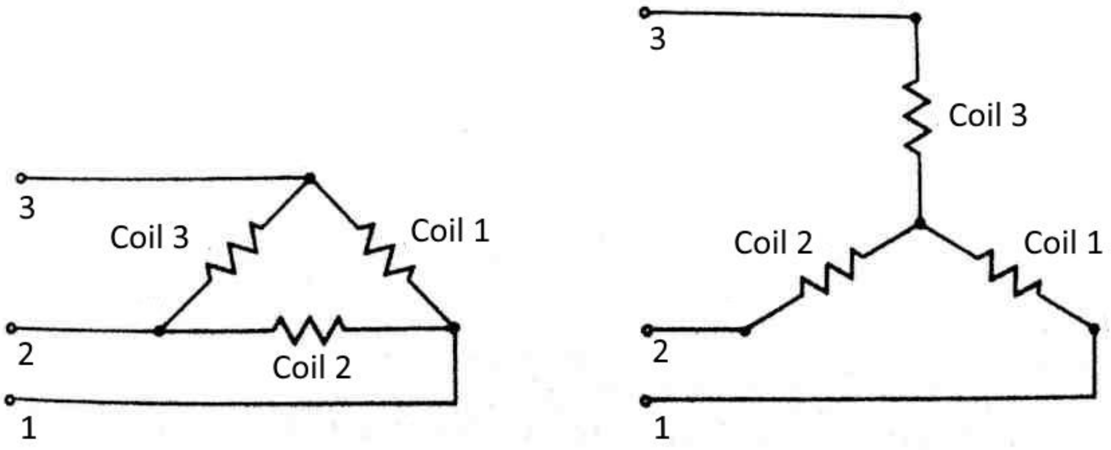

Figure 2. 2 - Δ configuration (at the right) and Y configuration (at the left)

Steel shape of the stator can be slotted or slotless as shown in figure 2.3. A slotless core has lower inductance, so it can rotate at very high speeds. Thanks to the absence of teeth in the lamination stack, requirements for the cogging torque also decrease, thus they are a suitable solution for low speeds too. The cogging torques are caused in slotted stator by the interactions between the stator teeth and the magnetic rotor: the difference of the air gap causes a variable reluctance, so it provokes ripples when the rotating speed is low. The main disadvantage of a slotless stator is a higher cost because it requires more copper winding to compensate the larger air gap.

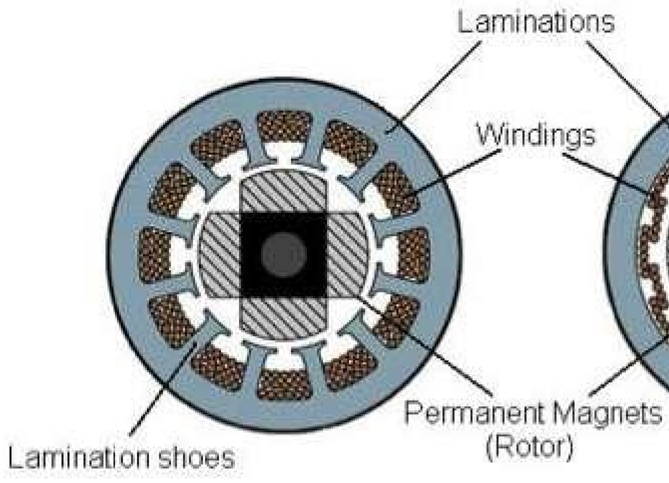

Figure 2. 3 - Slotted stator (at the left) and slotless stator (at the right)

To achieve the best motor performance the choice of the steel used for the stator is a critical issue: an inappropriate selection could cause problems during the normal working activity.

2.2. Rotor

The rotor is the rotating part of the motor and it’s made by permanent magnets, which could be arranged in different ways (in figure 2.4 some BLDC-motor rotor configurations are shown).

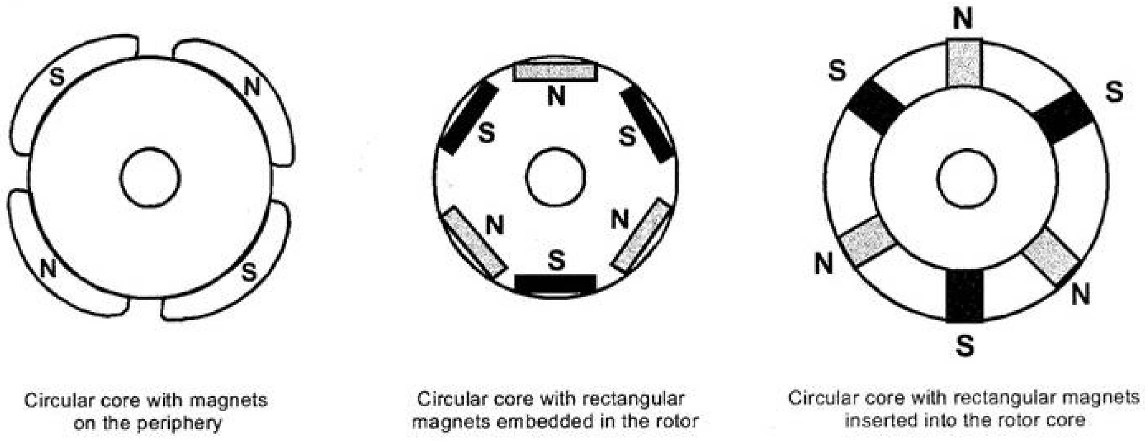
S
N
N
S
Circular core with magnets on the periphery
N
S
S
N
N
S
Circular core with rectangular magnets embedded in the rotor
N
S
S
N
N
S
Circular core with rectangular magnets inserted into the rotor core

Figure 2. 4 - Rotor magnet cross section

If the magnets are located outside the central cylinder, the rotor is called isotropic (the first example on the left in figure 2.4); if the magnet are in the internal part of the central cylinder the rotor is called anisotropic. In both configurations, particular attention must be given to the attachment between the rotor and magnets: during nominal working they rotate at several thousand of RPM, causing a strong centrifugal stress which could provoke the detachment of these two parts.

Depending on the application requirements, the number of poles and the materials of magnets may vary. Increasing the number of poles-pairs of the motor it’s possible to achieve a smoother torque delivery, but such a system could reach lower speeds due to maximum frequency of current commutation provided by the electronic control. The materials of the rotor magnets could be different depending on the magnetic field density required. Historically the first permanent magnets were made by ferrite but, as technology goes further, rare earth alloy magnets become more important and widely used thanks to their higher magnetic flux density for a given volume. Furthermore, this type of magnets improves the size-to-weight ratio: in this way it’s possible to provide a higher torque for the same size motor using ferrite magnets. Typical examples of rare earth alloy for magnets are Neodymium (Nd), Samarium Cobalt (SmCo) and Neodymium-Ferrite-Boron (NdFeB).

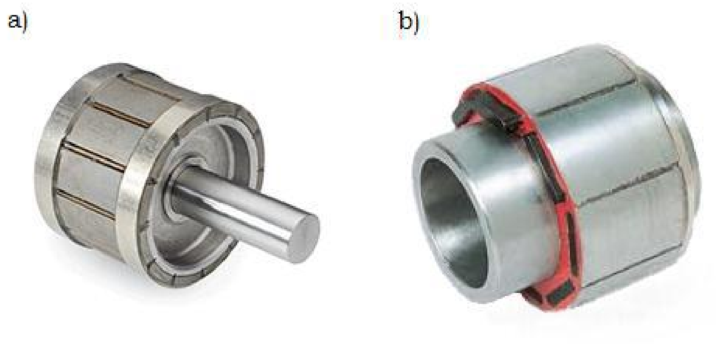

Figure 2. 5 - Isotropic (a) and anisotropic (b) rotors

2.3. Working principle

A brushless motor can be controlled by a square wave (brushless DC motor) or by a sine wave (brushless AC synchronous motor) depending on the waveform of the current provided by the controller and the back electromotive force trend given by the stator coil and rotor magnet disposition and shape. This work will focus on brushless DC motor which are the most used thanks to their control simplicity.

The torque on BLDC motors is provided by the interaction between the magnetic field generated by the current in the windings and the permanent magnet on the rotor. The highest torque is achieved ideally when this two field are at 90°: in order to keep the actuator rotating, the magnetic field produced by stator windings should shift from one winding to another, to allow the rotor field to follow it. In a three phases motor, each commutation sequence has one winding powered by an incoming current, one non-energized winding and one from which the current goes out (positive, neutral and negative power).

The commutation of a BLDC motor is electronically commanded, as already said. It implies that controller needs to know the position of the rotor to shift the power from one winding to another. Rotor position is obtained by sensors located all around the rotating body and they can catch its precise position time by time. These sensors exploit the Hall effect (they are also called Hall sensors): if a conductor which carries current is immersed in a magnetic field, this one applies a transverse force on the moving charges, generating a voltage in the conductor (figure 2.6).

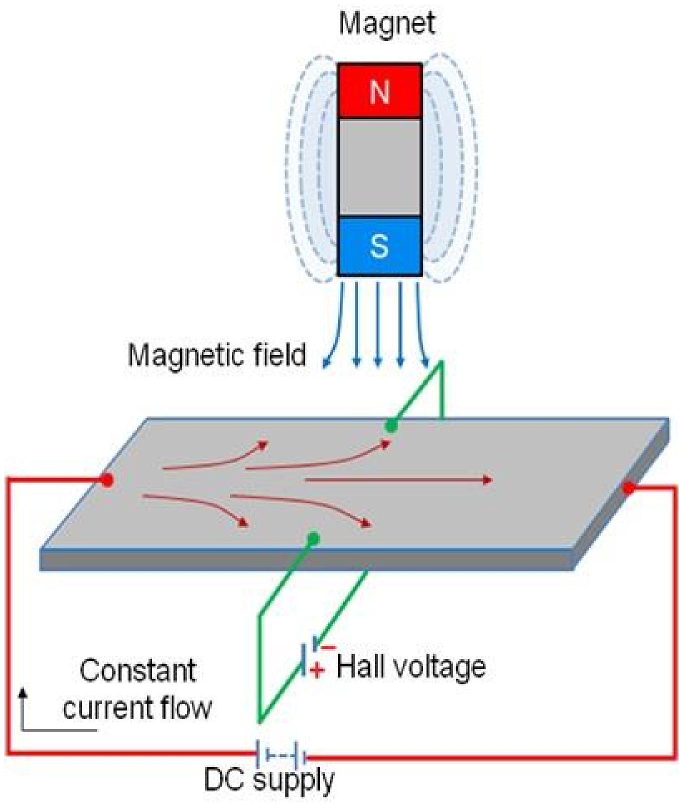

Figure 2. 6 - Hall effect

Most BLDC motors have three Hall sensors displaced at 120°, with 60° of resolution, embedded into the stator. Whenever one of the rotor poles pass near the sensor, it provides an electric signal positive (indicating the N pole) or negative (S pole). Combining the signals of the three Hall sensors the rotor position could be evaluated. Usually, embedding the sensor in the stator is a very difficult task because also small angular displacement could cause problem to the controller: often, near the Hall sensor, a Hall sensor magnet is placed, which is only a scaled replica of the rotor. When the permanent magnets turn, its effect is replicated by this device, allowing a right acquiring campaign.

The commutation sequence in one mechanical counterclockwise revolution is represented in figure 2.7. Hall sensors are indicated with H1, H2 and H3 and are displaced 120° one to another. In figure 2.7a, H1 and H3 see one S pole, so they provide a positive signal, while H2 detect a N pole. In this configuration, phase B (in red) is powered positively to attract the closest north pole, phase A (in black) is zero and phase C is connected to ground. When the north pole is approximately aligned with the phase B, sensor H3 turn off the signal, indicating that a shift of alimentation is needed. Hence, the controller reverses the power to phase A and C: at the former a current is subtracted, the latter is turned off. The next step (120° electrical degree) is shown in figure 2.7c, where H1 and H2 sensor provide a positive signal, therefore the electronic controller allows the current flow to pass in the phase C and turn off the phase B. In figure 2.7d is represented the situation at 180° electrical degree (the electrical angle is the mechanical one multiplied for the number of pole pairs), in which only the sensor H2 provide a tension signal, causing the shutdown of the phase A and the connection to the ground of the phase B (phase C remained powered positively). The cycle continues in the last two images (2.7e and 2.7f) in the same way described above.

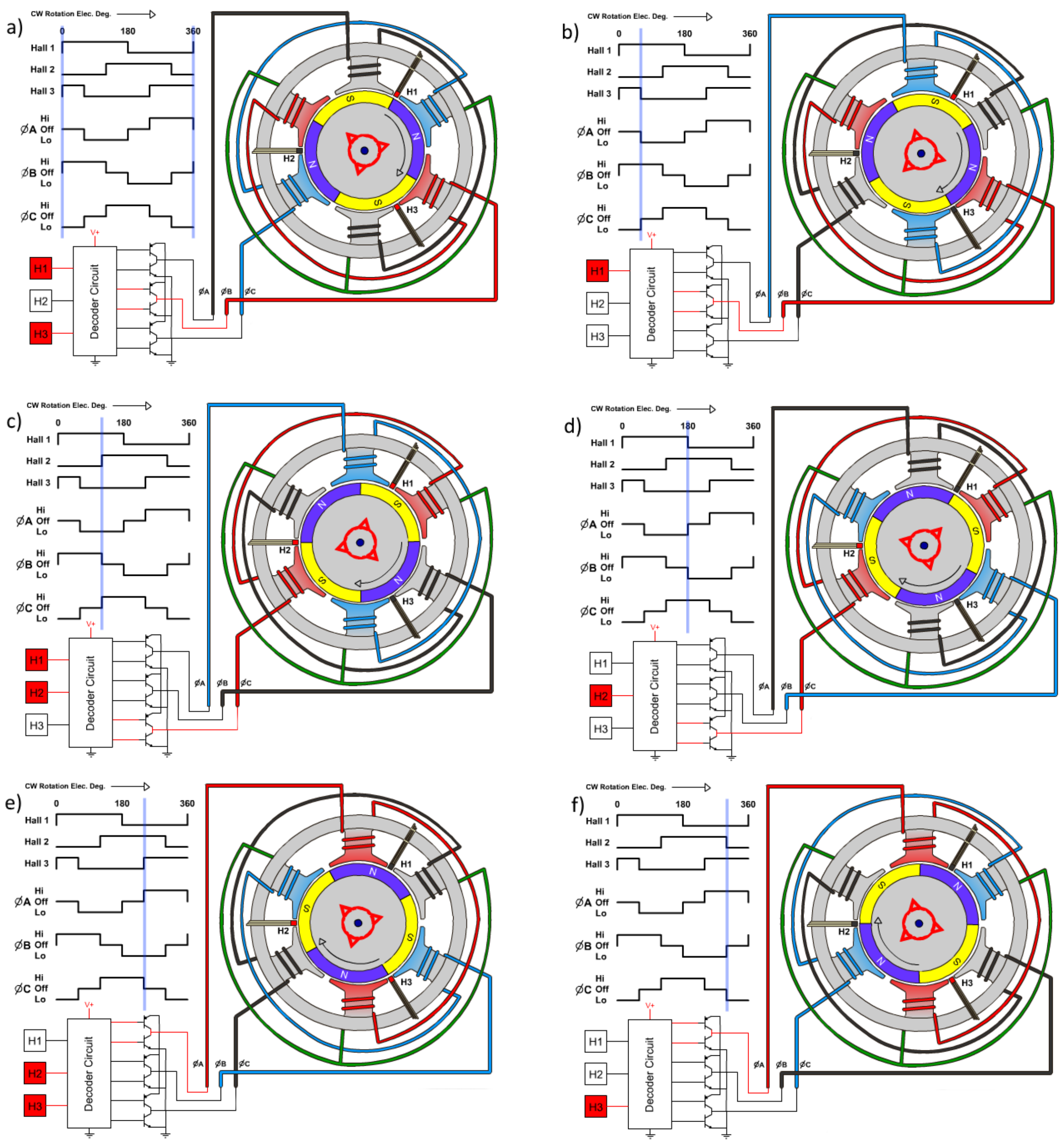

Figure 2. 7 - Commutation sequence in a three-phases two-pole pairs BLDC motor

It’s clear that this type of motor, powering only two phases each time, allows the cooling of the unpowered coil, but provides a smaller torque compared with sinusoidal AC induction motors. Furthermore, the continue commutation of the phases provokes a slight ripple in the speed, as shown in figure 2.8 (here, the speed error after the transient is due to the necessity of saturation of the controller, as discussed in next chapters).

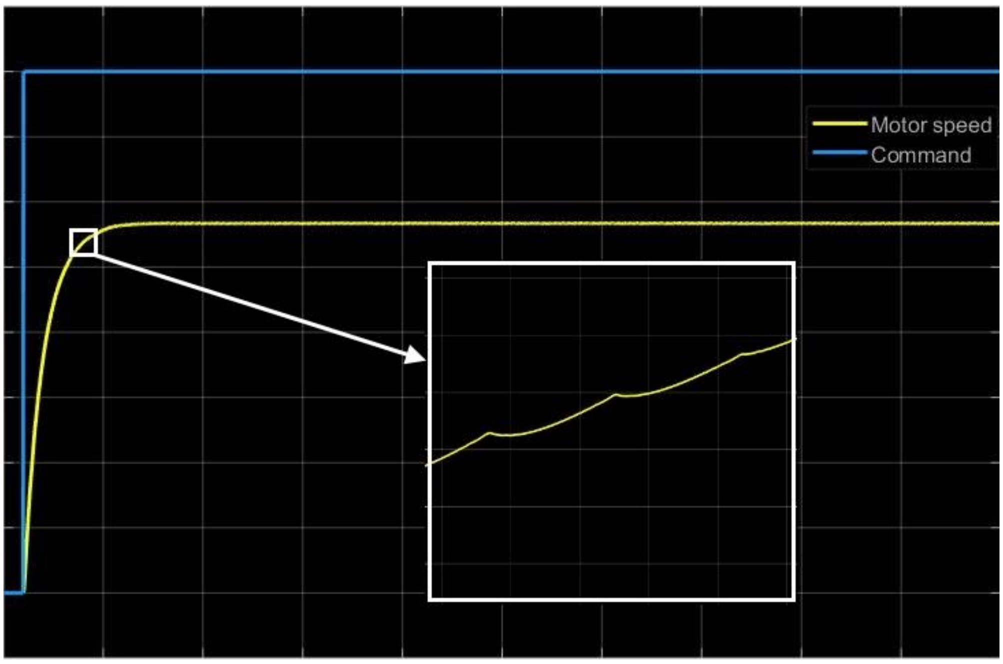

Figure 2. 8 - Speed ripple in a BLDC motor with step command

For a better explanation of the commutation sequence, in figure 2.9 is represented the three-phase inverter which controls the commutation of the motor phases thanks to a logic circuit acting on six power transistors; in table 1 the status of the sensor and motor phases for every degree are summarized.

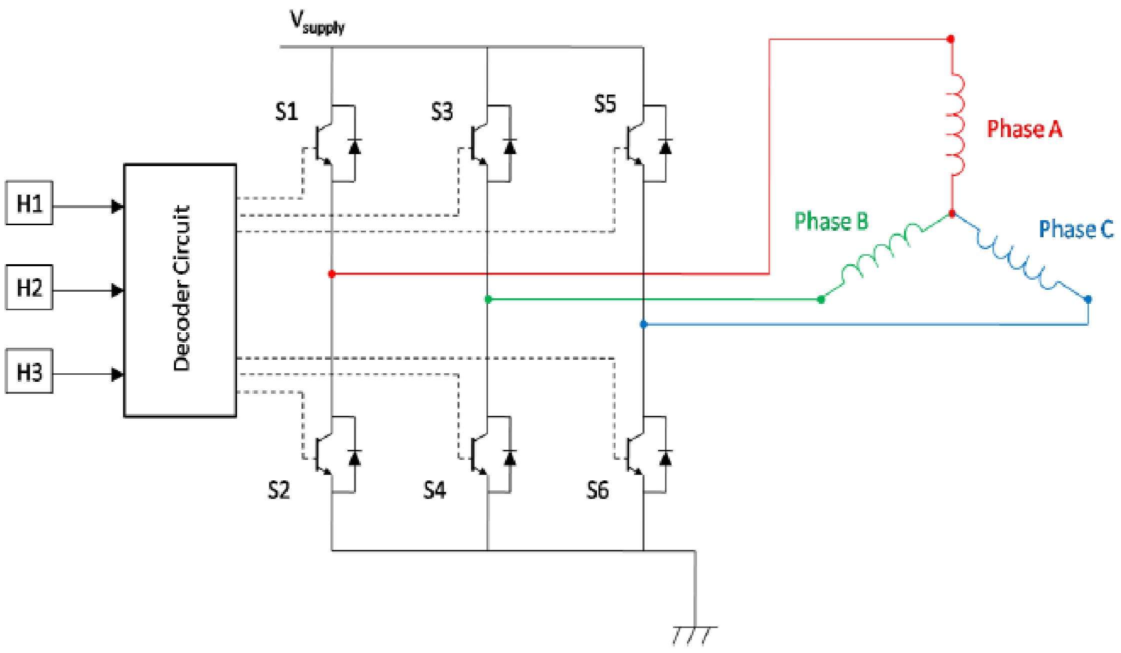

Figure 2. 9 - Electical scheme of the static inverter

Table 1 - Switching sequence

Electrical position		$0<\theta_e<60°$	$60°<\theta_e<120°$	$120°<\theta_e<180°$	$180°<\theta_e<240°$	$240°<\theta_e<300°$	$300°<\theta_e<360°$
Hall sensors	H1	1	1	1	0	0	0
	H2	0	0	1	1	1	0
	H3	1	0	0	0	1	1
Motor phases	A	off	ground	ground	off	supply	supply
	B	supply	supply	off	ground	ground	off
	C	ground	off	supply	supply	off	ground

2.4. Torque and efficiency

Torque on a BLDC motor is provided by the interactions between the stator windings and the rotor permanent magnets. Generically, for a coil in which the current pass through and immersed in a magnetic field (figure 2.10), the Lorentz law says that:

$$\vec{F} = i\vec{B}x\vec{l}$$

Where $\vec{B}$ is the magnetic induction vector (expressed in Wb/m^2), *i* is the current, and *l* is the length of the coil.

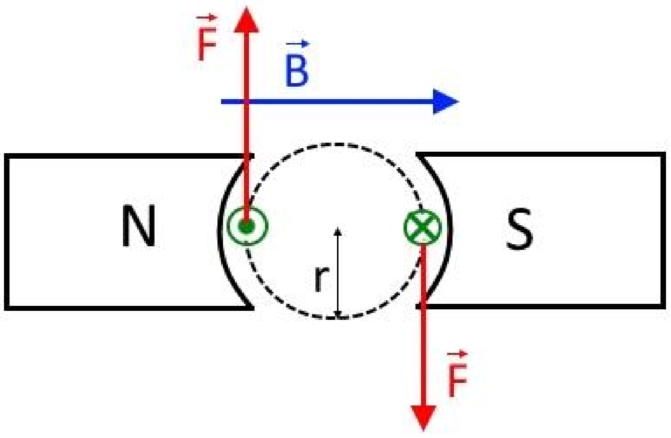

Figure 2. 10 - The coil in which pass the current (in green) immersed in a magnetic field (in blue) generates a force (in red)

Knowing that the motor torque could be expressed as:

$$C_m = F \cdot r \tag{2.1}$$

where *r* is the radius of the rotation of the coil (in the figure represented by a black dashed line), substituting the expression of the Lorentz law when the angle between magnetic induction and the direction of the coil is 90°, we could obtain:

$$C_m = iBlr = i\frac{\phi}{A}lr \tag{2.2}$$

$$A = \frac{2\pi rl}{p} \tag{2.3}$$

where ϕ is the intensity of the magnetic flux (expressed in [Wb]) and A the area interested by the magnetic flux: the rotation of the motor modifies the magnetic flux crossing each coil in stator windings according to Faraday's law, so a counterelectromotive force is applied at each phase. To find the expression of the torque is required to combine the expressions (2.2) and (2.3), finding:

$$C_m = k_c i \tag{2.4}$$

$$k_c = \frac{\phi p}{2\pi} \tag{2.5}$$

where k_c is the torque constant.

The total torque could be calculated adding the contributions of the three phases of the actuator. This value is maximum when the magnetic field generated by the permanent magnet of the rotor and by the stator are perpendicular: the control logic tries to maintain the phases closest to π/2.

Using the expressions listed above, the torque could be simply expressed as a function of speed. Knowing that a BLDC motor could be modelled with:

$$V_m = Ri + k_c\omega \tag{2.6}$$

isolating the current and remembering the equation (2.4), the torque is:

$$C_m = k_c i = \frac{V_m k_c}{R} - \frac{k_c^2}{R}\omega \tag{2.7}$$

In the same way, we could also express the power of the actuator as:

$$P_{out} = C_m\omega = \frac{V_m k_c}{R}\omega - \frac{k_c^2}{R}\omega^2 \tag{2.8}$$

It's important to underline the speed dependency from the torque and the power: the former is a linear function, the latter is a parabola. These two relations could be represented in a graph (figure 2.11).

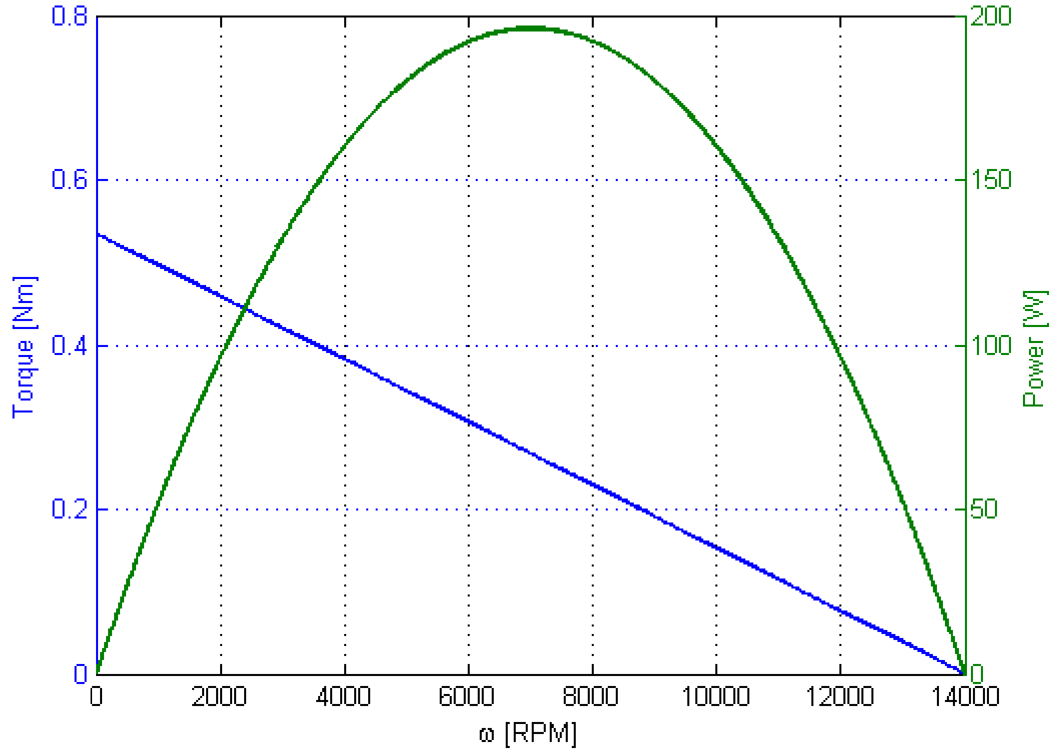

Figure 2. 11 - Torque and power trend in a BLDC motor

During nominal operations, the motor can be loaded up to the rated torque (this value is reported in the motor's datasheet). Up to the rated speed, the torque in a BLDC motor remains steady. Once arrived at the rated speed, this one could be further increased until the 150% of the nominal speed, but the torque drops down. In applications which require frequent start-and-stop operations, there is the necessity of a torque superior to the rated one (frictions and inertia must be overwhelmed). The actuator can provide a higher torque for a brief period, but the rotational speed is low, and the dissipations become very important. In fact, in the graph it's possible to see that the maximum power is obtainable at the rated speed with the rated torque; every other position in the graph implies a reduction of the available power due to dissipations.

Efficiency of the system is defined as the ratio between the output power and the provided power. The friction and viscous effect are neglected in this explanation because are small compared with the Joule effect dissipations. Hence:

$$\eta = \frac{P_{out}}{P_{in}} \tag{2.9}$$

Knowing the expression of the P_{in}, it's possible to write that:

$$P_{in} = Vi \tag{2.10}$$

$$\eta = \frac{C_m \omega}{Vi} = \frac{k_c \omega}{V} \tag{2.11}$$

2.5. Control

The comprehension of the main features of a BLDC motor allow the description of its control, which ensure reliable and safe operations. There are several different control modes, depending mainly on the purpose of the application.

The most used is the *speed control*, employed in systems which receive an on/off command such hydraulic valves or reservoir or compressors. In BLDC motor speed control, the actual speed is compared with the commanded one and the error is sent a P.I.D. controller (Proportional – Integrative – Derivative). This device calculates if the error lays inside a two-line limit zone, which represent the acceptable error of the system. If the speed is higher than the upper limit or lower than the lower limit, the control logic gives a step command to the motor to accelerate. This acceleration is achieved modifying the voltage which powers the actuator: usually a constant DC voltage is source is available. When the Pulse Width Modulation (PWM) signal is 1, the motor is forced to accelerate due to the positive voltage, when the duty cycle is 0 the applied voltage is 0. The frequency of the PWM signal is proportional to the analogic signal wanted. A simple example of this method is shown in figure 2.12.

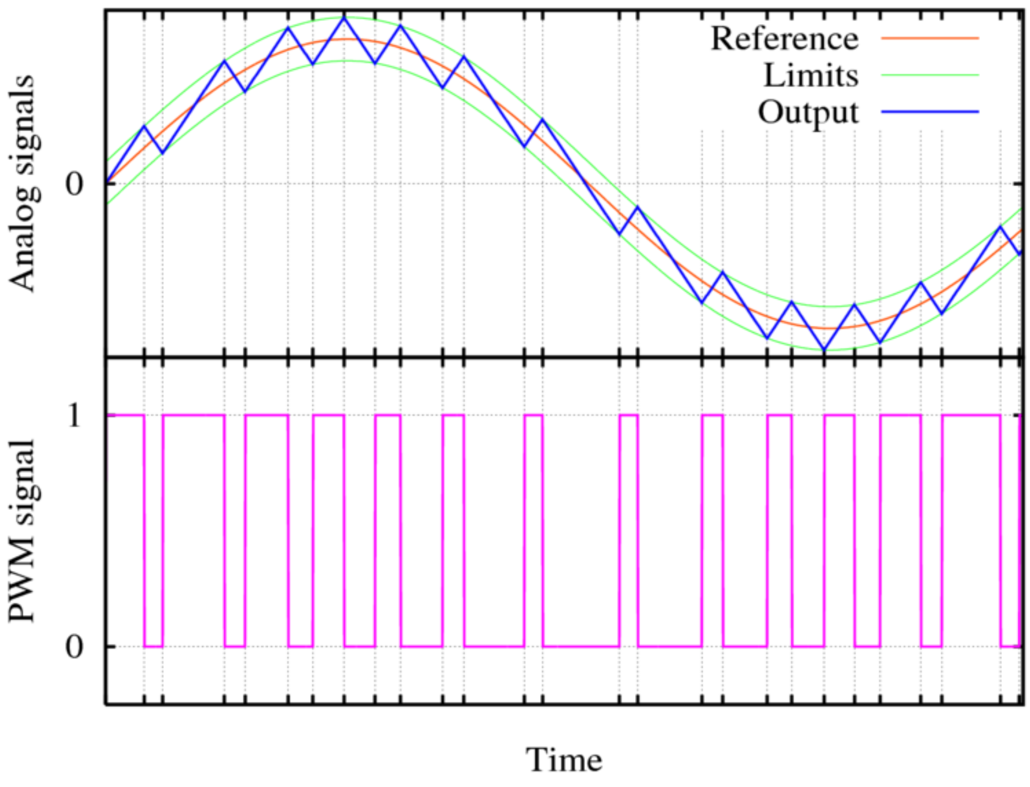

Figure 2. 12 - PWM control logic

The speed is evaluated using the signals of the Hall sensor or an encoder located properly on the moving shaft. The frequency of the PWM signal is given by the P.I.D. controller proportionally to the trend of the error: if this one is high, the signal will be 1 for a longer time than the case in which the error is small. A block diagram of a speed control loop is represented in figure 2.13.

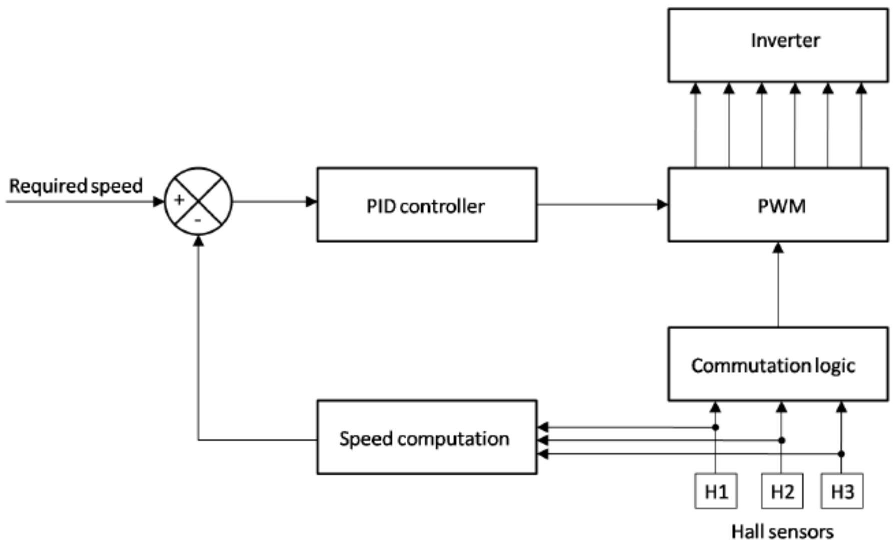

Figure 2. 13 - Block diagram for a speed control loop

Another type of control loop is the *torque control*, which allows the actuator to have always the same output torque regardless the external load, the position or the speed. It's already been said in paragraph 2.4. that the torque depends on the magnetic flux on the phase windings, but it also depends on the current through the torque constant k_c. Hence, it's possible to control the torque modifying the current which flows in the phase windings. This control is widely used also as inner loop in system controlled by a speed loop logic. The block diagram for this type of control is shown in figure 2.14.

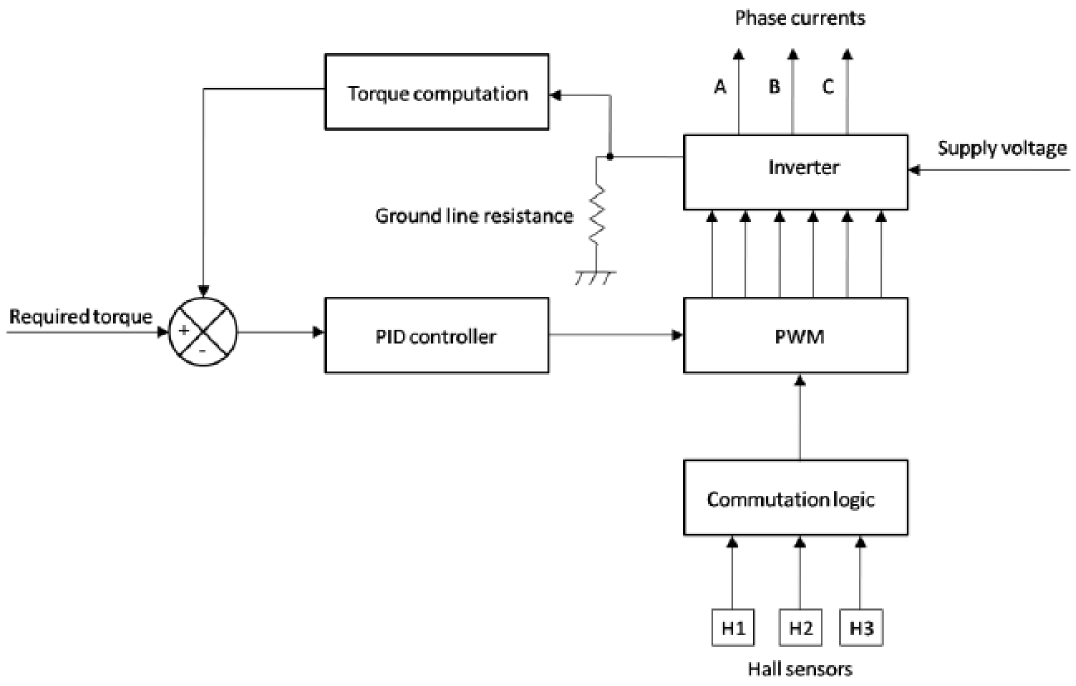

Figure 2. 14 - Block diagram for a torque control loop

In a motor control design, it's important to take care of the safety and reliability of the actuator: in order to do this, *motor protection* control logic has been developed. A simple example could be the rotor stuck: in this case the current increases strongly, overheating the windings and possibly burning the power electronic devices driving the motor. This system takes care about:

- Peak current: the maximum instantaneous current allowed to flow in phase windings. This condition may occur when the windings cause a short circuit: in this case the control turn off the PWM signal in order to interrupt the power to the windings;
- Maximum working current: the extreme value of the output current when the motor needs to bear an overload. The implementation of this logic is similar to a torque control;

- Under voltage: If the system is powered with battery, it’s important to turn off the actuator when the voltage drop off a lower limit;
- Hall sensor failure: position and speed are determined using the signals provided by Hall sensors. If one of these devices break down, the commutation sequence will interrupt, and it can cause the rotor gets stuck or the growing of too high currents. The sensor failure could be detached via firmware, which controls if the logic of the sensor changes during the normal working activities.

3. Electro-mechanical actuator models

The aim of this work is to simply detect some types of progressive fault affecting electro-mechanical actuators, in order to make maintenance planning more efficient and the reliability of the system higher. To simulate the real behaviour of an EMA a reference model has been developed in Simulink environment. This is a high-fidelity model which provides the working outputs such as position, speed and absorbed current coming from the normal activities of the actuator. Once obtained these values, a simplified EMA monitor model needs to detect as best as it can the faults which have been introduced in the reference model. Below there is a deeper description of these two actuator models.

3.1. Reference model

The aim of the reference model, as already said, is to simulate the real working behaviour of an electromechanical system, avoiding the necessity to have a real and expensive test bench. This model is developed and implemented in a Matlab-Simulink environment and represents a flap control, which has a dynamic response in an intermediate range between a primary and a secondary flight control.

This model simulates the first 0.5 seconds of the behaviour of the electromechanical system. The simulation is very complex and computationally expensive, so a Euler first-order fixed-step resolution method, which is the simplest and most controllable possible, is employed. The time step between an evaluation and the next is set at $1 \cdot 10^{-6}$ s, at least two order less than every system in the model, in order to take care about the dynamics of all devices. It’s impossible to use second order methods as Runge-Kutta or Dormand-Price, because during the simulation they will interpolate data during the recursive evaluation of the error, causing the non-linearity of the convergence of the problem.

The main features and parameters of the reference model are listed in table 2 below.

Table 2 - Main parameters of the EMA system

Parameter	Symbol	Value	Measure Unit
Error proportional gain	G_{prop}	10^5	-
PID controller: proportional gain	GAP	0.05	$\frac{Nms}{rad}$
PID controller: integrative gain	GAI	0	$\frac{Nm}{rad}$
PID controller: derivative gain	GAD	0	$\frac{Nms^2}{rad}$
Maximum power supply voltage	V_{max}	48	V
Maximum current	I_{max}	22.5	A
Maximum motor torque	$T_{m,max}$	1.689	Nm
Torque constant	k_t	0.0752	$\frac{Nm}{A}$
Back-EMF constant	k_e	0.0752	$\frac{Vs}{rad}$
Phase-to-phase resistance	R_s	2.13	Ω
Phase-to-phase inductance	L_s	$7.2 \cdot 10^{-4}$	H
RL time constant of BLDC motor	τ_{RLs}	$\frac{R_s}{L_s}$	s
Polar expansions per phase	$2P$	4	
Number of polepairs per phase	P	2	
Current hysteresis band width	hb	0.5	A
Inertial Torque of the motor	J_m	$1.3 \cdot 10^{-5}$	$kg \cdot m^2$
Viscous damping coefficient of the motor	C_m	$\frac{30}{\pi} \cdot 10^{-6}$	$\frac{Nms}{rad}$
Inertial Torque of the user	J_u	$1.2 \cdot 10^{-5}$	$kg \cdot m^2$
Viscous damping coefficient of the user	C_u	$4.5 \cdot 10^{-7}$	$\frac{Nms}{rad}$
Static friction torque of the motor	f_{sm}	$0.06 \cdot T_{m,max}$	Nm
Dynamic friction torque of the motor	f_{dm}	$\frac{f_{sm}}{2}$	Nm
Static friction torque of the user	f_{su}	$0.04 \cdot T_{m,max}$	Nm
Dynamic friction torque of the user	f_{du}	$\frac{f_{su}}{2}$	Nm
Nominal backlash	BLK	$5 \cdot 10^{-3}$	rad

Schematically, the reference model is divided into four main blocks, as shown in figure 3.1.

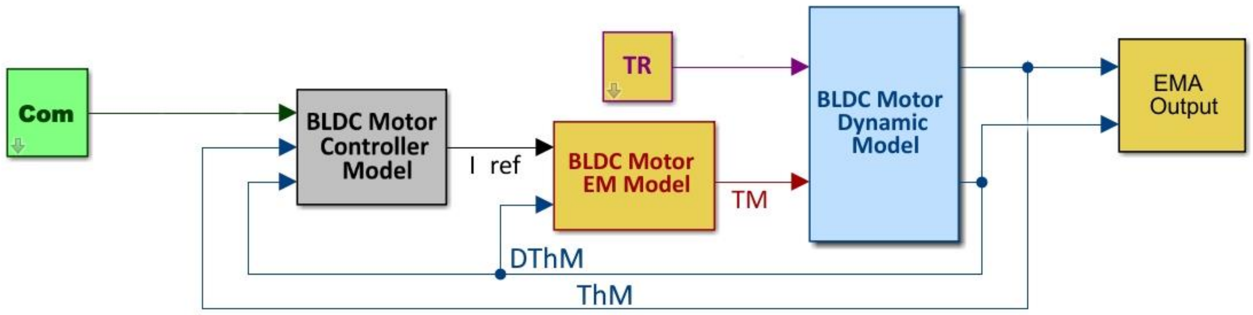

Figure 3. 1 - Reference model

Com block represent the command given to the actuator; the output signal is an angle and it's sent to *BLDC Motor Controller Model*. It compares the commanded signal with the speed and position feedbacks (DThM and ThM) and provide the reference current to give to the BLDC motor. This parameter enters in the *BLDC motor electromechanical model*, which evaluates the torque developed by the actuator. The *BLDC Dynamic Model,* comparing the torque of the motor and the resistant torque of the system is able to calculate the actual speed and position of the rotor, in order to close the two control rings. Speed, position and a lot of intermediate parameters are also sent to Matlab workspace, to make the post-processing activities easier. In the next paragraphs all blocks are deeply described and commented.

3.1.1. *Com* block

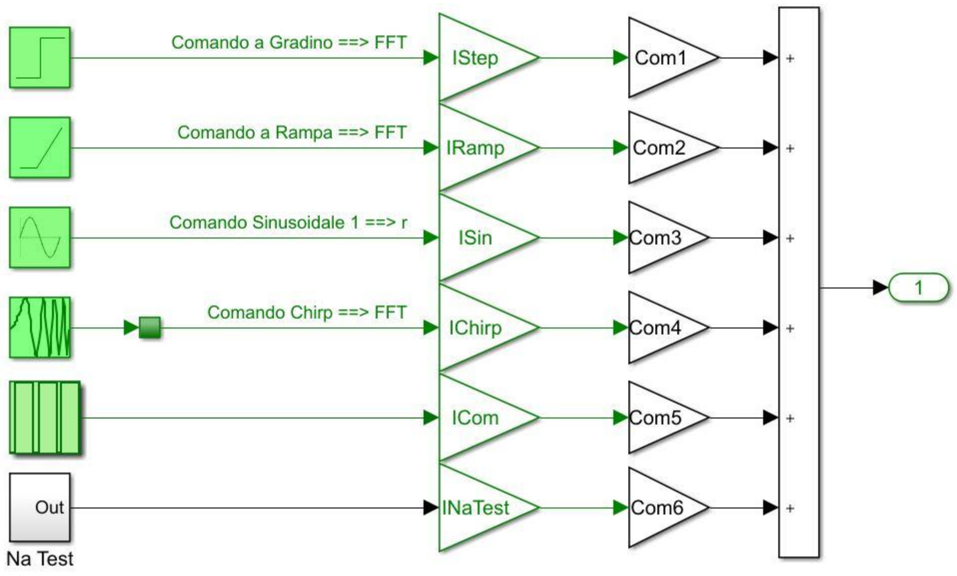

Figure 3. 2 - Com block

From the Com block, is possible to set the type of command given to the system. User could choose a step, ramp, sine wave, chirp command or a custom time history. The parameters named “Com” followed by a number are used to select from the Matlab workspace the command; their amplitudes are set from a Simulink’s dialogue window. It’s important to underline that the chirp command could be obtained in two different ways, selectable from the small green square close to the command square: one uses the parameters coming from the Simulink’s dialogue window, the other employs a handmade function which decreases from 1 to 0.

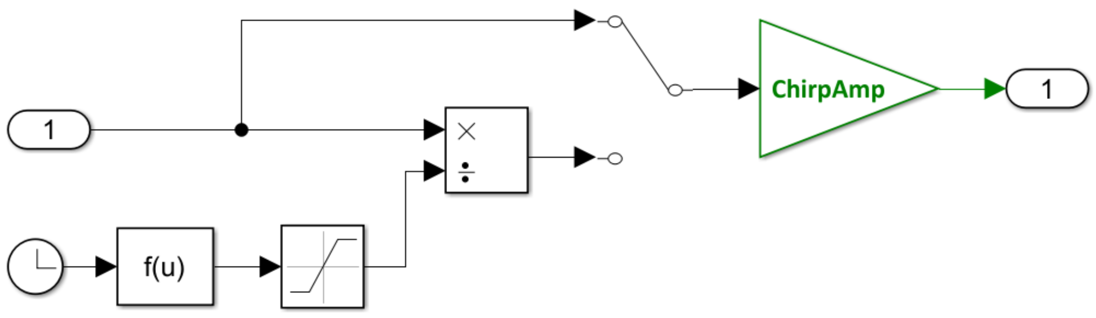

Figure 3. 3 - Chirp command in the Com block

3.1.2. *BLDC Motor Controller Model* block

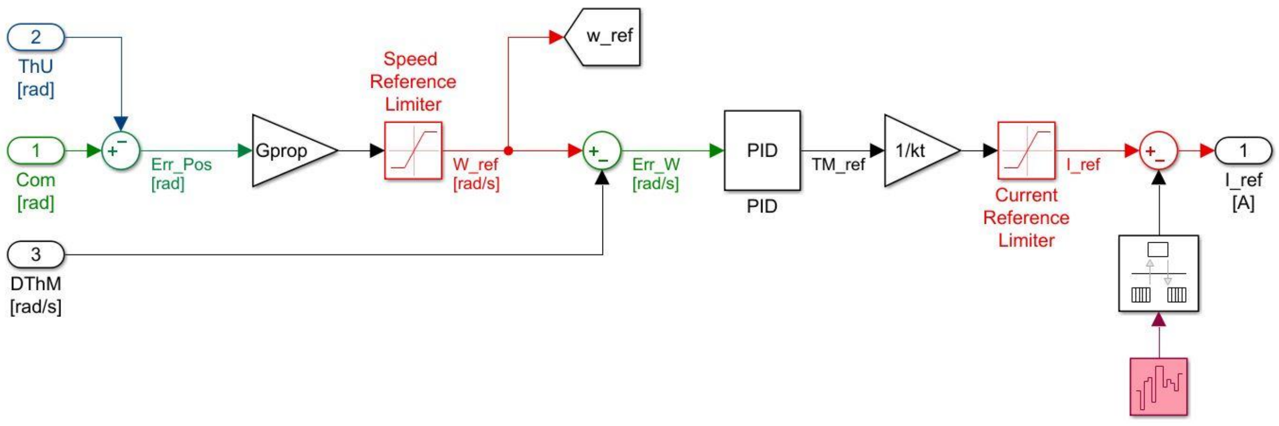

Figure 3. 4 - BLDC Motor Control Model block

This block allows the control of the entire EMA system. It compares the position command and the position feedback, providing the position error; this is suitably transformed in a speed signal by the G_{prop} block and then limited by a saturation block, which maximum and minimum values are equal to ±8000 rpm. The reference speed is subsequently compared with the feedback one and the error is sent into the PID controller: here the input signal is transformed in a reference torque. The division for the torque constant allows to find the reference current: this parameter needs to be saturated at the maximum current (22.5 A) to avoid breakages or faulty conditions of the motor. It's important to highlight that the reference current is a parameter which have only a control meaning, and it's not related with the real current in the stator coils. Before the real calculation of the reference current is possible to add a white-noise disturbance block, which generates normally distributed random numbers that are suitable for use in continuous or hybrid systems. These number are obviously multiplied by 10^{-6} in order to make the two signals comparable. In our work, the noise gain (K_{noise}) is set to 0, because is proved that its effect on the system is negligible.

3.1.3. *BLDC Motor ElectroMechanical Model* block

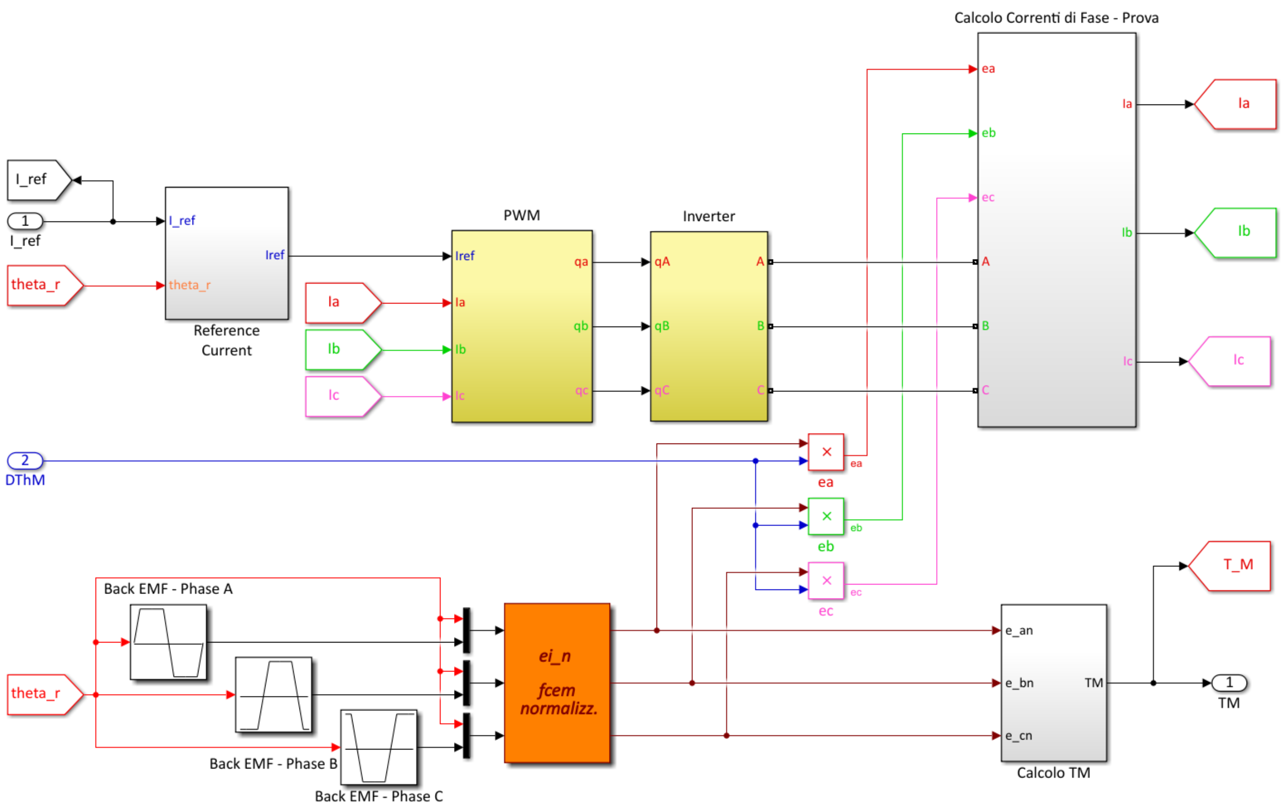

Figure 3. 5 - BLDC Motor Electromechanical Model block

This subsystem is maybe the most complex of the reference model because it takes care about a lot of phenomena acting on the EMA. Its aims are mainly to distribute the reference current calculated by the control electronics on the three phases of the motor depending on the actual current of each phase and to evaluate the motor torque. This block is composed by several subsystems:

- *Reference current* subsystem has as input the reference current previously evaluated and the actual position of the rotor. In this block are evaluated the three phase currents, using a lookup table block to model the three functions of the phases (as shown in figure 3.6).

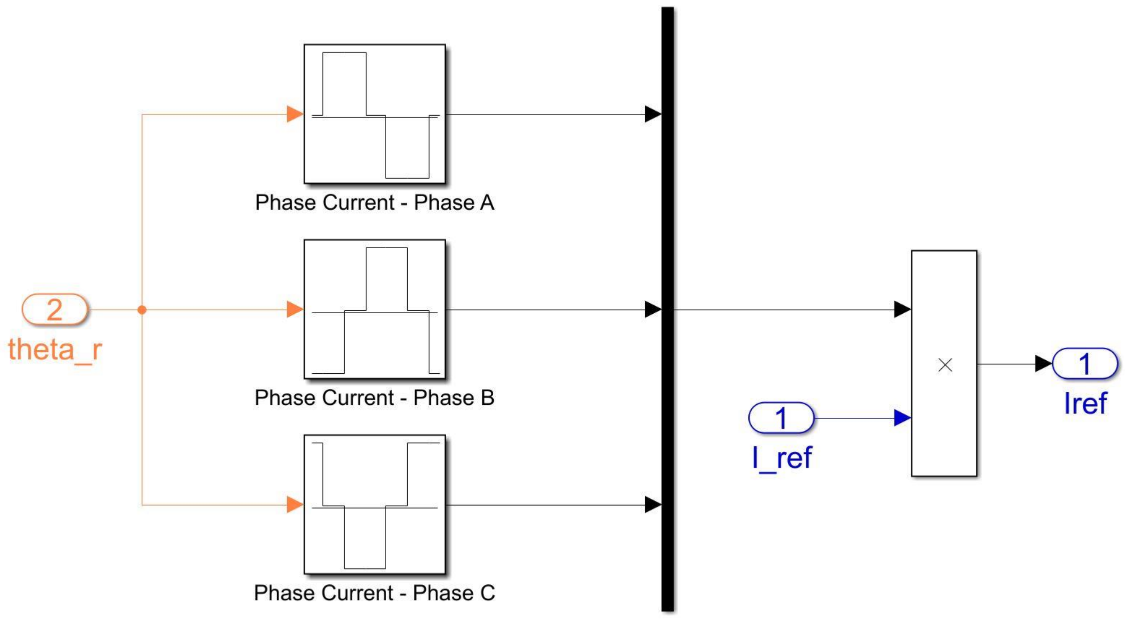

Figure 3. 6 - Evaluating of the three phase currents

The rotor angle is split in three different functions (which values could be only 1, 0 and -1), which take care about the characteristics of the trapezoidal BLDC motor: one phase is powered positively (current in, signal 1), another is turned off (signal 0) and the third is powered negatively (current out, signal -1). These signals are then multiplied by the value of reference current to obtain the time trend of three phase currents.

- The *PWM* block (represented in figure 3.7) receives as input the three-phase reference currents just evaluated and compares every row with the actual current circulating in the stator windings. If the difference is a value greater or lower than a hysteretic value, set as *hb*=0.5 A, the output is a Boolean positive value (q_a, q_b or q_c). This signal is useful for the motor control.
- In the *Inverter* block (figure 3.8) every Boolean signal coming from PWM block is negated and then the six signals are sent to a H-bridge with six power transistors controlled by the electronics of the system. It's important to highlight that the H-bridge

provides as output the time-trend of the voltages of three phases of the motor, which will be used later to calculate the effective phase currents.

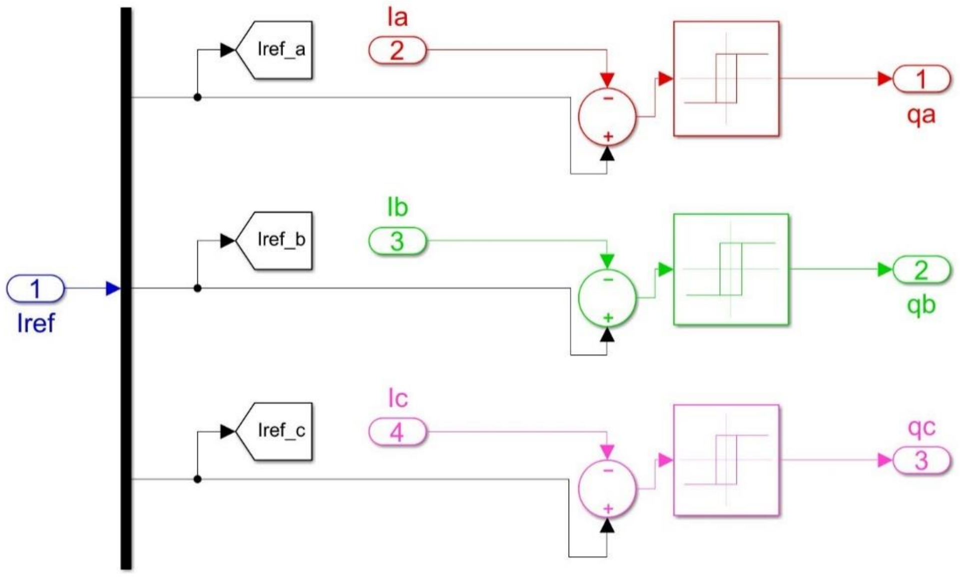

Figure 3. 7 - PWM block

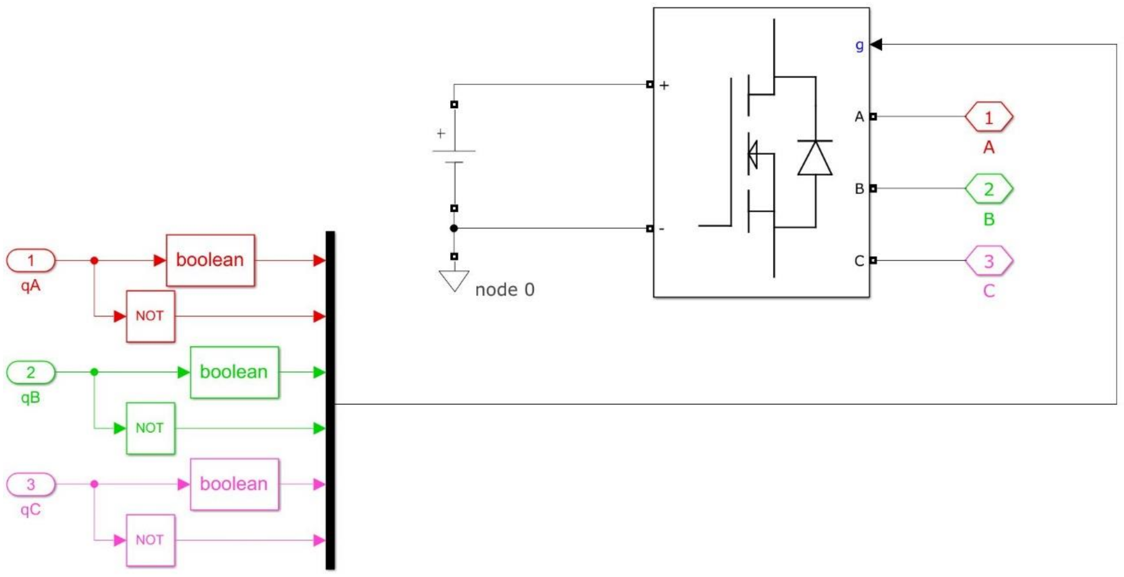

Figure 3. 8 - Inverter block

- Exploiting the rotor angular position for the three phases, into the *Normalized F_CEM* block (figure 3.9) the counter-electromotive force is evaluated. It’s possible to write:

$$F_{cem} = k_{fcem} \cdot \omega_m \tag{3.1}$$

$$F_{nfcem} = k_{fcem} = \frac{F_{cem}}{\omega_m} \tag{3.2}$$

where F_{nfcem} is the normalized counter-electromotive force. For a system with a number of pole-pairs greater than one, the evaluation of the k_{fcem} is strictly dependent from the rotor angle:

$$k_{fcem} = k_e(\theta_m) \cdot \left(1 + \zeta \cos\left(\theta_m + \frac{2 \cdot (i-1)}{3}\pi\right)\right) \tag{3.3}$$

where $k_e(\theta_m)$ is the trapezoidal wave-shaped normalized counter-electromotive force of the *i-th* phase of the non-faulty motor and $\zeta = \frac{x_0}{g_0}$ is the ratio between the misalignment of rotor and stator axis (x_0) and the nominal value of the gap between rotor and stator.

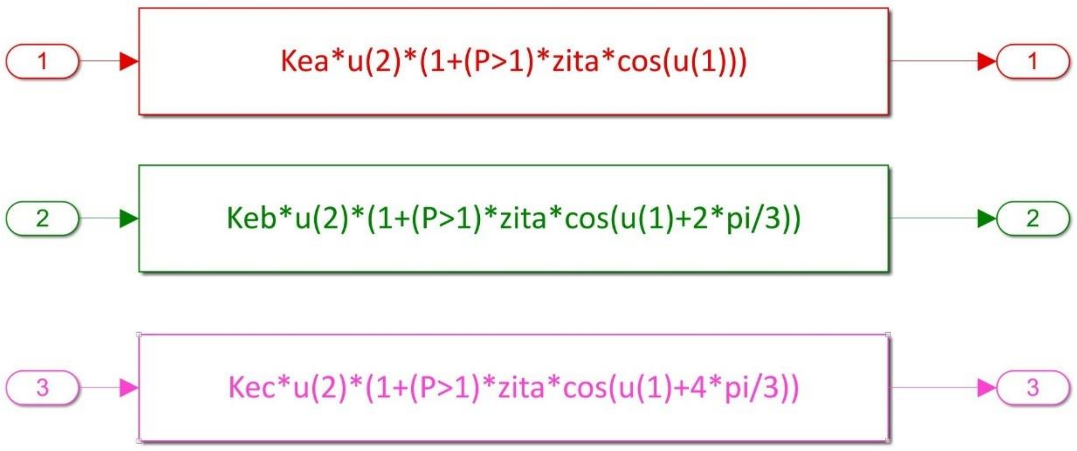
1
Kea*u(2)*(1+(P>1)*zita*cos(u(1)))
1
2
Keb*u(2)*(1+(P>1)*zita*cos(u(1)+2*pi/3))
2
3
Kec*u(2)*(1+(P>1)*zita*cos(u(1)+4*pi/3))
3

Figure 3. 9 - The evaluation of the normalized CEMF for the three motor phases

These equations allow to implement two possible faulty conditions of the motor: the eccentricity of the motor and the possible overheat, which could modify the value of the normalized CEMF.
The outputs of this block are then multiplied for the angular speed of the actuator according to equation (3.1) in order to find the real counter-electromotive force of each phase, as shown in figure 3.10. These parameters are useful to calculate the effective phase current and the torque generated by the motor.

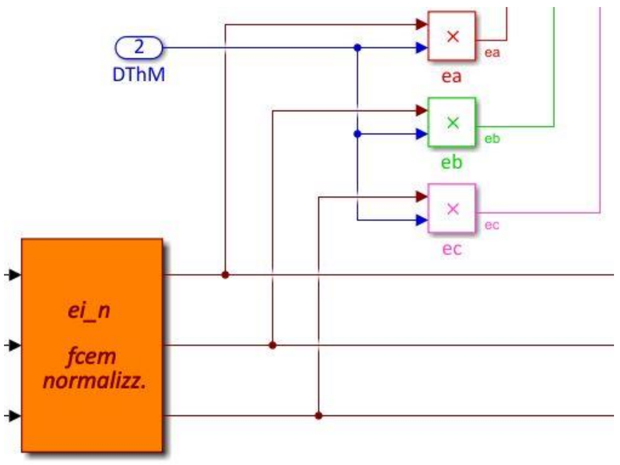

Figure 3. 10 - The product between the speed and the normalized CEMF

- ➢ The *Phase current calculation* block receives as inputs the phase voltages and the counter-electromotive forces to find the effective phase currents.

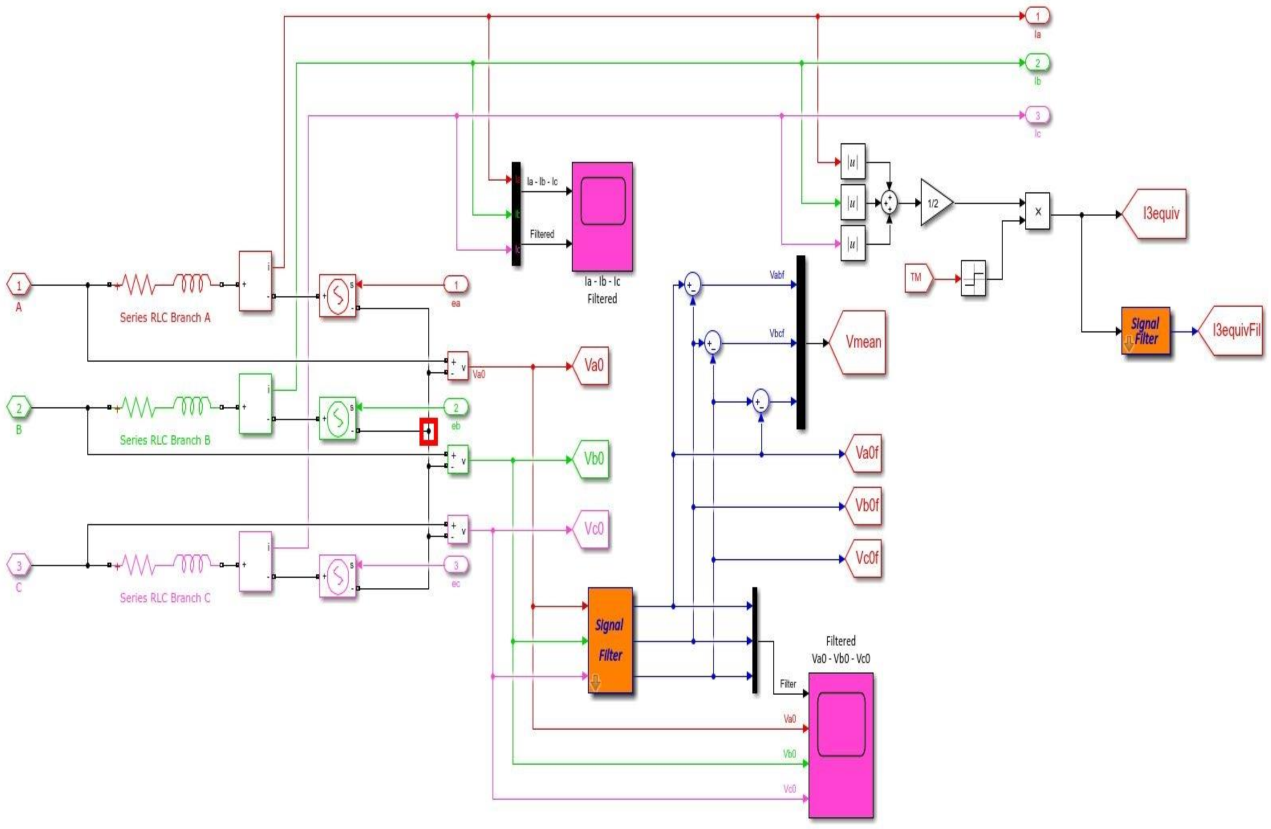

Figure 3. 11 - Phase current calculation subsystem

This subsystem is a multi-domain because Simulink was not able to manage the portion of the model which describes the physics of the stator circuit. Indeed, the stator phases could be described as three solenoids linked in star-pattern with floating centre: this configuration is not conceived by Simulink, which could only design symmetric star-pattern circuits. The voltage in the star centre is simply evaluated as the average of the voltages on the three branches, but with the introduction of possible overheat, overcurrent and eccentricity conditions this calculation is not right anymore. Hence, on the left of the figure 3.11, it's possible to notice a SIM Power System (it is a Simulink tool) configuration, which takes care about the real behaviour of the stator windings and about the counter-electromotive force previously calculated. In this way, the Simulink model is able to evaluate time by time the current circulating in the windings and find the tension on the star centre (highlighted in the figure 3.11 with a thick red square).

In the upper part of the subsystem, phase currents are suitably processed in order to obtain a parameter usable in the simplified monitor model, called $I_{3\ equiv}$, which has the same sign of the torque acting on the rotor and it is the equivalent single-phase current.

- The last subsystem of BLDC Motor Electromechanical Model is the *Torque computation* block.

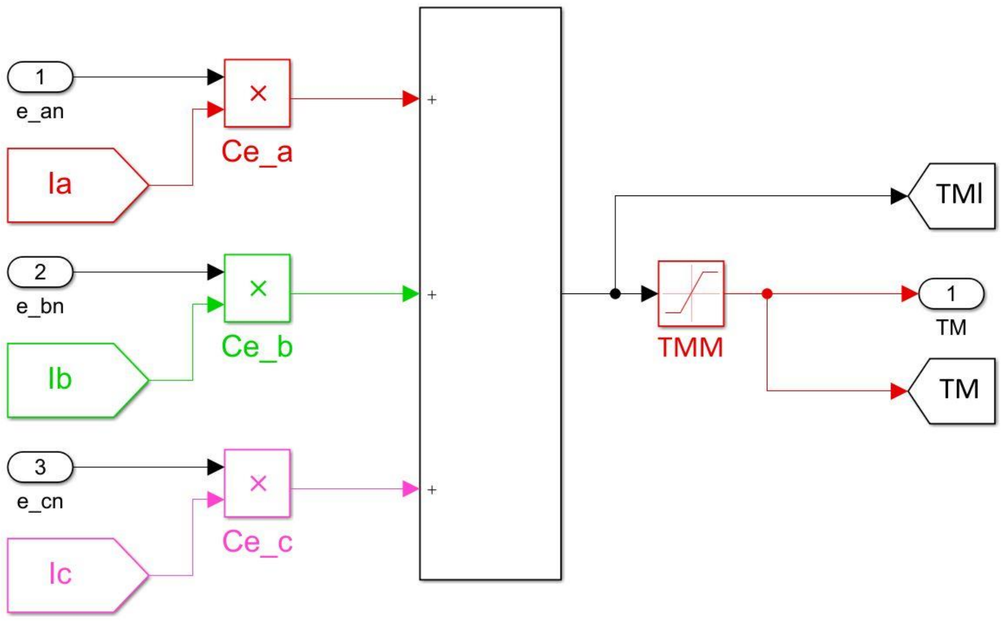

Figure 3. 12 - Torque computation block

In this block the torque is not calculated with the equation (3.4) displayed below, because when the motor stops, the torque should diverge to infinite.

$$T_m = \frac{P}{\omega_m} \tag{3.4}$$

From literature, is known that the torque could be also calculated with:

$$T_m = \sum_{i=1}^{3} F_{nfcem_i} \cdot I_i \tag{3.5}$$

where I_i is the current of the *i-th* phase. In this subsystem is also necessary the saturation block, to avoid the possibility of a torque superior than the maximum admitted by the structure of the motor.

3.1.4. *BLDC Motor Dynamic Model* block

In this block is computed the dynamic of the actuator using a mechanical device: in our case is a nut-screw system which moves the end-effector surface.

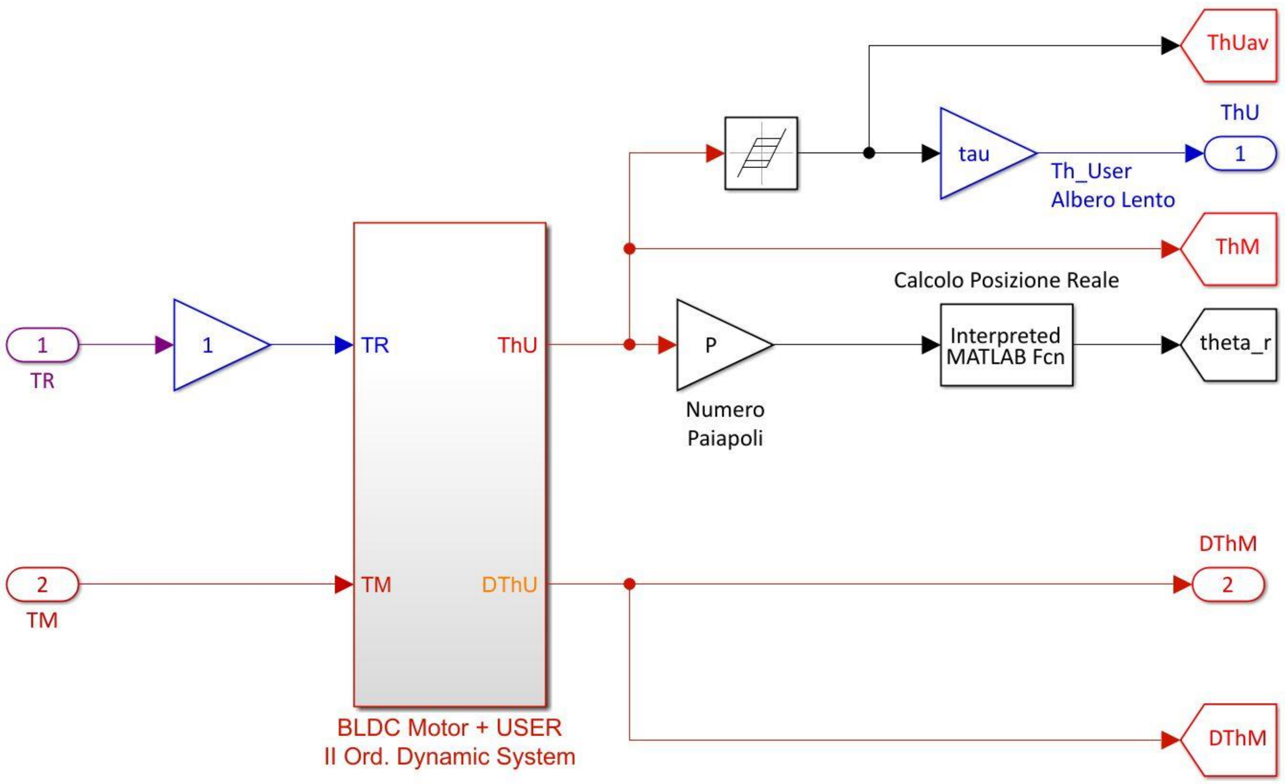

Figure 3. 13 - Dynamic subsystem

As shown in figure 3.13, the motor and the resistant torques are the input of a second order dynamic system, which provides as outputs the position and the speed of the user. These parameters are then manipulated to achieve the angular position of the rotor and of the motor, the real position of the user which takes care about the backlash of the nut-screw system and the speed of the motor.

The second order system is represented in figure 3.14.

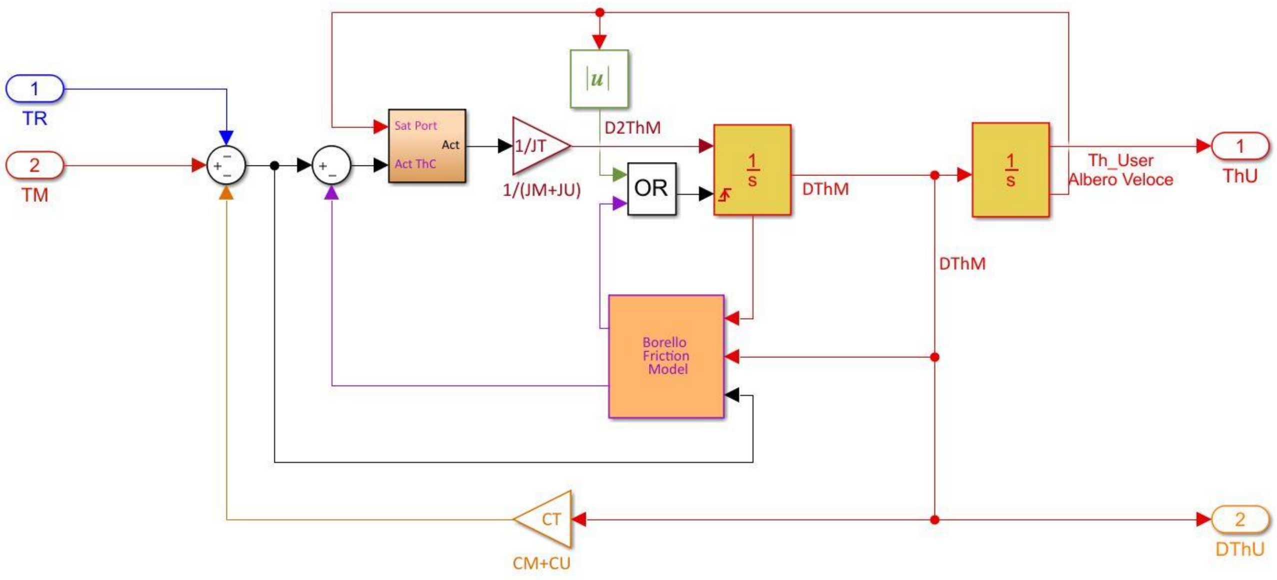

Figure 3. 14 - Second order dynamic model

The actual value of the net driving torque T_Act (calculated as the algebraic sum of the motor torque TM, resistant torque TR and friction torque) is then processed by a control subsystem apt to simulate the effect of the SM mechanical hard-stops. Subsequently, it is divided by total inertia (which is the sum of the motor and user inertias) and subsequently is twice integrated to find speed and position. Once reached the SM physical limit (i.e. its mechanical end-strokes), the system sets the output position to 0, but needs to stop to iterate the speed and the torque: from the second integrator (now in the saturation condition) it is possible to notice that an arrow come back to a saturation port located in the control block (shown in figure 3.15 below).

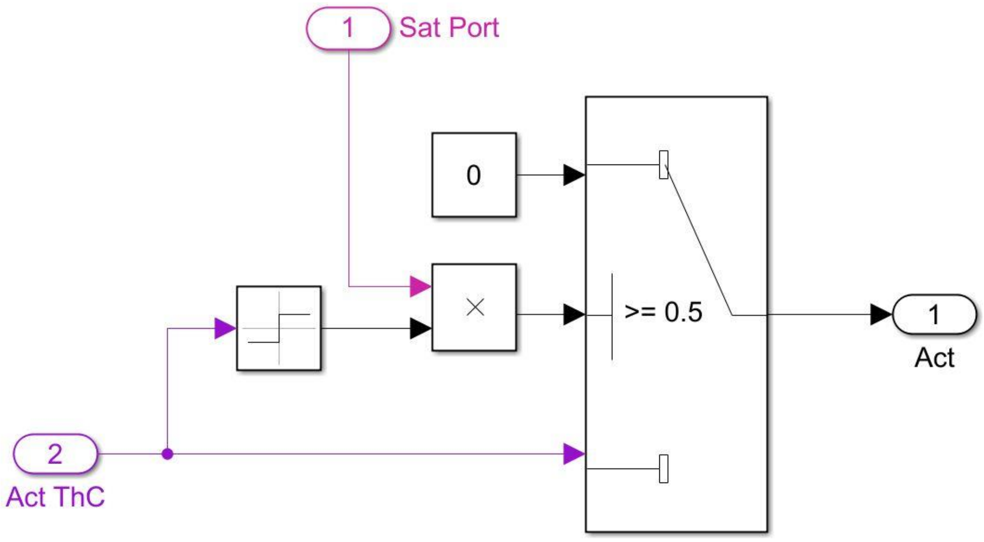

Figure 3. 15 - Limit-control block

The saturation port simulates the three conditions detachable in a mechanical system:

- the end-effector is not at one of its limits;
- the end effector is at one of its limits and it is applying a force directed toward the limit;
- the end effector is at one of its limits and it is applying a force directed opposite the limit.

The values of the saturation port could be 0 if the end-effector is not at its limit and ±1 depending on the limit reached. If the product between the saturation port and the sign of the active theoretic force (it does not take care about the limit) is greater than 0.5 (it could be only 1, 0 or -1), the acceleration is 0. If the saturation port is 0 (limit not reached yet), the theoretic force pass through the switch and becomes the real torque needed.

Concerning the first integrator, which stops the calculation if receives as external reset a rising number, is controlled by an OR block. It takes care about the conditions:

1. The end-effector is at one of its limits, so the saturation port returns a value equal to ±1. With the absolute-value block, the integrator will see a 0→1 commutation and will stop its work;
2. When the speed is close to 0 due to the friction, the Simulink model is not able to evaluate time by time the sign of the force, so the Stribeck effect could take place. In this case, the mass which should stop is moving in the opposite direction due to friction force, which acts as an active strength. This is a physics absurd, so a reset of

the first integration is required when the speed decreases more than a pre-fixed value.

Also the *Borello Friction Model* block (represented in figure 3.16), like the *limit-control* block, exploits a switch to pass from two conditions: in this case are the static and the dynamic friction conditions. During the evaluation of the speed, the middle purple line with the *hit crossing* block verifies if the speed maintains the same sign between two consequent instants. If this condition is true, the total friction is evaluated with the red part of the subsystem, which represents the dynamic friction; else the blue lines, which correspond of the static friction condition, are used. The static friction can be found relating to the motive force: until the body is still, the friction and the motive force are the same. When the first separation takes place, the value of the static friction force has been determined and remains constant. A deeper mathematical discussion is reported at chapter 4.2.

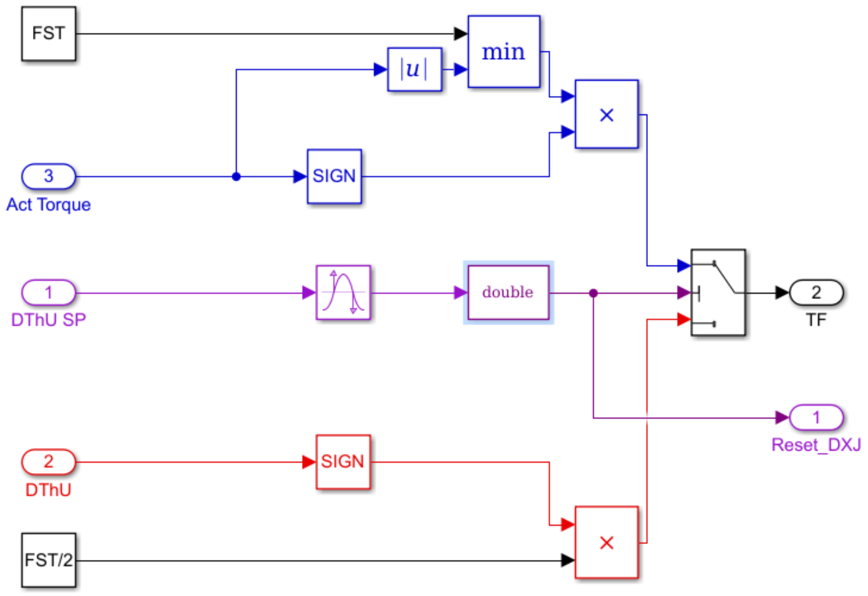

Figure 3. 16 - Borello Friction Model

3.2. Monitor Model

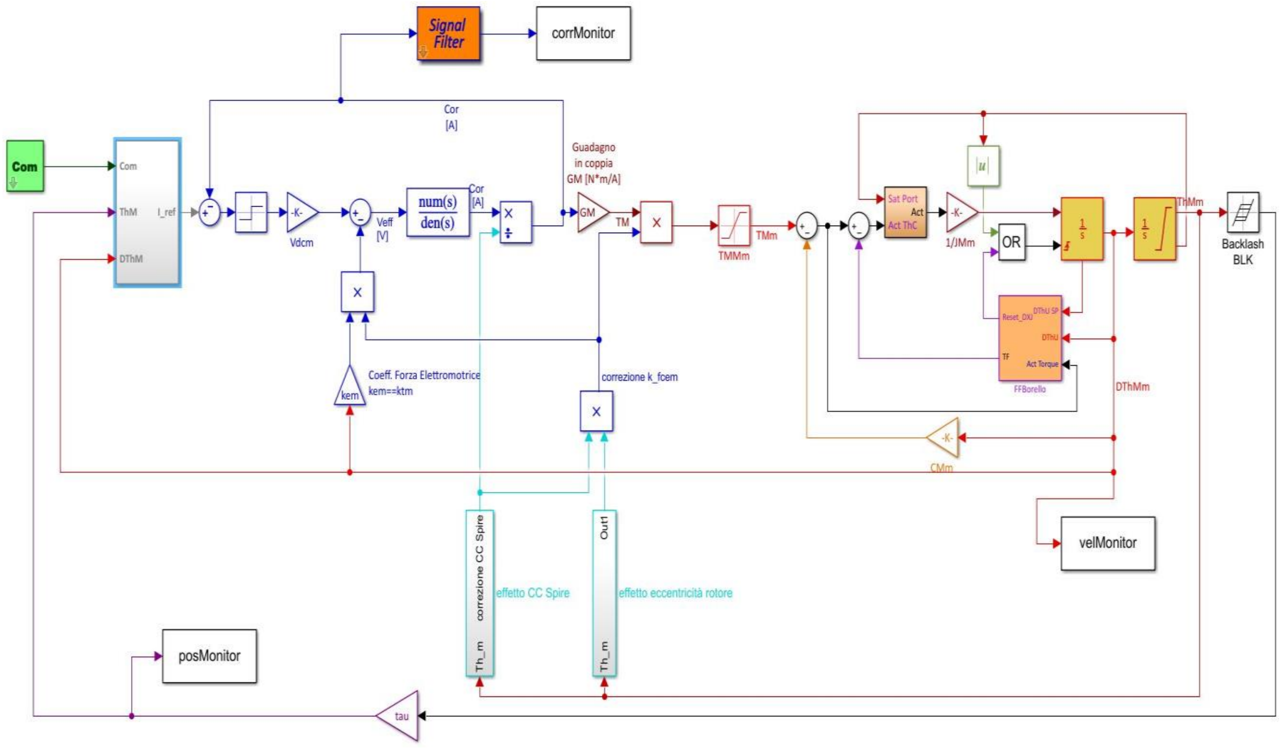

Figure 3. 17 - Monitor model developed in Simulink enviroment

A simplified monitor model is required to approximate as best possible the high-fidelity one, in order to be used for monitoring tasks. The aim is to use this Simulink file to detect the state of a real actuator compiling time by time the error between the currents required by the windings. The aforementioned reference model, into the *Phase current calculation* subsystem, evaluate the equivalent current I_{3equiv} for an equivalent single-phase actuator: this parameter is then compared with the current of the monitor model, because it simulates the behaviour of a single-phase actuator. In this way it is possible to relate the real actuator's parameters with the monitor ones, finding the fault issues.

It's composed by some parts, deepened below.

3.2.1. *Controller* subsystem

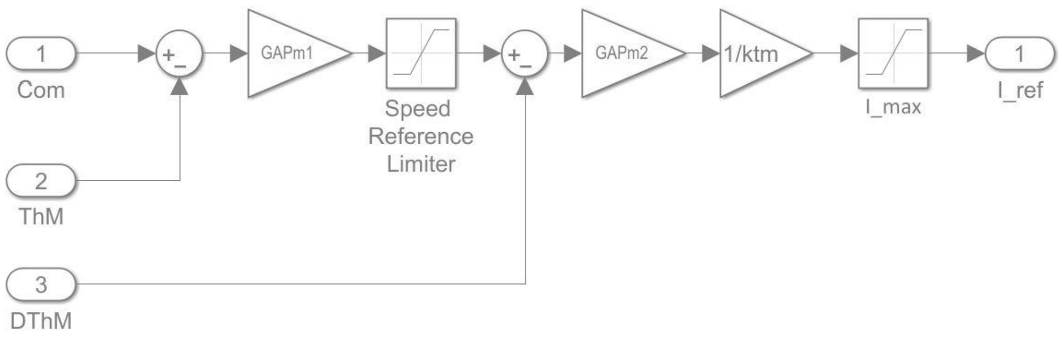

Figure 3. 18 - Controller subsystem of the monitor model

This subsystem receives as inputs the command given to the actuator (*Com*), the actual position (*ThM*) and the speed feedback (*DThM*). After the comparison between the commanded and the real position of the motor, with the *GAPm1* gain (set as 10^5) the signal is transformed in a speed and enters in a saturation block, which limits this value between its maximum and minimum. Once the reference speed is compared with the actual one, with a *GAPm2* gain -which substitutes a PID controller- and the division for the torque constant, it's possible to obtain the reference current. It's important to underline the strong similarity with the *BLDC Motor controller* block of the reference model: the only difference is that here the noise is not complained.

3.2.2. *Electromechanical* model

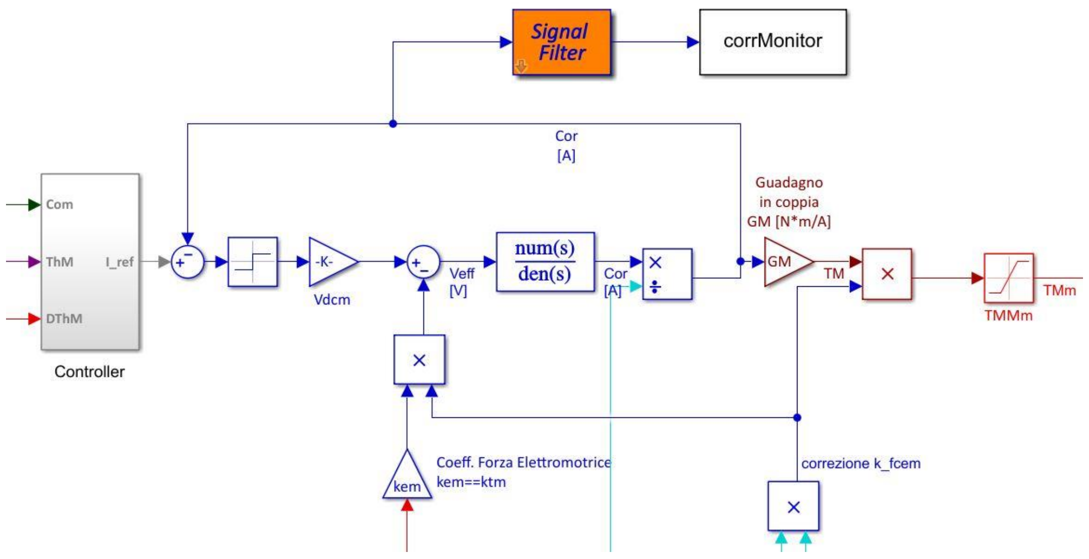

Figure 3. 19 – Schematic of the numerical algorithm implementing the simplified BLDC motor electromechanical model

In the monitor model, the transformation between reference current and reference torque executed by the actuator, is modelled with a simple structure into the main Simulink system.

The motor modelled is an equivalent single-phase in which the tension could only be up or down (if up, it can be positive or negative, depending on the rotation direction). The control is current feedback-based: the error between the two electric parameters is sent into a sign block, which identifies if the actuator requires a positive (output equal to 1), negative (output equal to -1) or neutral (output equal to 0) tension. To obtain the effective tension acting on the motor rotor, the output value is multiplied for the nominal inverter tension value (48 V) and then cleaned by that part dissipated due to faulty conditions: the main causes of power loss are the short-circuit of the stator windings and rotor eccentricity.

The effective tension is the input of the motor, represented by the first order transfer function:

$$TF = \frac{\frac{1}{R_m}}{\tau_{RLm} + 1} \tag{3.6}$$

where R_m is the resistance of the windings expressed in $[\Omega]$ and τ_{RLm} is the ratio between R_m and the inductance L_m, representing the time constant of the system. The output current takes care again of the short-circuit with a division for the percentage of the non-faulty coils because the absorbed current, if R_m and L_m decrease after a small winding short-circuit, must be greater than the nominal condition. The torque of the monitor model is subsequently obtained after the multiplication for the torque gain (*GM*=0.07322 $\frac{Nm}{A}$) and the limitation between the maximum and the minimum value with the saturation block.

3.2.3. *Mechanical* part

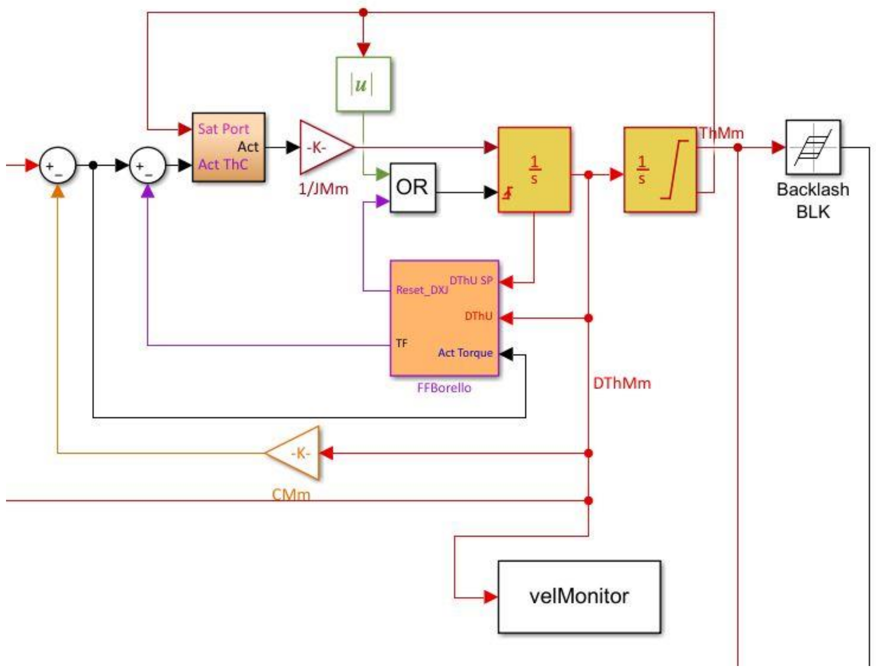

Figure 3. 20 - Mechanical section of the monitor model

The mechanical part of the model does not require any further explanation because is the same of the high-fidelity model.

4. Faults analysis and their implementation

In next paragraphs, after a brief introduction to the main types of progressive failures affecting EMAs, all faults considered are deeply discussed and their consequences to the system analysed. The dynamic response of faulty motor is described for a step and a chirp command: the first represents the open-loop response of the system, the second simulates the behaviour of the actuator in closed-loop (there are a lot of direction changes and the feedback becomes very important). The features of these two commands are summarized in the table below:

Table 3 - Step and chirp command features

Step command	
Initial amplitude	0 rad
Final amplitude	1 rad
Application time	0.01 s

Chirp command	
Initial amplitude	0.005 rad
Initial frequency	0 Hz
Final frequency	15 Hz

4.1. Introduction to faults

The precise definition of fault has been made by Isermann and Ballè [19]: "*A fault is an unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable/usual/ standard condition*". Depending on the seriousness of the fault impact, the system could be affected by a small reduction in efficiency to an overall failure.

Concerning the description of the fault modes on electromechanical actuators, the main issue is the lack of reliable statistic data, because the study of their behaviour in aerospace purposes is relatively new and not large enough to accumulate adequate informations.

A great number of aircraft used for the transportation still employs hydraulic actuation for primary and secondary flight controls, leaving to EMAs less important tasks, such as trim tabs actuation and speed break deployment. However, some recently-designed aircraft like Boeing 787 and Airbus 380 exploits EMAs in roles traditionally assigned to hydraulic or hydrostatic commands. In military field the situation is similar, but there are strong efforts to deploy electromechanical actuators in utility roles, such as landing gears, aerial refuelling doors and weapon bay doors. Concerning space vehicles, only few electromechanical actuators are used for small tasks such the motion of little robotic arms or the calibration of the antennas position. Hence, it’s important to study their response when they are affected by some types of faults, in order to maintain pre-determined levels of reliability and safety of the system.

Faults in EMAs are categorized into four groups, according to their location of occurrence in the system:

- *Mechanical or structural faults* are the main issue in electromechanical actuators. They are mainly caused by excessive loads, lubrification problems, unfriendly environmental condition and manufacturing defects. They mostly affect gear reducers and transmission.
- *Motor faults* are the next most important category of EMA faults. The high rotational rates at which a motor could rotate, may cause a temperature increasing leading to mechanical stress due to materials expansion. The main faults are windings short-circuit and rotor shaft eccentricity. In this work connection faults like cut or burned wiring and the presence of a foreign body in the actuator are not considered.
- *Electrical/Electronic faults* in the power and control systems are similar to the same type of faults in other aerospace systems. The main causes of their appearance are overheating, overcurrents, particle contamination responsible to short-circuit, vibrations and wear.
- *Sensor faults* could provoke the incorrect signal measurement, causing errors during the evaluation of the control law. Sensor faults could be also divided in *total* and *partial*: former type provides some informations not correlated with the physical value they are monitoring (e.g. lost contact with the surface or between wires), latter type produces signals still cleanable to obtain reliable data. The most widespread sensor faults are bias, scaling, drift, noise and intermittent dropout.

In tables below, are summarized the main fault modes described above with the relative probability and criticality depending on the component considered.

Table 4 - Mechanical and structural fault modes

Component	**Fault**	**Failure**	**Relative probability (1-10, low to high)**	**Relative criticality (1-10, low to high)**
Screw	Spalling	Severe vibrations, metal flakes separating	5	3
	Wear/backlash	Severe backlash	7	3
Nut	Spalling (mild)	Severe vibrations, metal flakes separating	5	3

	Backlash	Severe backlash	7	3
	Degraded operation	Seizure/disintegration	3	5
Nut	Binding/sticking	Seizure/disintegration	3	3
	Bent/dented/warped	Seizure/disintegration	1	5
Ball returns	Jam	Seizure/disintegration	5	8
Bearings	Spalling	Severe vibrations, metal flakes separating	5	3
	Binding/sticking	Seizure/disintegration	2	4
	Corroded	Severe vibrations, metal flakes separating, seizure/disintegration	2	5
	Backlash	Severe backlash, vibrations, disintegration	7	3
Piston	Crak(s), slop/play	Structural failure	1	10
Dynamic seals	Wear	Structural failure	4	6
	Structural failure	Structural failure	3	8
Static seals	Structural failure	Structural failure	2	8
Balls	Spalling/deformation	Severe vibrations, metal flakes separating	5	3
	Excessive wear	Backlash	7	5
Mountings	Crack(s), slop/play	Complete failure	1	7
Lubricant	Contamination	Seizure/disintegration	8	5
	Chemical breakdown	Seizure/disintegration	4	5
	Run-dry	Seizure/disintegration	3	10

Table 5 - Motor fault modes

Component	**Fault**	**Failure**	**Relative probability (1-10, low to high)**	**Relative criticality (1-10, low to high)**
Connectors	Degraded operation (increase of resistance)	Disconnect	5	6
	Intermittent contact	Disconnect	3	7
Stator	Stator coil fails open (results in degraded EMA performance)	Opening failure	4	4
	Insulation deterioration/wire chafing (reduced or intermittent current through stator coil or intermittent short	Short-circuit	5	5
Resolver	Coil fails open (can result in inaccurate position reports)	Opening failure	4	10
	Intermittent coil failures	Permanent coil failure	5	7
	Insulation deterioration/wire chafing	Short-circuit	5	7
Rotor and magnets	Rotor-magnets chemical bond deterioration	Complete magnet separation, likely leading to motor failure	2	10
	Rotor eccentricity	Bearing support failure	3	6

Table 6 - Electrical/Electronic faults

Component	**Fault**	**Failure**	**Relative probability (1-10, low to high)**	**Relative criticality (1-10, low to high)**
Power supply	Short-circuit	Short-circuit	5	10
	Open circuit	Open circuit	5	10
	Intermittent performance	Short-circuit or open circuit	5	8
	Thermal runaway	Dielectric breakdown of components, leading to open or short-circuit	6	10
Controller capacitors	Dielectric breakdown	Short-circuit or open circuit	4	8
Controller transistors	Dielectric breakdown	Short-circuit or open circuit	4	8
Wiring	Short-circuit	Short-circuit	5	10
	Open circuit	Open circuit	5	10
	Insulation deterioration/wire chafing	Short-circuit or open circuit	5	8
Solder joints	Intermittent contact	Disconnect	5	8

Concerning the implementation in a model, faults could be classified as additive or multiplicative, as depicted in figure 4.1 taken from [19]. Normally, additive faults describe better components' breakout, while actuator and sensor faults are best represented by a multiplicative action.

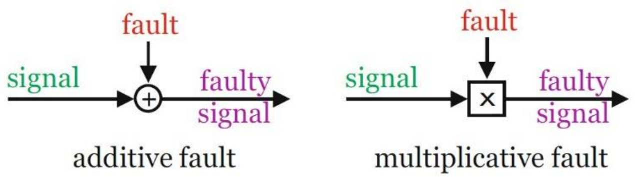

Figure 4. 1 - Additive and multiplicative fault

Furthermore, another important classification of faults could be made due to the type of appearance:

- *Abrupt faults*, which could have the most severe consequences, appear instantaneously without any pre-alert signal. If they affect the control or motor components, it could be very harmful;
- *Incipient faults* provoke slow changes in dynamic response's characteristic and they are less dangerous than the previous ones;
- *Intermittent faults* appear and disappear during normal life cycle and may be caused by partially-damaged components.

A possible example of their time trend is shown in figure 4.2 (taken from [19]).

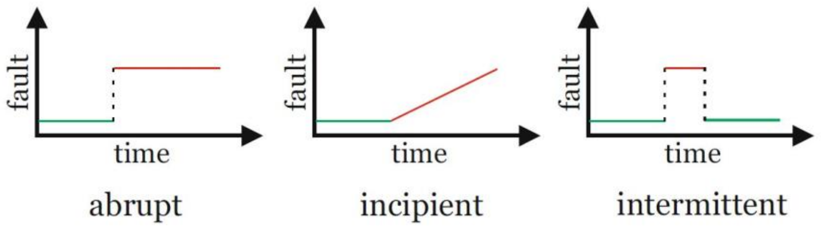

Figure 4. 2 - Different types of faults

4.2. Dry Friction

4.2.1. Description

The dry friction acting between mechanical components in relative motion can be schematically described as a dissipative force which opposes the motion and which varies according to the physical characteristics of the considered system (materials, type of connection, lubrications, etc) and to the forces exchanged between its moving parts. If friction fault is neglected, a jamming or break-down events could possibly take place on the actuator, with catastrophic consequences. During normal working activities, the Coulomb friction -employed in this model- states that in standstill conditions the friction force is lower or equal (in module) to the static friction value and that, otherwise, the force module has a constant value equal to the dynamic friction value.

4.2.2. Implementation

As aforesaid, in the *Borello* block described in chapter 3.1.4. and used in this work, a linear Coulomb friction has been developed and implemented. The nature of this phenomenon does not allow an entirely linear description, but the complexity of a non-linear model and the consequent long computational time suggests the utilization of numerical method in the time domain; howsoever, these numerical solutions are affected by shortcomings due to math models. The *Borello friction model* block [17]:

- Selects the correct sign for the friction torque identifying the direction of rotation;
- Evaluates the torque taking care about the load acting on the mechanical part;
- Selects the static condition or the dynamic one depending on the load;
- Verifies the undesired stop of the mechanical element;
- Calculates the eventual break away of the previously standstill mechanical element;
- Is able to simulate the dynamic of both reversible or irreversible actuators.

The corresponding mathematical model is entirely equal to the Coulomb one:

$$F_f = \begin{cases} F_{act}, & \dot{x} = 0 \cap |F_{act}| \leq F_{sj} \\ F_{dj} \cdot sign(F_{act}), & \dot{x} = 0 \cap |F_{act}| > F_{sj} \\ F_{dj} \cdot sign(\dot{x}), & \dot{x} \neq 0 \end{cases} \quad (4.1)$$

where F_f is the evaluated friction force, F_{act} is the active force applied to the system, F_{sj} the friction force in stick condition and F_{dj} the friction force in dynamic conditions.

4.2.3. Dynamic Response to a step command

Introducing in the reference model the friction fault, clearly the position, speed and equivalent current trends change. In figures from 4.3 to 4.5 these trends are depicted for the nominal conditions and for a friction fault growing from 1 to 3 times the nominal behaviour.

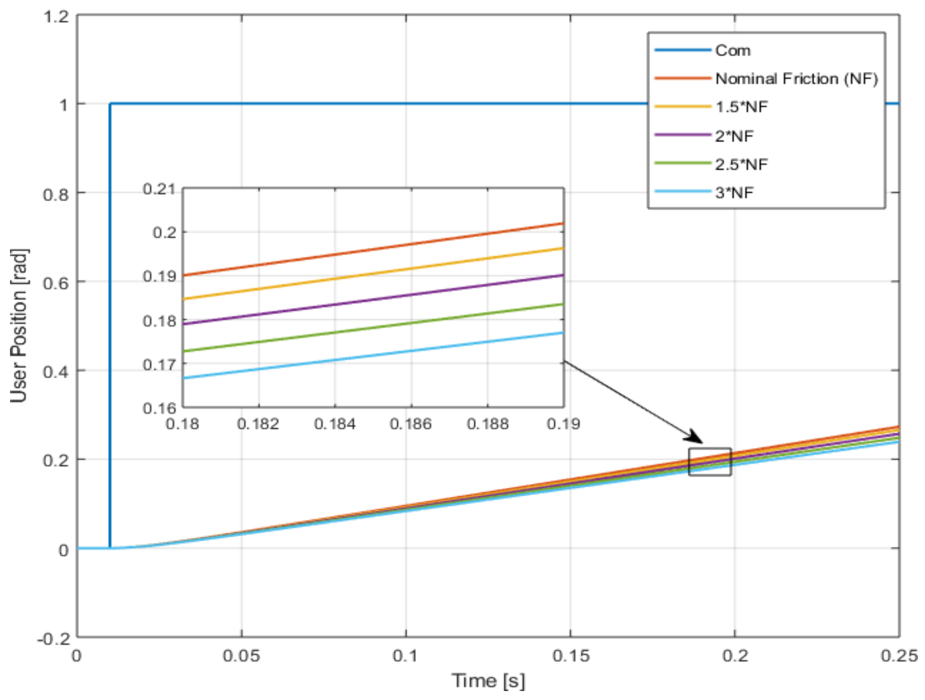

Figure 4. 3 - User position for a step command with a friction fault from 1 to 3 times the nominal conditions

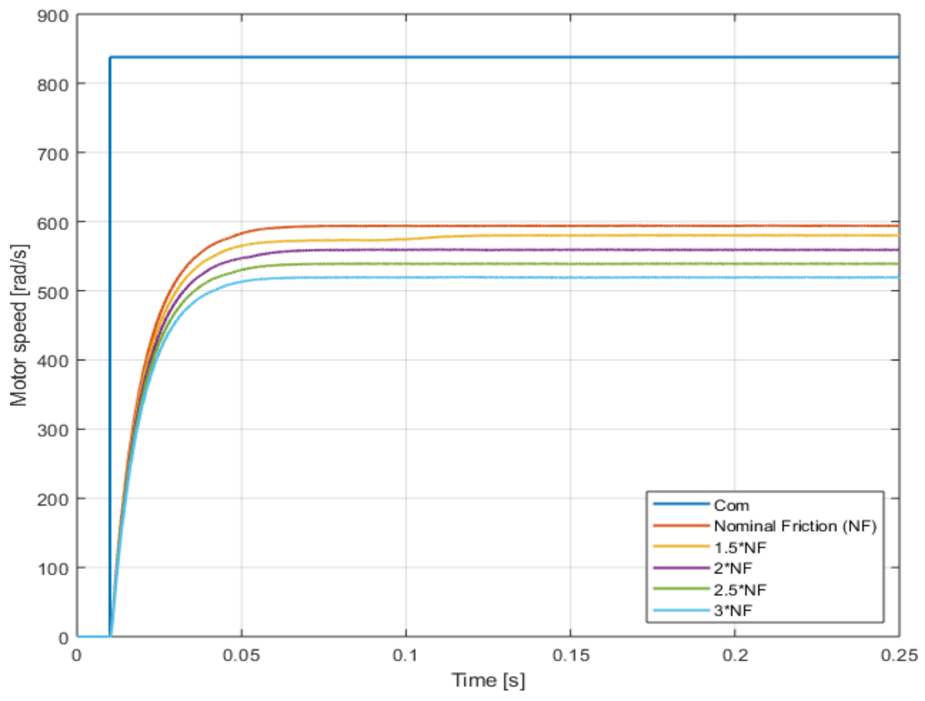

Figure 4. 4 – Motor speed for a step command with a friction fault from 1 to 3 times the nominal conditions

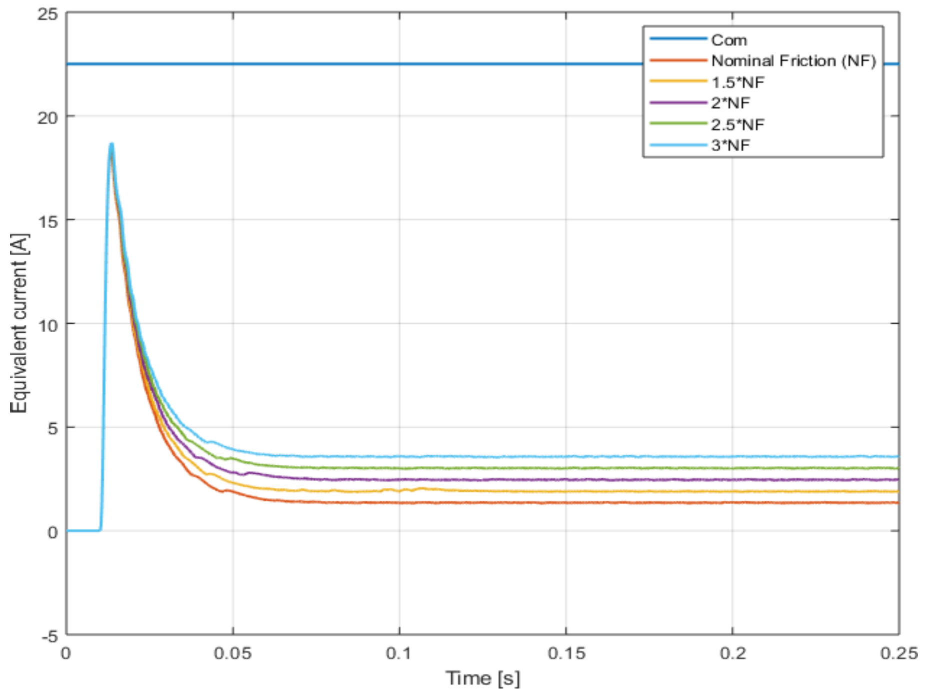

Figure 4. 5 – Equivalent single-phase current for a step command with a friction fault from 1 to 3 times the nominal conditions

The step command creates an open-loop situation, in which the controller is saturated and the error between set and feedback is null. Increasing the value of the friction from 1 to 3 times the nominal condition, it's possible to see in the box in figure 4.3 that the user position increases more slowly, because the motor has a higher resistant torque to overcome and because the dynamic friction coefficient (which is half of the static one) becomes higher. For this reason, also the maximum speed reachable by the motor decreases and the time constant of the equivalent first-order model increases if the friction grows up. The equivalent single-phase current reflects also that fact: the motor needs to be powered stronger to reach and maintain the maximum rotational speed. It’s clear that this condition is undesirable, because the power loss by Joule effect are very important and the temperature may cause problems of material expansion and then of jamming.

In figure 4.6 below, it’s possible to observe a comparison between the absorbed equivalent current in the reference and in the monitor models for the Nominal Friction condition (NF) and for a double value of friction. Even if the reference and monitor trends look pretty equal, a deeper inspection carried out by the box in the figure suggests that there is a little deviation between the two curves, with a percentual error from 2% to 12%. This fact is due to the calibration made for the optimization algorithms: only for the chirp command these two trends are very stackable. In next chapters, reliability of the monitor model is evaluated only for the chirp command for the reason just explained.

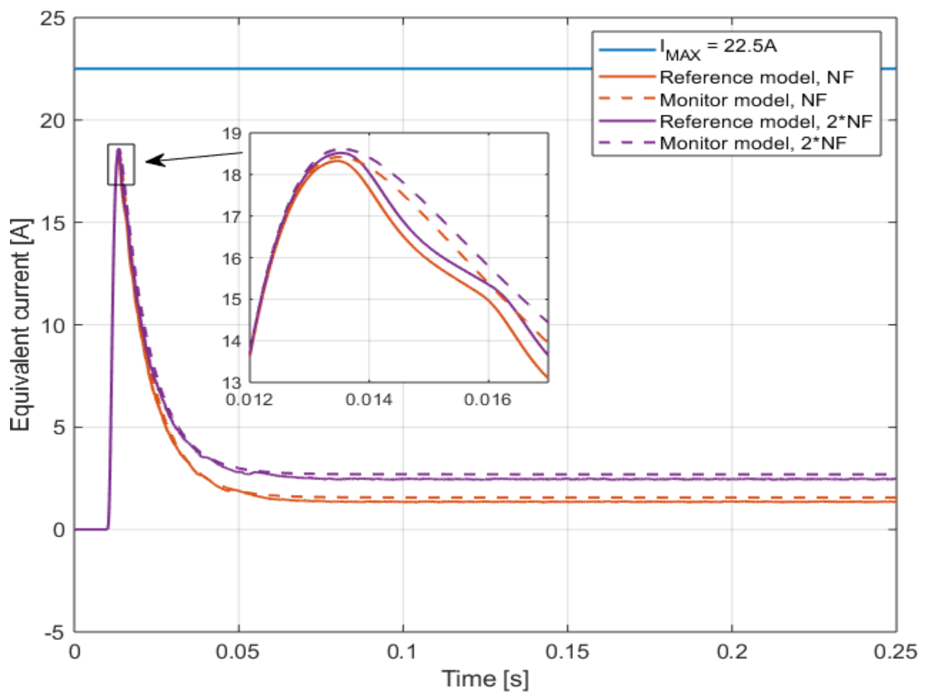

Figure 4. 6 - Current absorbed by the motor in the reference and the monitor models

4.2.4. Dynamic Response to a chirp command

For the chirp command is plotted the effect of the friction too: in this case a closed-loop situation is simulated, because the inversion of the sense of rotation requires to analyse the error between the set command and the feedback coming from sensors.

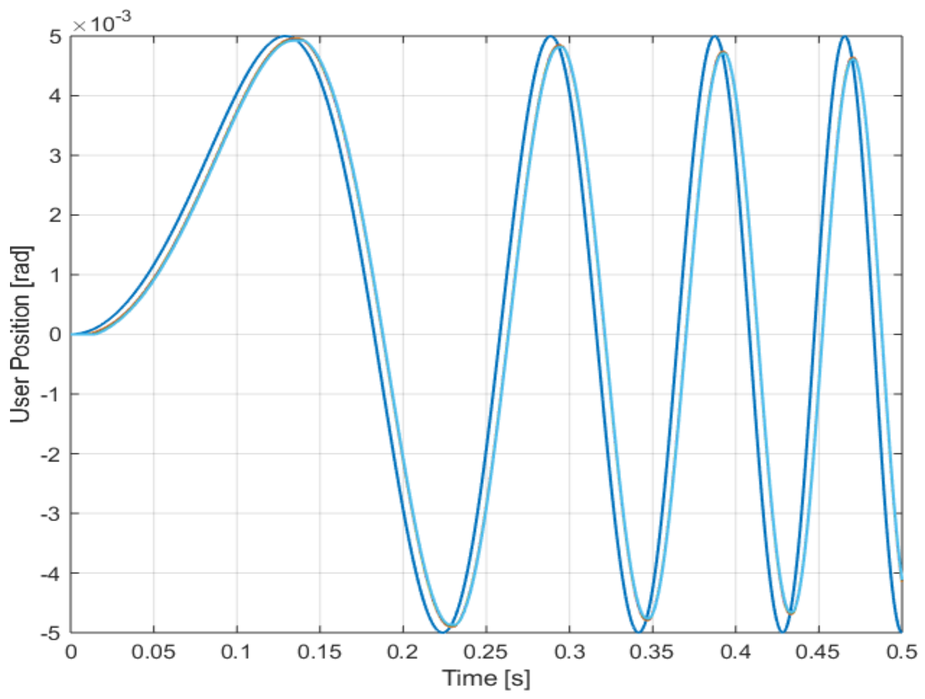

Figure 4. 7 - User position for a chirp command with a friction fault from 1 to 3 times the nominal conditions

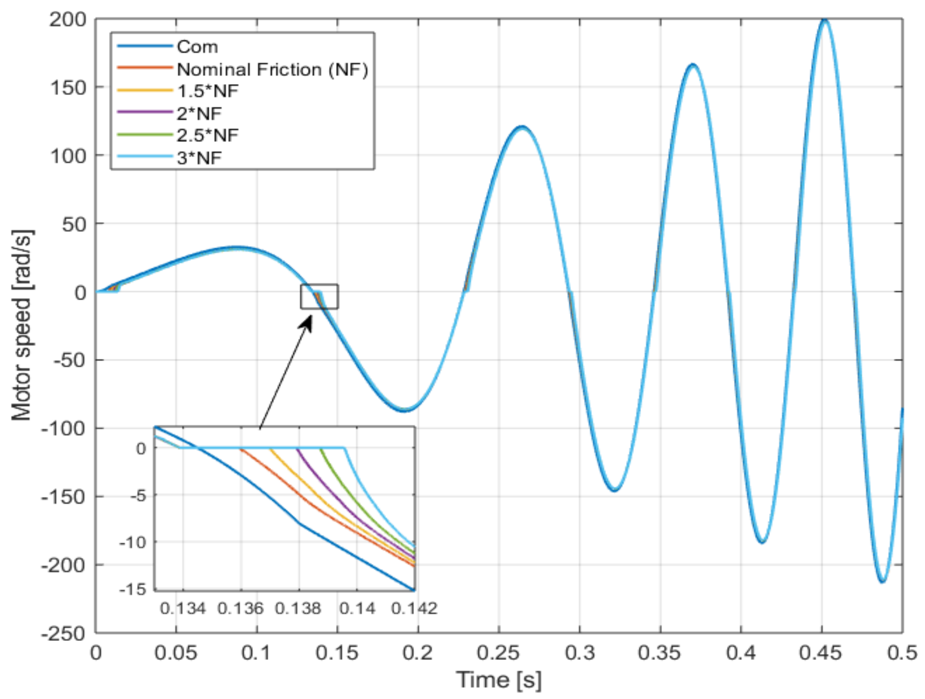

Figure 4. 8 - Motor speed for a chirp command with a friction fault from 1 to 3 times the nominal conditions

As it’s possible to see in figure 4.7, the user position is not affected by the friction fault, because the absolute displacement is very small, and the current could deliver the power needed. There is obviously a difference between the commanded position and the real one: the delay is due to the controller evaluation time, the dynamic of all the components and backlashes. The effect of the friction could be seen when the motor speed is close to zero (box on the left of the graph 4.8): increasing the friction means that the static-friction coefficient grows up, so when the direction of the speed changes, the motor takes more time to overwhelm that fault and provides the required power.

Concerning the absorbed current, it’s clear from figure 4.9 that to maintain the same displacement and speed, the current required to overcome the superior torque needs to be higher. Increasing the command frequency, the curves tend to become more similar, but the vertical parts corresponding to static friction situation – in the point of change sense of rotation - are always very different, meaning that the rotor has a higher difficulty to move in when the fault has a great value (it confirms what has been already stated from the speed graph).

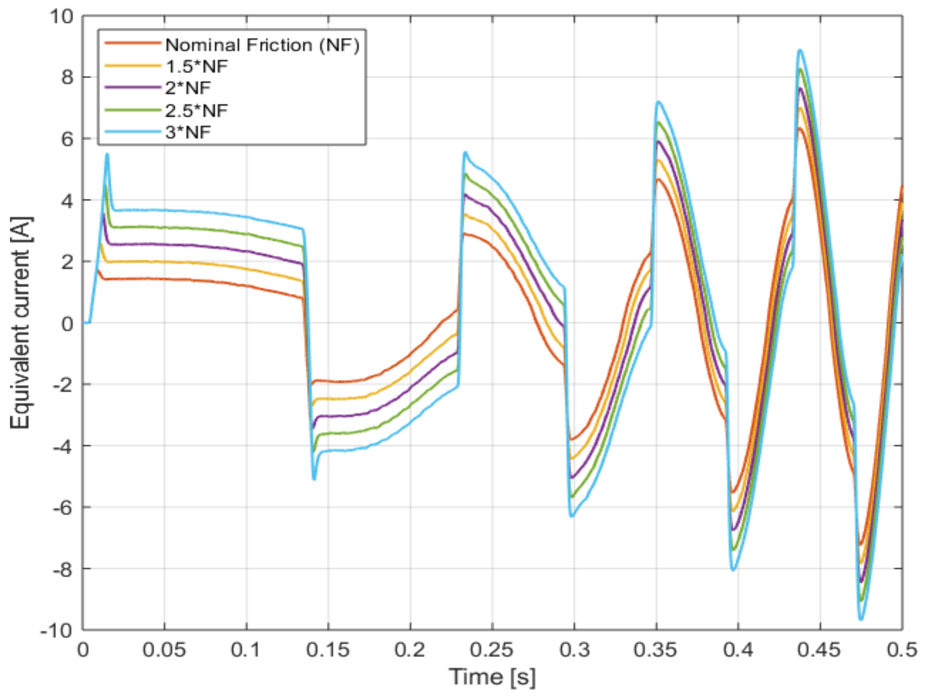

Figure 4. 9 - Absorbed current for a chirp command with a friction fault from 1 to 3 times the nominal condition

The figure 4.10 below indicates the difference between reference and monitor model in nominal and faulty condition. The two models are clearly very similar in both conditions, especially when the frequency grows up, indeed the error is contained between 1% and 6%. This fact is due to the aforementioned calibration: the chirp command will be used in the optimization algorithm as command, so the monitor has to approximate as best as it can the behaviour of the high-fidelity model for this type of command.

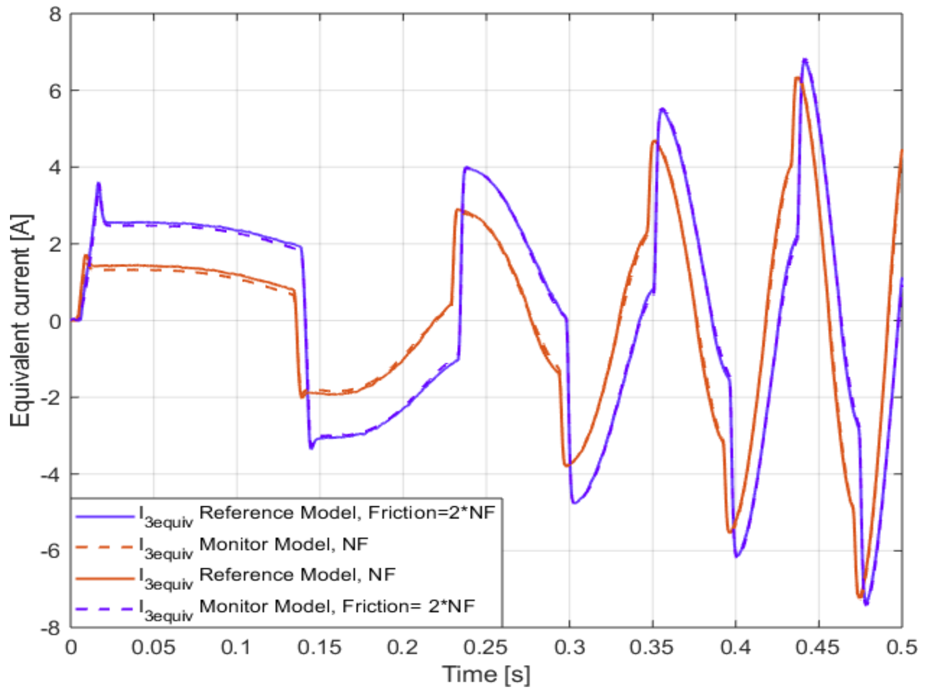

Figure 4. 10 - Equivalent single-phase current of reference and monitor model in nominal and faulty conditions

4.3. Backlash

4.3.1. Description

Usually the rotor shaft on electromechanical actuator systems is linked with the user shaft with a mechanical component. This device, during the normal working activities, is subject of mechanical wear which could cause severe problems on the system, such as the aforementioned friction, lubrification problems or backlash issue. These phenomena provoke firstly a superior power consumption, which leads to jamming or premature break-down of the motor if the problem is not fixed steadily.

The backlash is the mechanical play between two movable parts; in our case is the axial distance between the surface of the motor and user shafts, neglecting elasticity and Hertz theory. Often, the contact between the two power shafts is assured by a ball-screw system, in order to transform a rotating movement into a translational one. Ball-screws (represented in figure 1.11) are widely used thanks to the high efficiency -superior to 90%- but they need a severe design to avoid delays in motion transmission and to determine the correct pre-load [11]. During normal working activities, the wear increases the axial play between the two parts: in this way, backlash is a powerful indicator of the actual state of a system and eventually it suggests the substitution of a damaged ball-screw system.

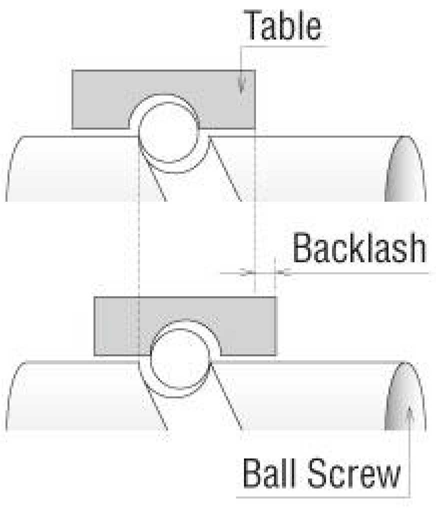

Figure 4. 11 - Backlash representation

4.3.2. Implementation

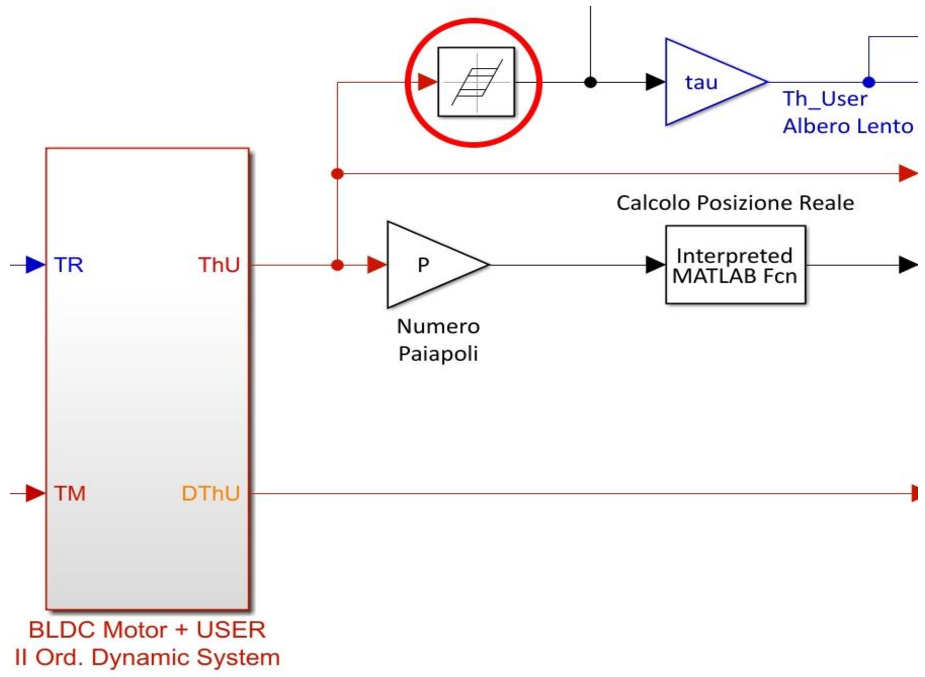

Figure 4. 12 - Backlash block in the reference model

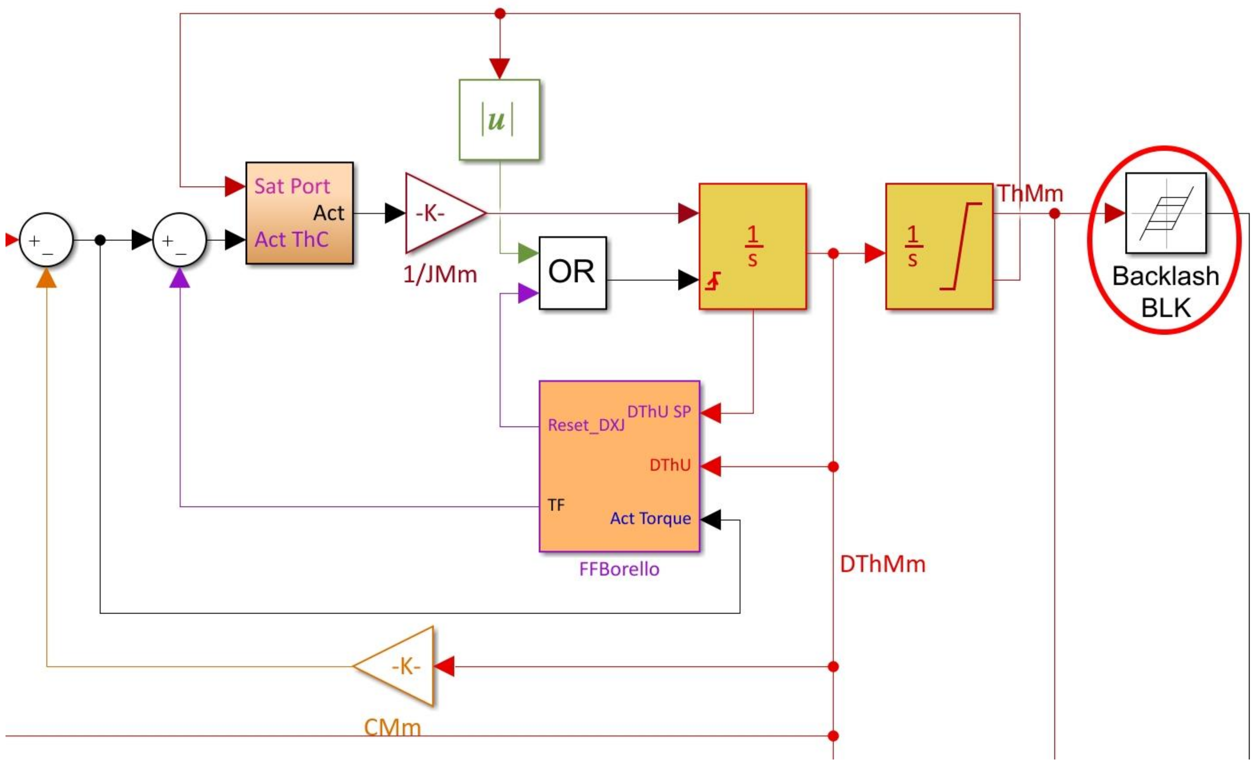

Figure 4. 13 – Backlash block in the monitor model

Both in reference and monitor models, backlash is introduced with a suitable block, which simulates the behaviour of a system with a determined mechanical play. Into the *BLDC Motor Dynamic Model* block in the high-fidelity model and in the main system of the monitor model it's possible to observe a block represented in figure 4.12 and 4.13, which introduces a dead-band on the fast shaft (before the multiplication for the τ). In this way it affects only the position feedback and maintains unchanged the speed feedback and, consequently, the control law of the system.

4.3.3. Dynamic Response to a step command

To study the effect of the backlash into the dynamic response of the EMA, the user position, motor speed and equivalent single-phase current are evaluated for the nominal backlash condition (equal to 0.005 rad) and for a backlash 2, 10, 50 and 100 times higher.

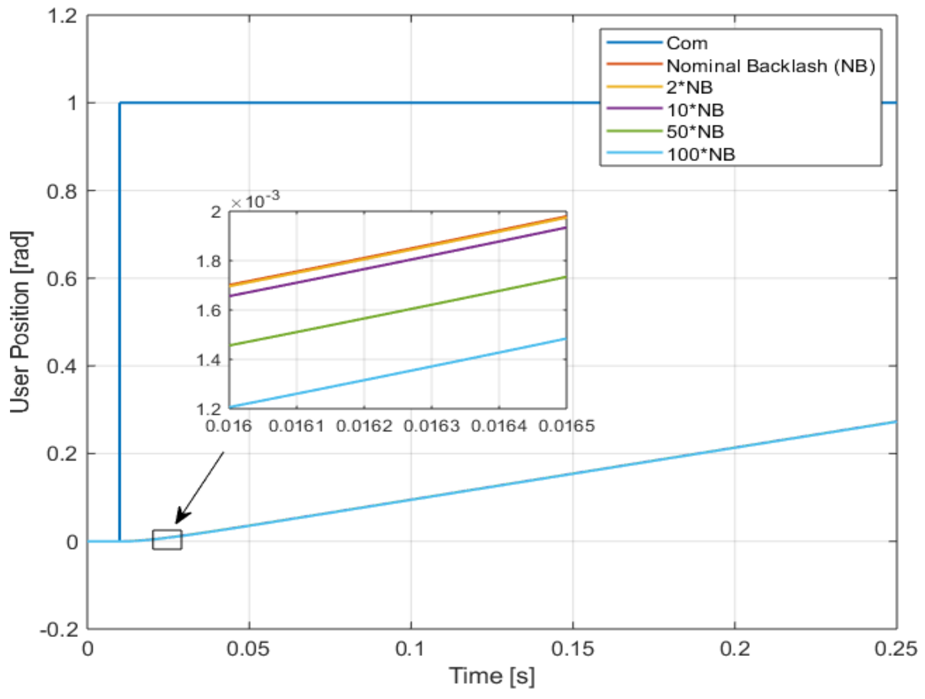

Figure 4. 14 – User position for a step command with a backlash fault from 1 to 100 times the nominal condition

Looking the figure above, it's possible to affirm that also a one-hundred bigger backlash fault than nominal condition does not change sharply the trend of the user position with a step command. There is only a slight delay at the start of the motion, observable by the box in figure 4.14. This event is coherent with the fact that there are not inversions of rotation sense, so the backlash acts only at the start of the rotation.

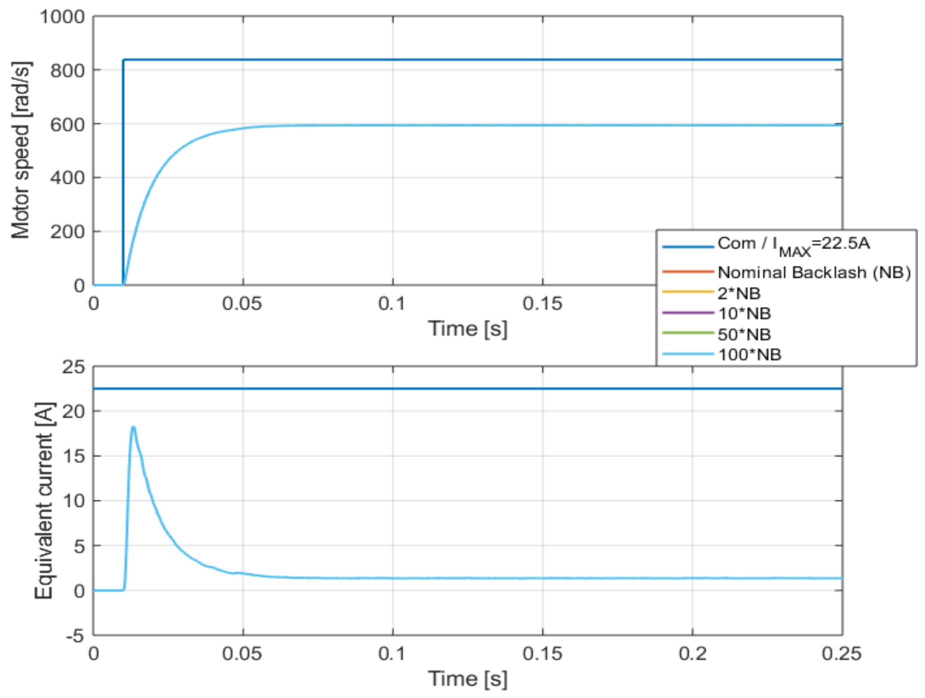

Figure 4. 15 – Rotational speed and absorbed current for a step command with a backlash fault from 1 to 100 times the nominal condition

The figure 4.15 confirms what has been already stated: the play between the two shafts acts only on the position feedback, allowing to maintain unchanged the speed and current trends due to the saturated controller. The conclusion is that a step command is not useful to isolate and study the backlash fault.

4.3.4. Dynamic Response to a chirp command

On the other hand, the chirp command has been very effective to investigate the mechanical play fault. The user position shown in figure 4.16 reveals what could be imaginable: when there is an inversion in the rotational sense, a system with a greater backlash have a superior delay. A high backlash fault delays the start of the motion until about the 275%: in a non-faulty system, the motor inverts the motion after about $8 \cdot 10^{-3}$ s, in a one-hundred times bigger backlash-affected system, the switch of the sense takes place after $22 \cdot 10^{-3}$ s approximately. By increasing the chirp frequency, the aforementioned percentage decreases.

The speed trend follows the same line of reasoning, because when the motor is not able to invert the sense of motion, the error between commanded position and feedback rises up, increasing consequently the speed in absolute value. When the motor manages to switch rotational sense, the positions of non-faulty and faulty systems become overlapped, thanks to the superior speed of the latter.

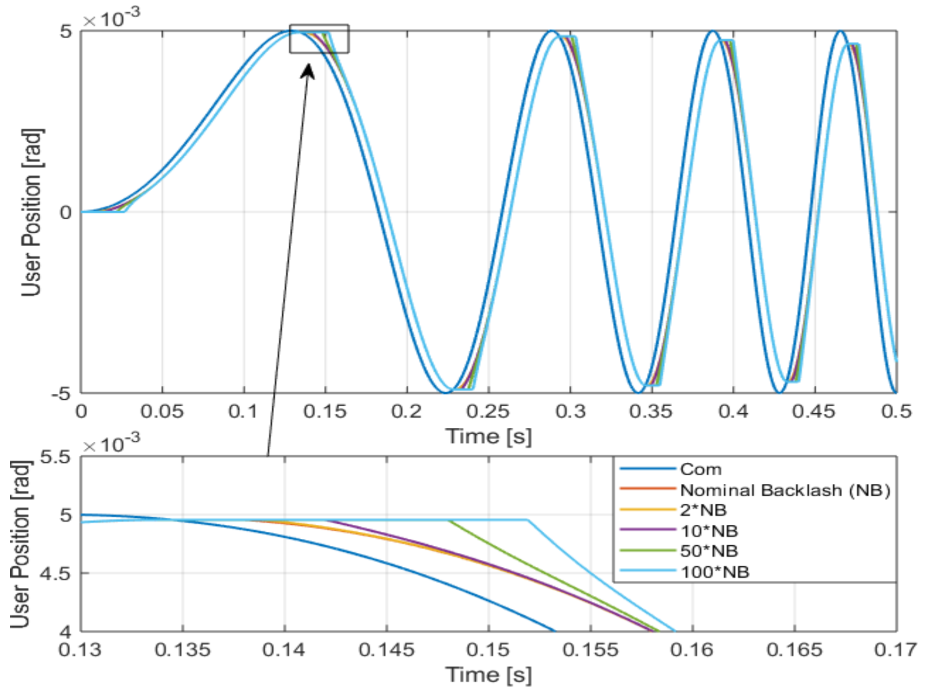

Figure 4. 16 - User position for a chirp command with a backlash fault from 1 to 100 times the nominal condition

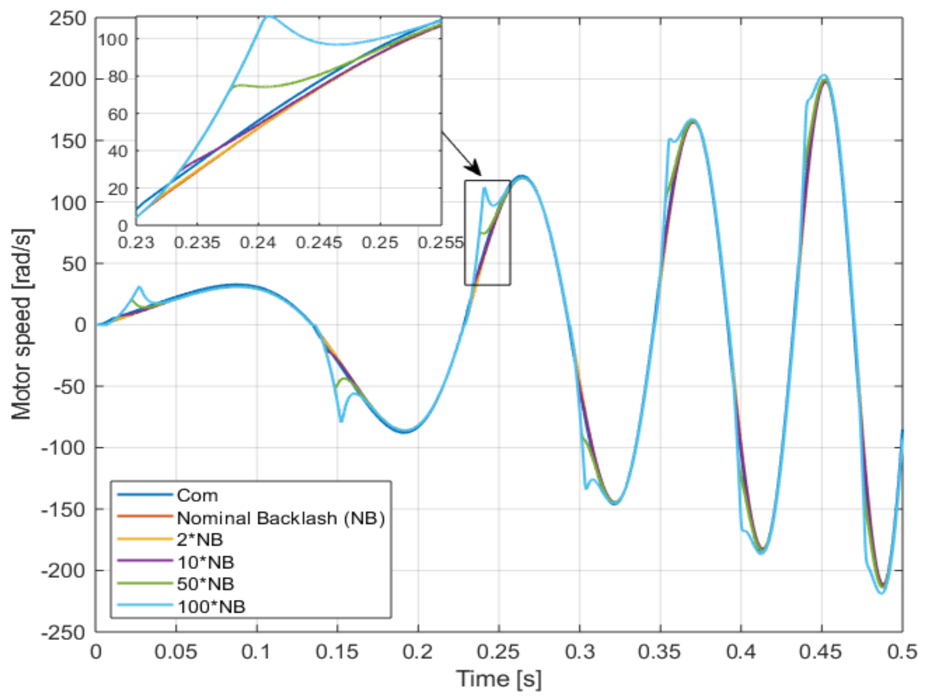

Figure 4. 17 – Motor speed for a chirp command with a backlash fault from 1 to 100 times the nominal condition

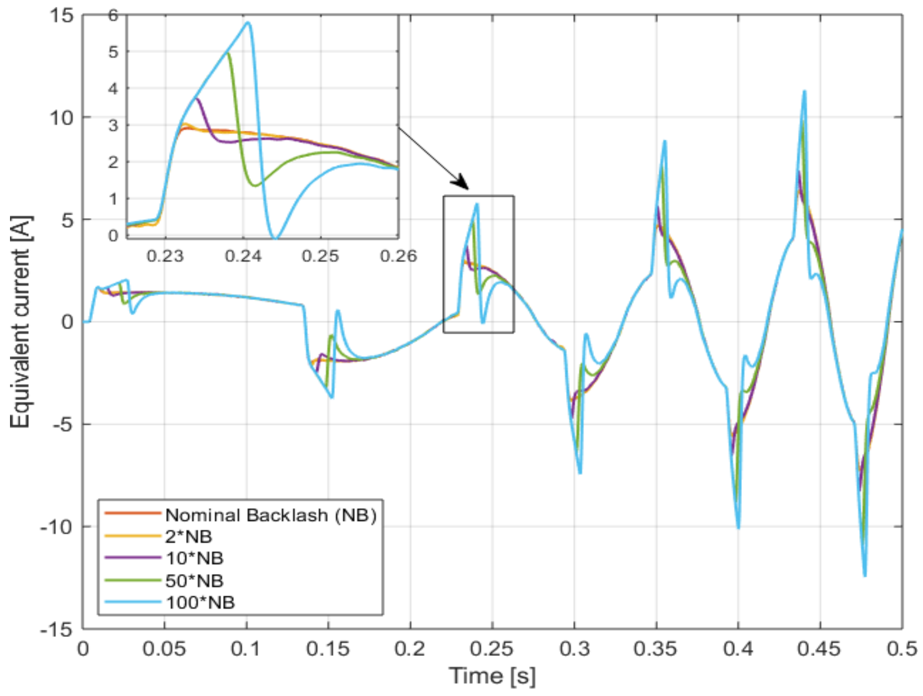

Figure 4. 18 – Equivalent single-phase current for a chirp command with a backlash fault from 1 to 100 times the nominal condition

To increase the speed when the motor needs it, the equivalent single-phase current grows up sharply and quickly. The current percentage increase from the nominal condition to one-hundred times bigger backlash is between 70% to 100%, causing overtemperature problems in the motor. After the initial peak, the current (in absolute value) drops because the speed able to move the system is higher than the nominal one, so the system needs to slow down. Increasing the command frequency, the current variation is always quicker and sharper.

From the figure in the next page, it’s possible to state that the monitor model approximates in a very good manner the parameters of the high-fidelity one, both in faulty (50 times the nominal backlash) and non-faulty cases.

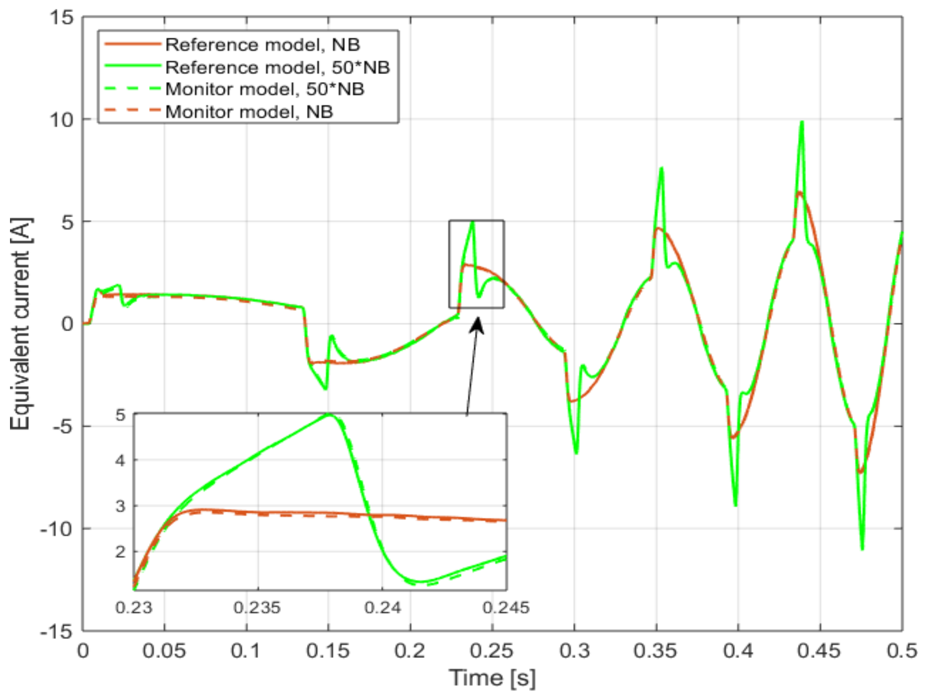

Figure 4. 19 - Equivalent single-phase current of reference and monitor models in nominal and faulty conditions

4.4. Short circuit

4.4.1. Description

During the normal working activities of a motor, repeated overcurrents and the subsequent overtemperatures could lead to a degradation of the polymeric insulating parts. If this event takes place in the stator close to the copper windings, possibly the phase coils come in direct contact, allowing the current to bypass a part of the winding. In this case, the resistance and the inductance of the coil decreases, so with the same tension acting on the winding, the current which pass through is higher. If the motor absorbs a greater value of current, this phenomenon tends to propagate by itself, increasing Joule effect and then leading to a complete failure of the system.

There are three possible short circuit modes:

- Between windings of the same phase (coil-coil)
- Between windings of a different phase (phase-phase)
- Between a winding and the iron of the stator core (phase-ground).

Usually a short circuit fault starts in the first mode, then it propagates and could possibly become the second or the third type. The last two types are not progressive: once they appear usually there is the sudden breakdown of the actuator caused by damaged internal parts. Hence, in this work only the predictable coil-coil short circuit is investigated.

4.4.2. Implementation

The implementation of the short circuit in the reference model can rely on the deeply detailed three-phase modelled actuator. From the workspace, it's possible to introduce the percentage bypassed windings per each phase (0% means that there is not short circuit, 100% means that an entire phase is bypassed, with a possible breakdown). The values of N_a, N_b and N_c are used in the calculation of the normalized counterelectromotive gain, because if the inductance decreases, also this force will become lower. In first approximation, we could state that:

$$k_{fcem} = G_M = \frac{\partial \Phi}{\partial \theta_m} = NA \frac{\partial \left(\int_A B \cdot \bar{n} dS \right)}{\partial \theta_m} \tag{4.1}$$

where A is the area of a winding, N the number of coils composing a winding and B the magnetic flux density of the rotor.

Hence, the N_i (percentage of short-circuit windings of the *i-th* phase) affects the calculation of:

$$K_{ei} = k_e \cdot N_i \tag{4.2}$$

$$R_{ij} = \frac{R_s}{2 \cdot (N_i + N_j)} \tag{4.3}$$

$$L_{ij} = \frac{L_s}{2 \cdot \left(N_i^2 + N_j^2\right)} \tag{4.4}$$

$$R_i = \frac{R_s}{2 \cdot N_i} \tag{4.5}$$

$$L_i = \frac{L_s}{2 \cdot N_i^2} \tag{4.6}$$

where:

- K_{ei} is the counter electromotive coefficient used in the calculation of the counter electromotive force (see figure 3.9);

- R_s and L_s are the phase-phase resistance and inductance of the non-faulty motor;
- R_{ij} and L_{ij} are the phase-phase resistance and inductance of the faulty motor;
- R_i and L_i are the coil-coil resistance and inductance of the faulty motor.

When the model runs in nominal conditions (N_a, N_b and N_c equal to 1), $R_i = \frac{R_s}{2}$ and $L_i = \frac{L_s}{2}$.

The implementation in the monitor model is slightly different due to an issue of recognition: the monitor model simulates the behaviour of a single-phase equivalent motor, but from the reference model it's impossible to distinguish which phase is in short circuit, because the three currents are evaluated for a single-phase equivalent actuator. Hence, in first approximation, for the monitor model the percentage of short circuit is the average between the three reference coefficients:

$$N_{equiv} = \frac{N_a + N_b + N_c}{3} \tag{4.7}$$

The electrical parameters are now calculated as:

$$R_{equiv} = R_{equivNC} \cdot N_{equiv} \tag{4.8}$$

$$L_{equiv} = L_{equivNC} \cdot N_{equiv}^2 \tag{4.9}$$

$$k_{fcem} = k_{fcemNC} \cdot N_{equiv} \tag{4.10}$$

$$G_{Mequiv} = G_{MequivNC} \cdot N_{equiv} \tag{4.11}$$

Where the subscript NC refers to nominal conditions.

With this design, when both short circuit and eccentricity are introduced in the reference model, the dynamic response can be wrong because of the current modulation. Both of them have the carrier frequency of $2\omega_m P$, where P is the number of pole-pairs. To overpass this problem, the rotor angular position is used to modulate the electrical characteristic of the motor. The modulating function is:

$$f(\theta_m) = \begin{cases} \dfrac{N_b + N_c}{2}, & \text{if } -\frac{\pi}{6} < \theta_e < \frac{\pi}{6} \\ \dfrac{N_a + N_b}{2}, & \text{if } \frac{\pi}{6} < \theta_e < \frac{\pi}{2} \\ \dfrac{N_a + N_c}{2}, & \text{if } \frac{\pi}{2} < \theta_e < \frac{5\pi}{6} \end{cases} \tag{4.12}$$

where $\theta_e = P\theta_m$ is the normalized electrical rotor angle, contained in a π range. The equation 4.12 allows to consider only two phase each time, when they are positive (24V) or negative (-24V). To implement it in the Simulink monitor model, the construction shown in figure 4.20 has been used.

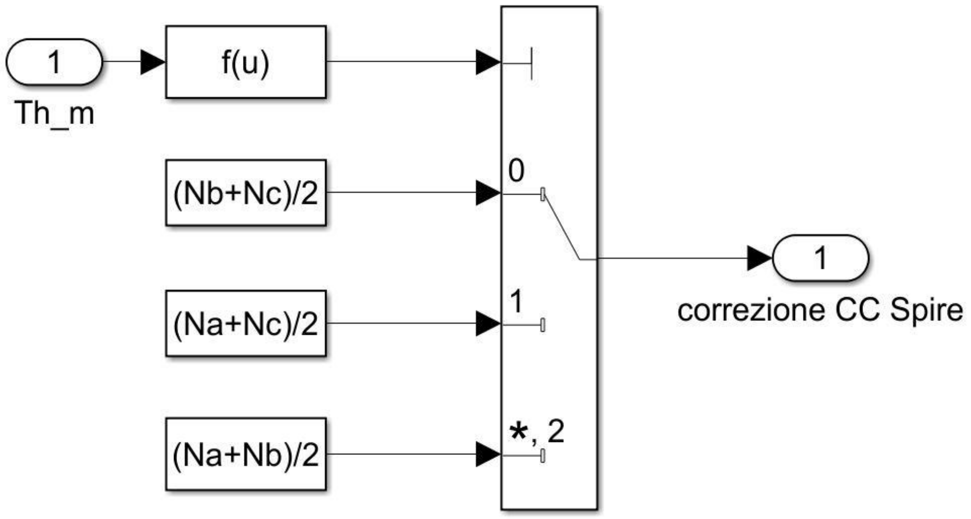

Figure 4. 20 - Simulink model of the modulating function

The rotor position is the input of the *active phase computation* block, which provides as output a value equal to 1, 2 or 3 depending on the active phases evaluating the expression:

$$f(u) = floor\left[3 * \left(\frac{Pu}{\pi} + \frac{1}{6}\right) - 3 \cdot floor\left(\frac{Pu}{\pi} + \frac{1}{6}\right)\right] \qquad (4.13)$$

where $u = \theta_m$ is the input of the block. Depending on the value given, the subsequent switch chooses the correct output to take care about the short circuit.

As it's possible to note in figure 4.21, the product of the *short circuit correction* block just described acts in three different points of the monitor model. Initially it modifies the counter electromotive coefficient as already stated in expression 4.10 (with the eccentricity fault, see paragraph 4.5), then multiplies the equivalent resistance R_{equiv} as described in equation 4.7 and finally decrease the torque gain G_{Mequiv}. The first order transfer function used in the monitor model (equation 3.6 chapter 3) does not allow the multiplication of the equivalent inductance L_{equiv}, which is contained in the characteristic time coefficient τ_{equiv} in the denominator of the transfer function. In first approximation, the equivalent inductance is evaluated as the average of the inductances of the three phases: this simplification does not affect results of the model, because the first order transfer function has a fast dynamic response. The implementation described, allows the monitor model to approximate in a better way the high-fidelity one: furthermore, it's possible to identify the faulty phase, but it has not a high importance for maintenance purposes.

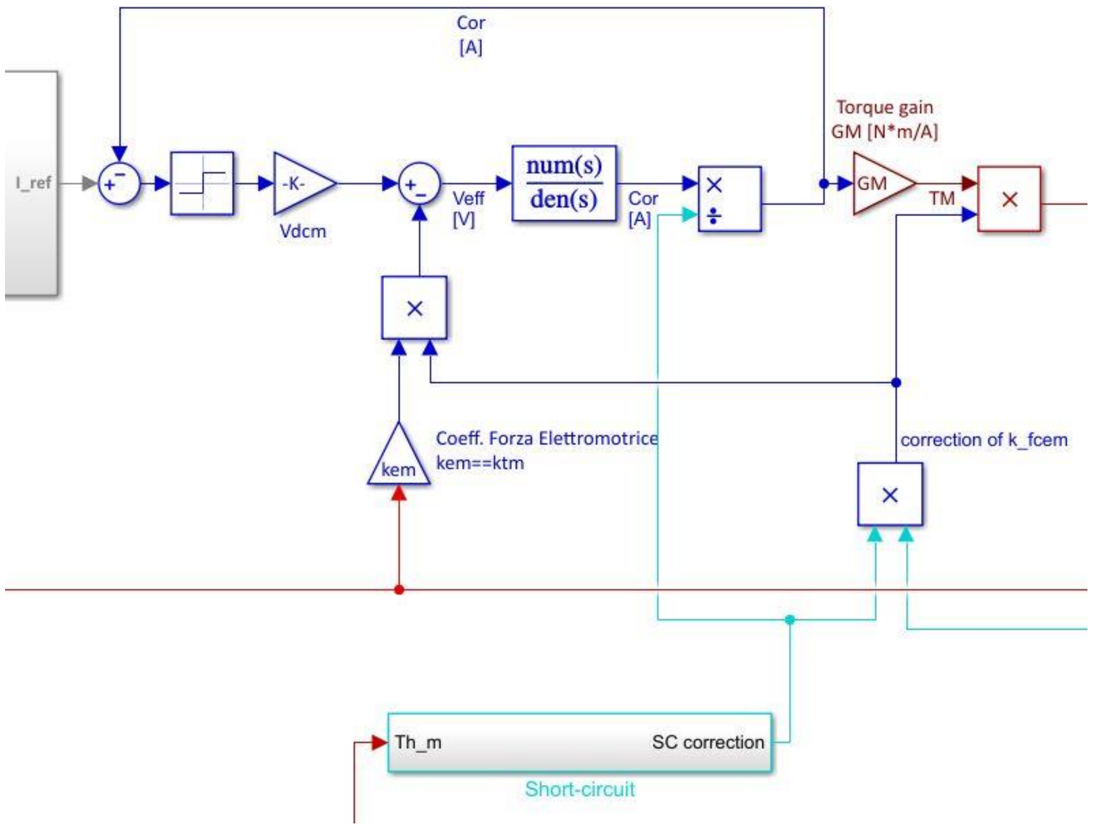

Figure 4. 21 - Correction of the short circuit in the monitor model

4.4.3. Dynamic Response to a step command

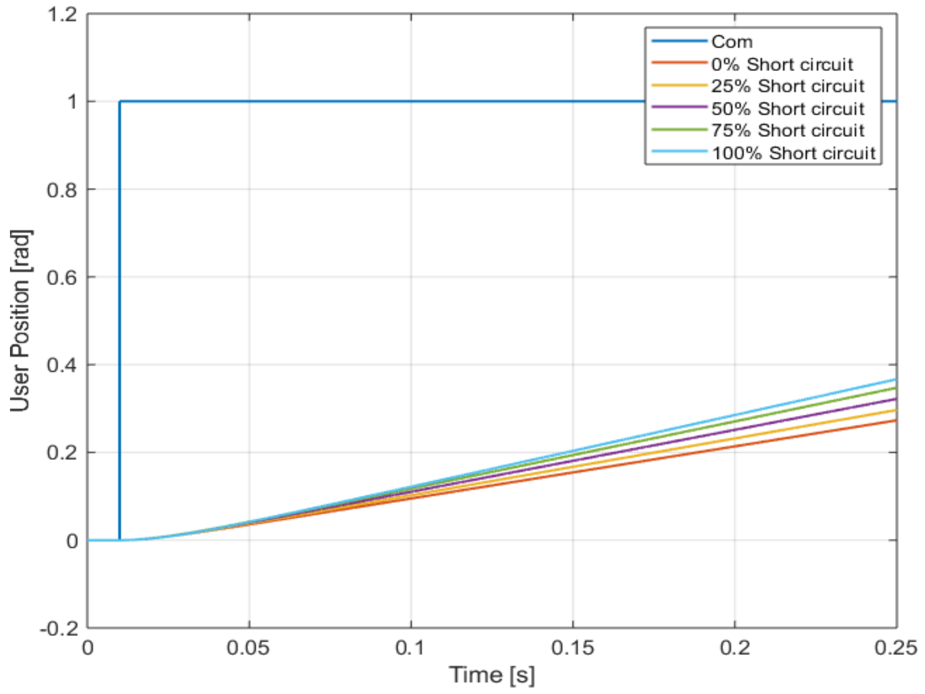

Figure 4. 22 - User position for a step command with a short circuit fault from 0% to 100% of coils bypassed

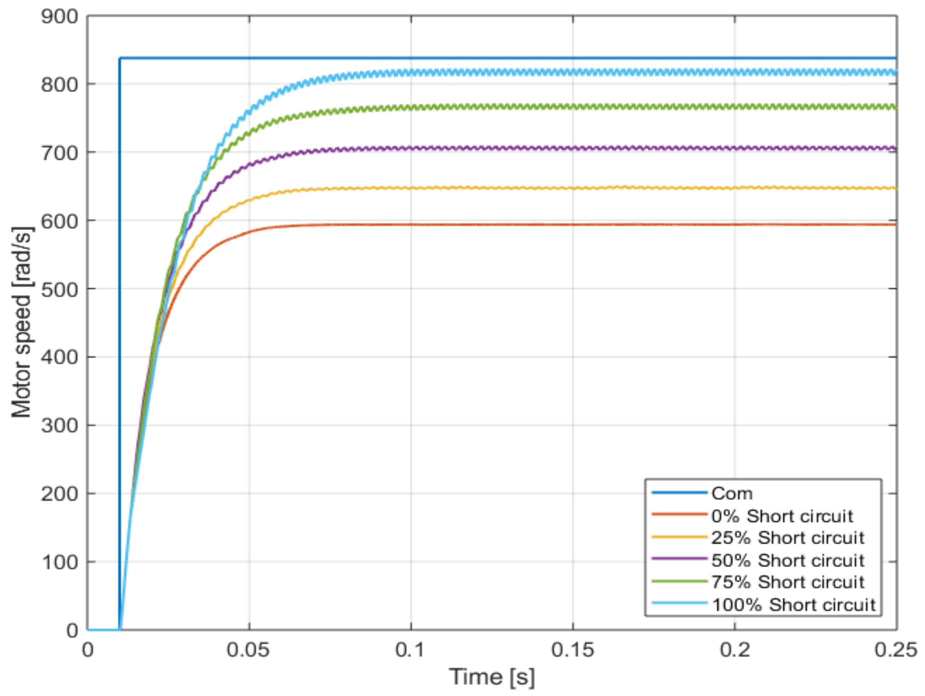

Figure 4. 23 - Motor speed for a step command with a short circuit fault from 0% to 100% of coils bypassed

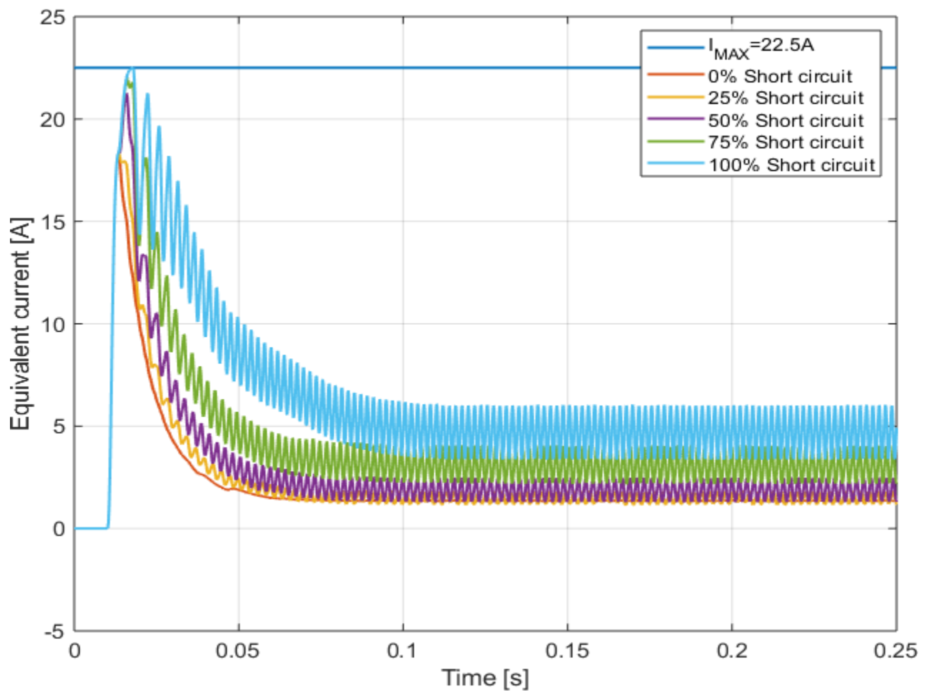

Figure 4. 24 - Equivalent single-phase current for a step command with a short circuit fault from 0% to 100% of coils bypassed

In the three figures above, the dynamic behaviour of the reference system powered by a step command is described. As already said in the *Description* paragraph, when a coil-coil short circuit takes place, the resistance and the inductance decrease, allowing the flow of a higher amount of current. The magnetic flux of the faulty phase is less than the others (because is directly proportional to the number of windings), therefore a motion anomaly takes place, both in the current and, consequently, in the torque. This irregularity is also fed by an event which occurs also in the nominal rotational behaviour, indeed when there is the commutation between two phases, the third current trend shows a local drop-off (*two-phase-on*, represented in figure 4.25).

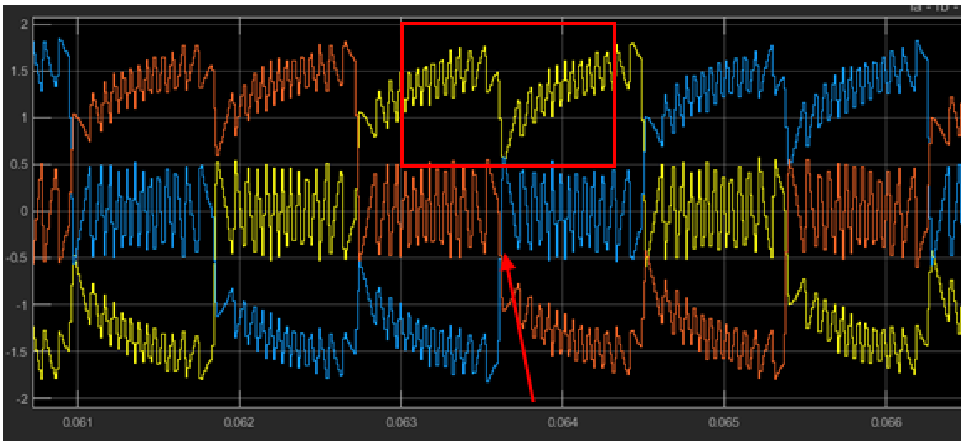

Figure 4. 25 - Two-phase-on phenomenon

Introducing the short circuit fault, this behaviour is worsened, and the motion becomes strongly intermittent, as represented in graph 4.23. At the beginning of the motion the faulty motor is slightly less performant, but at the steady state the speed is greater, even if very intermittent. The current depicted in 4.24 confirms what just explained: it becomes inconstant if the fault value grows up.

It's important to underline that it's impossible to obtain a short circuit of 100% with the reference model: the position, speed and current trend are evaluated for a number of non-faulty windings equal to 10^{-16} (the smallest number recognizable by Matlab).

4.4.4. Dynamic Response to a chirp command

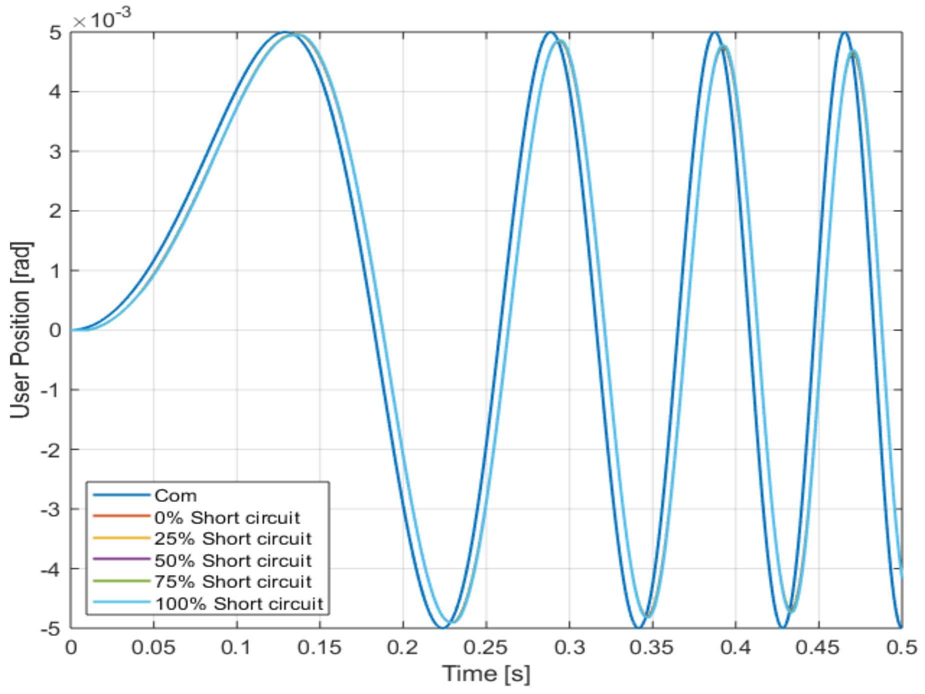

Figure 4. 26 - User position for a chirp command with a short circuit fault from 0% to 100% of coils bypassed

With a chirp command, the position and the speed graphs are not influenced by the faulty condition; the unique consideration is that there is a little delay on the speed when it changes sign, as already seen in friction fault.

The current reflects the treatise just exposed: when the faulty phase is fed (for 2/3 of the total simulation time) there is a current increase, when the other two phases are powered, the conditions are the same as the nominal ones.

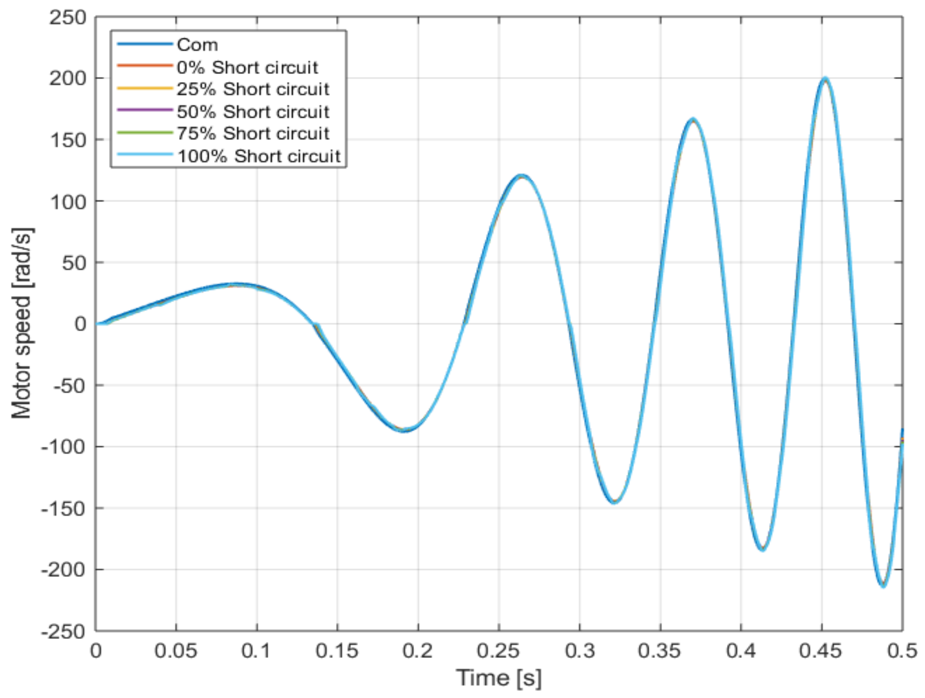

Figure 4. 27 - Rotor speed for a chirp command with a short circuit fault from 0% to 100% of coils bypassed

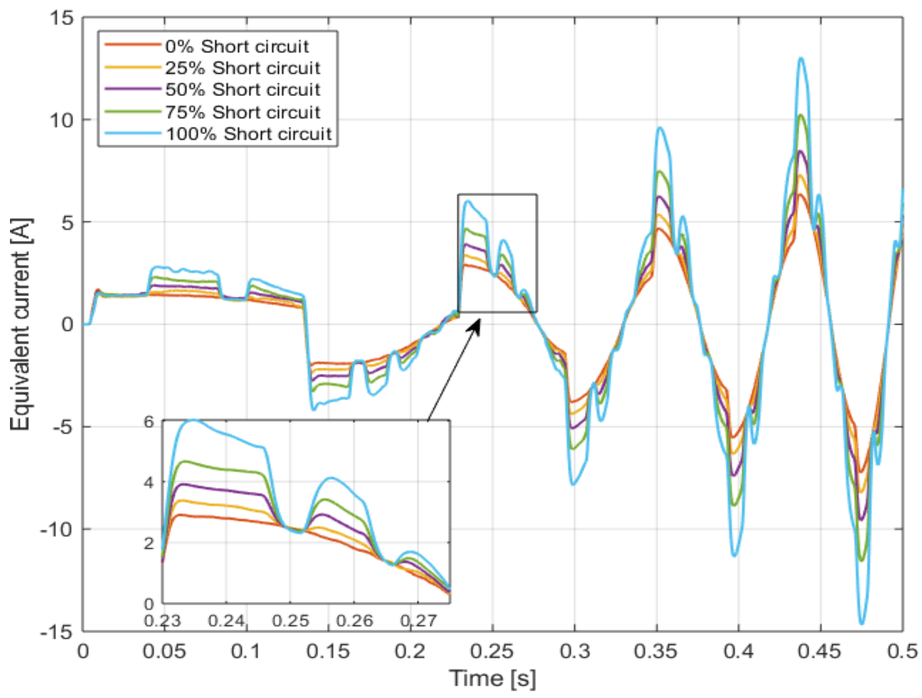

Figure 4. 28 - Equivalent single-phase current for a chirp command with a short circuit fault from 0% to 100% of coils bypassed

It has been also proved that the monitor model approximate in a very satisfying way the high-fidelity one, both in nominal and faulty conditions. The figure 4.29 shows the nominal equivalent current and the 25% of short circuit-faulty one: this value has been chosen because it’s the most probable to meet between all the values analysed before in an electromechanical actuator (with a greater fault usually the motor is substituted).

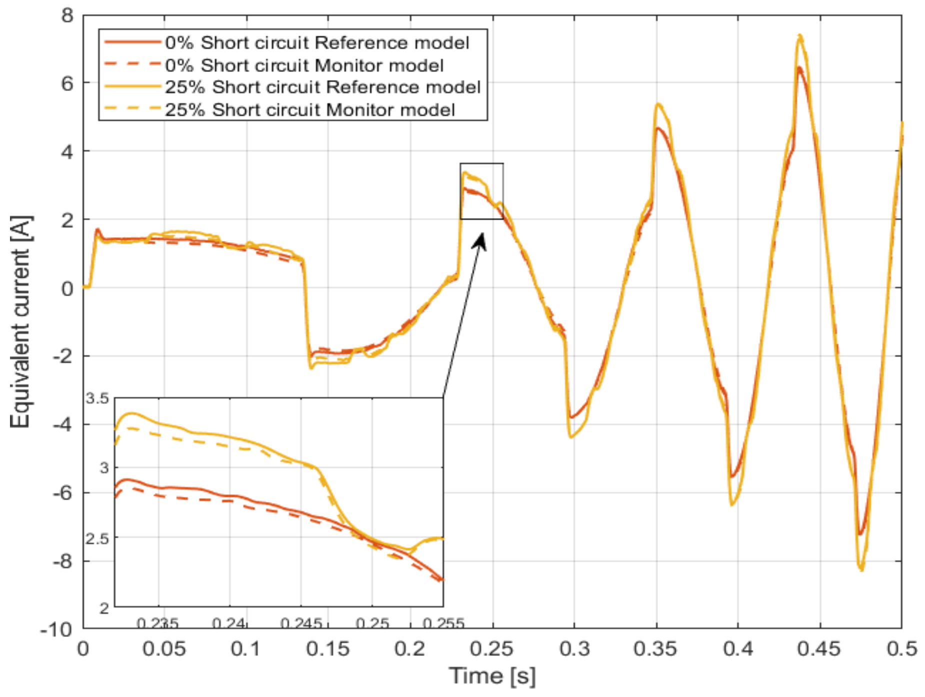

Figure 4. 29 - Equivalent single-phase current of reference and monitor models in nominal and faulty conditions

4.5. Eccentricity

4.5.1. Description

Bearings degradation, manufacturing tolerances, load unbalanced, improper mounting, bent rotor shaft and the mechanical wear which occurs during normal working life of actuator are the main causes of the eccentricity fault [24]. It could be static or dynamic (see figure 4.30): the former is the misalignment between the rotation axis and the stator axis of symmetry, the latter is the misalignment between the rotation axis and the rotor axis of symmetry. Static eccentricity causes a modification of the air gap between the stator and the rotor, dynamic eccentricity provokes harmful vibrations due to non-symmetrical distribution of rotating masses; both types of eccentricity add torque pulsations to rotational movement. In this work, only the static eccentricity is analysed, because is the only one that can be evaluated with the current and speed trends; for the survey of dynamic eccentricity an in-depth Simulink vibration model or a real test benchmark are required.

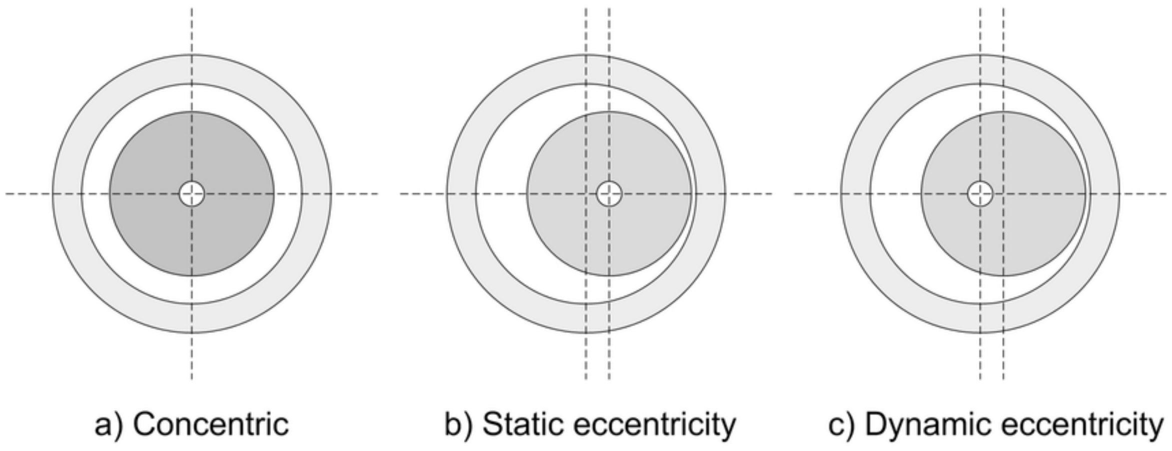

Figure 4. 30 - Concentric configuration (a), static (b) and dynamic (c) eccentricity

Considering the stator and the rotor as perfectly rigid bodies, the system under analysis is depicted in figure 4.31. The expressions of the two circumferences are:

$$x^2 + y^2 = R_r^2 \tag{4.14}$$

$$(x - x_0)^2 + y^2 = R_s^2 \tag{4.15}$$

Combining the (4.14) and (4.15) and introducing the polar coordinates:

$$\rho = R_r \tag{4.16}$$

$$\begin{cases} x = \rho\cos(\theta_r) \\ y = \rho\sin(\theta_r) \end{cases} \tag{4.17}$$

It's possible to obtain (4.18):

$$\rho^2 - 2\rho x_0 \cos(\theta_r) + x_0^2 - R_s^2 = 0 \tag{4.18}$$

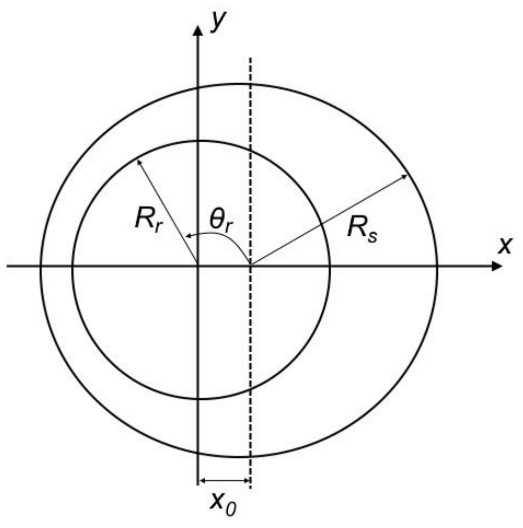

Figure 4. 31 - Rotor reference system for air gap definition

The air gap g can be measured from the centre of the reference system and, approximating a square root with its Taylor series at second order, it's possible to write:

$$g \cong x_0 \cos(\theta_r) + g_0 \tag{4.19}$$

$$g \cong g_0 \cdot (1 + \zeta \cos(\theta_r)) \tag{4.20}$$

where $g_0 = R_s - R_r$ is the air gap in non-faulty conditions and $\zeta = \frac{x_0}{g_0}$ the ratio between the misalignment and the g_0. Looking deeper in the situation, the magnetic flux can be written with the Hopkinson's law:

$$F_{mm} = \Phi \Re \tag{4.21}$$

where

$$\Re = \frac{l}{\mu_0 \mu_R S} \tag{4.22}$$

is the reluctance of the system, F_{mm} is the magnetomotive force and S the surface of the rotor interested by the magnetic flux. In our case, depicted in figure 4.32 (taken from [3]), the magnetic flux through the air gap, using (4.22) and (4.23), is:

$$\Phi = \frac{F_{mm}}{\frac{g(\theta_1)}{\mu_0 S} + \frac{g\left(\theta_1 + \frac{\pi}{P}\right)}{\mu_0 S}} = \frac{F_{mm}\mu_0 S}{g(\theta_1) + g\left(\theta_1 + \frac{\pi}{P}\right)} \tag{4.22}$$

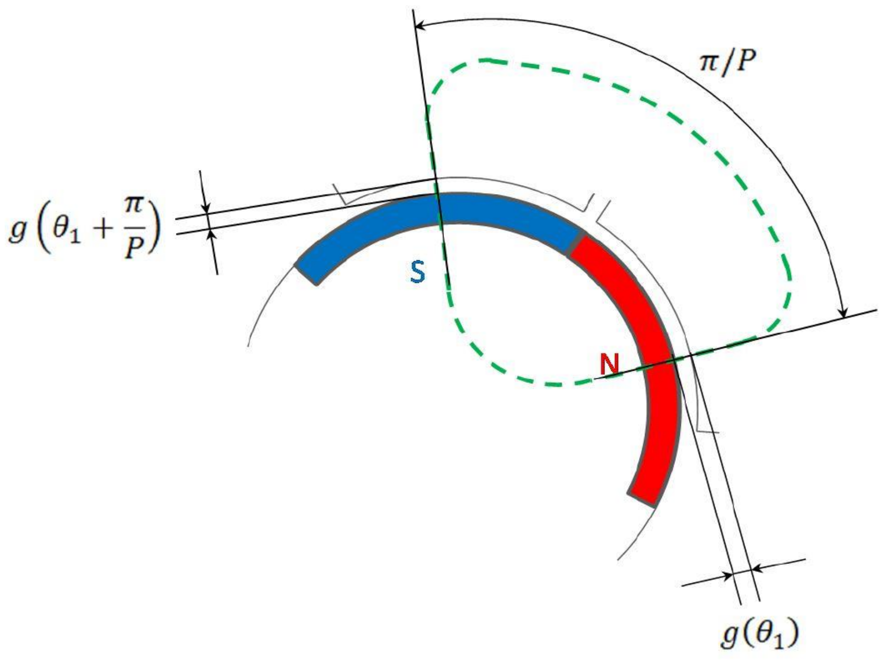

Figure 4. 32 - Magnetic circuit through the air gap

It’s important to underline that, given the 2π periodicity for the air gap, it affects the magnetic flux only if the motor has a number of pole-pairs greater than one.

When an eccentricity fault takes place, unbalanced magnetic forces are created, because the permanent magnets are closer to windings, generating an attractive force acting on the rotor. When eccentricity becomes too large, the resultant of the aforementioned forces could cause the stator-to-rotor rub, causing possible harmful damages. [24]

4.5.2. Implementation

In the reference model the implementation of the eccentricity fault is carried out in the *BLDC Motor Electro-Mechanical Model* block, with the calculation of the counter electromotive force already expressed in equation (3.3) and reported below:

$$k_{fcem} = k_e(\theta_m) \cdot \left(1 + \zeta \cos\left(\theta_m + \frac{2 \cdot (i-1)}{3}\pi\right)\right) \tag{3.3}$$

The eccentricity fault, modifying the air gap between the stator and the rotor, changes the magnetic coupling between these two parts: in this way, the counter electromotive force gain and the torque gain are directly dependant from the angular position. As shown in [22], this is a suitable method to avoid the implementation of complex and heavy FEM analysis.

The eccentricity has a light effect on the response of the actuator (see next paragraph): the torque and the counter electromotive force coefficients increase and decrease depending on the angular position, but in a 360° angle the average disturbance value is null.

In the monitor model, ζ is not used and it is replaced by the coefficient Z: the value of the fault (Z) is limited from 0 to 0.42, corresponding to a 0-1 values of ζ, thanks to the relations below taken from [3]:

$$Z = 0.42\zeta \tag{4.23}$$

$$K'_{fcem} = K_{fcem}\big(1 - Z(\cos(P\theta_m + \phi) + sawtooth(6P\theta_m - \pi)\sin(P\theta_m - \pi))\big) \tag{4.24}$$

Simulink environment does not allow the implementation of the $sawtooth$ function, so it has been replaced with:

$$sawtooth(x) = 2\left(\frac{x}{2\pi} - floor\left(\frac{x}{2\pi}\right)\right) \tag{4.25}$$

The relationship (4.24) with the correction explained in (4.25) is introduced in the monitor model combined with the short circuit fault, as shown in figure (4.33) and (4.34).

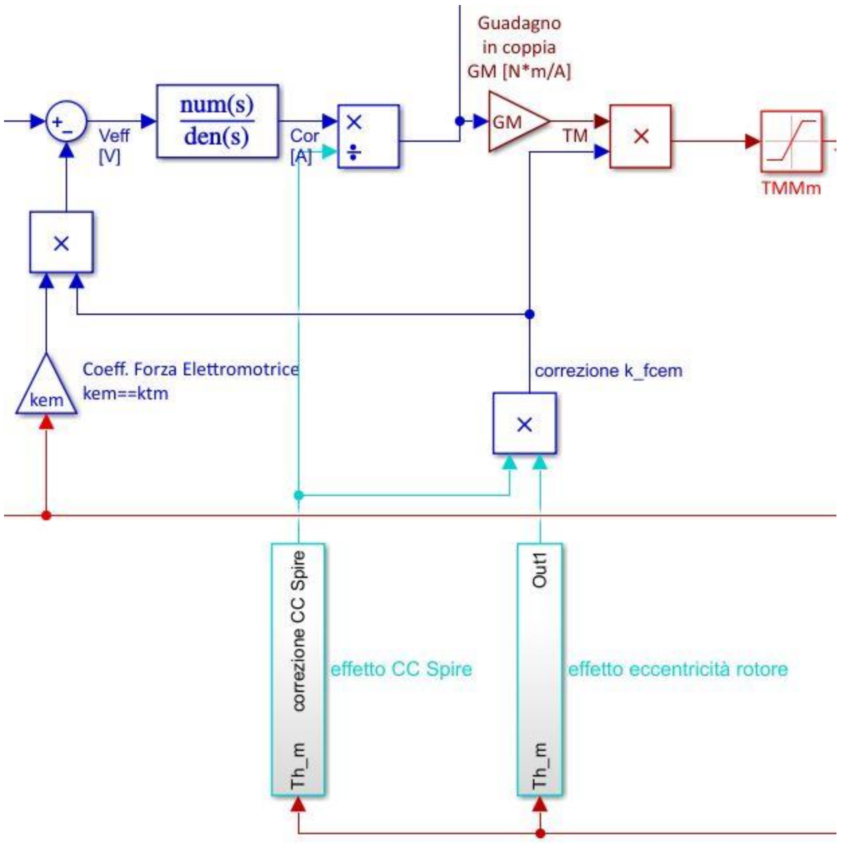

Figure 4. 33 - Implementation of the eccentricity fault in the monitor model

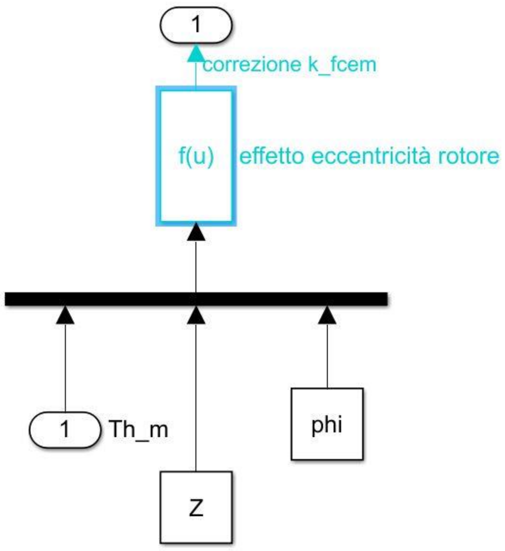

Figure 4. 34 - Eccentricity modification block

4.5.3. Dynamic Response to a step command

Also for this fault, an investigation regarding the dynamic response has been carried out for values of ζ from 0 to 1, equal to a situation in which rotor and stator touch each other.

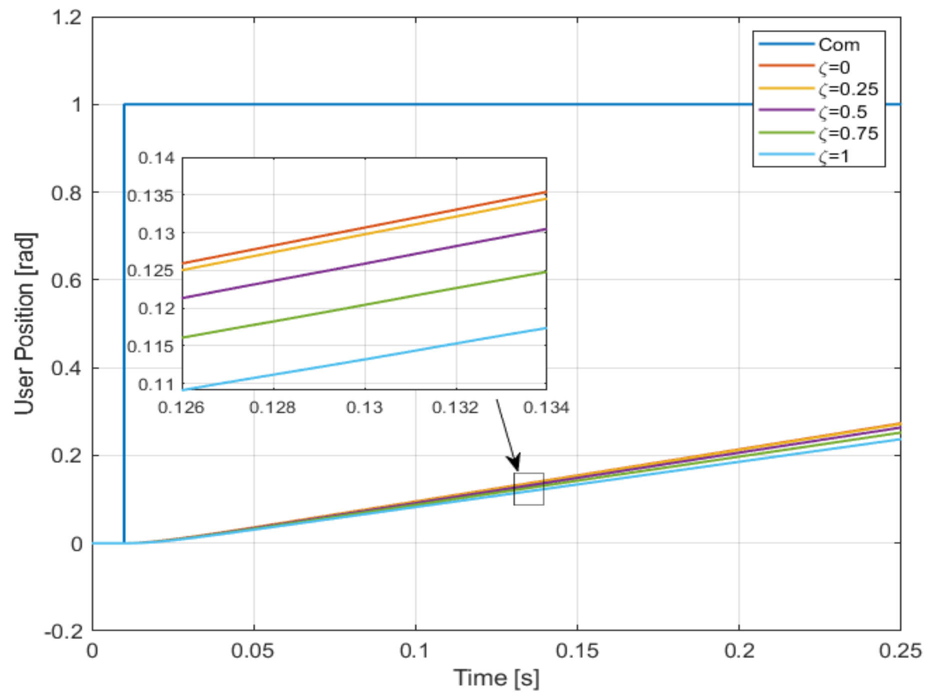

Figure 4. 35 - User position for a step command with an eccentricity fault from 0% to 100%

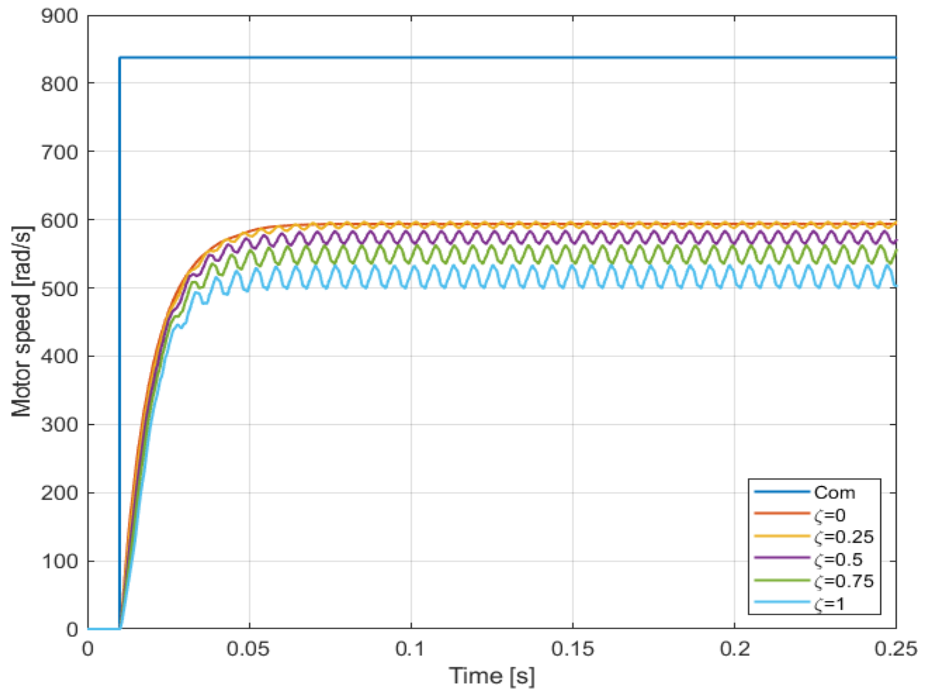

Figure 4. 36 - Motor speed for a step command with an eccentricity fault from 0% to 100%

The dynamic response of the speed to a step command (open-loop response) is represented in graph 4.36. It's possible to see that is similar to the response of the short circuit fault, but with the characteristic time of the mechanical system longer: indeed, the inertia of the motor acts like a low-pass filter, damping the high-frequency peaks of the speed trend. Another little difference is in the value of the maximum speed, lower than the short circuit fault.

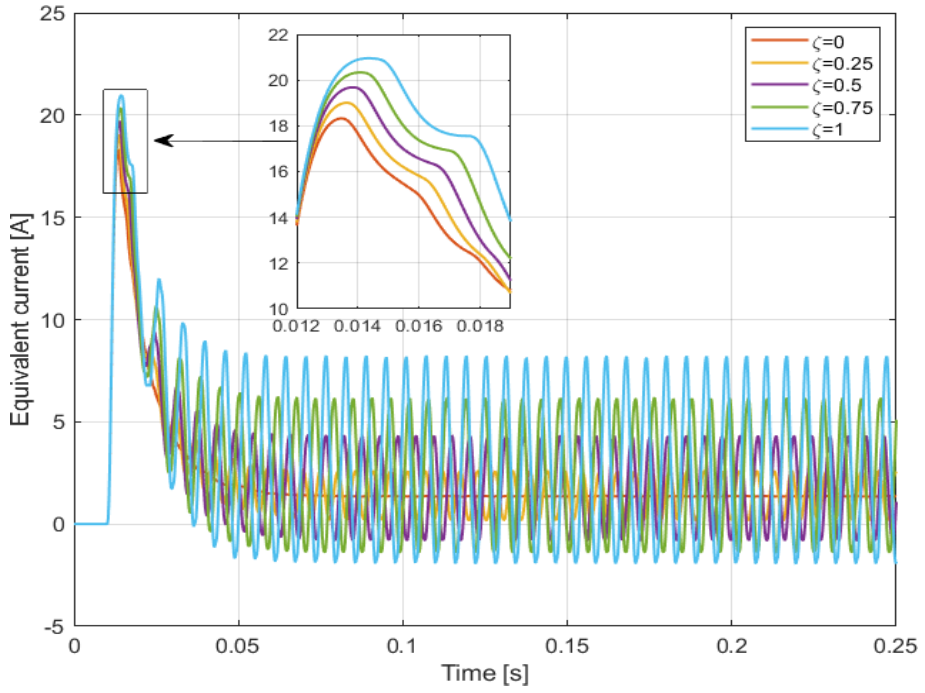

Figure 4. 37 – Equivalent single-phase current for a step command with an eccentricity fault from 0% to 100%

As already said, the counter electromotive force and torque gains modify their value depending on the angular position. In figure 4.37 is depicted the equivalent single-phase current in nominal and faulty conditions: it's important to underline that the average value of the oscillations is the same of the nominal condition, and the ripple is more accentuated than the short circuit fault. Concerning the peak current, a faultier motor requires a higher and higher value.

4.5.4. Dynamic Response to a chirp command

The figures in the next page describe the dynamic response of the electromechanical actuator to a chirp command. Comparing them with the graphs coming from the short circuit fault, it's possible to state that user position and rotor speed are very similar, because the controller is able to compensate the unbalancing magnetic forces and it is capable to maintain the nominal position and speed (with a slight delay at the moment of inversion of the speed).

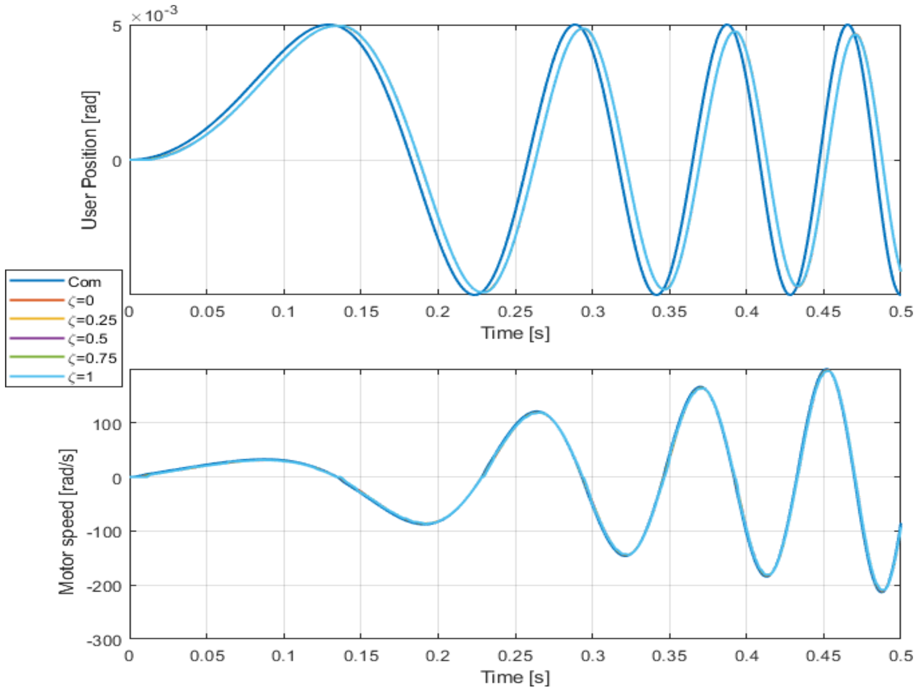

Figure 4. 38 – User position and motor speed for a chirp command with an eccentricity fault from 0% to 100%

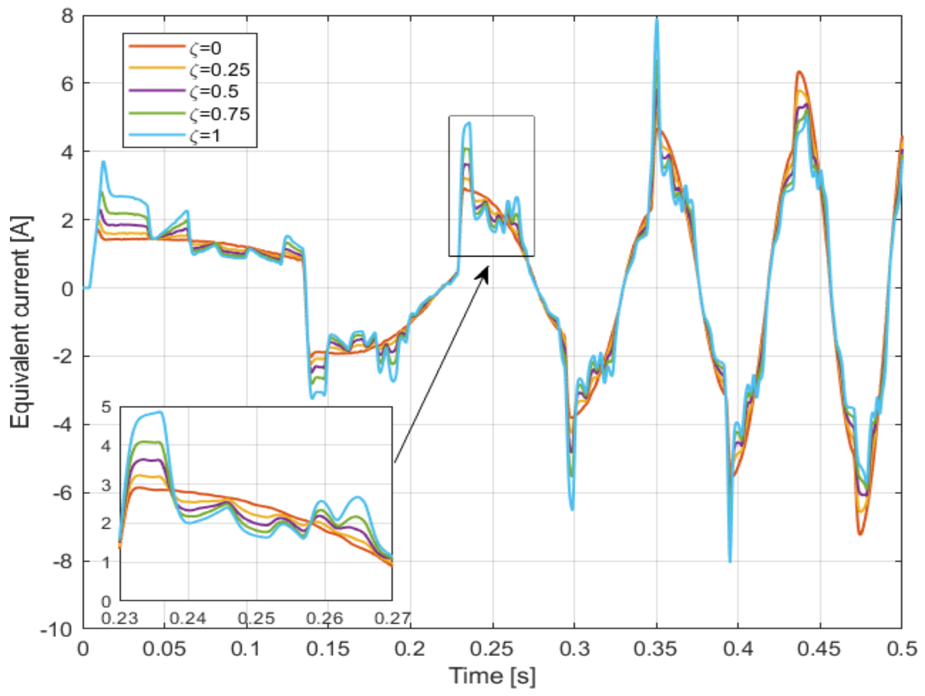

Figure 4. 39 — Equivalent single-phase current for a chirp command with an eccentricity fault from 0% to 100%

Concerning the current, figure 4.39 highlights the increase of ripples when the motor is afflicted by a superior value of fault. This fact could be explained as an overlap of effect: there is a sine wave with the same frequency of the rotational movement coming from the variation of the air gap, and the high-frequency sawtooth ripple generated by the activation and deactivation of the motor phases.

From the figure below the capability of the monitor model to approximate the high-fidelity one is described both in nominal and faulty conditions. In the left part of the graph, at the start of motion, the approximation is good but not optimal (the percentual error is about 7%), after 0.1 s the percentual error become approximately the 2%, confirming the validity of the previous treatise.

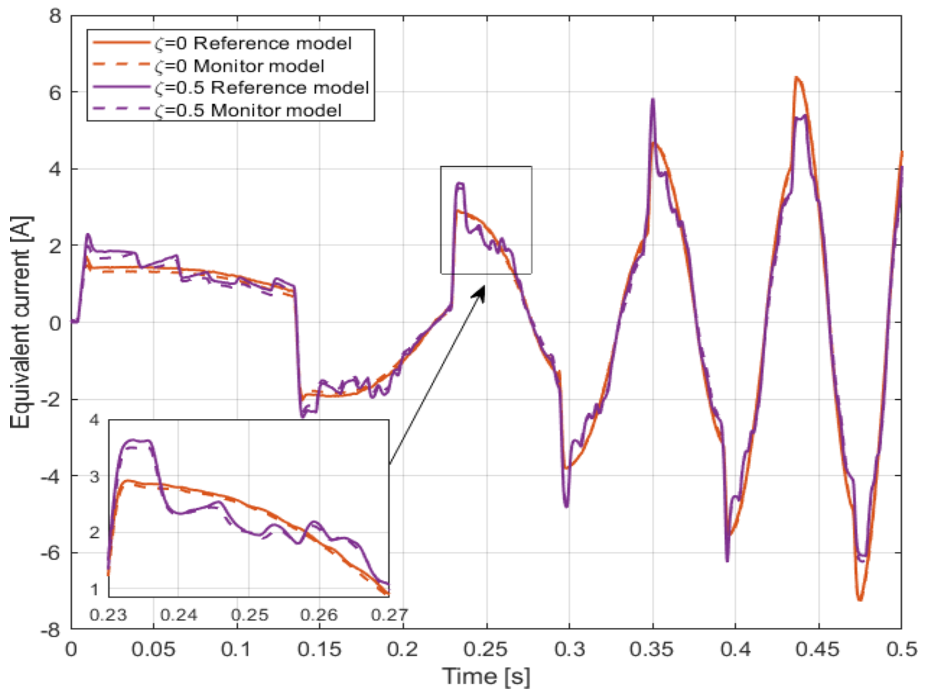

Figure 4. 40 - Equivalent single-phase current of reference and monitor models in nominal and faulty conditions

4.6. Proportional gain

4.6.1. Description

Electronic components are assuming an increasingly critical role in a lot of different fields, such as on-board function, communications and autonomous functions. All these new functionalities, together with the growth of lead-free electronics and microelectronic devices, could increase the number of electronic faults and maybe result in unknown behaviours. Hence, to assure a high reliability and safety of the flight control system, is mandatory to provide a system health awareness. [25]

Electronic faults can be categorized for example by type of component afflicted, as already described in table 6. Main issues with the control electronics may take place in the capacitors or in the transistor, with a possible open or short circuit failure, which can lead to an entire break-down of the system. Other types of electronic failures can be found in wiring connection due to an overheating, or in the power supply with the intermittent performance or thermal runaway. [16]

Usually, a great part of electronic faults arise without any pre-alerting signals, so they are very difficult to implement in a prognostic study. The main solution to fix this problem is the implementation of multiple redundancies, in order to substitute the broken controller as soon as possible without interrupting the normal working activities of the system.

4.6.2. Implementation

In this work a generic progressive electronic fault is implemented with a variation of the proportional gain applied by the control electronics: in this way it's possible to see the modification of the dynamic response of the models.

The practical implementation is quite simple: the proportional gain into the reference and the monitor model (both represented in a light-blue square respectively in figures 4.41 and 4.42) is multiplied for a suitable value which can vary in the range 0.5-1.5: in this way the nominal proportional gain 10^5 [s^{-1}] can assume values between $5\cdot10^4$ [s^{-1}] and $1.5\cdot10^5$ [s^{-1}].

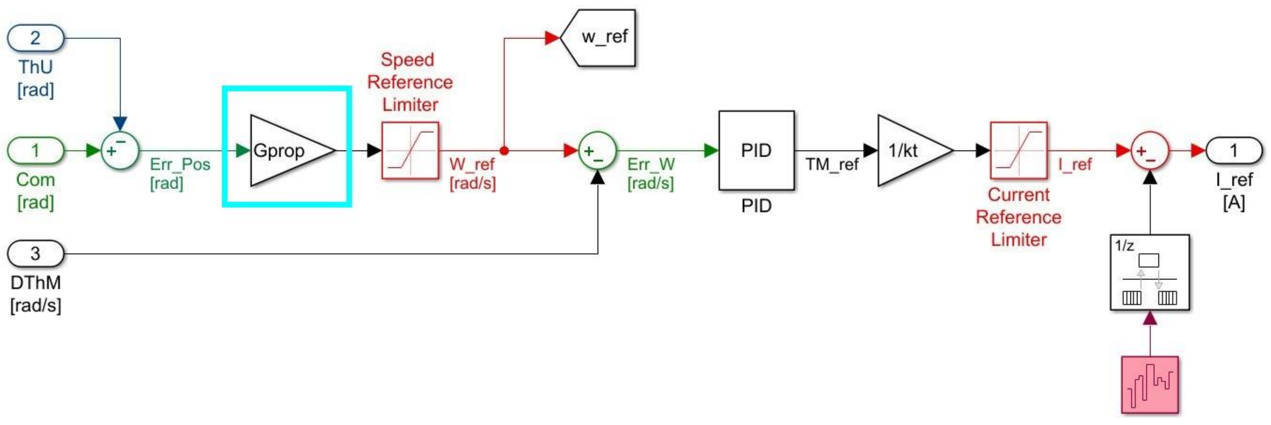

Figure 4. 41 - Proportional gain implemented in the reference model

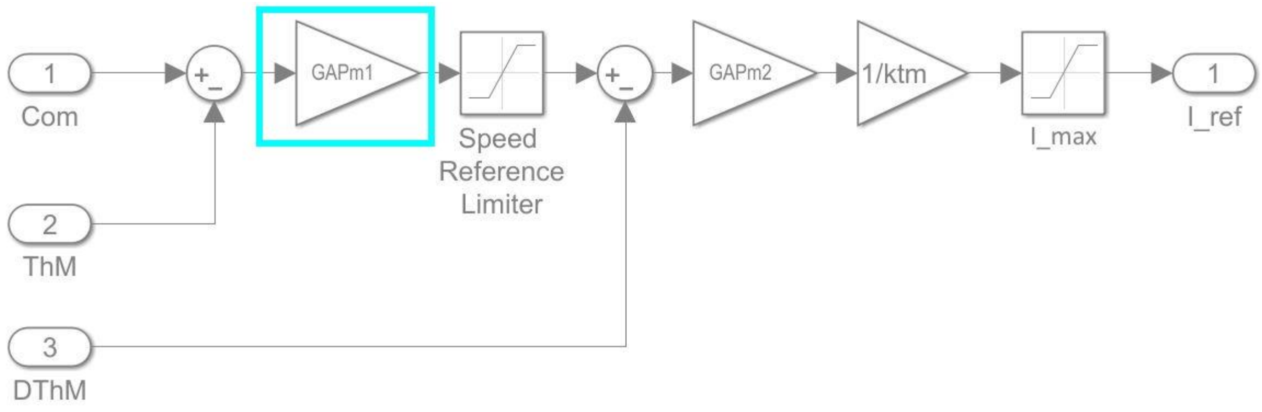

Figure 4. 42 - Proportional gain implemented in the monitor model

4.6.3. Dynamic Response to a step command

In this paragraph the open-loop dynamic response to a step command is investigated. The user position, motor speed and equivalent single-phase current are not affected by the modification of the proportional gain fault parameter from 50% to 150% and they are perfectly overlapped to nominal condition trends. Indeed, when the proportional gain is half of the nominal conditions, the error evaluated by the controller is high enough to require the maximum current available.

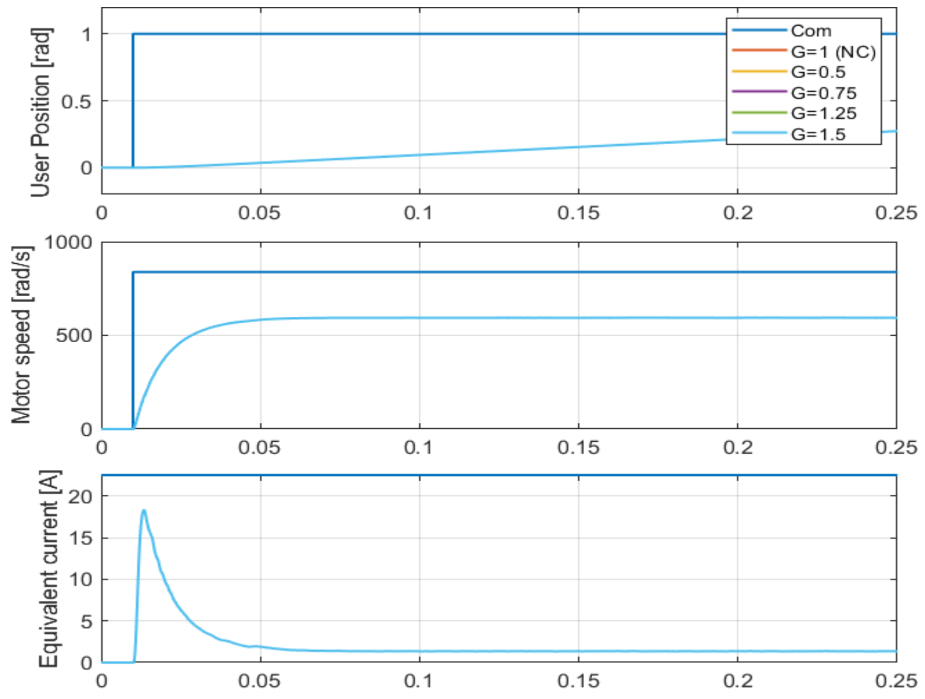

Figure 4. 43 - User position, motor speed and equivalent single-phase current for a proportional gain fault from 0.5 to 1.5

4.6.4. Dynamic Response to a chirp command

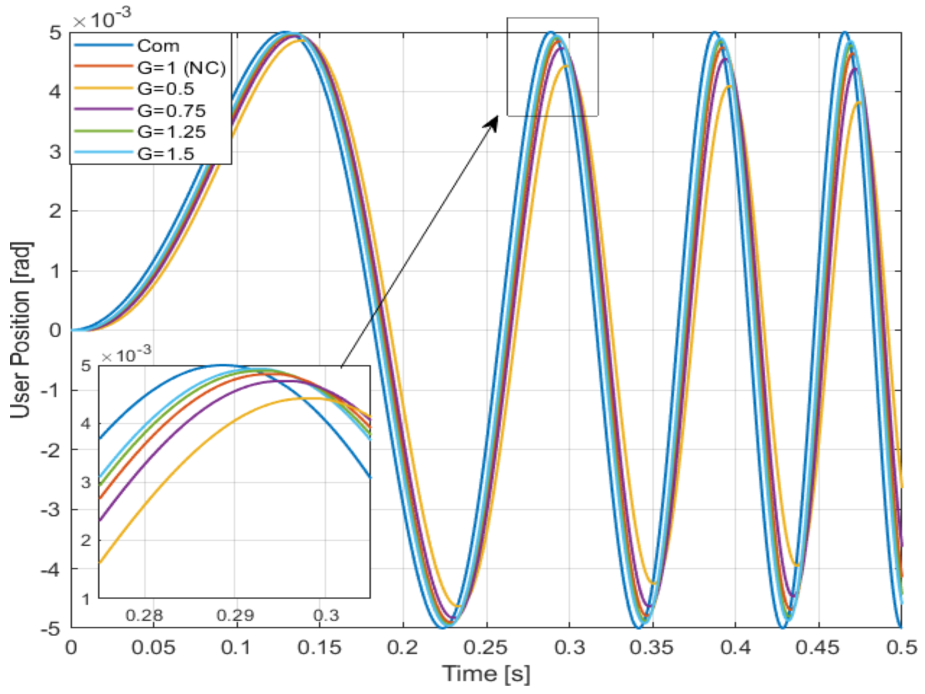

Figure 4. 44 – Rotor position for a chirp command with gain fault from 50% to 150% of nominal conditions

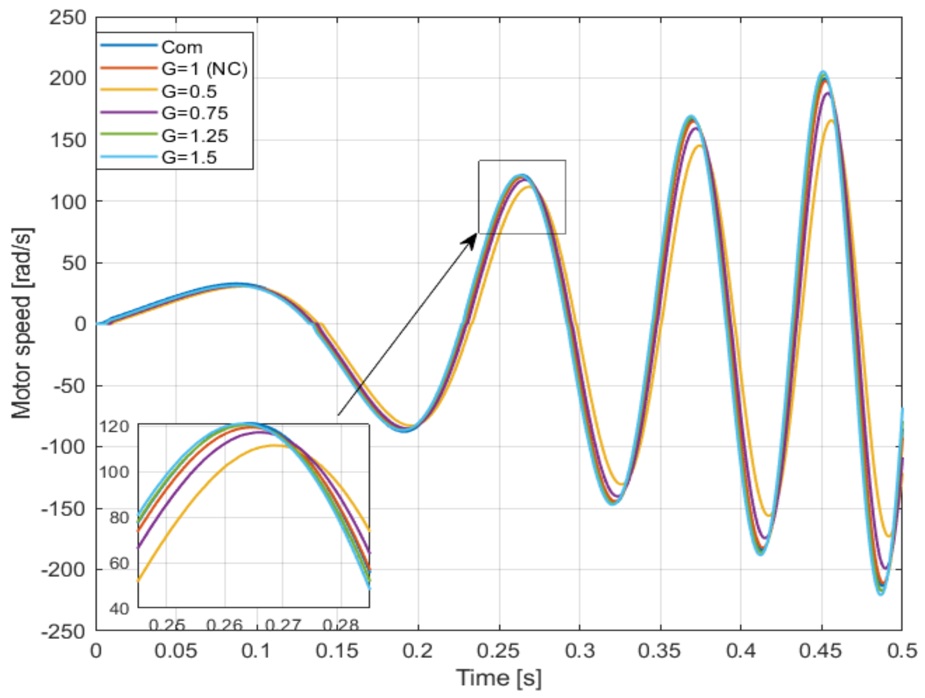

Figure 4. 45 - Motor speed for a chirp command with gain fault from 50% to 150% of nominal conditions

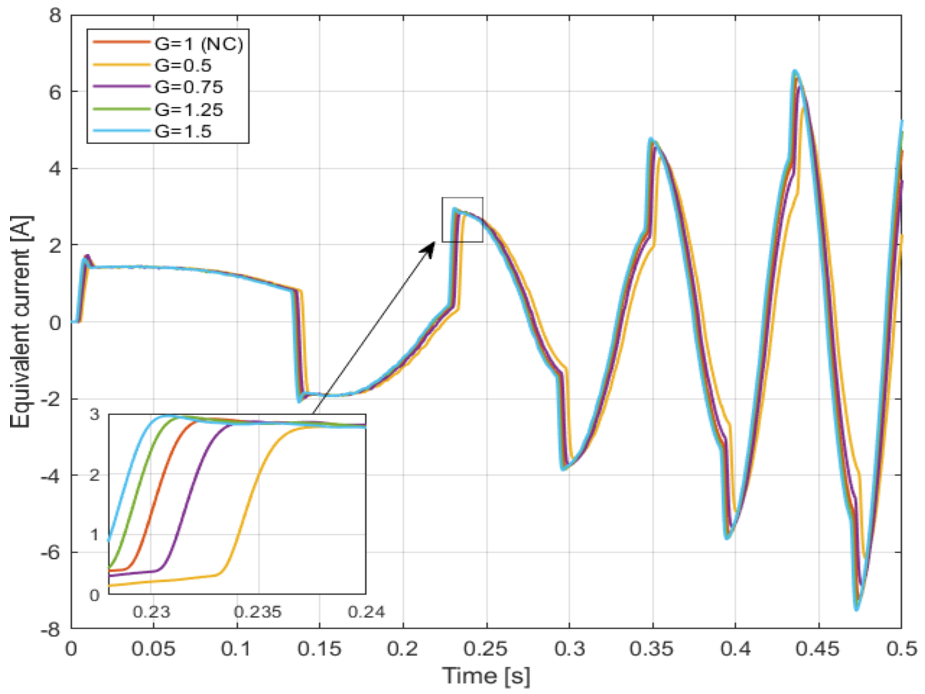

Figure 4. 46 – Equivalent single-phase current for a chirp command with gain fault from 50% to 150% of nominal conditions

From the figures above, it's possible to understand the effect of a proportional gain fault in the controller. Assuming as nominal conditions (NC) the orange curve, a decrease of proportional gain (yellow and purple trends) leads to a decline of the readiness of the motor, causing a delay in the current calculation by the control law and therefore in the velocity and position of the motor. In particular, if the acceleration acts on the motor in delay, the maximum speed reached by the rotor is less than the case with a superior gain, because the command forces the current to decrease before the achievement of the maximum speed allowed. The characteristic time of the system increase if the gain rises up: the light-blue trend, corresponding to a gain equal to $1.5 \cdot 10^5$, represents the most similar curve to the command (in blue).

Figure 4.47 represents the non-faulty and faulty condition for reference and monitor models. The proportional gain does not affect the reliability of the approximation capabilities of the monitor model, which overlaps in long part the high-fidelity trend.

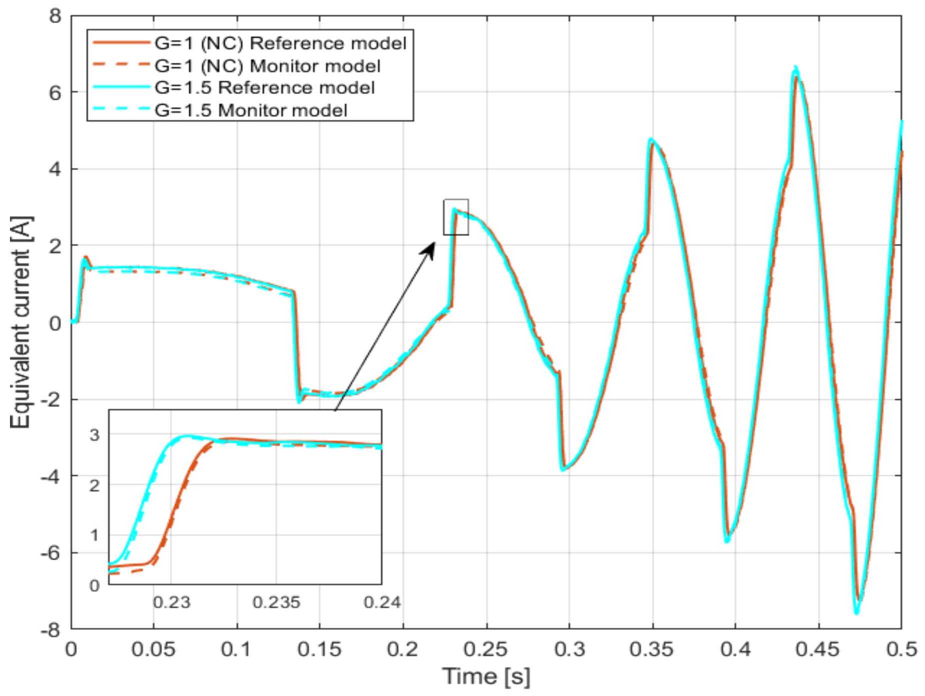

Figure 4. 47 - Equivalent single-phase current of reference and monitor models in nominal and faulty conditions

4.7. Noise

4.7.1. Description

Noise is any unwanted signal which interferes with the measurement or communication of another signal: it conveys information regarding the sources of noise and the environment in which it propagates. Noise is present in almost all environments in the form of different disturbances such as cellular mobile communication, speech recognition, image processing, medical signal processing, radar and sonar acquisition data. In general, noise and distortion are the main factors which limit the capacity of signal transmission and the precision of the results coming from measurement; if the disturbances are too high relating to the signals acquired, they can cause transmission errors, until even disrupt the communication process. Their removal is a critical issue, deeply studied in a lot of disciplines. [26]

In literature there are a lot of different classification of the noise; in this paragraph, the main two are described.

Depending on its source and physics, a noise can be described as [26]:

- *Acoustic disturbances*, which include the noise, wrong feedbacks and echoes. The noise emanates from the movement, vibrations, weather conditions (rain, wind, et c.) or from the utilization of everyday stuff such as air-conditioners, vehicles or fans; the feedbacks and echoes coming from the reflection of the sound carried out by walls, especially if they are made with particular materials;
- *Electronic device* noise, which could further be divided into *thermal noise, shot noise, flicker noise* and *burst noise*;
  - Thermal noise is created by the movement of thermally energised particles in an electric/electronic conductor;
  - Shot noise represents the random fluctuation of electric current due to its nature, indeed the electrons are discrete charges with different arrival time;
  - Flicker noise is caused by the inclusions in the conductive channel and recombination of noise carried out by the transistors;
  - Burst noise composed by step transitions of several hundred millivolts, at random time and durations.

- *Electromagnetic noise*, particularly critical at radio frequencies (kHz to GHz range), is the combination of natural and man-made sources;
- *Electrostatic noise,* generated by the voltage with or without current flow;
- *Channel distortions, multipath, echo and fading*, created by the non-ideal conditions and characteristic of communication channels, which are very sensitive to environment, multipath effect and fading of signals;
- *Co-Channel interference* appears when two different radio transmitters are on the same channel frequency: the effect is a sort of crosstalk;
- *Missing samples,* several part of the signal could lack due to packet loss in communication systems;
- *Processing noise*, related to the analogic and digital signal acquisition noise.

Depending on its frequency spectrum or time characteristics, a noise process can be also divided into [26]:

- *White noise*, purely random noise, theoretically contains all frequencies in equal power;
- *Band-limited white noise*, characterized by a flat power spectrum and a limited bandwidth;
- *Narrowband noise,* limited between 50 and 60 Hz;
- *Coloured noise*, whose spectrum has a non-flat trend;
- *Impulsive noise*, a short duration pulses of random amplitude;
- *Transient noise pulses*, long duration noise pulses such as clicks or burst noise.

Relating to this work, a white noise is implemented in the reference model, to simulate in a realistic way the test benchmark in which the monitor model has to work. White noise is theoretically defined as an uncorrelated random noise process with equal power to all frequencies: this fact leads to the concept that it requires to have infinite power and a flat power spectral density. In reality, physical systems are never affected by white noise, but it is only a useful approximation when the real disturbance has only a slight correlation with the trend analysed. Figure 4.48 represents a possible white noise time trend to a flat type acquisition:

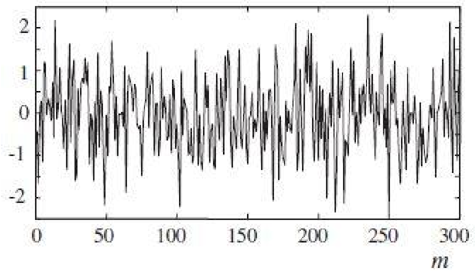

Figure 4. 48 - White noise time trend

4.7.2. Implementation

The implementation of the noise disturbance in reference system is carried out by a suitable Simulink white noise block, represented in the figure below, which generates a normally-distributed random numbers suitable for hybrid systems.

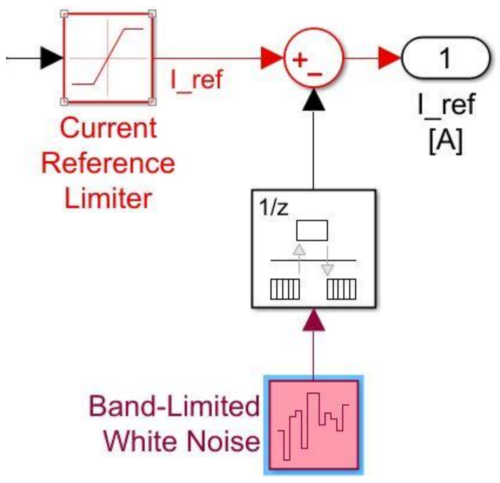

Figure 4. 49 – Implementation of the noise in the reference system

This block allows the scaling of the covariance of the noise from a continuous power spectral density to a discrete one, in order to obtain the correct intensity of the noise [15]. The noise signal coming from that block goes into a *rate transition* block, which transform the disturbance frequency into the frequency of the system, to have the same rate of simulation.

The monitor model does not simulate the noise, because it will extend the computational time without adding any improvement to the algorithms' precision.

4.7.3. Dynamic response

The response of the system to different values of noise coefficient are investigated only for a chirp command: this decision has been made due to the small influence of that disturbance of the system and because, during the optimization paths, the noise has not been introduced.

The Simulink block receive as input a noise coefficient, K_{noise}, which is suitably multiplied by 10^{-6}, in order to make the values of disturbance and current of the same order. Increasing the values of K_{noise} from 1 to 100, the system clearly becomes always more unstable, showing a more indented trend.

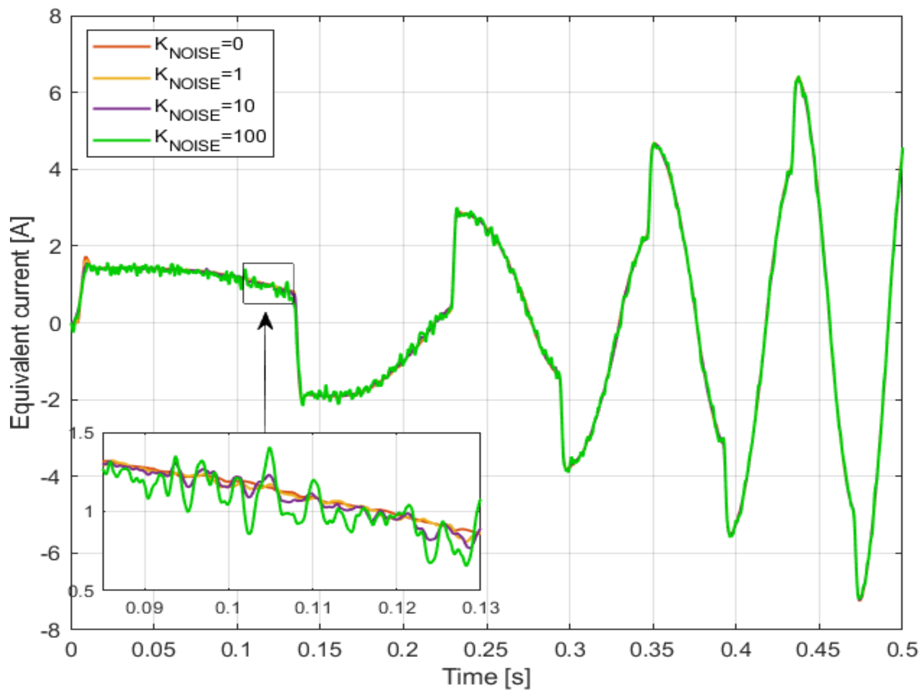

Figure 4. 50 - Effect of the growing noise on the equivalent single-phase current

5. Optimization algorithms

As already said in the “Introduction” chapter, Prognostics is the discipline which evaluates the current state of a system or single component and estimates the Remaining Useful Life (RUL), namely how much time will pass until the object of the study will no longer able to operate within its stated specification [2]. In this work a model-based prognostics approach is proposed: it will be applied to the study of fault appearance in flight controls.

Optimization algorithms are used for the monitor model to approximate in the best way possible the parameters of the reference model. Indeed, design optimization is an important part of every design problem in engineering and industry: it focuses on finding the optimal and practical solution to complex problems, also under non-linear constraints.

The first step is to launch the reference model in which a pre-determined fault is implemented. In this way with the simulation it’s possible to obtain as outputs all signals of position, speed, torque and current in faulty condition. The second step is the simulation of the monitor model: it is required to approximate as best as it can the parameters of the reference, in order to detect the fault introduced. To achieve that goal, the monitor needs to run a lot of times with different parameters, choosing as best those closest to reference model: these variations of the main characteristics of the monitor model at each simulation are made by suitable optimization algorithms.

In this work, four different algorithms have been evaluted to solve the problem and compared each other: Genetic Algorithm (GA) and Differential Evolution (DE) as Evolutionary Algorithms, Particle Swarm Optimization (PSO) and Greywolf Optimization (GWO) as Swarm Intelligence. In this chapter, after a brief introduction to different types of optimization patterns, they are deeply described.

5.1. Introduction to problem solving algorithms

Optimization problems are wide ranging and numerous, hence also methods for solving them need to be an active research topic [27]. Classical problem-solving methodologies could be easily divided into two branches: *deterministic* or *stochastic*. The former are the exact methods, as logical or mathematical formulas, which solves a problem with a pre-determined sequence of search points, obtaining a solution very close to the global minimum. However, they require huge computational efforts to achieve the solution, and if the problem's size grows up this is not an accessible path anymore. This is the reason of the development of stochastic and heuristic methods: these rely on the iterative improvement of a population of solutions created with a randomization. For this reason, different runs of the same algorithm applied to the same problem lead to different results due to the random nature of the solutions.

Development of stochastic and meta-heuristic nature-inspired algorithms started from 1973 with the publication by Holland [29] of the first Genetic Algorithm based on Darwin's principle of development of the species. Since then, with the growing interest in these new approaches also Ant Colony Optimization (ACO), Simulated Annealing (SA) and Particle Swarm Optimization (PSO) have been introduced. Their generality, facility and convergence attributions allowed their fast development and application to an enormous range of different applications [28]. On the other hand, also deterministic methods met a remarkable growing through the past years, developing Direct search, branch and bound, clustering and tunneling methods. In this work only stochastic algorithms are described and investigated.

A first classification of metaheuristic algorithms is performed on the basis of the number of initial attempts: *single-solution-based* (e.g. Simulated Annealing) the optimization process starts with a single solution tried which is improved iteration after iteration, *population-based* exploit a set of search agent which work contemporary to find a greater number of suitable solutions.

Heuristic methods can be further divided into *Evolutionary Algorithms (EA)*, based on the natural evolution, and *Swarm-based Algorithms (SI)*, which rely on collective behaviour of a group of animals.

- Evolutionary algorithms are a particular application of artificial intelligence, which exploit the iterative progress of a population: birth, growth, development, reproduction, selection and survival. These methodologies are the most well-known, used and established among the nature-inspired algorithms. The exploited generic process starts with the creation of random search points, whose deviation have a poor suitability with the objective function carried out by a selection mechanism, and the subsequent variation of the points left to reach different solutions in the function domain. The most famous algorithms are: Genetic Algorithm (GA), Genetic

Programming (GP), Differential Evolution (DE), Evolutionary strategies (ES) and Paddy Field Algorithm (PF).

- Swarm intelligence (SI) concept has been introduced in 1993 [37] and takes inspiration from colonies, flocks, packs, herds and schools present in nature. It is a recent and emerging path of problem resolution: it efficiently finds suitable solutions to intractable and complex optimization problems exploiting the social behaviour of group of organisms. The word "swarm" comes from the movement of particular agents in the problem's domain and it is defined as "any loosely structured collection of agents that interact among each other" [30]. Every agent of the swarm is required to evaluate the objective function a lot of times, so a parallelization is very easy to implement in order to reduce computational time and to obtain more reliable results. Furthermore, SI usually has less parameter to adjust and a lower number of operators in comparison with Evolutionary Algorithms. The most used swarm-based methodologies are Ant Colony Optimization, Particle Swarm Optimization and Artificial Bee Colony, but in last years a huge number of nature-inspired algorithms has been developed.

It's important to underline that not all the algorithms behave in the same way. Indeed, the No Free Lunch Theorem (NFL, [38]) states that if an algorithm works particularly well on a class of problems, it will show poor performances on other sets of problems. In particular, if an algorithm performs better than a random search in a determined class of problem, in other types of problem it will behave worse than the random search. Summarizing, it means that a perfect heuristic algorithm able to solve all optimization problems does not exist.

5.2. Genetic Algorithm (GA)

Genetic algorithms are a class of Evolutionary-based self-adaptive stochastic optimization algorithms with a global search potential proposed by Holland in 1973 [29] and based on the Charles Darwin's theory of the survival of the fittest.

Initially, this algorithm creates randomly possible solutions of the problem, called *initial population*. In its most common representation, a population is composed by a variable number of *individuals* (a binary string, called *chromosome*), thinkable as group of different *genotypes* (the 1 and 0). Genotypes represents the characteristic of a solution, e.g. the parameter of an electric motor for an electrical optimization purpose. The three main operations carried out by the GA are *selection*, *crossover* and *mutation*.

In figure 5.1, taken from [31], is represented first steps of genetic algorithm path.

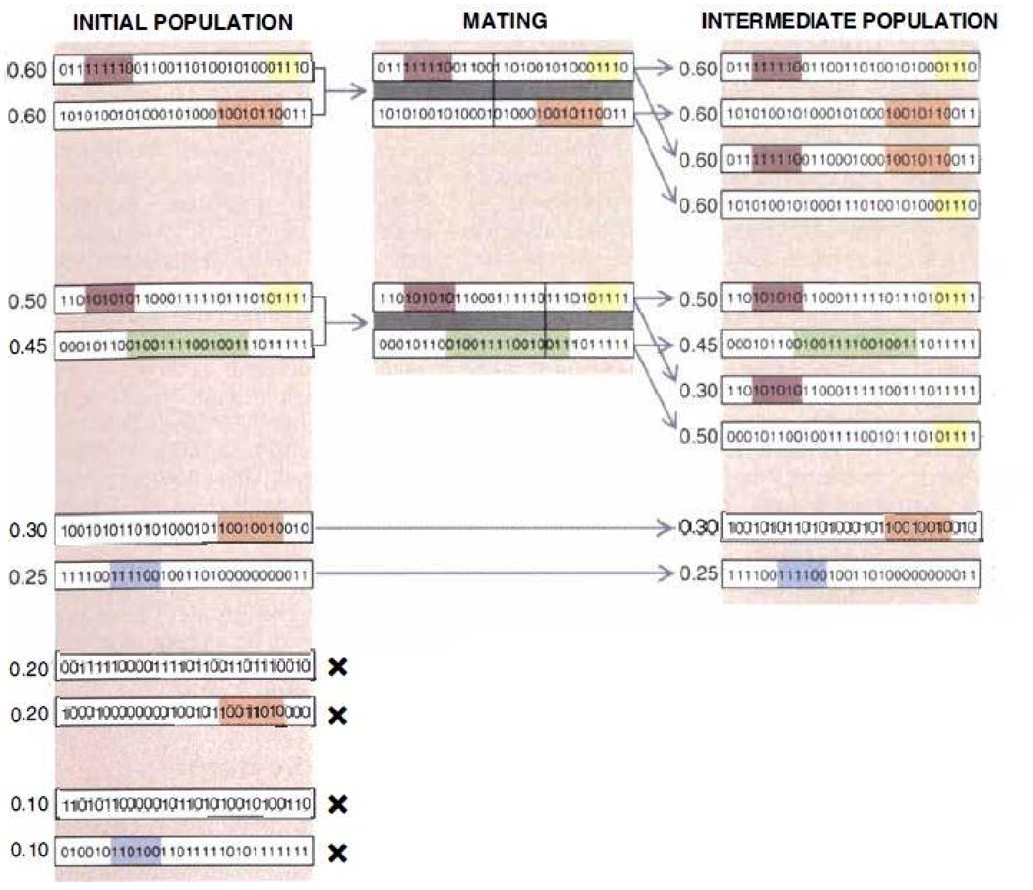

Figure 5. 1 - Representation of first steps of Genetic Algorithm path

Exploiting a suitable fitness function which changes depending on the type of the problem, each string of population is evaluated to discover its effectiveness, which could be seen as the capability of each individual to adapt at the environment. Near every "population" column, in figure 5.1 there are several numbers varying from 0 to 1: these are the values of effectiveness of that agent to solve the problem. Once evaluated this coefficient, the algorithm ranks the most effective solutions from the best to the worst.

The second step is the *cross-over*. Between the best solutions there is a recombination of portion of strings carried out by random points. They divide the individuals in several parts and mix each other in the "mating pool" to obtain new possible solutions, called off-springs. It's possible to classify the cross-over in "*N-points cross-over*" where N is the number of cutting points. The second generation of population is not only the result of the cross-over, but also of the *mutation*. This phenomenon consists in the introduction of discarded individuals or new ones in the population, in order to reintroduce material loss or explore new possible random solutions. After this action, the effectiveness of every individual of the new population is evaluated with the help of the fitness function and the cycle restarts.

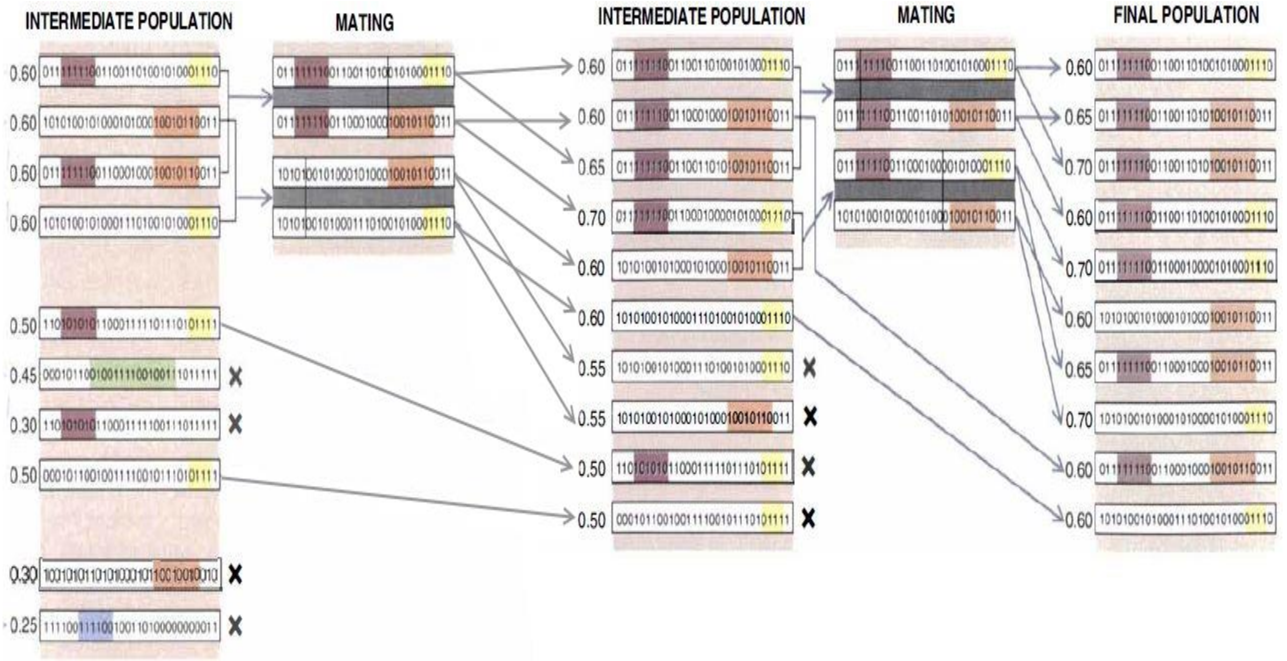

Figure 5. 2 - Final steps of genetic algorithm path

In figure 5.2, also taken from [31], final steps of the genetic algorithm process are depicted. Continuing this process for more steps, efficiency of solutions rises up, allowing the discover of higher effectiveness results. It's possible to put together the algorithm and determined stopping criteria, such as the maximum number of iterations or the objective function tolerance between a step and the next.

For further clarity, below the pseudo-code of the genetic algorithm is reported:

```
$t$=0;
initialize (randomly) a chromosome population $P(t)$;
evaluating $P(t)$ using the fitness function;

while (Stopping criteria not respected)

do
   -  Select individuals from $P(t)$;
   -  Insert them into the mating pool $MP$;
   -  Apply crossover to the individuals of the $MP$, forming $P'(t)$;
   -  Apply mutation to the individuals of $P'(t)$ forming $P''(t)$;
   -  Form $P(t+1)$ by selecting for replacement individuals from
$P''(t)$ and $P(t)$;

$t$=$t$+1;

End
```

Genetic Algorithm is useful when [27]:

- ✓ there is not any mathematical treatise available;
- ✓ the knowledge of the function domain is poor;
- ✓ the search space is large, complex and highly constrained;
- ✓ there is the necessity to parallelize the computation;
- ✓ traditional search method fails.

As just explained, the genetic algorithm is a very powerful tool to obtain reliable optimized data, but it has also some drawbacks:

- often it converges to local optima if the objective function is not defined properly;
- the implementation in dynamic set of data is critical;
- for specific problems, some other simpler optimization algorithms could achieve more precise results.

The usage of this algorithm has been carried out using the equivalent command in Matlab environment [15], **ga**, together with the modification of some the optimization options. The syntax of the command is:

```
x = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

where:

- *fun* is the fitness function to be optimized;
- *nvars* is the number of variables which have to be optimized (in our case, the eight values of the fitness vector: see chapter 7);
- *A, b, Aeq, beq* are the coefficients of the linear equalities and inequalities the fitness function is subjected (`Aeq*x = beq` or `A*x ≤ b`);
- *lb, ub* are the lower and the upper bounds, in our case set to zero and one;
- *nonlcon* introduces non-linear constraints;
- *options* are the possible modifications to implement in the optimization. In our case the options used are:
  - *Display* set the level of optimization detail returned to the user. In our case I've set 'iter' because it's important how the error behaves iteration after iteration to see if the code works rightly;
  - *FitnessLimit* has been used to set the maximum objective value to 10^{-3} during the multiple optimizations;
  - *MaxGenerations* impose the maximum number of iterations. For all the results shown in next chapters, this value is set to 200;
  - *MaxTime* is set only once to calibrate the right algorithm's parameters;
  - *PopulationSize* limits the number of individuals per each generation to the value set;
  - *UseParallel* exploit the parallelization of the calculations.

5.3. Differential Evolution (DE)

Differential Evolution (DE) is a population-based derivative-free stochastic evolutionary algorithm, firstly introduced by Storn and Price in 1995 [32], very powerful and efficient in the continuous search domain. This algorithm is very similar to the GA, since a population of individuals is exploited to find the optima in a fitness function; on the other hand, in the Differential Evolution the *mutation* is the result of arithmetic combination of factors and not the merely randomization of genes as in GA. Furthermore, the fact that it does not exploit derivative operations makes the DE a suitable path when the gradient is difficult or even impossible to calculate.

Although slight differences are present between an evolutionary algorithm and another, they all rely on Darwin's theory of evolution and exploit the concept that a population of individuals could find a problem solution with some fixed steps which are repeated cyclically. The steps involved in the algorithm are barely the same of the Genetic Algorithm: initialization, mutation, cross-over and selection, as depicted in figure 5.3 taken from [33].

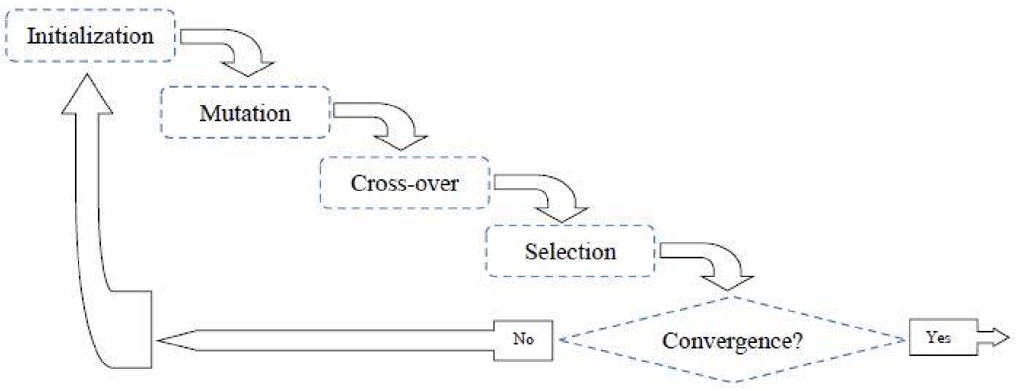

Figure 5. 3 - Working flow chart of the Differential Evolution

The *initialization* creates a pre-determined number of new individuals (NP) with generic features according to a normal or uniform distribution.

With the *mutation* event some new material is implemented in the population by generating variations to existing individuals. This operation creates at each generation a mutation vector which could take care about the best solution, the current solution or the difference between the current and the best solution. In the Matlab code developed for this work, the last type of mutation vector is exploited (see Appendix A). Usually, the mutation or scaling factor F_i is the same for every generation, but it's common to meet in literature some DE codes which take advantage of mutation factors different for different generations.

The *cross-over* action mixes up some informations between different individuals. It forms a trial vector with the help of the characteristic parameter $CR_i \in [0,1]$: it is the cross-over ratio, which represent the average fraction of vector components that are inherited from the mutation vector [33]. Similarly, also CR_i as F_i is often fixed through different generations.

After the evaluation of the fitting value carried out by the fitness function, which identifies how satisfying is the suitable solution, the *selection* chooses as best solution the trial vector or the original parent depending on the type of the problem. In our case, we need to find a minimum of a function, so the best value $x_{i,g+1}$ of the second generation $(g+1)$ is:

$$x_{i,g+1} = \begin{cases} u_{i,g}, & f(u_{i,1}) < x_{i,g} \\ x_{i,g}, & otherwise \end{cases}$$

where $u_{i,g}$ is the *i-th* components of the trial vector at generation g and $x_{i,g}$ the start value of the previous generation (parent). All these operations explained are parts of a loop, until stopping criteria are fulfilled.

The main advantages of the Differential Evolution are:

- ✓ ease to implement;
- ✓ excellent speed of convergence;
- ✓ perfect for problem which has solution close or even on the boundaries;
- ✓ it's widely used thanks to the high reliability and achievable accuracy.

DE algorithms have also some limitations:

- the presence of noise could roughly affect the precision of the results;
- tuning difficulties. The number of individuals (NP), mutation and cross-over ratios (F_i and CR_i) are uneasy to find, because they require to perform time-consuming tries. Furthermore, in domain space it's possible to meet several places in which different optimization parameters work better than others. To fix this problem, in past years some self-adaptive Differential Evolution algorithms (SaDE) have been developed [e. g. 34].

The implementation of this algorithm has been made in Matlab environment starting with the code contained in [33], strongly modified in order to adapt it at our problem (see all the code in Appendix A). The code developed receives as input the fitness values from the main code (reported in Appendix C) as the maximum number of iterations, set like GA as 200, the population size, also set to 50, and the lower and the upper boundaries. After a random initialization of the positions, the current objective fitness for every individual is evaluated, stored in a vector and investigated in order to find the minimum of the function.

The subsequent part is the mutation, which exploit int8 values and Boolean operations to become faster. As explained in the description part, this method utilizes only additions, subtractions and multiplications to introduce a “chaos” component into the code. Once generated the new individuals from the fittest of the previous generation, a new investigation finds the best solution through the minimum values and substitutes it in the best-solution vector. Implementing the stopping criteria, the algorithm stops if requirements are fulfilled and the output is the last value of the best vector.

5.4. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is an intelligence-oriented, stochastic swarm-based global optimization technique firstly introduced by Kennedy and Eberhart in 1995 [35]. It was born from these two authors as lightening research of bird-flocking algorithms already published, but during some tests the continuous changes made it more powerful than the previous ones.

The term “particle” in the name of the algorithms indicates the population members which are mass and volume free, or even characterised by arbitrary small mass and volume. They represent possible solutions in a high-dimensional space with four vectors: actual position, best position already found, the position of the neighbourhood and its velocity. Every iteration, all particles calculate the fitness value of their position exploiting a suitable fitness function; if it is better than the best position they have already reached, it becomes the new best position.

Main steps in PSO algorithm represented in figure 5.4 could be summarized as:

- *initialization* of the particle swarm composed by the pre-determined number of individuals. Every agent takes a random position in the space and, in minima research problems, its best value is set as first approximation to a high value, in order to be eliminated immediately;
- for each particle, it’s evaluated the fitness value with a suitable objective function (*evaluation*). It’s clear that in first iteration whatever is the position in the space of the particle, it has to eliminate the set best position;

- ➢ in the loop iteration, the position evaluated is compared with the best position for each particle. As already said, if the former's fitness value is better – in our case smaller – than the latter's, current position becomes the new best position.
- ➢ The algorithm subsequently finds the best absolute position of the entire swarm and saves it as the actual best solution found;
- ➢ Particles update their speeds and positions according to their best position and best position of the neighbourhood. These steps continue until stopping criteria are fulfilled.

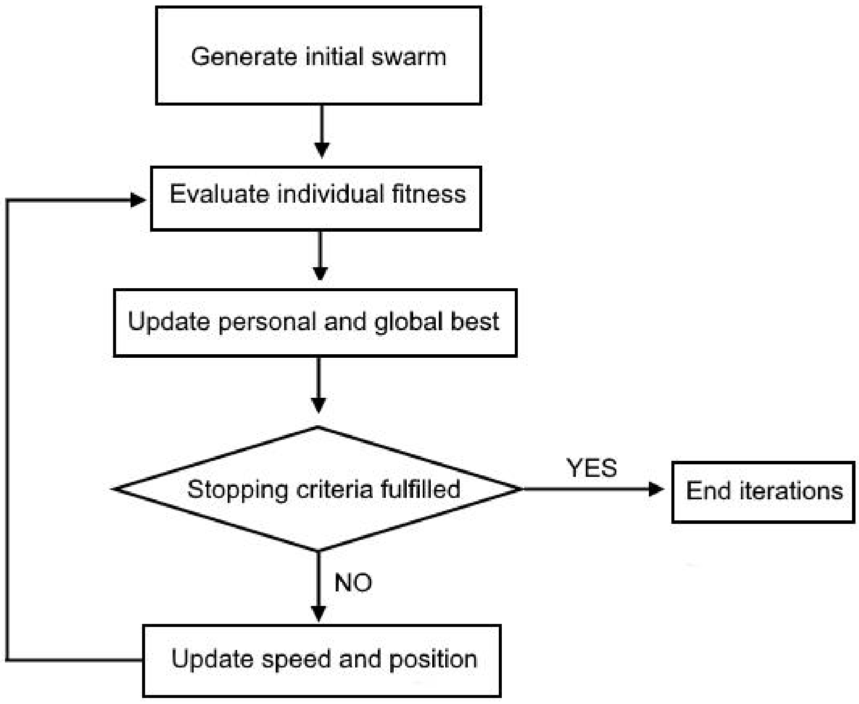
Generate initial swarm
Evaluate individual fitness
Update personal and global best
Stopping criteria fulfilled
YES
End iterations
NO
Update speed and position

Figure 5. 4 - Conceptul flow chart of the Particle Swarm Optimization

As it's possible to discuss, Particle Swarm Optimization is similar to generic Evolutionary Algorithms, because it also has initialized solutions which update themselves during iterative paths. The strong difference is the type of updating: Swarm Intelligence algorithms such as PSO exploit solution already found by agents to improve the precision of the final result, Evolutionary algorithms introduce different amounts of chaos with the randomization of mutation vectors.

Main advantages of PSO over other types of optimization are:

- ✓ marked ease to be implemented in a lot of different kinds of problems;
- ✓ simple concept;
- ✓ complete absence of parameters tuning except for the swarm size;

- ✓ more effective memory capability in comparison with GAs;
- ✓ this algorithm is able to maintain the diversity of the particles thanks to the updating method: in this way it's possible to explore a greater part of the solutions' domain.

This method is implemented exploiting the right command of Matlab and setting the optimization options as already done for the Genetic Algorithm. The implementation is similar, the syntax is:

```
x = particleswarm(fun,nvars,lb,ub,options)
```

where the parameters are the same already described for the GA in some paragraphs before. The settings are coherent with the analysis carried out: population size of 50 individuals and 200 iteration as maximum. The only difference is the expression of the maximum objective value, here recalled by the option *ObjectiveLimit.*

5.5. Greywolf Optimization (GWO)

This algorithm is the newest between those already described and it mimics the hierarch behaviour of grey wolves in nature. They have a strict hierarchy:

- the leaders are the *alphas*, usually a male and a female, which have the task to make decision and lead the group in everyday activities. However, there is also a sort of democracy in the pack concerning the less important decisions;
- the *betas* are the second hierarchic level and help the alpha to make decisions. When an alpha becomes old or dies, one of the betas take his/her position as leader;
- lowest rank is occupied by the *omegas*. They are the "scapegoat" of the pack and are dominated by all other wolves of the pack;
- wolves which are not alpha, beta and omega are *deltas.* Usually they are sentinels, scouts, hunters and caretakers.

The algorithm [36] faithfully reproduces the hunting activity of a pack: the prey is the best value of the function to obtain and the wolves are the search agents. Mathematically, alphas (α) are the fittest solution, the second and the third best solution are betas (β) and deltas (δ) respectively, the other solutions are categorised as omegas (ω).

A grey wolves hunt is composed by some moments: first there is the tracking and the approaching the prey, in a second moment the wolves encircle and harass the prey until it is jaded and stops to move and then they attack. In a 2-D domain, the situation could be represented as depicted in figure 5.5, taken from [36].

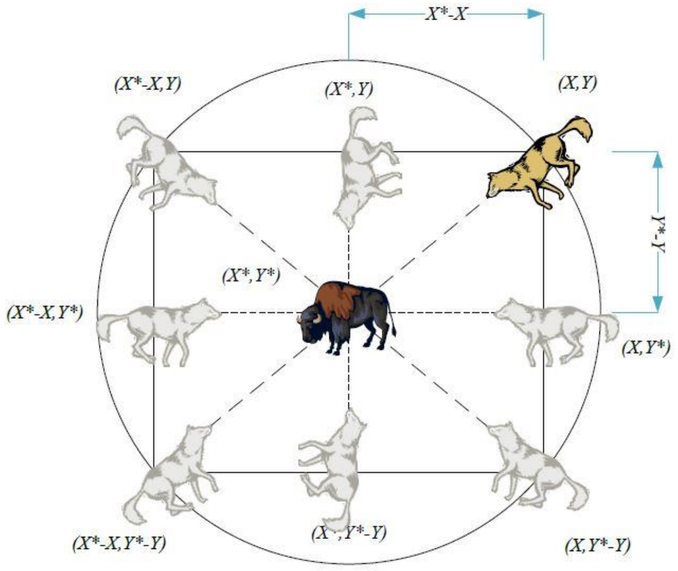

Figure 5. 5 - Encircling action of grey wolf in a 2-D domain

In an abstract search space, we do not know where the optimum value is (in other words, where is the prey), but in nature the hunt is led by the alphas who see the prey; mathematically this fact has been introduced by the recombination moment. Indeed, βs' and δs' positions at the time (t+1) are evaluated introducing random movement around the position of the αs at the time (t+1), calculated as the barycenter between the positions of the three search agents at the time (t). The position of the ωs is random and around the estimated position of the prey: in every instant of simulation the algorithm evaluates the fitting values for the wolves exploited and it assigns the label α, β, δ or ω depending on the objective function (e.g. for a minima research, αs are the wolves with the lower fitting values).

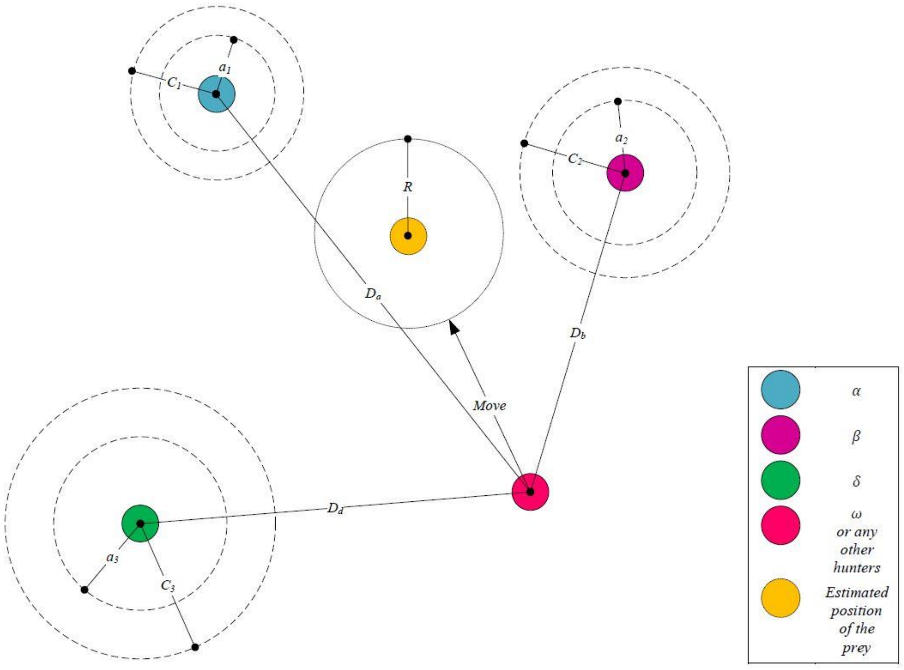

Figure 5. 6 - Update of the wolves' position time by time (taken from [36])

Also the exploration of the function domain is a priority for this method: usually wolves diverge from each other to search new preys and converge to attack a prey.

For further clarity, below the pseudo-code is reported.

```
Initialize the grey wolf population $X_i$ (i = 1, 2, ..., n)
Initialize exploration and exploitation parameters
Calculate the fitness of each search agent

Xα=the best search agent
Xβ=the second best search agent
Xδ=the third best search agent

while (t < Max number of iterations)

    for each search agent
        Update the position of the current search agent
    end for

    Update exploration and exploitation parameters
    Calculate the fitness of all search agents
    Update Xα, Xβ, and Xδ
    t=t+1

end while

return Xα
```

Concerning the description of the algorithm's parameters, three of them are particularly important:

- a is the controller of the behaviour of the pack and it decreases gradually from 2 to 0. It is used in the settings of
- $\vec{A}$ which represents the ratio between an exploration and exploitation behaviour;
- $\vec{C} \in [0,2]$ is the randomization member of the algorithm and it is used in the evaluation of the new position of every wolf.

This algorithm is implemented in Matlab environment as a code written starting from whose already present in [36], but suitably modified (see Appendix B). Particularly difficult has been the implementation of the parallelization: GWO does not support the split of the calculations.

It receives as input the initial positions of the search wolves from an external supporting code; then starts the definition of the optimized fitness vectors and best values for alphas, betas and gammas wolves, represented respectively by their positions and closeness to a prey. In an inner for loop, the best values for the random positions are evaluated taking care of the boundaries provided by the main code and subsequently the best vectors are assigned to alphas, betas and deltas. The recombination is carried out with six different random vectors in order to achieve the best exploration of the function domain possible. After the storage of the best values in a suitable variable, the loop restarts with a movement of the wolves starting from their previous position.

6. Failure detection and calibration

The algorithms just described in the previous chapter need a suitable fitness function in order to detect precisely faults growth in the electromechanical system. The definition of the fitness function is a quite critical issue, because from that depends most of the features and behaviours of optimization algorithms. It's important to use as objective function an easy-to-measure quantity, maybe without adding new sensors to the system: in fact, increasing its complexity means introducing new components liable to damages, so a possible reason of fault or breakdown (and so, reducing the system reliability).

The three main parameters measured in the considered EMA system under studies are the user position, the rotor speed and phase current, as already explained in chapter 4. Speed signal could be a great parameter to detect failures, but the necessity to generalize the treatise for different systems, maybe devoid of speed feedback ring, suggest to not use this value. The user position is useful for prognostics, because a controller can compensate the early phases of progressive damages. Hence, the current signal is exploited as failure detection parameter.

The current is evaluated in different ways in the reference and monitor model: the former calculates the three phase currents of the star-pattern circuit and then assesses the equivalent one, the latter uses the equivalent single-phase current ($I_m(t)$). In the high-fidelity model, contained in the *BLDC Motor ElectroMechanical Model,* is implemented the evaluation of the equivalent single phase, as depicted in the figure 5.7.

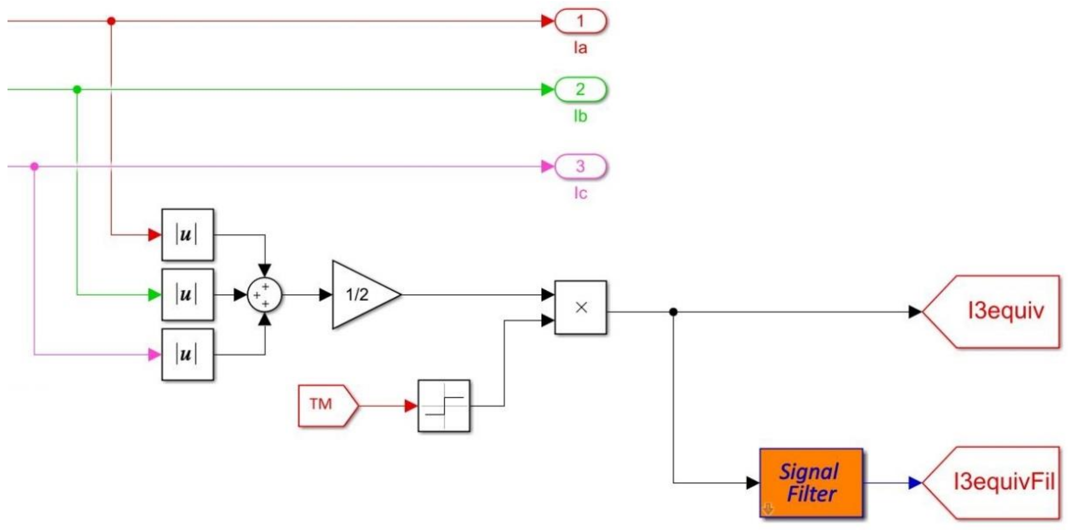

Figure 6. 1 - Evaluation of the equivalent single-phase current

Following the Kirchoff's laws, in the BLDC motor the sum of the three phase currents is always null, because two has the same value and a different sign, the third is null. Hence, the equivalent single-phase current ($I_{3equiv}(t)$) is evaluated as the sum of the modules of the three currents, subsequently divided by 2 (only two are the active phases) and then multiplied for the sign of the torque: in this way it's possible to obtain the envelope of the three phases. In order to use a simpler signal, and to not analyse too high frequencies, a filter composed by a series of three first-order transfer functions with a time constant of 10^{-5} s is exploited. The application of this filter generates an unavoidable delay in the signal response, so the same filter is applied to the monitor model: the two signals are thereby comparable.

6.1. Faults implementation

Every sample time, the equivalent current of the monitor model $I_m(t)$ is evaluated and then compared with the single-phase current of the reference model $I_{3equiv}(t)$: the error is stored, some parameters of the monitoring changes and then a new current calculation starts. Not all monitor parameters change, but the fault ones, introduced as variables in the Simulink environment.

Faults described in chapter 4 are introduced in the reference and monitor model as an eight-columns normalized row vector. The normalization has been carried out because Evolutionary Algorithms showed a faster convergence if the objective value is contained between 0 and 1 [3]. The aforementioned vector is:

$$k = [k(1), k(2), k(3), k(4), k(5), k(6), k(7), k(8)] \tag{6.1}$$

where:

- $k(1) \in [0,1]$ represents the normalized friction fault: $k(1) = 0$ describes nominal conditions, $k(1) = 1$ introduces a friction fault equal to 300% nominal conditions.
- $k(2) \in [0,1]$ introduces the normalized backlash fault: $k(2) = 0$ means nominal backlash, equal to 0.005 rad (≅0.29 degrees), $k(2) = 1$ represents a backlash equal to 100 times nominal conditions (0.5 rad ≅ 29 degrees). It's important to underline that such backlash needs to be multiplied for the gear ratio in order to find the user play.

- $k(3), k(4), k(5) \in [0,1]$ are the short circuit parameters for the phase A, phase B and phase C respectively. $k(i) = 0$ means that the i-th phase is correctly working (the current flow through the 100% of the copper coil), $k(i) = 1$ means that the i-th phase is in a complete short-circuit situation. In the simulation, the maximum of the short circuit parameter is set to 0.99, because if two phases are in short circuit at the same time, the current will diverge to infinite. The complete short circuit situation is not very interesting for prognostics purposes, because the motor results entirely broken.
- $k(6), k(7) \in [0,1]$ describes the normalized eccentricity fault. In particular, $k(6)$ is the rotor eccentricity ratio $\zeta = \frac{x_0}{g_0}$ and if it equal to 0 means that there is not static eccentricity, if it is equal to 1 the rotor touches the stator because the air gap is decreased to zero. On the other hand, $k(7)$ represents the phase of the rotor eccentricity: $k(7) = 0$ implies a φ=-π, $k(7) = 1$ is the same of an angle φ=π. This last parameter, during the evaluation the function error, is suitably treated, because if the eccentricity is null, the eccentricity phase can assume all values between -π and π.
- $k(8) \in [0,1]$ represents the gain fault. Nominal conditions are equal to $k(8) = 0.5$; $k(8) = 0$ means that the gain is decreased of the 50%, $k(8) = 1$ means that gain is increased of the 150%.

Hence, in nominal conditions the fault vector is:

$$k = [0, 0, 0, 0, 0, 0, 0.5, 0.5] \tag{6.2}$$

When the reference model runs, one or more faults are implemented exploiting this vector: all the parameters coming out from the simulation are those related to the conditions of the faulty motor. Launching the optimization algorithm, its task is to continuously runs the monitor model in order to find the right values of these eight fault parameters previously implemented in the reference model. To achieve this goal, the algorithm has to compare the equivalent single-phase current trends of the two models.

6.2. Fitness function

The objective function to be optimized is known as fitness function: it expresses how the monitor model is approximated in a satisfying way the high fidelity one. When, every sample time, the algorithm runs the monitor model with a determined fault vector k, the single-phase current is stored in a suitable variable in Matlab's workspace. Then, this value is compared with the reference equivalent current evaluated as described at the start of chapter 6 in order to calculate the error between the two parameters. The comparison, in first approximation, were carried out exploiting the *least squares method*:

$$err = \sum_{t} \left(I_{3equiv}(t_0) - I_m(t_0) \right)^2 \tag{6.3}$$

where t_0 is a generic instant contained in the simulated time. With the (6.3), the two current trends are compared finding the minimum of the parabola described, which means that the error must remain as close possible to zero. With a deeper inspection, it should be noted that equation (6.3) resulted to be unable to recognize very small differences between the behaviour of the reference and monitor model, which could generate phase displacement in the rotor angular position measured between the two models. This inaccuracy is particularly accentuated in case of abrupt change of the commanded current.

To avoid this problem, it's possible to implement an error value calculated with the *total least squares method* (TLSM): it takes care for the error evaluation not only of the dependent variable, but also of the independent one. It was firstly introduced by Golub and Van Loan as a solution to the problem $AX \cong B$ and it is particularly useful when data stored in A and B are disturbed by a noise or generally perturbed [39]. Least-squares method and total least-squares method assess the precision of the fitting in different ways: former minimizes the sum of the squared vertical distances from the acquired data to the fitting line, latter takes care about the squared perpendicular distance from the data to the fitting curve. The figure 6.2, taken from [39] and slightly modified, shows efficiently this difference:

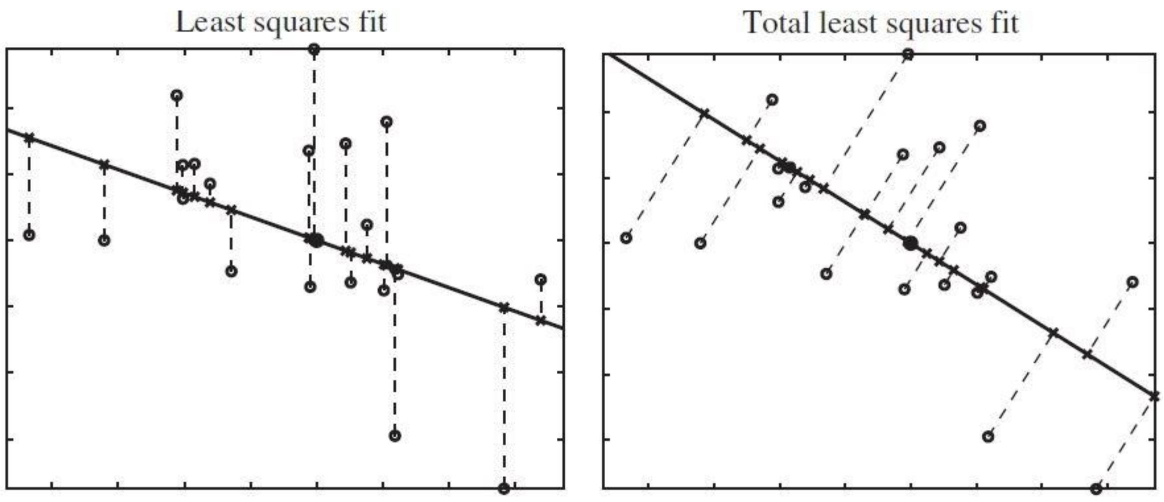

Figure 6. 2 - Differences between least-squares method and total least-squares method

The circles are the acquired data, probably coming from a sensor test campaign, the dashed lines are the distances used in the evaluation of the error and the solid black line is the approximation trend optimized.

Concerning our problem, the acquired data are the output of the reference model and the approximation is carried out by the optimization algorithms and the monitor model. Applying the TLSM, the assessment of the error is possible only evaluating the normal distance between a fit curve and a data trend (see figure 6.3).

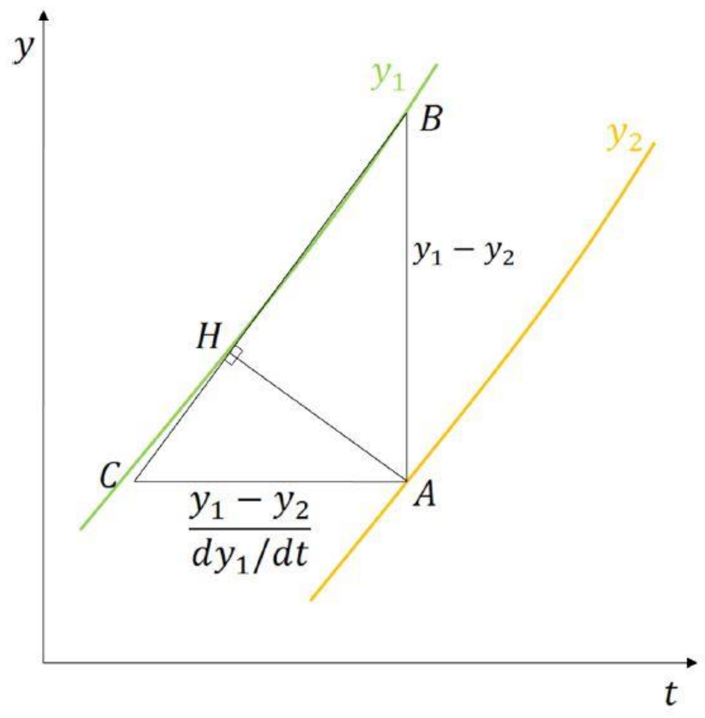

Figure 6. 3 - Total least squares method representation

Assuming $y_1 = I_{3equiv}(t)$ and $y_2 = I_m(t)$, and considering that the distance $\overline{BC}$ is small enough to approximate the curve with the segment, the length of the horizontal line could be evaluated as:

$$\overline{AC} = \frac{y_1 - y_2}{\frac{dy_1}{dt}} \tag{6.4}$$

Knowing that $\overline{AH}$ is the height of the hypotenuse, for every time instant it's possible to state that:

$$\overline{AH} = \frac{\overline{AB} \cdot \overline{AC}}{\overline{BC}} = \frac{(y_1 - y_2) \cdot \frac{y_1 - y_2}{\frac{dy_1}{dt}}}{\sqrt{(y_1 - y_2)^2 + \left(\frac{y_1 - y_2}{\frac{dy_1}{dt}}\right)^2}} = \frac{(y_1 - y_2)}{\sqrt{\frac{dy_1}{dt}^2 + 1}} \tag{6.5}$$

Hence, the error with the total least-squares method can be evaluated as:

$$err = \sum_t \frac{(y_1(t_0) - y_2(t_0))^2}{\sqrt{\frac{dy_1(t_0)}{dt}^2 + 1}} = \sum_t \frac{(I_{3equiv}(t_0) - I_m(t_0))^2}{\sqrt{\frac{dI_{3equiv}(t_0)}{dt}^2 + 1}} \tag{6.6}$$

The total error, eventually, is then multiplied for the sampling time in order to avoid its dependence from the compiling time.

A critical issue in the definition of the error is the measure unit: total least squares method uses the Pythagoras' theorem to sum two values with different measure unit. Indeed, the first member $(y_1 - y_2)$ is a distance between values with the same measure unit, the second member $\left(\frac{y_1-y_2}{\frac{dy_1}{dt}}\right)$ introduces the derivative at the denominator. To fix this problem, in literature are findable some solutions. The first is the variables normalization carried out by the analysis of the precision measurement, the second is the substitution of the normal distance with the horizontal and vertical distances' residuals. Both these solutions are affected by criticalities: precision measurement is not simple to obtain because time by time the error is forced to be equal to 0 in nominal conditions, and the second underestimates the error when the curve derivative is too small. The strategy thought is the substitution of the derivative at the denominator with its root mean square value: thereby the current has unitary average derivative.

The root mean square (RMS) value has been assessed using the suitable Matlab command "rms"; the value of the derivative can be evaluated in two different ways with the "diff" or "gradient" Matlab commands. Trying these two solutions, the results were the same, so the "diff" command has been chosen thanks to its slightly higher computational speed.

6.3. Command choice and calibration

In the next chapter, the results coming from the optimization paths are reported. All data contained in those tables represents the fault parameters resulting from a chirp command in the Simulink models. The command time history strongly affects the reliability and precision of the data, since it can change the behaviour and the dynamic response of the two models, as already seen in chapter 4.

The chirp is a signal in which frequency linearly varies (can both increase or decrease) and particularly applies to laser, radar and sonar purposes. This command has been chosen for our problem due to multiple reasons: firstly, it’s important to see the evaluation of the position, speed and current errors in the closed-loop response, in which the controller iterates time by time the difference between sets and feedbacks, in a second time this command give the possibility to analyse the changes in the motor characteristic which follow the inversion of rotation sense. With only a sense of rotation, for example in open-loop condition as carried out by the step command, some faults are not recognizable (see, for example, the dynamic response to a step command of the proportional gain fault, chapter 4.6.3). The parameters used for the chirp command during optimization process are summarized in table below.

Table 7 - Chirp parameters used in optimization process

Initial amplitude of the signal	0.005 [rad]
Initial frequency	0 [Hz]
Target frequency	15 [Hz]
Time in which the target frequency is achieved	0.5 [s]

The chirp command, due to the multiple simplifications of the monitor model, introduces a slightly different behaviour between the currents of the high-fidelity and monitor models.

Figure 6.4 depicts the trend of the current coming from the two simulations in nominal conditions.

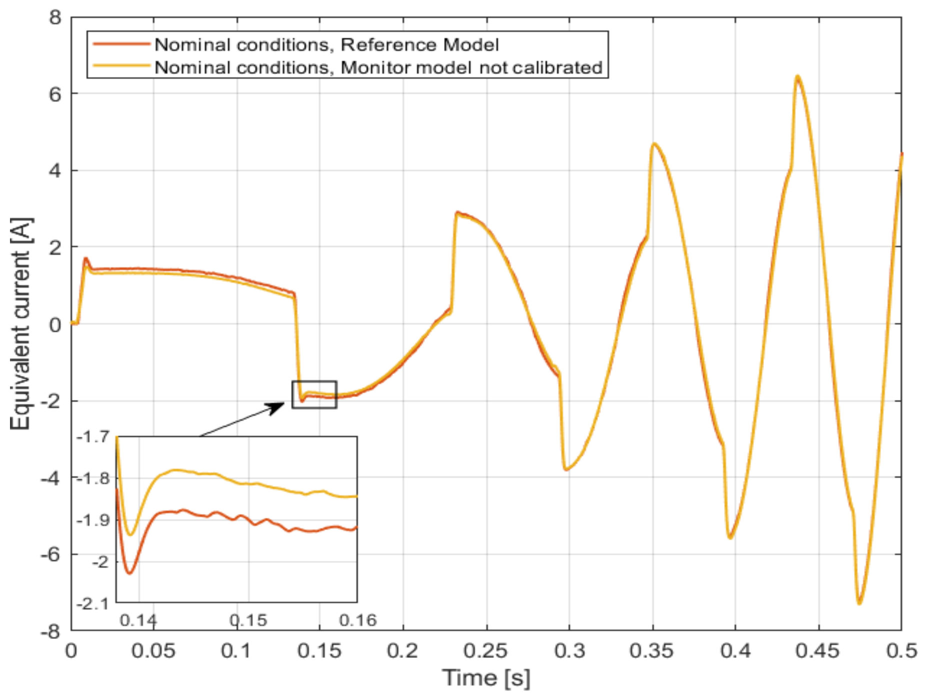

Figure 6. 4 - Comparison between nominal conditions in monitor and reference models

This static error is required to be null in nominal conditions, in order to realize the simulations of the two models from same starting trends. To achieve this goal, a static calibration has been carried out: in Matlab environment the difference between the two aforementioned trends is evaluated point by point and stored in a suitable external file as a row vector. In the fitness function, this error is recalled every iteration to evaluate the right monitor model's current.

7. Results

This chapter focuses on the optimization results obtained by the simulations performed in Matlab-Simulink environment. Matlab is the R2018a version, Simulink is the 1.61 version, calculations are managed by an Intel®Core™ i5-6200U with a CPU @2.3-2.4 GHz and 8 GB of Random Access Memory (RAM), using Window 10 Pro 64 bit.

7.1. Optimization parameters

Once the writing of the algorithms was completed, there was the necessity to find the rightest optimization parameters possible. The aim was to obtain reliable data, characterised by a high precision, in a short amount of time. To achieve this goal, a long calibration has been carried out to choose the common main parameters: the population size (number of search agents), the number of generations, the function tolerance and the parallelization.

- *Population size* is the number of individuals of each generation in Genetic Algorithm and Differential Evolution (in Particle Swarm Optimization is called the “swarm size”, in Grey wolf Optimization is the “number of search agents”). Increasing this parameter allows to achieve a better precision and a reliable data, but the time is unacceptable longer.
- *Number of generations* is the maximum number of different iterations. In every generation the best individuals are chosen as a temporary solution to the problem: if one of these solutions meets the stopping criteria, the algorithm stops its work. In PSO and GWO this parameter is called “number of iterations”, because these two algorithms do not exploit the evolution strategies of species. Increase the number of generations, produce the same results given by the population size increase, i.e. a better quality of the solutions, but an incredibly longer computational time.
- *Function tolerance* is the average relative change in the best fitness function value. Imposing this number, the algorithm stops iterating if the relative change between the best value of a generation and that of the next is less than the value set. It’s important to set a number which allows the right stop: if an iteration recalls as best value the individual of the previous generation (as DE algorithm does), whichever number has been set the iteration will stop. In this case it’s important to add another stopping criterium.

➢ *Parallelization* is the choice to split or not the computational efforts over different cores. Do not parallelize means that the evaluation is carried out in a serial mode, so the algorithm calls the fitness function on only one individual at a time; using parallel computation means that the algorithm calls multiple individuals at the same time (depending on the number of core processor) and evaluates their fitness with the objective. In our case, two individuals are investigated at the same time, thanks to the Dual Core processor. The computational time, with the parallelization on, is less than the serial process, but it requires strong modifications to DE and GWO codes in order to implement this feature.

In table 8 is reported an example of the calibration process, in this case applied to the Genetic Algorithm. The name on the left are highlighted in red if the parameters chosen do not have achieved satisfying results, in yellow if the parameters was right but the computational time or the precision can be improved, in green the optimal combination.

Table 8 – Calibration process for Genetic Algorithm

#				Population		StoppingCrit			UseParallel
Rif	#Iterations	OptTime	FAILURE	PopType	PopSize	Gen	TimeLim	FuncTol	
GA001		>2400	Na	DV	200	def	def	def	FALSE
GA002	11	540	Na	DV	100	def	def	def	FALSE
GA003	43	>2400	Na	DV	100	def	def	def	FALSE
GA004	68	>2400	Na	DV	100	def	def	def	FALSE
GA005	46	2400	Na	DV	100	300	2400	1E-05	FALSE
GA006	46	2400	Na	DV	100	300	2400	1E-05	FALSE
GA007	200	2091	Na	DV	20	200	def	1E-09	FALSE
GA008	200	1388	Bklsh	DV	20	200	def	1E-09	FALSE
GA009	200	2738	Bklsh	DV	40	200	def	1E-09	FALSE
GA010	200	970	Bklsh	DV	20	200	def	1E-09	TRUE
GA011	143	1097	Na	DV	20	200	def	1E-09	TRUE
GA012	200	2175	Gain	DV	50	200	def	1E-09	TRUE

Yellow cells are the modified parameters: the aim of each try is to test the effectiveness of the modification implemented. The “def” label indicates Matlab’s default values:

➢ number of generations is set to 100·(variables number=8). These number is constantly decreased to drop the optimization time;
➢ limit of time is set to default as infinite: I’ve introduced it only twice to see how many optimizations the algorithm could complete in 40 minutes.

- Function tolerance is set to 10^{-6} (1E-06), and after some tries it has been set to 10^{-9}.

It's important to underline that after the sixth attempt (GA006), due to the small number of iterations carried out in a long time, I've suitably modified the fitness function, getting rid of all code rows which only introduce additional computational time and do not provide essential outputs. It's possible to see how the number of iterations per minute moves from 1.15 of GA006 to 5.74 of GA007.

After calibration process, the population size is set to 50, the maximum number of generations is set to 200, the function tolerance is set to 10^{-9} for GA and PSO, 10^{-12} for DE and 10^{-16} for GWO, and the parallelization is set to true. The stopping criterium has been developed taking care of the number of the actual iteration, the value of the objective function (in our case the error) and the difference between the error of one iteration and the same value in the previous iteration. The code developed could be summarized with the pseudo-code below:

```
if (Iteration number > 20 && Error value < $10^{-3}$)
    criterium=error value(previous iteration)-error value(current
    iteration)
    if criterium < Function tolerance
        break (stop the algorithm's for loop)
    end if
end if
```

The iteration number and the error value used in the first `if` have been chosen after some attempts. Using the "display" option of the algorithms, with which it's possible to see the objective function evaluated for each iteration, it was clear that from the thirtieth generation the value of the error slightly changes, arriving around $5 \cdot 10^{-4}$. Setting this value as error goal in the first `if`, the iterations never stop, because only in some random conditions the algorithm reaches this level of precision. The value 10^{-3} is the result of a compromise, because if the fault is high (> 0.7) the optimization makes a greater effort than if the fault is low (< 0.25), because it starts from nominal condition (usually 0). The 10^{-3} value allows to stop pretty always the low fault detection and sometimes the high fault detection.

Differential Evolution is a particular case, because all other algorithms has implemented determined mutation coefficient or random number in order to explore all function domain, DE instead requires the setting of the mutation factor F and cross-over ratio CR. In literature, it's possible to find some papers oriented to the research on the optimal combination of these two parameters. Using the suggestions coming from [40], in first approximation the mutation factor $F = 0.5$ and cross-over ratio $CR = 0.9$ are set. With these values, ten different optimizations are carried out, in order to find the mean percentual error (see next paragraph for the definition of the error) and the average computational time. After some attempts in which 15 different combination are tested (so the overall number of optimizations has been 150), the chosen parameters are $F = 0.9$ and $CR = 0.95$. In tables below the differences concerning the precision and the computational time between two set of coefficients are reported. It's clear that both in terms of average percentual error and compiling time, the second combination of options is strongly better than the first, in which in only one case the error is less than the 2%. The computational time strongly drop from the first case and the second because the parallelization has been implemented. It's important to underline that in both optimizations shown below, the stopping criterium was not implemented yet.

Table 9 - Low friction fault detection with DE optimization (F=0.5 and CR=0.9)

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE Friction Low fault F=0.5 CR=0.9	# 1	0,2181	0,0041	0,0147	0,0653	0,0051	0,0324	0,5214	0,4327	3793	10,545	200	Avg % error
	# 2	0,2450	0,0030	0,0006	0,0004	0,0024	0,0061	0,0464	0,4804	3753	2,166	200	
	# 3	0,2472	0,0010	0,0002	0,0000	0,0007	0,0027	0,9856	0,4910	3749	0,997	200	9,12
	# 4	0,2269	0,0120	0,0165	0,0007	0,0142	0,0455	0,0775	0,4226	3760	9,789	200	
	# 5	0,2320	0,0009	0,0030	0,0442	0,0005	0,0220	0,5757	0,4798	3771	5,641	200	
	# 6	0,1420	0,0117	0,1368	0,0519	0,0135	0,0212	0,5405	0,2631	3782	29,994	200	Avg comp. Time
	# 7	0,2263	0,0061	0,0216	0,0075	0,0168	0,0094	0,5385	0,4538	3779	6,022	200	
	# 8	0,2374	0,0072	0,0001	0,0198	0,0021	0,0056	0,9733	0,4702	3793	3,916	200	3773
	# 9	0,2398	0,0038	0,0001	0,0000	0,0151	0,0027	0,3701	0,4761	3768	3,042	200	
	# 10	0,1870	0,0061	0,0779	0,0344	0,0452	0,1032	0,4411	0,3892	3787	19,043	200	

Table 10 - Low friction fault detection with DE optimization (F=0.9 and CR=0.95)

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE Friction Low fault F=0.9 CR=0.95	# 1	0,2462	0,0020	0,0001	0,0000	0,0007	0,0026	0,1874	0,4888	2692	1,232	200	Avg % error
	# 2	0,2477	0,0003	0,0000	0,0004	0,0002	0,0028	0,1403	0,4929	2582	0,805	200	
	# 3	0,2459	0,0026	0,0002	0,0001	0,0005	0,0055	0,0513	0,4885	2621	1,387	200	1,12
	# 4	0,2479	0,0011	0,0000	0,0002	0,0000	0,0023	0,9437	0,4880	2418	1,246	200	
	# 5	0,2473	0,0004	0,0001	0,0000	0,0000	0,0023	0,1536	0,4932	2472	0,772	200	
	# 6	0,2461	0,0012	0,0002	0,0005	0,0006	0,0031	0,0698	0,4892	2472	1,203	200	Avg comp. Time
	# 7	0,2464	0,0017	0,0001	0,0007	0,0000	0,0009	0,1437	0,4882	2458	1,254	200	
	# 8	0,2461	0,0025	0,0001	0,0000	0,0001	0,0055	0,0675	0,4872	2479	1,490	200	2511
	# 9	0,2475	0,0017	0,0001	0,0006	0,0002	0,0037	0,1537	0,4930	2488	0,861	200	
	# 10	0,2471	0,0012	0,0001	0,0002	0,0001	0,0023	0,9857	0,4915	2430	0,939	200	

7.2. Faults detection

As already explained, the reference current is generated by injecting a fault vector k in the high-fidelity model; then faults detection is carried out by the different algorithms, which, using the monitor model, try to approximate as best as they can the values of each fault coefficient matching the equivalent current trend. All results obtained will be displayed as table 9 and 10: for each objective function, reported in light blue in the first row, ten different optimizations are evaluated and listed below. In each row, the percentual error is calculated with the relation above:

$$Err_{\%} = 100 \cdot \sqrt{\sum_{i=1}^{6} \left(k_i - \widehat{k_i}\right)^2 + \widehat{k_6} \cdot \left(k_7 - \widehat{k_7}\right)^2 + \left(k_8 - \widehat{k_8}\right)^2} \tag{7.1}$$

where $\boldsymbol{\widehat{k}} = [\widehat{k_1}, \widehat{k_2}, \widehat{k_3}, \widehat{k_4}, \widehat{k_5}, \widehat{k_6}, \widehat{k_7}, \widehat{k_8}]$ are the values of the reference model's fault vector. The aforementioned relation is entirely equal to a mean square error, with a slight difference in the definition of the eccentricity phase error k_7, because when the eccentricity coefficient ζ is zero, the phase eccentricity can be whatever value.

For every fault, two different objective functions are investigated: the low fault detection (with $\widehat{k_i} \leq 0.25$) and the high fault detection ($\widehat{k_i} \geq 0.7$), indicated with a bold font in the first row. Every objective function is obviously approximated by all the four optimization algorithms.

The optimization has been carried out not only for the single fault implementation, but also for the multiple fault implementation. To simulate in a more accurate way the real behaviour of an electromechanical actuator, in which faults are usually small due to the planned maintenance of the system, the multiple fault is implemented with this pseudo-code (special thanks to Pier Carlo Berri for the suggestion):

```
RandomFaultParams=rand(1,8);
RandomFaultParams(:,1:6)=RandomFaultParams(:,1:6).^7;
RandomFaultParams(:,8)=((RandomFaultParams(:,8)*2-1).^7+1)/2;
```

It defines the fault parameters randomly and then raises to exponent equal to 7 to decrease their absolute values (they are normalized); for the eccentricity and gain faults, which nominal condition are $k_i = 0.5$, the definition of the parameter is slightly different, but follows the same reasoning.

7.2.1. Single fault detection – Friction

The results for a low friction fault are now reported.

Table 11 - Low friction fault detection with Genetic Algorithm

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_001 Friction Low fault	# 1	0,2446	0,0009	0,0030	0,0023	0,0003	0,0015	0,7045	0,4851	2622	1,64	200	Avg % error
	# 2	0,2378	0,0033	0,0099	0,0059	0,0060	0,0095	0,3734	0,4685	2557	3,76	200	2,09
	# 3	0,2443	0,0032	0,0012	0,0010	0,0019	0,0058	0,8864	0,4818	2560	2,05	200	
	# 4	0,2439	0,0004	0,0038	0,0025	0,0014	0,0015	0,6673	0,4838	2556	1,80	200	
	# 5	0,2460	0,0005	0,0004	0,0009	0,0024	0,0019	0,3378	0,4861	2560	1,48	200	
	# 6	0,2419	0,0043	0,0002	0,0048	0,0054	0,0007	0,5121	0,4830	2562	2,06	200	Avg comp. Time
	# 7	0,2443	0,0024	0,0012	0,0003	0,0010	0,0075	0,9905	0,4829	2571	2,01	200	2633
	# 8	0,2445	0,0001	0,0010	0,0100	0,0003	0,0135	0,6144	0,4889	2568	2,10	200	
	# 9	0,2456	0,0038	0,0001	0,0024	0,0002	0,0053	0,0766	0,4838	2719	1,83	200	
	# 10	0,2418	0,0014	0,0034	0,0081	0,0024	0,0064	0,5147	0,4829	3055	2,20	200	

Table 12 - Low friction fault detection with Particle Swarm Optimization

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_001 Friction Low fault	# 1	0,2478	0,0001	0,0000	0,0002	0,0001	0,0015	0,9263	0,4946	1953	0,60	152	Avg % error
	# 2	0,2478	0,0016	0,0000	0,0000	0,0000	0,0004	0,1100	0,4912	2565	0,92	200	0,93
	# 3	0,2481	0,0000	0,0000	0,0001	0,0000	0,0000	0,7033	0,4914	966	0,89	75	
	# 4	0,2481	0,0017	0,0000	0,0000	0,0000	0,0001	0,3409	0,4869	1159	1,33	91	
	# 5	0,2487	0,0008	0,0000	0,0000	0,0000	0,0005	0,8649	0,4961	1820	0,43	140	
	# 6	0,2481	0,0002	0,0000	0,0001	0,0000	0,0004	0,9995	0,4912	1171	0,91	92	Avg comp. Time
	# 7	0,2479	0,0013	0,0000	0,0000	0,0000	0,0000	0,6329	0,4912	2175	0,92	169	1662
	# 8	0,2476	0,0000	0,0000	0,0000	0,0000	0,0042	0,1160	0,4912	896	1,02	69	
	# 9	0,2479	0,0015	0,0000	0,0001	0,0000	0,0019	0,9266	0,4879	2006	1,26	156	
	# 10	0,2478	0,0010	0,0000	0,0001	0,0000	0,0002	0,0677	0,4899	1911	1,03	148	

Table 13 - Low friction fault detection with Differential Evolution

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_001 Friction Low fault	# 1	0,2420	0,0080	0,0033	0,0019	0,0122	0,0052	0,0630	0,4908	414	2,03	26	Avg % error
	# 2	0,2428	0,0040	0,0008	0,0070	0,0024	0,0064	0,6693	0,4843	577	2,03	45	2,50
	# 3	0,2409	0,0036	0,0062	0,0106	0,0049	0,0051	0,2256	0,4743	363	3,10	28	
	# 4	0,2395	0,0027	0,0038	0,0010	0,0091	0,0062	0,2565	0,4864	420	2,10	33	
	# 5	0,2362	0,0011	0,0011	0,0102	0,0026	0,0006	0,8173	0,4758	444	2,98	35	
	# 6	0,2536	0,0039	0,0013	0,0040	0,0001	0,0002	0,2549	0,4831	293	1,82	22	Avg comp. Time
	# 7	0,2487	0,0044	0,0007	0,0006	0,0082	0,0113	0,1880	0,4783	339	2,65	27	393
	# 8	0,2418	0,0031	0,0044	0,0089	0,0008	0,0058	0,7012	0,4770	345	2,72	27	
	# 9	0,2459	0,0025	0,0003	0,0128	0,0058	0,0046	0,7530	0,4857	408	2,11	32	
	# 10	0,2454	0,0079	0,0014	0,0047	0,0016	0,0047	0,9569	0,4675	335	3,45	27	

Table 14 - Low friction fault detection with Grey Wolf Optimization

Obj.		***0,25***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_001 Friction Low fault	# 1	0,2453	0,0117	0,0000	0,0000	0,0000	0,0000	0,0013	0,4867	1478	1,83	112	Avg % error
	# 2	0,2489	0,0091	0,0000	0,0005	0,0000	0,0000	0,0000	0,4931	327	1,15	26	1,56
	# 3	0,2477	0,0000	0,0000	0,0000	0,0000	0,0000	0,0001	0,5103	564	1,06	45	
	# 4	0,2437	0,0033	0,0000	0,0000	0,0000	0,0000	0,0000	0,4994	262	0,72	21	
	# 5	0,2416	0,0001	0,0000	0,0007	0,0000	0,0019	0,0002	0,4635	349	3,76	28	
	# 6	0,2534	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,4954	592	0,57	47	Avg comp. Time
	# 7	0,2501	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,5153	267	1,53	21	484
	# 8	0,2500	0,0000	0,0003	0,0000	0,0000	0,0000	0,0000	0,4788	269	2,12	21	
	# 9	0,2484	0,0000	0,0002	0,0000	0,0000	0,0000	0,0000	0,4878	264	1,23	21	
	# 10	0,2393	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,4873	475	1,66	38	

The results coming from a high friction fault are now reported.

Table 15 - High friction fault detection with Genetic Algorithm

Obj.		***0,75***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_002 Friction High fault	# 1	0,7307	0,0051	0,0000	0,0040	0,0024	0,0055	0,9739	0,4665	2589	3,98	200	Avg % error
	# 2	0,7337	0,0053	0,0017	0,0010	0,0019	0,0002	0,4181	0,4717	2494	3,32	200	4,49
	# 3	0,7337	0,0012	0,0003	0,0009	0,0022	0,0081	0,2181	0,4641	2495	4,03	200	
	# 4	0,7327	0,0037	0,0021	0,0005	0,0012	0,0019	0,6858	0,4643	2491	3,99	200	
	# 5	0,7205	0,0146	0,0195	0,0012	0,0119	0,0094	0,2609	0,4490	2491	6,55	200	
	# 6	0,7359	0,0045	0,0016	0,0003	0,0031	0,0032	0,9954	0,4655	2490	3,79	200	Avg comp. Time
	# 7	0,7350	0,0037	0,0006	0,0029	0,0019	0,0019	0,9808	0,4650	2491	3,84	200	2501
	# 8	0,7228	0,0104	0,0123	0,0169	0,0097	0,0071	0,4346	0,4740	2490	4,58	200	
	# 9	0,7083	0,0141	0,0061	0,0179	0,0298	0,0109	0,3478	0,4428	2489	8,11	200	
	# 10	0,7365	0,0039	0,0003	0,0005	0,0012	0,0001	0,4713	0,4768	2489	2,71	200	

Table 16 - High friction fault detection with Particle Swarm Optimization

Obj.		***0,75***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_002 Friction High fault	# 1	0,7386	0,0000	0,0000	0,0000	0,0000	0,0028	0,1419	0,4767	969	2,61	77	Avg % error
	# 2	0,7376	0,0000	0,0000	0,0000	0,0000	0,0001	0,2152	0,4754	2173	2,75	174	2,77
	# 3	0,7393	0,0001	0,0000	0,0000	0,0000	0,0020	0,1293	0,4795	2038	2,32	163	
	# 4	0,7388	0,0001	0,0000	0,0000	0,0000	0,0000	0,5592	0,4780	2491	2,47	200	
	# 5	0,7369	0,0031	0,0000	0,0000	0,0000	0,0082	0,0350	0,4734	1725	3,12	138	
	# 6	0,7380	0,0018	0,0001	0,0000	0,0000	0,0000	0,3575	0,4754	2495	2,74	200	Avg comp. Time
	# 7	0,7391	0,0020	0,0000	0,0000	0,0000	0,0022	0,0322	0,4756	2094	2,69	168	1875
	# 8	0,7389	0,0012	0,0001	0,0003	0,0000	0,0049	0,2341	0,4776	918	2,55	73	
	# 9	0,7368	0,0021	0,0000	0,0000	0,0001	0,0076	0,0402	0,4700	1994	3,38	160	
	# 10	0,7361	0,0002	0,0000	0,0002	0,0000	0,0049	0,9970	0,4732	1853	3,07	148	

Table 17 - High friction fault detection with Differential Evolution

Obj.		***0,75***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_002 Friction High fault	# 1	0,7366	0,0034	0,0000	0,0000	0,0002	0,0064	0,0910	0,4732	2730	3,09	200	Avg % error
	# 2	0,7368	0,0045	0,0001	0,0000	0,0000	0,0091	0,1242	0,4702	2675	3,43	200	3,20
	# 3	0,7353	0,0025	0,0004	0,0001	0,0002	0,0040	0,0070	0,4705	2733	3,34	200	
	# 4	0,7356	0,0042	0,0001	0,0002	0,0001	0,0082	0,0495	0,4740	2654	3,13	200	
	# 5	0,7351	0,0023	0,0001	0,0002	0,0000	0,0045	0,9983	0,4702	2518	3,37	200	
	# 6	0,7363	0,0013	0,0001	0,0003	0,0001	0,0056	0,0431	0,4737	2683	3,03	200	Avg comp. Time
	# 7	0,7368	0,0031	0,0007	0,0006	0,0001	0,0073	0,0844	0,4736	2632	3,08	200	2649
	# 8	0,7366	0,0026	0,0008	0,0004	0,0002	0,0067	0,0393	0,4735	2648	3,07	200	
	# 9	0,7369	0,0042	0,0000	0,0002	0,0000	0,0088	0,1282	0,4705	2648	3,39	200	
	# 10	0,7362	0,0039	0,0000	0,0001	0,0004	0,0046	0,9955	0,4734	2573	3,07	200	

Table 18 - High friction fault detection with Grey Wolf Optimization

Obj.		***0,75***	0	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_002 Friction High fault	# 1	0,7319	0,0218	0,0071	0,0000	0,0000	0,0000	0,0000	0,4551	2649	5,35	200	Avg % error
	# 2	0,7334	0,0000	0,0000	0,0000	0,0000	0,0089	0,0350	0,4737	2704	3,26	200	3,49
	# 3	0,7345	0,0000	0,0000	0,0000	0,0000	0,0083	0,0310	0,4662	2598	3,83	200	
	# 4	0,7357	0,0000	0,0000	0,0000	0,0000	0,0058	0,0271	0,4709	2601	3,31	200	
	# 5	0,7346	0,0000	0,0000	0,0000	0,0000	0,0058	0,0270	0,4733	2582	3,15	200	
	# 6	0,7365	0,0000	0,0000	0,0000	0,0000	0,0072	0,0290	0,4705	2578	3,34	200	Avg comp. Time
	# 7	0,7366	0,0000	0,0000	0,0000	0,0000	0,0072	0,0200	0,4713	2537	3,27	200	2585
	# 8	0,7362	0,0000	0,0000	0,0000	0,0000	0,0072	0,0195	0,4702	2531	3,38	200	
	# 9	0,7373	0,0000	0,0000	0,0000	0,0000	0,0073	0,0192	0,4743	2534	2,98	200	
	# 10	0,7365	0,0000	0,0000	0,0000	0,0000	0,0076	0,0194	0,4735	2534	3,09	200	

Summarizing the average values of data obtained in table 19, it’s clear that a low value of friction fault is simpler to recognize and optimize than a high fault: indeed, both compiling time and percentual error are quite the half. This is due the nature of the optimizations, because all iterations start from nominal conditions, and the friction in a non-faulty motor is equal to 0. The quickest algorithm for low fault detection has been Differential Evolution, with around 6 minutes and half, but the best precision is achieved with Particle Swarm Optimization (less than 1%); for the high fault the best algorithm both for speed and precision has been the Particle Swarm Optimization (table 16).

Table 19 - Summary of friction fault average values

	Type	Avg Comp. Time (s)	% error
Friction	Low fault	1293	1,77
	High Fault	2403	3,49

7.2.2. Single fault detection – Backlash

As already done for the friction fault, now the results of optimization of backlash fault are listed below.

Table 20 - Low backlash fault detection with Genetic Algorithm

Obj.		0	***<u>0,0909</u>***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_003 Backlash Low fault	# 1	0,0022	0,0881	0,0008	0,0023	0,0034	0,0147	0,4823	0,5094	2552	1,83	200	Avg % error
	# 2	0,0012	0,0860	0,0006	0,0012	0,0004	0,0076	0,4937	0,5063	2316	1,12	200	1,68
	# 3	0,0024	0,0887	0,0028	0,0049	0,0024	0,0097	0,4559	0,5067	2058	1,37	178	
	# 4	0,0008	0,0891	0,0006	0,0028	0,0027	0,0114	0,5152	0,5122	2315	1,73	200	
	# 5	0,0006	0,0866	0,0011	0,0019	0,0007	0,0067	0,5189	0,5064	2440	1,05	200	
	# 6	0,0042	0,0875	0,0002	0,0084	0,0007	0,0110	0,4524	0,5267	2400	3,06	196	Avg comp. Time
	# 7	0,0002	0,0872	0,0017	0,0026	0,0075	0,0092	0,4447	0,5049	2445	1,38	200	2336
	# 8	0,0000	0,0871	0,0036	0,0078	0,0018	0,0140	0,4908	0,5026	1919	1,71	157	
	# 9	0,0010	0,0870	0,0026	0,0038	0,0015	0,0063	0,4864	0,5066	2452	1,11	200	
	# 10	0,0000	0,0873	0,0001	0,0090	0,0062	0,0186	0,5073	0,5104	2467	2,42	200	

Table 21 - Low backlash fault detection with Particle Swarm Optimization

Obj.		0	***<u>0,0909</u>***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_003 Backlash Low fault	# 1	0,0000	0,0881	0,0000	0,0001	0,0001	0,0009	0,3684	0,5004	1376	0,30	118	Avg % error
	# 2	0,0000	0,0880	0,0000	0,0000	0,0000	0,0000	0,0234	0,5006	726	0,30	62	0,29
	# 3	0,0000	0,0885	0,0001	0,0002	0,0001	0,0000	0,9767	0,4999	1125	0,25	97	
	# 4	0,0000	0,0879	0,0009	0,0000	0,0000	0,0015	0,5617	0,5003	1185	0,35	102	
	# 5	0,0001	0,0880	0,0001	0,0002	0,0000	0,0007	0,1135	0,5006	1221	0,30	99	
	# 6	0,0000	0,0884	0,0002	0,0000	0,0000	0,0001	0,9311	0,5015	1178	0,29	96	Avg comp. Time
	# 7	0,0000	0,0885	0,0001	0,0000	0,0003	0,0000	0,9999	0,4998	1104	0,24	90	1242
	# 8	0,0002	0,0877	0,0008	0,0000	0,0001	0,0020	0,4326	0,5005	1981	0,39	162	
	# 9	0,0000	0,0885	0,0002	0,0001	0,0000	0,0004	0,3193	0,5001	1109	0,25	90	
	# 10	0,0000	0,0883	0,0000	0,0000	0,0000	0,0005	0,0847	0,5006	1413	0,27	114	

Table 22 - Low backlash fault detection with Differential Evolution

Obj.		0	***<u>0,0909</u>***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_003 Backlash Low fault	# 1	0,0072	0,0956	0,0003	0,0044	0,0052	0,0071	0,5636	0,4830	407	2,15	26	Avg % error
	# 2	0,0045	0,0730	0,0052	0,0253	0,0113	0,0742	0,5831	0,5176	299	8,36	21	3,93
	# 3	0,0108	0,0826	0,0063	0,0067	0,0215	0,0110	0,3518	0,5068	290	3,01	22	
	# 4	0,0006	0,0996	0,0008	0,0030	0,0275	0,0277	0,4377	0,4890	296	4,16	22	
	# 5	0,0011	0,0876	0,0051	0,0180	0,0157	0,0451	0,5171	0,5008	290	5,14	21	
	# 6	0,0160	0,0925	0,0062	0,0021	0,0086	0,0035	0,3176	0,5163	549	2,55	59	Avg comp. Time
	# 7	0,0113	0,0749	0,0024	0,0160	0,0120	0,0355	0,3931	0,5227	303	5,08	23	334
	# 8	0,0017	0,0729	0,0305	0,0001	0,0083	0,0072	0,8085	0,5173	282	4,10	21	
	# 9	0,0029	0,0970	0,0144	0,0050	0,0035	0,0106	0,4829	0,4999	352	2,01	27	
	# 10	0,0012	0,1023	0,0089	0,0104	0,0088	0,0036	0,5291	0,5184	268	2,73	21	

Table 23 - Low backlash fault detection with Grey Wolf Optimization

Obj.		0	***0,0909***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_003 Backlash Low fault	# 1	0,0000	0,0656	0,0000	0,0000	0,0000	0,0000	0,0000	0,5413	319	4,84	21	Avg % error
	# 2	0,0000	0,1050	0,0000	0,0000	0,0000	0,0000	0,0025	0,4856	283	2,01	21	3,91
	# 3	0,0000	0,0902	0,0000	0,0009	0,0073	0,0000	0,0000	0,4655	310	3,52	21	
	# 4	0,0000	0,0822	0,0000	0,0000	0,0000	0,0000	0,0013	0,4524	286	4,84	21	
	# 5	0,0000	0,0644	0,0000	0,0000	0,0000	0,0095	0,0000	0,5161	859	3,27	64	
	# 6	0,0000	0,0608	0,0000	0,0000	0,0000	0,0018	0,0000	0,4982	311	3,02	24	Avg comp. Time
	# 7	0,0000	0,0881	0,0000	0,0000	0,0000	0,0006	0,0000	0,4865	297	1,38	23	585
	# 8	0,0000	0,0698	0,0000	0,0000	0,0000	0,0004	0,0000	0,5247	284	3,25	21	
	# 9	0,0000	0,0201	0,0000	0,0000	0,0000	0,0001	0,0000	0,5171	2610	7,29	200	
	# 10	0,0000	0,0659	0,0000	0,0000	0,0000	0,0000	0,0000	0,5515	289	5,72	23	

Now results coming from the optimization of the high backlash fault are shown.

Table 24 - High backlash fault detection with Genetic Algorithm

Obj.		0	***0,747***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_004 Backlash High fault	# 1	0,0007	0,7341	0,0026	0,0009	0,0011	0,0026	0,2601	0,4949	2615	1,48	200	Avg % error
	# 2	0,0002	0,7358	0,0014	0,0009	0,0039	0,0111	0,0528	0,4973	2557	1,75	200	2,25
	# 3	0,0018	0,7335	0,0039	0,0012	0,0012	0,0089	0,0979	0,4964	2579	1,79	200	
	# 4	0,0005	0,7438	0,0014	0,0076	0,0011	0,0293	0,9921	0,4977	2656	3,39	200	
	# 5	0,0004	0,7404	0,0005	0,0072	0,0000	0,0274	0,0351	0,5028	2671	3,20	200	
	# 6	0,0014	0,7339	0,0008	0,0007	0,0015	0,0078	0,9651	0,4959	2558	1,67	200	Avg comp. Time
	# 7	0,0018	0,7406	0,0001	0,0004	0,0010	0,0140	0,9946	0,5002	2552	1,72	200	2524
	# 8	0,0018	0,7380	0,0004	0,0011	0,0031	0,0128	1,0000	0,4987	2351	1,76	182	
	# 9	0,0009	0,7386	0,0007	0,0028	0,0009	0,0175	0,9984	0,5007	2548	2,17	200	
	# 10	0,0002	0,7470	0,0037	0,0002	0,0010	0,0317	0,0156	0,4978	2155	3,55	169	

Table 25 - High backlash fault detection with Particle Swarm Optimization

Obj.		0	***0,747***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_004 Backlash High fault	# 1	0,0000	0,7299	0,0019	0,0000	0,0003	0,0000	0,1664	0,4954	1005	1,82	78	Avg % error
	# 2	0,0002	0,7356	0,0000	0,0000	0,0000	0,0069	0,0560	0,4989	2249	1,40	176	1,63
	# 3	0,0000	0,7298	0,0000	0,0010	0,0010	0,0000	0,6263	0,4954	1963	1,83	153	
	# 4	0,0000	0,7317	0,0000	0,0000	0,0000	0,0001	0,9926	0,4973	1211	1,60	88	
	# 5	0,0000	0,7332	0,0008	0,0013	0,0000	0,0063	0,0123	0,4982	1981	1,60	155	
	# 6	0,0002	0,7331	0,0000	0,0064	0,0001	0,0053	0,9815	0,4963	1673	1,71	131	Avg comp. Time
	# 7	0,0001	0,7326	0,0000	0,0000	0,0000	0,0001	0,3726	0,4975	1419	1,50	111	1634
	# 8	0,0000	0,7330	0,0000	0,0004	0,0005	0,0072	0,0590	0,4970	1562	1,67	122	
	# 9	0,0001	0,7326	0,0000	0,0000	0,0000	0,0000	0,9962	0,4976	1728	1,50	135	
	# 10	0,0000	0,7325	0,0000	0,0000	0,0010	0,0062	0,0629	0,4977	1547	1,66	121	

Table 26 - High backlash fault detection with Differential Evolution

Obj.		***<u>0</u>***	0,747	0	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_004 Backlash High fault	# 1	0,0041	0,7264	0,0115	0,0070	0,0054	0,0244	0,3770	0,4914	383	3,67	27	Avg % error
	# 2	0,0005	0,7608	0,0033	0,0247	0,0045	0,0181	0,1349	0,4815	317	3,91	24	3,92
	# 3	0,0064	0,7385	0,0316	0,0110	0,0060	0,0185	0,9579	0,5007	328	4,11	26	
	# 4	0,0010	0,7484	0,0122	0,0008	0,0025	0,0473	0,4070	0,4763	335	5,46	26	
	# 5	0,0080	0,7603	0,0054	0,0194	0,0067	0,0210	0,9578	0,4864	358	3,74	29	
	# 6	0,0052	0,7368	0,0066	0,0220	0,0213	0,0020	0,7914	0,5117	265	3,55	20	Avg comp. Time
	# 7	0,0027	0,7346	0,0056	0,0137	0,0231	0,0164	0,3190	0,4973	297	3,48	24	323
	# 8	0,0123	0,7629	0,0061	0,0043	0,0176	0,0022	0,9359	0,4814	312	3,33	25	
	# 9	0,0006	0,7559	0,0058	0,0097	0,0075	0,0160	0,0918	0,5083	350	2,50	29	
	# 10	0,0003	0,7281	0,0279	0,0081	0,0314	0,0206	0,0306	0,4854	280	5,42	22	

Table 27 - High backlash fault detection with Grey Wolf Optimization

Obj.		0	***<u>0,747</u>***	0	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_004 Backlash High fault	# 1	0,0039	0,7510	0,0035	0,0192	0,0000	0,0000	0,0208	0,5270	556	3,37	39	Avg % error
	# 2	0,0030	0,7330	0,0020	0,0052	0,0002	0,0004	0,0182	0,4632	279	4,00	23	3,80
	# 3	0,0003	0,7216	0,0055	0,0000	0,0029	0,0000	0,0000	0,5145	265	3,03	21	
	# 4	0,0000	0,6857	0,0000	0,0000	0,0002	0,0000	0,0060	0,5104	633	6,25	49	
	# 5	0,0003	0,7124	0,0000	0,0000	0,0000	0,0000	0,0000	0,4778	378	4,15	29	
	# 6	0,0000	0,6857	0,0000	0,0000	0,0001	0,0030	0,0000	0,4881	671	6,30	53	Avg comp. Time
	# 7	0,0000	0,7142	0,0000	0,0000	0,0000	0,0015	0,0000	0,4856	515	3,62	41	476
	# 8	0,0000	0,7699	0,0000	0,0000	0,0000	0,0014	0,0000	0,4966	330	2,29	26	
	# 9	0,0000	0,7322	0,0000	0,0000	0,0000	0,0000	0,0000	0,5283	251	3,21	21	
	# 10	0,0009	0,7309	0,0000	0,0000	0,0000	0,0000	0,0000	0,4942	882	1,75	68	

As it's summarised in tables above, backlash is a fault relatively simple to detect. For this reason, the average computational time is low both for low and high fault, with a value slightly greater than 1100 s; the average error is comparable too. Concerning the time, the best choice is Differential Evolution Algorithm, both for low and high fault; speaking about the precision, Particle Swarm Optimization provides the best performance, with an incredible 0.3% of percentual error in low fault detection.

Table 28 - Summary of backlash fault average values

	Type	Avg Comp. Time (s)	% error
Backlash	Low fault	1124	2,45
	High Fault	1239	2,90

7.2.3. Single fault detection – Short Circuit

Obviously, the three parameters $k(3), k(4), k(5)$ act in the same way into the model. For this reason, only the short circuit of the phase A ($k(3)$) has been tested.

Table 29 - Low short circuit fault detection with Genetic Algorithm

Obj.		0	0	**0,2**	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_005 Short Circuit Low fault	# 1	0,0007	0,0037	0,1875	0,0063	0,0002	0,0092	0,9737	0,5002	1672	1,77	200	Avg % error
	# 2	0,0001	0,0003	0,1722	0,0144	0,0242	0,0076	0,4304	0,5077	1662	4,11	200	5,04
	# 3	0,0073	0,0093	0,1395	0,0251	0,0066	0,0478	0,9482	0,5094	1670	8,55	200	
	# 4	0,0025	0,0032	0,1558	0,0097	0,0489	0,0074	0,9703	0,4949	1671	6,75	200	
	# 5	0,0002	0,0210	0,1553	0,0040	0,0478	0,0189	0,0894	0,4843	1666	7,35	200	
	# 6	0,0000	0,0572	0,1602	0,0128	0,0282	0,0313	0,9619	0,4482	1441	9,84	173	Avg comp. Time
	# 7	0,0020	0,0073	0,1839	0,0003	0,0051	0,0010	0,7861	0,5120	1665	2,21	200	1644
	# 8	0,0019	0,0003	0,1771	0,0126	0,0087	0,0005	0,9637	0,5050	1665	2,81	200	
	# 9	0,0001	0,0007	0,1741	0,0285	0,0065	0,0187	0,5177	0,5165	1662	4,63	200	
	# 10	0,0009	0,0077	0,1823	0,0034	0,0039	0,0116	0,9997	0,5027	1665	2,40	200	

Table 30 - Low short circuit fault detection with Particle Swarm Optimization

Obj.		0	0	**0,2**	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_005 Short Circuit Low fault	# 1	0,0000	0,0000	0,1960	0,0000	0,0000	0,0031	0,0000	0,4963	1688	0,65	112	Avg % error
	# 2	0,0000	0,0000	0,1917	0,0000	0,0001	0,0060	0,9683	0,4997	1399	1,06	87	0,86
	# 3	0,0000	0,0000	0,1965	0,0000	0,0000	0,0000	0,0348	0,4997	1507	0,35	113	
	# 4	0,0000	0,0010	0,1943	0,0001	0,0003	0,0069	0,1069	0,4968	3215	0,99	200	
	# 5	0,0000	0,0005	0,1945	0,0000	0,0005	0,0056	0,1125	0,4976	1007	0,86	78	
	# 6	0,0000	0,0012	0,1925	0,0000	0,0000	0,0050	0,0206	0,4969	1635	0,99	130	Avg comp. Time
	# 7	0,0000	0,0000	0,1942	0,0000	0,0000	0,0000	0,0000	0,5039	1120	0,70	89	1695
	# 8	0,0000	0,0000	0,1951	0,0000	0,0000	0,0042	0,0470	0,4950	1691	0,84	134	
	# 9	0,0000	0,0000	0,1919	0,0000	0,0003	0,0052	0,9578	0,5000	1622	0,99	125	
	# 10	0,0000	0,0000	0,1913	0,0000	0,0002	0,0068	0,9583	0,4962	2061	1,21	166	

Table 31 - Low short circuit fault detection with Differential Evolution

Obj.		0	0	**0,2**	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_005 Short Circuit Low fault	# 1	0,0026	0,0008	0,1886	0,0016	0,0027	0,0056	0,1016	0,5154	830	2,05	33	Avg % error
	# 2	0,0015	0,0015	0,1789	0,0168	0,0006	0,0123	0,8214	0,5078	768	3,09	36	2,44
	# 3	0,0051	0,0012	0,1768	0,0046	0,0107	0,0015	0,6343	0,5141	577	3,00	28	
	# 4	0,0010	0,0026	0,1904	0,0081	0,0065	0,0014	0,7832	0,4929	642	1,62	31	
	# 5	0,0015	0,0008	0,1825	0,0012	0,0026	0,0066	0,1345	0,5185	440	2,66	23	
	# 6	0,0001	0,0011	0,1991	0,0052	0,0006	0,0032	0,0522	0,4730	428	2,77	26	Avg comp. Time
	# 7	0,0002	0,0052	0,1919	0,0064	0,0019	0,0014	0,1134	0,4978	486	1,20	31	540
	# 8	0,0003	0,0039	0,1921	0,0170	0,0089	0,0030	0,5972	0,5096	346	2,34	21	
	# 9	0,0014	0,0010	0,1818	0,0005	0,0043	0,0132	0,9578	0,5008	503	2,38	34	
	# 10	0,0005	0,0015	0,1862	0,0014	0,0002	0,0273	0,1351	0,5052	380	3,27	23	

Table 32 - Low short circuit fault detection with Grey Wolf Optimization

Obj.		0	0	**<u>0,2</u>**	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_005 Short Circuit Low fault	# 1	0,0000	0,0000	0,1976	0,0000	0,0000	0,0000	0,0000	0,5340	635	3,41	44	Avg % error
	# 2	0,0000	0,0037	0,1793	0,0005	0,0000	0,0168	0,0122	0,5336	455	4,39	35	2,53
	# 3	0,0000	0,0006	0,1828	0,0005	0,0000	0,0131	0,0017	0,5081	324	2,40	25	
	# 4	0,0000	0,0000	0,1955	0,0000	0,0000	0,0000	0,0000	0,5126	274	1,34	21	
	# 5	0,0000	0,0006	0,1824	0,0000	0,0000	0,0022	0,0000	0,4735	278	3,19	21	
	# 6	0,0000	0,0000	0,1776	0,0003	0,0000	0,0000	0,0098	0,5043	472	2,28	36	Avg comp. Time
	# 7	0,0000	0,0000	0,1822	0,0000	0,0000	0,0000	0,0000	0,5064	273	1,90	21	392
	# 8	0,0000	0,0000	0,1837	0,0000	0,0000	0,0000	0,0169	0,4878	285	2,04	22	
	# 9	0,0000	0,0000	0,1916	0,0000	0,0000	0,0000	0,0000	0,5282	652	2,94	50	
	# 10	0,0000	0,0000	0,2093	0,0000	0,0000	0,0000	0,0000	0,5110	273	1,45	21	

The next four tables (32 to 35) summarize the data coming from high fault optimizations.

Table 33 - High short circuit fault detection with Genetic Algorithm

Obj.		0	0	**<u>0,7</u>**	0	0	0	0,5	0,5	Time	% Error	Generations	
GA_006 Short Circuit High fault	# 1	0,0012	0,0051	0,6595	0,0121	0,0175	0,0138	0,9937	0,5045	2638	4,88	200	Avg % error
	# 2	0,0023	0,0020	0,6665	0,0040	0,0084	0,0149	0,1233	0,5170	2675	4,20	200	6,25
	# 3	0,0044	0,0075	0,6453	0,0230	0,0282	0,0220	0,0131	0,5245	2420	7,48	178	
	# 4	0,0029	0,0043	0,6419	0,0234	0,0299	0,0198	0,9892	0,5118	2697	7,39	200	
	# 5	0,0018	0,0038	0,6624	0,0186	0,0125	0,0063	0,6178	0,5134	2547	4,64	200	
	# 6	0,0014	0,0049	0,6444	0,0215	0,0348	0,0146	0,0856	0,5033	2545	7,11	200	Avg comp. Time
	# 7	0,0044	0,0072	0,6476	0,0230	0,0231	0,0169	0,0155	0,5119	2541	6,61	200	2559
	# 8	0,0016	0,0028	0,6606	0,0047	0,0132	0,0094	0,0702	0,5079	2565	4,38	200	
	# 9	0,0010	0,0012	0,6435	0,0305	0,0309	0,0022	0,9766	0,5108	2541	7,22	200	
	# 10	0,0002	0,0030	0,6340	0,0378	0,0385	0,0077	0,1327	0,5101	2419	8,63	187	

Table 34 - High short circuit fault detection with Particle Swarm Optimization

Obj.		0	0	**<u>0,7</u>**	0	0	0	0,5	0,5	Time	% Error	Generations	
PSO_006 Short Circuit High fault	# 1	0,0017	0,0017	0,6745	0,0000	0,0000	0,0077	0,0000	0,5063	1241	2,77	92	Avg % error
	# 2	0,0000	0,0021	0,6760	0,0000	0,0002	0,0082	1,0000	0,5070	2375	2,67	177	2,73
	# 3	0,0000	0,0031	0,6763	0,0001	0,0000	0,0081	0,9990	0,5031	1358	2,58	96	
	# 4	0,0000	0,0000	0,6770	0,0006	0,0000	0,0000	0,9831	0,5074	1013	2,41	74	
	# 5	0,0002	0,0000	0,6706	0,0022	0,0037	0,0093	0,0510	0,5111	1101	3,33	86	
	# 6	0,0001	0,0025	0,6756	0,0014	0,0024	0,0098	0,0369	0,5036	1731	2,71	136	Avg comp. Time
	# 7	0,0021	0,0000	0,6769	0,0000	0,0000	0,0002	0,8760	0,5144	2535	2,73	200	1474
	# 8	0,0000	0,0005	0,6771	0,0001	0,0007	0,0001	0,9982	0,5113	1363	2,56	107	
	# 9	0,0000	0,0023	0,6740	0,0048	0,0019	0,0069	1,0000	0,5065	857	2,85	67	
	# 10	0,0000	0,0014	0,6763	0,0000	0,0000	0,0092	0,0247	0,5090	1161	2,73	91	

Table 35 - High short circuit fault detection with Differential Evolution

Obj.		0	0	***0,7***	0	0	0	0,5	0,5	Time	% Error	Generations	
DE_006 Short Circuit High fault	# 1	0,0020	0,0079	0,6565	0,0191	0,0151	0,0135	0,0285	0,5094	590	5,35	34	Avg % error
	# 2	0,0063	0,0041	0,6582	0,0172	0,0197	0,0004	0,8104	0,5281	529	5,73	35	4,78
	# 3	0,0011	0,0024	0,6685	0,0075	0,0044	0,0123	0,0533	0,5139	517	3,81	32	
	# 4	0,0018	0,0039	0,6603	0,0099	0,0211	0,0168	0,9897	0,5112	634	5,12	43	
	# 5	0,0005	0,0060	0,6734	0,0136	0,0007	0,0057	0,9784	0,4952	552	3,15	35	
	# 6	0,0036	0,0054	0,6581	0,0169	0,0026	0,0256	0,9927	0,4964	623	5,40	42	Avg comp. Time
	# 7	0,0007	0,0016	0,6507	0,0176	0,0144	0,0083	0,0262	0,5216	466	5,92	29	534
	# 8	0,0007	0,0072	0,6613	0,0132	0,0159	0,0026	0,2807	0,4916	479	4,53	30	
	# 9	0,0019	0,0097	0,6652	0,0140	0,0130	0,0084	0,1137	0,5211	465	4,69	30	
	# 10	0,0014	0,0057	0,6658	0,0126	0,0135	0,0017	0,3572	0,4892	482	4,08	32	

Table 36 - High short circuit fault detection with Grey Wolf Optimization

Obj.		0	0	***0,7***	0	0	0	0,5	0,5	Time	% Error	Generations	
GWO_006 Short Circuit High fault	# 1	0,0000	0,0000	0,6641	0,0241	0,0000	0,0000	0,0738	0,5168	1800	4,64	124	Avg % error
	# 2	0,0000	0,0000	0,6736	0,0000	0,0000	0,0000	0,0000	0,4812	295	3,24	21	3,49
	# 3	0,0001	0,0039	0,6805	0,0000	0,0000	0,0000	0,0000	0,4936	816	2,10	58	
	# 4	0,0000	0,0000	0,6705	0,0000	0,0000	0,0000	0,0000	0,5386	419	4,85	30	
	# 5	0,0000	0,0000	0,6722	0,0000	0,0000	0,0327	0,0000	0,5061	293	4,63	21	
	# 6	0,0000	0,0031	0,6750	0,0049	0,0000	0,0000	0,0110	0,5250	990	3,58	70	Avg comp. Time
	# 7	0,0008	0,0000	0,6659	0,0000	0,0000	0,0000	0,0000	0,5189	943	3,90	67	655
	# 8	0,0011	0,0000	0,6847	0,0000	0,0000	0,0000	0,0000	0,4824	292	2,33	21	
	# 9	0,0000	0,0000	0,6823	0,0000	0,0000	0,0000	0,0430	0,4781	406	2,82	29	
	# 10	0,0000	0,0000	0,6821	0,0010	0,0000	0,0000	0,0000	0,5211	299	2,77	21	

As it's possible to observe in tables above and in the summarizing table below, the average computational time is slightly different, but the percentual error maintains a big gap between the more precise low fault and the high fault. Particle Swarm Optimization has been the best algorithm in terms of precision, with a percentual error less than 1% in low fault detection and 2.7% in high fault detection. The quickest algorithms are Grey Wolf Optimization for low fault and Differential Evolution for high fault.

Table 37 - Summary of short circuit fault average values

	Type	Avg Comp. time (s)	% error
Short circuit	Low Fault	1068	2,72
	High Fault	1305	4,31

7.2.4. Single fault detection – Eccentricity

To simulate the eccentricity fault, only $k(6)$ has been modified, but in the calculation of the error also $k(7)$ is taken in care (see the definition of the error in the previous paragraph).

Table 38 - Low eccentricity fault detection with Genetic Algorithm

Obj.		0	0	0	0	0	***0,25***	0,5	0,5	Time	% Error	Generations	
GA_007 Eccentricity Low fault	# 1	0,0008	0,0037	0,0002	0,0026	0,0001	0,2551	0,5002	0,4921	2511	1,05	200	Avg % error
	# 2	0,0001	0,0029	0,0006	0,0038	0,0004	0,2546	0,5028	0,4967	2564	0,75	200	3,32
	# 3	0,0004	0,0030	0,0029	0,0066	0,0016	0,2555	0,5017	0,4953	2742	1,08	178	
	# 4	0,0004	0,0291	0,0026	0,0003	0,0000	0,0000	0,0052	0,4578	2461	25,52	200	
	# 5	0,0011	0,0018	0,0000	0,0038	0,0029	0,2526	0,4997	0,4952	2615	0,76	200	
	# 6	0,0009	0,0018	0,0012	0,0029	0,0016	0,2518	0,5015	0,4969	2625	0,55	196	Avg comp. Time
	# 7	0,0028	0,0041	0,0023	0,0017	0,0001	0,2500	0,4992	0,5030	2624	0,65	200	2502
	# 8	0,0001	0,0012	0,0040	0,0084	0,0010	0,2580	0,4960	0,4893	2210	1,64	157	
	# 9	0,0000	0,0019	0,0000	0,0024	0,0001	0,2513	0,5022	0,4957	2004	0,55	200	
	# 10	0,0012	0,0014	0,0004	0,0042	0,0033	0,2521	0,4963	0,4967	2666	0,70	200	

Table 39 - Low eccentricity fault detection with Particle Swarm Optimization

Obj.		0	0	0	0	0	***0,25***	0,5	0,5	Time	% Error	Generations	
PSO_007 Eccentricity Low fault	# 1	0,0000	0,0011	0,0001	0,0000	0,0014	0,2470	0,5001	0,4932	1976	0,76	118	Avg % error
	# 2	0,0000	0,0009	0,0000	0,0000	0,0000	0,2494	0,4980	0,4987	901	0,18	62	1,01
	# 3	0,0000	0,0008	0,0003	0,0008	0,0018	0,2498	0,4968	0,4989	2201	0,26	97	
	# 4	0,0015	0,0027	0,0000	0,0000	0,0000	0,2503	0,4993	0,4966	2108	0,46	102	
	# 5	0,0001	0,0025	0,0000	0,0007	0,0000	0,2492	0,4979	0,4977	2460	0,36	99	
	# 6	0,0001	0,0299	0,0000	0,0000	0,0000	0,2451	0,4999	0,4463	2614	6,17	96	Avg comp. Time
	# 7	0,0001	0,0002	0,0000	0,0008	0,0000	0,2493	0,4978	0,4991	2615	0,15	90	2099
	# 8	0,0000	0,0001	0,0003	0,0039	0,0003	0,2517	0,4980	0,4989	2055	0,44	162	
	# 9	0,0000	0,0027	0,0000	0,0006	0,0027	0,2455	0,4951	0,4935	2615	0,89	90	
	# 10	0,0000	0,0000	0,0000	0,0002	0,0035	0,2512	0,4974	0,4973	1442	0,46	114	

Table 40 - Low eccentricity fault detection with Differential Evolution

Obj.		0	0	0	0	0	***0,25***	0,5	0,5	Time	% Error	Generations	
DE_007 Eccentricity Low fault	# 1	0,0023	0,0085	0,0022	0,0011	0,0020	0,2515	0,5179	0,4867	412	1,70	27	Avg % error
	# 2	0,0023	0,0017	0,0056	0,0096	0,0115	0,2541	0,5165	0,4720	335	3,29	24	3,62
	# 3	0,0054	0,0104	0,0004	0,0049	0,0180	0,2267	0,4973	0,4783	459	3,87	35	
	# 4	0,0138	0,0011	0,0022	0,0104	0,0013	0,2648	0,5047	0,5314	457	3,89	35	
	# 5	0,0010	0,0003	0,0055	0,0037	0,0163	0,2991	0,4983	0,5234	305	5,72	23	
	# 6	0,0036	0,0041	0,0097	0,0005	0,0252	0,2431	0,4802	0,5224	454	3,65	35	Avg comp. Time
	# 7	0,0021	0,0108	0,0093	0,0015	0,0129	0,2686	0,4949	0,5219	418	3,47	32	401
	# 8	0,0006	0,0139	0,0010	0,0027	0,0245	0,2494	0,4788	0,5031	412	2,90	31	
	# 9	0,0002	0,0158	0,0106	0,0065	0,0004	0,2065	0,4950	0,4685	364	5,74	28	
	# 10	0,0005	0,0017	0,0179	0,0031	0,0023	0,2544	0,4877	0,4963	390	1,95	30	

Table 41 - Low eccentricity fault detection with Grey Wolf Optimization

Obj.		0	0	0	0	0	***0,25***	0,5	0,5	Time	% Error	Generations	
GWO_007 Eccentricity Low fault	# 1	0,0000	0,0000	0,0097	0,0000	0,0000	0,1447	0,4994	0,4865	2747	10,66	200	Avg % error
	# 2	0,0000	0,0000	0,0107	0,0000	0,0000	0,2594	0,4812	0,5112	641	1,88	48	13,54
	# 3	0,0000	0,0000	0,0000	0,0000	0,0000	0,2044	0,4825	0,5180	1047	4,92	78	
	# 4	0,0000	0,0124	0,0000	0,0000	0,0000	0,2158	0,4858	0,4569	976	5,65	73	
	# 5	0,0000	0,0062	0,0000	0,0000	0,0000	0,2067	0,4883	0,4801	806	4,81	60	
	# 6	0,0000	0,0000	0,0000	0,0000	0,0000	0,2093	0,4848	0,4827	739	4,43	55	Avg comp. Time
	# 7	0,0000	0,0299	0,0000	0,0000	0,0000	0,0000	0,0000	0,4403	2683	25,88	200	1771
	# 8	0,0000	0,0276	0,0000	0,0000	0,0000	0,0000	0,0000	0,4471	2679	25,70	200	
	# 9	0,0000	0,0282	0,0000	0,0000	0,0000	0,0000	0,0000	0,4431	2709	25,79	200	
	# 10	0,0000	0,0272	0,0000	0,0000	0,0000	0,0000	0,0000	0,4488	2683	25,66	200	

Table 42 - High eccentricity fault detection with Genetic Algorithm

Obj.		0	0	0	0	0	***0,75***	0,5	0,5	Time	% Error	Generations	
GA_008 Eccentricity High fault	# 1	0,0044	0,0038	0,0021	0,0091	0,0069	0,7520	0,5003	0,4954	2690	1,39	200	Avg % error
	# 2	0,0045	0,0052	0,0007	0,0100	0,0056	0,7543	0,5002	0,4887	2117	1,81	167	1,78
	# 3	0,0015	0,0048	0,0001	0,0202	0,0021	0,7544	0,5007	0,4991	2526	2,14	200	
	# 4	0,0081	0,0093	0,0009	0,0101	0,0075	0,7415	0,5011	0,4880	2587	2,30	200	
	# 5	0,0002	0,0042	0,0030	0,0141	0,0028	0,7578	0,5015	0,4934	2325	1,84	183	
	# 6	0,0025	0,0090	0,0003	0,0084	0,0122	0,7497	0,4991	0,4946	2204	1,83	175	Avg comp. Time
	# 7	0,0070	0,0058	0,0026	0,0045	0,0002	0,7419	0,5013	0,4866	2514	1,89	200	2453
	# 8	0,0047	0,0058	0,0003	0,0067	0,0008	0,7505	0,5004	0,4976	2527	1,03	200	
	# 9	0,0046	0,0096	0,0037	0,0034	0,0027	0,7561	0,5019	0,4838	2521	2,12	200	
	# 10	0,0061	0,0056	0,0024	0,0092	0,0048	0,7485	0,5000	0,4955	2522	1,42	200	

Table 43 - High eccentricity fault detection with Particle Swarm Optimization

Obj.		0	0	0	0	0	***0,75***	0,5	0,5	Time	% Error	Generations	
PSO_008 Eccentricity High fault	# 1	0,0030	0,0052	0,0000	0,0140	0,0008	0,7516	0,5008	0,4914	1010	1,76	79	Avg % error
	# 2	0,0038	0,0065	0,0000	0,0062	0,0027	0,7523	0,5004	0,4925	1561	1,28	123	1,44
	# 3	0,0078	0,0019	0,0005	0,0004	0,0001	0,7634	0,4994	0,5022	1495	1,58	118	
	# 4	0,0037	0,0000	0,0028	0,0000	0,0004	0,7561	0,5002	0,5012	2520	0,78	200	
	# 5	0,0003	0,0049	0,0001	0,0132	0,0019	0,7528	0,5013	0,4957	1780	1,51	141	
	# 6	0,0072	0,0018	0,0000	0,0000	0,0013	0,7660	0,5004	0,5024	1468	1,79	116	Avg comp. Time
	# 7	0,0000	0,0042	0,0000	0,0146	0,0010	0,7571	0,5007	0,4957	1340	1,73	106	1506
	# 8	0,0087	0,0015	0,0001	0,0001	0,0002	0,7605	0,5001	0,5027	1563	1,40	124	
	# 9	0,0108	0,0000	0,0002	0,0000	0,0016	0,7537	0,4993	0,5071	814	1,36	64	
	# 10	0,0078	0,0019	0,0019	0,0030	0,0020	0,7499	0,5091	0,5040	1509	1,20	120	

Table 44 - High eccentricity fault detection with Differential Evolution

Obj.		0	0	0	0	0	***<u>0,75</u>***	0,5	0,5	Time	% Error	Generations	
DE_008 Eccentricity High fault	# 1	0,0044	0,0066	0,0002	0,0078	0,0008	0,7509	0,5001	0,4918	2889	1,38	200	Avg % error
	# 2	0,0039	0,0046	0,0006	0,0132	0,0001	0,7544	0,5002	0,4947	2810	1,61	200	
	# 3	0,0035	0,0053	0,0001	0,0082	0,0015	0,7541	0,4998	0,4936	2809	1,29	200	1,41
	# 4	0,0044	0,0071	0,0004	0,0072	0,0029	0,7515	0,4996	0,4924	2810	1,38	200	
	# 5	0,0027	0,0037	0,0002	0,0114	0,0009	0,7564	0,5002	0,4952	2809	1,47	200	
	# 6	0,0042	0,0066	0,0002	0,0082	0,0003	0,7511	0,5005	0,4919	2813	1,40	200	Avg comp. Time
	# 7	0,0047	0,0054	0,0001	0,0098	0,0012	0,7534	0,5005	0,4944	2818	1,38	200	
	# 8	0,0040	0,0049	0,0000	0,0089	0,0039	0,7528	0,5003	0,4945	2801	1,31	200	2809
	# 9	0,0033	0,0048	0,0002	0,0110	0,0027	0,7549	0,4998	0,4950	2777	1,45	200	
	# 10	0,0039	0,0051	0,0007	0,0101	0,0041	0,7533	0,5000	0,4943	2754	1,43	200	

Table 45 - High eccentricity fault detection with Grey Wolf Optimization

Obj.		0	0	0	0	0	***<u>0,75</u>***	0,5	0,5	Time	% Error	Generations	
GWO_008 Eccentricity High fault	# 1	0,0335	0,0000	0,0000	0,0000	0,0043	0,7139	0,4971	0,5236	2924	5,48	200	Avg % error
	# 2	0,0328	0,0000	0,0000	0,0000	0,0045	0,7175	0,5011	0,5235	2856	5,20	200	
	# 3	0,0132	0,0000	0,0000	0,0000	0,0042	0,7482	0,4975	0,4953	2858	1,49	200	2,30
	# 4	0,0145	0,0000	0,0000	0,0000	0,0038	0,7553	0,4991	0,5001	2862	1,59	200	
	# 5	0,0136	0,0000	0,0000	0,0000	0,0035	0,7493	0,4997	0,5029	2859	1,44	200	
	# 6	0,0135	0,0000	0,0000	0,0000	0,0036	0,7476	0,5023	0,5062	2855	1,56	200	Avg comp. Time
	# 7	0,0133	0,0000	0,0000	0,0000	0,0034	0,7473	0,5009	0,5068	2856	1,56	200	
	# 8	0,0133	0,0000	0,0000	0,0000	0,0035	0,7467	0,5003	0,5070	2855	1,58	200	2866
	# 9	0,0133	0,0000	0,0000	0,0000	0,0035	0,7486	0,5013	0,5066	2865	1,54	200	
	# 10	0,0134	0,0000	0,0000	0,0000	0,0034	0,7481	0,5020	0,5072	2867	1,58	200	

These data are afflicted by a GWO problem in the detection of low faults, in fact it is possible to see how the optimizations from the 7th to the 10th have an average percentage error of about 25%. This fact may be caused by the stabilization of the algorithm into a local minimum and the inability to overwhelm the boundary of that trend. In table below the average values are affected by this problem and the high fault detection proves to be more performant in term of precision than the low fault, but the latter is always quicker. In particular, Differential Evolution takes only about 400 s (slightly more than 6 minutes and half) to perform 10 low fault optimizations, against PSO which employs around 25 minutes to approximate the high fault parameters.

Table 46 - Summary of eccentricity fault average values

	Type	Avg Comp. Time (s)	% error
Eccentricity	Low Fault	1693	5,37
	High Fault	2409	1,73

7.2.5. Single fault detection – Gain

Table 47 - Low gain fault detection with Genetic Algorithm

Obj.		0	0	0	0	0	0	0,5	***<u>0,25</u>***	Time	% Error	Generations	
GA_009 Gain Low fault	# 1	0,0032	0,0017	0,0031	0,0051	0,0011	0,0139	0,4965	0,2602	2715	1,87	200	Avg % error
	# 2	0,0035	0,0023	0,0073	0,0054	0,0059	0,0241	0,4763	0,2601	2609	2,86	200	1,93
	# 3	0,0001	0,0076	0,0040	0,0008	0,0109	0,0175	0,3904	0,2532	2499	2,26	200	
	# 4	0,0011	0,0041	0,0002	0,0105	0,0023	0,0062	0,6193	0,2566	2469	1,47	200	
	# 5	0,0024	0,0036	0,0015	0,0010	0,0102	0,0155	0,4190	0,2560	2469	2,01	200	
	# 6	0,0019	0,0014	0,0026	0,0023	0,0010	0,0088	0,4568	0,2594	2245	1,36	182	Avg comp. Time
	# 7	0,0028	0,0003	0,0018	0,0025	0,0106	0,0227	0,4936	0,2582	2469	2,67	200	2492
	# 8	0,0018	0,0014	0,0004	0,0047	0,0018	0,0138	0,4714	0,2593	2469	1,76	200	
	# 9	0,0031	0,0023	0,0012	0,0037	0,0036	0,0135	0,5350	0,2572	2474	1,66	200	
	# 10	0,0015	0,0020	0,0028	0,0020	0,0093	0,0086	0,4446	0,2525	2506	1,36	200	

Table 48 - Low gain fault detection with Particle Swarm Optimization

Obj.		0	0	0	0	0	0	0,5	***<u>0,25</u>***	Time	% Error	Generations	
PSO_009 Gain Low fault	# 1	0,0000	0,0000	0,0004	0,0015	0,0000	0,0024	0,8937	0,2527	1944	0,40	147	Avg % error
	# 2	0,0000	0,0021	0,0002	0,0011	0,0001	0,0048	0,9776	0,2499	872	0,58	67	0,45
	# 3	0,0002	0,0022	0,0021	0,0026	0,0009	0,0000	1,0000	0,2503	1435	0,41	116	
	# 4	0,0017	0,0001	0,0000	0,0000	0,0000	0,0000	0,1586	0,2525	1006	0,31	81	
	# 5	0,0017	0,0000	0,0000	0,0000	0,0000	0,0000	0,5943	0,2525	1297	0,31	105	
	# 6	0,0000	0,0026	0,0040	0,0010	0,0001	0,0000	0,7587	0,2503	1239	0,49	100	Avg comp. Time
	# 7	0,0017	0,0000	0,0000	0,0000	0,0001	0,0001	0,9721	0,2525	1905	0,31	1503	1395
	# 8	0,0019	0,0000	0,0007	0,0000	0,0000	0,0000	0,9718	0,2529	944	0,35	76	
	# 9	0,0000	0,0005	0,0012	0,0013	0,0001	0,0033	0,7725	0,2525	2499	0,46	200	
	# 10	0,0003	0,0000	0,0000	0,0064	0,0000	0,0000	0,0012	0,2563	811	0,90	64	

Table 49 - Low gain fault detection with Differential Evolution

Obj.		0	0	0	0	0	0	0,5	***<u>0,25</u>***	Time	% Error	Generations	
DE_009 Gain Low fault	# 1	0,0033	0,0073	0,0020	0,0117	0,0003	0,0003	0,5064	0,2566	456	1,57	29	Avg % error
	# 2	0,0030	0,0060	0,0012	0,0085	0,0011	0,0028	0,1819	0,2609	372	1,57	25	2,52
	# 3	0,0005	0,0017	0,0076	0,0082	0,0017	0,0118	0,3931	0,2511	358	1,65	25	
	# 4	0,0024	0,0024	0,0014	0,0141	0,0119	0,0401	0,5881	0,2455	381	4,47	26	
	# 5	0,0088	0,0110	0,0054	0,0101	0,0035	0,0093	0,3196	0,2417	309	2,24	21	
	# 6	0,0006	0,0049	0,0014	0,0077	0,0071	0,0136	0,8270	0,2446	337	1,92	24	Avg comp. Time
	# 7	0,0029	0,0111	0,0135	0,0021	0,0096	0,0326	0,5224	0,2447	302	3,87	21	355
	# 8	0,0092	0,0028	0,0129	0,0030	0,0075	0,0002	0,2398	0,2697	317	2,67	22	
	# 9	0,0018	0,0088	0,0058	0,0148	0,0171	0,0244	0,5640	0,2368	342	3,73	24	
	# 10	0,0039	0,0058	0,0076	0,0094	0,0039	0,0041	0,5848	0,2481	371	1,52	26	

Table 50 - Low gain fault detection with Grey Wolf Optimization

Obj.		0	0	0	0	0	0	0,5	***0,25***	Time	% Error	Generations	
GWO_009 Gain Low fault	# 1	0,0000	0,0000	0,0008	0,0011	0,0000	0,0000	0,3217	0,2475	367	0,29	22	Avg % error
	# 2	0,0000	0,0002	0,0029	0,0000	0,0000	0,0000	0,0002	0,2555	310	0,62	21	0,34
	# 3	0,0000	0,0000	0,0049	0,0001	0,0000	0,0001	0,1831	0,2501	301	0,49	21	
	# 4	0,0001	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,2526	301	0,26	21	
	# 5	0,0000	0,0000	0,0000	0,0040	0,0000	0,0000	0,0000	0,2521	302	0,45	21	
	# 6	0,0000	0,0000	0,0024	0,0000	0,0005	0,0000	0,0009	0,2526	302	0,36	21	Avg comp. Time
	# 7	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,2502	304	0,02	21	312
	# 8	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,2551	301	0,51	21	
	# 9	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,2529	300	0,29	21	
	# 10	0,0000	0,0000	0,0001	0,0000	0,0000	0,0000	0,0000	0,2487	330	0,13	23	

Table 51 - High gain fault detection with Genetic Algorithm

Obj.		0	0	0	0	0	0	0,5	***0,75***	Time	% Error	Generations	
GA_010 Gain High fault	# 1	0,0002	0,0008	0,0010	0,0010	0,0002	0,0055	0,1712	0,7554	2633	0,81	200	Avg % error
	# 2	0,0001	0,0009	0,0013	0,0003	0,0002	0,0053	0,0625	0,7496	2521	0,60	200	0,95
	# 3	0,0011	0,0018	0,0004	0,0010	0,0001	0,0098	0,0133	0,7496	2525	1,11	200	
	# 4	0,0006	0,0002	0,0011	0,0005	0,0004	0,0116	0,0980	0,7476	2526	1,28	200	
	# 5	0,0001	0,0001	0,0001	0,0013	0,0017	0,0016	0,4625	0,7413	2536	0,91	199	
	# 6	0,0004	0,0001	0,0012	0,0010	0,0038	0,0017	0,4765	0,7486	2531	0,47	200	Avg comp. Time
	# 7	0,0006	0,0006	0,0003	0,0001	0,0013	0,0061	0,9995	0,7492	2535	0,71	200	2527
	# 8	0,0016	0,0010	0,0002	0,0000	0,0016	0,0072	0,1098	0,7653	2377	1,73	187	
	# 9	0,0001	0,0000	0,0004	0,0011	0,0008	0,0058	0,0581	0,7588	2550	1,10	200	
	# 10	0,0008	0,0008	0,0018	0,0001	0,0005	0,0068	0,0732	0,7509	2536	0,77	200	

Table 52 - High gain fault detection with Particle Swarm Optimization

Obj.		0	0	0	0	0	0	0,5	***0,75***	Time	% Error	Generations	
PSO_010 Gain High fault	# 1	0,0000	0,0000	0,0000	0,0006	0,0000	0,0000	0,0779	0,7496	602	0,07	47	Avg % error
	# 2	0,0000	0,0000	0,0000	0,0004	0,0000	0,0000	0,6434	0,7486	939	0,14	74	0,21
	# 3	0,0000	0,0000	0,0001	0,0000	0,0000	0,0033	0,0019	0,7513	1190	0,39	89	
	# 4	0,0000	0,0000	0,0000	0,0003	0,0001	0,0000	0,9813	0,7487	1041	0,14	82	
	# 5	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	1,0000	0,7490	566	0,10	44	
	# 6	0,0000	0,0000	0,0000	0,0004	0,0000	0,0000	0,6674	0,7480	1423	0,20	112	Avg comp. Time
	# 7	0,0000	0,0000	0,0000	0,0000	0,0000	0,0012	0,4422	0,7491	1020	0,15	80	1078
	# 8	0,0000	0,0000	0,0000	0,0000	0,0000	0,0021	0,0017	0,7498	1373	0,23	108	
	# 9	0,0000	0,0000	0,0000	0,0000	0,0000	0,0033	1,0000	0,7511	1656	0,39	130	
	# 10	0,0000	0,0000	0,0000	0,0000	0,0000	0,0021	0,0030	0,7498	974	0,23	76	

Table 53 - High gain fault detection with Differential Evolution

Obj.		0	0	0	0	0	0	0,5	***0,75***	Time	% Error	Generations	
DE_010 Gain High fault	# 1	0,0066	0,0044	0,0036	0,0012	0,0151	0,0194	0,3940	0,7793	436	3,93	31	Avg % error
	# 2	0,0010	0,0017	0,0066	0,0027	0,0065	0,0101	0,9850	0,7410	305	1,74	21	3,31
	# 3	0,0034	0,0002	0,0026	0,0005	0,0124	0,0153	0,5508	0,7378	316	2,36	24	
	# 4	0,0013	0,0011	0,0014	0,0027	0,0021	0,0021	0,1979	0,7726	352	2,31	27	
	# 5	0,0067	0,0008	0,0004	0,0004	0,0087	0,0084	0,3909	0,7306	310	2,39	24	
	# 6	0,0001	0,0092	0,0023	0,0073	0,0205	0,0013	0,3177	0,7724	306	3,26	23	Avg comp. Time
	# 7	0,0046	0,0017	0,0067	0,0014	0,0010	0,0128	0,7895	0,8034	354	5,57	27	332
	# 8	0,0054	0,0047	0,0027	0,0116	0,0100	0,0058	0,9330	0,8082	270	6,10	21	
	# 9	0,0047	0,0035	0,0051	0,0142	0,0035	0,0139	0,0301	0,7268	342	3,23	27	
	# 10	0,0076	0,0029	0,0052	0,0049	0,0062	0,0149	0,9758	0,7564	329	2,17	27	

Table 54 - High gain fault detection with Grey Wolf Optimization

Obj.		0	0	0	0	0	0	0,5	***0,75***	Time	% Error	Generations	
GWO_010 Gain High fault	# 1	0,0104	0,0090	0,0000	0,0079	0,0015	0,0196	0,0000	0,7436	502	2,79	32	Avg % error
	# 2	0,0000	0,0011	0,0000	0,0000	0,0000	0,0021	0,0706	0,7484	301	0,30	21	0,63
	# 3	0,0000	0,0000	0,0000	0,0000	0,0049	0,0000	0,0000	0,7537	293	0,62	21	
	# 4	0,0000	0,0000	0,0015	0,0000	0,0000	0,0066	0,0015	0,7498	307	0,75	22	
	# 5	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,7486	291	0,14	21	
	# 6	0,0000	0,0000	0,0000	0,0002	0,0000	0,0000	0,0000	0,7506	304	0,07	22	Avg comp. Time
	# 7	0,0000	0,0000	0,0000	0,0000	0,0000	0,0014	0,0000	0,7489	291	0,20	21	316
	# 8	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,7490	291	0,10	21	
	# 9	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,7388	289	1,12	21	
	# 10	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,7479	290	0,21	21	

From previous tables it's possible to see how gain fault is the most detectable failure between those already studied: only once (Differential Evolution for high fault) the overall precision exceeds the 3% of average error. The most promising result comes from Particle Swarm Optimization for high fault, in which the percentage error is about 0.2%. Concerning the computational time, both low and high fault detections are comparable: this fact is due to the start point of optimization, because in nominal conditions gain is set to 0.5. The quickest algorithm both in low and high fault conditions is Grey Wolf Optimization.

Table 55 - Summary of gain fault average values

	Type	Avg Comp. Time (s)	% error
Gain	Low Fault	1138	1,31
	High Fault	1063	1,27

7.2.6. Multiple fault detection

To test the reliability of the algorithms treated in a real situation, a multiple fault optimization has been carried out: indeed, a situation in which the electromechanical actuator is slightly faulty is met easily in usual working behaviour of secondary flight control. This has been possible thanks to the Matlab code developed, which allows to implement a random fault or to introduce by yourself the fault parameters. The optimizations have been carried out in a slightly different way compared to single fault detections. Initially, the stopping criterium which force the algorithm to do more than 20 iterations has been eliminated, in order to test the convergence speed of each code (data summarised in table 56): the unique constraint is to achieve an objective function's value less or equal than 10^{-3}.

Table 56 - Random fault detection with all algorithms studied to investigate the convergence speed

Obj.		0,0068	0,0000	9,19E-03	0,0013	8,60E-04	0,0111	0,8009	0,3804	Time	% Error	Generations	
	# 1	0,0000	0,0000	0,0234	0,0163	0,0304	0,0219	0,5311	0,3518	324	4,816	20	Avg % error
	# 2	0,0000	0,0258	0,0406	0,0000	0,0000	0,0097	0,4491	0,3588	268	4,667	18	5,45
	# 3	0,0077	0,0160	0,0048	0,0199	0,0226	0,0560	0,4328	0,3723	546	6,005	40	
	# 4	0,0000	0,0000	0,0000	0,0105	0,0000	0,0000	0,0000	0,4148	108	3,899	7	
GA Multiple	# 5	0,0039	0,0251	0,0071	0,0000	0,0029	0,0373	0,0313	0,3537	862	5,364	62	
	# 6	0,0140	0,0057	0,0145	0,0423	0,0000	0,0331	0,5437	0,3885	743	4,915	54	Avg comp. Time (s)
	# 7	0,0051	0,0122	0,0047	0,0029	0,0398	0,0507	0,3982	0,3826	852	6,066	59	551
	# 8	0,0028	0,0108	0,0326	0,0137	0,0000	0,0671	0,5262	0,3926	938	6,678	67	
	# 9	0,0000	0,0241	0,0217	0,0039	0,0000	0,0039	0,8268	0,3296	294	5,846	20	
	# 10	0,0043	0,0063	0,0337	0,0417	0,0000	0,0437	0,5425	0,3586	574	6,285	40	
	# 1	0,0000	0,0000	0,0000	0,0000	0,0042	0,0000	0,0000	0,4038	116	2,855	5	Avg % error
	# 2	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	1,0000	0,3454	156	3,849	10	3,43
	# 3	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,1339	0,3638	86	2,301	5	
	# 4	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,9018	0,3537	80	3,110	5	
PSO Multiple	# 5	0,0000	0,0057	0,0000	0,0352	0,0000	0,0000	0,9899	0,3519	145	4,740	9	
	# 6	0,0000	0,0000	0,0000	0,0000	0,0106	0,0335	0,9659	0,3476	72	4,285	4	Avg comp. Time (s)
	# 7	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,2357	0,4269	54	4,920	3	97
	# 8	0,0180	0,0000	0,0000	0,0000	0,0000	0,0000	1,0000	0,3538	67	3,227	4	
	# 9	0,0000	0,0000	0,0167	0,0000	0,0000	0,0000	1,0000	0,3619	125	2,389	8	
	# 10	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,7979	0,3601	68	2,584	4	
	# 1	0,0035	0,0057	0,0172	0,0012	0,0027	0,0075	0,1683	0,3790	383	1,215	24	Avg % error
	# 2	0,0012	0,0176	0,0101	0,0106	0,0201	0,0328	0,5752	0,3970	315	4,002	22	3,54
	# 3	0,0031	0,0043	0,0057	0,0373	0,0001	0,0267	0,6549	0,3835	294	4,012	22	
	# 4	0,0056	0,0177	0,0031	0,0254	0,0037	0,0009	0,1569	0,3645	369	3,597	27	
DE Multiple	# 5	0,0116	0,0015	0,0020	0,0061	0,0049	0,0316	0,2453	0,4136	219	4,419	15	
	# 6	0,0054	0,0084	0,0036	0,0072	0,0097	0,0174	0,1430	0,3697	285	2,236	21	Avg comp. Time (s)
	# 7	0,0029	0,0051	0,0025	0,0073	0,0152	0,0228	0,2295	0,4079	251	3,734	18	305
	# 8	0,0121	0,0038	0,0106	0,0368	0,0034	0,0464	0,5970	0,3712	282	5,228	21	
	# 9	0,0075	0,0009	0,0201	0,0210	0,0104	0,0420	0,5062	0,3773	285	4,151	20	
	# 10	0,0003	0,0061	0,0001	0,0202	0,0088	0,0012	0,3114	0,3693	371	2,837	28	

	# 1	0,0000	0,0083	0,0000	0,0000	0,0069	0,0059	0,0000	0,3518	86	3,324	4	Avg % error
	# 2	0,0000	0,0000	0,0000	0,0000	0,0000	0,0089	0,0000	0,3738	189	1,520	14	
	# 3	0,0000	0,0000	0,0000	0,0000	0,0000	0,0035	0,0000	0,3613	82	2,372	6	3,49
	# 4	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,3315	118	5,142	9	
GWO Multiple	# 5	0,0000	0,0000	0,0004	0,0000	0,0008	0,0002	0,0001	0,3350	124	4,795	9	
	# 6	0,0004	0,0000	0,0000	0,0000	0,0000	0,0025	0,8164	0,3803	181	1,421	11	Avg comp. Time (s)
	# 7	0,0000	0,0000	0,0000	0,0000	0,0335	0,0000	0,4526	0,3318	135	6,068	10	
	# 8	0,0005	0,0000	0,0000	0,0000	0,0000	0,0004	0,0205	0,3273	277	5,532	18	134
	# 9	0,0035	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,4017	43	2,596	3	
	# 10	0,0000	0,0000	0,0000	0,0000	0,0018	0,0000	0,0000	0,3658	100	2,166	5	

In the second table below, the results for the four different algorithms are summarised, taking care only for the precision of the data and neglecting the computational time. To do that, the precision constraint has been eliminated and the only one remained is the stall condition criterium: if the best fault vector is the same for ten consecutive iterations, the algorithm must stop the evaluation.

Table 57 - Random fault detection with all algorithms studied to investigate the overall precision

Obj.		***0,0068***	***0,0000***	***9,19E-03***	***0,0013***	***8,60E-04***	***0,0111***	***0,8009***	***0,38038***	Time	% Error	Generations	
	# 1	0,0055	0,0016	0,0083	0,0023	0,0013	0,0000	0,9999	0,3819	2716	1,145	200	Avg % error
	# 2	0,0051	0,0010	0,0093	0,0079	0,0041	0,0125	0,4984	0,3797	2106	0,866	158	
	# 3	0,0048	0,0042	0,0113	0,0093	0,0019	0,0191	0,6194	0,3757	2663	1,380	200	1,07
	# 4	0,0073	0,0018	0,0084	0,0057	0,0019	0,0051	0,3723	0,3782	2687	0,840	200	
GA Multiple	# 5	0,0084	0,0006	0,0113	0,0015	0,0009	0,0187	0,5266	0,3895	2665	1,319	200	
	# 6	0,0051	0,0022	0,0040	0,0061	0,0065	0,0012	0,9920	0,3726	2671	1,574	200	Avg comp. Time
	# 7	0,0084	0,0007	0,0081	0,0038	0,0009	0,0117	0,5038	0,3825	2688	0,524	200	
	# 8	0,0072	0,0016	0,0039	0,0049	0,0033	0,0100	0,5457	0,3841	2663	0,845	200	2637
	# 9	0,0049	0,0009	0,0100	0,0008	0,0092	0,0139	0,3982	0,3847	2739	1,155	200	
	# 10	0,0065	0,0002	0,0136	0,0006	0,0025	0,0156	0,4796	0,3869	2775	1,050	200	
	# 1	0,0078	0,0010	0,0077	0,0003	0,0007	0,0048	0,4782	0,3808	2358	0,690	174	Avg % error
	# 2	0,0090	0,0001	0,0000	0,0045	0,0004	0,0008	0,3616	0,3842	2297	1,485	157	
	# 3	0,0066	0,0021	0,0077	0,0023	0,0000	0,0008	0,4415	0,3769	1696	1,122	127	1,06
	# 4	0,0093	0,0001	0,0049	0,0002	0,0000	0,0020	0,0728	0,3818	2761	1,068	200	
PSO Multiple	# 5	0,0074	0,0015	0,0104	0,0038	0,0000	0,0076	0,4502	0,3784	2021	0,583	152	
	# 6	0,0044	0,0016	0,0115	0,0007	0,0021	0,0075	0,3978	0,3762	1400	0,738	105	Avg comp. Time
	# 7	0,0000	0,0000	0,0140	0,0061	0,0000	0,0000	0,3373	0,3687	2664	1,874	200	
	# 8	0,0048	0,0000	0,0115	0,0028	0,0025	0,0104	0,4778	0,3773	2659	0,594	200	2292
	# 9	0,0093	0,0000	0,0045	0,0000	0,0000	0,0003	0,2928	0,3821	2653	1,218	200	
	# 10	0,0067	0,0004	0,0045	0,0054	0,0002	0,0000	0,9530	0,3796	2409	1,274	181	
	# 1	0,0079	0,0004	0,0078	0,0006	0,0000	0,0057	0,4455	0,3823	3022	0,640	200	Avg % error
	# 2	0,0072	0,0004	0,0121	0,0008	0,0014	0,0097	0,4660	0,3814	2679	0,475	200	
	# 3	0,0071	0,0003	0,0084	0,0001	0,0015	0,0075	0,4514	0,3812	2656	0,481	200	0,58
	# 4	0,0077	0,0004	0,0078	0,0000	0,0000	0,0053	0,4400	0,3823	2832	0,675	200	
DE Multiple	# 5	0,0076	0,0005	0,0057	0,0017	0,0008	0,0083	0,4577	0,3832	2631	0,612	200	
	# 6	0,0072	0,0002	0,0092	0,0009	0,0000	0,0068	0,4932	0,3810	2653	0,490	200	Avg comp. Time
	# 7	0,0079	0,0000	0,0082	0,0004	0,0007	0,0073	0,4585	0,3822	2684	0,515	200	
	# 8	0,0077	0,0002	0,0072	0,0000	0,0003	0,0053	0,4433	0,3822	3741	0,686	200	2840
	# 9	0,0073	0,0002	0,0097	0,0000	0,0001	0,0062	0,4530	0,3800	2824	0,558	200	
	# 10	0,0078	0,0006	0,0082	0,0001	0,0000	0,0044	0,4690	0,3807	2678	0,716	200	

	# 1	0,0047	0,0000	0,0147	0,0000	0,0000	0,0000	0,0000	0,3772	3077	1,302	200	Avg % error
	# 2	0,0087	0,0000	0,0038	0,0000	0,0000	0,0000	0,0000	0,3839	2856	1,305	200	
	# 3	0,0075	0,0000	0,0061	0,0000	0,0000	0,0000	0,0000	0,3834	2825	1,200	200	1,21
	# 4	0,0074	0,0000	0,0054	0,0000	0,0000	0,0000	0,0000	0,3838	2879	1,231	200	
GWO Multiple	# 5	0,0073	0,0000	0,0047	0,0000	0,0000	0,0000	0,0000	0,3813	2677	1,209	200	
	# 6	0,0065	0,0000	0,0068	0,0000	0,0000	0,0000	0,0000	0,3810	3043	1,147	200	Avg comp. Time
	# 7	0,0073	0,0000	0,0074	0,0000	0,0000	0,0000	0,0000	0,3803	2678	1,134	200	
	# 8	0,0081	0,0000	0,0056	0,0000	0,0000	0,0000	0,0000	0,3850	2988	1,269	200	2898
	# 9	0,0069	0,0000	0,0056	0,0000	0,0000	0,0000	0,0000	0,3800	2930	1,175	200	
	# 10	0,0078	0,0000	0,0094	0,0000	0,0002	0,0000	0,0134	0,3798	3022	1,121	200	

7.3. Comparison of the results

In this paragraph all the results coming from the optimizations reported in the previous pages are summarized and commented. Initially, speaking about the results for the single fault detection, the graph depicted in figure 7.1 and summarizing table 58 could be useful to better understand the error trend. The same types of algorithm are pictured with same colours: low fault is darker and high fault is lighter.

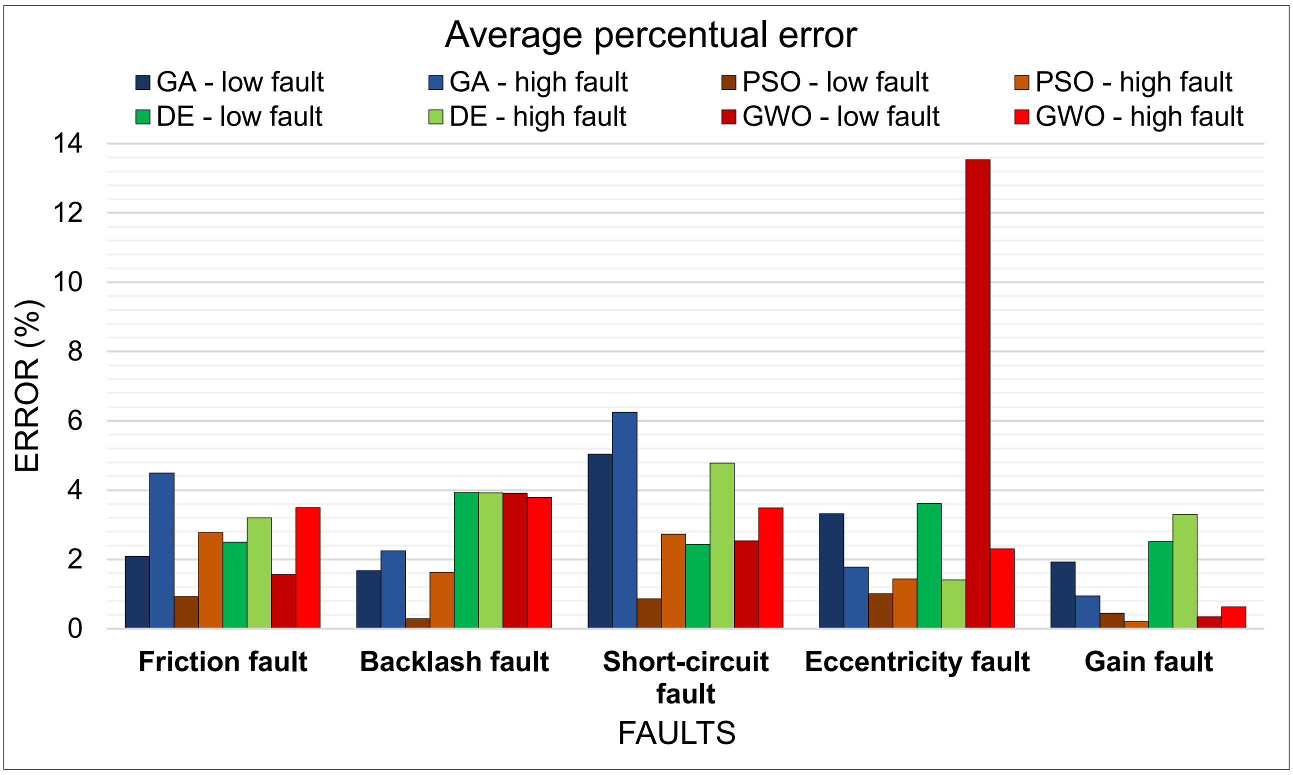

Figure 7. 1 - Average percentual error for all the algorithms analysed

Table 58 - Average percentual error summarized for all faults analysed

	Friction	Backlash	Short Circuit	Eccentricity	Gain	Total	
	Avg error (%)	Avg error (%)	Avg error (%)	Avg error (%)	Avg error (%)	Avg error (%)	σ
GA - low fault	2,093	1,678	5,042	3,323	1,928	2,813	1,398
GA - high fault	4,491	2,247	6,253	1,778	0,950	3,144	2,177
PSO - low fault	0,930	0,294	0,863	1,013	0,452	0,711	0,318
PSO - high fault	2,773	1,629	2,734	1,465	0,205	1,761	1,060
DE - low fault	2,499	3,930	2,438	3,618	2,522	3,001	0,714
DE - high fault	3,199	3,917	4,778	1,411	3,305	3,322	1,239
GWO - low fault	1,562	3,915	2,532	13,537	0,342	4,378	5,285
GWO - high fault	3,494	3,797	3,486	2,301	0,631	2,742	1,312
Average	2,630	2,676	3,516	3,556	1,292		

As it clearly reported above, PSO algorithms offer the best precision in terms of percentual error, in particular for the detection of low faults. This property is especially shown in backlash fault and gain fault, where the percentual error is less than 0.5%. The worst precision is achieved by the Grey Wolf Optimization, with a total average percentual error of about 3.5%. This bad behaviour is caused by the nature of this algorithm: it encourages the exploration of the search agents instead exploitation: in this way the precision is badly affected.

Concerning the stability of the algorithm, the small standard deviation suggests that Particle Swarm Optimization and Differential Evolution provide percentual error very similar throughout different fault implemented, in particular for low fault detection.

Most difficult faults to detect are short circuit and eccentricity, with an average percentual error of about 3.5%: this fact is caused by the relative complexity and the consequences of these failures, which affect multiple components in both reference and monitor model. The strong differences between the description of these two dysfunctions could be part of the detection problem: a possible development of this work is the deeper study of their implementation in the monitor model. As already said, gain fault is the most detectable fault with an average percentual error of 1.3%, due to its simplicity of description.

As already done for the average error, also the computational time is investigated. In figure 7.2 the histogram of time trends is depicted; in subsequent table the main values are listed.

Grey Wolf Optimization and Differential Evolution are the computationally fastest algorithms, especially to carry out the optimization of the low faults. This is clear from the fact that these two codes have been written by hand and as stopping criteria has an objective function's value of 10^{-3}. This value has been set after the observation of the behaviour of the GA and PSO, which usually arrives at a precision of 10^{-4}. Calibrating this number taking care both low and high fault detection allow a greater convergence speed for these two solutions. The high fault convergence speed is comparable between PSO, DE and GWO; Genetic Algorithm provides a terrible performance, with an average computational time of about 40 minutes.

However, GA applied to a high fault is a very stable method, because it shows a standard deviation of only 39 s.

The time required from each algorithm to reach the precision wanted is pretty comparable, with a minimum obtained by the gain fault and a maximum by the eccentricity.

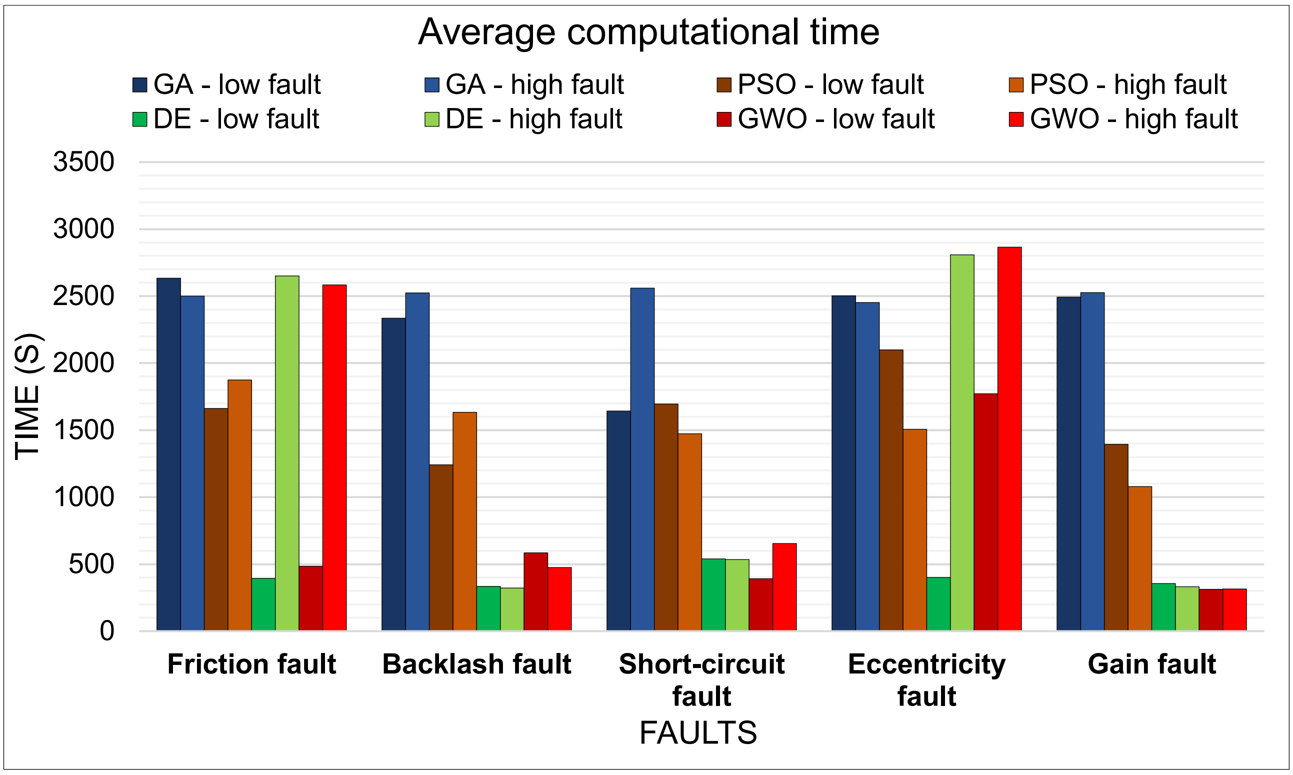

Figure 7. 2 - Average computational time for all the algorithms analysed

Table 59 - Average computational time summarized for all faults analysed

	Friction	Backlash	Short Circuit	Eccentricity	Gain	**Total**	
	Avg comp. Time (s)	Avg comp. Time (s)	Avg comp. Time (s)	Avg comp. Time (s)	Avg comp. Time (s)	Avg comp. Time (s)	σ
GA - low fault	2.633	2.336	1.644	2.502	2.492	2.322	393
GA - high fault	2.501	2.524	2.559	2.453	2.527	2.513	39
PSO - low fault	1.662	1.242	2.152	2.099	1.395	1.710	408
PSO - high fault	1.875	1.634	1.474	1.506	1.078	1.513	290
DE - low fault	394	334	540	401	355	405	81
DE - high fault	2.649	323	534	2.809	332	1.329	1.282
GWO - low fault	485	585	392	1.771	312	709	602
GWO - high fault	2.585	476	655	2.866	316	1.380	1.238
Avg	1.848	1.182	1.244	2.051	1.101		

To effectively understand the performance of all optimization algorithm applied to a single fault detection, a suitable parameter has been introduced, called *reliability coefficient*. The average error in table 58 and the average computational time in table 59 need to be the lowest possible: the reliability coefficient has to be a percentage which indicates the fitness power of that algorithm for each fault. The relation thought is:

$$RC_i(\%) = 100 \cdot \left(1 - \frac{t_i \cdot err_i(\%)}{\sum_{i=1}^{4} t_i \cdot err_i(\%)}\right) \tag{7.1}$$

where:

- RC_i is the reliability coefficient (expressed in %) of the i-th algorithm;
- t_i is the average computational time of the i-th algorithm;
- $err_i(\%)$ is the average percentual error of the i-th algorithm.

The division for the sum of the multiplied average values has been carried out in order to have as output a non-dimensional value; subtracting the resulting value to 1 allows to overturn the problem, in order to have a bigger reliability coefficient if the suitability is high. The next multiplication for 100 transforms it in a percentual value. In this way, choosing the highest reliability coefficient means choosing the best algorithm for that problem.

For every fault, the reliability coefficient has been evaluated, in order to find which algorithm is better for a determined fault. The results are summarised in table below.

Table 60 - Reliability coefficient for single fault detection

	Friction fault		Backlash fault		Short-circuit fault		Eccentricity fault		Gain fault		**Total**	
	time*err	RC(%)	time*err	RC(%)	time*err	RC(%)	time*err	RC(%)	time*err	RC(%)	time*err	RC(%)
GA	8.451	57,62	4.769	49,71	11.867	36,34	6.319	79,58	3.611	30,16	7.199	54,00
PSO	3.275	83,58	1.383	85,42	3.260	82,51	2.233	92,78	407	92,14	1.992	87,27
DE	4.335	78,26	1.287	86,43	1.937	89,61	4.035	86,96	1.000	80,66	2.741	82,49
GWO	3.880	80,54	2.045	78,44	1.576	91,55	18.359	40,67	153	97,05	3.717	76,25
sum	19941		9484		18641		30947		5170		15649	

The green values represent the best solution for the detection of that fault. Particle Swarm results are for twice (friction and eccentricity) the most reliable data, such as Grey Wolf Optimization, which is the best for short-circuit and proportional gain fault; Differential Evolution is the most suitable algorithm to detect the backlash. Only in one case (eccentricity fault) Genetic Algorithm resulted to be appreciable, with a reliability coefficient pretty equal to 80%. Concerning the overall values, the most reliable algorithm to detect single fault implementation is Particle Swarm Optimization, with a RC equal to 87%; however also Differential Evolution provides a great optimization performance, with a RC=82.5%.

Concerning the multiple fault detection, from table 56 reported in the previous paragraph, it’s possible to understand the power of the swarm-based algorithms: indeed, both Particle Swarm Optimization and Grey Wolf Optimization provide the fastest results. In particular PSO is the fastest and the most precise between the algorithms tested, because has also a 3.4% of percentual error in only 97 s of average computational time. Concerning the number of iterations, in ten different tries PSO has an average value equal to 5.7 iterations, GWO of about 9: that means that the former manages to reach faster the target error’s value. The iterations do not have the same speed in different algorithms: evaluating the average number of iterations for the four algorithms tested, using the average computational time, it’s possible to find that the quickest iterations are those of Genetic Algorithm and Differential Evolution, with approximately 14 seconds used per each iteration; the o PSO shows the slowest iterations, with 17 second per iteration. This consideration strengthens the astounding performance of the Particle Swarm Optimization, because every iteration has an astonishing optimization power.

The precision, as already said, is not a study object, because it has been set previously to a determined value, equal for all the optimizations: it’s only possible to comment that every algorithm is around the 3.5% of error except for Genetic Algorithm, the less reliable one, which is around the 5.5%.

In the second table the situation is very different: focusing on the overall precision of the algorithm and neglecting the computational time, it has been possible to investigate which algorithm allows to obtain the most reliable and data. This simulation is different from the single fault isolation, because algorithms do not have stopping criteria concerning the objective value or the maximum computational time, but only the maximum number of stall iterations, set at 10.

From table 57 it’s possible to see how Differential Evolution provides the best results in term of precision, with only the 0.58% of average error. It’s important to underline that this algorithm detects brilliantly also the friction fault, implemented in the objective function with a very small value, with a percentage error referred to this fault of only 0.07%. The other algorithms are around the 1% of average error, meaning that in any case they represent a reliable alternative to the DE.

Another important characteristic is the stability of the algorithms: the standard deviation provides the information referred to the nominal displacement of output from the average error value. During the usual 10 optimizations, Grey Wolf Optimization appears to be the most stable, with a σ equal to 0.067; on the other hand, Particle Swarm shows a particularly big dispersion of the values, with a σ=0.42.

As already done for single faults, also for the multiple fault detection the reliability coefficient is evaluated. In this case, it assumes the importance of mixing together tables 56 and 57, in order to find a perfect equilibrium between error value and computational time.

In table 61 RC is evaluated for all the algorithms.

Table 61 – Reliability coefficient for multiple fault detection

	Average Comp. Time (s)	Average % error	Time*error	$RC_i(\%)$
Genetic Algorithm	551	1,07	589,57	42,84
Particle Swarm Optimization	97	1,06	102,82	90,03
Differential Evolution	305	0,58	176,9	82,85
Grey Wolf Optimization	134	1,21	162,14	84,28
sum(t*err)			1031,43	

The best optimization to detect a multiple fault implemented in our models is the Particle Swarm Optimization, with a brilliant 90% of reliability coefficient. DE and GWO have very similar behaviours and close to the best, so they are also efficient solutions for this type of problem. It’s important to underline also the bad behaviour of the Genetic Algorithm, which provides values with only 42% of reliability coefficient.

8. Conclusions and future perspectives

In this work the behaviour of different algorithms to a single fault and multiple fault detection in an electromechanical actuator (EMA) has been deeply studied.

Firstly, after a theoretical part in which the prognostics concepts, the functioning of BLDC motor and the structure of the models are described, all the faults are investigated concerning their effect to the dynamic response of the electromechanical actuator. All the position, speed and current trends are depicted with a single fault growth, from the nominal condition to the end of the failure domain. Before the explanation of the parameters used in optimization process, the four different algorithms studied are described and analysed.

The results chapter highlights the importance of Particle Swarm Optimization both for single fault and multiple fault detection. Indeed, it shows the best reliability coefficient – a suitable parameter described in chapter 7 – throughout the ten optimizations carried out for the multiple random fault and for three different single fault implementations (friction, eccentricity and gain). Particularly important is also the information coming from one of the two multiple optimization paths: every iteration has an astonishing optimization power, because in only 3 generations evaluated in 54 seconds, this algorithm is capable to obtain a 5% error, equal to that coming from the average values of all the 10 genetic algorithm's optimizations (for 551 s of average computational time).

Differential Evolution provides data with elevated level of reliability, close to the behaviour of the PSO in the single fault detection: this algorithm is the best for the detection of backlash fault. It has also a great stability in the provided results, both in terms of computational time and percentual error.

The results coming from Grey Wolf Optimization have an intermediate level of efficiency, because they are the best to detect short-circuit faults, but their nature to promote exploration instead of exploitation do not allow it to have a great overall reliability. In the detection of multiple faults, it behaves slightly better, providing results comparable to those coming from Differential Evolution, but in general very similar to PSO. It could be a great choice if the will is to study the problem with new nature-inspired approaches.

In conclusion, it’s important to underline the bad behaviour of the Genetic Algorithm, which do not have satisfactory results both for single or multiple fault detection. It is badly affected by the complexity of the problem and the presence of multi-objective target.

This work surely makes room for further developments.

Initially, it’s possible to improve the reliability of the data coming from reference model implementing a better model for the mechanical transmission of the motion. For example, a subsystem referred to the mechanical behaviour of the nut-screw device could be studied and subsequently introduced. In this way, data take care also of the friction and little malfunctions of the transmission. Another possible development is the implementation of the temperature effects over the overall working behaviour of the BLDC Motor, in order to not neglect the viscous effects of the temperature.

In a second time, surely different optimization algorithms could be introduced and studied, in order to find which is the best compromise to study our problem: indeed, there are a lot of fast-growing new bio-inspired algorithms which could achieve new optima results. These algorithms could be also tested on a real situation, maybe exploiting the data coming from real sensor located on a test benchmark.

Changing the development environment could be useful to accelerate the simulation. Instead of Matlab, lower level languages can be used as *C, C#, C*++ or *Fortran* and maybe this simulation will be tested also on a flight control computer, in order to approaching always more the implementation in a real aerospace purpose.

Appendix A

Differential Evolution Matlab code (Copyright© Stefano Re – Politecnico di Torino 2018):

```
%% Function to be minimized

D=8; %number of variables

fobj=@(k) FitnessFunction(k);
% objf=inline('4*x1^2-2.1*x1^4+(x1^6)/3+x1*x2-4*x2^2+4*x2^4','x1','x2');
% objf=vectorize(objf);

%% Initialization of DE parameters

% N=50; %population size (total function evaluations will be itmax*N, must
be>=5)
% itmax=200;
% F=0.5; CR=0.95; %mutation and crossover ratio

%% Problem bounds

% LB=zeros(N,D);
% UB=ones(N,D);

% a(1:N,1)=-1.9; b(1:N,1)=1.9; %bounds on variable x1
% a(1:N,2)=-1.1; b(1:N,2)=1.1; %bounds on variable x2

d=(UB-LB);
basemat=repmat(int8(linspace(1,N,N)),N,1);
basemat2=repmat(int8(linspace(1,D,D)),N,1);

%% Optimization

% numOpt=10;
xbest=zeros(numOpt,8);

for ii=1:numOpt
      %Random initialization of positions
      tic
      x=LB+d.*rand(N,D);

      %Evaluate objective for all particles
      % fx=objf(x(:,1),x(:,2));

      fx=zeros(N,1);

      for i=1:N
            fx(i)=fobj(x(i,:));
      end

      %Find best
      [fxbest,ixbest]=min(fx);
      xbest(ii,:)=x(ixbest,:);
      xbestit=repmat(2,itmax,D);
      %Iterate
      for it=1:itmax
            disp(['Iteration number ',num2str(it),' of the optimization number
',num2str(ii)])

            permat=bsxfun(@(x,y) x(randperm(y(1))),basemat',N(ones(N,1)))';

            %Generate donors by mutation
            v=repmat(xbest(ii,:),N,1)+F*(x(permat(:,1),:)-x(permat(:,2),:));

            %Perform recombination
            r=repmat(randi([1 D],N,1),1,D);
            muv = ((rand(N,D)<CR) + (basemat2==r)) ~= 0;
            mux = 1-muv;      %negation of muv

            % u(1:N,1:D)=x(1:N,1:D).*mux(1:N,1:D)+v(1:N,1:D).*muv(1:N,1:D);
            u=abs(x.*mux+v.*muv);

            %Greedy selection
            fu=zeros(N,1);

            parfor iii=1:N
                  fu(iii)=fobj(u(iii,:));
            end

            idx=fu<fx;
            fx(idx)=fu(idx); %subtitution of the lowest values of obj func in
function fx
            x(idx,:)=u(idx,:); % subtitution of the worst fitting vector with
most fitting vector


            %Find best
            [fxbest(it,1),ixbest]=min(fx);
            xbest(ii,:)=x(ixbest,:);
            xbestit(it,:)=x(ixbest,:);
%           disp(xbestit(it,:));

%           if it>2
%
%                 stopCrit=mean(abs(xbestit(it,:)-xbestit(it-1,:)));
%                 if stopCrit<1e-6
%                       it=itmax;
%                 end
%
%           end

            %% Stopping criteria

            if it>20 && fxbest(it,1)<1e-3
                  crit=fxbest(it-1,1)-fxbest(it,1);
                  if crit<1e-12
                        break
                  end
            end

      end %end loop on iterations
      %% Saving results

      save('Results_DE','xbest')
      % [xbest,fxbest]
      toc
end
```

Appendix B

Greywolf Optimization Matlab code (Copyright© Stefano Re – Politecnico di Torino 2018):

```
%% Parameters from Main script

% SearchAgents_no=50;
% Max_iter=200;
% dim=8;
% lb=zeros(1,dim);
% ub=ones(1,dim);
fobj=@FitnessFunction;

%Initialize the positions of search agents
Positions=GWO_initialization(SearchAgents_no,dim,ub,lb);

%% Main loop

% numOpt=input('How many optimizations do you like to do?');

Z=zeros(numOpt,dim);

for ii=1:numOpt %numero di ottimizzazioni

    tic

    Alpha_pos=zeros(SearchAgents_no,dim);
    Alpha_score=1e6*ones(SearchAgents_no,1); %change this to -inf for
maximization problems

    Beta_pos=zeros(SearchAgents_no,dim);
    Beta_score=1e6*ones(SearchAgents_no,1); %change this to -inf for
maximization problems

    Delta_pos=zeros(SearchAgents_no,dim);
    Delta_score=1e6*ones(SearchAgents_no,1); %change this to -inf for
maximization problems

    Alpha_score_best=1e6*ones(Max_iter,1);
    Alpha_pos_best=repmat(10,Max_iter,dim); %metto 2 così so che sono numeri
impossibili


    for l=1:Max_iter

        disp(['Iteration number ',num2str(l),' of the optimization number
',num2str(ii)])

            parfor i=1:SearchAgents_no

                % Return back the search agents that go beyond the boundaries of
the search space
                 Flag4ub=Positions(i,:)>ub;
                 Flag4lb=Positions(i,:)<lb;
                 Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag
                 4ub+lb.*Flag4lb;

                % Calculate objective function for each search agent
                 fitness(i,1)=fobj(Positions(i,:));
            end

            parfor i=1:SearchAgents_no
            % Update Alpha, Beta, and Delta
                if fitness(i,1)<Alpha_score(i)
                    Alpha_score(i)=fitness(i,1); % Update alpha
                    Alpha_pos(i,:)=Positions(i,:);

                elseif fitness(i,1)>Alpha_score(i) && fitness(i,1)<Beta_score(i)
                    Beta_score(i)=fitness(i,1); % Update beta
                    Beta_pos(i,:)=Positions(i,:);

                elseif fitness(i,1)>Alpha_score(i) && fitness(i,1)>Beta_score(i)
&& fitness(i,1)<Delta_score(i)
                    Delta_score(i)=fitness(i,1); % Update delta
                    Delta_pos(i,:)=Positions(i,:);
                end
            end

    a=2-l*((2)/Max_iter); % a decreases linearly fron 2 to 0

    % Update the Position of search agents including omegas
        for i=1:SearchAgents_no
            for j=1:dim

                r1=rand(); % r1 is a random number in [0,1]
                r2=rand(); % r2 is a random number in [0,1]

                A1=2*a*r1-a;)
                C1=2*r2;

                D_alpha=abs(C1*Alpha_pos(i,j)-Positions(i,j));
                X1=Alpha_pos(i,j)-A1*D_alpha;

                r1=rand();
                r2=rand();

                A2=2*a*r1-a;
                C2=2*r2;

                D_beta=abs(C2*Beta_pos(i,j)-Positions(i,j));
                X2=Beta_pos(i,j)-A2*D_beta;

                r1=rand();
                r2=rand();

                A3=2*a*r1-a;
                C3=2*r2;

                D_delta=abs(C3*Delta_pos(i,j)-Positions(i,j));
                X3=Delta_pos(i,j)-A3*D_delta;
```

```
                Positions(i,j)=(X1+X2+X3)/3;

            end
        end

        [Alpha_score_best(l,1),Best_index]=min(Alpha_score);
        Alpha_pos_best(l,:)=Alpha_pos(Best_index,:);

        %% Stopping criteria
%         l>20 &&
%         if Alpha_score_best(l,1)<1e-3
%             crit=Alpha_score_best(l-1,1)-Alpha_score_best(l,1);
%             if crit<1e-16
%                  break
%             end
%         end

    end

    [best_solution(ii), indexx]=min(Alpha_score_best);
    Z(ii,:)=Alpha_pos_best(indexx,:);
    save('GWO_Results','Z')

toc

end
```

Appendix C

Main optimization Matlab code (Copyright© Stefano Re – Politecnico di Torino 2018):

```
%% Introduzione

clear
close all
clc

disp('%%%%%%%%%%%%%%-------    Fault parameters optimization for EMA model    --
-----%%%%%%%%%%%%%%')
disp(' ')
disp('Script developed by Stefano Re, with Pier Carlo Berri and Matteo Dalla
Vedova (DIMEAS Polito)')
disp(' ')

%% Command choice

% User could choice the type of the command for the EMA. Commands are
% expressed in [rad]:

a=input('Choose the command: (1=step, 2=ramp, 3=sinusoidal, 4=chirp): ');

if a==1
        Com1=1;
        Com2=0; Com3=0; Com4=0; Com5=0; Com6=0;
elseif a==2
        Com2=1;
        Com1=0; Com3=0; Com4=0; Com5=0; Com6=0;
elseif a==3
        Com3=1;
        Com1=0; Com2=0; Com4=0; Com5=0; Com6=0;
elseif a==4
        Com4=1;
        Com1=0; Com2=0; Com3=0; Com5=0; Com6=0;
else
        disp('Do a valid choice!')
        clear
end


%% Optimization choice

OptChoice=input('Would you like to study a single-faulty or a multiple-faulty
BLDC motor? (1=single, 2=multiple): ');
disp(' ');

if OptChoice==1

    %% Fault choice

    disp(' ')
    disp('Choose the fault you would like to introduce:')
    disp('1= Friction fault')
    disp('2= Backlash fault ')
    disp('3= Phase A short circuit fault')
    disp('4= Phase B short circuit fault')
    disp('5= Phase C short circuit fault')
    disp('6= Eccentricity parameter fault (zita) ')
    disp('7= Eccentricity phase fault ')
    disp('8= Gain fault ')
    disp(' ')

    b=input('Your choice: ');

    if b==1
        F=input('Give a value for the friction [1 - 3] --> [NC - 3*NC]: ');
        B=1;
        Z=0;
        phi=0;
        G=1;

        if F<1 || F>3
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT')
        end
    elseif b==2
        F=1;
        B=input('Give a value for the backlash [1 - 100] --> [NC - 100*NC]: ');
        Z=0;
        phi=0;
        G=1;

        if B<1 || B>100
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT')
        end
    elseif b==3
        F=1;
        B=1;
        Na=input('Give a value for the phase A short-circuit [1 - 0] --> [0% -
100%]: ');
        Nb=1;
        Nc=1;
        Z=0;
        phi=0;
        G=1;

        if Na<0 || Na>1
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT2')
        end
    elseif b==4
        F=1;
        B=1;
        Na=1;
        Nb=input('Give a value for the phase B short-circuit [1 - 0] --> [0% -
100%]: ');
        Nc=1;
        Z=0;
        phi=0;
        G=1;

        if Nb<0 || Nb>1
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT2')
        end
    elseif b==5
        F=1;
        B=1;
        Na=1;
        Nb=1;
        Nc=input('Give a value for the phase C short-circuit [1 - 0] --> [0% -
100%]: ');
        Z=0;
        phi=0;
        G=1;

        if Nc<0 || Nc>1
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT2')
        end
    elseif b==6
        F=1;
        B=1;
        Z=input('Give a value for the Z [0 - 0.42]: ');
        phi=0;
        G=1;

        if Z<0 || Z>0.42
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT')
        end
    elseif b==7
        F=1;
        B=1;
        Z=0;
        phi=input('Give a value for the eccentricity phase [-pi - +pi]: ');
        G=1;

        if phi<-pi || phi>pi
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT')
        end
    elseif b==8
        F=1;
        B=1;
        Z=0;
        phi=0;
        G=input('Give a value for the gain [0.5 - 1.5] --> [0.5*NC - 1.5*NC]:
');
```

```
        if G<0.5 || G>1.5
            disp('Do a valid choice!')
            clear
        else
            run('EMA_Re_DAT')
        end
    else
        disp('Do a valid choice!')
        clear
    end

    % Single fault part termined

elseif OptChoice==2     %% Start multiple fault part
    InputType=input('Would you like to introduce a random multiple fault (choose
1) or write the fault parameters by your own (choose 2)?: ');

    if InputType==1         %% random multiple fault

        RandomFaultParams=rand(1,8);
        RandomFaultParams(:,1:6)=RandomFaultParams(:,1:6).^7;
        RandomFaultParams(:,8)=((RandomFaultParams(:,8)*2-1).^7+1)/2;
        disp(' ')
        disp(['The random fault vector is: ',mat2str(RandomFaultParams(:))])
        save('Random_fault', 'RandomFaultParams')
        disp(' ')


        F=2*(RandomFaultParams(1,1))+1;
        B=99*(RandomFaultParams(1,2))+1;
        Na=1-RandomFaultParams(1,3);
        Nb=1-RandomFaultParams(1,4);
        Nc=1-RandomFaultParams(1,5);
        Z=0.42*(RandomFaultParams(1,6));
        phi=RandomFaultParams(1,7);
        G=RandomFaultParams(1,8)+0.5;

        run('EMA_Re_DAT2');

    elseif InputType==2     %% User-introduced fault

        disp(' ');
        F=2*input('Introduce the friction fault: ')+1;
        B=99*input('Introduce the backlash fault: ')+1;
        Na=1-input('Introduce the phase A short-circuit fault: ');
        Nb=1-input('Introduce the phase B short-circuit fault: ');
        Nc=1-input('Introduce the phase C short-circuit fault: ');
        Z=0.42*input('Introduce the eccentricity fault: ');
        phi=input('Introduce the eccentricity phase fault: ');
        G=input('Introduce the gain fault: ')+0.5;

        run('EMA_Re_DAT2');

    else
        disp('Do a valid choice!')
        clear
    end
```

```
end

%% Simulation of the reference Model

sim('EMA_Re_EVO');

%% Optimizations

disp(' ')
disp('Choose the optimization algorithm: ')
disp('1= Genetic Algorithm')
disp('2= Particle Swarm Optimization')
disp('3= Differential Evolution')
disp('4= Greywolf Optimization')
disp(' ')
c=input('Your choice: ');
disp(' ')
d=input('Would you like to parallelize the optimization? (1=yes, 2=no)');

%Parallelization of the optimizations

if d==1
    parpool
    Parp=true(1);
else
    Parp=false(1);
end

% For every set of input, I do 10 optimizations (heuristic!=deterministic)
numOpt=input('How many optimizations do you like to do?');

X=zeros(numOpt,8);
Y=zeros(numOpt,8);
popsize=input('Specify the population size: ');
numgen=input('Specify the number of generations: ');


for i=1:numOpt

     if c==1
        LB=zeros(1,8);
        UB=ones(1,8);
        disp(' ')

        tic

opt1=optimoptions(@ga,'Display','iter','PopulationSize',popsize,'Generations',nu
mgen,'FunctionTolerance',1e-12, 'UseParallel',Parp);
%          'FitnessLimit',1e-3
        X(i,:)=ga(@FitnessFunction,8,[],[],[],[],LB,UB,[],[],opt1);
        toc

        save('Results_GA','X')
     elseif c==2
        LB=zeros(1,8);
        UB=ones(1,8);
        disp(' ')
        disp(' ')

        tic

opt2=optimoptions(@particleswarm,'Display','iter','SwarmSize',popsize,'MaxIterat
ions',200,'FunctionTolerance',1e-12,'UseParallel',Parp);
%          'ObjectiveLimit',1e-3
        Y(i,:)=particleswarm(@FitnessFunction,8,LB,UB,opt2);
        toc

        save('Results_PSO','Y')




     elseif c==3
        N=input('Specify the population size: ');
        D=8;
        LB=zeros(N,D);
        UB=ones(N,D);



        itmax=input('Specify the maximum number of iterations: ');
        disp(' ');
        disp('CHOICE OF THE D.E. PARAMETERS:');
        F=input('Mutation ratio (F): ');
        CR=input('Crossover ratio (CR): ');

        run('DE_main');     %the "save" command is already inside the script
(xbest)

        break

     elseif c==4
        SearchAgents_no=input('Specify the number of search agents: ');
        Max_iter=input('Specify the maximum number of iterations: ');
        dim=8;
        lb=zeros(1,dim);
        ub=ones(1,dim);

        run('GWO_remix');   %the "save" command is already inside the script (Z)

        break
     else
         disp('Do a valid choice!')
         break
     end

%    save('Results','X','Y')
end

disp(' ')
disp('Optimization successfully termined')
```