POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Development of innovative prognostic methods for EMAs

Applied to aerospace systems

Supervisors
Eng. Matteo D. L. Dalla Vedova
Prof. Paolo Maggiore

Candidate
Gaetano Quattrocchi

October 2019

Ringraziamenti

Vorrei in primis ringraziare l'Ing. Matteo Dalla Vedova, relatore di questa tesi, per la sua disponibilità e per i suoi consigli, fondamentali per la stesura di questo lavoro. Inoltre, un grazie va anche al prof. Paolo Maggiore per il suo supporto durante la realizzazione del presente. Un ringraziamento sentito va all'Ing. Pier Carlo Berri per la sua continua disponibilità e per il suo fondamentale supporto nella stesura e revisione del codice di calcolo.

Un grazie sentito va a tutti i miei colleghi ed amici, che hanno reso questi due anni un'esperienza memorabile. In particolare Samuele, Alessandro, Francesco e Giuseppe.

È doveroso un grande ringraziamento a Giorgia, la mia fidanzata, che mi è sempre stata vicina, anche nei momenti di difficoltà; il suo supporto è stato fondamentale.

Volevo infine ringraziare la mia famiglia sia per il sostegno economico ma principalmente per quello morale; ringrazio mia sorella Francesca per la sua pazienza e per la comprensione che ha sempre avuto. Ringrazio mio padre, mia madre e i miei nonni per il continuo incoraggiamento e per l'aver sempre creduto in me.

Infine, un ringraziamento al piccolo Attila, che è sempre riuscito a strapparmi una risata, anche nei momenti di solitudine e scoramento.

Gaetano Quattrocchi

Summary

In recent years, the adoption of Electro-Mechanical Actuators (EMA) in the aerospace sector, mainly as secondary actuation devices, is strongly increasing, in particular in the *more electric* and *all electric* design philosophies. These approaches aim at creating a single form of secondary power, in order to drive all users (systems); the only form of energy so versatile to be capable of accomplishing such task is electric energy. While using electricity, EMAs provide the natural electro-mechanical interface needed to convert secondary power, electricity, to useful mechanical work.

At this moment, the use of EMAs in large scale commercial aircrafts is relegated to secondary flight control actuation (flaps, slats, airbrakes), while the use as main flight control surfaces actuation is still limited to small UAVs. The manufacturers' choice has multiple reason: firstly, in large power applications, electro-hydraulic actuators are still lighter and more compact; in second place, EMAs are still recent technology, so there is not a complete literature on their failure modes nor an established prognostic methodology.

One approach that could led to fault detection and subsequent isolation is the use of a properly trained Neural Network, evaluating the response of the EMAs and/or the motor to a given signal and outputting an estimate of convenient values measuring a relative level of performance degradation. This is the aim of this work, to test and validate innovative methods for Electro-Mechanical Actuators (EMAs) prognosis, based on the use of Neural Networks.

In this work, the motors analyzed and modeled are 3-phases Brushless DC traprzoidal motors (BLDCs), widely used in the aerospace sector.

In order to collect the substantial amount of data needed to properly train the Network, a very detailed Simulink model has been used, modeling in detail both electrical and mechanical components; the model derives directly form Eng. M. Dalla Vedova PhD dissertation [12] . This approach has multiple benefits compared to physical testing: low cost, fast deployment, good accuracy.

There are five variables modeling all the possible faults condition analyzed: three representing a partial shortage of each phase, one modeling the static eccentricity value and the last representing the phase of such imbalance.

The Simulink model has been run thousands of times, imposing each time a different set of faults conditions; current, voltage and position have then been logged for each iteration.

Using the aforementioned physical data, a MATLAB algorithm has been used in order to properly reconstruct the Counter Electro-Motive Force (CEMF); the CEMF signals have then been suitably sampled and used as training set for various neural networks.

Performance are evaluated on a test set not previously used in training, highlighting the difference between different architectures and sampling strategies.

Contents

Ringraziamenti **iii**

Summary **iv**

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Prognostics . . . 3
- 1.3 Flight controls . . . 5
  - 1.3.1 Primary flight controls . . . 5
  - 1.3.2 Secondary flight controls . . . 8
- 1.4 Actuation systems . . . 10
  - 1.4.1 Hydromechanical . . . 11
  - 1.4.2 Electrohydraulic . . . 11
  - 1.4.3 Electrohydrostatic (EHA) . . . 12
  - 1.4.4 Electromechanical (EMA) . . . 12

2 Brushless motors **16**
- 2.1 Stator . . . 17
- 2.2 Rotor . . . 19
- 2.3 Operation principles . . . 20
- 2.4 Mechanical characteristics . . . 23
- 2.5 Motor control . . . 26
  - 2.5.1 Speed control . . . 26
  - 2.5.2 Torque control . . . 28
- 2.6 Protection systems . . . 29

3 EMA model description **31**
3.1 Model overview . . . 31
3.2 Com subsystem . . . 32
3.3 Trapezoidal EMA subsystem . . . 33
3.3.1 Control electronics . . . 34
3.3.2 Hall sensors . . . 34
3.3.3 Inverter model . . . 35
3.3.4 BLDC electromagnetic model . . . 37
3.4 Transmission dynamical model . . . 38
3.5 Longitudinal dynamics block . . . 40

4 Fault modes and modeling **42**
4.1 EMAs fault modes . . . 43
4.2 Short circuit fault . . . 46
4.2.1 Short circuit fault implementation . . . 46
4.3 Rotor eccentricity fault . . . 47
4.3.1 Rotor eccentricity fault implementation . . . 49
4.4 Noise fault . . . 50
4.4.1 Noise fault implementation . . . 52
4.5 Friction fault . . . 53
4.5.1 Friction fault implementation . . . 54

5 Data analysis **56**
5.1 Faults generation . . . 56
5.2 CEMF reconstruction . . . 57
5.3 Sampling strategy . . . 58
5.4 ANN overview . . . 61
5.4.1 Components . . . 62
Neurons . . . 63
Layers . . . 63
Connections and weights . . . 63
Propagation function . . . 64
5.4.2 Learning . . . 64
Learning rate . . . 64
Cost function . . . 64
Backpropagation . . . 65
5.4.3 Training paradigms . . . 65
Supervised learning . . . 65
Unsupervised learning . . . 65
Reinforcement learning . . . 66
5.5 ANN implementation . . . 66

5.5.1 Parameters choice 66
Network topology 67
Training function 67
Training goal 68
Performance function 68
Transfer function 68
5.6 Results 69
5.6.1 One sample per commutation 69
5.6.2 Two samples per commutation 70
5.6.3 Three samples per commutation 71
5.6.4 Three samples per commutation, two layers 72

6 Conclusions **73**

Appendix 1 MATLAB code **75**
1.1 Faults generation 75
1.2 Optimal sampling strategy 78
1.3 Neural network creation and training 80

Appendix 2 Detailed results **82**
2.1 Verification set 82
2.2 3 inputs per commutation, shallow networks 83
2.3 3 inputs per commutation, deep networks 85
2.4 2 inputs per commutation, shallow networks 86
2.5 1 input per commutation, shallow networks 88

Bibliography **89**

List of Figures

1.1 Prognostics and diagnostics [6] 2
1.2 Flight control surfaces of Boeing 727 [16] 6
1.3 Aircraft body axis convention [15] 7
1.4 BAC/Aérospace Concorde [25] 8
1.5 Typical scheme of powered flight controls [23] 9
1.6 Flaps and slats types [3] 10
1.7 Typical scheme of hydromechanical actuation [27] 11
1.8 Two stage flapper-nozzle valve [13] 12
1.9 EHA system architecture and valve schematic 13
1.10 Electromechanical actuator scheme [6] 14
1.11 Ball screw and roller screw [27] 15
2.1 BLDC Motor Cross Section [7] 17
2.2 Slotted and slotless configurations [34] 18
2.3 Star (left) and triangle (right) configurations [6] 18
2.4 Isotropic (left) and anisotropic (right) rotors [6] 20
2.5 Typical BLDC commutation scheme [23] 21
2.6 BLDC electrical scheme [27] 22
2.7 BLDC motor timing diagram [11] 23
2.8 Ideal BLDC speed-torque characteristic [23] 25
2.9 Realistic BLDC speed-torque characteristic [23] 25
2.10 Realistic BLDC speed-torque characteristic, simplified [23] . . . 26
2.11 PWM signal generation [6] 27
2.12 Speed control implementation, block diagram [6] 28
2.13 Torque control implementation, block diagram [6] 29
3.1 Simulink model overview 32
3.2 Com block overview 33
3.3 Trapezoidal EMA subsystem 33
3.4 Control electronic subsystem 34
3.5 Hall sensors subsystem 35
3.6 Inverter model in detail 36
3.7 Phase evaluation and hysteresis PWM subsystems details 36

3.8 H-bridge subsystem 37
3.9 CEMF coefficient calculation block 38
3.10 3-phase RL model block 39
3.11 Motor torque evaluation block 39
3.12 Transmission dynamical model 40
3.13 Longitudinal dynamics block 41
4.1 Air gap considering static eccentricity [6] 48
4.2 Air gap magnetic circuit approximation [6] 49
4.3 White noise (left), autocorrelation (center) and power spectrum (right) [33] 52
4.4 EM noise implementation in Simulink model 53
4.5 Borello dry friction model implementation in Simulink 55
5.1 Sample CEMF reconstruction 59
5.2 Sample CEMF reconstruction, 0ř − 180ř mechanical 59
5.3 CEMF reconstruction, 5 conditions 60
5.4 Angular coefficient in function of sampling interval 61
5.5 CEMF final sampling 62
5.6 Feedforward neural network 62
5.7 Cascadeforward neural network 63
5.8 Commonly used transfer functions 69
5.9 6 inputs neural networks performance 70
5.10 12 inputs neural networks performance 70
5.11 18 inputs neural networks performance 71
5.12 18 inputs neural networks, 2 layers performance 72

List of Tables

2.1 Permanent magnet materials characteristics 19
4.1 Mechanical and structural fault modes 44
4.2 Motor fault modes 45
4.3 Electrical and electronic fault modes 45
2.1 Verification faults set 82
2.2 Relative errors, percentage, 3-inputs per commutation shallow networks 85
2.3 Relative errors, percentage, 3-inputs per commutation deep networks 86
2.4 Relative errors, percentage, 2-inputs per commutation shallow networks 87
2.5 Relative errors, percentage, 1-input per commutation shallow networks 88

Chapter 1

Introduction

1.1 Overview

In recent years, prognostics techniques used to timely detect incipient failures in components have sparked great interest in the aeronautical world, especially in commercial and defense sectors.

But what is *prognostics*? Vachtsevanos et al. [32] give the following definition: *"Prognostics is an engineering discipline focused on predicting the time at which a system or a component will no longer perform its intended function."*

Every system and components has certain level of performance required; when this condition is no longer satisfied, the system becomes failed. Integrating prognostics in the design process has numerous advantages, including less time spent on maintenance, cost saving and increased safety of operations. This new design philosophy is called *Prognostics and Health Management (PHM)*.

An important concept correlated to prognostics is the *Remaining Useful Life (RUL)*; it represents the estimated remaining time before the element will not comply to the required specifications.

This concept is very significant when applied to safety critical components such as flight control systems: today, such systems are designed using a *safe-life* approach, where each component is replaced after a predetermined amount time, irregardless of its status. Such approach has the benefit of having ample margins of safety, since components are not used close to the theoretical maximum useful life; on the other hand there is a twofold disadvantage: no initial defects deriving from manufacture, capable of abruptly compromising the system and aircraft safety, are considered in this approach and also still working components are replaced, increasing cost.

It is then evident that a modern PHM approach used in the design process of these systmes has numerous advantages: reduction of cost thanks to decreased frequency of maintenance events and to the lowering of needed redundancies, thus making the system more appealing to customers; at the same time, an increased safety and reliability is obtained.

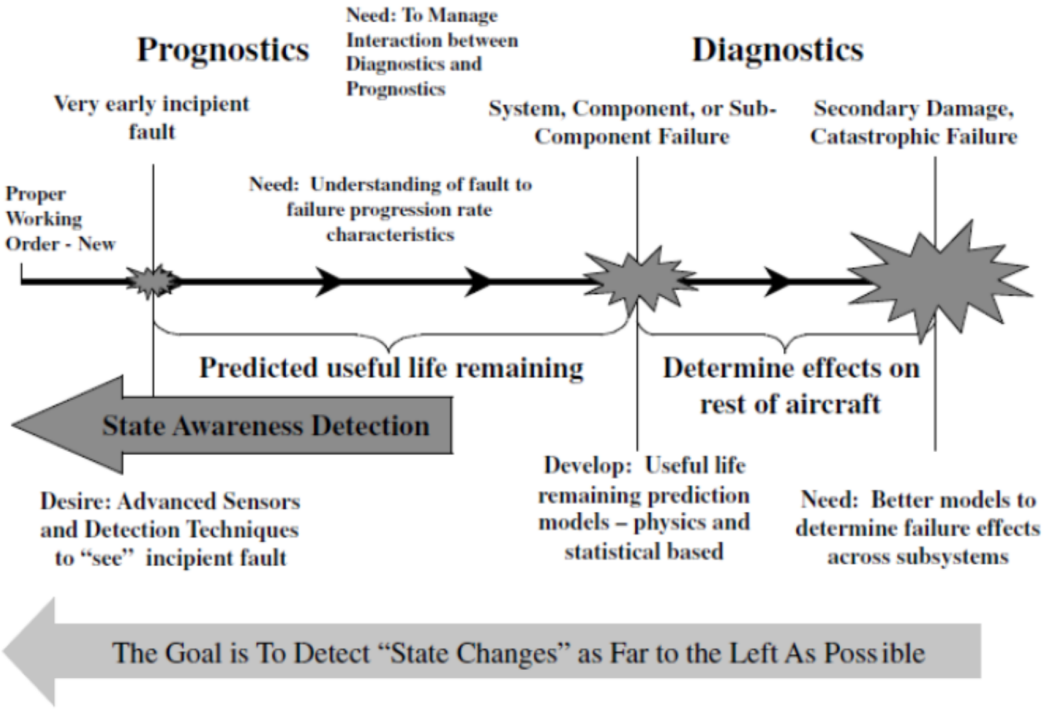


Figure 1.1: Prognostics and diagnostics [6]

The previous figure (1.1) shows the evolution of the impact of a flaw in a component: initially, when an incipient fault occurs, the effect is limited to the component and not affecting other elements in the system. As the fault starts to propagate to the related components, the margin of safety of the system rapidly decreases; if no corrective measure is taken, this can lead to secondary damage and catastrophic failure.

The time interval between fault detection and propagation is indeed the RUL, so it is paramount to timely detect any fault occurring in each component before they can propagate to other elements affecting the performance level; by doing so, an effective maintenance schedule can be planned and executed.

If the fault has already compromised system performance, then that is the field of diagnostics, whose scope is to identify and isolate the faulted component(s) while maintaining an acceptable level of performance in order to safely complete the task and mission, by virtue of hot and cold redundancies. Generally, in this

phase, faults effects are much more evident, so intervention is somewhat easier.

Many different specialties are involved in the PHM philosophy, including:

- Monitoring: in order to have an updated situation report, this is a fundamental process. To properly evaluate the condition of a system and/or component, data gathering and analysis is needed. This is accomplished by the use of sensors; these are critical elements in the monitoring process: high accuracy and precision is required in order to correctly sense any anomaly in the behavior of the monitored element; furthermore, a substantial level of reliability is needed, e.g. a greater MTBF compared to the monitored element, to avoid false detection or even the complete fault misidentification.
- Prognostics: this engineering discipline focuses on making reasonable prediction about the health level of a component, allowing maintenance events to occur before failure propagation and thus augmenting the safety margin of the system. Prognostics approaches are applicable in those cases where the failure modes are progressive and monitorable, i.e. affecting the system performance in a measurable way; on the contrary, these approaches are not useful when dealing with elements showing sudden, unpredictable failure modes. Redundancy and other means of safety augmentation are needed in these cases if such events compromise safety and reliability.
- Diagnostics: this is a process used to detect, diagnose and isolate any failure point in a system. Three main phases are comprised in the process: system monitoring, gathering and collecting data; following a successful detection of a non-routine behavior by comparison to another source (i.e. duplicated element, control algorithm, etc.), the process aims at the identification of the severity of the event (e.g. safety-critical) to take the appropriate corrective measures; finally, the failed element is deactivated or bypassed and, if present, redundant components are activated to bring back the system as close as possible to nominal status.
- Health Management: consists in saving gathered data in order to use them as comparison to highlight degradation trends; this process also aims at assessing, in case of a non time sensitive failure, a more detailed diagnosis of the fault. Furthermore, these data can be used for statistical purposes, e.g. to correct life estimates or as feedback for product improvement.

1.2 Prognostics

As briefly introduced in the previous section, prognostics is the engineering discipline of making accurate prediction regarding the remaining useful life (RUL)

of a component or system, i.e. the amount of time until the element will be able to operate within its specifications [32].

Several steps can be identified in the prognostics process:

- Data collection: the element under examination is equipped with sensors, in order to evaluate fundamentals parameters characterizing the current status. In the case of this work, since a BLDC driven EMA is analyzed, important parameters are currents, voltages, torques, angular position and velocities. Since several sensors are used to sense each variable, it is imperative to include a logic capable of detecting anomalies in the readings, caused by electrical or mechanical noise.

- Data filtering: since each sensor not only senses the value to be measured but also the inevitable noise introduced by the operating environment, a filtering stage, either analog or digital, is fundamental in order to eliminate fluctuations that can have detrimental effects on the subsequent stages.

- Threshold comparison: having now a clean signal, a threshold crossing verification is made, either using historical data or requirement imposed to the element.

- RUL prediction: using suitable methods (e.g. neural networks, algorithms, statistics), an accurate prediction of the RUL is made and future maintenance events are planned accordingly.

Several prognostics approaches can be defined, based on the fundamental characterization of the system:

- Model-based prognostics: this approach focuses on the modeling of suitable physical models in the prediction of the RUL. Modeling can either be done at the *macro* or *micro* level; in the former, a relation is obtained between the inputs and state variables, in a simplified form; this approach is useful for the analysis of whole systems; in the latter case, a system of dynamic equation relating degradation and the operating condition is used. This approach permits then the inclusion of empirical knowledge in the RUL estimation. Main disadvantages are uncertainty related to the simplifications made in the model and the generally high complexity, even at macro level, when modeling complex systems.

- Data-driven prognostics: in this case, the use of pattern recognition and machine learning is exploited to detect changes in system states [24]. Numerous techniques can be adopted, starting from autoregressive models up to Neural Networks (NN). This approach is appropriate in case of high

system complexity or when some aspect of the component behavior are not fully understood. The use of data-driven prognostics permit a wider scope for the analysis, being capable of analyzing whole systems, even tough a substantial amount of data is needed during the training phase. Two main strategies are possible: modeling cumulative damage and then evaluating a threshold or directly learning the RUL from data.

- Hybrid approaches: these approaches tries to mix the strength of the two previous approach. In general, most type of analysis are hybrid, since seldom occurs that it is either fully data-driven or completely model-based. There are two families of hybrid methods: pre-estimation fusion of models and data, used when run-to-failure data are not available [21] (i.e. when diagnostics positively identifies faults that are then corrected by maintenance), has the aim of better leveraging the RUL of the component; post-estimation fusion is generally used in uncertainty management, in order to reduce the uncertainty interval and thus increasing accuracy; the approach is based on the concept that more classifiers are better than a single one.

In this work, an hybrid, pre-estimation approach is used; CEMF data gathered using a detailed *micro*, component-level physical model are used in the training of a Neural Network, subsequently used to estimate the damage level of the same system.

In conclusion, a successful and accurate prognostics can ultimately lead to increased safety and reliability, while decreasing logistics effort and maintenance cost.

1.3 Flight controls

Flight controls are a fundamental element to control an aircraft. The scope is to properly modify the shape of some aerodynamic surfaces in order to change the flight characteristics of the aircraft itself. A distinction is made between *primary* and *secondary* flight controls.

1.3.1 Primary flight controls

Primary flight controls are used to create imbalance torques around the three body axis, namely pitch, roll and yaw, resulting in a change of the aircraft attitude. Since these commands are operated continuously, their actuation must be *instinctive*, so that the pilot can be facilitated in the actuation, and *proportional*, especially in augmented flight controls, so that the feedback can give the

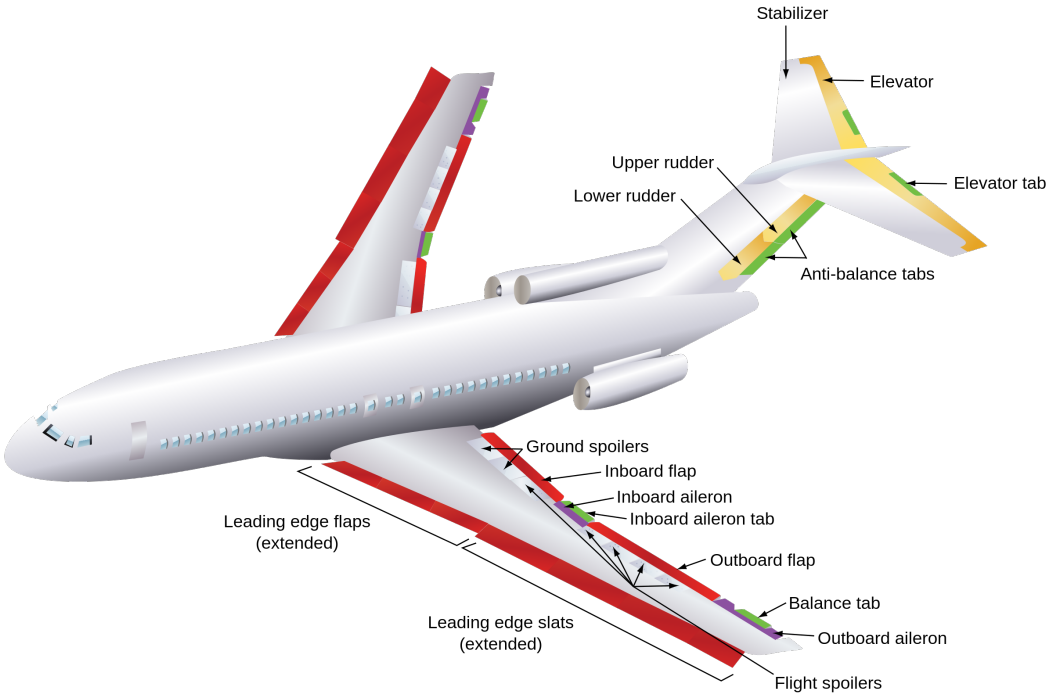


Figure 1.2: Flight control surfaces of Boeing 727 [16]

pilot back an indication of the forces acting on the moving surfaces. Additional requirements are that of high cut-off frequency, since the actuation command can vary multiple time a second and finally a very high level of reliability is required, since the failure of primary controls is a safety-critical event and often leads to aircraft loss. In order to mitigate such risk, multiple redundancies are used for various of the system in conjunction to alternative backup operating modes.

In traditional architectures, each control surface was design to maximize actuation on a single axis while decreasing, as much as possible, coupling on others.

Such surfaces are called:

- Elevators: surfaces actuated to create pitch moments along the longitudinal x axis (from aft to bow); located on the horizontal tailplane surface, behind the stabilizer.

- Ailerons: actuated in opposition to create a roll moment along the lateral y axis (from left to right wing); located on the trailing edge of each wing.

- Rudder: located on the vertical tailplane, the actuation creates yaw moments around the vertical z axis (from top to bottom of the aircraft).

The axis convention is shown in figure 1.3, following the one used in [15].

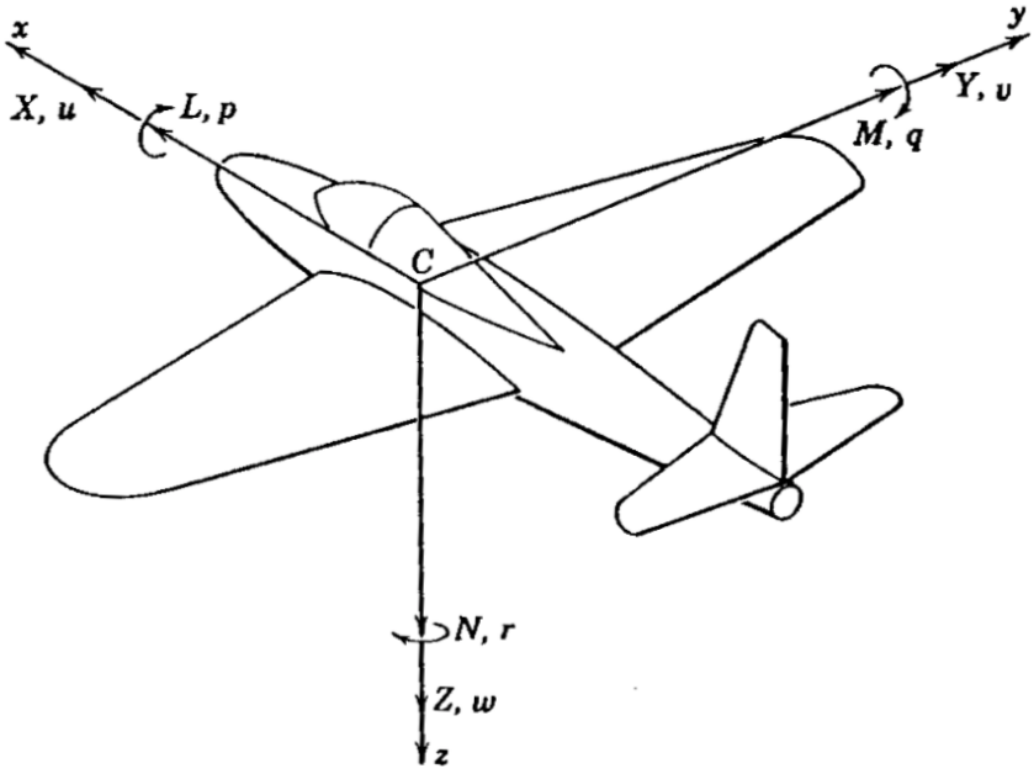


Figure 1.3: Aircraft body axis convention [15]

More modern implementations can include the coupling of two different functions in a single surface, like in the Aérospatiale/BAC Concorde, where control surfaces located along the wing trailing edge, called elevons, absolve both pitch and roll functions, since the aircraft architecture does not have any horizontal tailplane and subsequently no dedicated elevators (fig. 1.4).

Another distinction can be made between *reversible* and *powered* or *augmented* commands.

Reversible commands provide a direct mechanical connection between the surfaces and the pilot, in either a push configuration, using rigid bars, or in a push-pull configuration, using steel cables. The limit of such configuration is that the hinge moment of the surface must be overcome by pilot's strength alone. One method adopted to ease the pilot task was aerodynamic compensation, but these devices couldn't provide enough benefit at very high speed or with very large surfaces.

Figure 1.4: BAC/Aérospace Concorde [25]

The pilot inability to effectively actuate primary commands was overcome with the use of powered commands, where additional power is given by the hydraulic and more modernly electrical system. In today's aircrafts, the pilot only gives an input signal to a digital computer system that subsequently proceeds to the actuation and monitoring of the surfaces, in an implementation called *fly-by-wire* or even *fly-by-light*, depending on the nature of the data bus (electrical signals in the former, light signals in the latter).

Since the direct mechanical linkage between pilot and surface isn't anymore present, an artificial feel system needs to be included in the commands architecture to provide realistic feedback to the pilot, visible in fig. (1.5).

1.3.2 Secondary flight controls

Secondary flight controls function is to modify the geometrical and consequently aerodynamic characteristics of the main lifting surfaces, especially the wing, to adapt performances to all flight conditions (e.g. take-off, landing). Secondary flight control are not usually used continously, but are instead actuated in particular part of the flight mission; consequently, the actuation logic is typically stepped, with either ON/OFF logic (e.g. slats) or in well defined steps

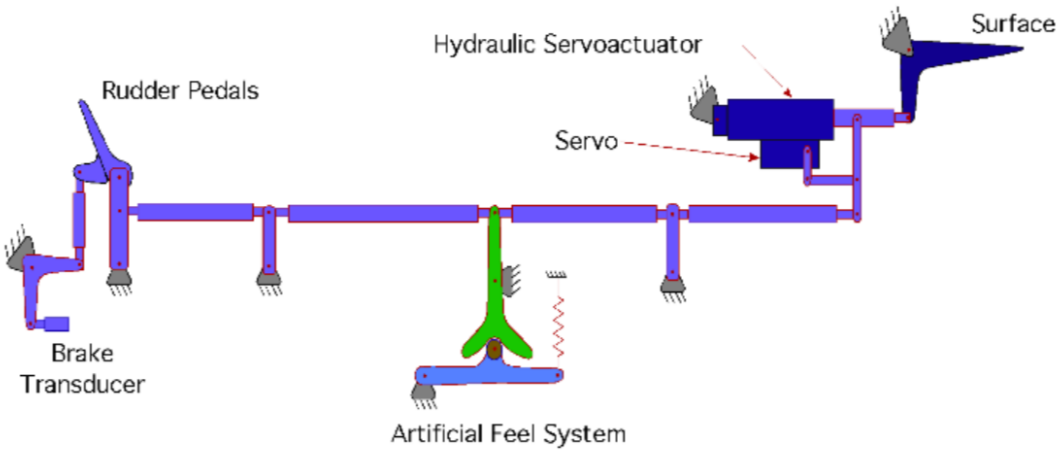


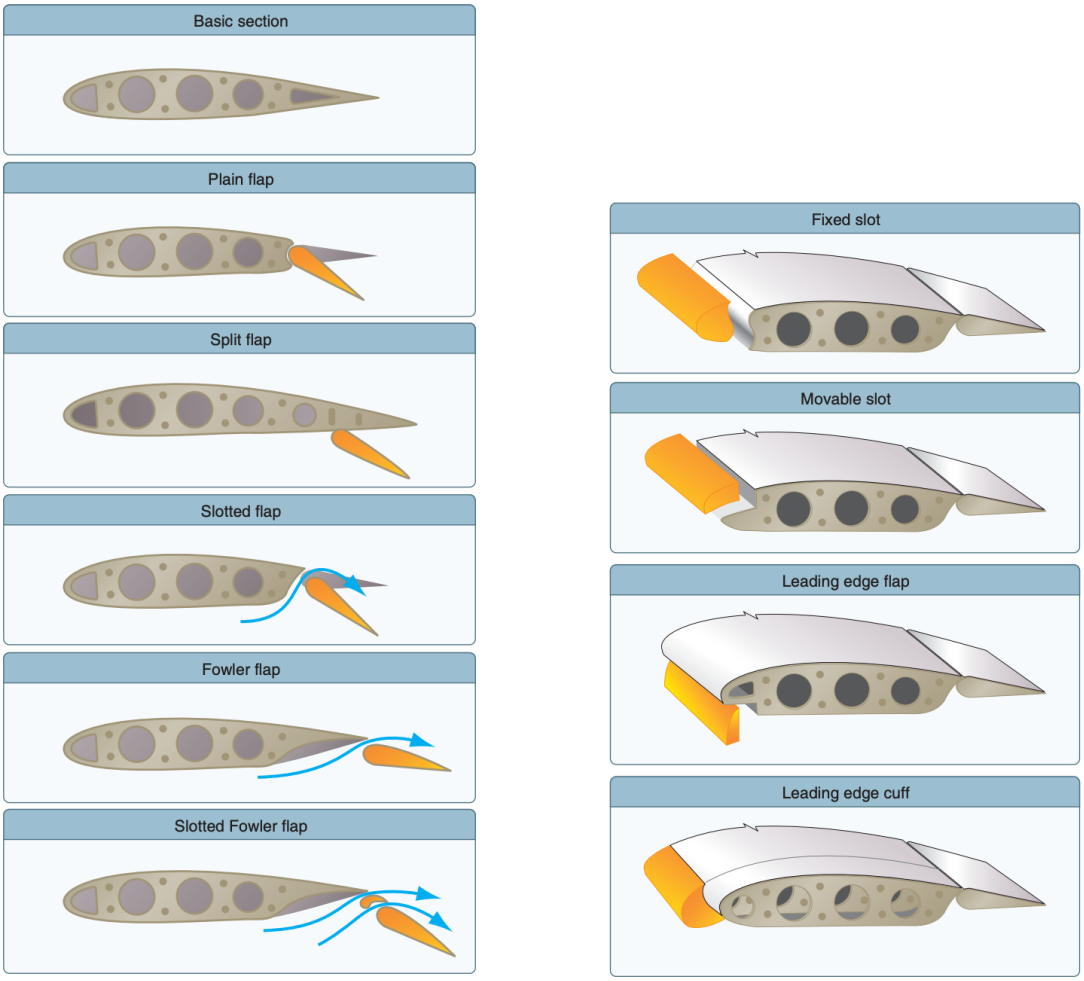
Figure 1.5: Typical scheme of powered flight controls [23]

(e.g. 0°-30°-45° for flaps). Actuation, except for spoilers in particular cases, is symmetrical.

The most common types of secondary flight controls are introduced.

- Flaps: typically installed on the trailing edge of the wing, flaps primary purpose is to increase lift coefficient C_L increasing camber of the surface, the wetted surface or both (fig. 1.6a), allowing lower stall speeds. The lift variation causes both an increase in induced drag and in positive (nose downward) pitch moment. Flaps are commonly used during take-off and especially during landing, allowing lower speed approach to the runway. Many different architectures have been proposed and installed during the years, starting form simple plain flap to double or even triple slotted fowler flaps on latest generation wide-body aircrafts.
- Slats: located on the leading edge of the wing, these surfaces energize and redirect the incoming airflow in order to increase adherence to the airfoil and delay separation to higher angle of attacks. Like flaps, many different architectures and implementations exist; some are shown in fig. 1.6b.
- Spoilers: installed on the upper wing surface, their function is to reduce lift while increasing drag. Useful for airspeed control and to descent without gaining excessive speed.

Modern implementations, like in military fighter aircrafts, include combination of secondary and primary flight control in a single surface. One example is the F16 flaperons, which has both primary (aileron) and secondary (flaps) functionality.



(a) Most common flap types

(b) Most common slat types

Figure 1.6: Flaps and slats types [3]

1.4 Actuation systems

Servomechanism are often used in aircrafts to actuate control surfaces. Servos generally measure their outputs and use them in a feedback loop, in order to achieve the required target requested, continuously updating the error value between command and current status. The most adopted actuation systems will now be briefly described.

1.4.1 Hydromechanical

Hydromechanical actuation has been the first to be employed on aircrafts. Pilot's command is transmitted via a mechanical linkage to an hydraulic valve, whose activation actuates the control surface. In typical configurations (e.g. F15, B737), a spool valve is used (fig. 1.7). Position feedback is achieved via a mechanical linkage; this type of response is subsequently proportional to the pilot input. After reaching the commanded position, error is null and no more actuation is provided.

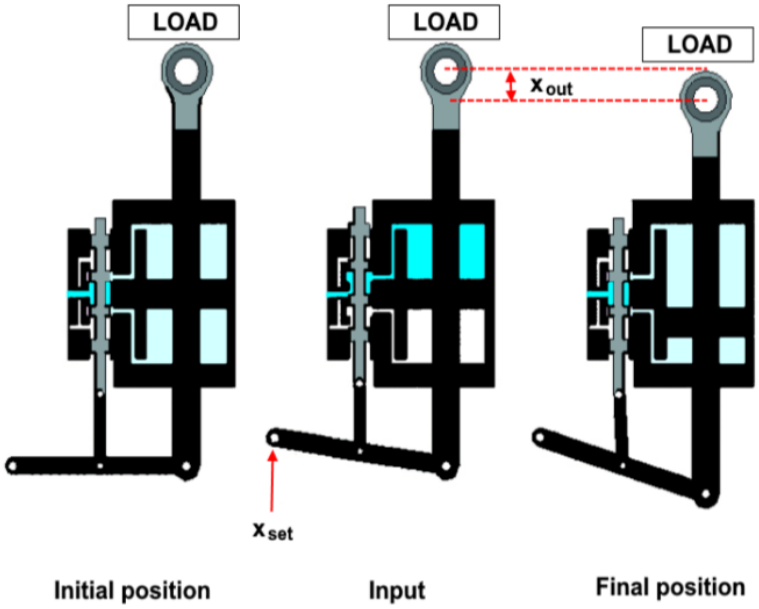


Figure 1.7: Typical scheme of hydromechanical actuation [27]

1.4.2 Electrohydraulic

An evolution of hydromechanical, electrohydraulic actuation uses an electric input instead of a mechanical one. The main element of the system is an electrohydraulic valve, commonly two-stage flapper-nozzle type (fig. 1.8), even tough many different types exist, like jet-pipe. Spool position is fed back to the first stage via a feedback spring, while the piston position is generally evaluated via a LVDT (Linear Variable Differential Transducer) to be sent to control electronics. Since the command is given using a fly-by-wire system, integration in autopilot and other control logics in the FCC (Flight Control Computer) is straightforward.

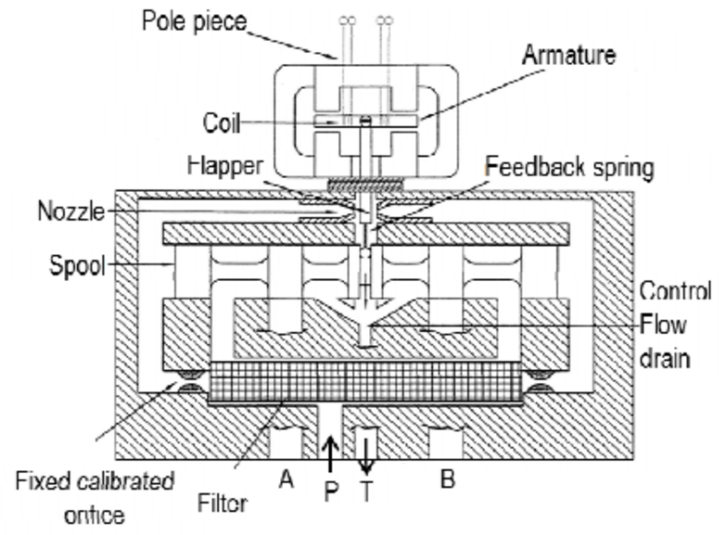


Figure 1.8: Two stage flapper-nozzle valve [13]

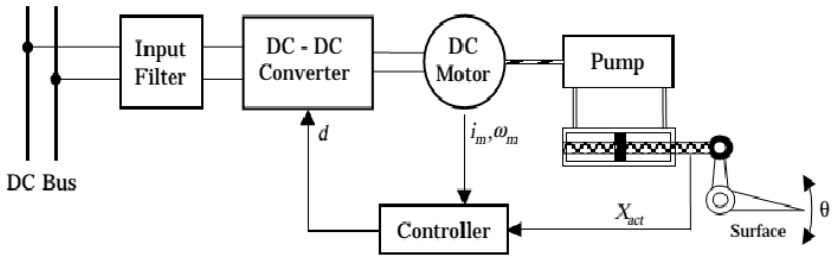
1.4.3 Electrohydrostatic (EHA)

Modern *more-electric* and *all-electric* trends favor the use of electric power for flight controls actuation; following these doctrines, hydraulics systems are becoming obsolete and are then being progressively replaced by integrated electrical actuators. One type is the electrohydrostatic actuator (EHA), where a brushless DC motor (BLDC) actuates a fixed displacement piston pump in a self-contained hydraulic loop: the resulting pressurized fluid is then used to actuate a jack connected to the surface to be controlled (fig. 1.9a). LVDTs are used to monitor jack position and feed it back to the control electronics. Additional elements includes anti cavitation valves, pressure relief valves and a small internal reservoir to account for thermal expansion of the fluid.

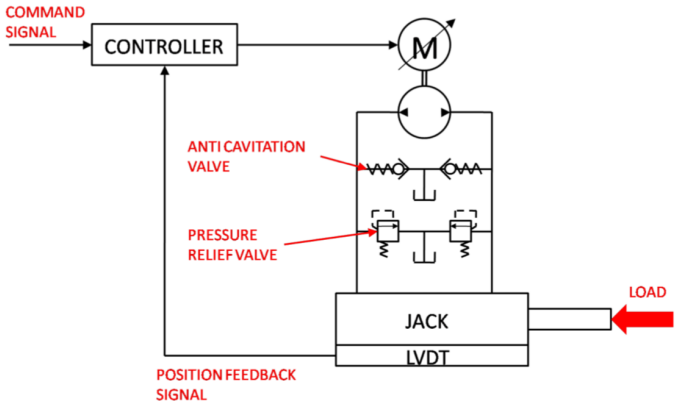
EHAs are becoming increasingly popular in new generation aircrafts, both military (e.g. F22 Raptor, F35 Lightning II) and civil (Airbus A350, A380), either as main or backup actuators.

1.4.4 Electromechanical (EMA)

The other typology of actuators favored in *more-electric* philosophy is the electromechanical actuator (EMA). This system is composed of a power source, almost exclusively a BLDC motor, a mechanical transmission connecting the power source and the final user, generally realized with gear reducer or rotary to linear converters (e.g. jack screw, ball screw, roller screw), and a feedback loop.



(a) EHA system implementation [2]



(b) EHA valve schematic [6]

Figure 1.9: EHA system architecture and valve schematic

EMAs are generally lighter, cheaper and easier to maintain compared to EHAs. To this date, on the other side, fault modes of EMAs are not fully understood, so their use as safety-critical elements (i.e. primary flight controls actuation) is still relegated to small scale UAVs, where hydraulic system installation would be too costly and impractical. EMAs are instead widely used as backup power source, like in the Boeing B787.

Referring to fig. 1.10, several elements are comprised a typical implementation:

- ACE (Advanced Control Electronics): this unit implements the particular control logic chosen for the actuator, calculating the error signal comparing the FBW signal and the feedback position signal.
- PDE (Power Drive Electronics): responsible of modulating the 3-phase AC power input in order to actuate the motor using the error signal deriving from the ACE.
- Electric Motor: usually a BLDC. It is the actuator source of mechanical power.

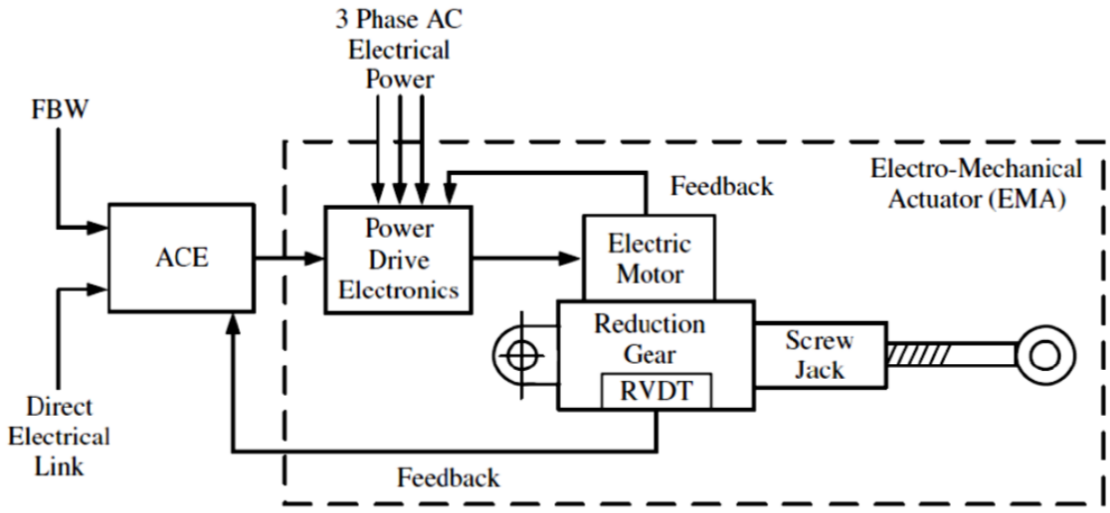


Figure 1.10: Electromechanical actuator scheme [6]

- Reduction gear and screw jack: this is the mechanical transmission used to connect the motor and the final user (typically a flight surface).

- RVDTs (Rotary Variable Differential Transducer): primary mean of angular position acquisition, used to create a feedback loop to the ACE. Usually used in conjunction with LVDTs to provide redundancies.

One important parameter characterizing EMAs is the transmission ratio, defined as

$$\tau = \frac{\dot{\vartheta_m}}{\dot{\vartheta_u}} = \frac{1}{\eta}\frac{T_u}{T_m}$$

that is, the ratio between motor and user angular speed or vice-versa the ratio between user and motor torque, divided by η which is the transmission efficiency.

The transmission ratio is important since reasonably sized BLDCs provide very little torque at very high angular speeds (several thousands to tens of thousands RPM) while the user requires angular speeds in the order of tens of degrees per second and high torques, so a transmission with a very high transmission ratio (hundreds or thousands to one) and high efficiency is mandatory to effectively actuate the user.

The type of reducers capable of such ratios while still having moderate size and weights are either epicyclic (planetary) gears or uncommon devices like harmonic gears.

Finally, rotary to linear conversion is achieved using either a ball or roller screw (fig. 1.11). A set of spheres or rollers rolls in a threaded path between the internal and external face of the screw. This leads to an overall higher efficiency compared to lead screws, since the elements are mainly rolling, so the main resistance is rolling resistance that is much lower than friction resistance. Higher maximum loads and better wear resistance are other prominent advantages.

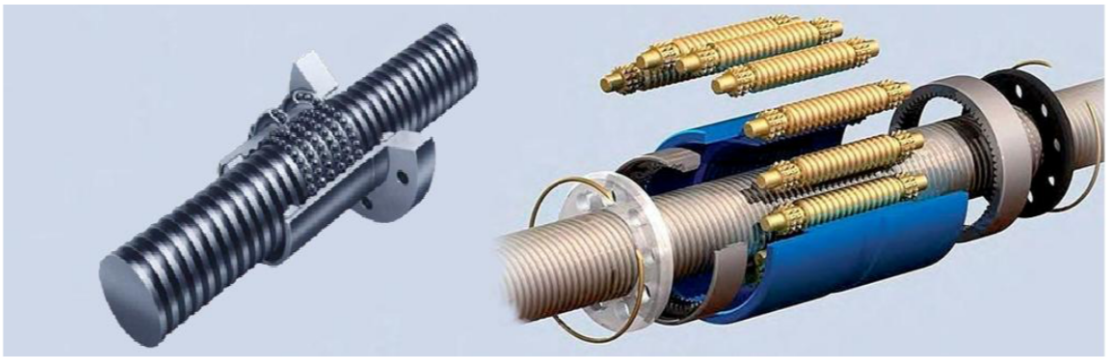


Figure 1.11: Ball screw and roller screw [27]

One of the biggest disadvantage of EMAs is the difficulty to hold a commanded position in presence of external disturbance loads: at zero or low speed, most of the energy given to actuate the BLDC is dissipated through the stator coils due to Joule effect. This could lead to overheating and possible damage of the motor. One solution could be the use of an irreversible transmission, but this approach has many drawbacks (lower efficiency, transmission lock in case of motor failure). A new solution to this problem is the adoption of a new type of motor, the hybrid stepper motor, capable of delivering high torque at zero or low speed.

Chapter 2

Brushless motors

The most common type of electrical motor used in aerospace applications is the BLDC Motor, or Brushless DC, also called Synchronous DC Motor, since magnetic field and rotor angular frequencies are the same. As the name implies, such motors transform DC power in rotary mechanical work; another advantage on competing architectures is the fully electronic commutation in contrast to, for example, classical brushed DC motors, where commutation is achieved by using mechanical commutators (the *brushes*). In BLDCs, the commutation, as mentioned before, is achieved electronically; there are multiple ways to achieve this, but the most common is the Hall-sensor based design, where Hall sensors located on the stator assembly senses the variation in magnetic filed induced by the rotor magnets. Consequently, it is possible to determinate the angular position of the rotor in order to properly activate the commutation sequence. Other sensing techniques include the measure of the CEMF on non-active coils to calculate rotor position, but this architecture is not well-suited for servo applications, since CEMF is difficult to be accurately measured at zero or close to zero angular velocity, where a servomechanism generally operates most of the time.

The absence of mechanical friction-based commutation is a great advantage, since there is much less wear and abrasive particle generation due to, as mentioned before, friction. Consequently, BLDCs operating life is much longer than comparable brushed DC motors.

There are even more positive aspects, including better torque-weight characteristics, higher maximum speed, better dynamic response and lower noise emissions. This long list of positive characteristics makes the BLDC a perfect candidate where reliability, weight saving and high performance are required, like in aerospace.

BLDCs ar generally quite simple machines (fig. 2.1), consisting in an external

enclosure (the *stator*), where the activation coils are located, and a rotor, where a certain number of permanent magnet is fixed in place. Another fundamental element is the control electronics and the relative position sensor for the rotor, used to correctly commute the various phases driving the motor itself.

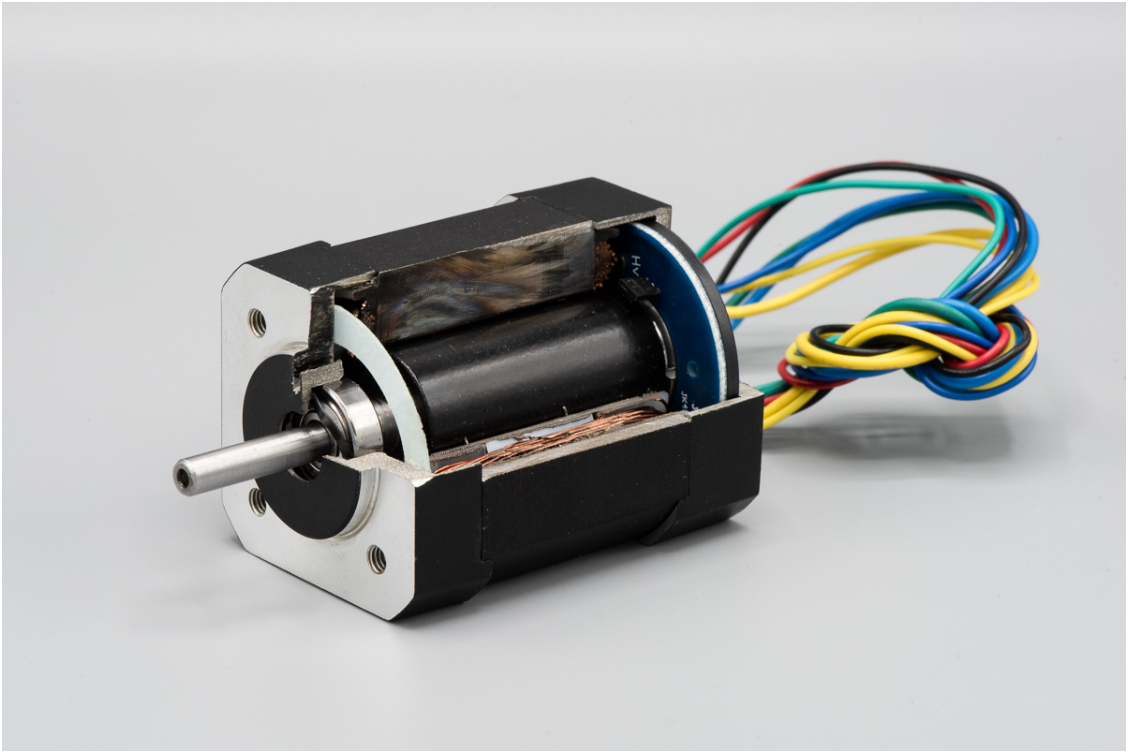


Figure 2.1: BLDC Motor Cross Section [7]

2.1 Stator

The stator of a BLDC motor is generally made of laminated steel, in order to minimize losses due to eddy currents. It's function is to hold the copper windings making up the excitation coils, wound up on the protruding ends of the core in the *slotted* configuration, one of the two possible configurations, the other being the *slotless* core (fig. 2.2).

The benefit of the slotless configuration is a lower inductance, leading to faster commutation and consequently higher maximum speed. Teeth absence in the core implies a reduction in *cogging* torques, making this architecture well-suited for low speed applications, while, on the other side, the increased air gap leads to an increase of ventilation losses and then a lower maximum useful torque.

Finally, slotless core have less irregularity in output torque, since the absence of teeth does not create ripples [10].

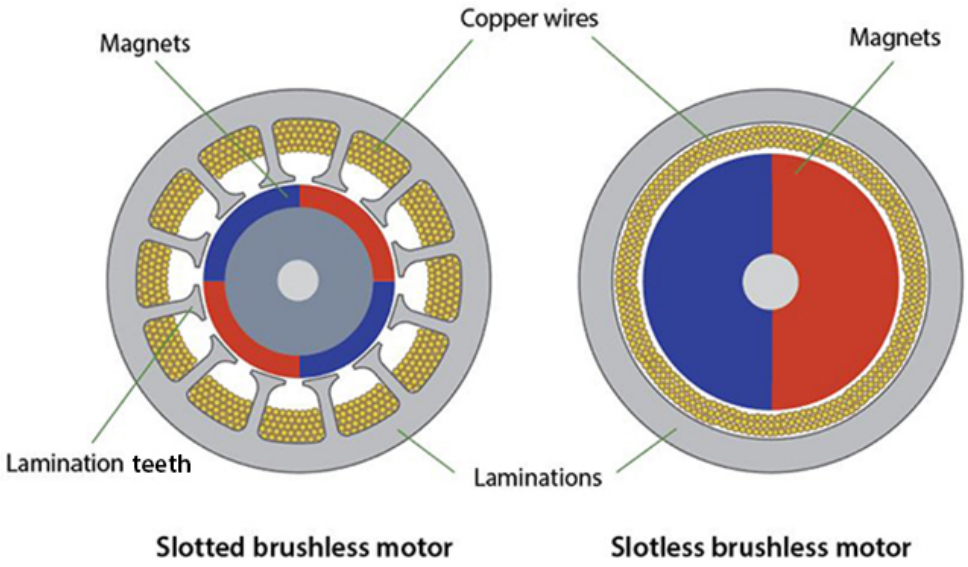


Figure 2.2: Slotted and slotless configurations [34]

Stators' electrical architectures can be of two different types, either a *star* or Δ (commonly called *triangle*) configuration (fig. 2.3).

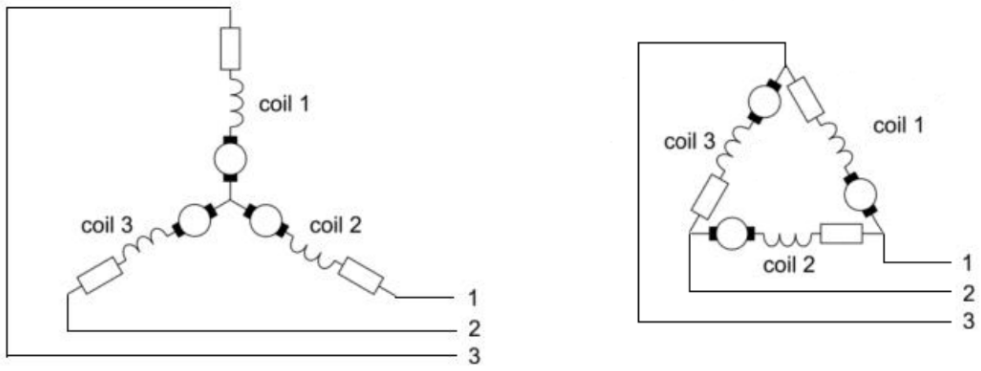


Figure 2.3: Star (left) and triangle (right) configurations [6]

The difference between the two architectures is the voltage seen by each winding: in the delta configuration, the phase voltage is equal to line voltage, while in the star configuration the phase voltage is equal to $\frac{1}{\sqrt{3}}$ of line voltage. This

implies, using the same line voltage, a higher current in the delta configuration, and thus higher torque. There is a drawback tough, since at each moment, using the Δ configuration, all the phases need to be powered. This also means a different commutation scheme compared to the star connection, which is why the latter is generally preferred, excluding some particular applications.

2.2 Rotor

The rotor is the mechanically active component of the motor. Several architectures have been developed, but all includes permanent magnets in the assembly. Magnets of various nature have been used, ranging from cheap but not very effective Ferrite to more expensive but performing rare earth based.

Material	**Ferrite**	**AlNiCo**	**SmCo**	**NdFeB**
Attraction force	Good	Average	Strong	Very strong
Max temperature	$\simeq 200°$	$\simeq 450°$	$\simeq 200°$	$\simeq 80°$
Corrosion resistance	Very good	Very good	Good	Average
Demagnet. resistance	Average	Low	Very high	High
Price	Very reasonable	High	Very high	Reasonable

Table 2.1: Permanent magnet materials characteristics

In aerospace, Neodymium based magnets (NdFeB - Neodynium-Iron-Boron) are almost exclusively used, given the very high magnetic flux generation (> 1 T). Such high value of the induction field are beneficial, increasing stator-rotor magnetic coupling constant, and thus torque.

Another design parameter is the number of rotor poles, varying depending on the particular application, generally ranging from two to eight pairs. A higher number of poles is useful to have smoother torque output, at the cost of maximum speed, given the increased commutation frequency and the physical limits of the bridge driving transistors (generally high power application MOSFETs).

Finally, the physical arrangement of the magnets inside the rotor core can be of two main typologies: *isotropic* and *anisotropic* (fig. 2.4). In the former configuration, magnets are installed on the external surface of the rotor, while in the latter they are inserted inside the rotor itself. A careful physical connection is mandatory in order to avoid magnets detachment and consequent catastrophic

motor failure at high speed, given the extremely high maximum speed, in the order of the tens of thousands of RPM and the subsequents centrifugal forces.

Figure 2.4: Isotropic (left) and anisotropic (right) rotors [6]

2.3 Operation principles

As the name implies, BLDCs convert DC current in mechanical energy. It is possible, in a very similar architecture, the Brushless AC Synchronous, to use sinusoidal AC current to drive the motor. Synchronous AC motors have the advantage of a more regular torque output, but are harder to manufacture and control, so BLDCs are more widely used.

Analogously to "traditional" brushed DC motors, the operation logic is a contionous rotor position feedback, to the control electronic in the brushless case, to actuate a proper commutation logic. In the brushed case, commutation is actuated by mechanical sliding contacts, the *brushes*, while, as aforementioned, in the brushless architecture, commutation is achieved by the use of a solid state inverter, creating AC phases from the supplied DC power.

To correctly time the commutation sequence, a rotor position feedback must exist. This is generally achieved using three Hall sensors, distributed 120° apart over the electrical revolution (function of number of poles), offering a 60° angular resolution. The placement is important since magnetic field changes every 60° and thus in one of the sensors. The sensed change is then fed back to the controller.

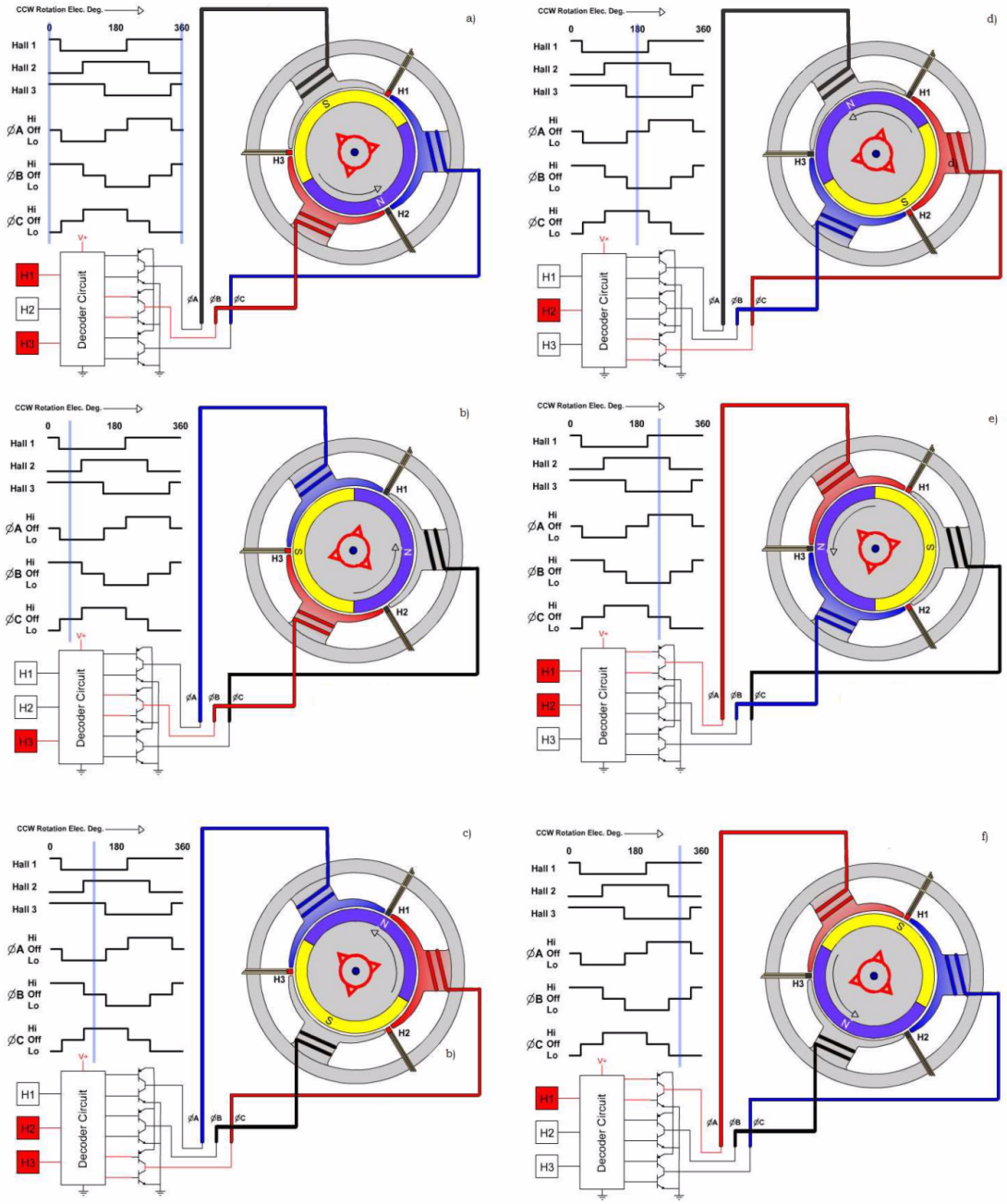


Figure 2.5: Typical BLDC commutation scheme [23]

In fig. 2.5 is shown the typical three phases BLDC commutation sequence. In subfigure *a*, Hall sensors H1 and H3 are located directly above the magnet south pole, so they give a high signal (logical 1), while sensor H2 is above the north pole giving a low signal (logical 0).

Phase A is then unpowered, phase B is connected to supply and finally phase C is shorted to ground. After 60°, north pole is under sensor H1, which then switches to a low signal; phase C is subsequently deactivated and phase A is shorted to ground, rotating the induction magnetic field 60° counterclockwise (subfig. *b*).

In subfig. *c* is shown the situation 60° later, where sensor H2 commutes to high, phase B is switched off and phase C is connected to line voltage, rotating the magnetic field further 60° counterclockwise. Commutation is then continued analogously to complete a full rotation as shown in the remaining subfigures.

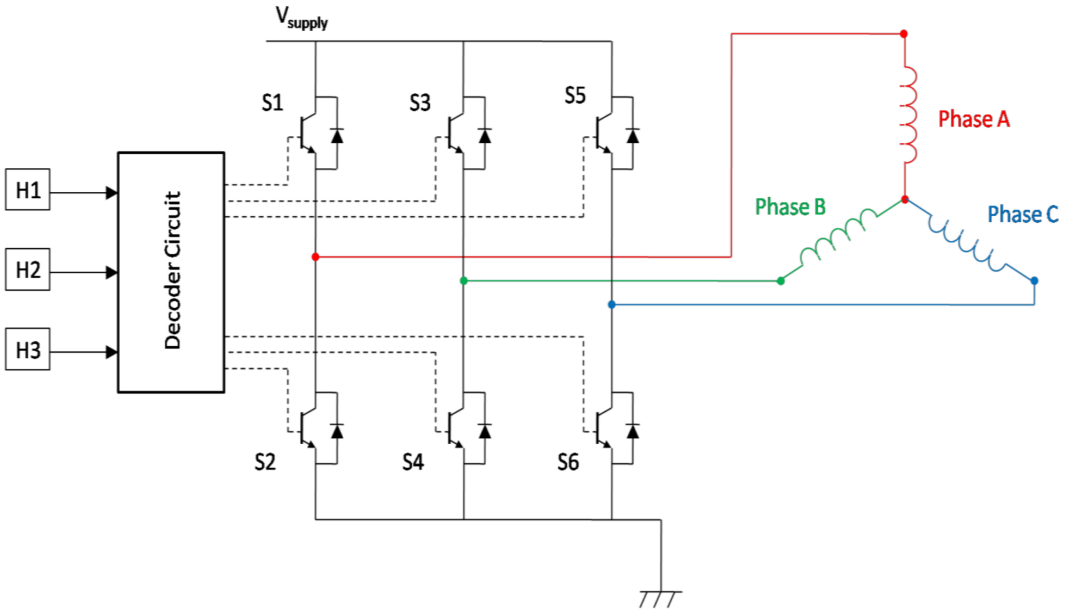


Figure 2.6: BLDC electrical scheme [27]

In fig. 2.6 is shown the complete electrical scheme, where Hall sensors' data are used as input in the decoder circuit where, using a timing table (fig. 2.7), six different signals are generated (three pairs), used as control signals for the six power transistors used to power the three motor coils. All transistors are connected to both line voltage and ground, and each couple control a single phase, to have both forward and backward action.

The timing diagram (fig. 2.7) shows the sequence used to activate the six transistors (Q1L to Q3H) and the state of the three phases in each commutation

	Hall sensors			Switchs						Phases			Windings		
Phase	H3	H2	H1	Q1L	Q1H	Q2L	Q2H	Q3L	Q3H	P1	P2	P3	V_{1-2}	V_{2-3}	V_{3-1}
I	1	0	1	0	1	1	0	0	0	$+V_m$	Gnd	NC	$-V_m$	-	-
II	0	0	1	0	1	0	0	1	0	$+V_m$	NC	Gnd	-	-	$+V_m$
III	0	1	1	0	0	0	1	1	0	NC	$+V_m$	Gnd	-	$-V_m$	-
IV	0	1	0	1	0	0	1	0	0	Gnd	$+V_m$	NC	$+V_m$	-	-
V	1	1	0	1	0	0	0	0	1	Gnd	NC	$+V_m$	-	-	$-V_m$
VI	1	0	0	0	0	1	0	0	1	NC	Gnd	$+V_m$	-	$+V_m$	-

Figure 2.7: BLDC motor timing diagram [11]

state depending on the signal given by each Hall sensor. The correct actuation of the transistors by the controller is of paramount importance to the proper function of the motor.

2.4 Mechanical characteristics

In a BLDC motor, as in every other electric motor, the mechanical action is created by the interaction of the two magnetic fields generated by stator and rotor. The stator coil can be modeled as a magnetic dipole immersed in an external magnetic field created by the rotor's permanent magnets. The magnetic dipole can be calculated as:

$$\mathbf{m} = NAi\mathbf{n}$$

where $\mathbf{m}$ is the magnetic dipole moment, $\mathbf{n}$ is the normal versor to the spire plane, A is the area of each winding and N is the number of turns in the coil.

The torque can then be evaluated as:

$$\mathbf{T} = \mathbf{m} \times \mathbf{B} \quad .$$

In general, torque is proportional to applied current by means of a coefficient k_T, called *motor torque constant*. Using a vector approach, torque can be thought as a vectorial sum of three contributes, one for each of the phases.

Using the previous equation, it can be seen that torque is maximum when magnetic dipole moment and magnetic field are orthogonal to each other; this phenomenon is used during commutation, trying to maintain the coil-induced magnetic field as close to 90° in advance in respect to the rotor magnetic field as possible.

There are several ways to increase maximum torque: either by changing coils geometry (increasing the area and/or the number of turns, but this means bulkier

and heavier motors), using higher rotor's magnetic fields (adopting stronger magnets) and by increasing supplied current. However, there is a maximum level of current, since Joule's Law states that ohmic losses are proportional to the square of resistance, so there is a non linear relation between current and heat, leading to overheating and, possibly, damage. Another limit is set by the saturation of flux density in the magnets, so no further torque can be generated in saturation condition.

A variation of the magnetic flux is generated in each coil due to rotor motion, which induces, according to Faraday's Law, a counter electromotive force on the coil itself; such voltage is, approximately, proportional only to angular speed by a coefficient k_e called *counter electromotive force coefficient* or simply *motor voltage constant*. It can be shown that $k_e = k_T$.

It is apparent that torque and current progressively decrease as angular speed increases, since:

$$V = Ri + k_T\omega \quad ,$$

and consequently:

$$T = k_T i = \frac{V k_T}{R} - \frac{k_T^2}{R}\omega \quad .$$

It follows that output power is:

$$P_{out} = T\omega = \frac{V k_T}{R}\omega - \frac{k_T^2}{R}\omega^2 \quad .$$

These trends, which are approximations, are shown in fig. 2.8.

Up to this point, a very simplified model has been used, not taking into account all non-linear effects. A much more realistic speed-torque characteristic is presented in fig. 2.9, where an efficiency graph is superimposed. It is apparent that highest efficiency is achieved at high RPMs; this is one of the reason why gearings are often necessary in servomechanisms operating at low speed.

Furthermore, two different regions can be individuated: the *continuous operation* zone, below the green line, and the *intermittent operation* zone, between the green and blue line. It is indeed possible to operate a BLDC at very high currents, much higher than nominal continuous condition, for short periods of time, and thus at very high torques. The drawback is that the action must be limited in time and an appropriate amount of time is needed in order to cool the stator coils.

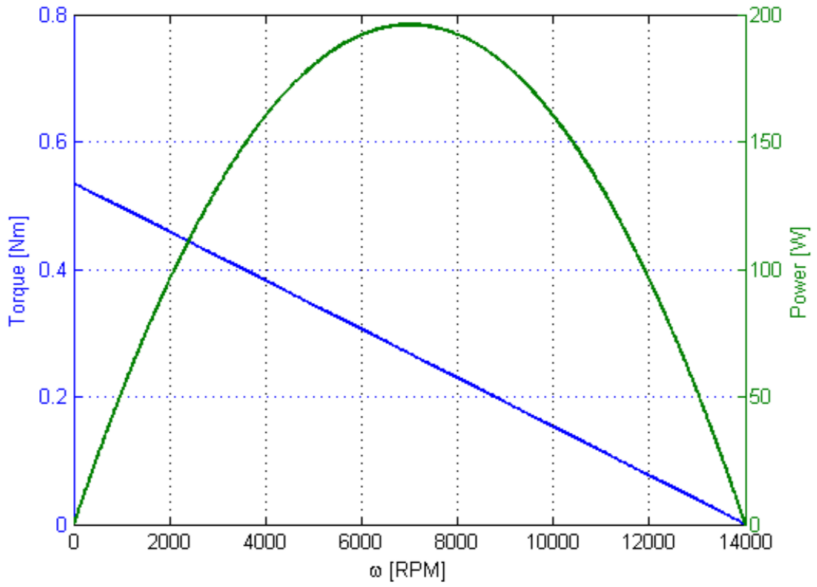


Figure 2.8: Ideal BLDC speed-torque characteristic [23]

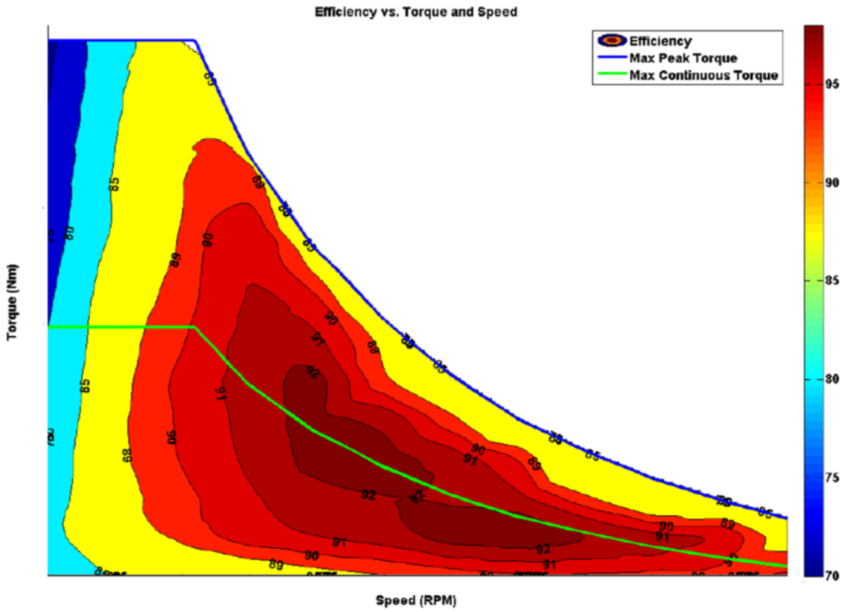


Figure 2.9: Realistic BLDC speed-torque characteristic [23]

In fig. 2.9, three saturations are introduced: one capping the max allowable angular speed, another limiting max stall torque and finally a limit on commutation frequency. It is then possible to graph a simplified motor characteristic (fig. 2.10).

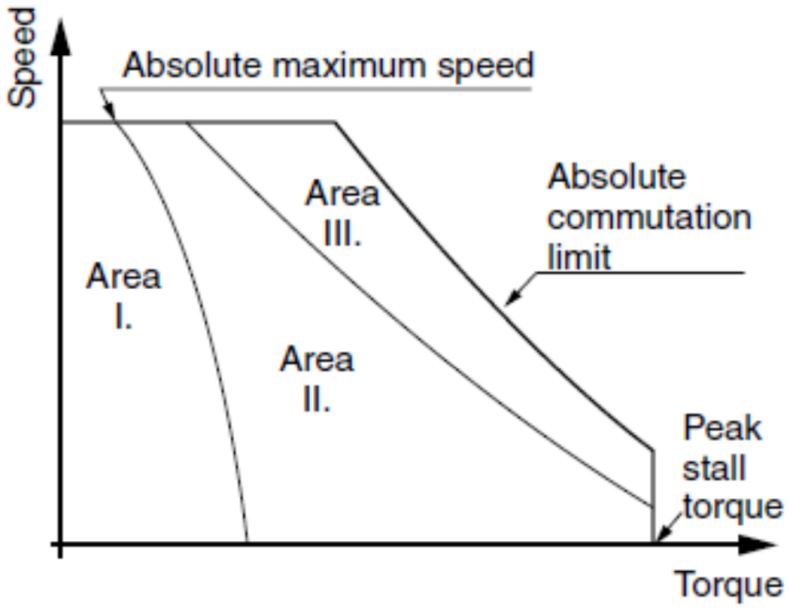


Figure 2.10: Realistic BLDC speed-torque characteristic, simplified [23]

2.5 Motor control

There are various ways to control a BLDC motor, using different control parameters depending on the type of regulation required.

2.5.1 Speed control

This kind of regulation is most often used in applications where a predetermined rate is used as control variable, e.g. hydraulic compressor, or when used as inner control loop in position control machines, e.g. servoactuators.

Since the phase commutation sequence is only controlled by the rotor position, as measured by sensors, it cannot be used as control variable. Angular speed regulation is then achieved by modifying the supplied voltage amplitude.

The problem is now shifted on creating a desired voltage amplitude by modifying supply voltage. This can be achieved either analogously, by using

a potentiometer, but this solution has terrible efficiency and relies on moving physical components reliability.

The modern solution is to use PWM - Pulse Width Modulation; using this fully-electronic digital technique, it is possible to create potentially every voltage between line and ground, depending on the maximum supported frequency. In essence, the techniques consists in the modulation of the duty cycle of a high-frequency carrier wave using the reference signal. The output voltage is then proportional to the duty cycle of each square wave.

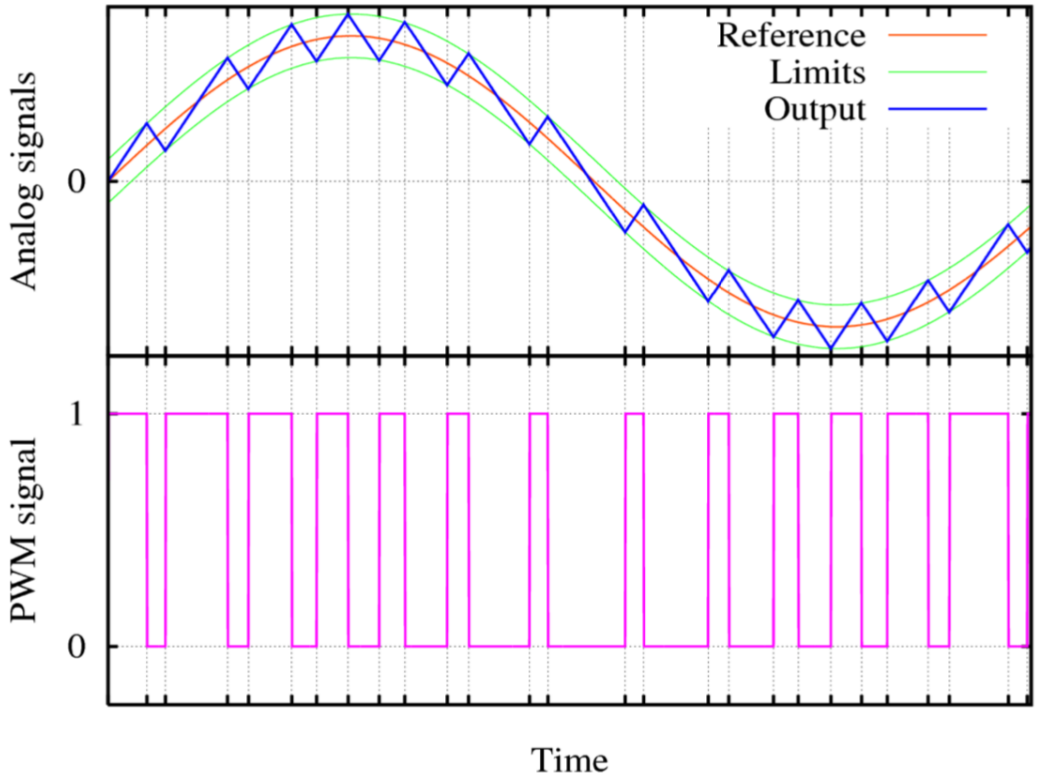


Figure 2.11: PWM signal generation [6]

The following block diagram (fig. 2.12) shows the most commonly used architecture to implement speed control: given a required speed constraint, the *speed computation* block calculates angular speed based on Hall sensor switching speed; this information is then fed to the controller that subtracts this value to desired one determining the error signal. Via a PID controller, or similar, a PWM signal is created and then feed to the inverter powering the transistors. The loop is then closed by the *commutation logic* block, where the signal for each of the six control lines is created.

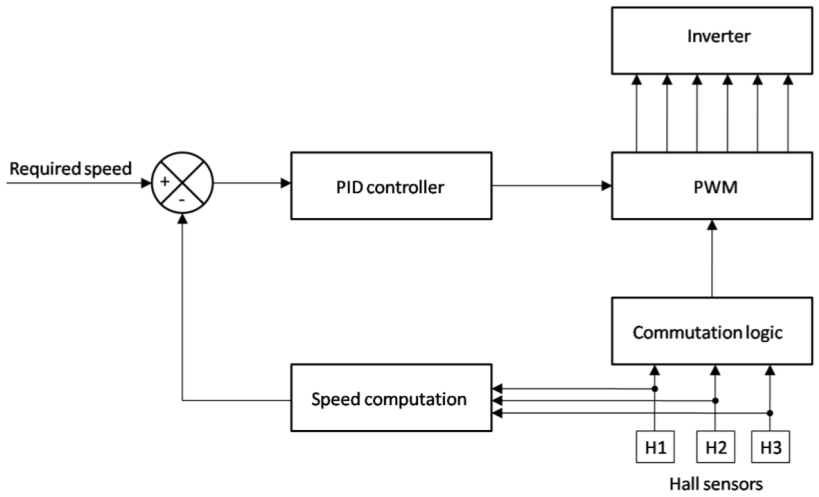


Figure 2.12: Speed control implementation, block diagram [6]

2.5.2 Torque control

The other method to achieve regulation is torque control, used in some industrial implementations where a continuous level of torque is required or, again, as inner feedback loop of a position control servomechanism.

In this logic, output torque is desired to be constant, so motor speed is changed to achieve this also accounting for varying external loads.

Since motor torque output is dependent on magnetic flux in the stator coils, a realistic implementation is not easy to achieve and advanced analysis, like electromagnetic FEM analysis will be required; nonetheless, a simplified model can be implemented, recalling the direct proportionality between torque and current by means of the motor torque coefficient, k_T.

Torque control is thus achieved modifying phase currents: total current can be computed using ground resistance line, since this resistance create a voltage drop. The measured voltage drop, multiplied by k_T, is compared to the required torque value; the difference is then the error signal, fed to the PID (or similar) controller in order to modify the PWM duty cycle, analogously to the speed control logic.

The only difference between this two logics is just the variable used to compute the error value, as visible in the block diagram representation (fig. 2.13).

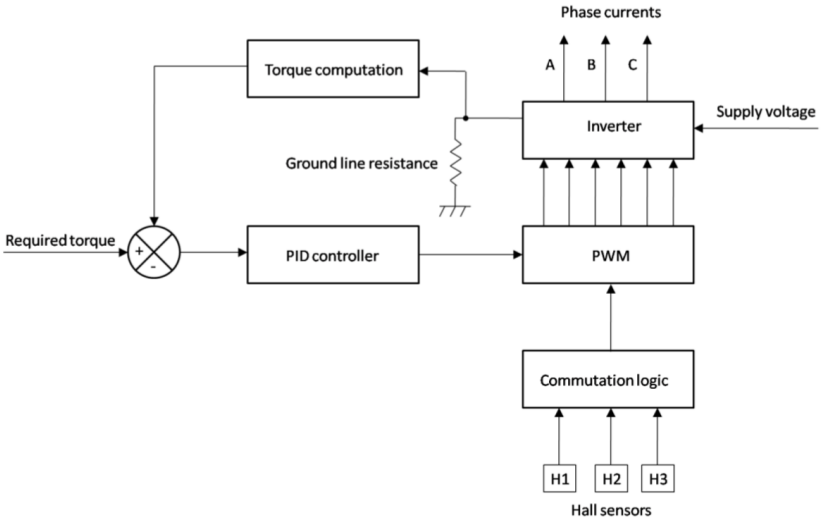


Figure 2.13: Torque control implementation, block diagram [6]

2.6 Protection systems

Several condition cam make the motor operation unsafe or even dangerous. In order to protect the motor itself, a series of limits need to be imposed regarding different variables. For example, if the rotor gets stuck, the CEMF drops to zero and there is a sudden current and power surge to the phase coils, causing overheating and possibly shorts, magnets demagnetizations (operation is confined under the Curie temperature) or power electronics destruction. Another dangerous situation is a speed reversal: the sudden change in magnetic flux induced by the stator can potentially demagnetize the rotor permanent magnets.

To prevent such conditions, a series of safes are set in place. One of them is *peak current*, i.e. the maximum allowable current allowed during motor start-up; if a higher value is computed by the driving electronics, a saturation at this level is set for each value above.

Similar to peak current, *maximum working current* is the maximum allowable current during nominal operations; the same saturation logic is applied in this case.

Other integrated protection system shield the motor from *undervoltage* while battery-powered, since lithium or lithium-polymer batteries rapidly degrades if voltage drops under a certain level. *Overvoltage* protection is always applied since high voltages can damage controller, power electronics or the coils, potentially causing an electric arc.

Finally, a logic to determine faulty Hall sensors is integrated, since such sensors are fundamental in the correct operation of the motor; implementation is relatively simple since only predetermined combination of their outputs is allowed by motor geometry. Furthermore, only one sensor can change its state at every commutation and the tree sensors can not output the same value. Anyway, since such failure is usually sudden and leads to total loss of the motor, it is not useful for prognostic purposes.

Chapter 3

EMA model description

In this chapter, the EMA model used for approximating the real-life response of an electromechanical actuator will be described.

As mentioned above, a very detailed MATLAB-Simulink EMA model has been used to perform analysis sensing the variation of CEMF in function of various damage parameters. This approach has many advantages over real-life systems: much lower cost, faster simulation, ease of simulation since progressive faults, e.g. partial coil shorts, are easily implemented by just varying the appropriate coefficients. In particular, the system modeled is a F-16 combat flap, which has an intermediate response time between a primary and secondary flight control.

Since a wide range of phenomena is modeled in the system, a very step size has been chosen, thus making the simulation very resource intensive. The time interval selected for the analysis is equal to 1 s, approximately taking 50 s to simulate using a step size equal to $1 \cdot 10^{-6}$ s $= 1$ μs.

The integrator chosen is first-order Runge-Kutta with a fixed step size, commonly known as Euler's method. Higher order integrators might have delivered faster simulations, but the stiff nature of the equations used might have posed a convergence problem where the solver would have decided to use longer time step sizes.

3.1 Model overview

The model (fig. 3.1) is composed of various subsystem that will be further explored in the following subsections.

On the left one can see the *Com* block, which is the subsystem used to model the commanded position to be achieved by the system.

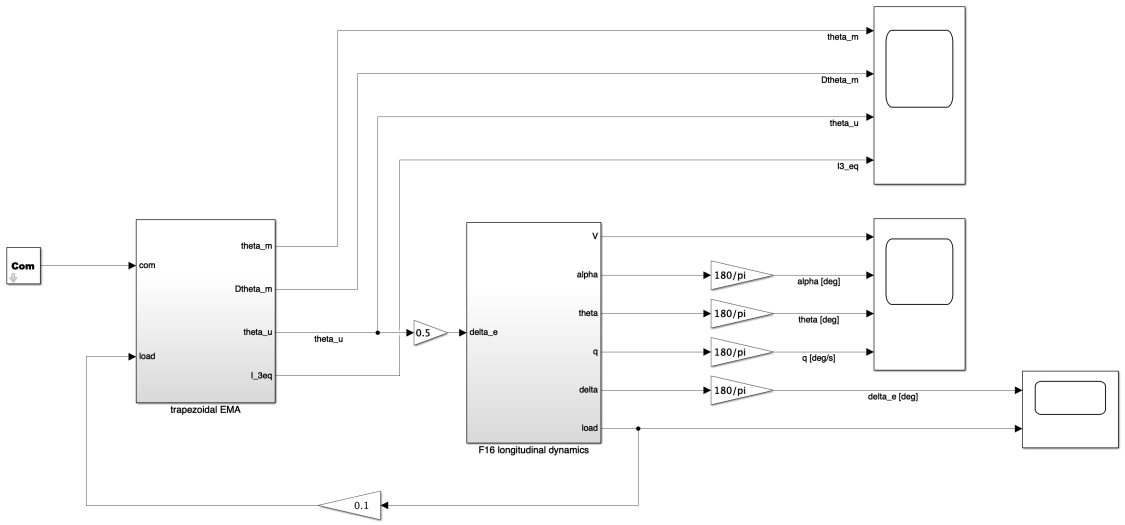


Figure 3.1: Simulink model overview

The commanded position is used as input in the *trapezoidal EMA* block, modeling the proper servoactuation system.

Finally, the *F16 longitudinal dynamics* subsystem models the response of the aerodynamic surface considered.

Additional elements are placed on the right-hand side; these elements are *scopes* used to quickly monitor important electric and mechanical variables during and after the simulation.

As previously mentioned, the system is quite complex since it is modeled up to component level, so every subsystem will now be further expanded and commented, expanding each sequent subsystem up to component level.

3.2 Com subsystem

This block (fig. 3.2) is responsible of generating the user command, choosing from different options (step, ramp, sine wave, chirp or user-defined). Several parameters can be altered in order to achieve the desired command. Several different can be used in conjunction simply by selecting the appropriate base functions.

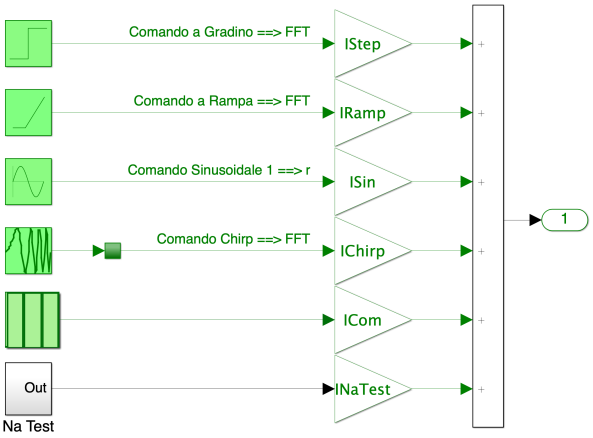


Figure 3.2: Com block overview

3.3 Trapezoidal EMA subsystem

The trapezoidal EMA subsystem is the most complex block included in the model. In fig. 3.3, the subsystem elements are clearly visible. This system is composed of multiple subsystems, including electric, electronic and mechanical elements. It has to be noted the useful values for further analysis are filtered using a low-pass filter before being saved in the MATLAB workspace.

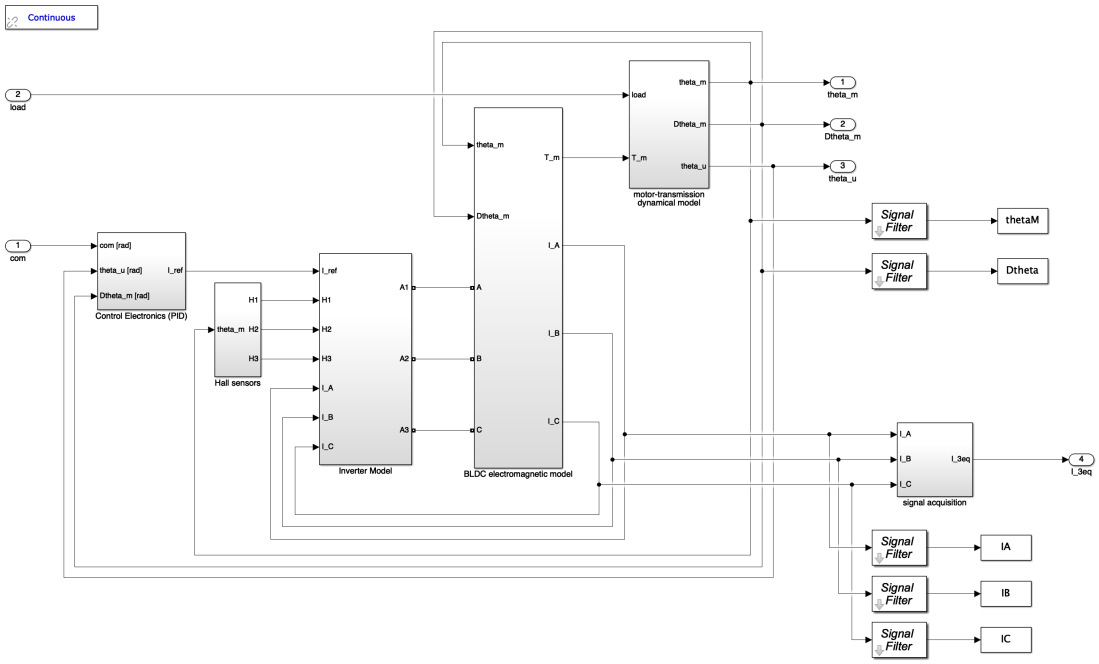


Figure 3.3: Trapezoidal EMA subsystem

3.3.1 Control electronics

Starting on the left-hand side, the first subsystem encountered is the *Control electonics (PID)* block (fig. 3.4). Its task is to calculate the error signal, based on commanded value (*com* input line) and user position (θ_u line). The block receives one more input, that is motor angular velocity ($\dot{\theta}_m$) to also evaluate the angular speed error.

The controller used in the system is a simple PID controller, which uses a combination of a proportional action, as the name implies having a proportional relation to the error signal via a coefficient, an integral action, used to correct steady-state errors that would not be possible to eliminate using a solely linear action and finally a derivative gain, useful during rapid command variation augmenting the system response time.

It has to be noted that both speed and current are limited using two *saturations*, since it is unrealistic to allow the system to operate over stated specifications, since such conditions will cause damage to the motor assembly.

Finally, some electrical noise is added in order to include small, realistic disturbance to the main signal.

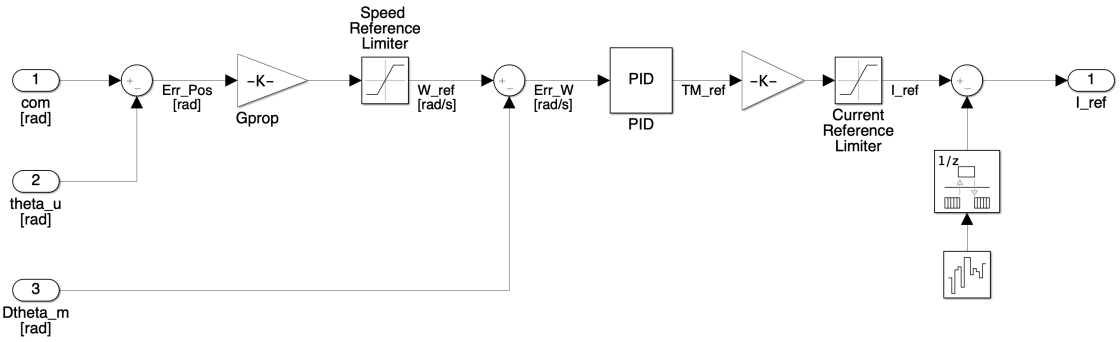


Figure 3.4: Control electronic subsystem

3.3.2 Hall sensors

Hall sensor block is relatively simple, taking as input the mechanical angular position of the motor and outputting three variables, H1, H2 and H3, representing the high/low signal emitted by the sensor themselves.

The implementation (fig. 3.5) starts with the computation of θ_e, the electrical angle, based on mechanical position θ_m and the number of motor pole pairs, p, using the following formula:

$$\theta_e = 2\pi \left(\frac{p\,\theta_m}{2\pi} - floor\left(\frac{p\,\theta_m}{2\pi} \right) \right)$$

After having determined the electrical angle, three simple lookup tables are used in order to decide if the signal of each sensor should either be high or low.

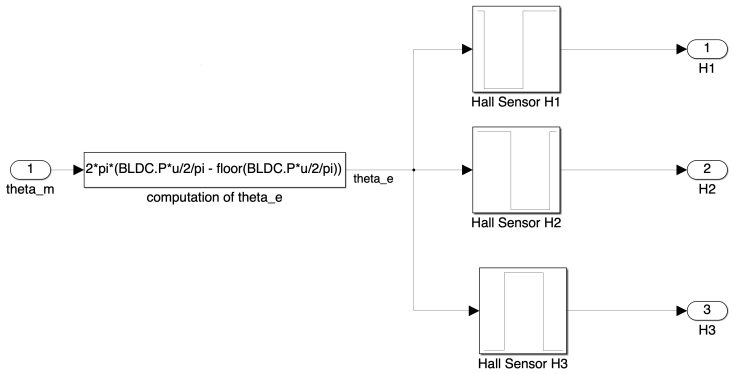


Figure 3.5: Hall sensors subsystem

3.3.3 Inverter model

The inverter model (fig. 3.6) is used to model a digital PWM inverter. The assembly takes 7 inputs (3 Hall sensors signals, 3 phase currents and the reference current).

Firstly, the reference current signal (a purely controllistic value, not related to any physical quantity) is 'split' in three different reference currents by using the three Hall sensor signals (fig. 3.7a). This is done passing as input each error signal through an hysteresis block, which returns 1 if the input is greater than $+hb$, that is greater than half of the set deadband, 0 if it is less than $-hb$ (again, half of the deadband) and keeps the previous value until the signal gets inside the deadband value.

One limitations on the integrations step is posed by the Nyquist-Shannon sampling theorem; the integration (and consequently discretization) step must be at least one order of magnitude greater than the intended maximum switching frequency (i.e. above 10 kHz) to avoid aliasing problems.

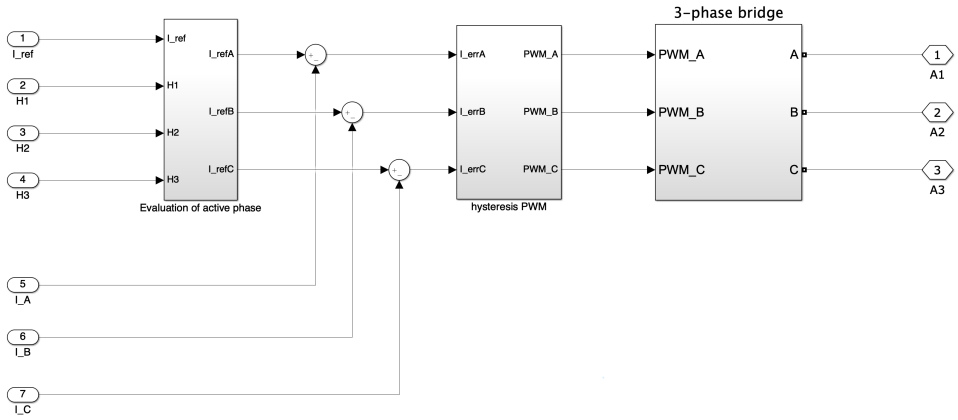
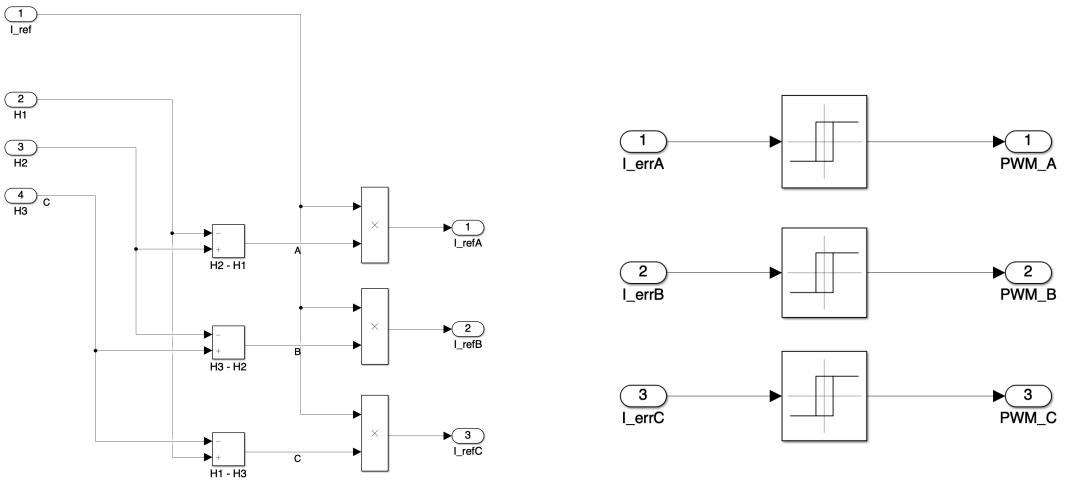


Figure 3.6: Inverter model in detail

After reference currents calculation, the current error signals are generated and fed to the *hysteresis PWM* subsystem, used to generate the three PWM signals used to drive the 3-phase H-bridge (fig. 3.7b).



(a) Active phase evaluation block

(b) Hysteresys PWM block

Figure 3.7: Phase evaluation and hysteresis PWM subsystems details

The last element composing the subsystem is the 3-phase H-bridge (fig. 3.8). Subsystem inputs are the 3 PWM signals. The actual modeling is done using a *Universal Bridge* Simulink block, from the Simscape library, capable of easily simulating various complex system, including electrical components; the bridge is then connected to a DC power source, that is the inverter supply voltage.

It has to be noted that, when one bridge transistor is active, that is connecting

one phase to line voltage, the corresponding same-phase transistor, connecting the same phase to ground is off, in order to avoid a short to ground. This is done easily since the PWM signals are complementary.

The output of the block and also of the Inverter model subsystem are the three voltages $A1$, $A2$ and $A3$, that will be passed to the *BLDC electromagnetic model* subsystem that will be shortly described.

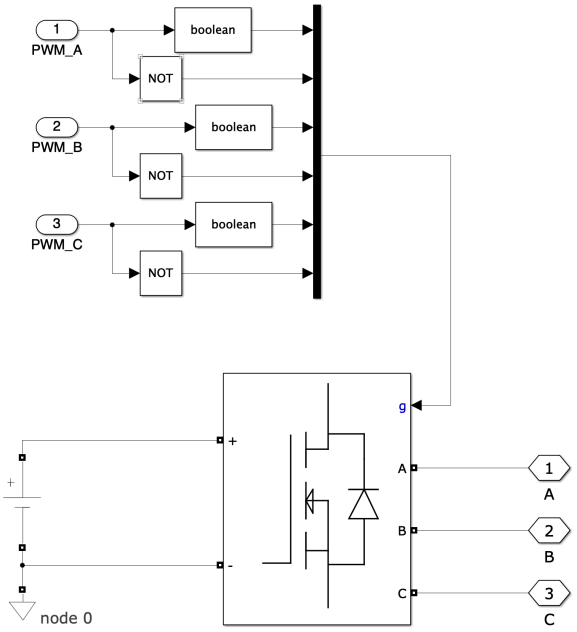


Figure 3.8: H-bridge subsystem

3.3.4 BLDC electromagnetic model

This subsystem aims at modeling the electromagnetic interaction between the motor rotor and stator.

The first subsystem encountered is block calculating the back-EMF (or CEMF) coefficients (fig. 3.9).

The actual value calculated is the *normalized* CEMF, defined as:

$$CEMF_{norm} = \frac{CEMF}{\omega_m} = k_{cemf}$$

that is the ratio between the actual CEMF and the motor angular speed.

In every motor with a number of pole pairs greater than one, the rotor eccentricity effect is function of the rotor position, such as:

$$k^i_{cemf} = k^i_e(\theta_m)\left(1 + \zeta\cos\left(\theta_m + \frac{2(i-1)}{3}\pi\right)\right)$$

where k^i_e is the trapezoidal wave-shaped normalized CEMF relative to the i-th phase of the non-damaged model, while $\zeta = \frac{x_0}{g_0}$ is the ratio between the rotor axis offset from the center and the nominal air gap.

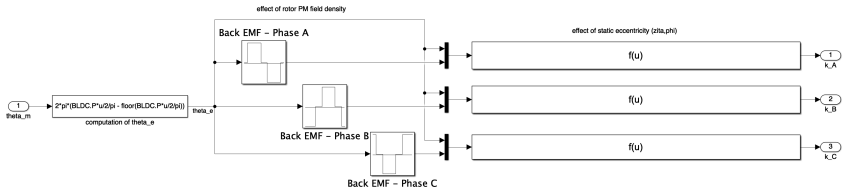


Figure 3.9: CEMF coefficient calculation block

After multiplication with the motor angular speed, the value is passed to the three phase RL model (fig. 3.10).

The block, fully developed using Simscape library, models each stator coil behavior. The block receives three commanded voltages and three CEMF values, previously calculated.

The block outputs are the actual voltage drops across the phases and each phase current. The subsystem includes conversion blocks, useful to convert the Simscape signals to Simulink data, just like a real-life sensor (e.g. voltmeter, ampmeter) in order to save them in the MATLAB workspace for further processing.

Finally, in the motor torque calculation block (fig. 3.11), the actual torque output is calculated in function of the three coils currents and CEMF coefficients. The three contributions are summed to obtain the resultant torque that is limited with a saturation block to account for the stator core magnetic flux saturation.

3.4 Transmission dynamical model

The next important subsystem aims at properly model the transmission between motor and final user.

The subsystem inputs are the motor torque and the external load applied to the surface. The system outputs are user position and angular speed.

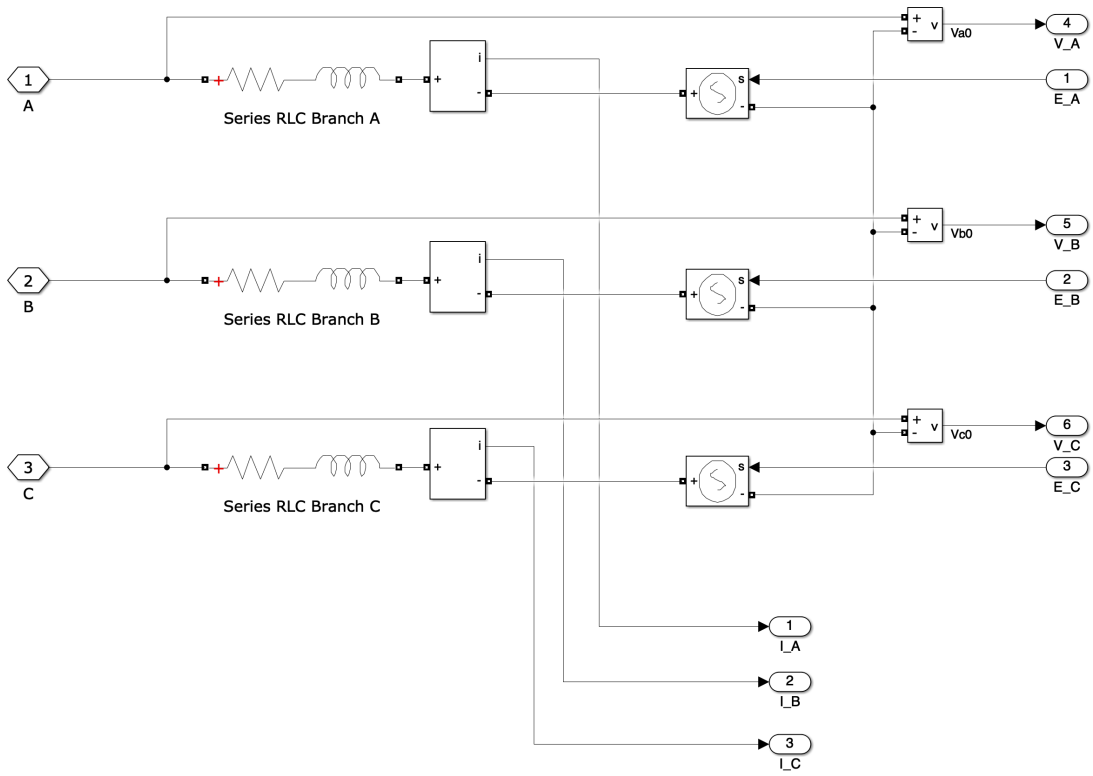


Figure 3.10: 3-phase RL model block

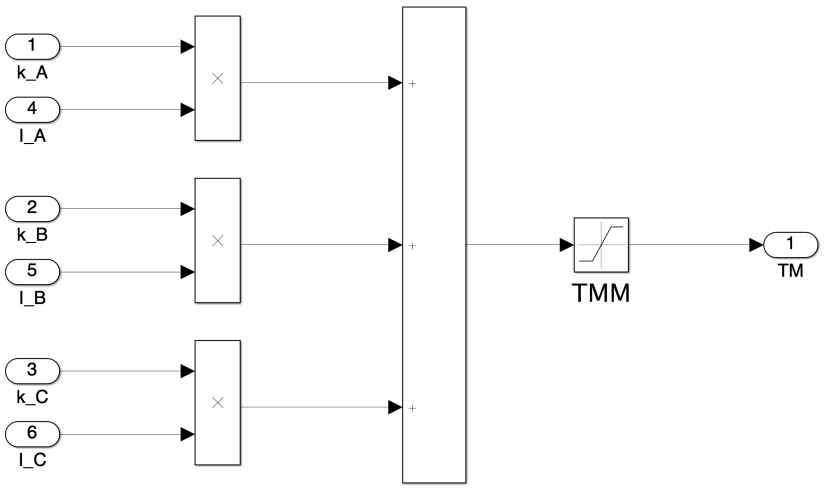


Figure 3.11: Motor torque evaluation block

The motor-user transmission is evaluated as a single degree of freedom, second order system, including inertial and viscous effects. Non-linearities are also

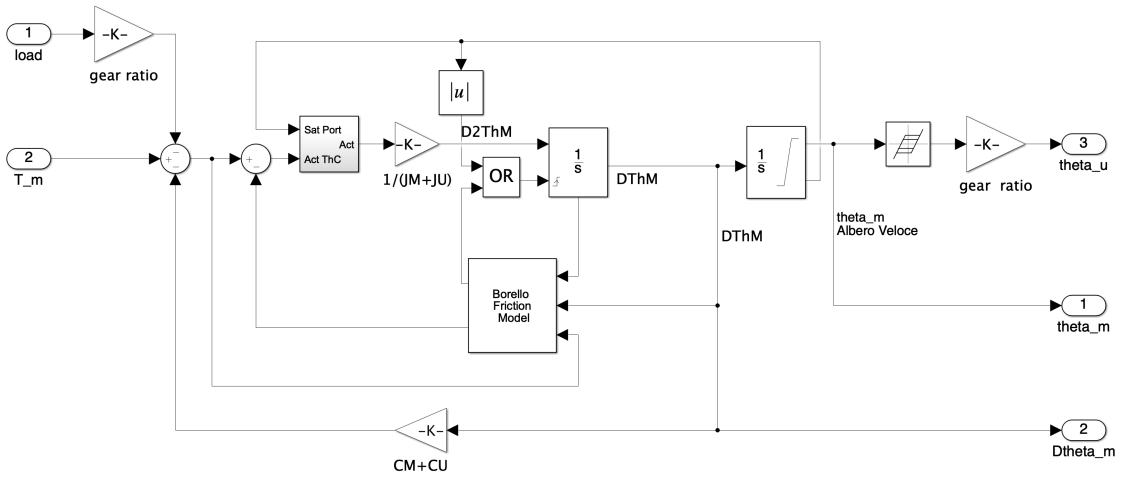


Figure 3.12: Transmission dynamical model

considered, including end-stops inside which the user is constrained to move and dry friction in the form of the Borello friction model, a more modern and robust implementation of the Coulomb friction model, which avoids limit cycles and with a small calculation footprint.

The model then estimates the user position accounting for backlash effect and the transmission ratio between motor and user, that is between fast and slow shaft. Finally, the rotor electric position is also evaluated, as described before (properly accounting for the number of pole pairs) in order to be fed back to the EM model, where is then used to evaluate the counter electromotive force and the phase switching logic.

3.5 Longitudinal dynamics block

The last block composing the Simulink model is the longitudinal dynamics block, representing the dynamic response of the entire aircraft considered (fig. 3.13).

The block takes as inputs both the initial conditions ($F16.x0(5)$) and the surface deflection previously calculated, δ_e.

The initial conditions are mandatory since the model is discretized in a space-state model, so the derivatives of the variables are equaled to a matrix product between a coefficient matrix and the state vector itself.

Finally, the block outputs are the aerodynamic variables of interest, that are speed (V), angle of attack (α), Euler body pitch angle (θ), pitch rate (q), surface deflection (δ_e) and hinge moment.

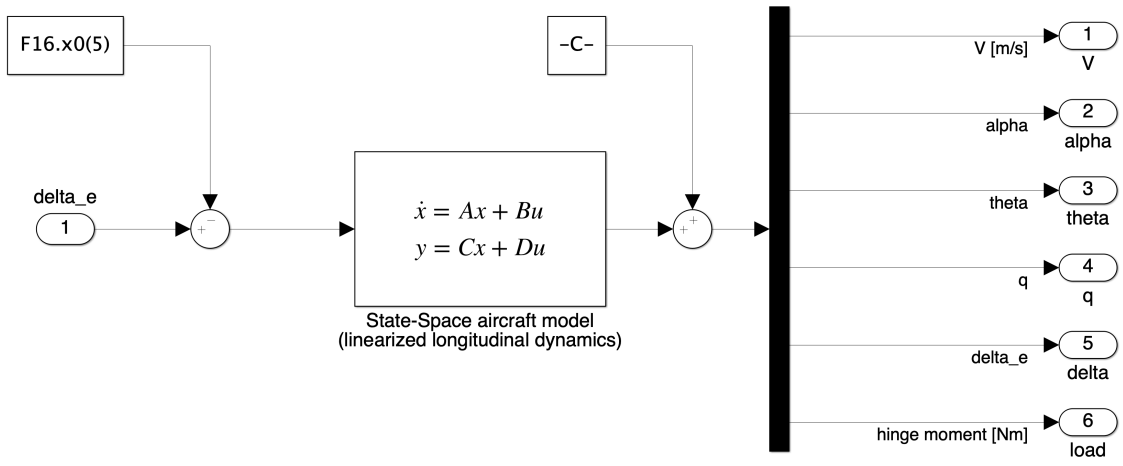


Figure 3.13: Longitudinal dynamics block

Chapter 4

Fault modes and modeling

As stated in the introduction, EMAs are a relatively new technology compared to hydromechanical or electrohydraulic actuators, so their use in aerospace is, at this time, confined to non-safety critical applications, since reliable fault statististics is not yet available and combined faults condition response is still not fully understood.

In modern aircrafts, EMAs are primarily used in trim-tab actuations, spoilers speedbrakes actuation. On some *more-electric* aircrafts, most notbaly Boeing 787, the use has been extended to slats and flaps operation. Typical configurations is based on a centralized power source, connected via kinematics chains to each user, ensuring symmetrical actuation of the command. In particular, a reducer is almost inevitably present since compact sized BLDCs delivers too little torque to directly actuate an aerodynamic surface, but on the flip side they usually have the most efficient operation zone at very high speed (thousands of RPMs); BLDCs are used in place of traditional hydraulic motors.

Military aircrafts follow the same trends, using EMAs in non-safety critical operations. On some current 5^{th} generation aircrafts, e.g. F-22 Raptor, F-35 Lightining II and F/A-18E/F/G Super Hornet, electrohydrostatic actuators are used in place of more traditional hydromechanical actuators in order to save weight and reduce the impact of a damaged hydraulic line. At the same time, much effort is put in the development of EMAs as utility actuators, replacing EHAs in roles like landing gear actuation, refueling doors and bomb bay doors operation on future 6^{th} generation aircrafts, and possibly their use as primary controls actuation giving enhanced performance compared to more traditional counterparts [9].

4.1 EMAs fault modes

EMAs fault modes can be broadly divided in four categories:

1. Motor faults: some of the most common typologies of failure in EMAs. BLDCs typically operated at very high speed, inducing vibrations on rotor bearings and inertial loads, due to high rotational speed, on the permanent magnets rotor core interface. Furthermore, heat management is another issue, due to the compact motor size and the absence of active cooling elements, in contrast tot hydraulic-based systems. Excessive heat can lead to several pathological conditions, including degradation of stator coil insulation and rotor permanent magnet demagnetization if temperature rises over the relative Curie temperature.

2. Mechanical faults: mostly affecting reducers and transmissions, these faults are generally of important entity considering the environment in which EMAs are operating is very demanding; proper monitoring is then required. Faults of this nature have several causes, including excessive external loads, lubrication deficiencies and unidentified manufacturing defects.

3. Sensor faults: since EMAs operation is based on feedbacks, even for motor control, sensor faults are generally of critical severity. Such faults often leads to dynamic characteristics alteration or even actuator loss. Classification has three types of sensor faults: *bias*, *drift* or *scaling*.

4. Electrical and electronic faults: they share many similarities with faults found in other similar aerospace system, including both power or control segment. Main causes are found in overheating, short circuits, possibly caused by particles contamination, overcurrents, overvoltages, vibration or out-of-specifications environment.

In tables 4.1, 4.2 and 4.3, based on [6], the most common EMAs failure modes are described.

The faults implemented in the work will now be described in detail; in particular, this work focuses on the effect of partial electrical shorts of stator coils and rotor eccentricity.

Component	Fault	Failure	Probability	Criticality	Model type
Screw	spalling	vibrations, metal flakes	5	3	Hybrid
	wear/backlash	backlash	7	3	Data
Nut	spalling	vibrations, metal flakes	5	3	Hybrid
	backlash	seizure/disintegration	7	3	Data
	degraded operation	seizure/disintegration	3	5	Data
	binding/sticking	seizure/disintegration	3	3	Data
	bent/dented/warped	seizure/disintegration	1	5	Data
Ball returns	jam	seizure/disintegration	5	8	Hybrid
Bearings	spalling	vibrations/metal flakes	5	3	Hybrid
	binding/sticking	seizure/disintegration	2	4	Data
	corroded	vibrations, metal flakes	2	5	Hybrid
	backlash	vibration, disintegration	7	3	Data
Piston	cracks	structural failure	1	10	Data
Dynamic seals	wear	structural failure	4	6	Physics
	structural failure	same	3	8	Data
Static seals	structural failure	same	2	8	Data
Balls	spalling	vibrations, metal flakes	5	3	Hybrid
	wear	backlash	7	5	Hybrid
Mountings	crack	complete failure	1	7	Data
Lubricant	contamination	seizure/disintegration	8	5	Data
	chemical breakdown	seizure/disintegration	4	5	Physics
	run-dry	seizure/disintegration	3	10	Hybrid

Table 4.1: Mechanical and structural fault modes

Component	Fault	Failure	Probability	Criticality	Model type
Connectors	degraded operation	disconnection	5	6	Data
	intermittent contact	disconnection	3	7	Data
Stator	stator coil fails open	same	4	4	Physics
	insulation deterioration	short circuit	5	5	Data
Resolver	coil fails open	same	4	10	Physics
	intermittent coil failures	permanent coil failures	5	7	Data
	insulation deterioration	short circuit	5	7	Data
Rotor/magnets	bond deterioration	magnet separation	2	10	Data
	rotor eccentricity	bearing failure	3	6	Physics

Table 4.2: Motor fault modes

Component	Fault	Failure	Probability	Criticality	Model type
Controller capacitors	dielectric breakdown	short/open circuit	4	8	Hybrid
Controller transistors	dielectric breakdown	short/open circuit	4	8	Hybrid
Wiring	short circuit	same	5	10	Hybrid
	open circuit	same	5	10	Hybrid
	insulation deterioration	short/open circuit	5	8	Data
Solder joints	intermittent contact	disconnection	5	8	Hybrid
Power supply	short circuit	same	5	10	Hybrid
	open circuit	same	5	10	Hybrid
	intermittent performance	short/open circuit	5	8	Data
	thermal runaway	dielectric breakdown	6	10	Hybrid

Table 4.3: Electrical and electronic fault modes

4.2 Short circuit fault

Short circuit events are generally caused by operating the motor over its designed maximum temperature. Heat dissipated via Joule effect may damage the polymeric insulation of the wires making up the stator coils windings. When the insulations is broken, it is possible that two contiguous spires may form a short circuit. This event is not desirable, since it tends to propagate and amplify the short circuit damage. In fact, in case of partial short, motor inductance and resistance decrease, and consequently an increase of the current passing through the coil, and consequently heat dissipated, is observed. The final outcome of the progressive advancement of the failure is coil total short and motor loss.

It is possible to identify several short circuit types, based on the components involved:

- coil-coil, short between windings of the same coil;
- phase-phase, between windings of different coils (phases);
- phase-ground, between one of the windings and the stator iron core.

In general, coil-coil are the initial manifestation of short circuit faults, eventually propagating to other modes when additional elements are involved, due to the progressive nature of the fault; when a phase-phase or phase-ground short circuit is observed, the motor is in total breakdown condition. The aim of prognostics is then to locate and identify short circuits in low-entity coil-coil conditions, before the rapid propagation to nearby components.

4.2.1 Short circuit fault implementation

Since the model used in the analysis models individual and with high-fidelity each of the three phases of the considered BLDC motor, it is relatively ease to implement such fault.

The effects of a partial coil-coil short are, as mentioned above, a reduction in coil inductance and resistance, proportional to the number of shorted windings, while at the same time reducing CEMF and torque coefficients. The variation of k_{cemf} can be approximated by:

$$k_{cemf} = G_M = \frac{\partial \phi}{\partial \theta_m} = nA \frac{\partial}{\partial \theta_m} \left(\int_A \mathbf{B} \cdot \mathbf{n} \, dS \right)$$

where n is the number of windings making up the coil, A is the total winding area and $\mathbf{B}$ is the magnetic flux density of the rotor.

Now, defining N_i as the normalized value of shorted coil windings in respect to n, the total number of coil windings, that is the fraction of shorted windings, it is possible to define:

$$R_i = N_i R$$
$$L_i = N_i^2 L$$
$$k_e^i = N_i k_e$$

where R_i, L_i and k_e^i are respectively resistance, inductance and normalized CEMF coefficient of the i-th phase and R, L and k_e are the nominal values, referring to a zero-fault condition.

4.3 Rotor eccentricity fault

Eccentricity is defined as an alignment error between a rotor rotation axis and its axis of symmetry. Two different types of eccentricities can be defined: static and dynamic. Static eccentricity refers to a condition where the rotor axis is not coincident with the rotor symmetry axis, while dynamic eccentricity refers to a mismatch between rotor rotation axis and its rotational inertia axis. Static eccentricity leads to irregularities in the air gap surrounding the rotor, making it different for each of the phases, while dynamic eccentricity produces vibrations due to the non-symmetrical distribution of masses around the rotation axis, causing premature bearing wear and irregular force output.

In this work only static eccentricity will be considered, since this kind is the only one strongly impacting motor, and thus actuator, performance, while dynamic eccentricity effects can only be evaluated by direct measurement of the vibrations of the components.

The rotor is assumed to be a rigid body, so no deformation is considered. With respect to the stator, the air gap is fixed and will not be affected by the rotor angle. A depiction is presented in fig. 4.1.

Two different circles can be individuated, describing rotor and stator:

$$x^2 + y^2 = R_r^2$$
$$(x - x_0)^2 + y^2 = R_s^2$$

In polar coordinates the equations are:

$$\rho = R_r$$
$$\rho^2 - 2\rho\cos\theta + x_0^2 = R_s^2$$

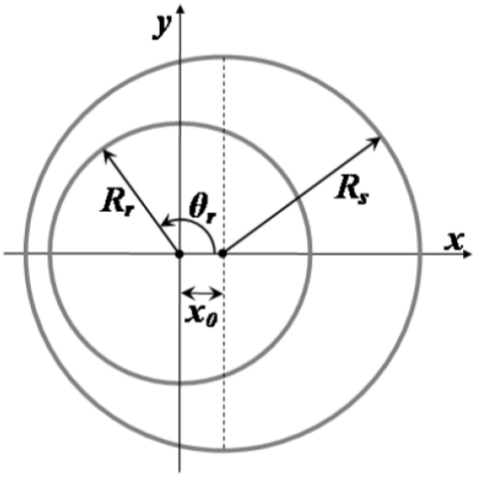


Figure 4.1: Air gap considering static eccentricity [6]

Air gap, g, is defined as:

$$\begin{aligned} g =& x_0 \cos\theta + R_s\sqrt{1 - \frac{{x_0}^2}{R_s}\sin^2\theta} - R_r \simeq \\ &\simeq x_0\cos\theta + R_s\left[1 - \frac{1}{2}\left(\frac{{x_0}^2}{R_2}\right)\sin^2\theta\right] - R_r \simeq \\ &\simeq x_0\cos\theta + g_0 \end{aligned}$$

where $g_0 = R_s - R_r$ is the air gap considered during nominal conditions. In other words, the air gap is function of the angle θ , and it is defined as the difference of distances ρ of rotor and stator.

A new parameter can be introduced, that is the *eccentricity parameter*, defined as:

$$\xi = \frac{x_0}{g_0}$$

so that the air gap can be expressed as:

$$g \simeq g_0(1 + \xi\cos\theta)$$

Now, the magnetic flux Φ, calculated solving the circuit between two consecutive rotor poles (fig. 4.2), is:

$$\Phi = \frac{F_m}{\frac{g(\theta_1)}{\mu_0 S} + \frac{g\left(\theta_1 + \frac{\pi}{P}\right)}{\mu_0 S}} = \frac{F_m \mu_0 S}{g(\theta_1) + g\left(\frac{\pi}{P}\right)}$$

where F_m is the rotor permanent magnet MMF (magneto-motive force), P is the number of poles and S is the surface crossed by the magnetic flux.

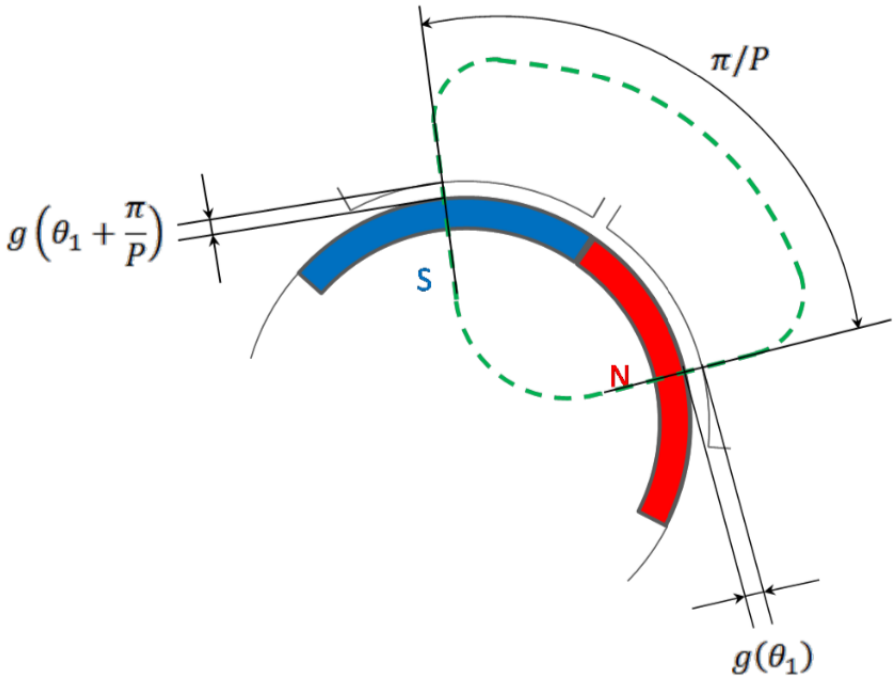


Figure 4.2: Air gap magnetic circuit approximation [6]

Expressing Φ as function of the nominal flux Φ_0 and of the nominal air gap g_0, one can found that:

$$\Phi = \frac{2\Phi_0 g_0}{g(\theta_1) + g\left(\frac{\pi}{P}\right)}$$

thus implying that, since the air gap has a 2π periodicity, the CEMF coefficient, being the magnetic flux time derivative, is only affected if the number of pole pairs is greater than one; in case of a single pole pair one can obtain:

$$f(\theta_1) + g(\theta_1 + \pi) = 2g_0$$

4.3.1 Rotor eccentricity fault implementation

The effect of rotor static eccentricity can be modeled quite accurately and relatively easily, as shown by [4], without the use of electromagentic FEM analysis; this result can be achieved by modulation of the CEMF coefficient as function

of both rotor position and eccentricity:

$$k^i_{cemf} = k^i_e(\theta_m) \cdot \left(1 + \xi \cos\left(\theta_m + \frac{2(i-1)}{3}\pi\right)\right)$$

where $k^i_e(\theta_m)$ is the non-faulty condition trapezodial-shaped coefficient, according to [31]. As described in the previous chapter, the eccentricity fault is implemented in the subsystem *BLDC Motor Electromechanical model*; this allows to accurately represent the motor behavior numerically simulating a magnetic interaction between the two main motor components, stator and rotor.

4.4 Noise fault

One definition for noise is *a general term for unwanted (and, in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing, or conversion* [30]. Noise is a physical phenomenon that is impossible to eliminate, albeit various means exist to reduce its effect. Noise can have different origins, like background environmental noise, thermal-induced noise, RF pollution and component-induced noise, since every component doesn't behave as an ideal component.

Noise can lead to error in signal transmission, so noise management is an essential element in every electronic circuit. In particular, processing noise can lead to false results not accurately representing system behavior; this can lead to improper commands imparted by the control system that might lead to reduced performance, unpredictable responses or even catastrophic failure; consequently, proper signal filtering, being it software or hardware, analogical or digital, is mandatory in complex electronic circuits.

A broad noise classification can be made based on the underlying phenomena generating it. One of such classification is:

1. *Electromagnetic noise*: generated by moving electrical charges, i.e. electric currents. Depending on the oscillating frequency, the whole EM spectrum can be covered, even tough almost exclusively present in the radio waves range. EM noise is generated by virtually any electric and electronic device, so a steady background noise is almost always present.

2. *Electrostatic noise*: generated by the accumulation of charges, i.e. in presence of high voltages, with or without current flow. One of the most common causes is the use of fluorescent lights.

3. *Channel noise*: encompassing a wide array of phenomena (distortions, fading, echoing), is caused by the non-ideality of components present in a transmission or reception line. Depending on the operating frequency, the environment can have a great impact on channel noise, e.g. microwave transmission adopted in mobile phones.

4. *Processing noise*: generated by, as the name implies, the either analogical or digital processing of any signal, e.g. discretization of a continuous signal or data packets alteration during encoding.

5. *Acoustic noise*: produced by any moving mechanical system, it is caused by imperfection in the elements, producing random vibrations; virtually every everyday machine produces acoustical noise, so if a high degree of accuracy is needed, proper measure must be enforce in order to reduce acoustic background noise.

As previously mentioned, noise in one of its form is produced by every electric, electronic or mechanical system. In EMAs case, the main noise generator are the electric motor (BLDC), producing acoustic noise (vibrations, audible sounds) and electric noise (rotating magnets inducing voltages in surrounding conductors) and the power electronics, producing EM noise, partly due to the PWM logic used for phase switching. Moreover, environment noise, being mechanical, electrical or else, has to be accounted for, e.g. lighting, communication and navigation equipment, radars or even ECM jammer in military implementations.

Another noise classification can be made considering the frequency band covered and the temporal characteristics. As reported in [33], the following categorization can be performed:

1. *Narrowbad noise*: characterized by limited spectral band, it can be caused by switching (e.g. PWM high frequency noise, around the switching frequency) or constant frequency transmission (e.g. electrical lines 'humming' at 50/60 Hz).

2. *White noise*: totally random noise with uniform power density distribution in respect to frequency (fig. 4.3).

3. *Band-limited white noise*: similar to white-noise, but spanning a finite frequency band.

4. *Colored noise*: random noise with a non-uniform power density distribution, unlike white noise. Examples are red, gray or pink noises.

5. *Impulsive noise*: it consists of relatively short-duration pulses, with random amplitude and duration.

6. *Transient noise pulses*: they consist of relatively long duration pulses, as opposed to impulsive noise, again with random duration and amplitude.

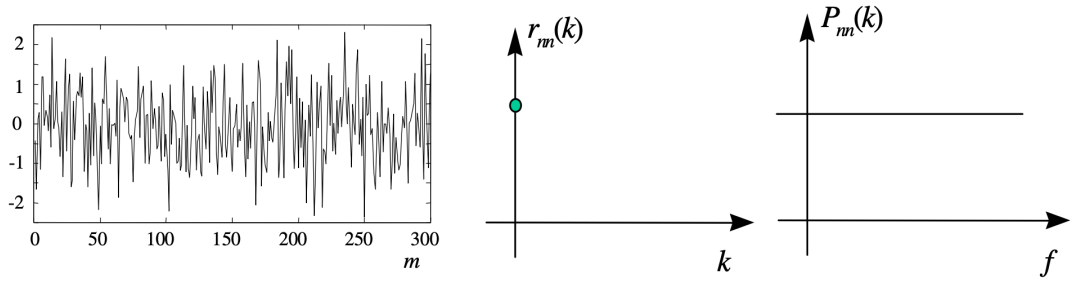


Figure 4.3: White noise (left), autocorrelation (center) and power spectrum (right) [33]

Ideal white noise is a unrealistic idealization, since its range would span from 0 to ∞ frequency, with a constant power density; this implies an infinite power.

Autocorrelation of continuous white noise, with zero mean and σ^2 variance is a particular delta function [33], assuming the form:

$$r_{nn}(\tau) = E[N(t)N(t+\tau)] = \sigma^2\delta(\tau)$$

while the power spectrum of white noise can be obtained by means of Fourier analysis applied to the precious equation, yielding:

$$P_{nn}(f) = \int_0^\infty r_{nn}(t)e^{-i2\pi ft}\ dt = \sigma^2$$

Last equation shows, as previously stated, a constant power distribution along all frequencies, thus infinite power.

4.4.1 Noise fault implementation

In the model, a band-limited white noise is introduced, since, as previously proved, white noise is related to infinite power. To obtain accurate results, the highest noise frequency is set at less than half of the sampling rate. The spectrum of band-limited noise, having half-bandwidth B and centered on frequency f_0,

can be expressed as:

$$P_{nn}(f) = \begin{cases} \sigma^2, & \text{if } |f - f_0| < B \\ 0, & \text{if } |f - f_0| \geq B \end{cases}$$

This implies that total power of band-limited white noise is equal to $2B\sigma^2$. The autocorrelation function is, in case of discrete-time, equal to:

$$r_{nn}(T_s k) = 2B\sigma^2 \frac{\sin(2\pi B T_s k)}{2\pi B T_s k}$$

where T_s is the sampling period, generally assumed to be equal to unity for simplicity sake.

In detail, noise is added to the reference current signal, that means downstream of the most sensitive electronic subsystems. Such approach allows to simulate the final effect of noise introduced by the electronic on the control signal used to drive power electronics (fig. 4.4).

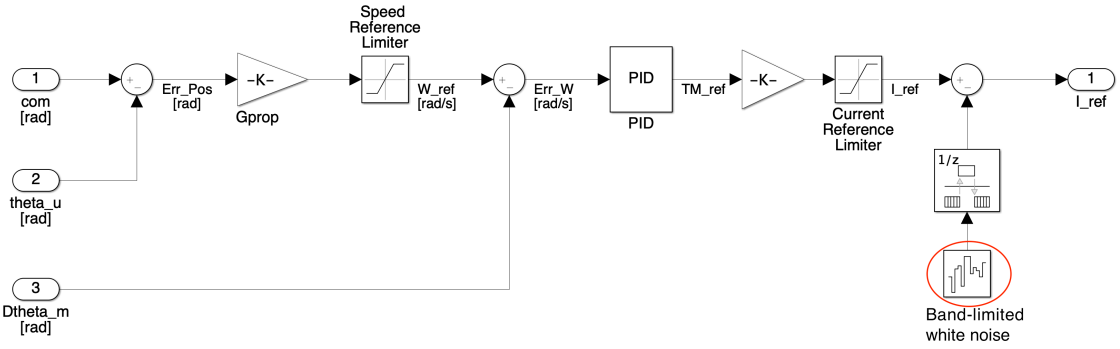


Figure 4.4: EM noise implementation in Simulink model

4.5 Friction fault

Dry friction is affected by mechanical wearing, increasing with time. The progressive deterioration of surface finish of components in mutual contact leads to an increase of the friction coefficient, and thus of the reaction forces present in the system; this implies higher motor currents, that is higher torques, in order to compensate for the increased forces that have to be overcome. If friction

deterioration is neglected, a number of dynamical condition can not be modeled, e.g. a possible actuator jam might not be correctly assessed, with disastrous consequences.

Many models have been proposed to model dry friction. One of the most accurate yet simple is Borello dry friction model [8], that will be adopted in the Simulink model. The model is an evolution of Coulomb's model, particularly apt for numerical simulations, allowing:

1. friction torque sign discrimination as function of velocity direction;
2. stick/dynamic friction evaluation, allowing different values for each condition;
3. sudden stop of moving part evaluation;
4. stick motion propagation in time;
5. sudden component starting;
6. ease of integration in end-stop model.

Mathematically, the model can be described as:

$$F_f = \begin{cases} F_{act}, & \dot{x} = 0 \wedge |F_{att}| \leq F_{sj} \\ F_{dj} \cdot sgn(F_{act}), & \dot{x} = 0 \wedge |F_{att}| > F_{sj} \\ F_{dj} \cdot sgn(\dot{x}), & \dot{x} \neq 0 \end{cases}$$

where F_f is the calculated friction force, F_{act} is the active force applied to the component, F_{sj} and F_{dj} are the friction force in stick (static) and dynamic condition, respectively. The model uses a zero-crossing detection on velocity to identify direction reversal and component stoppage: if a component velocity changes sign, velocity is reset to zero in the following timestep to allow stick condition identification. In sequent timesteps, if the acting force is greater than the maximum stick force, the component will start moving again; otherwise, it will not move. Another advantage in using Borello's model is the avoidance of non physical parameters (e.g. Karnopp's model dead-band, Quinn's model hyperviscous coefficient) used for convergence.

4.5.1 Friction fault implementation

As previously stated, the Borello dry friction model has been implemented in the Simulink model (fig. 4.5) in order to correctly simulate friction phenomena present in reality.

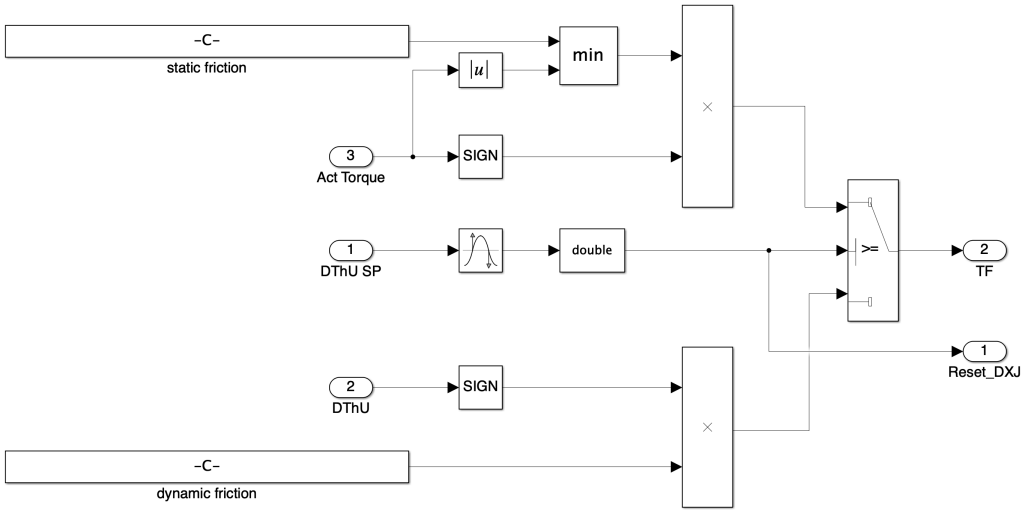


Figure 4.5: Borello dry friction model implementation in Simulink

It has to be noted that velocity signal used for zero-crossing detection must refer to the current timestep, in order to avoid an algebraic loop; this is achieved by using the integrator *State port*, used to reset the integrator itself; the state port signal is evaluated *before* the integration operation and has the same signal as the integrator output except when reset, where the value is equal to that the output would assume if wouldn't have been reset.

Chapter 5

Data analysis

In this chapter, the data elaboration and analysis will be described, starting from data faults list generation, to CEMF reconstruction from model variables to neural nets training and performance evaluation.

5.1 Faults generation

The first step in the analysis has been the generation of a suitable number of faults configuration, used to simulate the behavior of a damaged BLDC motor actuating an EMA. In order to get a suitable data sample, a relatively high number of faults combination has been generated and subsequently evaluated in the detailed Simulink EMA model described in previous chapters, logging relevant variables such as currents, voltages, angular position and speed.

The faults configuration have been modeled using a vector containing five different values:

$$k = [N_a, N_b, N_c, \xi, \phi]$$

where N_a, N_b, N_c represents the percentage of shorted coils relative to each phase, ξ is the nominal value of static eccentricity and ϕ is its relative phase.

To create a suitably diversified data sample, a modified latin hypercube approach has been used; the complete MATLAB code can be found in App. 1.1.

The regular latin hypercube uses a linear sampling for each of the N variables used in the sampling process, in this case, five. Since these faults conditions are relating to progressive faults, the linear modeling isn't the most well suited; instead, an exponential distribution, for each variable, has been achieved by

exponentiating the initial, linear distribution and then scaling it back to the $[0-1]$ interval, thus obtaining a non linear, exponentially biased variables distribution. It has to be noted that the scaling has only been carried out for the first four variables, that is N_a, N_b, N_c and ξ, since the static eccentricity phase has been kept as a linear distribution.

To obtain meaningful values, i.e. prognostics-compatible values, a maximum amount of cumulative damage has been imposed: to do so, the root sum squared value of the three phases damage (using the $rssq$ function) has to be less than an arbitrary threshold, otherwise the fault combination is discarded; simultaneously, the static eccentricity value has to be less than a different arbitrary threshold.

In this work, the two arbitrary thresholds are set as:

$$\begin{cases} n_{phases} = \sqrt{N_a^2 + N_b^2 + N_c^2} = 0.5 \\ n_{ecc} = \xi = 0.5 \end{cases}$$

This final check is done in order to avoid faults combination that would yield a damage too high to be still considered in the prognostics domain; in fact, too high faults values would lead to such degraded performance that diagnostics and fault isolation would be the primary disciplines involved.

5.2 CEMF reconstruction

After each simulation has been carried out using one faults vector, the CEMF along the whole mechanical rotation needs to be reconstructed based on the saved variables during the simulation, i.e. 3 currents, 3 voltages, mechanical angle and angular velocity, since direct measure is almost practically unfeasible. This prognostics approach, i.e. using the counter EMF has several advantages: the value is not significantly affected by either motor operation nor environmental conditions, and it is highly sensitive to electrical failure modes.

Assuming the motor is near nominal conditions, the following equation holds:

$$V_j - E_j = V_j - k_{cemf_j}\dot{\vartheta_m} = R_m i_j + L_m \frac{di_j}{dt}$$

where $j = A, B, C$. Solving for k_{cemf_j} one can obtain:

$$k_{cemf_j} = \frac{V_j(t) - R_m i_j(t) - L_m \dfrac{di_j(t))}{dt}}{\dot{\vartheta_m}(t)}$$

where R_m and L_m are known values, and all other quantities varies as time functions, sampled with a certain sampling frequency. As a result, a CEMF estimation is obtained at sampled time steps.

The following step is to correlate the calculated CEMF coefficient with the rotor angular position, thus obtaining $k_{cemf_j}(\dot{\vartheta_m})$. To do so, as a first approximation, ϑ_m vector is re-sampled and then average the CEMF coefficient values in proximity of each wanted angular position $\vartheta_{m,k}$, that is:

$$k_{cemf_j}(\vartheta_{m,k}) = \frac{1}{n}\sum_{l=1}^{n} k_{cemf_j}((\vartheta_{m,k} - \varepsilon) \leq \vartheta_m \leq (\vartheta_{m,k} + \varepsilon))$$

This calculation allows a reduction in numerical and sensor noise on the CEMF signals.

Finally, the equivalent single-phase CEMF coefficient is simply calculated as:

$$k_{cemf} = |k_{cemf_A}| + |k_{cemf_B}| + |k_{cemf_C}|$$

This value will then be sampled, as described in the following section and then used as input for a series of neural networks.

5.3 Sampling strategy

In order to feed the reconstructed CEMF data to a neural network, a sampling strategy has to be implemented. Since the CEMF curves have a certain regularity, corresponding to each electrical commutation, it was chosen not to define a custom function summarizing the CEMF behavior; instead, a direct sampling strategy has been implemented.

A sample CEMF reconstruction is reported in fig. 5.1, obtained with the following parameters $k = [0.3, 0.05, 0.12, 0.24, 0.4]$. The non constant nature of the CEMF constant can be identified. In particular, remembering that each commutation lasts 30° mechanical, and that the motor has 2 poles pairs, then each commutation happens every 15° electrical. This behavior can be exploited by using only 180° mechanical, corresponding to a full electrical revolution.

Around the end of the mechanical revolution, the reconstructed CEMF tends to diverge. This phenomenon can be explained considering that at low speeds the CEMF coefficient, being $k_{CEMF} = V/\omega$, will tend to have very high values. One possible mitigation technique is to simulate the system for a longer period of time and then discard all measures where the angular velocity is less than a certain value.

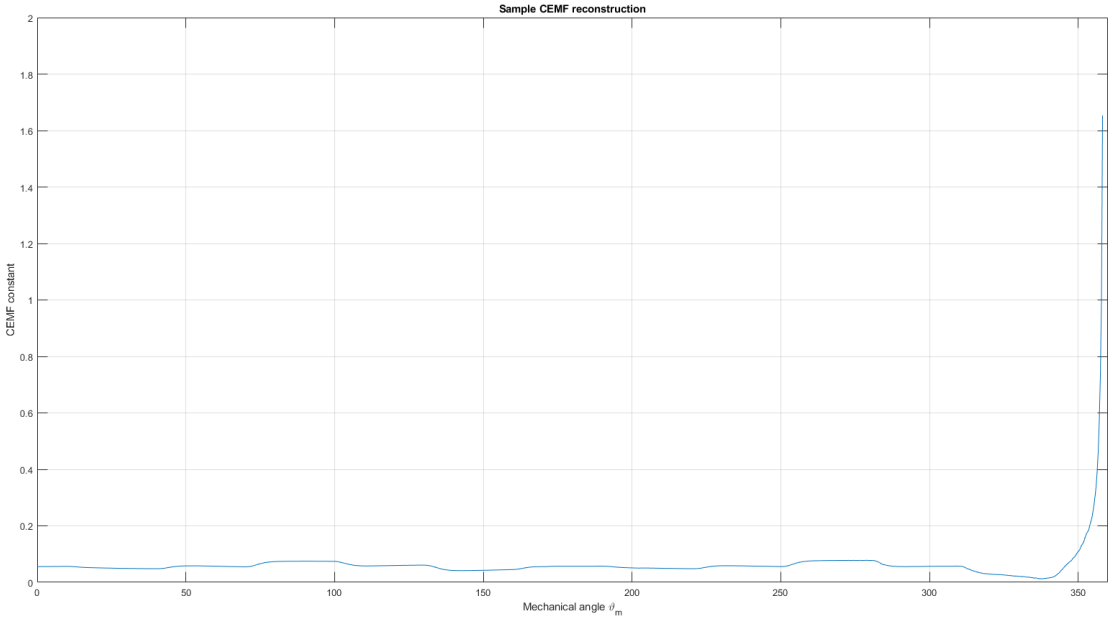


Figure 5.1: Sample CEMF reconstruction

Nonetheless, considering only the first 180° mechanical, corresponding to a full electric revolution, no divergence in CEMF constant is observed, so the problem is avoided altogether (fig. 5.2).

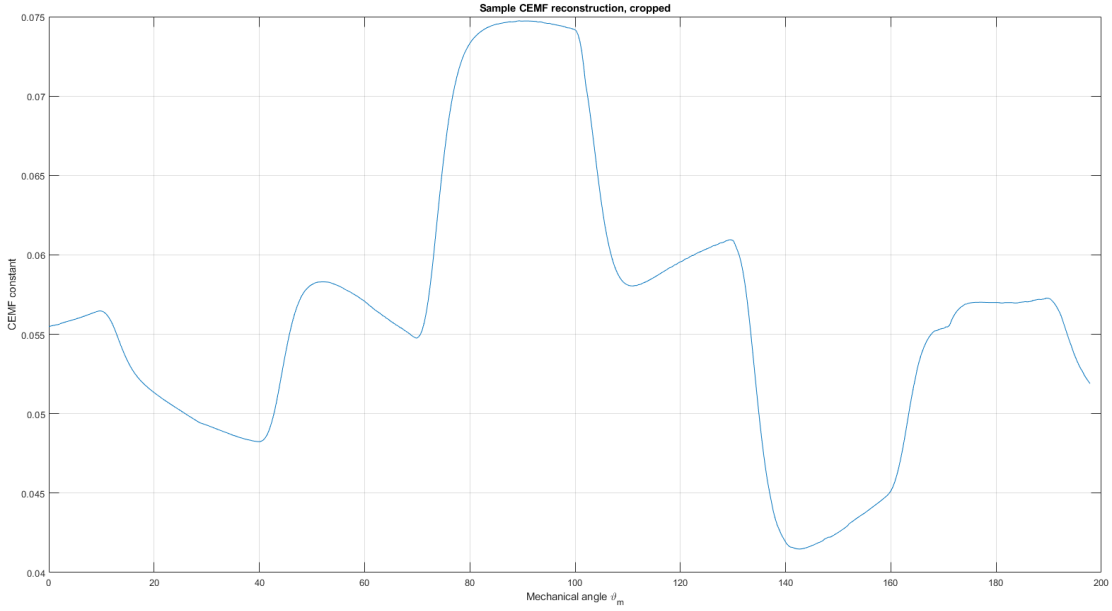


Figure 5.2: Sample CEMF reconstruction, $0° - 180°$ mechanical

As previously stated, the commutations positions are relatively regular in this model (ideally, commutations are instantaneous and always happen at the exact timestep) as visible in fig. 5.3, so this fact is exploited in order to get a sampling. Sampling is necessary to get values that will be then used as inputs in the neural networks.

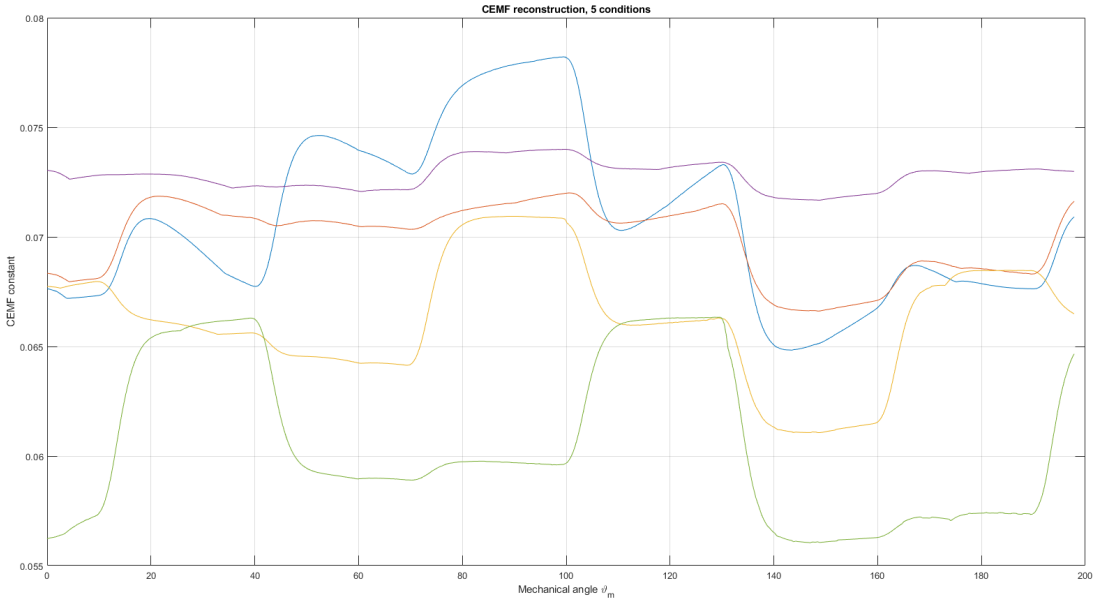


Figure 5.3: CEMF reconstruction, 5 conditions

The sampling strategy implemented has three different sampling modes: the first samples only the median point of each commutation, calculated based on the ideal commutation length; the second mode samples two points for each commutation, in order to have a better approximation of the commutation shape; finally, the third mode is a fusion of the previous two methods, i.e. sampling three points for each commutation, one being the median point.

In regard of the second sampling mode, a dedicated algorithm has been devised in order to evaluate the angular coefficient of each commutations. This is necessary since, for each different fault condition, the commutations shape varies, and so there is a need to determine the maximum sampling width, with respect to the midpoint of each commutation, that can be used. The bigger the sampling width, the better for global commutation approximation. This is true while the angular coefficient is relatively constant; this means the approximation is relatively accurate and local effects such as ripple are not affecting the sample. On the other hand, choosing a too big sampling width might mean falling in the commutation switch, totally falsifying the sampled results. As shown in figure 5.4,

the coefficients have a relatively constant value for δ values around 15 timesteps, so this value has been chosen. The complete script is reported in App. 1.2.

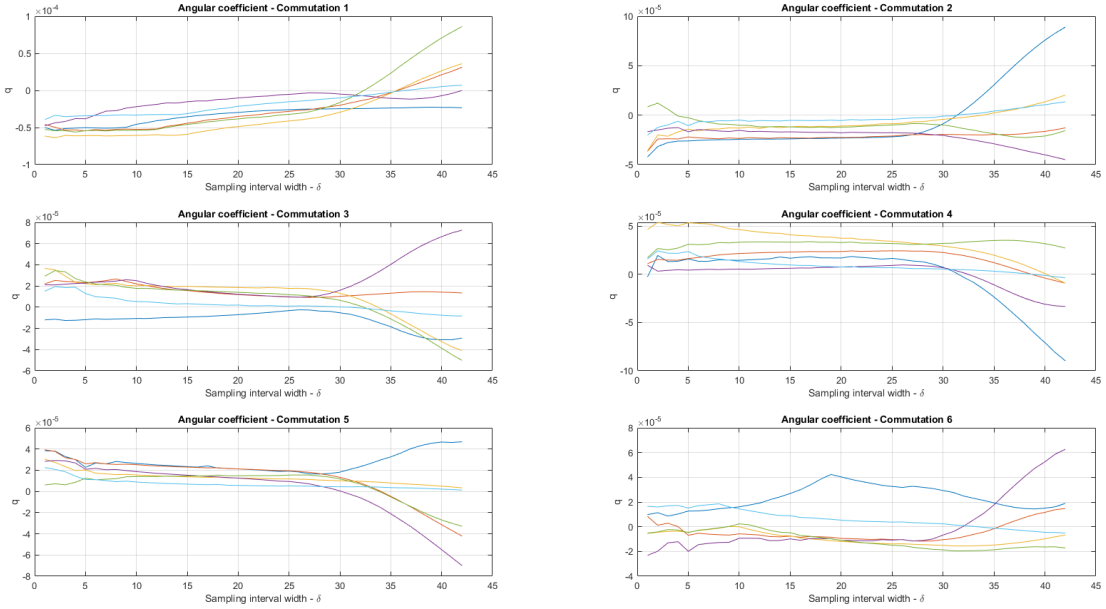


Figure 5.4: Angular coefficient in function of sampling interval

In fig. 5.5, the final sampling strategy has been reported for 5 different conditions; the red marks represent the theoretical commutations switch, while the green mark represent the median point of each commutation and finally the black marks represents the two additional sampled points.

Considering 6 different commutations, the number of points sampled is 6, 12 and 18, respectively choosing the first, second or third sampling strategy. All these conditions will be tested in opportune ANNs in order to evaluate eventual differences between each sampling strategy in complexity and accuracy.

5.4 ANN overview

Artificial neural networks (ANN) are a powerful machine learning tool. The name is a reflection on the principle of operation, resembling biological neural networks. The system 'learns' to perform different tasks based on examples, generally without specific programming rules.

The principal structure is based on a series of fundamental elements, called *neurons*, opportunely connected to others, depending on the architecture, similar to the biological synapses.

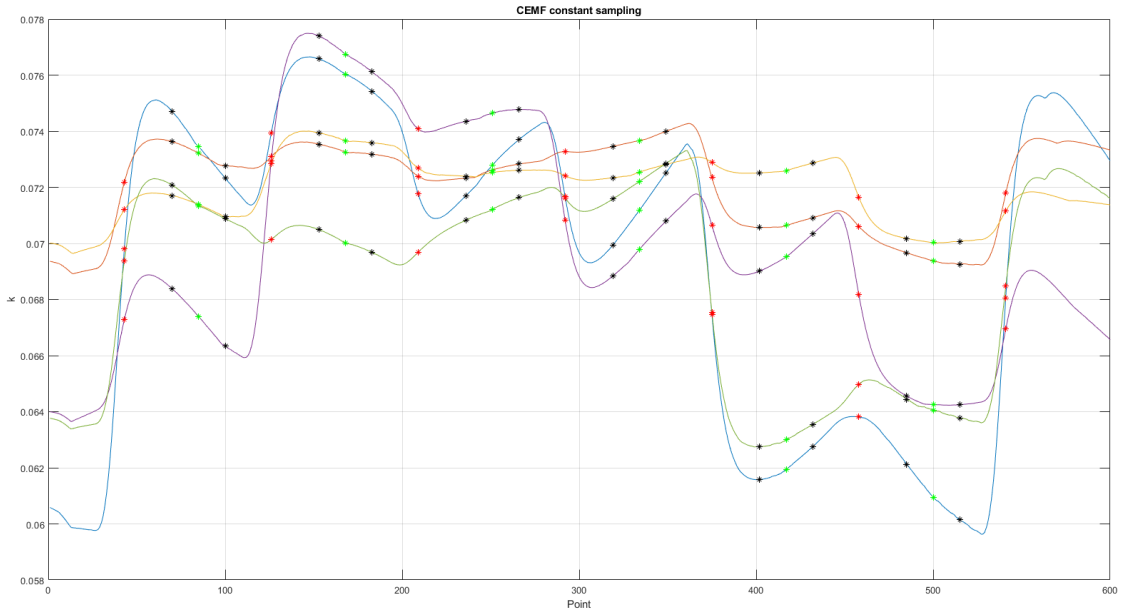


Figure 5.5: CEMF final sampling

5.4.1 Components

As previously mentioned, ANN are strongly inspired by biological neural networks. In the following pages the main elements constituting the ANN will be briefly described, based on [35]. The individual combination of neurons and connections uniquely defines the network's *topology*; two of the most common architectures are shown in figs. 5.6 and 5.7. The first topology describes a *feedforward* network, that is a network where each layer is fully connected to each neuron of the previous and following layers only; the other type, *cascadeforward* adds a cascade connection from each of the previous layers to the current layer.

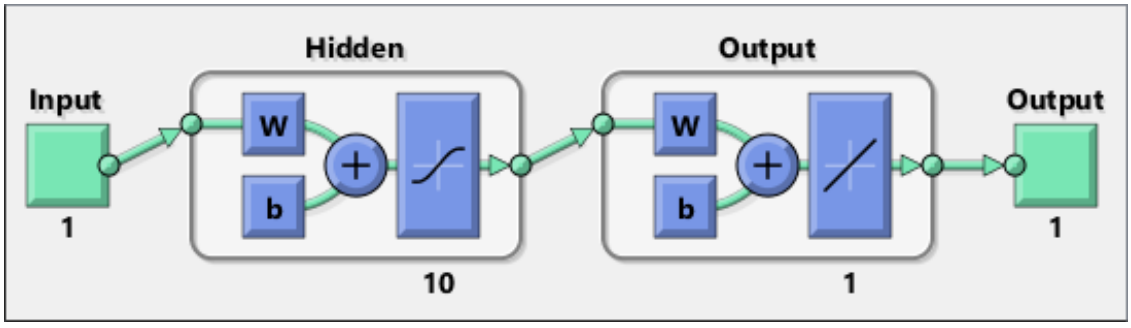


Figure 5.6: Feedforward neural network

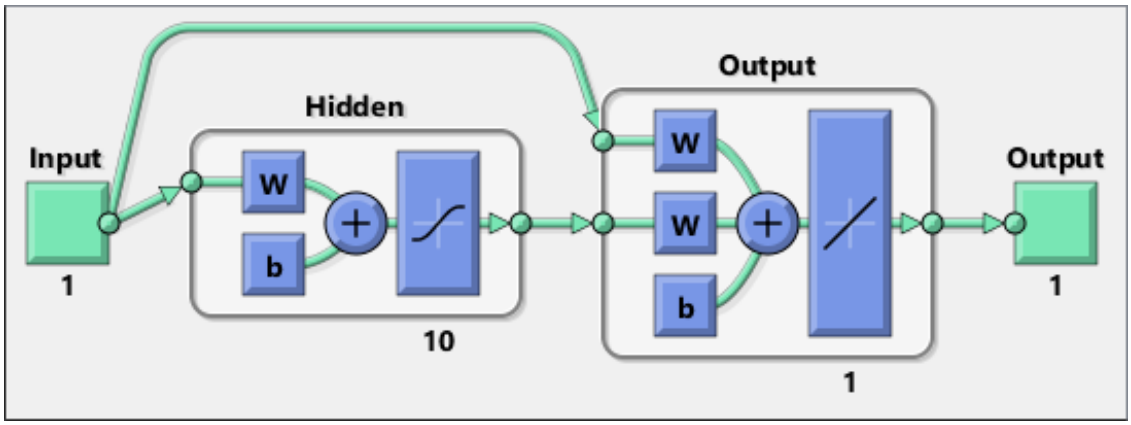


Figure 5.7: Cascadeforward neural network

Neurons

The fundamental element of an ANN is, as previously mentioned, the neuron. Each neuron receives one or more inputs, combine such values according to their internal state using a function called *activation function* and produces one output using an *output function*.

Layers

Depending on the type of layers, a network can be defined as *shallow* if there is only one hidden layer, while it is said to be *deep* otherwise. Shallow networks are very ease to setup and offer a good ease in learning and have proved effective at a multitude of task, while deep networks offers much more powerful capabilities. The drawback of deep networks is the high difficulty to achieve good training and general network complexity, tough its use is widespread in *convolutional neural networks*, where convolution filters are applied to the input data and possibly during the elaboration, between layers. CNNs can be extremely complex albeit have extremely powerful capabilities; in fact, CNNs are generally used in difficult task such as image and speech recognition.

Connections and weights

The network consists of a series of neurons opportunely connected, where each neuron's output is used as input in one or more neurons. Each connection has a *weight* associated, representing its relative importance.

Propagation function

The *propagation function* has the task of evaluating the neuron input based on the output of connected predecessor neurons as a weighted sum; optionally, a *bias* term can be added as an additional parameter to each neuron, both increasing network complexity and capabilities.

5.4.2 Learning

The learning process is fundamental before ANN operations. The process can be summarized as error minimization of samples outputs. Learning can be considered complete when additional observation do not improve the minimum error achievable by the network. Such error evaluation is typically achieved defining a *cost function* whose value is continuously evaluated during the learning process. If the cost function keeps on decreasing, the learning process continues; otherwise, the learning stops and the network is then fully trained. Cost function are numbers, so the difference between output and correct answer is small, when the error is low.

Learning rate

This parameter defines the size of the corrective steps the network has to take in order to reduce errors in each observation. Learning rate values defines the maximum potential ANN accuracy: in fact, lower level of learning rate offers higher potential accuracy, whilst increasing the learning time needed; on the other hand, higher learning rate values shortens training time at the expense of accuracy. Advanced refinements, especially the use of *adaptive learning* algorithms, introduces the concept of *momentum*, that is a weighting of the gradient and the last step, in order to avoid oscillation and increase stability. Momentum values are between 0 and 1, where the former emphasizes the gradient value while the latter only considers last change.

Cost function

The cost function is a measure of how precise the ANN in respect to the given training data sample and the wanted outputs. In general, cost functions can be dependent not only on the training sample inputs and the ANN outputs, but can also include weights and biases, such as:

$$C = C(W, B, S^r, E^r)$$

where W are the neural network's weights, B are ANN's biases, S^r is the input of a single training sample and E^r is the desired output relative to that training sample [1].

Backpropagation

Backpropagation is one of the methods that can be used to adjust connection weights, compensating for each error found during the learning process, basically dividing the error amount among the connections. In particular, backpropagation calculates the gradient of the cost function associated to a particular state in respect to the ANN's weights. The weight update can be done by means of different methods, e.g. stochastic gradient descent, 'weightless' networks [18], Extreme Learning Machines [19], training without backtracking [26].

5.4.3 Training paradigms

The three main learning paradigms are supervised learning, unsupervised learning and reinforcement learning, each corresponding to a particular task.

Supervised learning

Supervised learning uses a set of related inputs and outputs; the task is to produce a correct output given a particular input. In this paradigm, cost function is related to eliminating ANN's incorrect deductions. One of the most common cost function used is the *mean squared error* (MSE), whose objective is minimizing the average squared error between network's outputs and the desired (correct) inputs. These paradigm is effective when dealing with problem related to pattern recognition, i.e. classification and regression, and function fitting. Other notable uses are gestures and handwriting recognition, that is applications where sequential data are available.

In other words, the learning process can be viewed as learning with a 'teacher', in the form of a cost function, continuously monitoring and giving feedback about the network performance in order to improve solutions quality.

Unsupervised learning

In this learning paradigm, input data, cost function of the data d and ANN's output are given. Generally, an a priori assumption is made (e.g. model properties and parameters) and the definition of the cost function is made depending on the particular task to be solved. Unsupervised learning is applied to *estimation*

problem, including clustering, statistical distributions estimations, filtering and compression.

Reinforcement learning

The aim of this paradigm is to define the network's weight in order to perform actions minimizing long-term cost. At any point in time, the external agent performs an action, and the environment response is then measured and evaluated according to some arbitrary rules. It has to be noted that rules and long-term cost can only be estimated and not exactly calculated. Agent actions can uncover the cost of new, not yet tried actions or to use prior learning in order to have a quicker learning.

Reinforcment learning is mainly used in control problems, games and sequential decision making tasks. Furthermore, dynamic programming coupled with ANNs has been successfully applied to complex problems, i.e. vehicle routing [28], natural resource management [13] and medicine [14].

5.5 ANN implementation

The ANNs creation and training has been done in MATLAB, using the Deep Learning Toolbox. The Parallel Computing Toolbox has also been used in order to parallelize the training workload using multithreading; GPU acceleration has also been tested but has not been used since its limited training functions support.

Only feedforward networks have been used, even tough cascading network have been tried yielding very similar result to feedforward networks but with a greater complexity, given the higher number of connections, and thus weights. The full artificial neural network script is presented in App. 1.3.

5.5.1 Parameters choice

The main parameters set are:

```
net = feedforwardnet([5]);

net.trainFcn = 'trainlm'; % 'trainbr'
net.trainParam.showCommandLine = true;
net.trainParam.epochs = 1e3;
net.trainParam.goal = 1e-6;
```

```
net.trainParam.max_fail = 10;
net.trainParam.mu_max = 1e6;
net.performFcn = 'mse';

net.layers{:}.transferFcn = 'satlins';
```

Network topology

As previously described, only feedforward neural networks are used. The topology is created using the `feedforward([n])` function, creating a network object that has n neurons; n can be a vector, thus defining the number of neurons for each layer of the network.

Training function

The first parameter, `net.trainFcn` defines the type of training function used in the training process [22]. There are plenty of available training functions in the Deep Learning Toolbox, but in this study, two have been used.

`trainlm` The first function, `trainlm`, uses the Levenberg-Marquardt algorithm to perform backpropagation. In essence, the algorithm has the advantage of almost second-order training speed without calculating the Hessian matrix. In typical feedforward networks, the performance function can be expressed as the sum of squares; consequently, the Hessian matrix and the gradient can be approximated as:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} \qquad\qquad \mathbf{g} = \mathbf{J}^T\mathbf{e}$$

where $\mathbf{J}$ is the Jacobian matrix containing the derivatives of the network errors calculated in respect to the weight and biases, while $\mathbf{e}$ is a vector of network errors. The advantage of not directly calculating the Hessian matrix is that it is generally much harder to compute than the Jacobian.

The weights and biases updates follow a Newton-like form:

$$\mathbf{x_{k+1}} = \mathbf{x_k} - [\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e}$$

where μ is a scalar. In particular, when μ is zero, the method coincides with Newton's method; on the other hand, when μ is large, the method becomes a gradient descent algorithm with a small step size. The aim of the algorithm is to approach Newton's method as soon as possible, since Newton's method is

faster and more accurate in proximity of an error minimum. So the scalar μ is decreased after each successful step and is increased only when the tentative step would increase the performance function. The application of the algorithm to neural network is found at [17].

`traibr` The other function used for ANNs training was `traibr`, whose implementations uses Bayesian regularization to perform backpropagation. In short, Bayesian regularization minimizes a linear combination of squared errors and weights; it also modifies the linear combination so that at the end of training the resulting network has good generalization qualities [5].

Based on the Levenberg-Marquardt algorithm, it includes the following modifications:

$$\begin{aligned} jj &= jX \cdot jX \\ je &= jX \cdot e \\ dX &= -(jj + I \cdot \mu)/je \end{aligned}$$

where e represents the errors and I is the identity matrix [5].

The regularization process can help in those cases where the number of samples is not enough to train the network without overfitting; the method has also been used in this study since it wasn't known a priori if 3000 samples would be enough to achieve good accuracy without incurring in overfitting, even tough the number of samples is relatively high.

Training goal

The training goal, set using `net.trauParam.goal` has been arbitrarily set to 10^{-6}, since such value of performance would imply a very good accuracy.

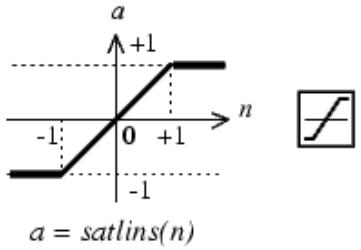
Performance function

As previously stated, the most common performance function used in relatively simple feedforward networks is mean squared error; the same is true for this work, where the performance function is set by `net.performFcn` and then set to `'mse'`.

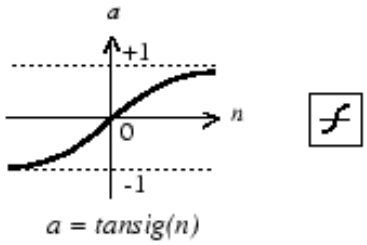
Transfer function

The Deep Learning toolbox offers several different transfer function, also called activation function. In this case, a saturated symmetrical linear transfer function (fig. 5.8a) has been used, even tough the hyperbolic tangent sigmoid function

(fig. 5.8b) has also been tested yielding marginally worse results. In any case, the definition is done by `net.layers{:}.transferFcn = 'satlins'`, that means that every neuron in every layers uses the saturated symmetrical linear transfer function.



(a) Symmetric saturating linear transfer function [29]



(b) Hyperbolic tangent sigmoid transfer function [20]

Figure 5.8: Commonly used transfer functions

5.6 Results

Different neural networks have been created depending on the number of inputs, that is on the sampling strategy used. In any case, the number of neurons has been varied between the input layer size (either 6, 12 or 18) and the output layer size (5 for every case). It has to be noted that all networks have one layer (shallow networks) except for the 18 inputs, 2 layers configuration.

5.6.1 One sample per commutation

This sampling strategy provides 6 inputs to the neural network, that is the median value for each commutation as previously described. The number of neurons has been set to either 5 or 6. In the following graph the performance has been evaluated for both cases, using both `trainlm` and `trainbr` methods.

As clearly visible form figure 5.9, there is a significant boost in performance just by adding a single neuron, even tough the performance score is still high (i.e. low accuracy). The `trainbr` achieves significantly better performance with 5 neurons, while the difference is negligible considering 6 neurons.

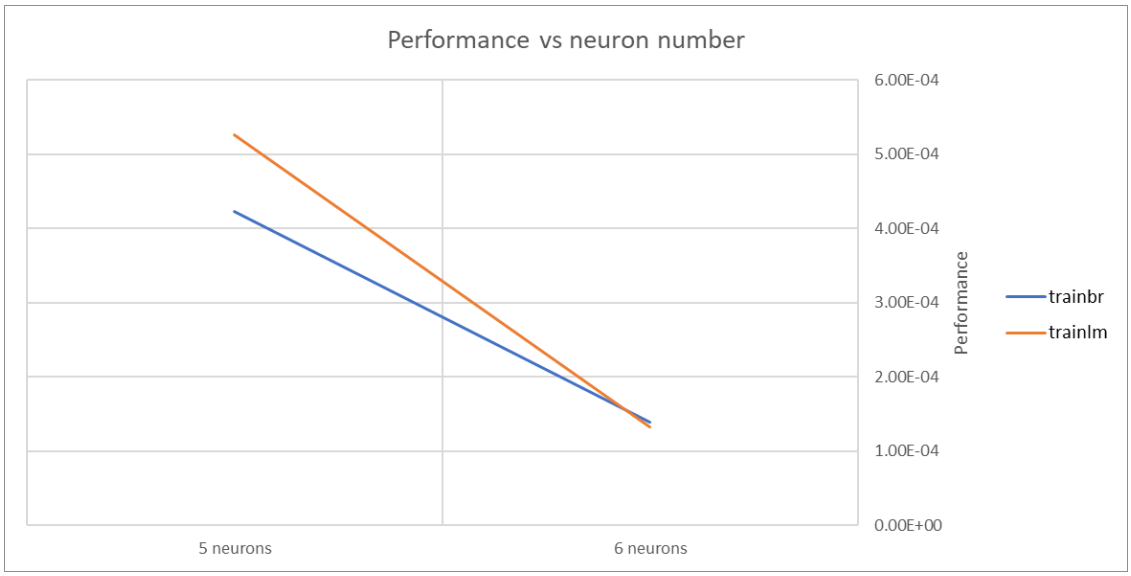


Figure 5.9: 6 inputs neural networks performance

5.6.2 Two samples per commutation

In this case, the number of inputs is increased to two per commutation, that is 12 in total. The neural networks have now increased complexity but also achieve, globally, a better performance score. For this condition, 6, 9 and 11 neurons networks have been considered.

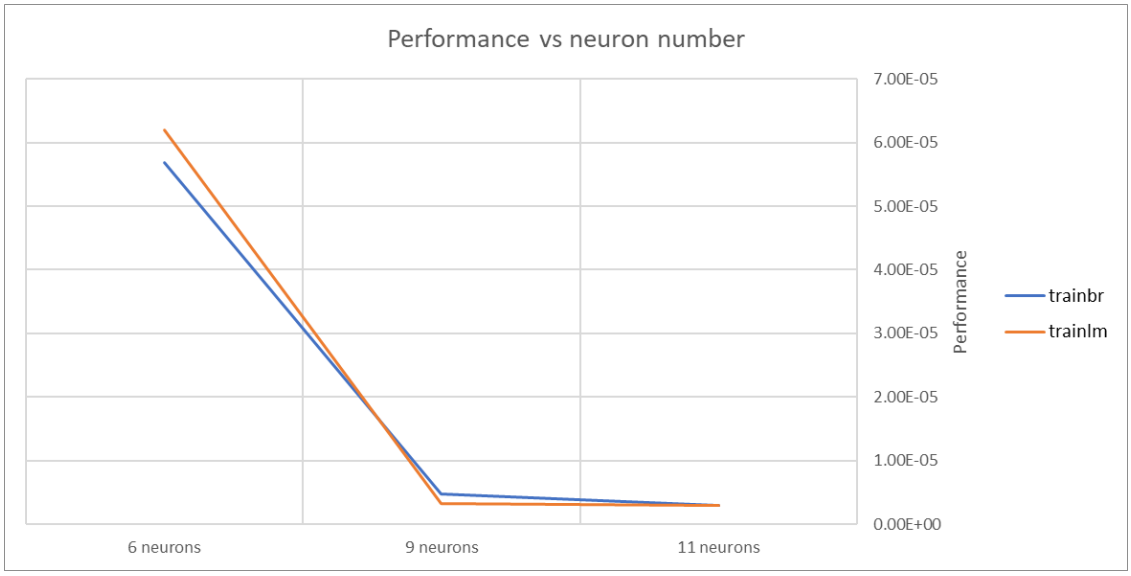


Figure 5.10: 12 inputs neural networks performance

As expected, the performance score (lower is better) decreases with the increase of neurons to value less than 10^{-5} in the case of 9 or 11 neurons (fig. 5.10), thus making much more accurate predictions compared to the 1 input per commutation architecture. The difference between the two training algorithm is marginal; considering 6 neurons, `trainbr` is superior, while the opposite is true when considering 9 neurons. The difference is basically negligible in the 11 neurons case.

5.6.3 Three samples per commutation

This is the most complex of the configuration analyzed thus far, having now three samples per commutation, thus 18 total inputs. The number of neurons has been set to 8, 10, 12, 14, 16 and 18 neurons.

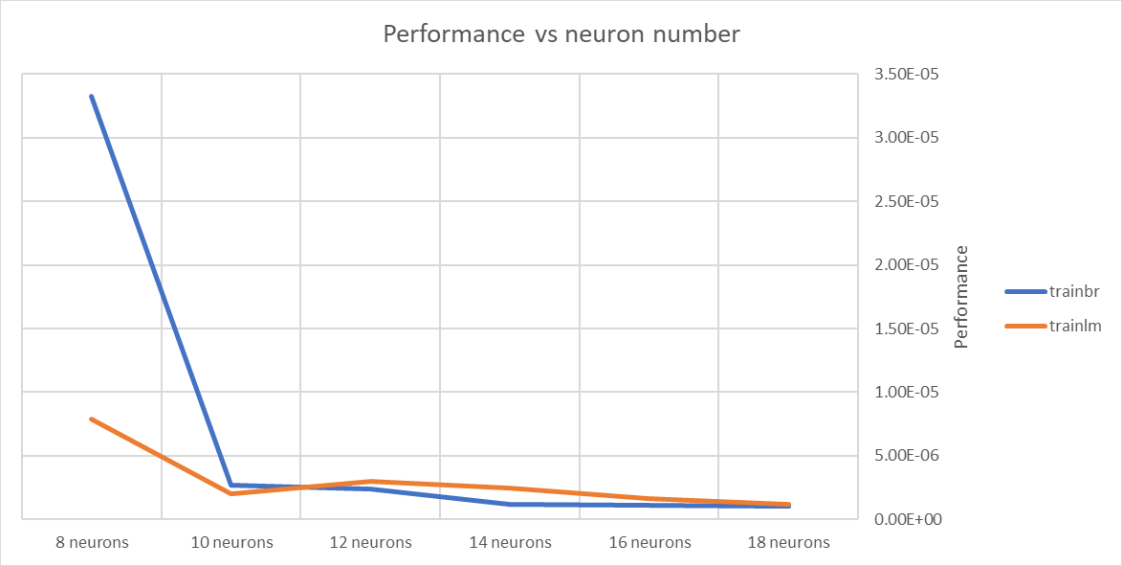


Figure 5.11: 18 inputs neural networks performance

Continuing the established trend, an increase in neuron numbers implies a decrease in performance score, meaning better and more accurate predictions. This time, the best score achieved was almost as close as the target score, 10^{-6}, as visible in fig. 5.11. Opposite to the previous cases, the `trainlm` algorithm performs better with fewer neurons, while the contrary is true for higher complexity networks.

5.6.4 Three samples per commutation, two layers

This configuration is different from the other analyzed since it is a deep network, having two layers in feedforward configuration.

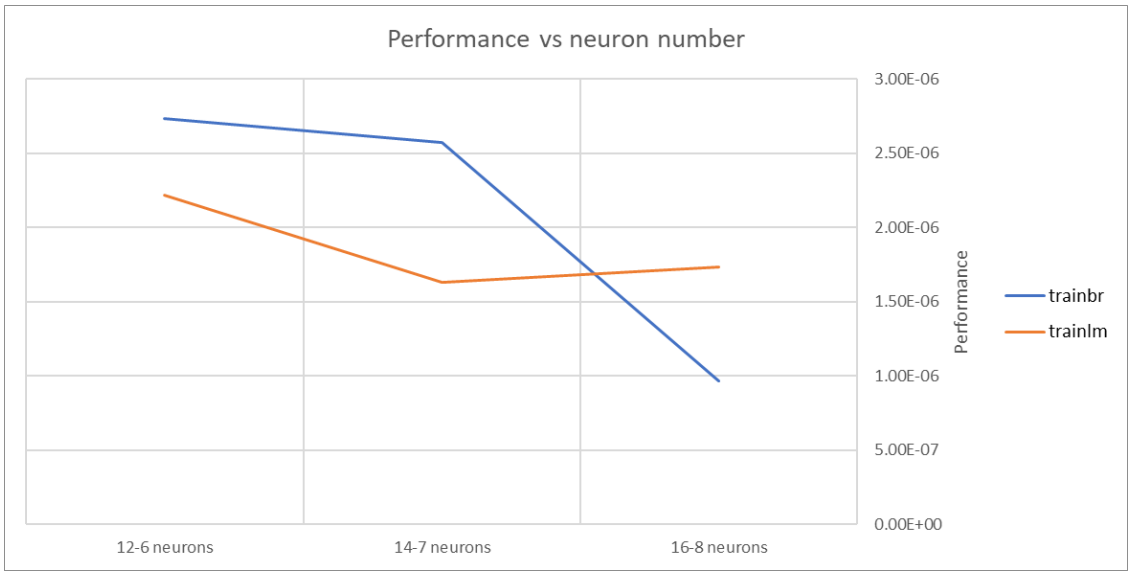


Figure 5.12: 18 inputs neural networks, 2 layers performance

The achieved score is not always better compared to the previous analysis. This means that an increase in the number of layers is not always associated with an increase in the neural network performance. It has to be noted that the increased complexity of the network, both in terms of neurons increase and connection increase has resulted in much longer training time, about a three-fold increase when compared to the most expensive, 1-layer configuration.

The only configuration capable of achieving score better than the shallow configuration is the 16-8 network using `trainbr` algorithm. In this case, the performance score has been less than the set 10^{-6}, thus achieving the best score of all the considered configurations.

Chapter 6

Conclusions

In this work, the application of artificial neural networks for prognostic purposes of electromechanical actuators for aerospace applications has been investigated.

After an initial description of the EMAs, the detailed Simulink model used in the analysis has been described, followed by a summary of the most common failure modes of such systems. The work focused only on particular motor progressive faults, i.e. partial electrical short-circuit and static eccentricity faults.

The process used to evaluate the performance of ANNs has been described in detail in chapter 5 and can be summarized as such: initially, an algorithm generating a vector of faults has been used to create a matrix describing three thousands faults combination between arbitrarily chosen thresholds. After that, every faults vector has been fed to the Simulink model in order to simulate the real system and important variables such as currents, voltages, mechanical position and angular speed have been saved. These data have then been used to reconstruct the counter electromotive force coefficient in function of the angular position of the motor. The reconstructed CEMF has then been sampled using an algorithm to provide data inputs for various ANNs architectures.

Different ANNs architecture have been used as prognostics tools, showing very good performance score, even for very basic architectures. Various sampling strategies have been adopted, generally improving the predictive accuracy of the network increasing the number of inputs of the network itself, i.e. using a higher number of samples for each electrical commutation (as shown in sec. 5.6).

The study has proved that ANNs can be effectively used as prognostics tools for aerospace EMAs. The next step would be to conduct a parametric analysis in function of the many different variables defining a neural network, e.g. the

network architecture, the performance function, the training function, etc. After such analysis, a development in lower level language, e.g. C++ can be used to speed up the computation and possibly to allow a field testing campaign in order to evaluate the real-world behavior and the possible implementation on OBCs (On-Board Computers) in prevision of a full-scale, commercial implementation.

Appendix 1

MATLAB code

1.1 Faults generation

This simple MATLAB script creates a custom bounded list of faults condition using a modified latin hypercube approach.

```
% Fault generation using LHS with exponential spacing
%
% G. Quattrocchi - 05/2019

% Initialization
clc; clear; close;

% Flags
useModifiedLHS = 1;
saveData = 0;

% Fundamental values
nVars = 5;
faultsDiscretization = 6;
sampleNumbers = 1e5;

% Fault treshold verification data
nTresh = 0.5;
nEcc = 0.5;


if ~useModifiedLHS

```

```
24  %% Method 1: Output matrix creation using combvec
25  %  If the average of the faults value is greater
       than a cutoff value,
26  %  the row might be canceled following the result of
       a rng
27
28  S = 10; % Scaling factor
29
30  % Generic fault vector - logarithmically spaced
31  faultVec = exp(linspace(log(1), log(2*S),
       faultsDiscretization)) - 1;
32  faultVec = faultVec/(2*S-1);
33  %plot(faultVec)
34
35  % Phase vector definition - linearly spaced
36  initAngle = 0;
37  finalAngle = 360;
38
39  phaseVec = linspace(initAngle, finalAngle,
       faultsDiscretization);
40
41  outMat = combvec(faultVec, faultVec, faultVec,
       faultVec, phaseVec)';
42
43  cutoff = 0.1;                              % Cut-off
       value used to evaluate possible row deletion
44
45  for ind = 1:size(outMat, 1)
46      rowSum = sum(outMat(ind, 1:4));
47      if rowSum > cutoff
48          if rowSum >= rand(1)            % Compare row
               average with a rng
49              outMat(ind, :) = NaN;       % If
                   verified, set the row to NaN
50          end
51      end
52  end
53
54  outMat = rmmissing(outMat);                % Delete all
       NaN rows
```

```
55
56 else
57
58     %% Method 2: Modified latin hypercube
59     % Scaling factor definition
60     SF = 1e3;
61
62     % Creation of LHS with exponential spacing
63     seedLHS = lhsdesigncon(sampleNumbers, 5, [1 1 1 1
           1], [SF SF SF SF 360],...
64         [true true true true false]);
65
66     % Normalization
67     for i = 1:sampleNumbers
68         seedLHS(i, 1:4) = seedLHS(i, 1:4)./norm(seedLHS(i
               , 1:4));
69     end
70     outMat = seedLHS;
71 end
72
73 % Phase scaling
74 outMat(:, 5) = outMat(:, 5)*pi/180 - pi;
75
76 % Fault treshold verification
77 for ind = 1:size(outMat, 1)
78     if rssq(outMat(ind, 1:3)) > nTresh || outMat(ind, 4)
           > nEcc
79         outMat(ind, :) = NaN;
80     end
81 end
82
83 outMat = rmmissing(outMat);
84
85 outMat(:, 5) = interp1([-pi pi], [0 1], outMat(:, 5));
86
87 faultsList = outMat;
88 clear outMat;
89
90 if saveData
91     save('faultsList_30k.mat','outMat');
```

```
92 end
```

1.2 Optimal sampling strategy

This scripts determines the optimal sampling spacing for the second sampling strategy.

```
1  % Optimal sampling spacing determination
2  %
3  % G. Quattrocchi - 09/2019
4
5  clc; clear; close;
6  qplot = 1;
7  fitPlot = 1;
8  fitPlotComms = 1;
9  fitPlotMid = 1;
10 fitPlotExt = 1;
11
12 load('BEMF_HF_3k.mat');
13
14 phaseLen =  round(500/6);
15 halfPhase = round(phaseLen/2);
16
17 commExt = 1:phaseLen:500;
18 commExt = commExt + halfPhase;
19 halfPts = commExt + halfPhase; halfPts(end) = [];
20
21 delta = 1:halfPhase;
22 rowsCons = [4 8 15 16 23 42];
23
24 for k = 1:6
25     for j = 1:length(rowsCons)
26         rowTrial = rowsCons(j);
27         for ind = delta
28             q(ind) = (BEMFMat(halfPts(k) + ind, rowTrial
                   ) - ...
29                 BEMFMat(halfPts(k) - ind, rowTrial))/ind
                       ;
30             qp(ind, j, k) = q(ind);
31         end
```

```
32      end
33  end
34
35  deltaChc = 15;
36  halfPts
37  addPts = sort([halfPts + deltaChc, halfPts - deltaChc])
38  totalPts = sort([halfPts, addPts])
39
40  if qplot
41      for l=1:6
42          subplot(3, 2, l, 'align')
43          plot(qp(:, :, l));
44          str = ['Angular coefficient - Commutation ',
                num2str(l)];
45          title(str)
46          xlabel('Sampling interval width - \delta'),
                ylabel('q')
47          grid on
48      end
49  end
50
51
52  if fitPlot
53      nCurve = sort(randi(3e3, 1, 5));
54      figure();
55      plot(BEMFMat(1:600, nCurve))
56      hold on, grid on
57      if fitPlotComms
58          plot(commExt, BEMFMat(commExt, nCurve), 'r*')
59      end
60      if fitPlotMid
61          plot(halfPts, BEMFMat(halfPts, nCurve), 'g*')
62      end
63      if fitPlotExt
64          plot(addPts, BEMFMat(addPts, nCurve), 'k*')
65      end
66      title('CEMF constant sampling'), xlabel('Point'),
            ylabel('k')
67  end
```

1.3 Neural network creation and training

This script is used to create the neural network object, set the various parameters, perform the training and finally evaluate the performance of the network.

```
% Neural network creation, training and evaluation
%
% G. Quattrocchi - 2019

%% Init
clc; clear; close;

%% Crete neural network object
net = feedforwardnet([5]); format short;

%% Load datasets
load('BEMF_HF3k_1pts.mat');
load('faults_3k.mat');
BEMF = BEMFMat_1points;
faultsList = faults_3k;

%% Training
net.trainFcn = 'trainlm'; % 'trainbr'
net.trainParam.showCommandLine = true;
net.trainParam.epochs = 1e3;
net.trainParam.goal = 1e-6;
net.trainParam.max_fail = 10;
net.trainParam.mu_max = 1e6;
net.performFcn = 'mse';

net.layers{:}.transferFcn = 'satlins';

net = train(net, BEMF', faultsList','useParallel', 'yes'
   ,'showResources','yes');

%% Verification
verificationFaults = cell2mat(struct2cell(load('
   verMatFaults.mat')))
verificationSet_3pts = cell2mat(struct2cell(load('
   BEMF_HF_ver_1pts.mat')))';

```

```
verificationOut = net(verificationSet_3pts)'

abserr = (verificationFaults - verificationOut)
relerr = 100*(verificationFaults - verificationOut)./
   verificationFaults
```

Appendix 2

Detailed results

2.1 Verification set

The following set has been used as verification set for each neural network. It is composed by 10 different faults conditions randomly generated, spanning the same possible interval used during training.

N_a	N_b	N_c	ξ	ϕ
6.58E-02	1.12E-02	2.44E-02	1.47E-01	3.50E-01
4.41E-02	1.95E-02	6.48E-02	5.18E-02	6.90E-01
1.36E-01	1.99E-02	7.51E-02	3.46E-02	3.40E-01
2.97E-02	4.00E-03	2.02E-02	1.34E-02	5.63E-02
8.63E-02	6.98E-02	2.10E-01	1.75E-02	5.43E-01
3.81E-02	3.43E-02	7.74E-02	3.76E-02	4.79E-01
7.46E-02	8.00E-04	3.45E-02	3.84E-02	6.24E-01
4.37E-02	1.62E-02	7.33E-02	3.26E-02	4.47E-01
3.68E-02	4.73E-02	1.36E-02	1.70E-03	3.38E-01
1.33E-01	8.80E-03	2.66E-02	7.93E-02	2.17E-01

Table 2.1: Verification faults set

For each neural network, the results are reported in the following sections, in the form of relative error with respect to the faults set.

2.2 3 inputs per commutation, shallow networks

trainlm					trainbr				
N_a	N_b	N_c	ξ	ϕ	N_a	N_b	N_c	ξ	ϕ
18 neurons					**18 neurons**				
0.27	-8.19	0.44	0.22	-7.03	1.92	-14.41	-1.92	-1.06	-1.09
-0.49	-4.33	0.32	3.34	5.02	0.32	-5.02	1.12	1.58	0.62
0.53	1.69	-0.96	1.74	-0.67	-0.23	2.39	1.27	3.47	-9.73
-2.53	-23.85	-2.62	3.77	99.77	0.86	-10.64	-3.88	0.07	-50.88
0.69	2.98	-2.13	-7.35	11.85	0.33	1.07	-1.66	-10.94	2.56
-0.62	-1.11	0.38	2.60	-4.43	0.19	0.58	0.93	1.88	7.78
0.85	-162.56	-2.71	2.29	-0.19	1.86	-206.91	-2.57	2.64	-5.57
-0.39	-4.84	0.33	2.46	-3.64	0.36	-1.58	0.83	2.01	3.67
-1.63	0.96	-6.82	-27.44	-31.00	0.81	1.26	-7.67	-30.09	-35.26
0.80	-2.06	-1.38	-2.28	-7.66	1.17	-11.29	0.80	-0.34	1.81
16 neurons					**16 neurons**				
0.87	-6.01	-1.28	-2.62	2.60	0.61	-10.44	2.33	-0.80	9.01
-0.23	-3.70	-0.23	3.21	0.14	-0.01	-2.25	0.40	3.68	-6.83
0.25	3.13	-1.11	0.01	8.78	0.02	0.37	-0.42	2.90	-2.96
1.09	-35.71	-7.66	6.78	99.77	1.36	-18.33	-1.49	-3.69	-9.11
0.43	3.52	-1.18	-11.58	6.32	0.22	2.47	-1.66	-14.28	5.71
0.16	-2.11	-0.50	-0.50	0.02	-0.22	-0.82	0.86	0.22	-1.38
1.69	-99.24	-2.13	3.79	-2.90	1.01	-169.04	-1.77	2.08	6.70
-0.14	-6.71	-1.06	-0.98	-2.84	-0.47	-5.24	0.59	0.58	-8.95
-0.44	-0.60	-10.35	-104.67	-18.28	-1.08	-0.74	-14.55	-56.65	-33.44
1.09	-6.62	-5.37	-3.06	-21.72	0.81	-2.43	-2.60	-1.16	5.13
14 neurons					**14 neruons**				
0.86	-3.96	-5.42	-2.87	-0.13	0.00	-17.69	0.93	0.25	-37.32
-1.04	-4.13	-1.72	0.49	-0.52	-1.32	-6.23	1.69	1.04	7.24

`trainlm`					`trainbr`				
-0.57	3.60	-2.68	-2.06	9.23	-0.19	0.95	0.42	5.45	-0.35
0.68	-17.48	-12.10	-3.68	99.77	-0.66	-34.84	-1.96	-1.99	-36.55
-0.91	3.58	-2.31	-9.49	4.42	0.33	-1.07	-1.46	-5.70	-8.23
-0.60	-1.58	-1.29	-3.53	5.14	-1.82	-1.66	1.61	1.63	-16.46
0.80	-158.55	-7.52	1.09	-3.97	1.01	-240.81	-0.50	3.17	1.75
-1.32	-4.42	-1.56	-4.25	5.92	-1.27	-5.38	1.42	2.81	-19.53
0.22	2.43	-15.98	-23.03	-27.15	2.39	2.43	-10.04	-2.15	-32.11
0.71	3.95	-7.60	-2.24	7.59	1.01	-13.61	0.31	-2.88	3.68
12 neurons					**12 neurons**				
4.93	-18.90	-1.25	2.48	-6.35	1.29	-0.39	1.55	-3.00	1.20
-2.46	-4.79	-1.21	1.33	1.72	-0.34	-6.07	0.66	3.45	2.11
0.83	-2.02	-0.82	3.72	1.22	-0.62	-2.25	-1.27	7.79	3.13
-4.62	-62.39	-10.64	15.31	99.77	3.24	-18.19	-0.96	7.36	99.77
1.61	3.44	-1.24	-9.61	1.73	0.87	4.11	-0.94	-8.36	4.65
-1.94	-1.16	0.38	0.61	6.83	-1.07	-2.44	-0.03	6.53	-4.24
0.48	-204.45	-7.62	2.84	-1.44	1.29	-71.17	1.67	5.29	-1.90
-0.22	-4.17	0.52	0.87	4.16	-0.64	-7.58	-0.08	7.66	-4.99
-1.65	-3.77	-24.20	41.45	-25.80	-5.18	-0.18	-28.10	-102.80	-36.09
2.96	-17.56	-4.99	1.88	6.00	0.38	2.11	-3.40	-1.52	-26.68
10 neurons					**10 neurons**				
-1.32	-12.50	-7.75	-3.59	0.11	3.46	10.18	-2.91	-2.74	-12.84
1.53	0.68	-0.80	2.59	-1.11	3.52	-14.66	-0.77	4.91	-4.68
-1.50	5.41	1.99	4.57	6.52	-0.64	4.62	-0.18	-2.07	-17.35
1.22	-2.68	1.14	-6.56	99.77	-3.60	11.81	-0.36	-3.09	99.77
0.35	2.32	-1.93	-8.12	3.76	-0.06	4.21	-1.74	-22.06	1.16
1.99	1.74	0.52	2.00	9.43	1.94	-1.90	0.79	-1.50	-0.86
2.23	11.98	-1.11	3.38	-2.13	1.37	-203.20	-2.65	-1.02	0.87
0.50	0.84	-0.01	1.51	12.06	0.66	-2.50	1.06	-3.23	-6.63
0.84	1.21	-2.71	-77.19	-31.35	-0.91	3.66	-4.91	-69.39	-35.86

trainlm					trainbr				
-0.95	-5.88	-1.09	-1.38	-3.73	2.39	12.12	-3.92	-0.48	-0.62
8 neurons					**8 neurons**				
18.47	-60.96	20.53	1.68	1.78	11.67	-1.11	99.98	-3.34	-1.56
-7.89	6.68	-2.22	-0.30	12.06	4.68	-1.87	0.05	-1.98	-2.13
0.06	2.26	-1.46	-0.78	10.37	0.30	-0.47	0.19	4.26	-3.02
1.22	-38.96	-4.56	-5.45	18.46	-2.45	-100.23	-25.86	-9.98	99.77
-2.17	3.16	-0.99	-5.88	-1.67	7.41	-2.09	2.73	68.14	11.96
4.33	-2.69	1.15	2.63	2.05	3.55	-0.22	-2.08	-3.38	-4.64
-0.38	-107.50	-1.97	-0.42	2.46	2.83	-194.68	-0.47	-2.88	-4.10
2.17	-7.37	0.16	2.92	11.87	4.24	-2.49	0.09	-5.83	-2.51
13.20	-6.40	12.96	-4.86	-23.42	0.21	6.95	-50.56	-487.84	-25.26
1.82	-23.08	-1.31	0.16	-6.92	5.19	44.96	28.17	-13.60	9.38

Table 2.2: Relative errors, percentage, 3-inputs per commutation shallow networks

2.3 3 inputs per commutation, deep networks

trainlm					trainbr				
N_a	N_b	N_c	ξ	ϕ	N_a	N_b	N_c	ξ	ϕ
16-8 neurons					**16-8 neurons**				
2.80	-4.35	-5.44	-1.41	9.22	1.62	-10.17	-2.49	-0.69	5.14
-0.13	-0.76	2.01	-2.09	1.64	-0.60	-2.59	0.90	-1.84	1.55
0.00	4.04	0.45	-2.03	-3.77	0.42	1.59	0.10	-0.49	5.68
-3.74	-21.00	-1.57	5.55	29.82	2.94	-16.60	-3.75	2.12	36.76
-0.85	2.51	-1.96	-5.74	2.13	-0.18	-0.11	-1.57	-10.58	9.36
-0.25	-1.66	1.58	-4.23	-4.38	-1.49	-0.97	0.52	1.78	-9.80
1.55	-161.91	1.54	-4.20	2.09	0.64	-136.97	-3.32	0.51	-0.16
0.00	-5.38	0.94	-5.72	-7.23	-0.26	-3.50	0.51	-0.21	-2.69
2.24	1.37	2.59	98.27	-12.88	-3.22	0.55	-9.41	98.27	-14.85
2.23	1.20	-4.69	-1.91	-11.09	1.29	-5.82	-3.49	-2.01	16.13

trainlm					trainbr				
14-7 neurons					**14-7 neurons**				
-0.51	-12.22	-1.83	-1.57	0.51	0.95	2.74	-2.37	-1.46	-3.56
-0.03	-1.36	1.28	0.84	0.51	1.57	-3.73	-0.15	1.53	-4.20
0.03	0.52	2.54	2.32	-5.9	-0.07	-0.71	0.39	0.54	-0.94
4.16	-15.37	-7.77	10.03	61.67	-1.20	-74.98	-3.25	-1.78	34.98
-0.75	1.44	-1.52	-16.7	-4.52	0.36	2.69	-1.88	-8.80	6.03
-0.31	0.44	2.04	1.14	5.49	1.67	0.90	0.45	5.71	-1.77
1.99	-146.09	-0.56	0.77	-0.5	1.39	-74.67	-2.06	4.70	-1.45
-0.41	-2.47	2.14	0.98	3.79	2.13	-2.35	-0.08	5.70	-0.64
2.87	3.46	-16.44	-61.66	-18.69	-0.76	3.69	-15.17	-249.07	-19.87
0.31	-12.1	-2.83	-2.91	-14.44	2.94	14.76	-7.95	-2.39	12.36
12-6 neurons					**12-6 neruons**				
-0.44	-3.84	3.49	-2.44	3.18	3.25	-2.83	2.62	-1.21	2.01
0.38	-2.11	0.43	-1.15	1.70	0.53	4.56	3.02	0.96	8.23
0.14	3.25	-1.92	4.22	-12.67	-0.58	-2.73	0.12	-1.82	-8.98
0.91	-19.07	-3.55	6.03	-67.40	0.30	2.60	-0.27	-3.10	-134.72
-2.21	0.19	-3.15	3.90	-6.23	0.92	1.76	-1.69	-13.34	-8.69
1.04	0.93	0.42	-1.73	-1.32	-0.37	-2.10	1.52	2.43	11.31
0.91	-49.20	-5.07	-1.61	4.02	1.32	-129.88	0.75	2.20	-1.79
1.35	2.59	-0.38	-2.31	-6.34	-0.89	-6.11	1.96	1.42	8.07
-1.30	-1.29	-5.46	98.27	-35.48	-0.79	-4.97	-50.25	4.60	-31.25
1.29	9.22	-0.19	-3.19	-1.94	1.27	-9.61	1.96	-3.88	-24.89

Table 2.3: Relative errors, percentage, 3-inputs per commutation deep networks

2.4 2 inputs per commutation, shallow networks

trainlm					trainbr				
N_a	N_b	N_c	ξ	ϕ	N_a	N_b	N_c	ξ	ϕ
11 neurons					**11 neurons**				
1.33	-9.11	3.78	-2.81	-1.35	3.34	0.60	-1.22	-3.29	2.65
-4.36	-1.72	2.19	1.89	-1.74	-2.38	-1.14	0.50	4.52	0.09

trainlm					trainbr				
1.02	-0.66	-1.01	7.41	6.85	1.03	12.30	-0.22	1.78	-12.27
-3.51	-76.26	-12.65	3.14	99.77	7.56	-28.94	-2.26	6.44	-45.02
-1.13	2.37	-1.62	-14.56	4.45	-1.16	1.74	-1.91	-5.52	1.46
0.69	0.01	0.29	8.23	2.96	2.55	2.57	0.98	1.93	15.85
-0.13	-168.08	-0.77	5.46	-11.04	0.76	-91.21	-1.89	5.82	-10.55
1.19	-3.56	-0.08	9.19	2.84	2.02	6.70	0.65	1.94	10.40
-4.81	2.77	-3.90	-52.36	-35.50	11.83	-2.01	3.35	-40.89	-33.55
2.32	-3.38	-0.21	-1.97	-11.83	2.26	21.18	-1.47	-2.60	-29.21
9 neurons					**9 neurons**				
4.88	8.44	0.23	-1.94	-0.17	8.40	25.08	13.02	-2.67	20.54
0.82	8.42	-0.06	-1.67	-9.48	-7.12	2.42	1.24	2.70	7.21
-0.89	9.25	-0.68	2.86	1.33	-2.20	5.04	2.23	4.94	-33.32
0.96	-52.14	-2.70	7.91	99.77	-11.12	16.47	-7.10	-1.40	-41.35
4.17	-1.96	-1.21	-28.11	5.10	-1.75	2.86	-1.78	-0.94	3.06
-1.49	-0.24	0.95	9.21	5.35	-7.59	2.53	2.32	2.39	-5.76
3.48	-213.05	-1.72	-0.13	-2.12	-2.42	-37.74	-0.85	3.84	9.52
-1.23	-1.78	0.66	9.16	8.65	-7.37	2.62	2.16	3.96	-12.28
1.49	-0.77	-3.40	-143.94	-30.58	-4.57	6.01	0.04	-160.65	-39.89
2.61	-2.15	-0.95	1.45	-11.22	2.10	29.51	2.15	1.45	-69.03
6 neurons					**6 neruons**				
-8.43	32.79	-10.07	-3.84	-4.95	3.35	-138.03	-9.16	-13.99	-15.06
43.99	13.23	0.35	3.57	14.19	0.53	99.91	0.72	1.42	14.94
14.12	91.10	1.30	12.31	-4.57	0.81	-18.45	-0.35	-7.26	-16.19
-4.81	-63.70	-0.73	-2.36	77.88	-2.02	-21.18	-3.56	-41.71	-96.04
19.87	21.72	-1.95	4.60	-8.00	-2.05	7.31	-0.75	11.27	-4.71
-0.45	-7.79	1.32	2.59	-0.56	0.32	-4.59	0.95	-10.74	-7.84
15.48	97.87	-0.79	6.57	11.47	2.16	97.87	-3.58	-15.59	6.32
-7.43	-19.36	2.76	2.28	-8.48	1.44	-77.75	0.49	-21.71	-18.84
-54.27	-16.45	4.06	-254.56	-21.85	-2.48	-39.05	-4.79	-381.34	-29.98
-2.69	99.81	-4.84	0.80	-31.98	1.96	-204.28	-5.55	-15.40	49.70

Table 2.4: Relative errors, percentage, 2-inputs per commutation shallow networks

2.5 1 input per commutation, shallow networks

trainlm					trainbr				
N_a	N_b	N_c	ξ	ϕ	N_a	N_b	N_c	ξ	ϕ
6 neurons					**6 neurons**				
30.86	-206.47	-4.05	-13.92	-13.46	23.34	27.82	71.22	-16.59	6.12
3.72	99.91	4.69	3.91	13.59	16.37	5.39	42.27	19.54	17.49
2.84	-42.53	1.63	-10.12	-16.24	1.62	15.66	-6.63	44.12	-18.65
11.77	-19.79	18.82	-5.13	-74.42	0.32	-46.89	-15.34	-54.62	-91.36
3.01	6.31	-1.83	-0.68	-15.99	15.88	12.74	-0.20	99.83	5.34
8.02	8.39	1.16	-3.68	-9.38	24.42	6.43	-3.74	21.51	-3.10
9.38	-599.47	5.98	-8.85	1.58	6.24	-44.88	16.54	-11.73	12.88
14.71	-50.71	0.54	-13.13	-21.46	21.15	10.24	-11.96	15.40	-9.60
4.10	55.98	-116.67	-942.72	-44.31	26.55	4.43	-103.01	-943.07	-48.26
14.65	-388.15	3.98	-18.04	23.67	6.02	22.87	-82.98	-20.29	-52.23
5 neurons					**5 neurons**				
99.99	30.32	-28.13	-3.76	-20.63	36.27	99.85	99.98	-9.76	-14.87
-43.21	99.91	-2.76	-21.14	13.80	-3.41	99.91	-7.48	-8.85	13.75
10.68	-5.29	-0.06	80.13	-8.42	10.38	-12.46	51.24	78.44	-13.85
-118.38	-159.31	-35.37	-255.38	-97.69	-102.19	-80.12	-193.51	-216.77	-94.88
4.11	11.02	-0.21	99.83	-8.04	78.13	7.05	7.02	99.83	6.51
-3.18	-23.59	-3.00	-21.97	-5.06	23.41	-24.95	10.60	-6.31	-10.95
-21.58	-335.31	-16.04	-29.55	6.79	-13.87	97.87	2.35	-22.35	6.05
-5.11	-111.36	-6.78	-45.01	-15.30	11.51	-115.42	12.49	-24.58	-21.16
-48.17	-36.71	-59.12	-935.84	-37.16	-70.55	-41.66	-171.43	-982.28	-44.24
73.12	-71.62	-22.95	22.29	32.88	7.62	-177.60	99.98	6.97	-21.69

Table 2.5: Relative errors, percentage, 1-input per commutation shallow networks

Bibliography

[1] *A list of cost functions used in neural networks, alongside applications.* URL: https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications.

[2] Reyad Abdel-Fadil, Ahmad Eid, and Mazen Abdel-Salam. "Fuzzy logic control of modern aircraft actuators". In: *3rd International Conference on Energy Systems and Technologies*.

[3] Federal Aviation Administration and Flight Standards Service. *Pilot's handbook of aeronautical knowledge*. US Dept. of Transportation, Federal Aviation Administration, 2003, p. 6.9.

[4] Manuela Battipede et al. "Model Based Analysis of Precursors of Electromechanical Servomechanism Failures". In: *AIAA Modeling and Simulation Technologies Conference*. 2015, p. 2035.

[5] *Bayesian regularization backpropagation.* URL: https://it.mathworks.com/help/deeplearning/ref/trainbr.html?searchHighlight=trainbr&s_tid=doc_srchtitle.

[6] Pier Carlo Berri. "Genetic Algorithms for Prognostics of Electromechanical Actuators". Master Degree Thesis. Politecnico di Torino, 2016.

[7] *BLDC Motor Cross Section.* URL: https://www.islproducts.com/motors/brushless-motors-dc-bldc/bldc-motor-cross-section/.

[8] L Borello and MDL Dalla Vedova. "Dry friction discontinuous computational algorithms". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 3.8 (2014), pp. 1–8.

[9] PM Churn et al. "Electro-hydraulic actuation of primary flight control surfaces". In: *IEE Colloquium on All Electric Aircraft*. IET, 1998.

[10] *Comparing Slotted vs. Slotless Brushless DC Motors*. URL: https://www.haydonkerkpittman.com/-/media/ametekhaydonkerk/downloads/white-papers/comparing_slotted_vs_slotless_brushless_dc_motors%20l.pdf?la=en.

[11] *Counterclockwise commutation table*. URL: https://www.lucidar.me/en/actuators/commutation-for-bldc-motors/.

[12] Matteo D.L. Dalla Vedova. "Design of physical mathematical models suitable for advanced simulations and design of flight control and study related innovative architecture". Ph.D. Dissertation. Politecnico di Torino, 2007.

[13] Matteo D.L. Dalla Vedova, Paolo Maggiore, and Lorenzo Pace. "Proposal of prognostic parametric method applied to an electrohydraulic servomechanism affected by multiple failures". In: *WSEAS Transactions on Environment and Development* 10 (2014), pp. 478–490.

[14] Geng Deng and Michael C Ferris. "Neuro-dynamic programming for fractionated radiotherapy planning". In: *Optimization in medicine*. Springer, 2008, pp. 47–70.

[15] Bernard Etkin. *Dynamics of atmospheric flight*. Courier Corporation, 2012, p. 112.

[16] *Flight control surfaces of Boeing 727*. URL: http://www.faa.gov/regulations_policies/handbooks_manuals/aircraft/amt_handbook/media/FAA-8083-30_Ch03.pdf.

[17] Martin T Hagan and Mohammad B Menhaj. "Training feedforward networks with the Marquardt algorithm". In: *IEEE transactions on Neural Networks* 5.6 (1994), pp. 989–993.

[18] Geoffrey E Hinton. "A practical guide to training restricted Boltzmann machines". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.

[19] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications". In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.

[20] *Hyperbolic tangent sigmoid transfer function*. URL: https://it.mathworks.com/help/deeplearning/ref/tansig.html?searchHighlight=tansig&s_tid=doc_srchtitle.

[21] Xiaodong Jia et al. "Assessment of data suitability for machine prognosis using maximum mean discrepancy". In: *IEEE Transactions on Industrial Electronics* 65.7 (2017), pp. 5872–5881.

[22] *Levenberg-Marquardt backpropagation.* URL: https://it.mathworks.com/help/deeplearning/ref/trainlm.html?searchHighlight=trainlm&s_tid=doc_srchtitle.

[23] Paolo Maggiore and Matteo D.L. Dalla Vedova. *Lectures notes from 'Modellazione, simulazione e sperimentazione dei sistemi aerospaziali' course.* Politecnico di Torino.

[24] Ahmed Mosallam, Kamal Medjaher, and Noureddine Zerhouni. "Data-driven prognostic method based on Bayesian approaches for direct remaining useful life prediction". In: *Journal of Intelligent Manufacturing* 27.5 (2016), pp. 1037–1048.

[25] Tim Ockenden. *Concorde takeoff.* 2000. URL: https://www.alamy.com/stock-photo-concorde-takeoff-106408809.html.

[26] Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. "Training recurrent networks online without backtracking". In: *arXiv preprint arXiv:1507.07680* (2015).

[27] Stefano Re. "Development and comparison of prognostic methodologies applied to electromechanical servosystems (EMA) for aerospace purposes". Master Degree Thesis. Politecnico di Torino, 2018.

[28] Nicola Secomandi. "Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands". In: *Computers & Operations Research* 27.11-12 (2000), pp. 1201–1225.

[29] *Symmetric saturating linear transfer function.* URL: https://it.mathworks.com/help/deeplearning/ref/satlins.html?searchHighlight=satlins&s_tid=doc_srchtitle.

[30] Vyacheslav Tuzlukov. *Signal processing noise.* CRC Press, 2018.

[31] Pranshu Upadhayay and KR Rajagopal. "Permanent magnet overhang effect in PM BLDC motor using 2D & 3D finite element analysis". In: *2012 Nirma University International Conference on Engineering (NUiCONE).* IEEE. 2012, pp. 1–3.

[32] George J Vachtsevanos et al. *Intelligent fault diagnosis and prognosis for engineering systems.* Vol. 456. Wiley Hoboken, 2006.

[33] Saeed V Vaseghi. *Advanced digital signal processing and noise reduction.* John Wiley & Sons, 2008.

[34] *What are the differences between slotted and slotless motors?* URL: https://www.motioncontroltips.com/whats-the-difference-between-slotted-and-slotless-motors/.

[35] Jacek M Zurada. *Introduction to artificial neural systems*. Vol. 8. West publishing company St. Paul, 1992.