POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Investigation on the use of residual torque as a prognostic indicator for aerospace EMAs

An ANN-based hybrid methodology

Supervisors
Eng. Matteo D. L. Dalla Vedova
Eng. Gaetano Quattrocchi
Eng. Pier C. Berri
Prof. Paolo Maggiore

Candidate
Alessandro Iacono

October 2020

Summary

Electromechanical Actuators (EMAs) offer great advantages over their traditional counterparts (namely old Hydromechanical and modern Electrohydraulic Actuators) when used as actuation devices on aircraft. They represent the natural evolution of actuation systems in the *more electric* and *all electric aircraft* design philosophies, as using EMAs for both primary and secondary flight controls would eliminate the need for hydraulic and pneumatic power aboard the aircraft, leading to an overall weight reduction and a more convenient way to distribute mechanical power across the aircraft, as distributing electrical power directly to the end users is easier and lighter than distributing pressurized hydraulic fluid.

Still, as of today, the use of EMAs is limited to secondary flight control (such as airbrakes, spoilers and high-lift devices) on large aircraft, and they are used as primary flight control actuators only on small UAVs, and, in general, application where the loss of actuation system is neither mission critical nor would lead to loss of life or expensive flying systems.
This is partially explained by the fact that EMAs are still a relatively new technology in the aerospace sector: their combined fault modes are yet to be fully understood and they generally lack established prognostic methodologies.

Nonetheless, in recent years, many diagnostic and prognostic methods for EMAs have been proposed. The aim of prognostic methods is the estimation of the health status and/or the Remaining Useful Life (RUL) of various components of the EMA so that they can be isolated or replaced accordingly, a cardinal principle of modern Prognostics & Health Management (PHM) philosophies. Many methods proposed to estimate the health status of components rely on the analysis of one or more signals outputted by the system or reconstructed from output variables (as in the case of the back-electromotive force, or BEMF), which are considered prognostic indicators; this approach is often described as hybrid since it leverages both machine learning techniques and knowledge of the physical system. The analysis is thus performed with specifically trained neural networks, which use said prognostic indicators to estimate the health status of

one or more components.

In this framework, the residual torque, defined as the sum of all the friction and viscous torques in the transmission of the actuator, stands out as a possible candidate to be a valid indicator, as it carries information about the friction coefficients (variation of which are a telling indicator of wearing, possible jamming and other kinds of degradation in the transmission) of the system and can be reconstructed from other data acquired during the functioning of the EMA, such as the electrical current in the motor, the acceleration of the shaft and the hinge moment on the actuator.

In this work the viability of the residual torque as a prognostic indicator for EMAs in a neural-network-based methodology is investigated, both in the context of a pre/post-flight routine on ground and of real time use during the flight. The static and dynamic friction coefficients, as well as their ratio and the transmission efficiency under both aiding and opposing loads are considered targets of interest for this application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As mentioned in the Summary, the aim of this work is to investigate the potential uses of the residual torque (defined as the sum of all the friction and viscous torques in the transmission of the actuator) as a prognostic tool in various context. The meaning of this statement will be explained in detail in the next Sections and Chapters, but it is necessary to spend some words on the methods on which this work is based.

The approach used in this work was defined "hybrid", as it combines a *model based* approach with a *data driven* one: the use of a neural network to estimate the health status of the component is a classic *data driven* approach, while the use of a simulated model to obtain the data used to train it is a typical *mode based* approach. The meaning of those terms is explained more in detail in Section 1.2.

The focus of this work are mechanical faults in the transmission of an actuation system connected to an EMA, and the health status of its component is represented by deviation from the nominal values of a series of otherwise constant mechanical parameters appropriately chosen. Deviation of, for instance, the static friction coefficient of the transmission from its nominal value means that the component may be more or less worn or damaged depending on how much the value differs from its nominal one.
Once a component is properly characterized, the values of its mechanical parameters and constants can be correlated to the Remaining Useful Life, a concept explained in more detail in Section 1.2.

For the goals of this work, the neural network must be able to estimate the status of multiple parameters at the same time, even when more than one or

all of them are different from their nominal value, as this is expected in a real system.
The neural network must work both in a "test routine" mode, simulating the use of a single predetermined input command of the elevator during flight or on-ground, and in a "real scenario" mode, simulating a real flight condition where non-predetermined commands are used.

1.1 Prognostics and Health Management (PHM)

As aerospace systems become more and more complex, a trend virtually inseparable from the growth of the aerospace sector itself, the costs and time (which translates back into costs for commercial and defense programs) associated with their maintenance are sharply growing too.
Therefore it is not surprising that the trend in aircraft maintenance is moving towards tailored programs that aim to reduce downtime and unnecessary interventions by employing predictive maintenance strategies based on large amounts of data collected during the operational life of the aircraft.
Prognostics techniques are sparking great interest exactly for this reason. Prognostic, in engineering, can be defined as *"an engineering discipline focused on predicting the time at which a system or a component will no longer perform its intended function"* [1].
Integrating prognostics techniques into the design process is a key enabler for *Condition Based Maintenance* (CBM). This kind of maintenance, often defined as *"maintenance when need arises"*, is historically one of the first kinds of maintenance ever employed, but today it uses much more refined techniques, often based on AI software, and finds its place in the broader predictive maintenance field. The status of a system is continuously or discretely monitored (*condition monitoring*), and CBM is performed when one or multiple indicators reach a threshold value that implies the system is no longer capable of delivering the expected performances with an adequate level of safety.
The discipline that studies the interaction between failure mechanism and system life cycle management is called *Prognostics and Health Management* (PHM), and CBM is one of its more prominent aspects.
Another important aspect of this discipline is the calculation of the *remaining useful life* (RUL) of a system, defined as the interval between the time an observation/estimate of the system status is made and the time its fault occurs. RUL must be estimated using models capable of propagating the system status in time based on previous observations or estimates. The model themselves require a precise understanding of the fault modes of the system.

PHM's design philosophy has great advantages over now classic design philosophy such as *safe-life* when applied to mission or life critical components. Where *safe-life* principles dictates that a component must be designed with enormous safety margins and must be replaced after a fixed amount of time regardless of its condition, CBM advocates for monitoring the condition of the component and replacing it only when it is strictly necessary, that is, on a *"when-need-arises"* basis, as previously stated. Another great advantage consists in the reduction of the number of necessary redundancies, as unexpected failures could be virtually eliminated using PHM.
However, the main challenges that PHM must overcome in order to become widespread are evident: PHM implies that all the fault modes of a system are known, that a condition/health monitoring system is in place and is effective in providing reliable status indicators to the decision makers, that the models used to calculate the RUL are correct, that the threshold values for each status indicator are reliable, and that all the uncertainties associated with the prognostic process can be managed correctly. In their simplicity, the *safe-life* and, for similar reasons, the *damage tolerant with fixed maintenance intervals* philosophies are still preferred, even if they are less cost and time efficient that CBM.

However difficult it may be, solving this challenges will be rewarding. PHM benefits are not relegated only to maintenance and maintainers (as stressed enough in the previous paragraphs), but also involve categories such as [19]:

- **Logistics:** Reduced spares count and reduced logistics footprint
- **Fleet Management:** Reduced downtime, reduced life cycle costs, better mission planning, fleet health management
- **Design process:** More robustness, overall more effective requirements satisfaction
- **Regulatory bodies:** Better safety, avoiding catastrophic failures, minimize safety measures impact on other systems
- **Customer satisfaction:** Meeting higher customer expectations with relative ease

Figure 1.1 sums up PHM philosophy and gives an overview of the interactions among its main sub-disciplines. Prognostics will be explored in-depth in Section 1.2 as it is of major importance for this work, while Monitoring, Diagnostic and Health Management will be briefly treated here:

- **Monitoring**
  To properly assess the condition of a system it is obviously necessary to

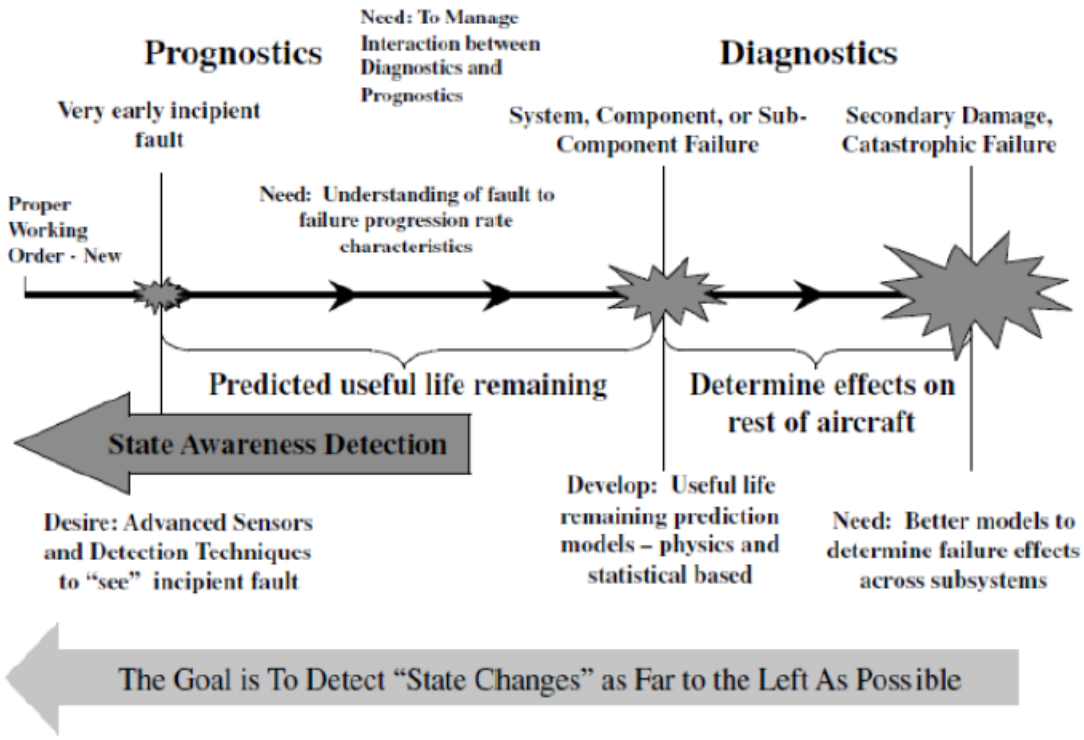


Figure 1.1. Prognostics and Health Management principles

gather and analyze direct or indirect data on the system. Just as obviously, this is done using an array of sensor tailored to the system at hand, the variables to be measured and the necessary accuracy. Reliability of the sensor is of a key requirement: their mean time between failures must be higher than the one of the systems and components they monitor, to avoid false positives or negatives. Another task of the monitoring process is to acquire data necessary to update and correct the estimates of the prognostic process.

- **Diagnostics**
  The diagnostic process detect, identify and isolate any failures in a given system. Following the detection of an unexpected behavior (through comparison with reference models, other systems, etc...) the process identifies the severity of the event and takes actions whose aim is to restore the system in a state as close as possible to the nominal one (for instance through the activation of redundancies and bypassing the faulty component).

- **Health Management**
  Health Managements typically consists of a series of strategies and methods whose aim is to assess the degradation trends, identify the root causes of the faults, correct RUL estimates and act to correct unforeseen trends or maximize the RUL of the components. All these actions are taken based on the data gathered during the monitoring, prognostics and diagnostics processes.

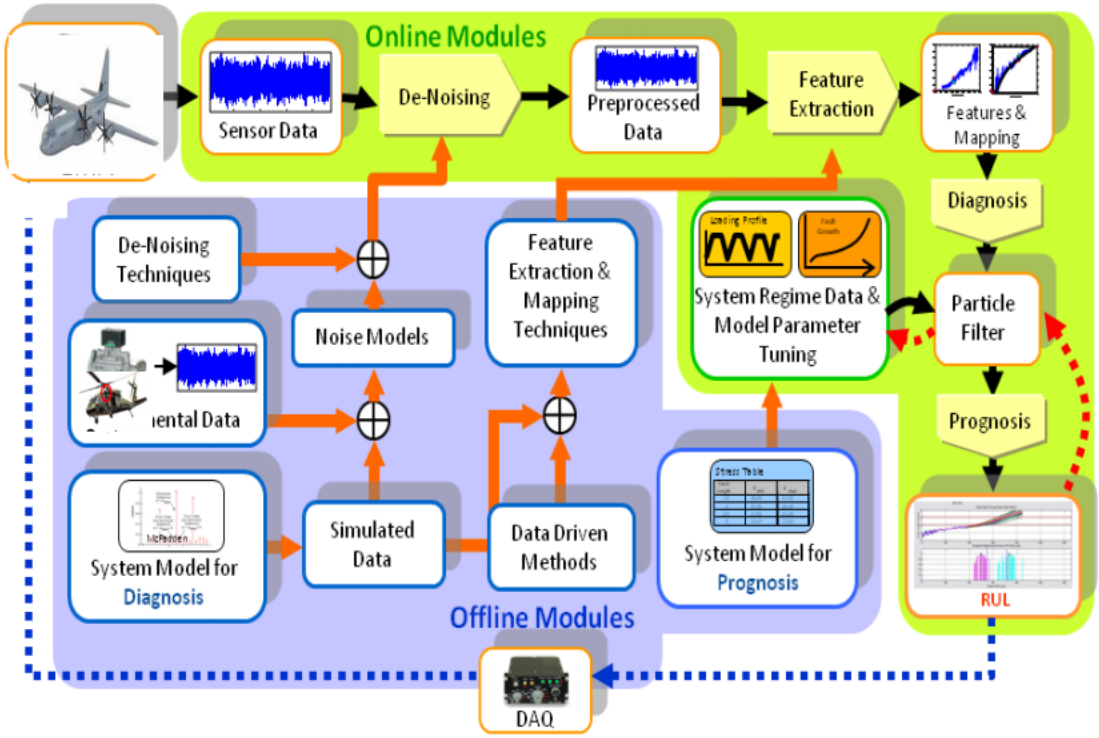


Figure 1.2. Prognostics and Health Management Architecture [19]

Figure 1.2 shows the general architecture of an aircraft designed using PHM principles, as it concern the deployment of a prognostics, diagnostics and monitoring system. As it pertains this work, the possibility of using system models for diagnosis and, in particular, prognostics is underlined.

1.2 Prognostics

Maybe the most important task of the prognostic process is to estimate the RUL of a component. This is done collecting data from the relevant components of subsystems through an array of sensors. These sensor must also sense each other to identify possible false reading, so it is important that the monitoring system is capable of detecting anomalies in th readings.
Data is subsequently filtered (with an analog or digital filter) to eliminate the influence of noise introduced by the operational environment. Eliminating noise is fundamental for the prognostic process as random fluctuations in the input data can negatively influence the accuracy of the process.
Finally, RUL is predicted using either adequate algorithms, statistical methods or ad-hoc trained neural networks. It is important to note that the estimated RUL must be confronted with a threshold value. Such threshold value usually includes a safety margin, and once it is reached the component requires maintenance.
Predicting RUL is not an easy task, and predictions over large periods of time can drift significantly or be invalidated by sudden micro-faults or the presence of unexpected defects in the component: for this reason RUL predictions must be updated whenever possible, a process that relies on knowing the real health status of a component at certain points in time. For instance let's imagine a simple algorithm that estimates the RUL by interpolating the known health status of a component over time and propagating the prediction to estimate when the component will reach unsafe operations territory: the often the real health status is known, the better the fit, the better the prediction; more frequent updated updates on the health status translate in a better ability of the algorithm to "catch" unexpected behavior from the component caused, among others, by unknown initial defects.

The classification of prognostics approaches is based on how the analyzed system is characterized:

- **Model-based prognostics**
  The prediction of the RUL is based on physical models. The system can be modeled either at the *macro* or *micro* level. The difference is that *macro* level models employ simplified relations and are most suited to analyze the whole system when there is no need to monitor and/or model the single components of the system at a *micro* level.
  The main drawbacks of this approach are the uncertainties introduced by the simplifications made and the general inherent complexity of a physical model.

- **Data-driven prognostics**

The prediction of the RUL is based on data-driven approaches such pattern recognition and machine learning techniques (such as deep learning or regressions and clustering). This approach is most suited to systems whose behavior is not fully understood or are too difficult to model.
The main drawback of this approach is the large amount of data needed to train the model.

- **Hybrid approaches**
  As the name implies, they are a combination of Model-based and Data-drive approaches. Most types of analysis are in practice hybrid. Hybrid approaches are subsequently divided in *pre-estimation* and *post-estimation fusion*. The former is used when diagnostic identifies faults to be corrected by maintenance, the latter to reduce uncertainty and thus increasing accuracy, and is based on the notion that more classifiers work better than a single classifier.

This work employs an hybrid pre-estimation approach. Residual torque data gathered using a *macro* component-level physical model are used to train a neural network, then used to estimate the RUL of the same system at unknown damage levels.

1.3 Flight Controls

A brief discussion of the flight controls follows for the sake of completeness. It is focused on aircraft with conventional architecture (i.e. fixed wing and T-shaped tail), but the methodology presented in this work could be applied to any flight control actuated through an EMA. In this work the EMA specifically controls one half of a split elevator.

1.3.1 Primary flight controls

Primary flight controls are the primary means to control the aircraft, controlling its rotation along its three main axis. They are used continuously by the pilot. Their operational principle is to change the shape of an aerodynamic surface in order to produce a small force far from the center of gravity, which in turns generates a large torque that is capable of rotating the aircraft. Primary flight controls must also produce a feedback (either natural of artificially reproduced) on the levers used for their actuation. Their actuation must be proportional and instinctive.

With reference to Figure 1.4, for an aircraft with conventional configuration the

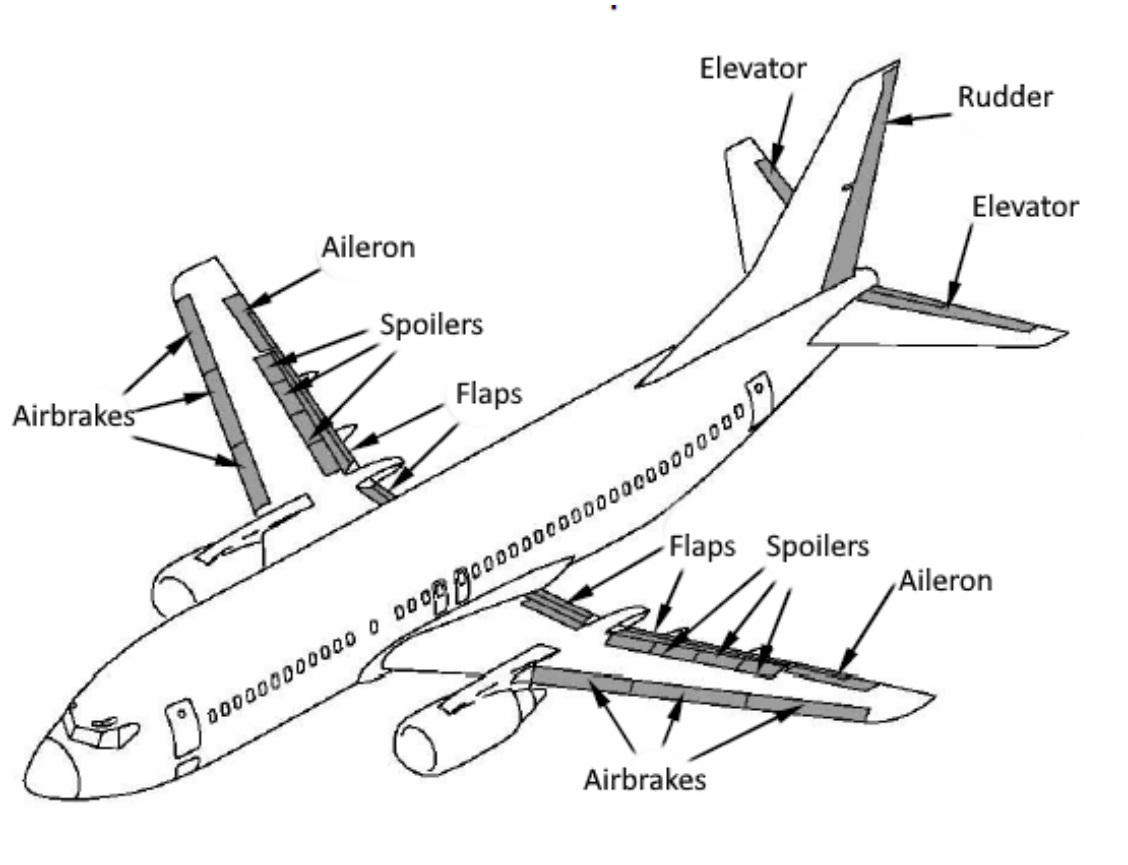


Figure 1.3. Classic layout of flight controls [8]

primary flight controls are:

- **Elevator:** It is positioned on the horizontal tail plane and controls the rotation along the Y-axis. The elevator is sometimes split in two separated but symmetrically actuated surfaces.
- **Rudder:** It is positioned on the vertical tail plane and controls the rotation along the Z-axis.
- **Ailerons:** They are positioned on both ends of the wing, are anti-symmetrically actuated and control the rotation along the X-axis

Flight controls can either be reversible (small aircraft and old large aircraft) or non-reversible (modern large aircraft), the former meaning that there is a direct connection between the control surfaces and the pilot (Figure 1.5), who must

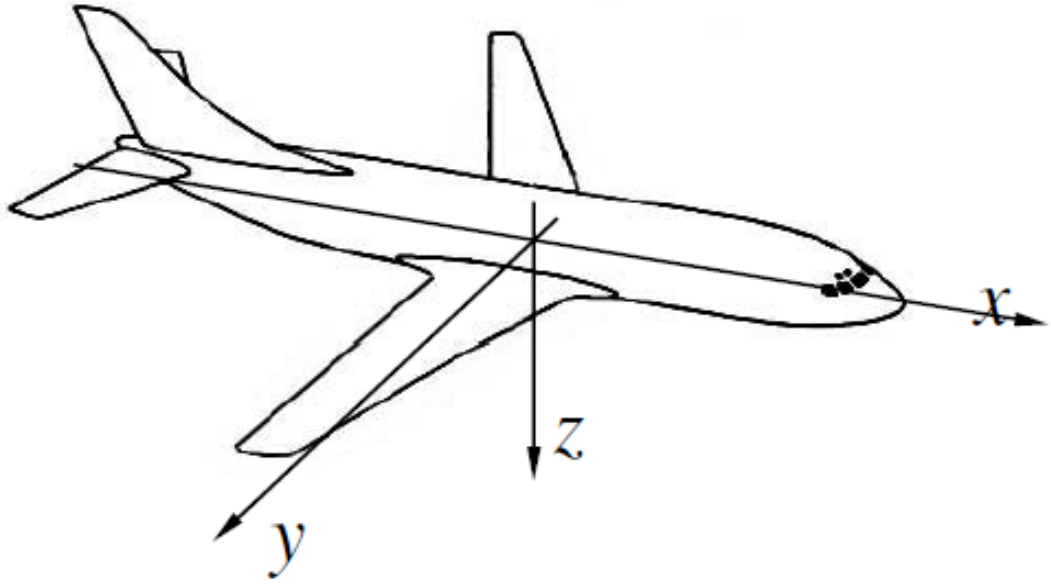


Figure 1.4. Aircraft Body Axis reference system [8]

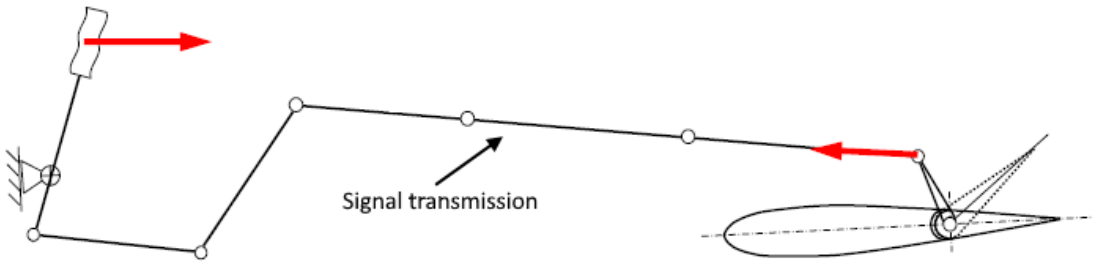


Figure 1.5. Example of simple reversible flight control [8]

overcome the aerodynamic forces on the control surface with his/her strength, and the latter meaning that such connection is indirect and often powered by a servo-actuator (Figure 1.6). For non-reversible primary flight controls, the feedback on the control stick must be artificially reproduced.

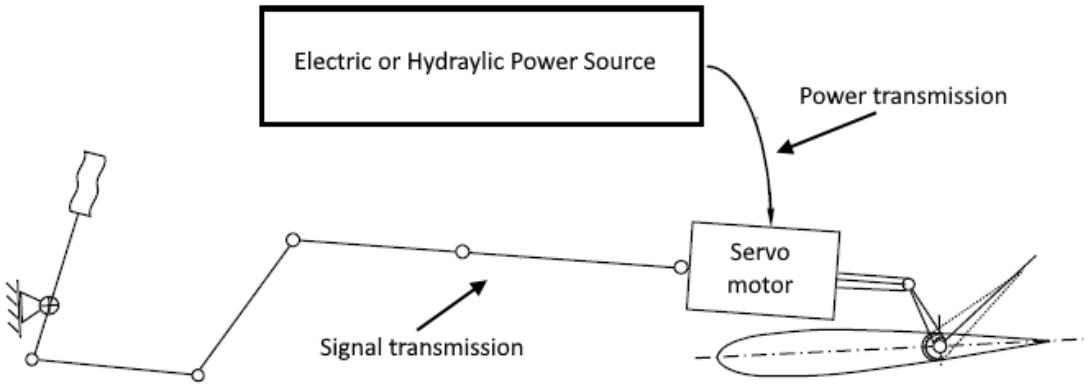


Figure 1.6. Example of simple non-reversible flight control [8]

1.3.2 Secondary flight controls

Secondary flight controls are used to modify the geometry and the aerodynamic characteristics of the aircraft. They are used occasionally (mainly at take-off and landing) by the pilot and usually have either on/off or stepped positions. Secondary flight controls should not produce a feedback on the levers used for their actuation and must remain in the commanded position even when the pilot hands are not on the controls.

Secondary flight controls may include:

- **High-lift devices:** Positioned either on the leading edge (*slats*) or the trailing edge (*flaps*) or both of the wing, they augment or reduce the camber and the planform of the wing, increasing C_L for the same angle of attack (*flaps*) or the available $C_{L_{MAX}}$ (*slats*). They are used at take-off and landing.

- **Airbrakes:** Positioned on the upper surface of the wing (or on the fuselage in military aircraft), they increase the C_D of the aircraft, slowing it down rapidly and enabling steeper descents at landing. They are also used to slow down after touching the ground.

- **Spoilers:** Mainly used on military aircraft, they locally disrupt the airflow leading to asymmetric increases of the C_D. The resulting torques on the aircraft help the pilot in performing tighter turns. For non military aircraft there is no distinction between airbrakes and spoilers.

1.4 Actuation systems

A discussion on the most common types of actuation systems for aircraft follows, again, for the sake of completeness. Greater attention will be paid to EMAs than the other actuation systems, as they are the object of this work.

Servomechanism usually work in a feedback loop, where their output is constantly measured against the required target, and the difference (*error*) is used as their proportional input.

1.4.1 Hydromechanical

The first non-reversible powered actuation systems used on aircraft where hydromechanical actuators. The pilot controls an hydraulic valve that moves the control surface usually through a spool. It is the hydraulic power (not the pilot strength) that moves the control surface and opposes the aerodynamic forces.

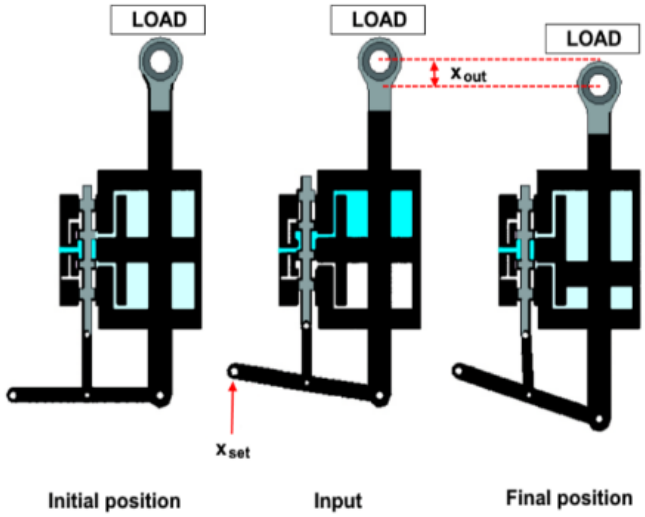


Figure 1.7. Example of Hydromechanical actuation [18]

1.4.2 Electrohydraulical

In Electrohydraulical Actuators the pilot usually controls a so called *flapper-nozzle servovalve*, that, in combination with a spool moved by the flapper, acts as the first stage of the actuator. In the flapper-nozzle servovalve an electric torque motor moves the flapper, that in turn, together with the nozzle, regulates

the pressure of the hydraulic fluid necessary to move the first stage spool. Subsequently the spool opens or closes other valves that regulate the flow of hydraulic fluid to the actuator jack, moving it accordingly to the required command. The control loop is closed via a LVDT sensor.
Electrohydraulical Actuators are extremely diffused nowadays, both in commercial and military aviation. They require a centralized hydraulic system to distribute hydraulic fluid to the users and, usually, a *fly-by-wire* control architecture, as the input for the flapper-nozzle servovalve is an electric signal generated from the input the pilot gave on the actual controls.

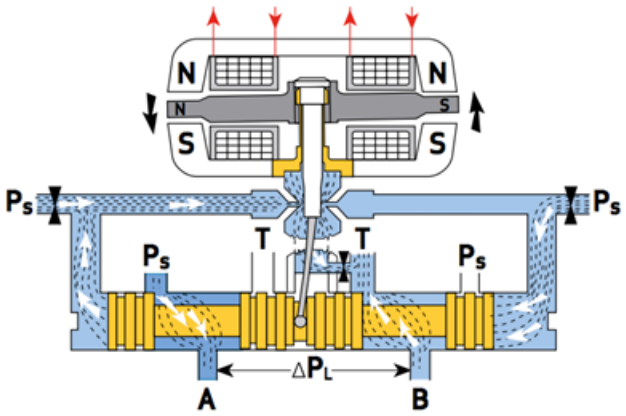


Figure 1.8. Schematics of a Flapper-Nozzle Servovalve [13]

1.4.3 Electrohydrostatic (EHA)

In Electrohydrostatic Actuators (EHA) a BLDC motor drives a relatively small piston pump (located near the flight control surface, for aircraft) that pressurizes hydraulic fluid in order to move a jack (just like in Hydromechanical and Electrohydraulical actuators). An EHA typically includes also a LVDT transducer to measure the piston position, anti-cavitation valves, pressure relief valves and a fluid reservoir to account for thermal expansion and off-nominal operations.
Compared to Hydromechanical and Electrohydraulical Actuators, EHAs do not require a distributed hydraulic system, as the necessary hydraulic power is generated closely to where it is needed and the hydraulic fluid is stored in proximity of the actuator itself.
In terms of weight savings, EHAs do not offer great advantages as the weight reduction from the lack of a centralized hydraulic system is offset by the need

of multiple piston pumps and BLDC motors. Nonetheless, their reliability is higher, they require low maintenance, and they are generally thinner than their traditional counterparts, as they are fully self-contained. These advantages are evident when considering that EHAs are currently used only on new generation fighters (F-22 and F-35, namely) and as backup on new Airbus aircraft. Their use in launch vehicles is currently being studied.

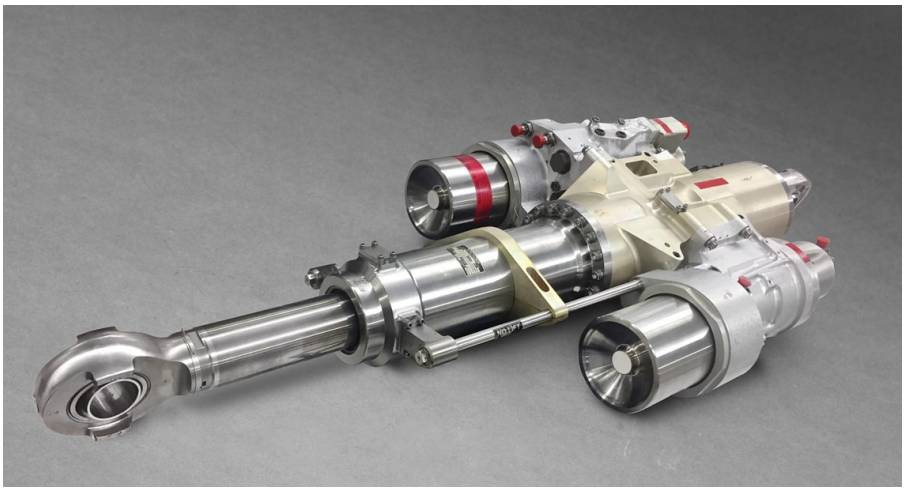


Figure 1.9. Example of an Electrohydrostatic Actuator [12]

1.4.4 Elecromechanical (EMA)

Electromechanical Actuators capabilities are sparking great interest as the aeronautic sector gravitates towards *more electric* and *all electric* philosophies, as making use of a single, electrical, power source for all the systems and subsystems of the aircraft, and in doing so eliminating both the hydraulic and the pneumatic systems, is certainly more convenient that employing multiple kinds of energy and fluids onboard.
EMAs typically receive their inputs through Advanced Control Electronics (ACE) that interpret the command sent by the pilot through the fly-by-wire system. The ACE then regulates the Power Drive Electronics (PDE), usually a 3-phase AC electrical power source, which in turn powers an electric motor (a BLDC, as it pertains this work) whose shaft, through a reduction gear, moves the screw jack connected to the control surface. The actual position is measured by an RVDT sensor and is fed back to the ACE to calculate the error with the commanded position that is fed to the PDE, repeating the loop. The use of a reduction gear is important because a typical BDLC motor spins at thousands or tens

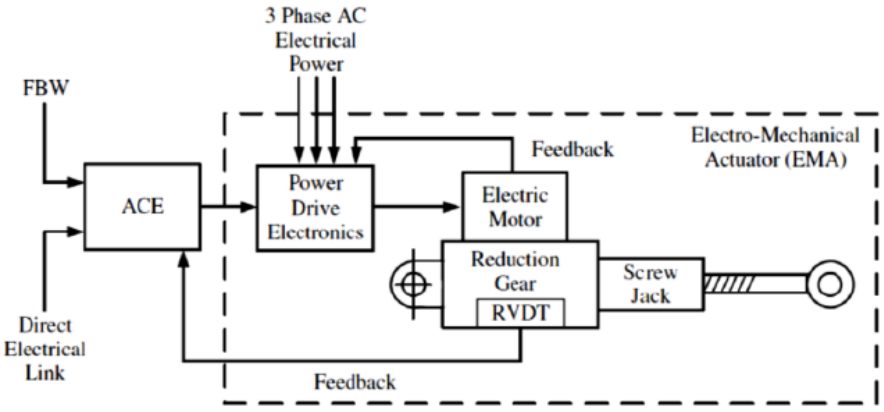


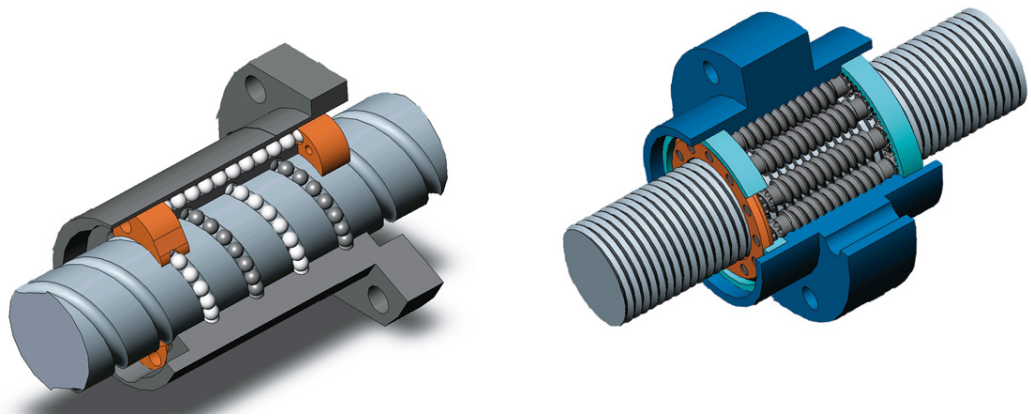
Figure 1.10. Schematics of an Electromechanical Actuator [2]

of thousand of RPM and provides very little torque, while the control surfaces of an aircraft requires high torque and low speed (tens of degrees per second). The transmission ratio, regardless of the type of transmission, can be defined as:

$$\tau = \frac{\dot{\theta_m}}{\dot{\theta_u}} = \frac{1}{\eta}\frac{T_u}{T_m}$$

So it represents the ratio of the angular speeds of the motor's and user's shaft, or the ratio of their torques mediated by the efficiency η of the transmission. As the transmission ratio τ value is typically several hundreds to one, it is important that the efficiency of the transmission is as high as possible, as to not waste torque.
After the reduction gear the rotary motion is converted into linear motion using epicyclic gears such as ball-screws or planetary roller-screws. Because of the typically low values of the rolling resistance and their good resistance to wear, among other qualities, these gears are extremely popular for this application.

BLDC motors are incapable of delivering high torque at speeds close to zero, so a prominent problem of EMAs is to hold a commanded position against external disturbances, as the energy used to power the BLDC motor in this scenario would be dissipated due to Joule effect and overheat the system, with consequent faster wear and degradation of the performances of the system. One solution could be to use an irreversible transmission, but this would lead to other problems, ad the control surface would remain stuck in the event of a failure of the EMA. For this reason, for same applications, the EMA comprehend an additional system used to disengage the transmission from the control surface in case of failure.



(a) Ball-screw (b) Planetary roller-screw

Figure 1.11. Mechanism of gears commonly used in EMAs [14]

Nonetheless, new hybrid stepper motors are being considered for use in EMAs, as they are able to deliver high torque at zero speed.

Chapter 2

Brushless DC Motors

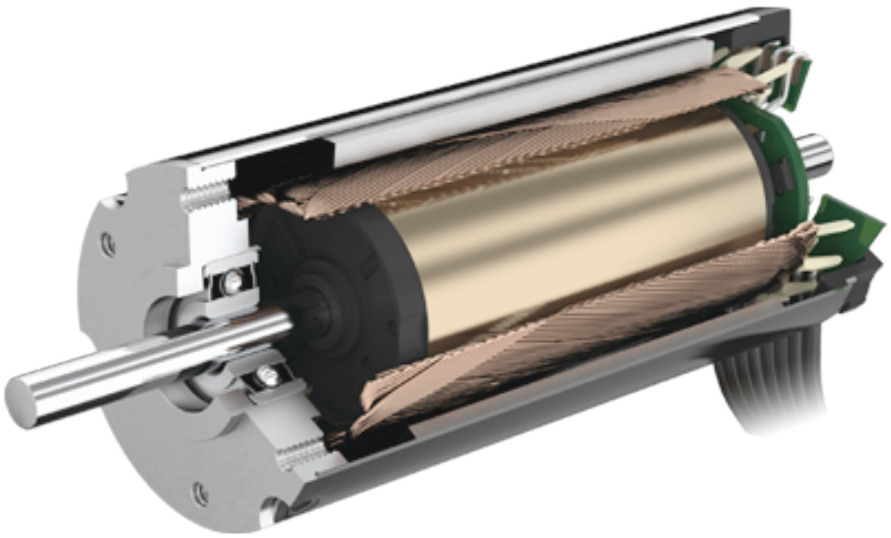


Figure 2.1. Cross-section of a BLDC Motor [6]

Over the years, Brushless DC Motors (BLDC motors) have taken over many of the tasks traditionally reserved to Brushed DC Motors in converting electrical power into mechanical power. Even though they require advanced control electronics to substitute the mechanical commutation with a fully digital one and are generally costlier than their brushed counterparts, the lack of sliding contacts makes for safer, longer lasting, cheaper to maintain and overall better electric motors, as they are not afflicted by the problems associated with the brushed commutation (sparks, constant wear, friction). Nowadays BLDC motors are the preferred choice in the aeronautic sector, from small RC planes to large

commercial aircraft and their electromechanical servoactuators.

Commutation of the electrical current, necessary to obtain a continuous rotation of the shaft, is done electronically in BLDCs through the use of either Hall sensors (which measure the angular position of the rotor/shaft sensing the variation of the electromagnetic field induced by the magnets) or designs based on the measuring of the Counter-ElectroMotive Force (CEMF). The latter options are effectively sensor-less architectures and are simpler and cheaper than the Hall sensor architectures; the drawback is that such architectures must be used in an open control loop, as measuring CEMF close to zero speed is extremely hard, making them unusable in servoactuator applications. However, the Hall-sensor based designs enable fluid and precise control of the position and speed of the shaft.

A description of the main parts of a BLDC Motor and its characteristics

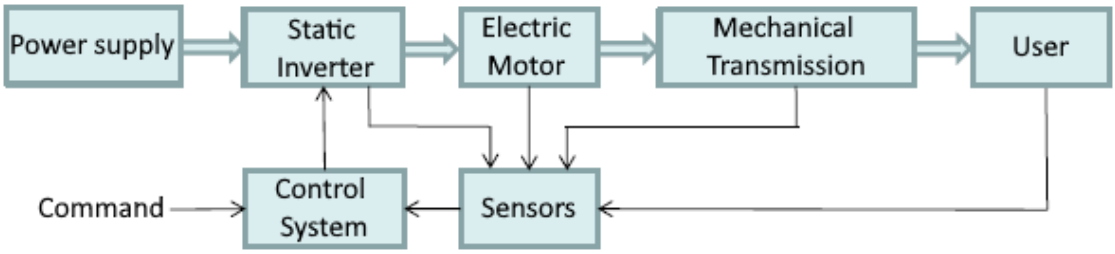


Figure 2.2. Block scheme of a BLDC architecture [9]

and performances follows.

2.1 Operational principle

As every other electrical motor [*citation needed*], a BLDC motor produce the rotation of its shaft thanks to a rotating magnetic field. The stator part of the motor actually generates the rotating magnetic field that interacts with the magnetic field of the rotor, generating a torque on the motor. Due to this torque, the rotor rotates to align the magnetic fields.

In order to keep the rotor spinning the rotating magnetic field of the stator must always anticipate the magnetic field of the rotor (Figure 2.3). The rotation is achieved commuting the currents of the electric phases (BLDC motors usually

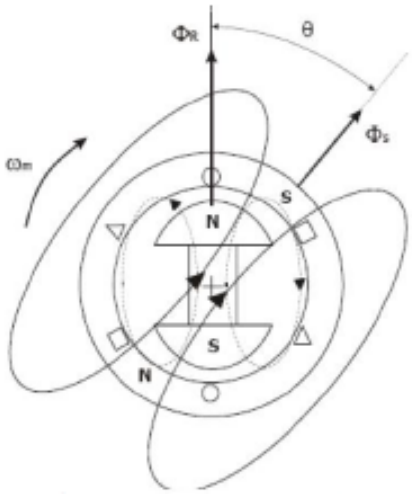


Figure 2.3. Schematics of a BLDC Motor showing the magnetic fields [9]

employ 3-phases configurations) which power the stator coils. This is done using an *inverter* (also called static commutator) commanded by a signal regulated by the angular position of the rotor, measured by an Hall-sensor.

The lag between the magnetic fields is kept constant. For this reason BLDC motors are considered a particular kind of Synchronous AC motors, implying that they are powered by DC current, but is is converted to AC current by an inverter in order to power the stator phases. A design that uses sinusoidal AC current to drive the motor is possible and would produce a more regular torque, but would also be much harder to control than a BLDC motor.

2.2 Stator and rotor

The stator of a BLDC motor is usually a 3-phases configuration (but this is not the only possible configuration) whose phase winding are arranged in either a "delta" or a "Y" (or "star") connection. In "delta" connections line voltage and phase voltage coincide, while in "Y" connections line voltage is equal to $\sqrt{3}$ times phase voltage; this means that the former configurations has an higher current flow in the coils than the latter for the same supply voltage, hence "delta" designs provide higher torque, but they are also less employed than "Y" designs because their commutation is substantially different and more complex, as a single phase cannot be left unpowered at any time during the sequence, due to its connections themselves.

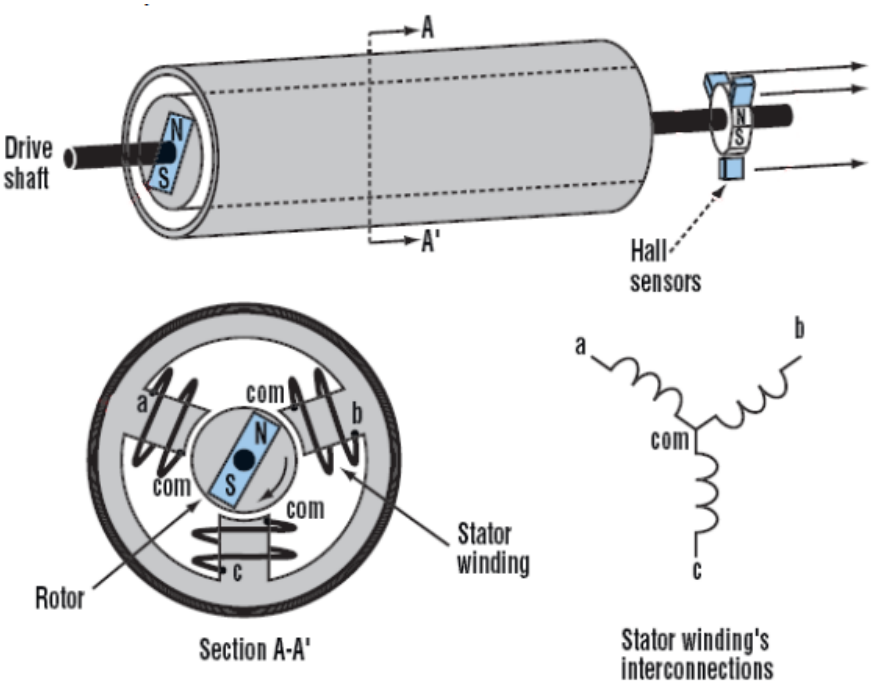
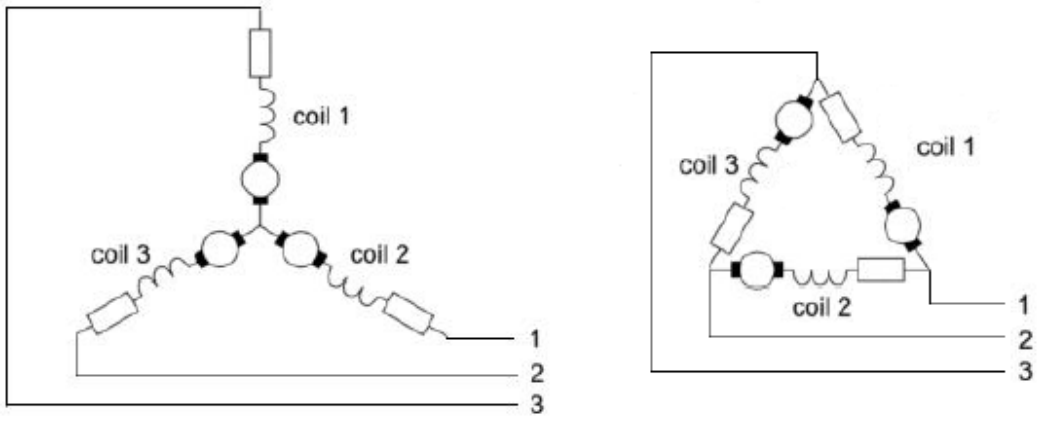


Figure 2.4. Schematics of a simple BLDC Motor [9]



(a) Y/Star connection (b) Delta connection

Figure 2.5. 3-phases configurations [2]

The main task of the structure of the stator is to hold up the windings of

phase coils, which can be coupled to the stator in either a *slotted* of *slotless* configuration (Figure 2.6). In *Slotted* configurations the coils are around the so called "teeth", forming "slots" in the stator; in this configuration the air gap between the stator and the rotor is smaller than in *slotless* configurations, so it has an higher maximum torque. On the contrary, *slottless* configurations have a lower maximum available torque, but the absence of teeth guarantees a lower inductance, that translates into an higher maximum available speed, as inductance represent an obstacle to extremely fast commutations [15].

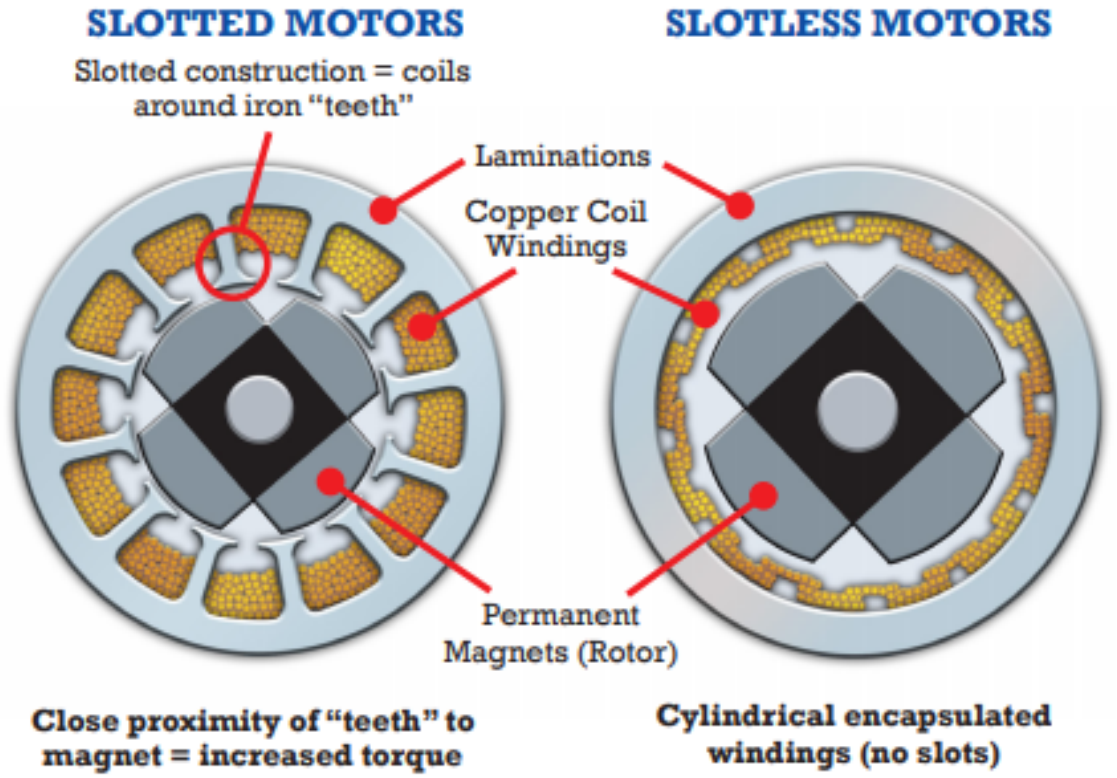


Figure 2.6. Difference between *slotted* and *slotless* stator core configurations [15]

As for the rotors, they consist of permanent magnets, made of either rare earths or Ferrite. In the aerospace sector Neodymium based magnets are the preferred choice because of their high magnetic flux generation capability, which allows to build smaller and lighter motors with the same torque, as torque is directly proportional to magnetic flux and electromagnetic coupling constant.

The magnets can be installed either *on* the rotor (*isotropic configuration*) or *inside* the rotor (*anisotropic configuration*)
The number of poles of the rotor may vary from 2 pair to 8 pair, but poles are always in even number for obvious reasons. A motor with a large number of pairs of poles produces a smoother torque, as there are less "gaps" or "ripples" in the magnetic field as the rotor spins, but also has a lower maximum speed, as the transistors used to commute the electric current have limits on their maximum operative frequency of activation and deactivation.

Material Name	**Ferrite (HF)**	**AlNiCo (AN)**	**SmCo (SC)**	**NdFeB (ND)**
Attraction Force	Good	Medium	Strong	Very Strong
Max. Operative Temperature	200řC	450řC	200řC	80řC
Resistence to Corrosion	Very Good	Very Good	Good	Average
Workability	Impossibile	Diamont cutting, grinding	Impossibile	Impossible
Ease of Demagnetization	Moderate	Easy	Very Hard	Hard
Price	Cheap	High	Very High	Cheap

Table 2.1. Characteristics of common permanent magnets [9]

Lastly, while most motors employ stator-outside-rotor-inside configurations, *outrunner* configurations do exist, albeit only used on small motors.

2.3 Control

The precision of the commutation sequence relies on the precise measurement of the rotor's angular position. As already mentioned, said angular position is measured by an array of Hall-sensors. These sensors are based on the homonymous principle, as they actually measure the variation of the charge across a conductor in a direction orthogonal to the magnetic field, as said variation is caused by the Lorentz Force. As the rotor's magnetic field rotates, the magnitude and direction of the Lorentz Force on the Hall-sensor surface changes, and can be correlated to the angular position of the rotor.
In Figure 2.7 3 Hall-sensors (indicated as H1, H2 and H3) are placed 120ř degrees apart of each other, meaning that the system has a 60ř angular resolution, which is important since in a 3-phases configuration the stator magnetic field reverses its polarity every 60ř of rotation.

The trapezoidal BLDC motor (shown in 2.7) gets its name from the shape of the phase current signal that determines a trapezoidal shape of the Back EMF

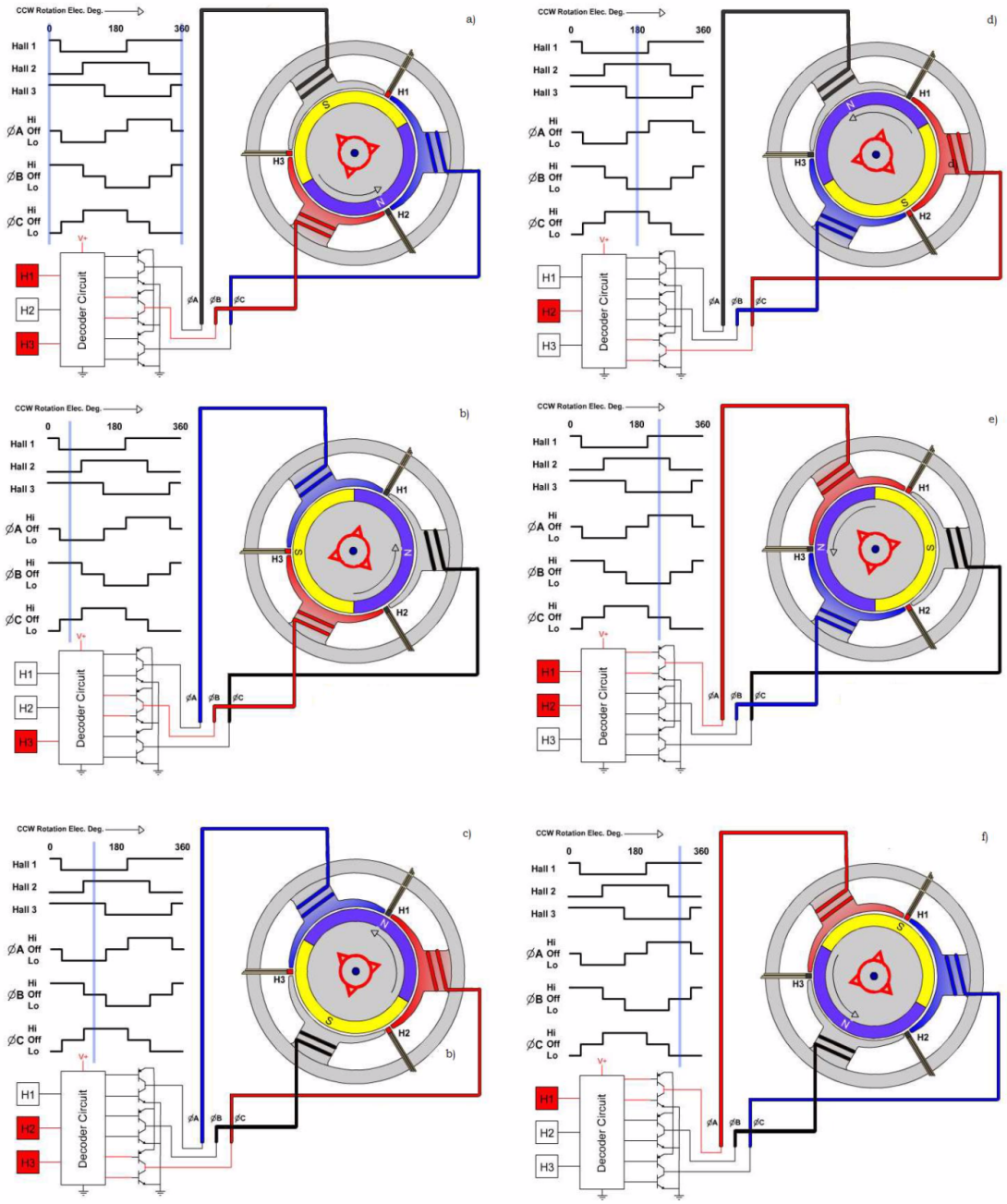


Figure 2.7. Commutation cycle of a 3-phases 2-poles trapezoidal BLDC motor (CCW Rotation) [9]

signal over time in the coils of the stator.

With reference to Figure 2.7, initially the sensors on the south pole of the rotor (H1 and H3) return a Logical 1 signal, while H2, on the north pole, returns a Logical 0 signal. Phase A, B and C are respectively unpowered, powered and shorted to ground. The north pole of the rotor lags 60ř behind the north pole of the stator magnetic field, and a torque that spins the rotor is produced.
As soon as the rotor makes a 60ř rotation, H1 sensor returns a Logical 0 signal. As a result, Phase C is deactivated and Phase A is shorted to ground, so that the north pole of the rotor is again lagging 60ř behind the north pole of the stator, ans so on.
Figure 2.8 shows the electrical scheme of the BLDC motor, highlighting the connection between the phases, the decoder circuit and the switches (transistors). It is noted that all transistors are connected to line voltage and ground, and that each phase is controlled by a pair of transistors, so that the rotor can spin both forward and backward.

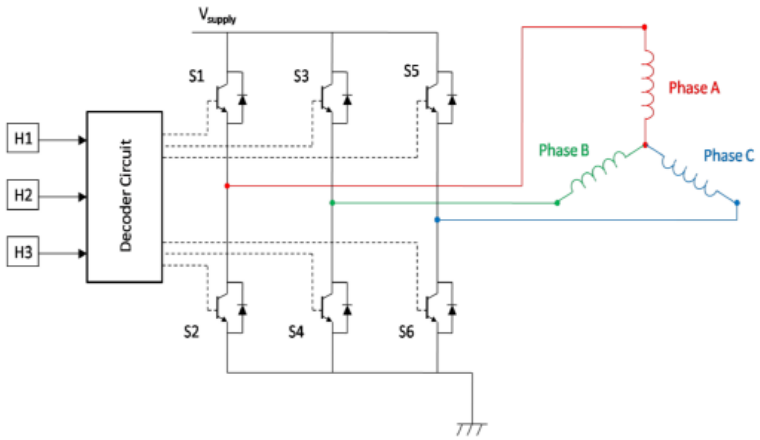


Figure 2.8. Electrical scheme of a 3-phases BLDC motors [18]

Table 2.2 instead shows the motor timing diagram, that's to say, the logic behind the activation of the Hall-sensors, transistors and phases. It is to be noted that each full revolution is composed of six 60ř rotations.

Lastly, different control modes are available depending on the circumstances and the desired control variables.

Position cannot be directly controlled, as it is measured by Hall-sensors in

	Hall-sensors			Transistors						Electric Phases		
	H1	**H2**	**H3**	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**A**	**B**	**C**
I	**1**	0	**1**	0	**1**	**1**	0	0	0	NC	High	Low
II	0	0	**1**	0	**1**	0	0	**1**	0	Low	High	NC
III	0	**1**	**1**	0	0	0	**1**	**1**	0	Low	NC	High
IV	0	**1**	0	**1**	0	0	**1**	0	0	NC	Low	High
V	**1**	**1**	0	**1**	0	0	0	0	**1**	High	Low	NC
VI	**1**	0	0	0	0	**1**	0	0	**1**	High	NC	Low

Table 2.2. Time diagram of a 3-phases 2-poles trapezoidal BLDC motor

order to perform the current commutations, so that same measurement cannot be used to regulate the commutations themselves. Position control is usually achieved using an inner control loop and another control variable. The output of the inner control loop is then used to calculate the output position of the motor, and so the outer control loop is closed measuring the error against the desired position.
Position of the motor is usually calculated integrating speed signal over time.

Most common control modes are speed and torque modes.

2.3.1 Speed Control

Speed control mode is used as the inner feedback loop of a position control loop or when a predetermined speed is directly the chosen control variable, like in hydraulic pumps.

Rotor's angular speed is modified regulating the line voltage. When DC power supply is available, this control can be done simply by using a potentiometer; this method however is inefficient, as the excess power is dissipated via Joule effect on the resistance of the potentiometer, and may not always available, as potentiometers may not allow an extremely fine control, which is often needed in many application.
Another method, enabled by developments in digital circuitry and software integration in hardware, is the Pulse With Modulation (PWM) technique.
In PWM the voltage of the power supply is constant, and so is the amplitude of the electric control signal, a high frequency wave. The duty cycle of this input

signal (carrier) is modulated using a reference (modulating) signal (Figure **??**). At 100% duty cycle the signal is always on (nominal input voltage), while at 0% the signal is always off. All the intermediate voltages (and so speeds) can be achieved changing the duty cycle accordingly.

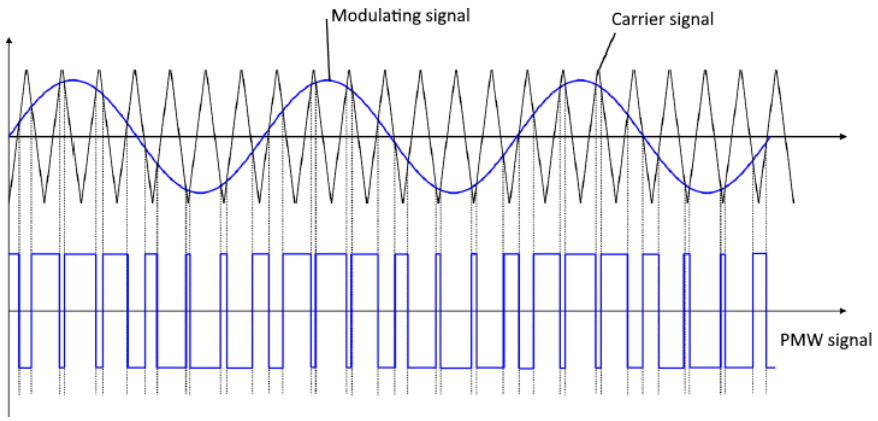


Figure 2.9. Example of derivation of a PWM signal [9]

The output angular speed of the rotor is calculated from the switching speed of the Hall-sensors and the error measured against the desired angular speed is calculated and used as input for the control system.

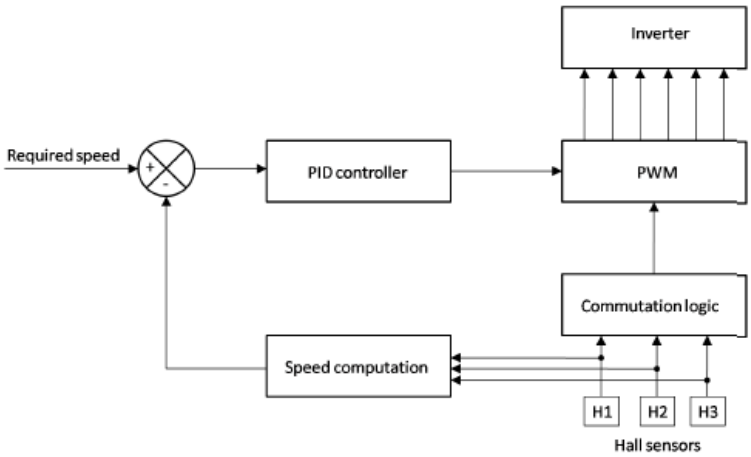


Figure 2.10. Block diagram of a speed control loop [2]

2.3.2 Torque Control

Torque control mode is generally used in industrial environment or as inner feedback loop for position control. In this mode the torque output is constant and the motor speed is changed accordingly to counteract external loads from the user.

An accurate implementation of such control mode is hard since it is dependent upon knowing the precise value of the output torque in every condition, which requires a deep characterization of the motor based on electromagnetic analysis, as the output torque is a function of (among other things) the magnetic flux in the stator coils, a variable hard to model or measure in an operative setting.
Nonetheless a simplified implementation can be achieved introducing the motor torque coefficient k_T, which represents the (direct) proportionality between torque and current. It follows that in this case control is achieved modifying phase currents.
The output torque is computed and compared to its desired value, and the error is fed to the control system in order to change the PWM duty cycle, just like in speed control mode.

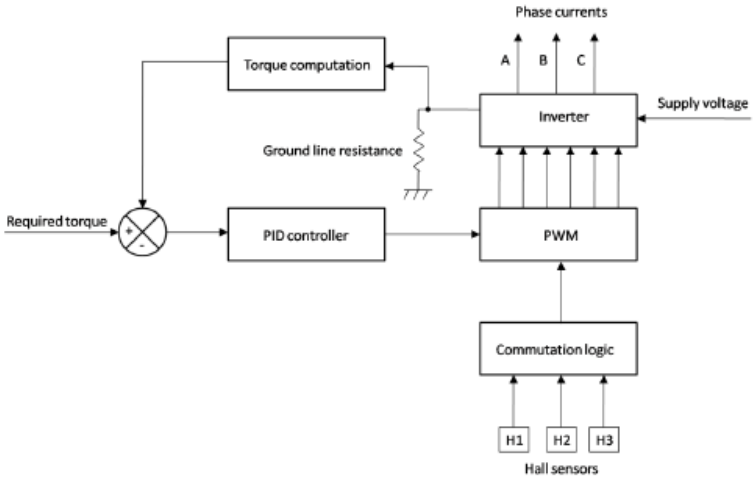


Figure 2.11. Block diagram of a torque control loop [2]

2.4 Protection systems

If a BLDC motor is employed in mission or safety critical applications, a number of sensors and software or hardware limitators must protect it against damages that could be cause by out of envelope operations.
If the motor is exposed to an high line current while it is stuck (the CEMF is null), the electrical current could overheat or short the stator coils and disrupt the power supply or the control electronics, or even demagnetize the permanent magnets of the rotor if the Curie temperature is reached during overheating. To protect from dangerously high currents, the supplied current is saturated to a *peak current* value if during motor start-up an higher current is computed.
With a similar principle the current is saturated to the value of the *maximum working current* if an higher value is computed during normal operations.
As the Hall-sensors are so important for the correct functioning of the motor, the control system must have a logic capable of detecting when they are faulty. Since the values returned by the Hall-sensors can only assume a limited number of predetermined combinations, the implementation of such a logic is relatively straightforward, as any unexpected combination is sign of a faulty Hall-sensor.
The motor must also be protected against *overvoltage* and *undervoltage*.
Overvoltage protection must always be applied as unexpected high voltages can create arcs that may damage the power supply or the control electronics, which are particularly sensible to such things.
On the other hand undervoltage protection is necessary only when the motor is battery powered, and in order to protect the batteries, as they would be greatly damaged if their voltage drops under a certain level.

2.5 Mechanical characteristics

In a BLDC motor the torque is generated by the interaction between the magnetic fields of the rotor permanent magnets and the stator coils. The stator coils themselves are magnetic dipoles surrounded by the magnetic field of the rotor, so, simplifying, the dipole moment of the j coil can be expressed as:

$$\vec{m_j} = NAi_j \cdot \hat{\mathbf{n}}$$

Where N is the number of turns in the coil, A their surface, i_j the phase current in the coil and $\hat{\mathbf{n}}$ the normal versor to the plane of the spires. Introducing $\vec{B}$ as the magnetic field of the rotor, the Torque produced by the coil j can be defines as:

$$\vec{T_j} = \vec{m_j} \times \vec{B}$$

The total Torque can then be calculated as the vectorial sum of the contribute of each stator phase.
Because of the cross product, it is evident that the Torque is maximized when the dipole moment of the coil is orthogonal to the rotor magnetic field, so during commutation the lag between the magnetic field is kept as close to 90ř as possible. The Torque can also be increased modifying the geometric of the coils by either increasing the number of windings or their area (albeit both solutions lead to an overall increase in the dimensions of the motor), using stronger permanent magnets in the rotor (increasing the $\vec{B}$ term of the equations) or increasing the supplied current. However, the quadratic proportion between electric resistance and heat produced via Joule effect poses a limit to the maximum value of the supplied coil current, as overheat would damage the motor. Furthermore, flux density in the permanent magnets of the rotor can be saturated, and no more Torque would be generated in this condition.

On a simplified level, applied current and produced Torque are directly proportional through a constant called *motor torque constant* k_T. On the other hand, due to Faraday's Law, a counter electromotive force is generated on the coil itself by the variation of the magnetic flux generated by the rotor motion. Said counter electromotive force is approximately directly proportional to the angular speed through a constant called *counter electromotive force coefficient*, k_e. It can be proved that the counter electromotive force constant (also called *motor voltage coefficient*) and the motor torque constant are the same, $k_T = k_e$.

As angular speed increases, torque and current decreasing, leading to the following relation:

$$V = Ri + k_T\omega \tag{2.1}$$

And, since $T = k_T i$, rearranging the previous expression to isolate i leads to:

$$T = \frac{Vk_T}{R} - \frac{k_T^2}{R}\omega \tag{2.2}$$

So the output power can be calculated as:

$$P_{out} = T\omega = \frac{Vk_T}{R}\omega - \frac{k_T^2}{R}\omega^2 \tag{2.3}$$

Equations 2.2 and 2.3 (whose trends are plotted in Figure 2.12) are approximations that doesn't account for non linear effects.

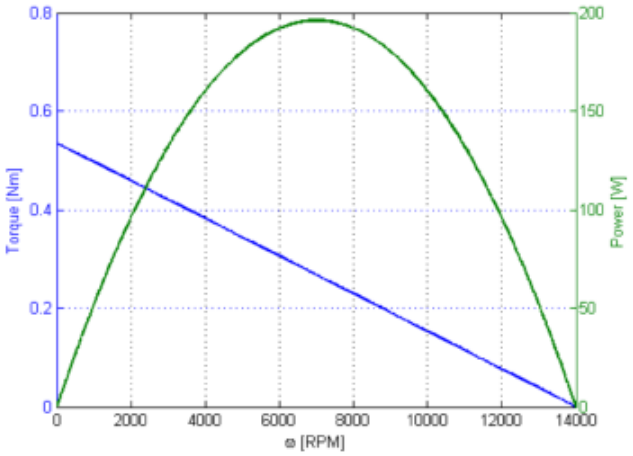


Figure 2.12. Linear speed-torque and power-torque characteristics of a BLDC motor [9]

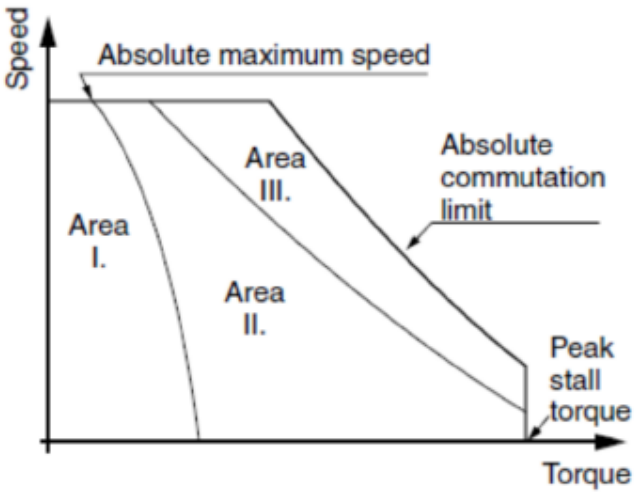


Figure 2.13. Saturations on the ideal speed-torque characteristic of a BLDC motor [9]

Figure 2.13 introduce saturation effects. The maximum speed and torque are capped, but commutation frequency also introduces an operational limit.
A more realistic and complete speed-torque characteristics is presented in Figure 2.14, with efficiency superimposed on the graph.
Maximum efficiency is reached at high RPM: that's the reason why high gear ratio transmission are often used in conjunction with BLDC motors, especially in low RPM applications.

The green line separates the *continuous operations zone* (below the green line) from the *intermittent operations zone* (above the green line but below the blue one). Operating at higher torques than the maximum continuous torque is possible, but only for short periods of time, as higher torque is achieved using higher line current, so prolonged operations would overheat the stator coils and damage the motor.

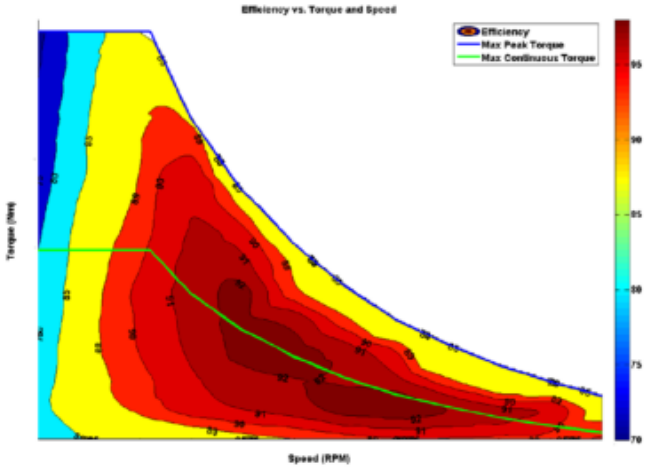


Figure 2.14. Realistic speed-torque characteristic of a BLDC motor [9]

However, a model based on equations 2.2 and 2.3 and the saturation introduces in Figure 2.13 is quite accurate when compared with the realistic characteristics, especially during nominal operations.

Chapter 3

Electro-Mechanic Actuator Model

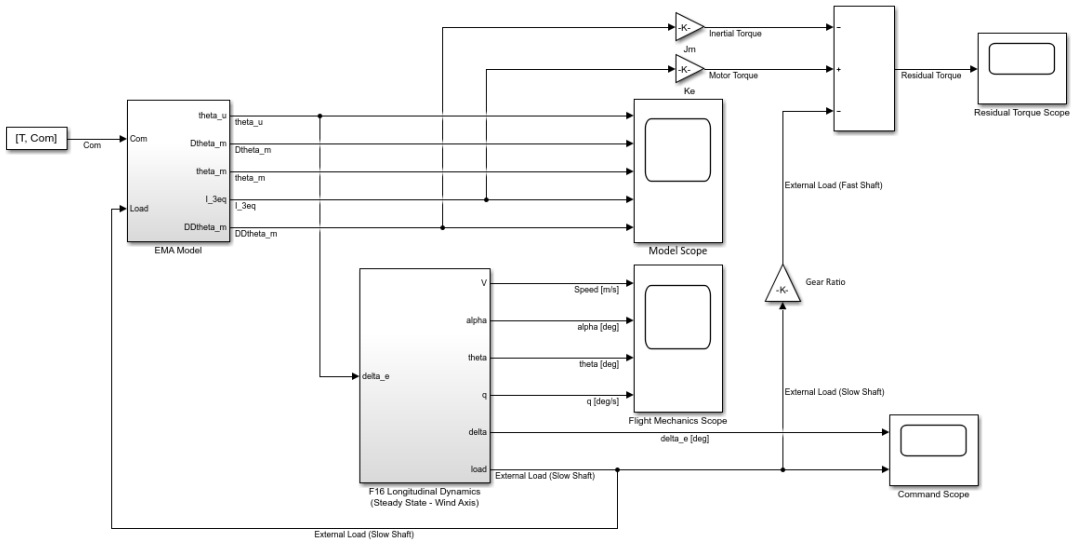


Figure 3.1. Overview of the Simulink System Model

The aim of this chapter is to describe in detail the model of the Electro-Mechanical Actuator used throughout this work.

For practical reasons, the data necessary to train the neural network has been harvested running a Simulink model of the EMA thousands of times. Said model

is both accurate enough to simulate most common fault modes and computationally light enough so that the dataset could be generated in a reasonable amount of time.

Should the hardware validation of this method prove successful, the use of a virtual model would be an important feature of the proposed prognostic method, as it would save the enormous amount of time needed to acquire an adequate amount of data for training with a physical test-bench.

The model of the EMA described in this chapter, as already said, must be computationally light but also accurate at the same time. However, accuracy is relative to the task at hand, and in this case it was possible to introduce simplifications on the electric part of the model, as this work is focused on mechanical faults correlated to mechanical parameters reconstructed (mainly) from a mechanical variable.
Such parameters and variables are little affected by the *micro* behavior of the electric motor, so a *macro* level electric model is more than acceptable for this application.

The model itself is derived from Pier Carlo Berri's Master's Thesis [2] (where it is called *"Monitor Model"*), but the command and mechanical parts of the model have been heavily modified to suit this work. In that same work, the electric model is compared to the extremely detailed BLDC motor model of Matteo Dalla Vedova's PhD Thesis [21] with good results.

The *macro* approach was used for the mechanical part too. This approach was chosen for two reasons: on one hand this makes for a lighter model, on the other hand it better suits the prognostic philosophy. The only appreciable advantage of modeling the system on a sub-component level for the purpose of fault identification is to be able to isolate a fault in a sub-component. In real word applications of the kind of this work, this would be useless unless there was a way to mechanically isolate or replace said sub-component without isolating or replacing the whole component.
For instance, while it is certainly possible to model a ball-screw gear considering the variations in the friction of each single ball across the entire length of their canal and identify a fault in a single or multiple balls, it would be impossible to isolate single balls in case of need, and, during ground maintenance, the whole gear would be replaced anyway as it is easier and safer.
Modeling each component using his "mean" or "equivalent" properties fits better the modern principles of onboard safety and on ground maintenance, while being

the fastest, computationally lighter and more generalizable way to model the mechanical part of the system.

The model represents a 3-phases 4-poles BLDC poles connected to the elevator of an aircraft through a ball-screw reducer. The aerodynamic load on the control surface (hinge moment) is calculated at each timestep through the longitudinal dynamic model of the aircraft, and the whole system is piloted via the elevator deflection and a proportional controller.
The model itself comprises 4 sub-models: the *Controller*, *Electric* and *Mechanical Transmission* sub-models and the *Aircraft Longitudinal Dynamics* Block

The numerical value of the integration timestep of the model must be at least one order of magnitude lower than the smallest characteristic time of any phenomenon modeled. The integration timestep of this model has been set to $10^{-6}s$ as using an higher value the electrical model would incur in numerical limit cycles, as explained in Section 3.2.

The resulting model can simulate $0.5s$ of functioning in, on average, $100 - 150ms$ on modern low-mid range hardware (described in Subsection 5.8.2) using Simulink's *Accelerator* mode. Simulink's *Rapid Accelerator* mode, albeit much faster on a single run, wasn't used since it would prevent the use of the *Fast Restart* option when launching multiple simulations, which radically reduces the total simulation time when thousands of simulations are required.

Constant and coefficient of the system are shown in Table 3.1.

Controller Parameters			
Proportional Gain of the Controller	G_P	10^5	–
Proportional Gain of the PID	GAP	$5 \cdot 10^{-2}$	$\frac{Nms}{rad}$
Electric Parameters			
Number of Stator phases	PS	3	–
Number of Rotor pairs of poles	PR	2	–
Phase Resistance	R_z	2.13	Ω
Phase Inductance	L_z	$7.20 \cdot 10^{-4}$	H
RL Time Constant	τ_{RL}	$\frac{L_z}{R_z}$	s
Motor Torque Coefficient	k_T	0.0752	$\frac{Nm}{A}$
Motor Voltage Coefficient	k_e	0.0752	$\frac{Vs}{rad}$
Maximum Torque	$T_{M,max}$	1.689	Nm
Maximum Voltage	V_{max}	48	V
Maximum Current	I_{max}	22.5	A
Mechanical Parameters			
Gear Ratio	τ	$\frac{1}{500}$	–
Moment of Inertia (on motor shaft)	J_M	$2.5 \cdot 10^{-5}$	$kg \cdot m^2$
Viscous Friction Coefficient	C_M	$5.172 \cdot 10^{-5}$	$\frac{Nms}{rad}$
Backlash	BLK	$5 \cdot 10^{-3}$	rad
Static Friction Torque Coeff. (nominal)	FSJ	0.1	–
Dynamic Friction Torque Coeff. (nominal)	FDJ	0.05	–
Static-to-Dynamic Friction ratio (nominal)	FSD	2	–
Efficency under Aiding Load (nominal)	η_A	0.6	–
Efficency under Opposing Load (nominal)	η_O	0.85	–

Table 3.1. Electro-Mechanical Actuator Model Datasheet

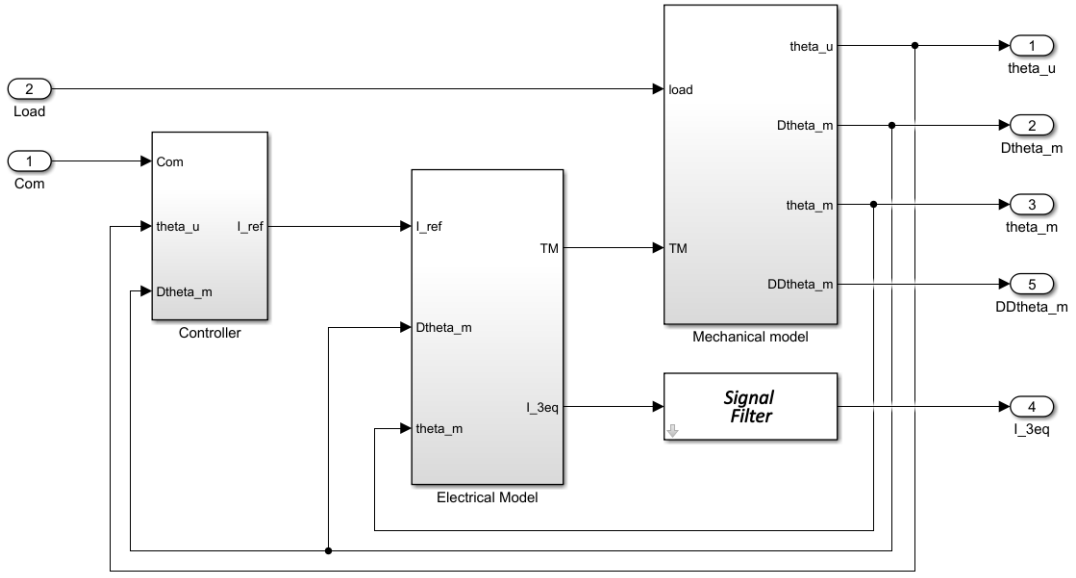


Figure 3.2. Overview of the EMA model

3.1 Controller sub-model

The Electrical sub-model is current-piloted, but the pilot of the aircraft actually controls the elevator position, so the pilot current (reference current I_{ref}) is derived from the error signal calculated as the difference between the desired position of the elevator (the Com port) and the actual position of the user, indicated as θ_u. This error, through the proportional gain G_P, is used to compute a reference angular speed (limited to its maximum allowed value by a saturation block) that is confronted with the actual angular speed of the motor ($Dtheta_m$ line).
The newly obtained error goes through another gain, GAP, which acts as the proportional part of a PID controller; the signal is now a reference torque. No integrative or derivative parts of a PID are present, as the purely proportional logic satisfies the control requirements by itself.
The reference torque signal is divided by the Motor Torque Coefficient k_T to obtain a reference current I_{ref} (saturated to its maximum value I_{max} as a protection system would do).

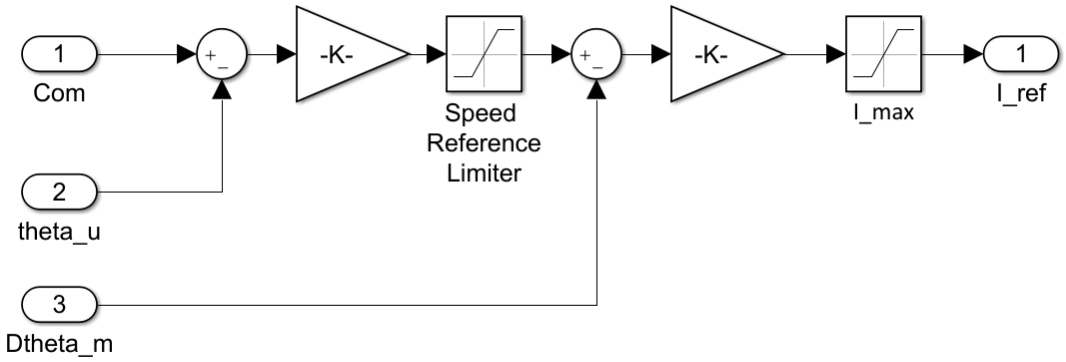


Figure 3.3. Controller sub-model

The desired elevator position Com comes from a *From Workspace* block. The command position is indeed generated outside the model before the simulation starts by a Matlab script that generates a random continuous pseudo-sinusoidal input as an array of the actual desired command values associated with another array T that contains the timestamp associated with each element of the Com vector.
Since the integration timestep of each simulation is set to $10^{-6}s$, the associated T array is actually a linear space from the start time to the end time of the simulation with 10^{-6} steps.
In case Simulink integrator uses a different (adaptive) timestep, Com and T are used to interpolate the appropriate Com_1 array, but this is not the case.

More information on the philosophy used to generate the random command are present in Section 5.6.

3.2 Electrical sub-model

The BLDC motor is modeled as an equivalent single phase current-piloted DC motor. The error between the reference current I_{ref} from the controller and the actual current in the stator coil is passed through a *signum* block and the resulting signal is multiplied by the line voltage of the DC power supply to obtain the supplied voltage V_a.
The interaction of the error with the *signum* block closely resembles a PWM

control logic, where the frequency of the carrier wave is proportional to the integration timestep of the model.
The voltage that powers the stator coil is the difference between the reference voltage previously computed and the Counter Electromotive Force e of the motor, computer multiplying the actual speed of the motor (computer in the Mechanical transmission sub-model) by the Motor Voltage Coefficient k_e.
The stator coil is modeled ad a RL circuit with a first order transfer function that represents the following phasor relationship:

$$\frac{I_a(s)}{V_a(s) - e(s)} = \frac{K_a}{1 + \tau_{RL} s} \tag{3.1}$$

Where $K_a = \frac{1}{R_a}$ is the motor gain, $\tau_{RL} = \frac{L_z}{R_z}$ is the time constant of the modeled RL circuit and I_a is the aforementioned difference between I_{ref} and the actual stator current, which is obtained as the output of this block,
The actual stator current is then multiplied by the Motor Torque Coefficient k_T to obtain the torque generated the motor (indicated as T_M), that is then limited to its maximum value $T_{M,max}$ by a saturation block that represents the saturation of the magnetic flux.

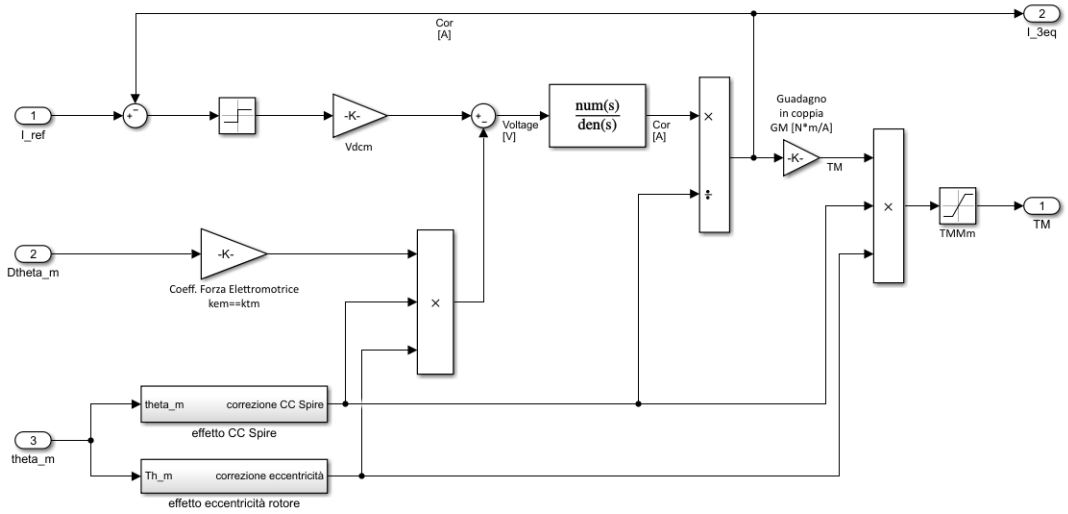


Figure 3.4. Electrical sub-model

3.3 Mechanical transmission sub-model

The Mechanical transmission sub-model is a second order model with a single degree of freedom.
Viscous and inertial effects are considered, and non linear effects like backlash are considered too.
Coulomb friction torque is evaluated through a load dependent Borello model with efficiency, described in Subsection 3.3.1.
The angular acceleration of the motor's shaft $\ddot{\theta}_m$, indicated as $DDTheta\_m$, is computed from its torque and the signal is then integrated two times to obtain the motor's shaft angular speed $\dot{\theta}_m$, indicated as $DTheta\_m$, and its angular position θ_m, indicated as $Theta\_m$.
Speeds and positions are saturated as needed according to the maximum possible speed and the mechanical stop-ends.
Finally, the angular position of the user's shaft θ_u is evaluated applying the gear ratio to the angular position of the motor's shaft.

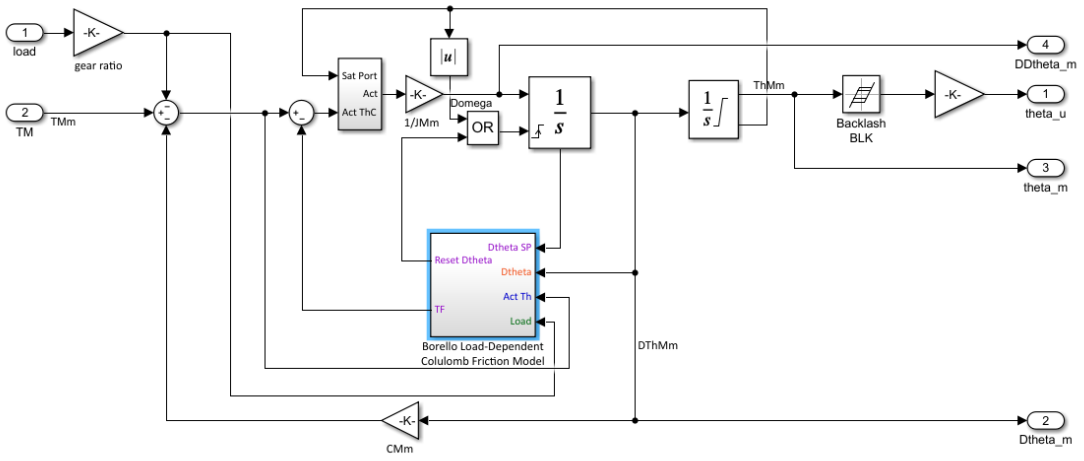


Figure 3.5. Mechanical transmission sub-model

3.3.1 Friction model

The friction model must be able to model both load dependent and load invariant friction torques, as friction coefficients are a telling sign of mechanical problems

in the actuator, which are the main focus of this work.
The description of the friction model will be done in the case of a friction force, but it is exactly the same minus the terminology in case of a friction torque.

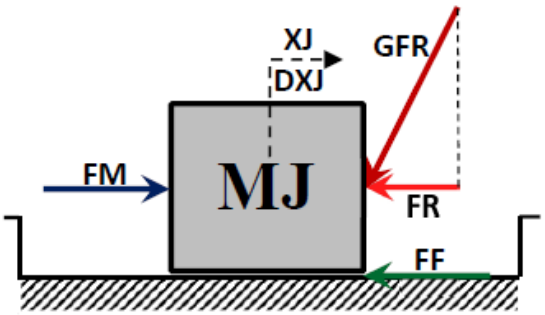


Figure 3.6. Reference scheme - Friction Force [4]

The friction model employed in the Mechanical transmission sub-model is based on Borello's Friction Model [5], which is itself an evolution of Coulomb's dry friction model.
According to Coulomb's hypothesis, the load invariant friction force FF is equal to its static (sticking) value FSJ if the driving force FM isn't greater than FSJ itself, and is equal to its dynamic (slipping) value FDJ if the contrary is true. In the former case, the model must not set the body in motion is $FM < FSJ$, as it would be a violation of Newton's Third Law of Motion; in the latter case the body is set in motion, so the model must take into account whether the speed $\dot{x}$ is null or not.
This translates into the following system of equations:

$$FF = \begin{cases} FM, & \dot{x} = 0 \wedge |FM| \leq FSJ \\ FDJ \cdot sgn(FM), & \dot{x} = 0 \wedge |FM| > FSJ \\ FDJ \cdot sgn(\dot{x}), & \dot{x} \neq 0 \end{cases} \tag{3.2}$$

This situation is visualized in Figure 3.7.

Following the derivation of the load dependent model in Reference [4], the load dependent friction forces are the accounted for considering the total friction force FF as the sum of the aforementioned load invariant part (FSJ or FDJ) and a load dependent part.
The load dependent friction force is proportional to the load FR through the

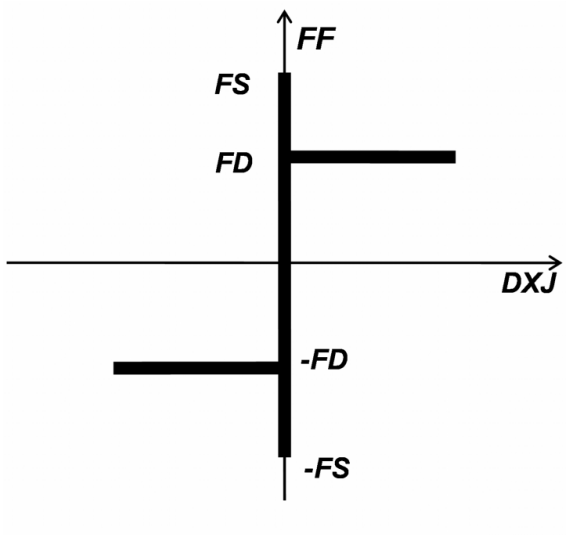


Figure 3.7. Borello Friction Model [4]

means of the efficiency of the transmission, but it must account for the verse of the load compared to the driving force.
When the load acts in the same direction of the motion, the load is *aiding* the movement, while, on the contrary, the load is *opposing* the movement when it acts in the opposite direction of the motion.
The Opposing Efficiency η_O and the Aiding Efficiency η_A of the transmission are respectively defined as the ratio between resisting and driving power and its inverse.
The total friction force in dynamic conditions is then calculated as:

$$FF = \begin{cases} FDJ + (1 - \eta_A) \cdot |FR|, & \text{under Aiding load} \\ FDJ + (\frac{1}{\eta_O} - 1) \cdot |FR|, & \text{under Opposing load} \end{cases} \tag{3.3}$$

The total friction force in static conditions is calculated, introducing the Static-to-Dynamic Friction Ratio FSD, multiplying the dynamic value by FSD.

Figure 3.8 shows the final implementation of the friction model in Simulink. The nomenclature is that of a torque load, angular positions and speeds and friction torque.

The model is able to handle the following eventualities:

- Mechanical element initially sticking that keeps sticking
- Mechanical element initially sticking that is set in motion

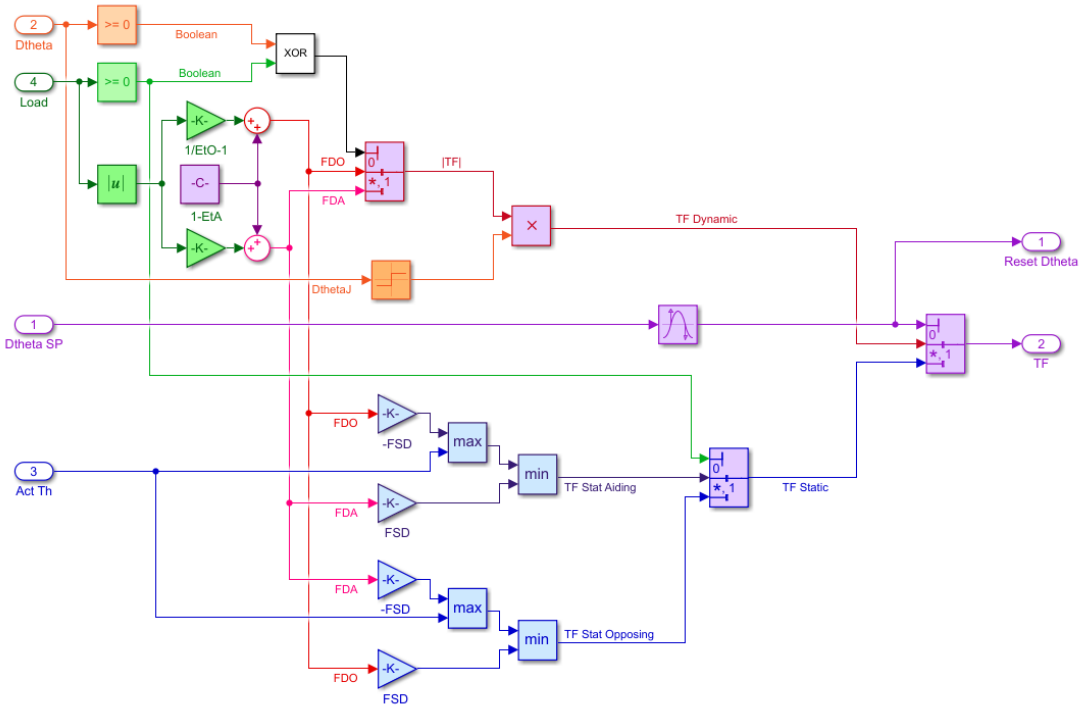


Figure 3.8. Load dependent Borello Friction model with Efficiency on Simulink

- Mechanical element initially slipping that keeps slipping in the same direction
- Mechanical element initially slipping that stops
- Mechanical element initially slipping that reverses its direction

This model was developed with ease of simulation in mind. As a result, it is much faster than other models such as Quinn's of Karnopp's and it doesn't include non-physical parameters.

It is to be noted that the value of η_O must be comprised between 0 and 1. Negative values would mean that the friction aids the motion.
The value of η_A could be negative, but must always be less than 1, and is actually a measure of the irreversibility of the mechanical system. A value of η_A between 0 and 1 means that the transmission is reversible. For $\eta_A = 0$, the friction force is equal and opposed to the load and their combined effect is null. Finally, for $\eta_A < 0$ the system is irreversible and the resulting friction opposes the action of the motor [4].

Furthermore, since it would be relevant in Section 5.4, a relation between η_A and η_O can be established through the Gear Ratio τ [3]:

$$\eta_A = \frac{2\tau^2\eta_O - \tau^2 + \tau}{\tau^2\eta_O + \tau - \eta_O + 1} \tag{3.4}$$

As one would expect, for high values of the transmission ratio, η_A approaches

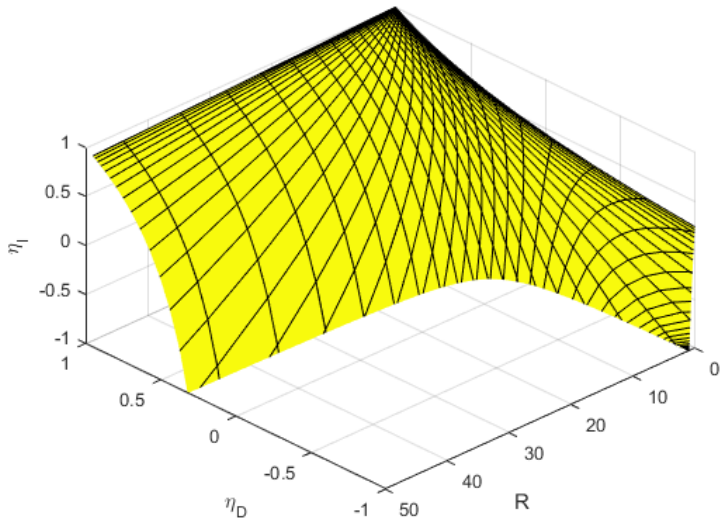


Figure 3.9. Plot of Equation 3.4 [3]

zero.

3.4 Aircraft longitudinal dynamics

The load on the control surface and its actual angular position are evaluated with a State Space model of the linearized longitudinal dynamics of an aircraft.

$$\begin{cases} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{cases} \tag{3.5}$$

The state vector x contains the variation of the Airspeed, Angle of Attack, Horizontal Euler Angle and Pitch Rate from the initial conditions, while the output vector u also contains the variation of the elevator deflection $\Delta\delta_e$ and its

hinge moment ΔM_{H_e}.

$$x = (\Delta V,\, \Delta\alpha,\, \Delta\theta,\, \Delta q) \tag{3.6}$$

The block uses the actual elevator deflection δ_e, and the block $[F16.x0(5)]$ in necessary to update the initial position of the elevator at each step.

The matrices A (*State Matrix*), B (*Input Matrix*), C (*Output Matrix*), D

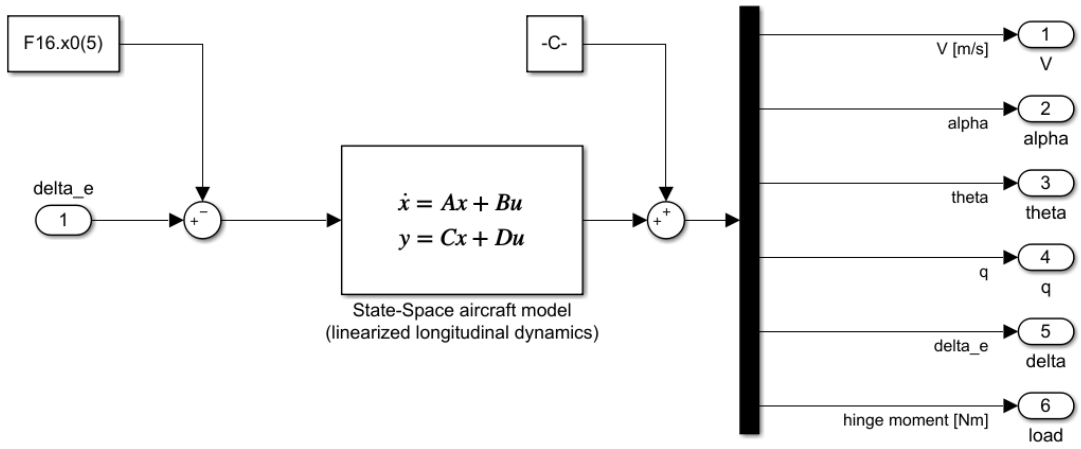


Figure 3.10. Aircraft longitudinal dynamics block

(*Feedforward Matrix*) are populated with the aerodynamic, stability and control derivatives of the aircraft and the control surface.

Chapter 4

EMA Fault Modes

As already mentioned in the Introduction 1, EMAs in aerospace are currently relegated to non-safety critical applications. One of the reasons for this is that EMAs are still a relatively new technology in this field and there is still a lack of literature about their use and the understanding of the effects of their combined fault modes are not fully understood as of yet.

Nonetheless, the interest for this technology is extremely high because of its potential, and EMAs are gaining more and more traction as a result. On modern aircraft EMAs are commonly employed to actuate trim-tabs, spoilers and airbrakes.
On A350 and A380 they are also used to control slats and flaps, and all new generation Airbus aircraft employ EMAs as backup for the Electrohydraulical actuators of the primary flight controls.
Boeing, on the other end, also uses EMAs to actuate the landing gear's brakes on their B787, in addition to spoiler and slat control.
In the military field EMAs are used to a greater extent to cut weight and consequences of damage to the hydraulic system, and their use as primary flight controls actuators is being considered for G^{th} generation fighter aircraft together with the more compact Electrohydrostatic actuators.

It follows that today the employment of EMAs must be coupled with a strong testing campaign, to fully characterize the system and their possible fault modes and their interactions, but also with a reliable monitoring system and a comprehensive maintenance plan tailored to the system.

EMA fault modes can be generally attributed to four main categories that

will be described in the next sections: *Motor Faults* 4.1, *Mechanical and Structural Faults* 4.2, *Electrical and Electronic Faults* 4.3, *Sensor Faults* 4.4.

This work focuses on mechanical faults, of which changes in the characteristics and performance of the reducer and the transmission are a telling indicator.

4.1 Motor Faults

Motor faults are among the most common fault modes for EMAs. Due to the fact the BLDC shafts spin at thousands of RPMs, the motor is subject to vibrations and inertial loads that can damage the bearings and the permanent magnets of the rotor.
Furthermore, excessive heat is another concern due to the compact size of typical BLDCs and their lack of cooling systems. High temperature can damage wire insulation, change electrical resistance in the coils and even demagnetize the permanent magnets of the rotor if the Curie temperature is surpassed.

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

Table 4.1. Motor Faults modes [2], [16]

4.2 Mechanical and Structural Faults

Mechanical and structural fault modes are numerous and they mostly affect the reducer and the transmission. Proper monitoring is necessary as the ambient in which EMAs operate is extremely demanding and mechanical failures can have dangerous consequences on nearby or mechanically connected systems.
These faults are usually caused by excessive loads, out-of-envelope operation,

manufacturing defects, improper maintenance, general wearing and lack of lubrication.

Component	Fault	Failure	Probability	Criticality
Screw	spalling	vibrations, metal flakes	5	3
	wear/backlash	backlash	7	3
Nut	spalling	vibrations, metal flakes	5	3
	backlash	seizure, disintegration	7	3
	degraded operations	seizure, disintegration	3	5
	binding/sticking	seizure, disintegration	3	3
	bent/dent/warp	seizure, disintegration	1	5
Ball returns	jam	seizure, disintegration	5	8
Bearings	spalling	vibrations, metal flakes	5	3
	binding/sticking	seizure, disintegration	2	4
	corrosion	vibrations, metal flakes	2	5
	backlash	vibrations, disintegration	7	3
Piston	cracks	structural failure	1	10
Dynamic seals	wear	structural failure	4	6
	structural failure	same	3	8
Static seals	structural failure	same	2	8
Balls	spalling	vibrations, metal flakes	5	3
	wear	backlash	7	5
Mountings	cracks	complete failure	1	7
Lubricant	contamination	seizure, disintegration	8	5
	chemical breakdown	seizure, disintegration	4	5
	run-dry	seizure, disintegration	3	10

Table 4.2. Mechanical and Structural Faults modes [2], [16]

4.3 Electrical and Electronic Faults

The main causes of electrical and electronic faults are overheating, insulation or dielectric degradation, overcurrents and over and under voltages, out-of-envelope operations and general wearing.
They affect mainly the power and control systems of the actuator.

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

Table 4.3. Electrical and Electronic Faults modes [2], [16]

4.4 Sensor Faults

Sensor faults consists in erroneous readings from the sensors that measure the angular position of the rotor (Hall-sensors) or the sensors that measure, for instance, the temperature of the motor for the protection system.
Since the proper functioning of the system rely on the sensors, more sensors than needed must be used and employed as same- type hot redundancy.
Sensor faults are sub-categorized into *bias*, *scaling* and *drift* faults.

4.5 Friction Faults

Friction fault modes can be described as a sub-set of the mechanical and structural fault modes described in Section 4.2 caused by the effect of an increment (or decrement) of the friction between two or more components.
Friction causes mechanical wearing, which in turn is responsible for the deterioration of mechanical parts and their surfaces and, eventually, the production of metal or plastic flakes.
As friction in the system increases the motor requires more current to sustain the same level of output torque, reducing the overall power efficiency of the system or even damaging electrical and electronic components due to over-current or excessive heat.
Excessive friction can eventually jam the actuator, with possibly catastrophic consequences.

4.5.1 Modeling Friction in the transmission

As already stated, this work is interested with mechanical faults in the transmission of the EMA. For this reason friction in the transmission is modeled as already described in Subsection 3.3.1, and the Simulink model of the transmission is shown in Figure 3.5, with the friction model block shown in Figure 3.8.

This modeling relies on using values for the transmission efficiencies (η_A and η_O) and its friction characteristics (FSJ, FDJ and FSD) already explained in Subsection 3.3.1. The drift of one or more of these parameters from its nominal values is, among other things, the early effect of wear and degradation of the mechanical components. For instance, a superficial damage to a moving part can increase the friction value, while the wear of a pre-loaded sealing can decrease it [3].

4.6 Noise

In engineering noise could be described as *"a general term for unwanted (and, in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing or conversion"* [20].
It is impossible to eliminate noise as a physical phenomenon, but its effects can (and in many applications *must*) be limited. It can originate from mechanical vibrations, environment background, thermal stress, RF pollution and many other sources, in a process that is almost always related to the fact that every component doesn't behave as its ideal self.

Noise management in electrical system is extremely important, as it could drive to a misleading representation of the system status, which in turn hinders the capabilities of the control system and, in general, the performances of the system. For this reason, electrical systems must incorporate filter elements.

After reading Chapter 3 one may argue that, by design, the Simulink model produces filtered data (although *not noisy* would be a better description), while raw data collected from a real EMA would necessarily have some amount of noise.
In reality, de-noising and filtering would be necessary pre-processing steps for the training data anyway, in order to avoid over-fitting of the neural network to the noise introduced by the measurement itself. The same is true for the data used to test the neural network and the data that would be used in a real scenario once the neural network is deployed to increase the accuracy of the predictions.

Chapter 5

Methodology

The aim of this chapter is to provide all the necessary information on the methodology used to provide the training data and train the neural networks. The dataset used to train the neural network of each study case reported in Chapter 6 is tailored to its application, but they all share large similarities since all the study cases share the same basic methodology and all datasets are generated running the same model. Specific differences are highlighted in this Chapter or in Chapter 6's Sections when relevant.

5.1 Applications of interest

The main study cases of interest for this work are of two kinds:

- Use of a pre-determined elevator command sequence
- Use of a generic random elevator command sequence

It should be noted that the distinction between the two main cases is based upon the nature of the elevator command sequence(s), or, for short, the elevator commands. It is not a coincidence: in both cases each simulation uses a different set of random values for the monitored parameters that the neural network will then try to estimate, but in the first case each simulation uses the same elevator command, while in the second case each simulation uses a different elevator command. This has implications with regards to the generalization of the neural network, and each case mimics a different real life scenario with different applications.
This is because in real aircraft (as in the model) the external load on the elevator (or, in general, the control surface actuated 0by the EMA) is determined also by the commanded position of the elevator, so the elevator command is effectively

an additional variable to the problem and is also related to the residual torque and other variables that influence the actual position of the elevator over time. This variable can be considered, as in the case of simulations with different random commands, or one could find a situation where the command doesn't influence the results (i.e. the neural network works properly only when the input data is generated with the same command sequence it is trained upon), as in the pre-determined elevator command sequence case, but, as already said, the two cases have different applications.

5.1.1 Same predetermined command case applications

Specifically the pre-determined elevator command sequence case could be applied to develop a routine that is to be executed either on ground before of after every flight or during a test flight, in both case executing always the same pre-determined command sequence, upon which the network is trained.
This could be used to evaluate the health status of the monitored components when such need arises, in other words, to assess whether or not maintenance is necessary on a part without the need to disassemble the system and inspect the monitored components. Furthermore, RUL prediction algorithms benefit from knowing the actual health status of the component often, as it could be used to adjust the predictions and asses whether or not a component is wearing as expected or if it is degrading faster than expected.

5.1.2 Multiple random commands case applications

On the other hand, having a neural network that is able to estimate the value of the monitored parameters from data generated with any command is useful for real time application during regular flight. It could be useful to assess whether or not the component degraded or is degrading faster after unexpected events such as dust ingestion, and it could also be able to detect incipient faults and implement a logic that is able to isolate a system during flight before the fault occur; this would lead to an increase in safety and the possibility to hot-swap components or systems that would be hard or even impossible to hot-swap after the fault has occurred.

5.2 Use of a Neural Network

The use of artificial neural networks (ANNs) for regression in complex problems is more and more common, as they offer the flexibility that "classical" regression algorithms lack. ANNs are capable of finding correlation between their inputs

and outputs even when the best regression equation for the data is not known or when it is not yet known if the data are correlated at all [11].

In line with other studies, such as the already referenced *Innovative Actuator Fault Identification Based on Back Electromotive Force Reconstruction* [17], shallow artificial neural networks in feed-forward configuration were employed for this work; shallow meaning that the ANN only has a single hidden layer, and feed-forward configuration means that the connections between the neuron don't form cycles, and information moves in a single direction from input nodes to output nodes and the output is determined only by the current input, unlike, for instance, recurrent neural networks.
The resulting architecture is fairly common in regression problems and the accuracy can be improved increasing the number of input nodes or hidden nodes or even adding more hidden layers.
Figure 5.1 shows a simple shallow feed-forward ANN, with inputs, neurons in

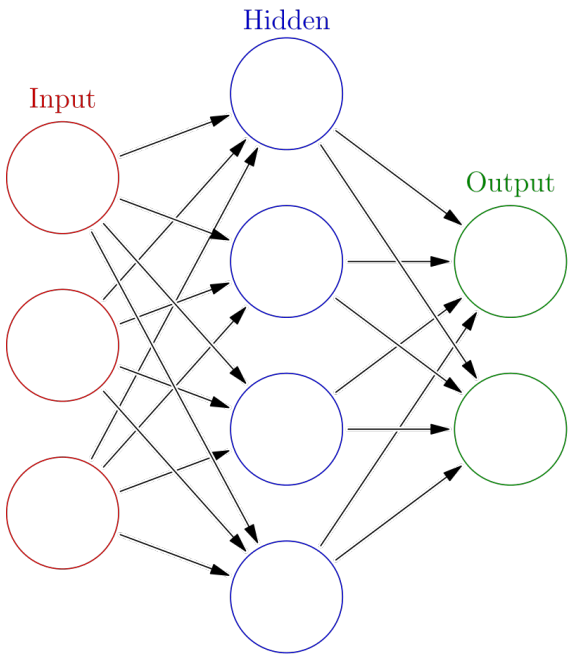


Figure 5.1. Simple shallow feed-forward neural network [7]

the hidden layers and outputs. The network is fully connected, as each node of each layer is connected to all the nodes of the following layer.

Typically the training cycle will continue until the optimal regression coefficients (the goal performance is often specified via a desired Mean Squared Error) are found, a limit number of cycles is reached or the networks fails too many validation checks: validation checks are performed after every training cycle with data that is not used for training to check whether or not the performance (usually in terms of Mean Squared Error) of the network on random data is still good. If too many validation checks are failed consecutively (i.e. there is a drop in the MSE of predictions on non-training data), it probably means that the network is over-fitting on the training data and further training cycles won't be beneficial.

A typical ANN requires a large number of sets of known inputs and a set of known targets for its training.
Since the objective of this work is to evaluate the health status of a series of components estimating the value of a series of parameters, such parameters (monitored or variable parameters from now on) are at the same time the targets of the neural network and the variable parameters of the simulations used to generate the dataset.
Each simulation runs with the same settings except for a series of parameters that are different for each simulation and are saved beforehand since they will be used as targets for the neural networks. After each simulation a number of outputs of the system is saved to be used as input for the neural network.
In short, the value of the monitored is estimated by the neural network through the reconstruction of a number of signals of the system. Targets of the ANN are describer in depth in Section 5.4, while inputs in Section 5.5.

It was decided to program and train the ANN using MATLAB and its Deep Learning Toolbox, as the software offers a variety of options at a reasonable performance level, and simplifies the passage of data from the Simulink simulations to the ANN.
The generic final architecture for this application is presented in figure 5.2. Since

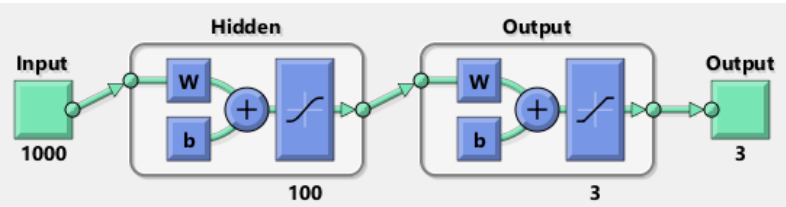


Figure 5.2. Neural Network architecture for one of the study cases

the inputs are technically a time sequence while the outputs are time invariant, it was decided to use a number of inputs nodes equal to the number of timestep of the sequence. However, to reduce complexity and training times, the number of selected timesteps was reduced through interpolation. For instance, due to the integration timestep of the model, $0.5s$ of simulation equate to 50000 timesteps of a variable, one every $10^{-6}s$. Reducing the number of timestep to 2000 means that the signal is interpolated so that now each timestep is $2.5 \cdot 10^{-4}s$.
When more than one input variable was used (for instance residual torque and external load) as input, the number of input nodes is actually the number of timesteps taken for each variable multiplied by the number of variable.
The size of the hidden layer varies according to the study case, as it is one of the first things to modify when seeking better performances from the network.

Details on these differences are presented in Chapter 6 when necessary, together with the respective results.

In each case, the transfer function of the hidden layer was set to Hyperbolic tangent sigmoid (*tansig*) and the activation function of the output layer was set to Linear Symmetrically Saturated (*satlins*) instead of the default Pure linear (*purelin*) since the maximum and minimum value that each monitored parameter can assume (the boundaries of the variation range) are known, and, should out-of-bounds values be predicted, they would be reduced to the boundary values which, in a real application, would mean that the component is already unsafe to use.
Those are relatively standard transfer function for a regression network.

Values of the monitored parameters are normalized between 0 and 1 using their boundary values as minimum and maximum rescaling values.
This helps when visualizing the results because the nominal values of any parameter would always be 0.5, its range 0 to 1 and absolute difference between predicted and actual values is itself the percentage error on the prediction.

All the ANNs were trained using the Scaled Conjugated Gradient training function (*trainscg*), as the more classical Levenberg-Marquardt Backpropagation (*trainlm*) required too much system RAM.

All the ANNs use a value of 12 for tolerance of consecutive failed validation checks, their performance is calculated as Mean Squared Error (MSE) with a goal value of 10^{-6} and a maximum number of iterations of 1000.

5.3 Use of a Simulink model

In order to train the neural network to identify variations of the value of monitored parameters a large amount of data about the system in working conditions is needed. This data was collected from a mathematical (Simulink) model of the system because this approach holds numerous advantages:

- The model can be run faster than real time, drastically reducing the time needed
- The model can simulate a fine granularity of the values of the control parameters
- The model approach is more generalizable
- The model can be tailored to almost every real piece of hardware
- The simulations doesn't require a physical test-bench

Since the model was developed in the Simulink environment, using a Matlab to modify the parameters, launch the simulation and post-process the data was the obvious choice.
Each simulation was run for $0.5s$, during which all the parameters but the monitored parameters are constant for each simulation. The monitored parameters are constant too for each simulation, but they are randomly generated beforehand around the nominal value of each parameter and a set of randomly generated monitored parameters is assigned to each simulation before the run.

The system has to run for some time since the signal used to estimate the parameters are not time invariant, so the way they evolve over time is an important feature that can be picked up by the neural network during training.

Each dataset is composed of data from thousands of simulations. The simulations where executed in batches using CPU parallel computing with MATLAB's *parsim* function (part of MATLAB Parallel Computing Toolbox). The model output signals of interest (neural network inputs) were extracted from the *Simulink Output Objects* after the process and then sampled as described in Section 5.7. The values of the variable parameters (neural network targets) where stored after their generation, as previously described.
The specific number of simulations used in each case is presented in Chapter 6 together with the results.

5.4 Choice of monitored parameters (Neural network targets)

As this work is concerned with faults of the transmission subsystem (the effects and modeling of which are described respectively in Section 4.5 and Subsection 3.3.1), of the five parameters chosen for monitoring, 3 are related to friction in the transmission and 2 to its performance (the latter can also be seen as measures of other kinds of losses in the transmission). Such parameters are:

- FSJ**:** Static Friction Torque Coefficient
- FDJ**:** Dynamic Friction Torque Coefficient
- FSD**:** Static-to-Dynamic Friction Ratio
- η_A**:** Transmission Efficiency under Aiding Load
- η_O**:** Transmission Efficiency under Opposing Load

Being defined as the ratio of FSJ to FDJ, FSD was treated as a dependent variable.
To reduce the amount of targets to be estimated by the neural network, Equation 3.4 was used to introduce a relationship between η_A and η_O. It was decided to treat η_A as a dependent variable.
So FSJ, FDJ and η_O were used as actual targets for the neural network. Their values used to perform the simulations were generated as already described, and the values of FSD and η_A were calculated (both before the simulations and after the use of the neural network) using the already established equations.
It is to be noted that FSD and η_A weren't used as targets for the neural network since their value is calculated from the predicted value of the actual targets, assuming that the relationships established among these parameters are exact. Using all 5 parameters as targets would violate this assumption, introduced to have realistic values and relative magnitudes for all the parameters, as in this case the relationship between the related would be different from the one assumed as exact.

As already mentioned, the value of the parameters were generated randomly in a range centered in their nominal values (which can be found in Table 3.1). The intervals considered a $\pm 20\%$ variation around the nominal values, as bigger variations would be caught by any simple diagnostic system, as their effects would be extremely visible on a performance level and the faulty system would be isolated anyway. Small drifts from the nominal values are not only of great interest for

prognostic purposes, but also make for a more sensitive neural network.
However, in a real application the variation range on which the ANN is trained would certainly be greater than the safe operations range, as the maximum allowed safe value of a parameter is different from the value for which the component is considered faulty and must be isolated and immediately replaced. As previously stated the use of a Linear symmetrically saturated (*satlins*) transfer function on the output layers ensure that out of range predicted values are reduced to a value that is already considered out of safe operations.

Negative values of η_A were excluded in all but one case, as they signal that the transmission has became irreversible, which is not the case for aircraft's control surfaces, and the faulty actuator would easily be identified and isolated by a diagnostic system.
Negative values of the friction coefficients would be excluded too for obvious reasons.

The value of the gear ratio of the reducer was not used as a parameter/target as variations of this value would be caused by structural failures of the reducer itself, which, again, would cause an evident alteration of the performances, which would be detected by a diagnostic system. It is extremely rare that a fault is capable of changing the gear ratio without causing noticeable disruption in the system.

5.5 Choice of model outputs (Neural network inputs)

The inputs used by the neural network to estimate the targets must contain information about the targets themselves. The neural network tries to find a correlation between the inputs and the targets trough a regression, so, if no such correlation exists or is weak, the predictions of the neural network will be inaccurate or even totally random.
Therefore is important to select as neural network inputs variables that are known to be correlated to the targets. Neural networks and other machine learning techniques are commonly used to discover previously unknown correlations too, but it is generally more optimal to discard input variables that are known to have little to no correlation with the targets, in order to avoid spurious correlations, over-fitting and useless complexity.

In this work, the main input variable (and so the main output signal of the

Simulink model) is the residual torque, defined as the sum of all friction torques in the transmission of the EMA.

The residual torque contains in itself information about the friction. Should friction unexpectedly increase or decrease, the residual torque would change accordingly.
It is important to recall Equation 3.3 introduced in Chapter 3 to express the dry friction torque. For the sake of convenience, the system is reported here, together with its formulation for static friction.

$$TF = \begin{cases} FSJ + (1 - \eta_A) \cdot |FR|, & \text{Static conditions, under Aiding load} \\ FSJ + (\frac{1}{\eta_O} - 1) \cdot |FR|, & \text{Static conditions, under Opposing load} \\ FDJ + (1 - \eta_A) \cdot |FR|, & \text{Dynamic conditions, under Aiding load} \\ FDJ + (\frac{1}{\eta_O} - 1) \cdot |FR|, & \text{Dynamic conditions, under Opposing load} \end{cases} \tag{5.1}$$

It is evident how η_A, η_O, FSJ, FDJ and indirectly FSD are related to the residual torque through the dry friction torque.
Viscous friction torque was neglected as its magnitude is usually negligible in aerospace EMAs. Indeed this was proven true analyzing the model, the viscous friction torque was orders of magnitude lower than the dry friction torque.

It is now clear that the residual torque may be a good prognostic indicator as it holds information about the friction coefficients and efficiencies of the transmission, which themselves are indicators of mechanical wearing and degradation, but it would be useless if the only way to know the instantaneous value of the residual torque was solving Equation 5.1, therefore already knowing the value of the parameters of interest.

However, the following relationship applies to EMAs (and, in general, mechanical transmissions connected to a mechanical user through a reducer):

$$T_M = J_M \ddot{\theta}_m + F_{V_m} \dot{\theta}_m + H_e + T_R \tag{5.2}$$

Where T_M is the motor torque, the m subscript means that the values are reduced to the motor's shaft (fast shaft), J_m is the moment of inertia of the user (reduced to the fast shaft through the gear ratio squared), $\ddot{\theta}_m$ the acceleration of the fast shaft, $\dot{\theta}_m$ its speed, F_{V_m} the viscous friction coefficient, H_e is the external load (hinge moment on the elevator) and T_R is the residual torque. The first term on the right side of the equation is the inertial torque of the elevator.
As already discussed, the viscous friction torque can be neglected.

The motor torque can also be expressed as $T_M = k_{CEMF}I$, where k_{CEMF} is the Motor Torque Coefficient (k_T, usually equal to the Motor Voltage coefficient k_e, also called counter electromotive force constant) and I the stator current. Rearranging Equation 5.2, one can obtain:

$$T_R = k_{CEMF}I - J_M\ddot{\theta}_m - H_e \tag{5.3}$$

That's to say, the residual torque is equal to the motor torque minus the inertial torque and the external load.
The constants (k_{CEMF} and J_m) are known values once the system is characterized. The current I is already measured by the system itself for other purposes (such as piloting the BLDC motor), and the shaft's acceleration $\ddot{\theta}_m$ is known once the shaft's speed $\dot{\theta}_m$ is known, and it is measured by the hall sensors of the BLDC motor. The external moment (hinge load on the elevator) can either be directly measured with pressure sensors on the elevator of be measured by the flight computer or be calculated once the state matrix of the aircraft and its controls is known (i.e. once the aircraft is characterized).
In short, the signal of the residual torque can easily be reconstructed using other already measured signals and some known constants unrelated to the mechanical faults. That's why on paper the residual torque is such a great candidate to be a good prognostic indicator.

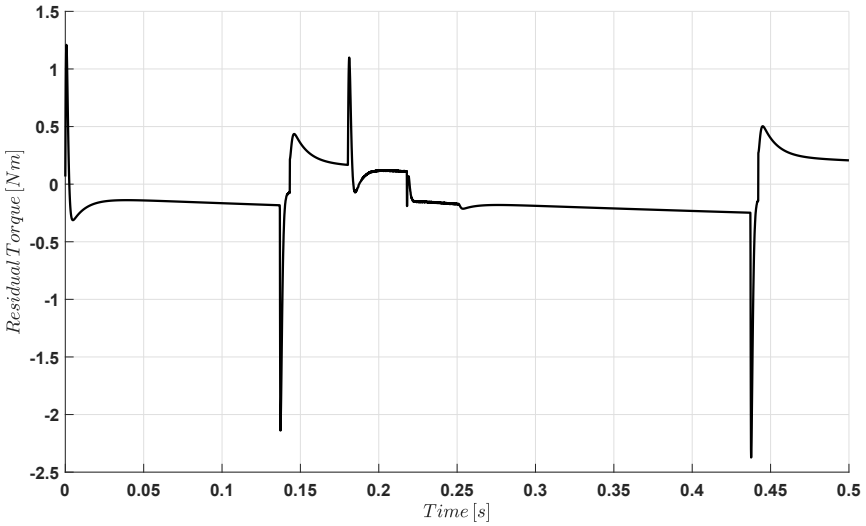


Figure 5.3. Trend of TR over time during a random simulation

Equations 5.2 and 5.3 also show that the friction coefficients and the transmission

efficiencies are indirectly dependent on other parameters, that could improve the accuracy of the network should the sole residual torque prove insufficient. In particular, it was decided to also investigate a combination of residual torque T_R, fast shaft's speed $\dot{\theta}_m$ and external load H_e, as whether or not the speed is null determines if the system is in static or dynamic friction conditions, and a combination of the signs of speed and external load determines whether the system is working under aiding or opposing load.
Also, the use of the external load in combination with the residual torque could lead to a better generalization of the problem, as the external load technically holds information about the commanded position of the elevator, which, as previously discussed in Section 5.1, is a variable of the problem, so it could help in the study case that uses multiple generic commands.

Figure 5.4 shows the trend of T_R, H_e and $\dot{\theta}_m$ as a 3D graph. Graphs of this kind will be called *Input Maps* for the purposes of this work.

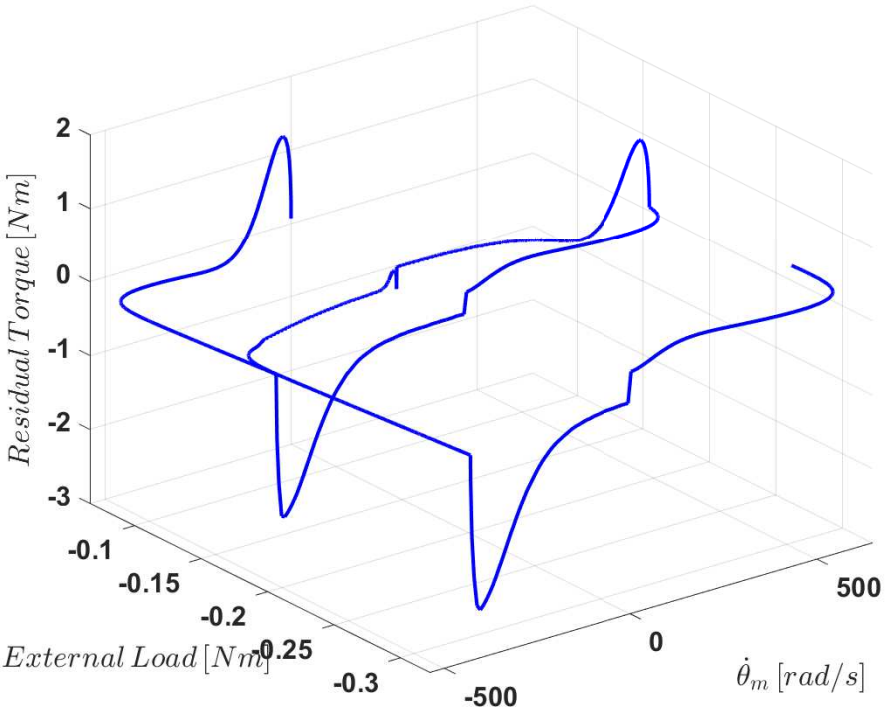


Figure 5.4. Example of Input Map

5.6 Random Command

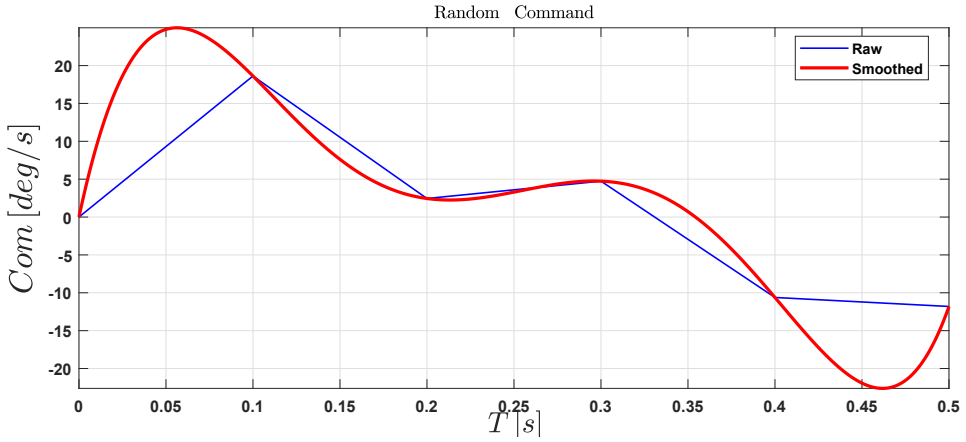


Figure 5.5. Example of Randomly Generated Command

Since the beginning of the study it was decided not to use "academic commands" (ramps, sinusoids, etc...) as command inputs for the model, as they do not necessarily represent real scenarios, and their use could hide problems that would only be caught using realistic commands.
This is particularly true for the real-time in-flight application discussed in Section 5.1, as the network must be able to work accurately using signals generated from any kind of command.
The datasets for this application are generated using a different random command for each simulation.

For the other application (pre-determined command) the specific command is not that important, as long as it can be reproduced on a real aircraft, since it is pre-determined.
The datasets for this application are generated using the same command for all the simulations.
However, it is interesting to know whether or not any kind of command can work, so the same method used to generate the random commands for the real-time application was used to generate the commands, and networks were trained using dataset generated each with a different commands to see if there were visible differences.

The commands in question are elevator commands, expressed as desired angular

position of the elevator in degrees over time in seconds.
They are generated choosing 5 random values for the elevator position, in a range of $\pm 25°$ for the sake of realism. Then, 6 linearly spaced timesteps between the start and the end of each simulation are selected: the neuter value of the elevator position is assigned to the first timestep ($0s$), and the remaining 5 timesteps are assigned to the randomly generated values for the elevator position. Finally, the resulting 6 points are interpolated with a 5th order polynomial function over the actual time array of the simulation.
An example is show in Figure 5.5.

5.7 Sampling and Data uniformity

ANN's input variables don't necessarily have to be physical or unadulterated quantities as long as they keep the information they carry about the targets; because of how neural networks work, it is useful or even necessary to perform pre-processing on the inputs to either reduce the complexity of the network and its training or increase its performances. Nonetheless, the pre-processing operations must be consistent and must be applied to both the training data and the data the network will receive after its deployment.
The appropriate pre-processing operations vary on a case-by-case basis as some operation may work fine for some applications and destroy crucial information in others. For instance, in regression problems it is usually fine to re-scale the input values over an arbitrary interval as long as the operation doesn't alter the proportions among the values of the variables, but neglecting the sign of the values may destroy information about the underlying phenomena and result in a less accurate network.

Sampling, both in time or space depending on the application, is another important aspect of input pre-processing. The time and spacial scale of the data must be coherent with those of the phenomena to be analyzed: an acquisition timestep too big or a sensor grid too loose may hide important quirks, trends and features in the data, while the contrary may result in over-fitting, bad generalization and waste of time.

Data must also be uniform. Typical neural networks operate on arrays of a predetermined size and, unless the network is built to handle such exceptions, the input data arrays must have that same size. Smaller or larger arrays have to be appropriately resized using interpolation, sampling or other techniques.
One particularly useful sampling technique that accomplishes so when the measures are time dependent is to acquire data at predetermined equally spaced

points in time: since the sampling method of the data is now equal for each and every array, the time at which each measure is taken is not a variable of the problem anymore. This can be accomplished either directly through periodic measuring or in pre-processing through interpolation.
A similar approach to reduce the number of input variables consists in doing the same with two non-time variables, for instances interpolating the angular speed of a shaft over a predetermined set of angular positions. However, what has already been said about grid discretization still holds true.

It is also important that the training data is well representative of the data that will be fed to the network once it is deployed: noise must be removed from the data, missing datapoints must be addressed and bad data must be eliminated.

As described in Section 3.1, the timestep of the model is set to $10^{-6}s$ to satisfy the requirements of the controller sub-model, but this timescale is much smaller of the timescale of the mechanical phenomena analyzed, so, for the purpose of this work, based on similar works [16], [17], only one every ten timesteps was initially used in the input arrays.
The first results showed that this operation, initially done to reduce the shear memory size of the data, wasn't harmful for the ANNs. The timesteps were then further reduced as previously mentioned in Section 5.2. Since this varies from case to case, more specific information is reported in Chapter 6 together with the results.

As it pertains data uniformity, an algorithm was implemented to exclude bad data (numerical limit cycles, numerical divergence, non physical scenarios, etc...) from the training dataset. It is then easy to implement another algorithm that converts data from unknown format to the specific sampling of the deployed neural network through linear interpolation of the values of each input variable.

5.8 Technical aspects

Some technical aspect that may be necessary to replicate this study, together with some extra considerations, are reported in this section.

5.8.1 Drawbacks of the MATLAB and Simulink environment

While extremely advantageous in terms of ease of use, the MATLAB/Simulink interface is not perfect. One of the hiccups of such interface is that each simulation, when running multiple simulations from MATLAB, must be initialized individually as its own individual object in the workspace, adding to the total process time.
But the by far major problem encountered has to do with objects and files dimensions and necessary RAM. It was immediately noticed that a batch of about 1200 simulation objects occupies about 1GB of space on the system, while the space occupied by the output files is enough that it was impossible to run more than 2000 simulations per batch without encountering out-of-memory errors as 8GB of RAM were not enough to store more simulations' output data.
A workaround was put into place, dividing the total number of simulations in a number of batches so that each batch didn't contain more than 2000 simulation. Each batch was then individually saved after its completion, and the now junk data was purged from MATLAB's workspace and consequently the RAM.

In training the neural network, the number of neurons, combined with the large number of samples, meant that it was not possible to use GPU training, as MATLAB's functions for shallow neural networks don't have options to reduce the minibatch size, while their deep neural network counterparts do have such option.

5.8.2 Hardware used

Most of this work was done using a PC equipped with an AMD Ryzen 3 1200 (4 Cores / 4 Threads, 3.1 GHz) CPU, 8 GB 2166 MHz RAM and a ZOTAC Nvidia GeForce 1060 3 GB GPU using an SSD for storage. Today this hardware would be considered low-mid range.
The low dedicated memory of the GPU and the large dimensions of the problem meant that it was impossible to use GPU Accelerations in most cases.
Even though the dataset generation code was optimized to work with 8 GB RAM, the RAM was later upgraded to 16 GB 2666 MHz mostly because of the necessities of the training of the neural network. This upgrade obviously had an impact on the dataset generation code too, shaving some more minutes from the total simulation time.

Chapter 6

Results

Results of the training of neural network on the study cases described in the previous chapter are reported in this Chapter.

The following table summarizes the common characteristics for all cases.

Parameter	**Value**
Timesteps per simulation	5000
Performance type	MSE
Performance goal	10^{-6}
Max training iterations	1000
Validation fail tolerance	12
Training data percentage	75%
Validation data percentage	15%
Test data percentage	10%
Training function	*trainscg*
Hidden layer transfer fcn.	*tansig*
Output layer transfer fcn.	*satlins*

Table 6.1. Common parameters for all the neural networks

Training, Validation and Test data are taken from the same dataset, so *"1000 simulations with 75% Training data, 15% Validation data and 10% Test data"* actually means that 750 simulations are used for training, 150 are used for validation checks and 100 are used for testing. From each dataset, 100 random simulations are reserved to be used for further testing and graphical representations; these are for all intents and purposes more testing samples, but are used

to validate that the networks has good accuracy even on data it has never seen in any way.
Training, Validation and Test data are assigned randomly.

Each target parameter is normalized between 0 and 1, assuming 0 and 1 are mapped respectively to the maximum and minimum of the range from which they are taken. The inverse transformation of the predicted value is possible and doesn't affect the results, so this operation is done for the sake of convenience and better data visualization.
The initial number of timestep per simulation is 5000 as per Table 6.1 as only a point every 100 timestep is taken from the raw simulation output, as extremely good fidelity to the raw data is observed even for such a drastic cut, and using arrays of 500000 elements is impractical.

6.1 Case 1: Same predetermined command

Initially, a dataset of 2000 simulation, each with the same randomly generated command, was generated. For trials involving a different number of simulations (samples) a subset of this dataset was created.
Case 1B was used for the first tests in order to have a good sense of the effects of changing each parameter.

It was immediately clear that the networks, while capable of predicting FDJ and η_O with good accuracy even with non optimal settings, were also unable to predict the values of FSJ.
This behavior is shown in Figure 6.1, where it is clear that the network, even during training, is unable to fit this particular training parameter. Figure 6.1, on the same note, shows the three variables separately, and it is immediately evident that the parameter the network has trouble with is FSJ.

Yet, this is not surprising when looking at Figure 6.5 that shows shaft's angular speed over time of a random simulation. The system works in static friction conditions only when the angular speed is $0\,rmp$, which happens very few times during each simulation. For the majority of the simulation the system operates in dynamic friction conditions, so it is understandable that a networks is unable of fitting the static friction coefficient FSJ but is able to pick up the correlation of the input with the dynamic coefficient FDJ.

The behavior didn't change modifying number of neurons, number of samples or even number of timesteps per simulation.

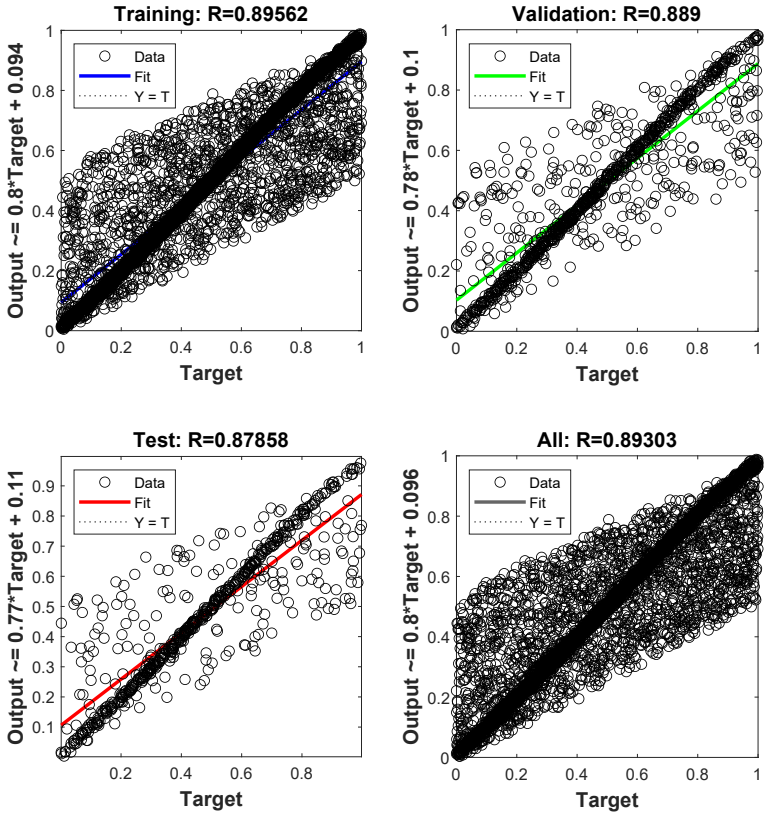


Figure 6.1. First test with FSJ, FDJ and η_O as targets

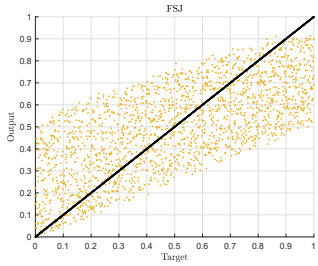


Figure 6.2. Individual fit of FSJ

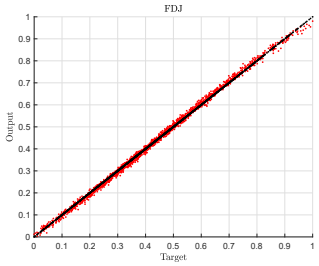


Figure 6.3. Individual fit of FDJ

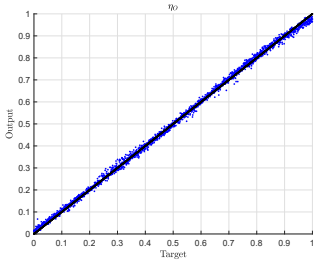


Figure 6.4. Individual fit of η_O

However, precisely because the system works most of the time in dynamic friction conditions, FDJ is more interesting for prognostic purposes. From now on, only

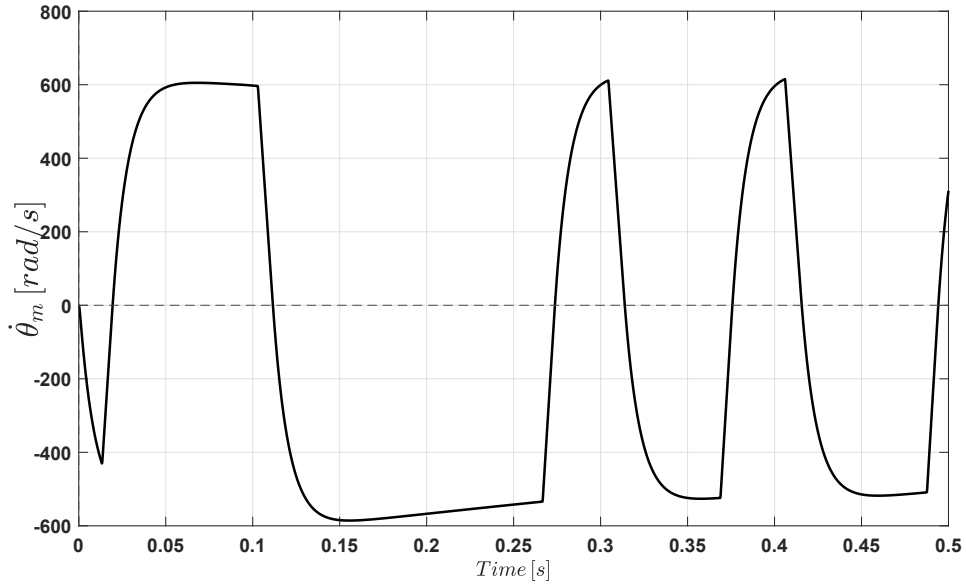


Figure 6.5. Random simulation: shaft's angular speed $\dot{\theta}_m$ over time

FDJ and η_O will be used as targets. The predicted value of η_A can still be obtained from the value of η_O using Equation 3.4, but now it isn't possible to obtain the values of FSJ and FSD; while considerations on the former have already been done, the latter is an indirect parameter anyway.
When showing results, the values of η_A won't be reported as in this work Equation 3.4 is considered exact, and so the relative error on η_O and η_A would be the same.

Nonetheless, in real life, FSJ and FDJ would vary independently (and thus FSD), so all the datasets will continue to be generated using FSJ as an "independent variable", proving that FDJ can be accurately predicted even when its static counterpart and/or their ratio are not assumed constant for every sample.

6.1.1 1A: On-ground routine (No aerodynamic loads)

A network architecture with 100 neurons and 1000 samples of 2000 timesteps each was used for the training of all the networks in this case.

For this case, the aircrafts dynamics block was removed from the Simulink model and substituted with a constant block to simulate the constant load

exerted by the elevator trying to return to its neutral position.
This simulates an aircraft that, while on-ground, performs a specific command sequence before of after a flight or during maintenance operations to check the status of some of its components.
The state-space block had to be substituted since it is not capable of representing an aircraft in static conditions.

Effectively, this case should be identical to Case 1B but without aerodynamic loads and effects related to aircraft dynamics.

However, even though the trend of the residual torque over time is similar to previous tests, including the aircrafts dynamics block, the constant external load seems to lose information regarding efficiency of the transmission that should be contained, at least in principle, in the residual torque.

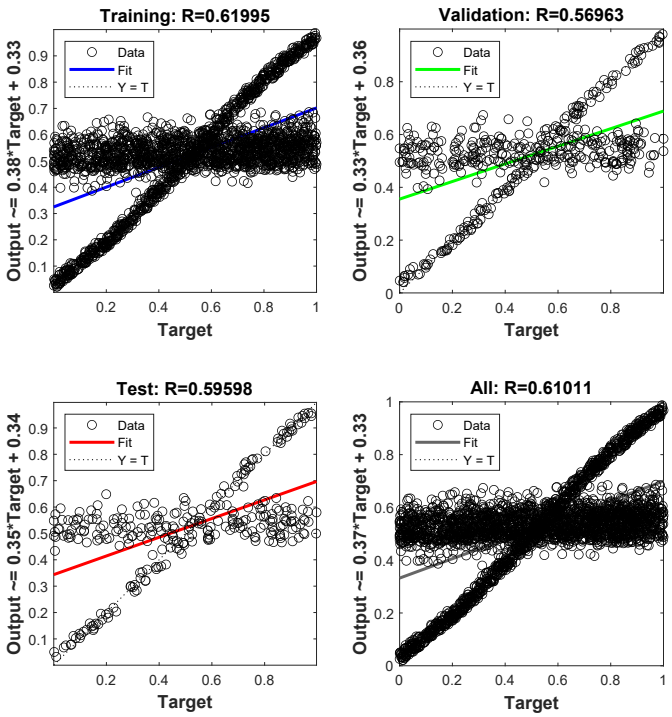


Figure 6.6. Overview of the initial results for Case 1A

Recalling Equation 5.1, this behavior is not surprising, since external load multiplies the efficiency both under aiding and opposing conditions.

Even adding the constant load or the angular speed of the fast shaft to the input vector, the performance did not change noticeably.
There was no improvement on prediction accuracy by using η_O in spite of η_A as target, meaning that this behavior is not depending on whether the actuator is operating under aiding or opposing load conditions.

However, the network is capable to accurately predict FDJ.

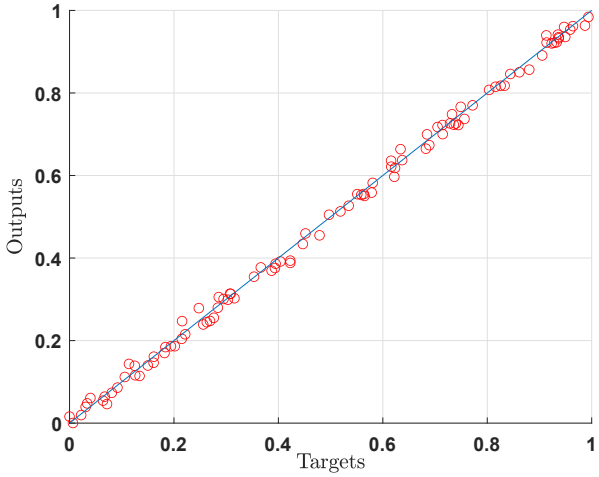


Figure 6.7. Fit for FDJ once it remained the only output, constant load

Training Performance	$2.5469 \cdot 10^{-4}$
Validation Performance	$3.0153 \cdot 10^{-4}$
Test Performance	$2.3936 \cdot 10^{-4}$

Table 6.2. Performances for Case 1A, constant load

The performance however is much worse than in Case 1B or 1C.

The training was then repeated using a dataset generated from the usual Simulink model but imposing a null initial state vector for dynamics of the aircraft. While

FDJ		
Target	**Prediction**	**Rel. err.**
0.5787	0.5586	3.46%
0.9053	0.8909	1.59%
0.7719	0.7700	0.24%
0.7048	0.7175	1.81%
0.4228	0.3938	6.86%
0.9588	0.9535	0.55%
0.7563	0.7371	2.54%
0.3663	0.3773	3.00%
0.5511	0.5548	0.68%
0.2215	0.2153	2.81%

Table 6.3. Validation for Case 1A, constant load

the results in terms of dynamics of the aircraft are neither accurate nor realistic, it was decided to use the dataset for a test.
The performance of the network was now similar to that of Case 1B.

The problem seems to be the fact that the external load is constant. Substituting again the aircrafts dynamics block, this time with a ramp command block, the results fell in between the two previously tested cases, even using only the residual torque. The regression, however, was still better for FDJ than η_O.

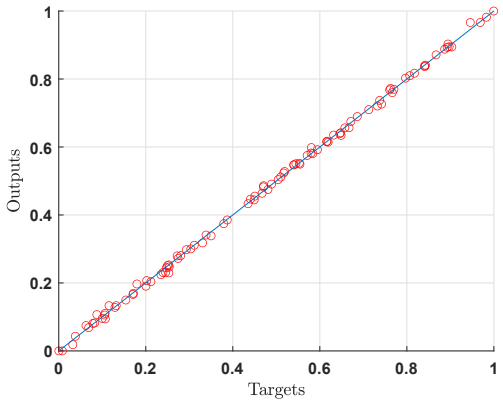


Figure 6.8. Individual fit of FDJ, Case 1A with ramping load

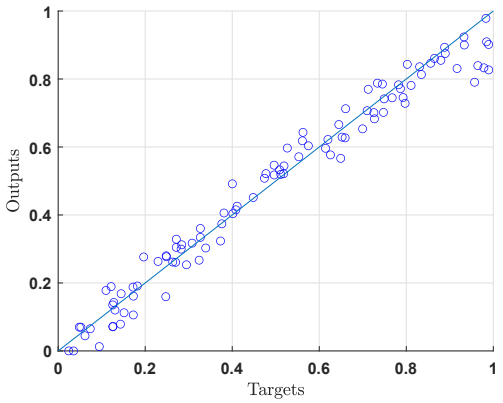


Figure 6.9. Individual fit of η_O, Case 1A with ramping load

6.1.2 1B: In-flight routine

In the dataset for this case all the simulation were performed using the same initial cruise conditions for the aircraft. This simulates an aircraft that during flight (like during a test flight or a specific routine procedure) performs a specific command to check the status of some of its components.

Timesteps	Simulations	Neurons	Epochs	Time	Stop	Train. MSE	Test MSE	Err. FDJ [2]	Err. η_O [2]
1000	1000	50	224	5 *s*	*Val.*	$2.66 \cdot 10^{-5}$	$3.80 \cdot 10^{-5}$	8.46%	1.61%
2500	1000	50	132	8 *s*	*Val.*	$5.96 \cdot 10^{-5}$	$6.08 \cdot 10^{-5}$	5.39%	3.54%
1000	500	100	199	6 *s*	*Val.*	$2.34 \cdot 10^{-5}$	$3.93 \cdot 10^{-5}$	1.71%	3.47%
2000	500	100	104	9 *s*	*Val.*	$5.34 \cdot 10^{-5}$	$6.88 \cdot 10^{-5}$	2.88%	3.17%
3000	500	100	140	20 *s*	*Val.*	$5.05 \cdot 10^{-5}$	$6.26 \cdot 10^{-5}$	2.47%	3.17%
1000	1000	100	229	8 *s*	*Val.*	$2.20 \cdot 10^{-5}$	$3.18 \cdot 10^{-5}$	3.36%	1.34%
2000	1000	100	217	21 *s*	*Val.*	$2.10 \cdot 10^{-5}$	$2.87 \cdot 10^{-5}$	3.07%	1.40%
3000	1000	100	306	44 *s*	*Val.*	$1.55 \cdot 10^{-5}$	$2.60 \cdot 10^{-5}$	3.34%	2.03%
2000	1000	200	354	69 *s*	*Val.*	$9.73 \cdot 10^{-6}$	$1.69 \cdot 10^{-5}$	2.26%	1.99%
2500	1900	250	368	155 *s*	*Val.*	$9.07 \cdot 10^{-6}$	$1.25 \cdot 10^{-5}$	3.88%[3]	1.83%[3]
5000	1900	500	612	1038 *s*	*Val.*	$4.62 \cdot 10^{-6}$	$8.93 \cdot 10^{-6}$	6.55%[3]	2.04%[3]

Table 6.4. Settings tried for Case 1B and relative performances

Table 6.4 shows really well the non linearity of the problem, as no immediate conclusion about which parameter (number of timesteps, samples and neurons) has the greatest influence on the performances.
At a closer inspection, however, an higher number of samples seems to be correlated with a smaller Test MSE (as expected, as it directly influences overfitting and underfitting), while a small number of neurons is related to higher errors. As a rule of thumb, ANNs should have a number of neurons equal to one tenth of the size of the input, and the results show that there is some truth to this.
In any case, there is clearly a point of diminishing returns after the setup with 2000 timesteps, 1000 simulations and 200 neurons, so that network was analyzed more in depth, and the results are shown below.
Figure 6.1.2 shows the regression of the normalized values of FDJ and η_O. It is evident that the network is extremely accurate (most point are close to the

[2] Average relative error, calculated averaging the relative error of 100 predictions made with data reserved for testing. It is reported to give the reader an idea of the magnitude of the error on the prediction in practical terms, but the MSE is a better indicator of the accuracy of the regression, especially the Test MSE.

[3] These high average relative errors don't tell the whole story. The network is actually extremely precise on most predictions, being consistently under 1% of relative error, but the average error is raised by the (normalized) values close to 0, where the network is far less accurate, even reaching 300% relative error. However, the absolute error is negligible even for practical applications. That's why the Test MSE paints a better picture.

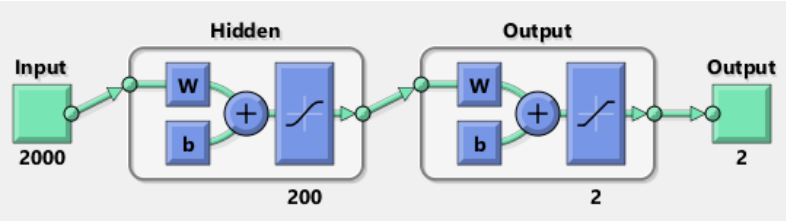


Figure 6.10. Neural network architecture chosen for Case 1B

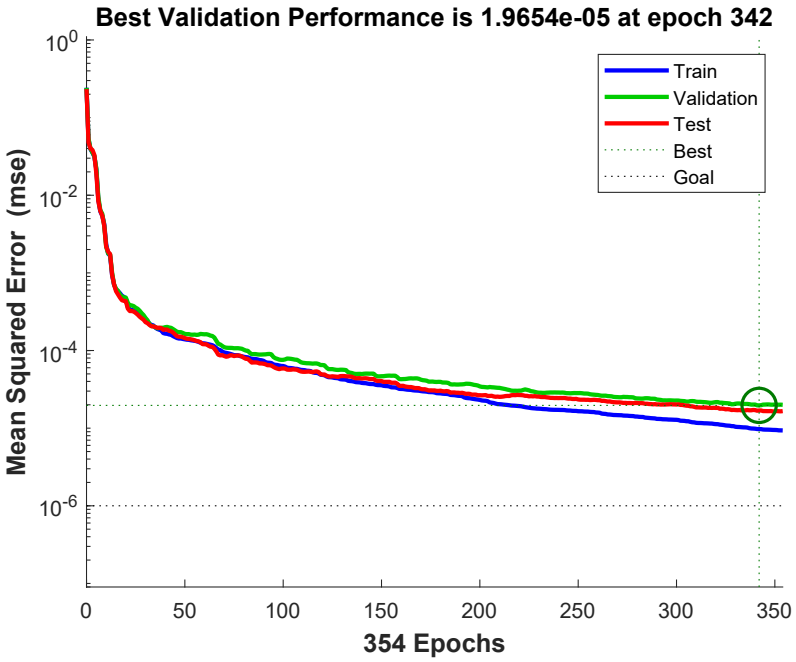


Figure 6.11. MSE during training for Case 1B

Training Performance	$9.7336 \cdot 10^{-6}$
Validation Performance	$1.9654 \cdot 10^{-5}$
Test Performance	$1.6883 \cdot 10^{-5}$

Table 6.5. Performances for Case 1B

$Output = Target$ line) even if its precision is slightly less so (the points don't actually sit on the $Output = Target$ line).
Finally, Table 6.6 shows the numerical values of 10 normalized targets and their

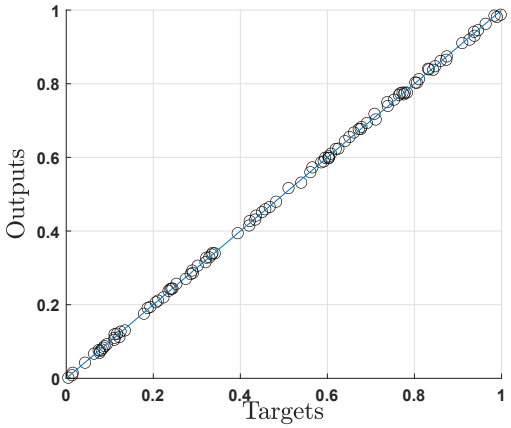


Figure 6.12. Individual fit of FDJ, Case 1B

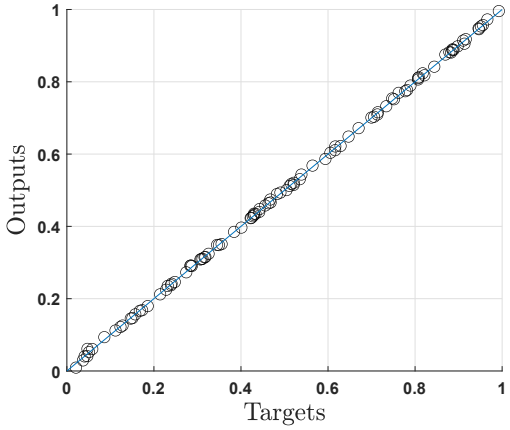


Figure 6.13. Individual fit of η_O, Case 1B

predicted value. Such targets are taken from data reserved for testing. Interestingly, the predictions for η_O are better than those for FDJ, and the higher errors happen for values close to 0.

FDJ			η_O		
Target	**Prediction**	**Rel. err.**	**Target**	**Prediction**	**Rel. err.**
0.4497	0.4513	0.35%	0.8085	0.8124	0.48%
0.0745	0.0768	3.08%	0.9165	0.9179	0.16%
0.1107	0.1049	5.31%	0.0583	0.0607	3.98%
0.8332	0.8390	0.69%	0.3069	0.3085	0.53%
0.3340	0.3362	0.66%	0.8695	0.8753	0.67%
0.8312	0.8407	1.14%	0.4272	0.4280	0.18%
0.0854	0.0831	2.61%	0.3451	0.3487	1.03%
0.2530	0.2567	1.45%	0.5351	0.5306	0.84%
0.9379	0.9302	0.83%	0.0861	0.0936	8.73%
0.2241	0.2438	0.14%	0.7816	0.7770	0.59%

Table 6.6. Validation for Case 1B

6.1.3 1C: Transmission reversibility

The dataset for this application was generated like the one of Case 1B, but choosing a nominal value of η_O such that Equation 3.4 would yield both positive

and negative values for η_A. As a reminder, negative values of η_A means that the transmission is irreversible.
Typically a reversible transmission becoming irreversible would indicate a large fault that would be caught by a diagnostic system, but rare cases where an irreversible transmission would become reversible without being considered faulty do exist. Nonetheless, the real interest for this case is to test the accuracy of the network in a situation where it has to deal with the fact that the sign of η_A determines two different types of dynamics in the model/real EMA, so this case is reported for academic purposes.

The nominal value of η_O chosen for this application is $\eta_O = 0.55$, meaning that is variation range is $0.44 < \eta_O < 0.66$, and the range for η_A can be obtained through Equation 3.4 as $-0.2668 < \eta_A < 0.4864$.

The network was trained using 1000 samples of 2000 timesteps each and 100 neurons, a combination that performed well for Case 1B.
The training stopped after the $12th$ consecutive validation check failed at Epoch 318, after 30 seconds.
The performances are reported in Table 6.7 and are better than the equivalent network for Case 1B. This may be due to the fact that the transmission is only slightly irreversible at worse for the lowest values of the chosen variation range for η_A, so the system's response is still fairly comparable between the reversible and irreversible cases.

Training Performance	$2.7147 \cdot 10^{-6}$
Validation Performance	$3.1985 \cdot 10^{-6}$
Test Performance	$3.4420 \cdot 10^{-6}$

Table 6.7. Performances for Case 1C

FDJ			textbfη_O		
Target	**Prediction**	**Rel. err.**	**Target**	**Prediction**	**Rel. err.**
0.8860	0.8878	0.21%	0.0629	0.0638	1.47%
0.4785	0.4774	0.22%	0.5940	0.5929	0.18%
0.0304	0.0328	7.86%	0.1907	0.1867	2.09%
0.9605	0.9579	0.27%	0.2957	0.2990	1.13%
0.5541	0.5547	0.12%	0.2353	0.2343	0.46%
0.2842	0.2867	0.89%	0.6308	0.6304	0.07%
0.3996	0.4019	0.56%	0.0377	0.0386	2.31%
0.3201	0.3178	0.71%	0.4491	0.4487	0.07%
0.7861	0.7846	0.19%	0.2137	0.2166	1.38%
0.0046	0.0056	20.73%	0.3878	0.3899	0.55%

Table 6.8. Validation for Case 1C

6.1.4 General comments

Case 1B was used again to test whether or not specific predetermined commands influence the results in terms of accuracy.
Multiple network were trained using the same number of timesteps, simulations and hidden nodes, but using different dataset, each generated with a different random command.
The answer to the question is that, using an high enough number of timesteps, there are no significant difference between networks trained on different commands. However, a large number of timesteps is already necessary to ensure high accuracy anyway.

Nonetheless, it was noted that difference existed for extremely low numbers of timesteps, as using only a handful of points makes the performances dependent upon which points are taken. If all the trends of the residual torque over time for a given dataset would be plotted on the same graph, some zone would highlight more differences than others. Taking only a handful of timesteps, specially selected from such zones of the graph, may result in good prediction accuracy. However, those zones may not be the same is the same plot is reproduced for a dataset trained on a different command.

Figure 6.14 shows very well that the response of the system to the same command but with different friction and efficiency coefficients of the transmission each time doesn't change very much and always follows more or less the same trend, with larger variations coinciding with points in time were the desired position of the elevator changes rapidly.
Such condition (relatively small variations between simulations) is beneficial for the neural network, as all the possible combinations of parameters are represented with a relatively small number of samples, and the fact that the network is able, in a sense, to discriminate among such tight curves is exactly what it's expected from useful prognostic methods and algorithms, as they usually deal with small variations of parameters that would otherwise make one curve indistinguishable from another.

The fine granularity of the data is not a coincidence either. For example, let's analyze 1000 samples of the values uniformly distributed over a $[0.4 - 0.6]$ range: when such values are arranged in an increasing or decreasing orders, contiguous values only differs by the fourth decimal place, and, if the fourth decimal place is neglected, each value appears five times. It is then unsurprising that the network of Case 1B, for instance, is capable of achieving precision to the third decimal place.

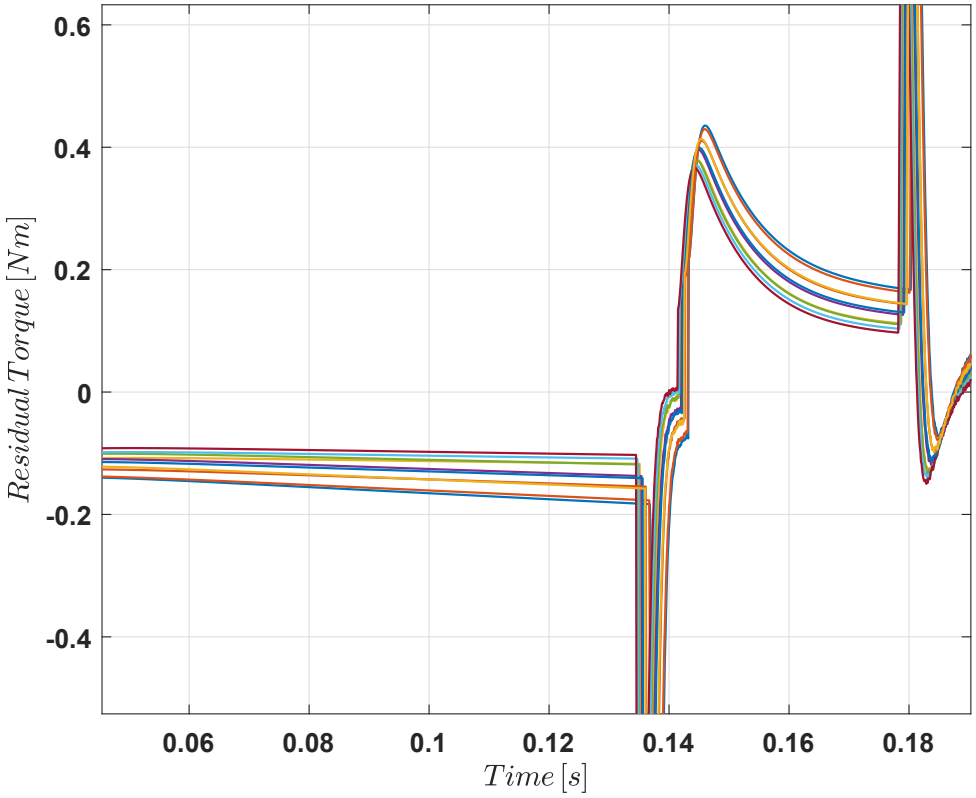


Figure 6.14. Detail of the superposition of 10 residual torque curves from the dataset for Case 1B

In a real life application a simpler command, such as a sinusoid, may be used. Based on the previous observations, it is not far fetched that a few strategically selected timesteps for each simulation (as long as they are the same for each simulation, obviously) may suffice to obtain the goal performance.
Theoretically, it would also be possible to use an algorithm to identify such zones and automatically sample the inputs accordingly, but, as the name implies, the use of a predetermined command means that the best sampling can also be predetermined.

Moreover, specifically engineered commands may be used to predict the value of other variables, such as FSJ, that was excluded as it was too much underrepresented in the data: an high frequency sinusoid command that passes enough

times across the no angular speed condition, making the system work in static friction conditions in a large number of instances, may be able to generate a dataset that could be useful for the prediction of FSJ. However, in this work this option wasn't explored, as the dynamic response of the system (as in reality) is too slow to follow rapid (when compared to the timescale of the problem) periodic variations of the command. One would need to perform the simulations over a larger total time for each simulation, but then the simulations would then take a enormous amount of time (compared to the time allotted to this work) to be performed, and the risk is that the static friction conditions would be underrepresented against the dynamic conditions anyway.

Nonetheless, the already mentioned fact that no significant performance variations appear when training networks on different commands is good news for the modularity of this method, as other networks trained to identify other parameters may instead require a specific command to highlight necessary features in the data. So, when deployed on a real aircraft, different (or maybe even the same) inputs may be extrapolated from the same run (maybe using the same tailored command) to predict multiple kinds of parameters without interfering with the parameters used in this work (nominally FDJ, η_O and η_A).

As a last observation, Case 1A is a good candidate for real hardware validation, since it was optimized to use the minimum possible amount of data for training and so that it wouldn't require too many hours of work on a specifically equipped test-bench to generate a dataset made of data acquired from real hardware.

6.2 Case 2: Multiple random commands

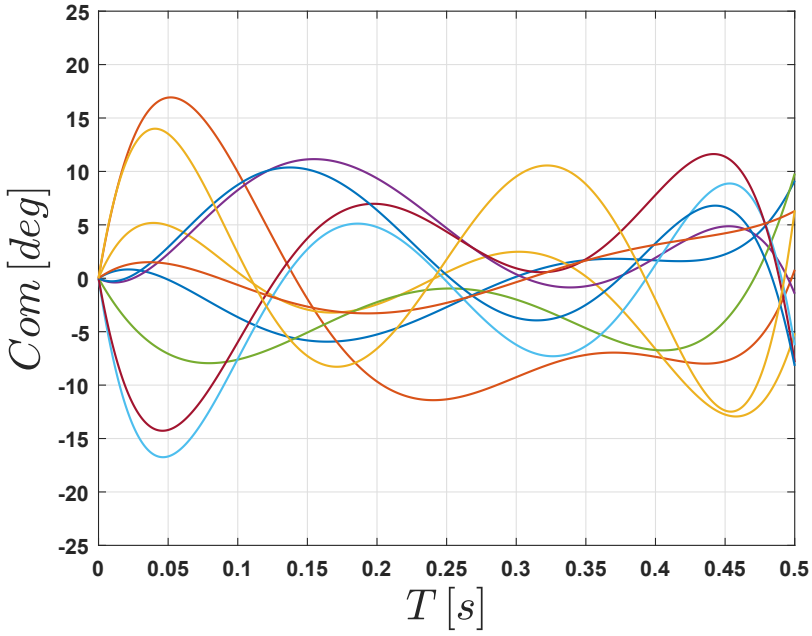


Figure 6.15. Plot of 10 of random commands used to generate the dataset for Case 2

Initially a dataset of 10000 simulations, each with its own randomly generated command and its randomly generated set of monitored parameters, was generated. Like Case 1, for trials involving a different number of simulations (samples) a subset of this dataset was created.
This dataset simulates an aircraft that, during regular flight, has a system that is able to update the health status of some of its components roughly once every second.
Again, borrowing from the experience accumulated during the study of Case 1, only FDJ and η_O were used as targets.

The use residual torque alone was immediately proven to be insufficient, as the first networks were unable to formulate good predictions even with a large number of simulations, timesteps and neurons. After even a network trained on 10000 simulations with 5000 timesteps of residual torque each and 350 neurons was unable to provide acceptable results, it was decided to also include fast shaft's speed $\dot{\theta}_m$ and external load H_e as inputs.

This behavior, however, is not a total surprise. Figure 6.16 is analogous to Figure 6.14 but for Case 2, where each simulation uses a different command.

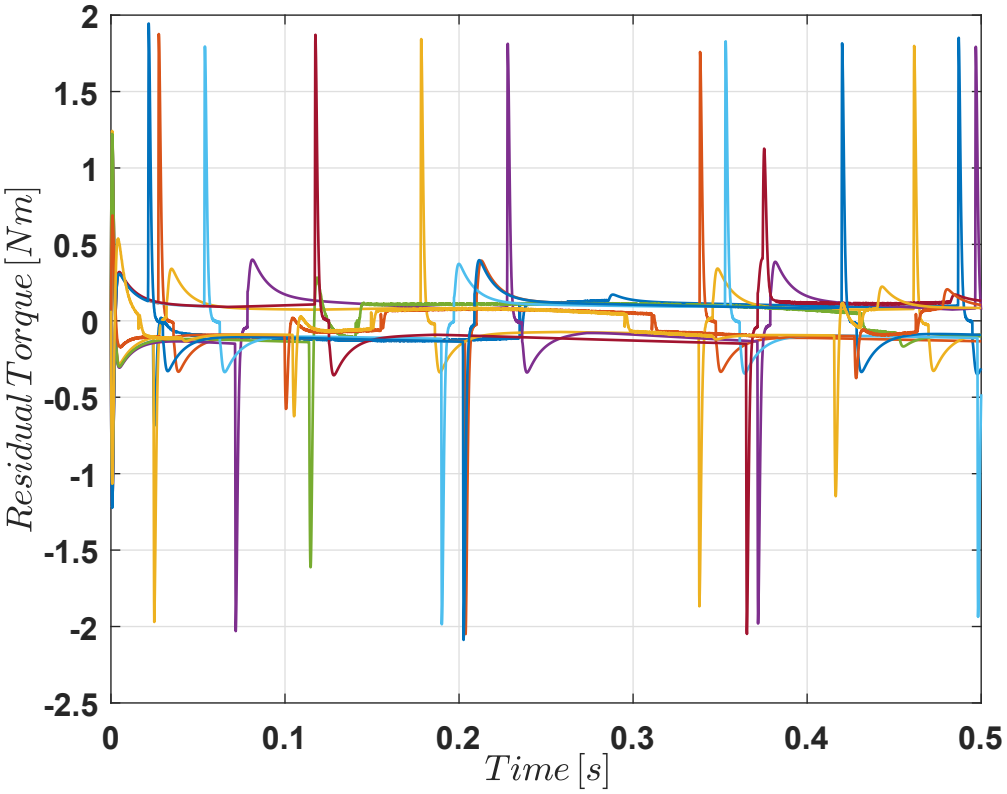


Figure 6.16. Superposition of residual torque for 10 random simulations from the dataset for Case 2

It is evident that the curves are much more different from command to command with different sets of parameters than they were when the command was the same for each simulation. The command (or, more precisely, the response of the system to a command) is in fact a variable of the problem that was set constant due the nature of the real-world applications of such networks. Applications for this case instead dictate that the network must be capable of making accurate predictions regardless of the command.

That's where adding the external load and the angular speed of the fast shaft to the inputs of the network as described in the end of Section 5.5 comes into place (Figure 5.4 is cited as reference of a "Input Map").

The external torque should contain information about the response of the system to the command and, provided the data is representative enough of all possible command-parameters combinations in the first place to make it possible for the network to find a correlation between inputs and outputs, should improve networks' prediction performances.

It is to be noted that the value of the residual torque at a given time can be the result of several combinations of commands and parameters, and that to use the command directly as an input would do more harm than good to the performances, as the response of the system (and so values of variables such as the residual torque) is also a function of the ability of the system to follow precisely the command: piloted systems usually lag behind the desired position/command and may not be fully capable of keeping up with fast variation of the commanded position, so using at the same time, for instance, residual torque and command would be like comparing different categories of things.

The training of all the network architectures hereby reported stopped either for validation tolerance of maximum number of iteration when the MSE was still too high for the network to be used in meaningful application.

- **More neurons**:
  Architectures with up to 500 neurons in the hidden layer were tested, with no avail.

- **More simulations (samples)**
  Up to 10000 simulations at a time were used as a test, with bad result. Few simulations (around 100 or less) produced clear overfitting, better described in 6.2.1.

- **More timesteps**
  The native number of timesteps of each simulation is 50000 timesteps, but it is reduced to 5000 in preprocessing for all the datasets to optimize the use of memory with no network performance lost. Using up to 5000 timesteps per simulation didn't make for an accurate network, but networks using less than 2000-3000 timesteps produced significantly worse results. It is clear that, unlike Case 1, the network are sensible to the number of timesteps per simulation, but it is not enough.

- **Different combinations of inputs**
  Other than using residual torque only or the full Input Map, combinations of residual torque and external load and residual torque and shaft's speed were tried, the former being more promising on paper, to exclude the possibility

that one of the two extra inputs of the Input Map was badly correlated to the monitored parameters and so it would harm performance. That was not the case.

- **Extra hidden layer**
  An architecture with two hidden layers and 200-100 hidden neurons was also tried, both with residual torque only and full input map. In all cases performances weren't noticeably different from the single hidden layer case, and all runs stopped after a few tens of iterations due to validation tolerance.

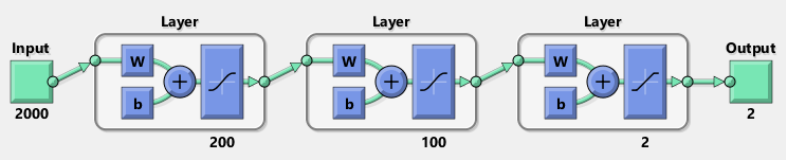


Figure 6.17. Neural Network architecture with 2 hidden layers

- **Time series formulation**
  When using more than one input variable, the arrays of each variable were simply concatenated, so that, while 1000 timesteps of a single variable translate to 1000 input nodes, 1000 timesteps of 3 variables translate to 3000 input nodes. As long as data is properly normalized and the network is trying to predict a single value for each output variable, the network shouldn't care about the order or the real nature of the inputs. In MATLAB this formulation is a matrix with dimensions $\sum R_i - by - Q$, where R_i is the size of each input and Q is the number of samples.
  Time series formulations are used when a network must be able to predict one sequence or more from input series. Usually time series as such networks are used to predict trends over time. In MATLAB this formulation is a cell array where each element $X\{i, ts\}$ is a $\sum R_i - by - Q$, where ts is the number of timesteps and R_i is now the number of inputs [10].
  Since the monitored parameters are constant during each simulation, the output sequence for this formulation will be a repetition of the same constant values for each timestep.
  This formulation was used as it is often recommended to give it a try should the matrix formulation not work, but it was just much slower in training due to the nature of operations with cells in MATLAB , with no perceivable increase in performance.

- **Exclusion of** FDJ

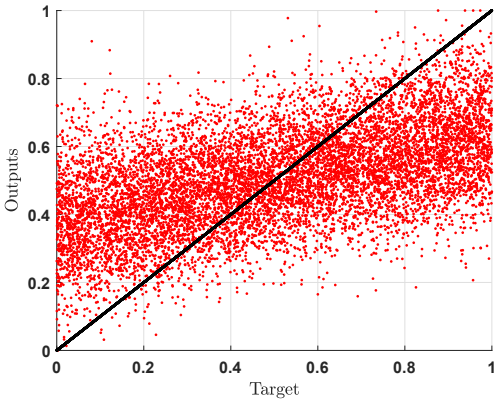


Figure 6.18. Individual fit of FDJ

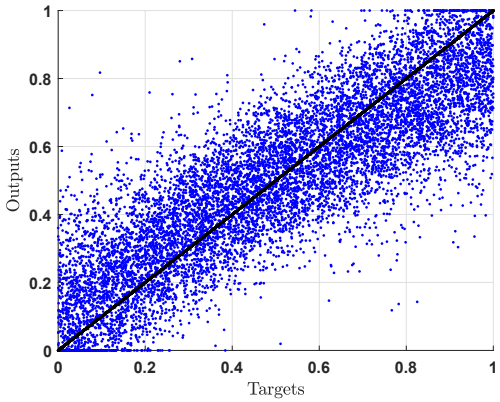


Figure 6.19. Individual fit of η_O

Figures 6.18 and 6.19 show the results of one of the cases tried (specifically a double hidden layer network). They show that the network is slightly better at prediction η_O than FDJ. This led to tentatively exclude FDJ too like FSJ to at least be able to predict η_O (and η_A).

The best result of this endeavor is presented in Figure 6.20. The architecture is composed of a single hidden layer of 100 neurons and a combination of residual torque and external load as input, with 2000 timesteps each. 5000 simulations were used for testing.
Training MSE: 0.0119, Test MSE: 0.0230.
The relative error on each prediction is still too high.

6.2.1 Comments on overfitting

When using a relatively small number of simulations (around 100, for the networks whose Figures are reported in this Subsection), clear overfitting was observed: overfitting happens when the training performance of the network (so the fit on data it constantly uses to update the value of its weights and biases) is much higher than its performance on data it has never seen. This is obviously the case of Figure 6.21.
This generally signifies that the network has bad generalization properties and the training samples must be increased or more diversity must be introduced

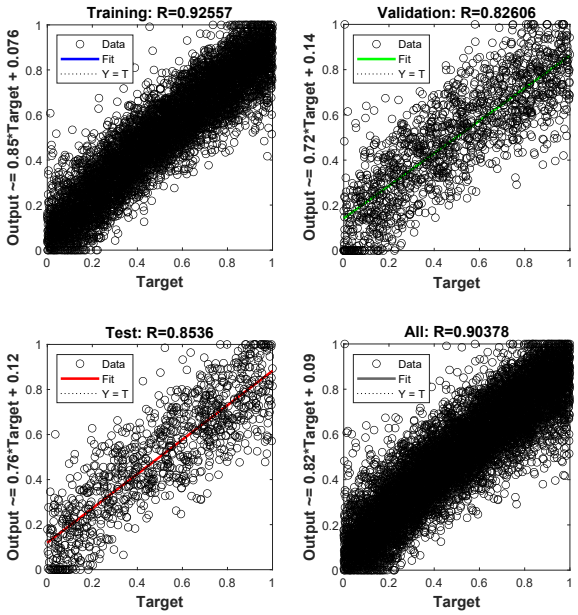


Figure 6.20. Best results for Case 2 with η_O only

in them. However, as already reported, adding samples just resulted in slower networks with bad performances even in training.

6.2.2 Comments on feasibility

At this point many options were explored to make this case work, and, while it was not possible to exclude that the training would have worked properly with a much larger number of sizes, consideration about the feasibility of Case 2 in a real-world application must be given.
The ability to know with reasonable precision the health status of the components of a system on an aircraft during a regular flight in near real-time is certainly desirable both as a means to catch incipient failures and as a multipurpose real-time monitoring system.

It is possible that increasing the number of samples in terms of combinations of commands and monitored parameters may yield satisfying results in the end, but the reality is that, as shown, not even with the additional inputs the network is able to pick up a correlation with the monitored parameters, so more samples may not be neither the simplest answer to the problem nor an answer at all.

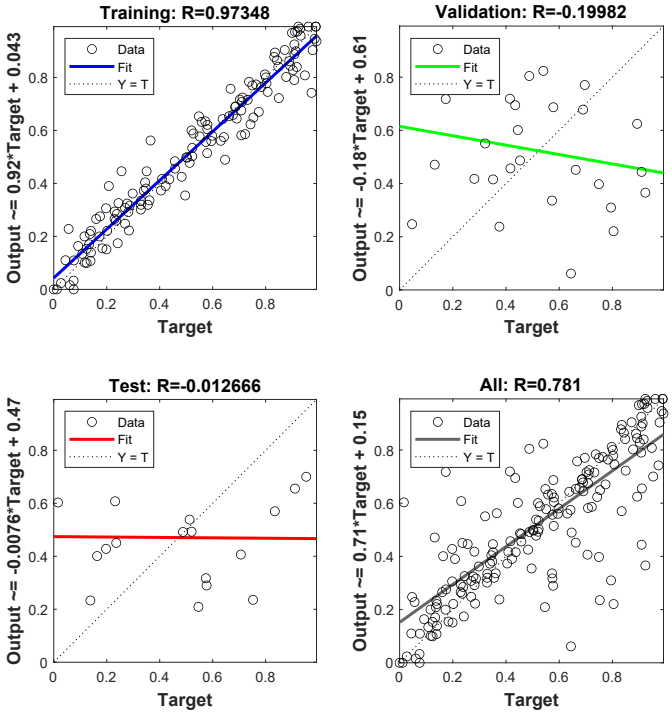


Figure 6.21. Overfitting training for Case 2 with 100 samples

The possible elevator commands that could be used on a real aircraft over a finite amount of time are effectively infinite, so this use case mush be completely rethought.

One possibility may be to exploit the fact that a command over a large time can be split in numerous simpler commands over shorter periods of time, reducing the proposed Case 2 to a more complex version of Case 1B, with datasets trained over a limited number of well representative simple commands: for instance, ramps with different inclinations and end values.
However it is impossible to say how the attribute *well represented* may translate into practice, and one may easily fall again into the fallacy of trying to train the network on any possible combination of commands and parameters.

Another approach, based on the previous consideration, may be to implement a system that only activates when a specific usable sequence of command is used. This would effectively be Case 1B with a different application in mind, as more than one network may be trained on different commands so that the system

could be activated more frequently.

Finally, the system may be implemented with a different philosophy altogether, one that trains the networks directly on board of the aircraft. This is explained in more detail in Subsection 6.3.

6.3 Proposal for near-real-time monitoring

While Cases 1B and 1C, and partially 1A, demonstrate that non-real-time monitoring applications described in Subsection 5.1.1 work in concept, the situation is different for applications described in Subsection 5.1.2, which rely on knowing the health status of the components in real-time. This means that the network must be able to estimate the monitored parameters under any condition, like in Case 2, but, as extensively commented in Section 6.2, this is no easy task.
Some alternatives were proposed in Subsection 6.2.2, but they all rely on the notion that the deployed neural network must be pre-trained once and for all beforehand.

However, particular emphasis has been put on Case 1B (network trained on a single predetermined command to be executed during flight) in this study, as it is also a case of interest for applications where the network is trained or adapted each time it is used.
Without delving into adaptive neural networks, the following proposal is for a system that is able to log the elevator command sent by the pilot during flight, the flight conditions and the response of the system and use system's response as input for a neural network and command and flight conditions to perform on the spot enough simulations of the systems to be used to train a neural network that will then estimate the health status of the monitored parameters.
In short, the proposal is to apply Case 1B performing simulations and ANN training directly onboard the aircraft during flight to obtain the status of the parameters at specific intervals of time, intervals determined by the time needed to perform measurement, simulation, training and prediction.

Measurement is obviously done by sensors onboard the aircraft and includes sampling. Prediction for ANNs of the scale of those analyzed in this work is almost immediate. As for training, Table 6.4 also shows training times for different network architecture and relative performances. It is to be noted that, sacrificing some accuracy, training can be completed in as little as 5 to 10 seconds.
As for the simulation part, it takes the most time among the four, but a speculative study on the time necessary to perform a certain amount of simulations

on the hardware used for this study (Described in Subsection 5.8.2), composed primarily by a Ryzen 3 1200 3.1 GHz 4C/4T CPU. The results are shown in figure 6.22. Simulation time also includes the time needed to initialize the simulations. Estimated simulation times for CPU with different number of cores (equal to the number of threads in this instance) but same base clock speed are drawn with dashed lines. Theoretically, the simulation time should halve doubling the number of threads.

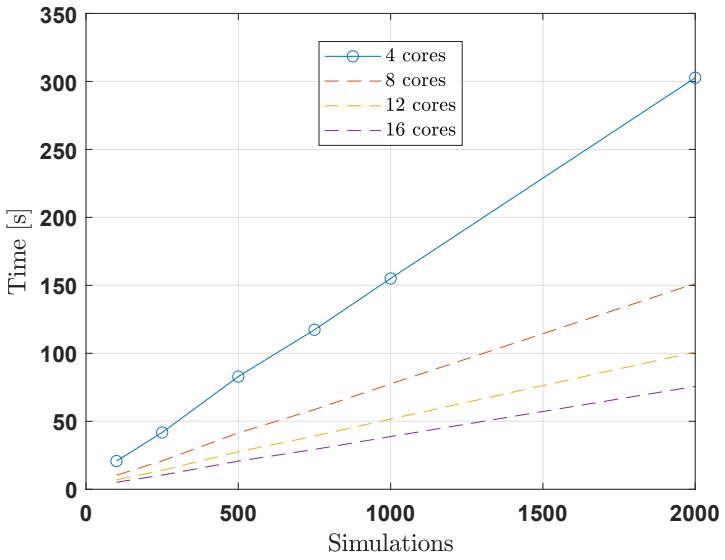


Figure 6.22. Simulation time vs Number of simulations

Referring again to Table 6.4, simulating data and training the network for the architecture with 1000 timesteps, 500 samples and 100 neurons would have taken about 90 seconds on the hardware used for this work or roughly 45 seconds on an 8 thread CPU and even less on many-cores CPUs.

The steady decline of the prices of workstation and professional grade many-cores CPUs over the years means that such system could be deployed for less than 3000$, a relatively low price for aeronautic systems.

At least in terms of update frequency and cost, shift the burden of simulating and training for the network on a system onboard the aircraft working in near-real-time, with update frequency as low as 20 seconds for a reasonable price, seems certainly feasible. And the time needed could be even lower using programming languages more efficient than MATLAB and making clever usage of memory

addressing during the initialization of the simulations.
This couldn't be possible without an hybrid method such as the one illustrated in this work.

6.4 Note on real-world performances

The tables and the results reported in this chapter show relative errors commonly between 1 and 3%, often even under 1%. In reality, should one feed data from real hardware to one of the successfully trained networks, the relative error will probably be between 5 and 10%.
For reasons explained in Section 4.6, the output of the Simulink model differs from the measurements from a real system in noise, in the sense that the output of the Simulink model is purposely free from any kind of noise that could otherwise be present in data obtained from real hardware for a number of reasons.

Because of this, the cases explored in this work may be labeled *synthetic*, and it is likely that the network won't perform as well on data from real hardware. That's because, even though preprocessing is always necessary in this kind of applications to eliminate as much noise and losses of fidelity as possible from the data, preprocessing isn't perfect. Preprocessing must be tailored to the measurement and sampling methods and the hardware at hand, so this variable wasn't taken into account as it is not the focus of this work and the proposed method was still not mature enough. It is, however, something that has to be taken into account should this method be put to test on real hardware.

Nonetheless, having a large performance headroom on synthetic tests translated in large headroom for the real application too.

Conclusions

In conclusion, the residual torque proved to be an extremely reliable prognostic indicator, but only for certain applications and use cases.
Nonetheless, such use cases could be extended with specific modification to also cover many of the applications where the residual torque was insufficient or unreliable, as proposed in Subsection **??**.

Now that the concept has been proven to work, it can be expanded upon in many ways:

- Compatible data obtained from real hardware could be fed to the trained network to measure the performance of the network once denoising strategies and input data preprocessing in general are considered.

- As the strong point of the Simulink model is its ability to be tailored to specific hardware, tests on real hardware in testbench conditions may be compared to the data produced by the Simulink model, to modify its parameters until the model reproduces satisfactory the reality.

- Regarding the deployment of the system, it may be useful to stakeholders to establish what could be a useful update frequency for the monitored parameters in near-real-time applications and which level of performance is required. Slow growing, predictable faults obviously may be caught even by applications such as that of the proposed Case 1A, but fast developing faults may require more frequent monitoring.

- However the potential faults indicated by the parameters monitored in this work have all to do with wearing, which is a slow process, unless other factors intervene. So, a better understanding of how such parameters are linked to fault modes is necessary.

- Throughout this work it has often been repeated that this method is useful to provide the actual health status of the components to improve the precision of algorithms used to calculate the Remaining Useful Life of a components, and it may be worthy to develop such an algorithm.

Other things instead could be improved from this work:

- The model of the longitudinal dynamics of the aircraft is the weak point of the Simulink model and, while it outputs realistic data, it is based on simplifications that make it much less accurate than it could be, and a better model must be implemented in order to tackle real life scenarios.

- It may be useful to develop a standard specifically designed predetermined command for all the applications that requires it in order to optimize the training using a smaller amount of more meaningful timesteps. On the other hand, it may also be useful to continue on the path traced and develop an algorithm capable of finding such meaningful points, which would be beneficial for the proposed system architecture for near-real-time applications.

- Bad generalization of the problem is behind the failure of Case 2. Other machine learning techniques and techniques to find the most relevant inputs should be explored.

All in all, the results are satisfactory to the point that validation on real hardware may begin, even is there is still a lot to do before such systems (not only the ones proposed in this work but also the similar ones found in literature) may be implemented on safety critical hardware in real aircraft.

The hybrid methodology used is the key enabler behind most of the proposals and suggestions made in this work, but, while in literature the interest about these methods is growing, they are still not mature enough for many applications. This was kept in mind during the work and so consideration must be given also to the fact that data for training could also be acquired either through testbench trials and testflights or from the archive of airline companies, which already log lots of data about their aircraft for maintenance and cost saving purposes.
If lots of data are required, the latter option is more appealing since it is easier to obtain a sizeable dataset monitoring a fleet of aircraft of different ages over the span of a cycle of maintenance than to design and build a testbench that is capable of artificially varying friction coefficients and transmission efficiencies.

Finally, it must be said that this method is highly modular: a system that uses network trained to use the reconstructed signal of the residual torque to estimate a series of parameters can easily coexist (and maybe even integrated together) with a similar system used to estimate different parameters with different inputs.

Bibliography

[1] George Vachtsevanos et al. *Intelligent fault diagnosis and prognosis for engineering systems*. Wiley, 2006.

[2] Pier Carlo Berri. "Genetic Algorithms for Prognostics of Electromechanical Actuators". MA thesis. Politecnico di Torino, 2016.

[3] Pier Carlo Berri et al. "Simplified models for mechanical transmission efficiency with opposing and aiding loads". In: *International Journal of Mechanics and Control* 20 (2019).

[4] Lorenzo Borello and Matteo D. L. Dalla Vedova. *Mathematical model and computational algorithm simulating the dynamic behaviour of a mechanical element affected by load, active and load dependent friction forces (or torques)*.

[5] Matteo Dalla Vedova and Lorenzo Borello. "Dry Friction Discontinuous Computational Algorithms". In: *International Journal of Engineering and Innovative Technology* 3 (Jan. 2014), pp. 1–8.

[6] FAULHABER. *FAULHABER BX4 Series BLDC Motor cross-section*. URL: https://www.faulhaber.com/it/prodotti/motori-brushless-cc/faulhaber-bx4/.

[7] Glosser.ca. *Simple ANN*. [CC BY-SA 3.0]. URL: https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg.

[8] Paolo Maggiore. *Lecture notes from "Evoluzione dei Veicoli Aerospaziali" course*. 2015.

[9] Paolo Maggiore and Matteo D.L. Dalla Vedova. *Lecture notes from "Modellazione, simulazione e sperimentazione dei sistemi aerospaziali" course*. 2019.

[10] Mathworks. *Function 'train' documentation*. Documentation. URL: https://it.mathworks.com/help/deeplearning/ref/network.train.html.

[11] missinglink.ai. *Neural Networks for Regression - Overkill or Opportunity?* Slides. URL: https://missinglink.ai/guides/neural-network-concepts/neural-networks-regression-part-1-overkill-opportunity/.

[12] MOOG. *Common Electro-Hydrostatic Actuator produced by MOOG.* URL: https://www.moog.com/products/actuators-servoactuators/space/launch-vehicles/common-electro-hydrostatic-actuator.html.

[13] MOOG. *Flapper-Nozzle Servovalve Schematics.* URL: http://www.moogvalves.com/ty/email/infographic-1550T7-4148ZT.html.

[14] MOOG. *Mechanism of a ball screw and a planetary roller screw.* URL: http://www.moogscrews.com/evolution/welcome-1453FD-4042LF.html.

[15] Engineering Team Members at Pittman Motors. *Comparing slotted vs. slotless brushless DC motors.* Tech. rep. URL: https://www.haydonkerkpittman.com/-/media/ametekhaydonkerk/downloads/white-papers/comparing_slotted_vs_slotless_brushless_dc_motors%20l.pdf?la=en.

[16] Gaetano Quattrocchi. "Development of innovative prognostic methods for EMAs". MA thesis. Politecnico di Torino, 2019.

[17] Gaetano Quattrocchi et al. "Innovative Actuator Fault Identification Based on Back Electromotive Force Reconstruction". In: *Actuators* 9 (July 2020), p. 50. DOI: 10.3390/act9030050.

[18] Stefano RE. "Development and comparison of prognostic methodologies applied to electromechanical servosystems (EMA) for aerospace purposes". MA thesis. Politecnico di Torino, 2018.

[19] PHM Society. *Introduction to Prognostics.* Slides. URL: phmsociety.org.

[20] Vyacheslav Tuzlukov. *Signal processing noise.* 2018.

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