Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Análisis y Visualización de Datos Masivos

Análisis del ensayo de fatiga de un actuador electromecánico del sector aeronáutico para detección de anomalías

Trabajo Fin de Máster

Tipo de trabajo: Piloto experimental

Presentado por: González Velázquez, Roberto Javier

Director/a: Garay Gallastegui, Luis Miguel

Resumen

Este trabajo presenta el análisis de un conjunto de datos obtenido por medio de una serie de experimentos realizados sobre el prototipo de un actuador electromecánico diseñado para el control de superficies primarias de vuelo en un avión. Los experimentos se han llevado a cabo en un banco de ensayos construido en el centro de investigación Tekniker.

Los actuadores electromecánicos son componentes que utilizan energía generada eléctricamente para mover elementos específicos de un avión. En los últimos años, hay un gran impulso en la industria aeronáutica para desarrollar este tipo de actuadores, los cuales sustituirán en un futuro a los sistemas electrohidráulicos y electroneumáticos. Este es uno de los principales objetivos que está detrás de los conceptos de "Avión más eléctrico" (“More Electric Aircraft”, MEA) y "Avión totalmente eléctrico" (“All Electric Aircraft”, AEA) para crear una nueva generación de aviones más eficientes.

Los actuadores electromecánicos ofrecen muchas ventajas (mayor eficiencia energética, mayor grado de control y facilidad de mantenimiento, menor complejidad, entre otras), y se han utilizado en aeronaves militares, o bien en funciones no críticas de aeronaves comerciales, pero plantean todavía algunos desafíos técnicos y de seguridad (bloqueos mecánicos, principalmente) que han limitado su uso en la industria aeronáutica.

La utilización de un banco de ensayos para provocar una degradación acelerada de estos actuadores ofrece una buena plataforma para desarrollar algoritmos de detección anticipada de fallos, lo cual mejorará su seguridad y fiabilidad.

Este documento describe un sistema de información basado en datos con un control estadístico de procesos multivariado que utiliza algunas señales obtenidas en el banco de ensayos para detectar potenciales fallos durante la operación del actuador, y para ello se emplean técnicas de aprendizaje automático y de inteligencia de datos, así como una infraestructura desarrollada en la nube.

Palabras Clave: Industria aeronáutica, Actuador electromecánico, Control estadístico de procesos multivariado, Detección de fallos basada en datos

Abstract

This work presents the analysis of a dataset coming from a series of tests performed on an electro-mechanical actuator (EMA) prototype, designed for primary flight control surfaces on an aircraft. The dataset has been obtained through a test bench designed and built at Tekniker research center.

Electro-mechanical actuators are components that use electrically-generated power for moving key parts of an airplane. Nowadays, there is a trend in the aviation industry for developing this kind of actuators, which will eventually replace electro-hydraulic and electro-pneumatic systems. This is the main idea behind the concepts of “More Electric Aircraft” and “All Electric Aircraft”.

EMAs have significant advantages (higher energy efficiency, controllability and maintainability, less complexity, among others), and have been used in non-critical and military applications, but still pose some technical challenges and safety considerations (mostly mechanical jamming) which have restricted their use in the aviation industry.

The use of a test bench for the accelerated degradation of this kind of components can be an efficient strategy to develop health monitoring algorithms for failure anticipation, which will help to improve their safety and reliability.

This document describes a data-driven information system based on a multivariate statistical process control (SPC) method, focusing on some of the signals obtained in the test bench for detecting failures during the EMA operation, leveraging big data, cloud, and machine learning techniques for this purpose.

Keywords: More Electric Aircraft**,** Electro-mechanical actuator, Multivariate statistical process control, Data-driven fault detection

Table of Contents

Índice de tablas

Índice de figuras

1 Introduction

1.1 Motivation

Traditionally, commercial aircrafts and helicopters have used a combination of technical solutions (hydraulic, pneumatic, mechanical, electric) to actuate on key parts of their systems, like flight control surfaces, landing gears, engines and other utilities like cargo doors, etc. (Giangrande et al., 2018).

An actuator is a device that uses some kind of power source to transmit mechanical motion (linear or rotary) to certain components of an airplane for controlling movements on the ground or in the air (Bennett, 2010).

Flight control surfaces are movable elements that can be classified into two different groups:

- Primary control surfaces, which generate the necessary torque for the basic rotation of the aircraft in all three dimensions, namely the roll (rotation around the front-to-back axis), the pitch (rotation around the side-to-side axis) and the yaw (rotation around the vertical axis), and include the ailerons, the elevators, and the rudder.

- Secondary control surfaces, which have an influence on the lift, drag, or speed of the aircraft, and include the flaps, the slats, the spoilers, and the trimmable horizontal stabiliser.

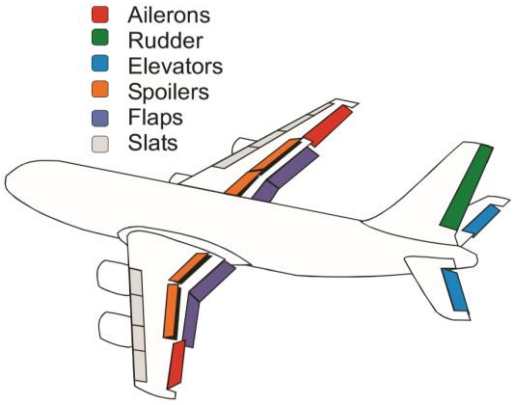


Figure 1: Flight control surfaces on an aircraft
(Source: Bennett, 2010)

All different forms of actuation typically found on an aircraft are usually integrated into a conventional architecture which has reached a good level of maturity as a result of many decades of evolution (Rosero et al., 2007).

The power distribution characterising this conventional architecture can be seen in the following picture:

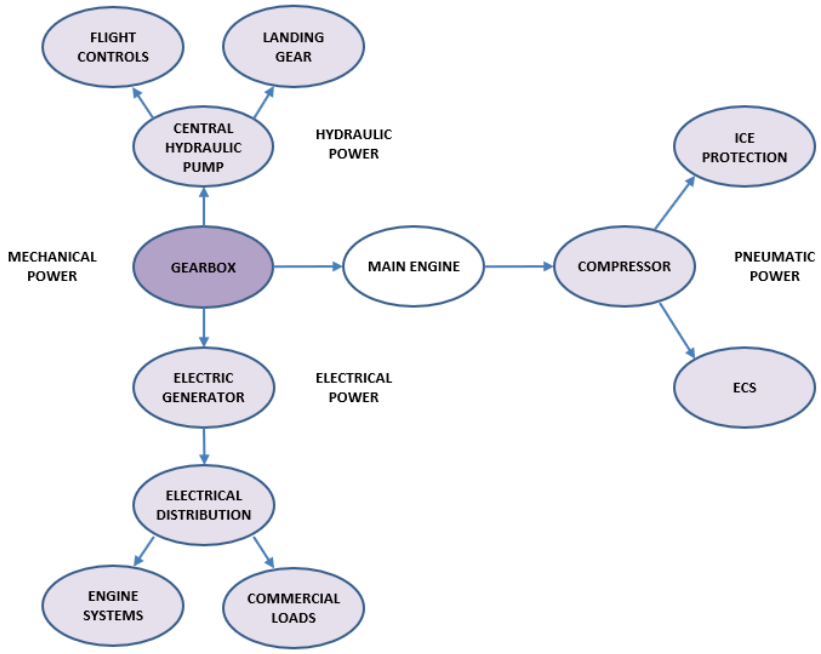


Figure 2: Conventional power distribution on an aircraft
(Source: Adapted from Rosero et al., 2007)

This power distribution presents several inconveniences in practice, ranging from maintenance (hydraulic fluid leak, pneumatic pressure losses), size and weight (tubing network, pumps, redundancy systems, etc.) and energy consumption (e.g. bleed air off-takes needed to feed different systems). In addition to this, there is limited evolution prospects for this kind of technologies (ACTUATION 2015).

Gaining momentum especially from the 90´s (Jones, 2002), there is a trend in the commercial aircraft industry to develop “power-by-wire” (PBW) actuators, as a key element to achieve the goal of what is called the “More Electric Aircraft” (MEA) paradigm, which in turn would be a transition to the more ambitious idea of the “All Electric Aircraft” (AEA).

This trend, summarised in the concepts presented above, has two main objectives (Rosero et al., 2007):

- To remove hydraulic power generation and bleed air off-takes, and to increase the use of electrical power generation.
- To replace hydraulic and pneumatic actuators by electrically-operated (“power-by-wire”) counterparts.

The term "power-by-wire" is a term applied to actuators powered by an electrical source (Bennett, 2010), and it is a very graphic way of describing the general idea behind this technological transition: the power for actuation is transmitted by electrical wires instead of using hydraulic lines and pumps, or pneumatic circuits.

The following figure shows a comparison in the power transmission between PBW actuators, such as electro-hydrostatic (EHA) and electromechanical (EMA) actuators, and the more traditional hydraulic actuators (HA):

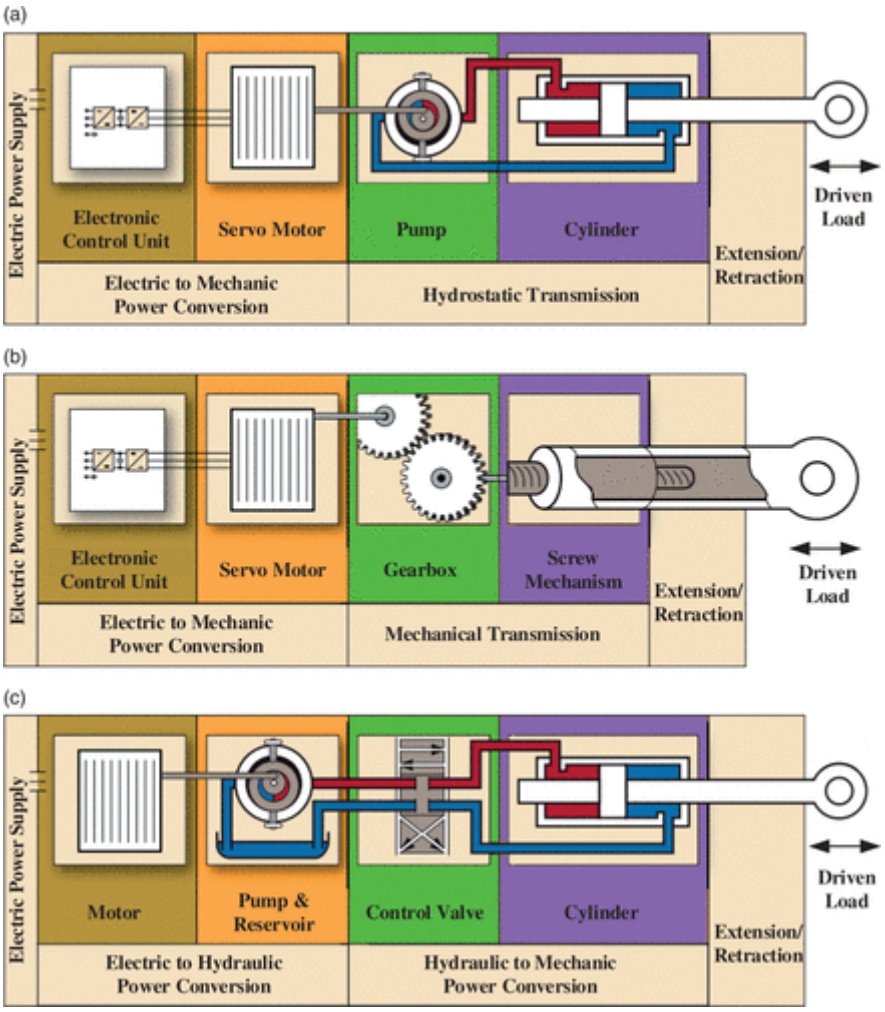


Figure 3: Power transmission in actuation systems (a) EHA (b) EMA (c) HA (Source: Qiao et al., 2018)

As can be seen in the picture above, transmission in EMA is achieved by mechanical means, whereas a local hydrostatic circuit is used in the case of EHAs, which are hybrids between electrical and hydraulic devices, but are classified into the power-by-wire segment.

The use of PWB systems on aircrafts can provide numerous advantages (Qiao et al., 2018) (Li et al., 2016), for example:

- Lighter aircrafts, as the removal of hydraulic circuits means a reduction in weight.
- Higher engine efficiency and a reduction in fuel consumption, due to the removal of non-propulsive bleed air off-takes in engines.
- Improved maintainability, because electrical systems are easier and faster to replace than most pneumatic, hydraulic, and mechanical systems.

The presence of PBW actuators have gradually increased over the years, and the technology has become mature enough to be introduced in large commercial aircrafts (Boeing 787, Airbus A380) and advanced combat fighters (Lockheed F35) (Giangrande et al., 2018).

In comparison, EHA technology is currently more used than EMA actuators due the accumulated knowledge and experience in hydraulic systems, in spite of the need of local hydraulics and the associated disadvantages that this brings (Giangrande et al., 2018).

EMA technology is mainly used on secondary flight control surfaces and, despite their benefits over EHAs (higher efficiency, further weight reduction), some breakthroughs are still needed in terms of safety and reliability for a more widespread use (Isturiz, A., Vinals, J., Abete, J. M., & Iturrospe, A., 2012).

EMAs can be divided into two groups according to how transmission is performed: linear EMAs (where linear motion is achieved via a ballscrew or a geared ball screw, among others), and rotary-geared EMAs (Li et al., 2016).

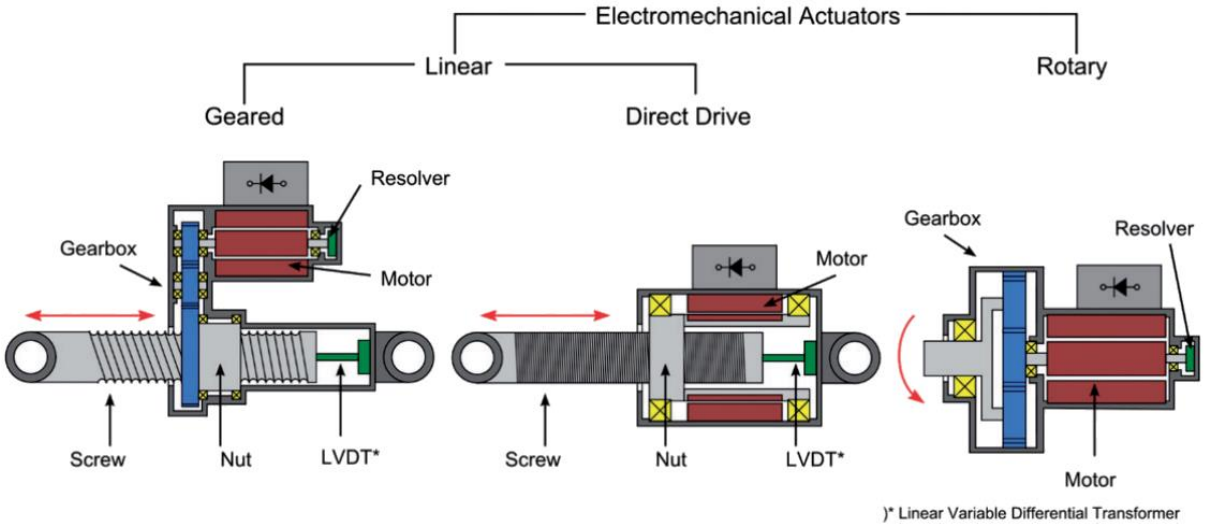


Figure 4: EMA classification according to transmission type
(Source: Qiao et al., 2018)

Given that EMAs are the future of the aviation industry, there has been a great effort at research level in different countries (Qiao et al., 2018), but some challenges still persist for the wide adoption of the technology, in particular (ACTUATION 2015):

- Economic issues, due to a low scale production and a high customization level, and to a lack of standardization in the design, testing, and certification of EMAs.
- Reliability issues, especially regarding the mechanical problem of jamming, which is considered one of most feared fault modes in EMAs (Isturiz et al., 2010), and an important factor in the reluctance of a widespread adoption of EMAs in critical applications, even with a very low failure rate.
- Technological issues, as new materials and architectures need to be developed for an overall weight reduction of components, and sensors must be optimised and use the latest state-of-the-art technologies.

In relation to jamming, there are some approaches that can be followed to mitigate this problem, including a fault tolerant design, improved maintenance and fault diagnosis (Hussain et al., 2018).

Fault diagnosis and condition monitoring techniques are widely spread in many industrial scenarios, but they are not a common practice in aeronautical actuators, where preventive maintenance is still a common practice (Zhang et al., 2017), and they are still in the early stages in EMA components (Ruiz-Carcel & Starr, 2018).

Therefore, the use of health monitoring techniques can be a key strategy for improving the reliability of EMAs, thus moving from preventive to predictive maintenance, and a step in the development and wider use of these components in the aeronautical industry (Todeschi, M., & Baxerres, L., 2015).

Once the need for EMA health management has been highlighted and justified, the following sections presents the object and approach for this work.

1.2 Objectives

The object of this work is the analysis of a dataset obtained through a campaign of tests performed on an linear EMA prototype with an internal ballscrew transmission mechanism designed for a primary flight control surface from an original equipment manufacturer (OEM) in the aviation industry.

These experiments have been conducted on a test bench designed and built specifically for this purpose at Tekniker research center, which collects large amounts of data through sensors, and periodically uploads this information to a remote repository on the cloud.

The strategy for the experiments conducted on this test rig is, on the one hand, to execute a fatigue test whose purpose is to continuously acquire information related to the EMA operation, such as speeds, positions, number of cycles, etc., and on the other hand, to trigger different condition tests from time to time to obtain information that can be used for health assessment.

This work focuses on the analysis of the EMA operation information obtained through the fatigue test with the objective of developing a data-driven, signal-based fault detection method. In this respect, the approach of collecting and analysing data at large scale in a multisensor fusion environment, using the cloud as a supporting element, is what makes this proposal innovative.

This data-driven strategy uses signal processing techniques on the time domain to reduce data size so that patterns can be identified and compared, in order to detect deviations from normality and raise alarms.

The goal of the fatigue test is to gradually wear-down the EMA in a series of continuous movements simulating operating conditions in an accelerated manner, aiming to reach some kind of mechanical fault, or else reach a point where degradation can be detected.

During the tests, a working load simulating a primary control surface is applied to the EMA by means of a hydraulic system.

1.3 Approach

The following section gives a general view of the information flow from the bench to the final repository on the cloud to understand the context behind the dataset. Next, a description of the approach followed for the analysis of this dataset is presented.

1.3.1 Test Bench Data Acquisition Flow

The data acquisition flow from the test bench to the final repository on the cloud can be seen in the following figure:

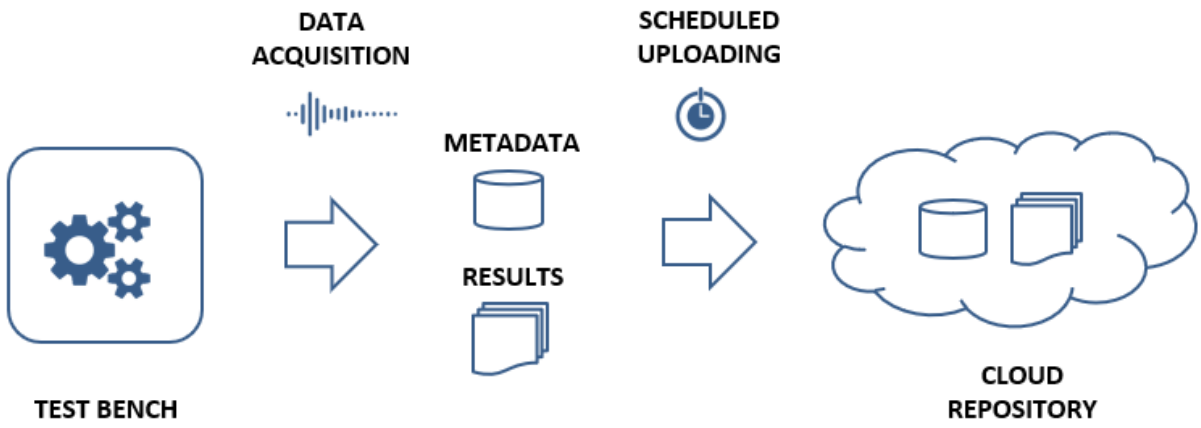


Figure 5: Test bench data acquisition flow diagram

Below is a brief description of the different elements of the data flow showed above. A more detailed diagram of this architecture can be seen in the section explaining the exploratory data analysis and the automated cloud platform.

Data Acquistion

During the process, several signals coming from sensors are stored in Matlab MAT files and database metadata.

The acquisition system is composed of several equiments that have to be synchronised to obtain a timestamp with a common time base.

EMA signals are acquired via the PLC (Programmable Logic Controller) system controlling the actuator's electrical motor. Also, supplementary sensors have been added to the bench for additional information.

Scheduled Uploading

Data is uploaded periodically (once a day) onto a central repository located in the Azure cloud to save space in the acquisition system. Files are stored on an Azure blob storage container, and test metadata is replicated into an Azure SQL Database instance.

1.3.2 Data Processing Approach

Once all the tests have been completed and data is uploaded on the Azure repository, a data processing pipeline is run for the analysis of the dataset.

The main steps taken during this analysis are represented in the following diagram:

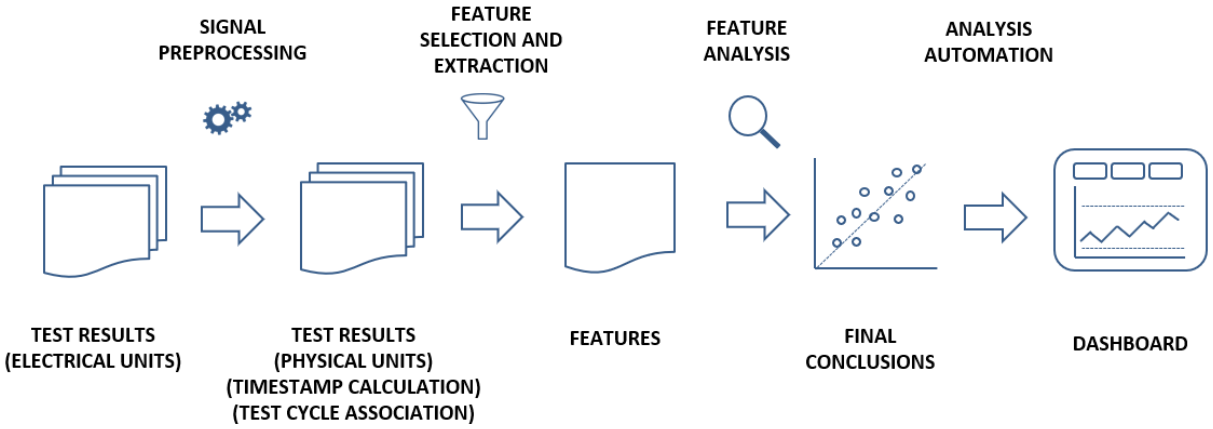


Figure 6: Data processing pipeline

The stages in the data pipeline are explained below.

Signal Preprocessing

The following processes are performed during this step:

- Conversion from electrical to physical units.

  Signals acquired from the bench consists of a series of electrical measurements that must be converted to physical units using metadata information stored in the system's database.

  Each signal has its own conversion ratio that has to be obtained and applied, and as a result, a set of files with physical measurements is generated.

- Timestamp calculation for each sample.

  Due to the syncronization mechanisms used to obtain a common time base for the different acquisition equipments, the timestamp has to be calculated for each of the samples stored in the files.

- EMA cycle counter association.

  A cycle is each one of the iterations which are repeated over and over, simulating the EMA operating conditions.

  The number of cycles performed by the EMA during the tests is stored periodically with a timestamp as an independent variable in the system.

  This counter has to be associated with the samples to complete their information.

More details about this stage are provided in the section explaining the methodology.

Feature Selection and Extraction

During this step, a set of statistical values describing “features” of the signals in the time domain are obtained. These features are aggregations and calculations performed on specific parts of the signal that might lead to conclusions about the degradation of the EMA over time.

The reason for calculating a group of several features is to try to detect different problems from different angles, and thus, they complement to each other.

For example, some of these features are good at detecting mechanical vibrations caused by faults, while others are more appropriate for measuring the impulses originated in the early stages of faults.

Features will be extracted in a given signal for each section representing a cycle in the test, and a cycle counter has to be asociated with this value.

The complete list of features and the strategy for feature extraction is presented later in the document in a specific section.

Feature Analysis

Once the features are obtained from the signals, an analysis can be performed on them.

Due to the nature of the fatigue test, data obtained during the process is unlabelled, which means that a strategy must be found without using the typical mechanisms of supervised methods.

In this respect, an alternative for this analysis can be found in the work of López de Calle et al. (2019), where a fault detection method is presented during the real-time monitoring of a machine, based on dimensionality reduction and statistical process control (SPC) techniques.

This method assumes a monotonic degradation of the asset during the process, so that an assessment can be made in the absence of previous performance experience in working conditions, or in the event of a non-labelled dataset, as it is the case of the scenario described in this work.

The difference between this procedure and a typical condition monitoring approach can be summarised in the following picture:

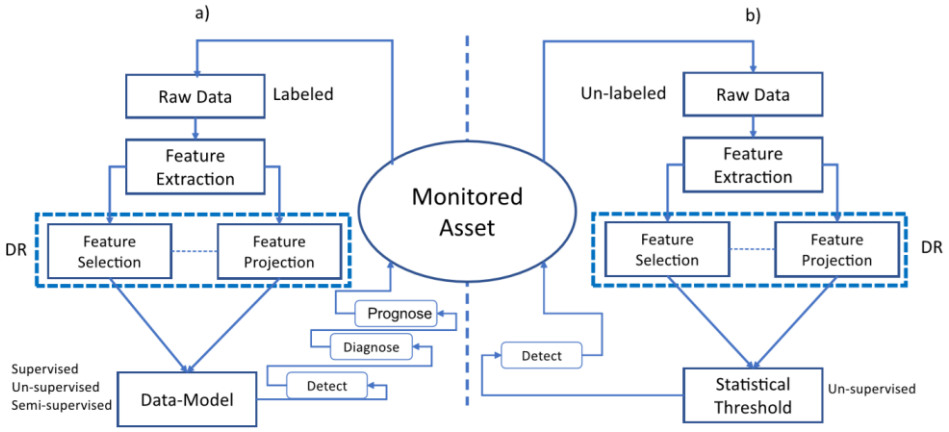


Figure 7: (a) Typical condition monitoring approach (b) Reference approach (Source: López de Calle et al., 2019)

The steps to be performed during the analysis phase are as follows:

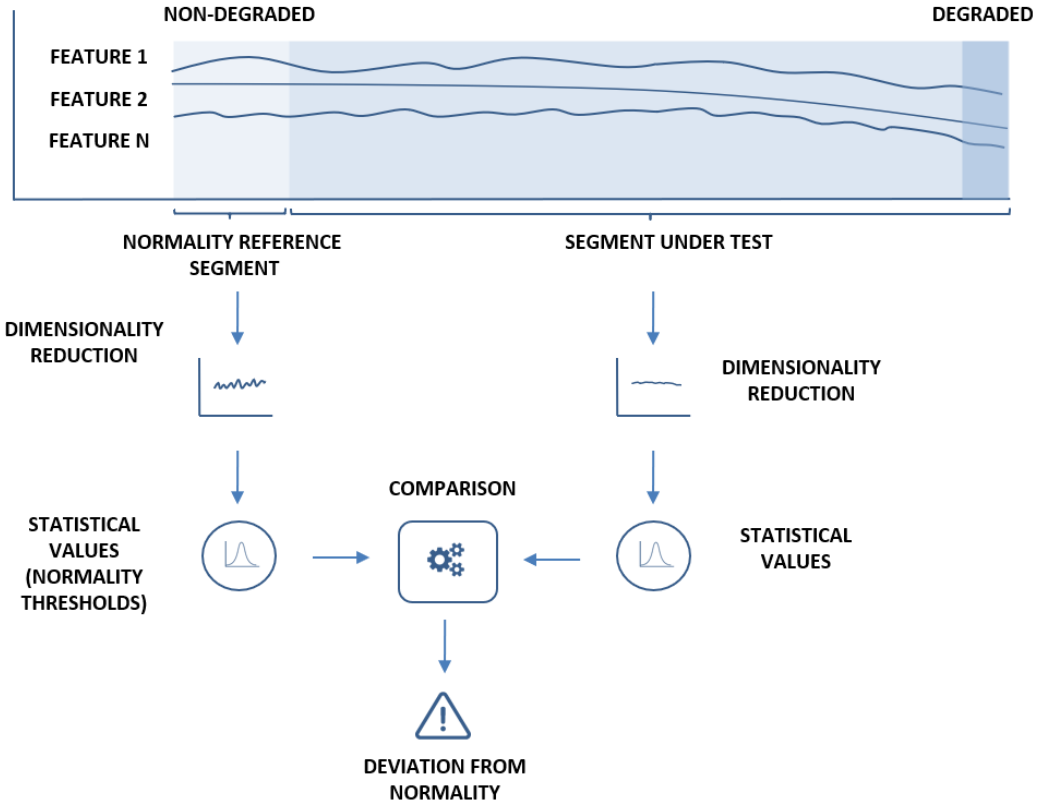


Figure 8: Analysis process diagram

As can be seen in the figure above, a multidimensional statistical process control, together with a previous dimensionality reduction technique, will be implemented to obtain a single-dimensional parameter from a non-degraded section of the features, and statistical thresholds will be estimated representing normality.

Thus, a comparison can be made with another single-dimentional parameter, obtained in a similar manner, coming from the other remaing section of the features, where the asset could be in a potencially degraded status, in order to detect deviations from normality.

Finally, a classifier will obtain those features maximizing the separation between the two different classes (non-degraded, degraded).

More details about this process are provided later in the section dedicated to feature analysis.

Analysis Automation

The goal of this stage is to automate the analytical methods developed in the previous step, so that results can be collected on a periodic basis for the monitoring and evaluation of the EMA degradation.

This automation will be created on a platform developed in the Azure cloud, and the results will be presented on a dashboard also hosted in the cloud. More details about this are described later.

1.4 Structure of the Document

Chapter 1 presents the background of the work, with a description of the EMA advantages and their current challenges. Finally, the chapter describes what is the problem to solve and the general approach to the solution.

Chapter 2 describes a state-of-the-art in EMA health monitoring, and different strategies used for this purpose.

Chapter 3 makes an introduction to the experimental environment, the data acquisition hardware and sensors, and a description of the fatigue test.

Chapter 4 introduces the structure of the metadata and result files that are generated by the experimental setup.

Chapter 5 explains the details of the different steps of the solution, including the techniques used for preparing the data, the analysis of the features, and the automation and presentation of the results.

Chapter 6 focuses on the details of the data analysis, and the evaluation of the results.

Chapter 7 summarises the main achievements of the work and suggests future improvements.

2 State-of-the-Art

Health monitoring has been considered by several recent international research programs as a key enabler for a wider use of EMA in the aviation industry (van der Linden et al., 2016). In particular, the problem of jamming, EMA remaining useful life and preventive maintenance have been highlighted as important areas of study (Todeschi, M., & Baxerres, L., 2015).

The following sections describe different research works in recent years focusing on EMA health monitoring. These studies are divided into three main groups:

- Data-driven approaches
- Model-based approaches
- Hybrid approaches

2.1 Data-driven Approaches

Data-driven approaches generally know very little about the system, and execute data processing techniques directly on sensor data, which is collected from experimental test benches or in-service systems. Some of the techniques used for data-driven aproches can involve the use of statistical models, neural networks, frequency domain analysis, or wavelet analysis, among others (Chirico & Kolodziej, 2014).

The goal here is to detect patterns in signals where the behaviour is different from normality, which may lead to conclusions about the condition of the EMA under study.

The main benefit of these methods is that they are not limited to a specific type of actuator, as is the case of model-based procedures, but they normally need to collect a faily large amount of information to establish normality and identify anomalies.

The approach followed by Mazzoleni et al. (2019) within the European H2020 REPRISE project is to devise health monitoring methods for a primary flight control surface EMA based on a multivariate Statistical Process Control (SPC) chart.

This approach shares things in common with respect to the SPC method proposed in this work, namely:

- A Hotelling's multivariate control chart, which in the case of Mazzoleni et al. also serves for the development of health monitoring indicators.
- The design of the experiments, with a fatigue test oriented to gradually wear down the EMA, and condition tests.

This document's work, however, presents several differences. On the one hand, the number of signals captured is greater, as Mazzoleni et al. only deal with EMA controller signals (motor phase currents), as no additional sensors have been added to the bench. On the other hand, the amount of information acquired is much higher.

The Hotelling's multivariate chart technique is also used in the work by Ruiz-Carcel and Starr (2018). Their paper describes a data-driven method for mechanical fault detection and diagnosis that produces a condition indicator based on features extracted exclusively from two signals generally available in an EMA controller: electric current and position measurement.

Tests were conducted in a test rig to evaluate the method with steady and transient operations. The features were explicitly selected to detect degradation problems for both types of operations, and the PCA technique was applied to reduce data dimensionality.

During the experiments, seeded faults were introduced to simulate mechanical problems, in particular lack of lubrication, spalling (metal flaking), and backlash. Then, a comparison with normality was done through the Hotelling's chart method.

This approach shares some similarities with the work presented in this document, but as was commented previously, the number of signals and the amount of information is lower. In addition to this, the experimental setup is much simpler and fewer computations are made.

Also, the faults were manually introduced in the system, whereas the aproach followed in this work is to execute a fatigue test until real degradation occurs, and the EMA is a real prototype from an aeronautics OEM.

(Isturiz et al., 2016) describe in their work a data-driven health monitoring technique where the EMA health status is determined by a combination of a usage module and a health module. The usage module computes EMA usage data, such as operational hours, number of cycles, or average load. In parallel, the health module executes health monitoring algorithms for detecting several failure modes. Finally, a health assessment module combines both outputs to determine the EMA health status.

The use of data-driven semi-supervised algoritms for anomaly detection has been proposed in the literature in recent years to address the scarcity of anomalous data in EMA systems.

One example of this paradigm can be found in the work of Pang et al.(2018) with the use of Gaussian process regression (GPR) and relevance vector machine (RVM) for anomaly detection in data series coming from sensor monitoring, although their method can be extended to other probability prediction models. They develop in their work a graphical

indicator of the receiver operating characteristic curve of prediction interval (ROC-PI), which is based on the ROC curve, to measure the model performance.

Another example of the use of semi-supervised algorithms is presented in the work by Zhang et al. (2017), where the estimation of the remaining useful life (RUL) is improved by the use of ensemble learning with a weighted bagging Gaussian process regression (WB_GPR) method. Experimentations were conducted to validate the results, where the bagging GPR algorithm showed a better performance for EMA RUL prediction than the standard GPR procedure.

Yang et al. (2019) present in their work a data-driven method with a recurrent neural network to consider the time dimension of EMA sensor data for fault detection and isolation. They propose some improvements in a standard long short-term memory (LSTM) network to achieve a better classification accuracy and training performance in the model, and to allow for correlation between sensors.

Chirico & Kolodziej (2014) chose a data-driven fault classification technique based on frequency domain features extracted from EMA signals (motor current, position, and motor velocity data) and accelerometers (vibration data). PCA was used to choose the most effective features, which were the input for a Bayesian classifier, and the results were validated on an experimental setup.

2.2 Model-based Approaches

Model-based approaches involve the development of mathematical formulations based on physical principles to determine the output of an EMA for a specific input. The error between the real and the model output provides the basis for obtaining indicators that can be compared against reference values for health assessment (Chirico & Kolodziej, 2014).

These approaches have the advantage that faults can be linked back to model parameters, so a meaningful physical explanation can be provided. The main drawback is that these models are very hard to be obtained, and very specific to the use case.

An example of a model-based method for EMA health monitoring can be found in the work by Dalla Vedova et al. (2016). Here, a fault detection and isolation (FDI) algorithm is developed based on an EMA simplified model to decrease computational requirements in testing, but accurate enough to simulate the effects of all the target faults.

Dalla Vedova et al. (2016) also developed a model-based optimization technique to implement a damage estimator for a scenario of combined failure modes in an EMA due to progressive

wear. The model uses a simulated annealing algorithm for identifying the incipient signs of three different configurations of seizure and backslash faults. The model was validated on a test bench, showing that the results and execution times of the optimization model were correct, with less accuracy as failure severity was lower.

Di Rito & Schettini (2018) used position tracking as the basis for the development of EMA health monitoring models to detect major faults in position-controlled flight controls. This approach offers the advantage of simplifying the software and reducing the number of additional sensors needed for the health monitoring functions.

The Matlab/Simulink model developed by Maggiore et al. (2014) focused on detecting early symptoms of two types of failures that can be expected to occur in EMAs: mechanical faults associated to progressive wear, such as friction and backslash, and also BLDC motor (electronically commutated motor) failures, specifically coil short-circuits, and the bearing wear generating rotor static eccentricity.

2.3 Hybrid Approaches

Hybrid approaches use a combination of model-based and data-driven methods to complement their strengths and weaknesses.

Ismail et al. (2016) presented a health monitoring system using a vibration-based hybrid technique for detecting the incipient symptoms of two types of latent EMA mechanical faults: ballscrew partial jamming, which is an example of a functional fault affecting the EMA performance by hindering its operation, and ballscrew spalling, a type of non-functional fault which may have a minor immediate impact on the EMA's performance, but that can evolve towards a functional fault.

The technique compares a set of model-based features against features obtained on an experimental setup, without the need of labelled data for training a supervised model.

Van der Linden et al.(2016) stated that simulation with damage components can help in the design of EMA health monitoring algorithms (in fact, it is nowadays a new research area), but tests have to be run on complete EMAs as there is no previous experience in this field to validate the results.

Using this approach, they presented a decade of work developing EMA health monitoring algorithms with the aid of different test benches, as a collaborative effort between Liebherr Aerospace in Lindenberg and the DLR Institute of System Dynamics and Control.

Zhang et al. (2018) applied in their work a hybrid technique for identifying EMA faults by calculating an estimation of motor voltage data from raw monitoring voltage values. The estimation is needed as raw voltage data can be affected by electromagnetic interference due to changes in operation conditions, which may mask potential faults.

To make this voltage estimation, they used a Kalman filtering algorithm where the state equation is replaced by a physical model of current and voltage, as the state equation is difficult to be obtained for an EMA.

As part of a NASA-funded program, Balaban et al. (2015) combined both model-based and data-driven approaches to develop a diagnostic and a prognostic mechanism to diagnose EMA faults, and to predict the EMA remaining useful life (RUL), respectively. The results were tested in a series of experiments executed in flight.

3 Experimental Setup

3.1 Test Bench

The testbed used for experimenting with the EMA is shown in the following pictures:

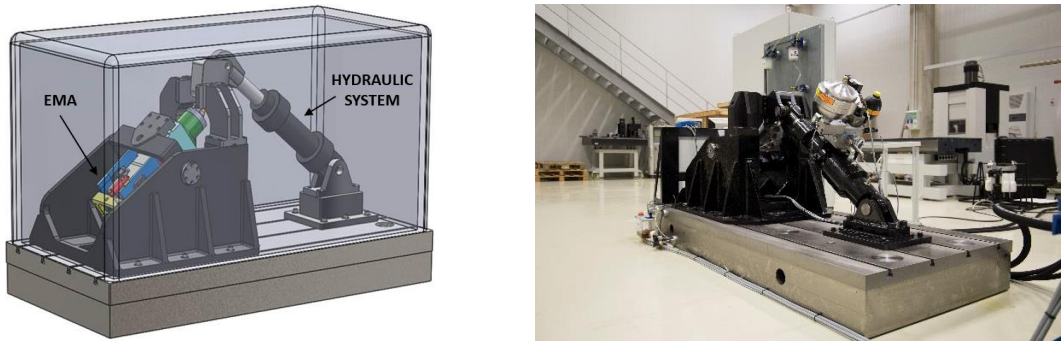


Figure 9: EMA test bench at Tekniker facilities

The actuator (the component that can be seen to the left on both pictures above) is mounted inside a metal housing that holds it, and it is placed at 32º, which is the operational angle on the aircraft.

The hydraulic system is situated right in front of it (to the right in the above pictures), and simulates the load during the flight, which is applied perpendicularly to the EMA. The actuator must in turn exert a force to counteract this force and move the primary flight control surface simulated by the hydraulic system.

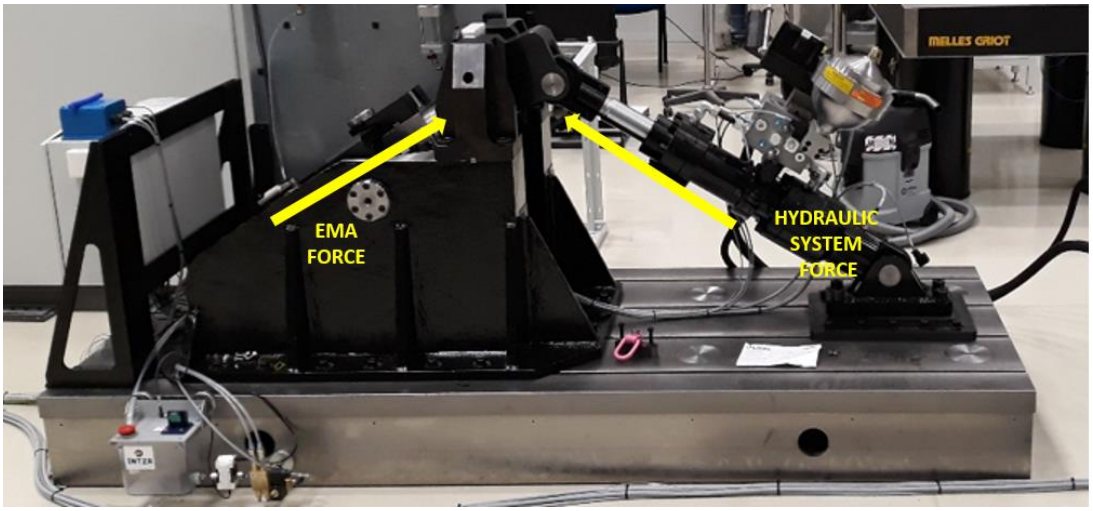


Figure 10: Forces applied by the EMA and the hydraulic system during the tests

The test rig is equipped with the necessary elements to prevent the overrun and overcharge of the actuator when it is running.

A control system is in charge of the testbed automated operation, and an acquisition platform obtains the different information coming from the tests. These tests can be divided into two groups:

- Fatigue test.

  This kind of data is acquired periodically (every 10 minutes) during a short period of time (few seconds, in order to preserve storage space) when the bench simulates the typical EMA operation.

  This simulation is run continuously in a loop, and information related to the EMA usage is captured, such as speed, position, number of cycles, etc.

  For safety reasons, an alarm would be generated each time certain conditions are met.

- Condition tests.

  In order to acquire this information, the normal operation loop of the bench must be stopped, so that specific tests can be performed under controlled conditions.

  The goal of the information acquired during these tests is to generate features which can be used to determine the actuator's condition by means of what is called a "Fingerprint" (Ferreiro et al., 2016).

  The concept of fingerprint states that each asset has a set of features characterising its condition, each of which has unique safe operating limits, the same way as the patterns on our fingertips are unique to each individual person.

  Given that those features are processed periodically in tests conducted under the same conditions, it is possible to make some conclusions about the EMA health state by comparing them over time.

3.2 Data Acquisition System

The data acquisition system basically consists of the following elements:

- Two acquisition devices.
- A set of sensors.
- A computer running a Matlab application.

The two acquisition devices used during the experiments are the following:

- A CompactDAC system from National Instruments (cDAQ-9171), which is a Hi-Speed USB 2.0 interface chassis for an Input/Output C Series module (NI-9223), aimed at reading data coming from acoustic emission sensors.

  The reason for having a separate acquisition system solely for acoustic information is the high frequency requirements for these readings.

Figure 11: cDAQ-9171 and NI-9223 acquisition devices

- An Ingesys IC3 device, which is a general-purpose control system in charge of gathering information other that acoustic emissions:

Figure 12: Ingesys IC3 acquisition device

The following sensors have been included on the bench:

- Thermocouples for registering the temperature in the room and in different critical positions on the actuator.
- A 3-axis high frequency accelerometer (CTC, AC230-2D/006M-F3C) with an acquisition rate of 25 KHz, and a measurement range from 0.6 Hz to 10 KHz, enough to cover the entire frequency spectrum in the test rig.
- An acoustic emission sensor (Kistler 8152B1), with an acquisition rate of 1 MHz, and a measurement range from 50 KHz to 400 KHz.
- A current sensor (LEM HTA 100S) installed in the electric cabinet for measuring the current consumed by the motor. The acquisition rate for this sensor is 25 KHz.

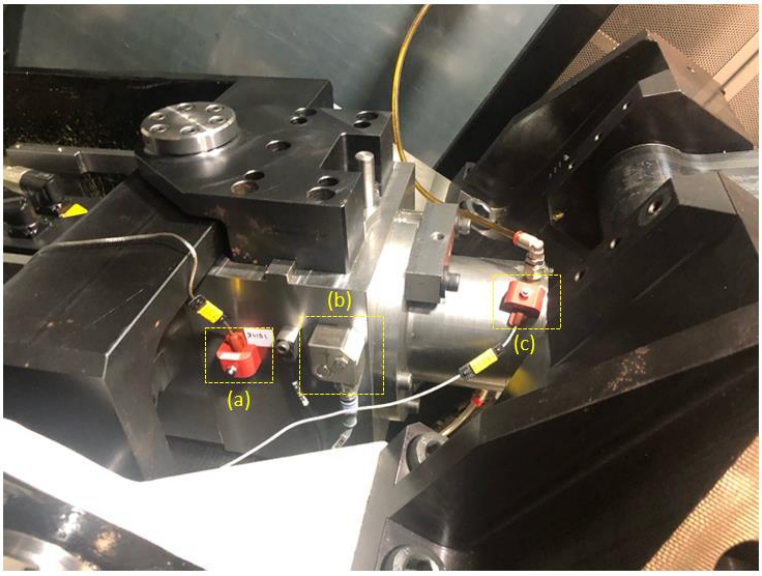


Figure 13: (a) Ball-screw rear temperature (b) Accelerometer (c) Ballscrew tip temperature

3.3 Fatigue Test

The EMA prototype is designed to perform an estimated $14.55 \cdot 10^6$ number of cycles, with each cycle spanning 36 seconds, which means an asset lifetime of approximately 16 years.

The strategy during the test is to accelerate the process by reducing the cycle length to 0.75 seconds, thus shortening this period to 4 months.

The following figure shows a cycle period on both the hydraulic system applied force and EMA position signals with the estimated length of 0.75 seconds mentioned above:

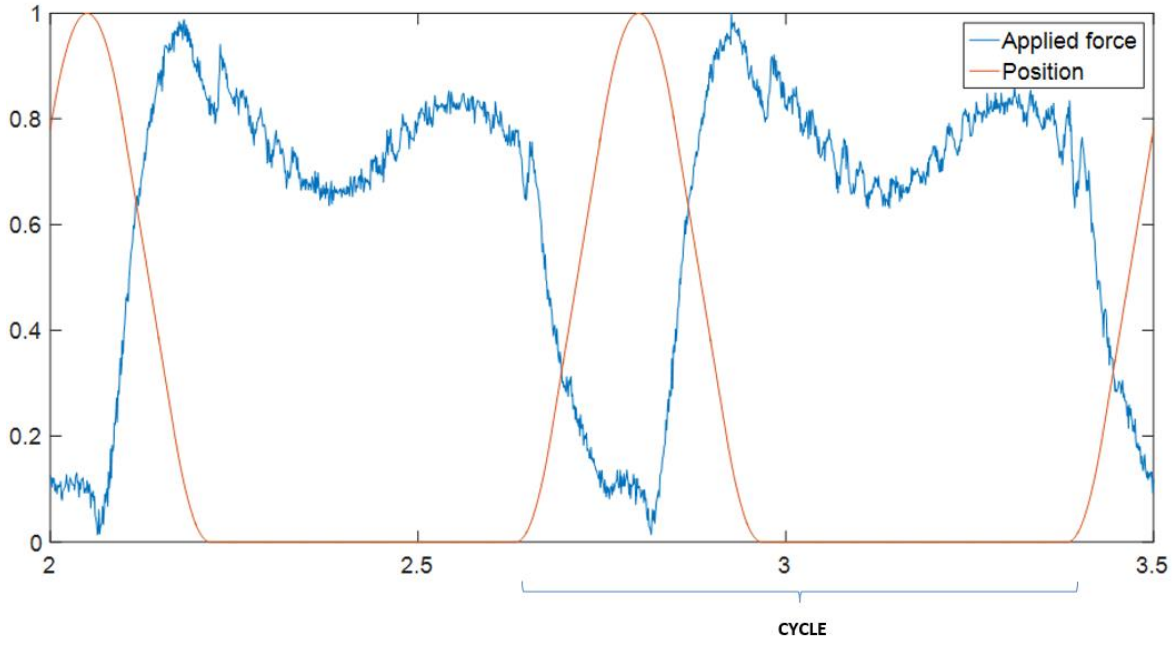


Figure 14: Cycle period on the hydraulic system applied force and EMA position signals

4 Exploratory Data Analysis

During the execution of tests, data and metadata are logged into a local MySQL database. All the information coming from the experiments is saved there, as well as all configuration settings for the bench and for the different sensors and actuators.

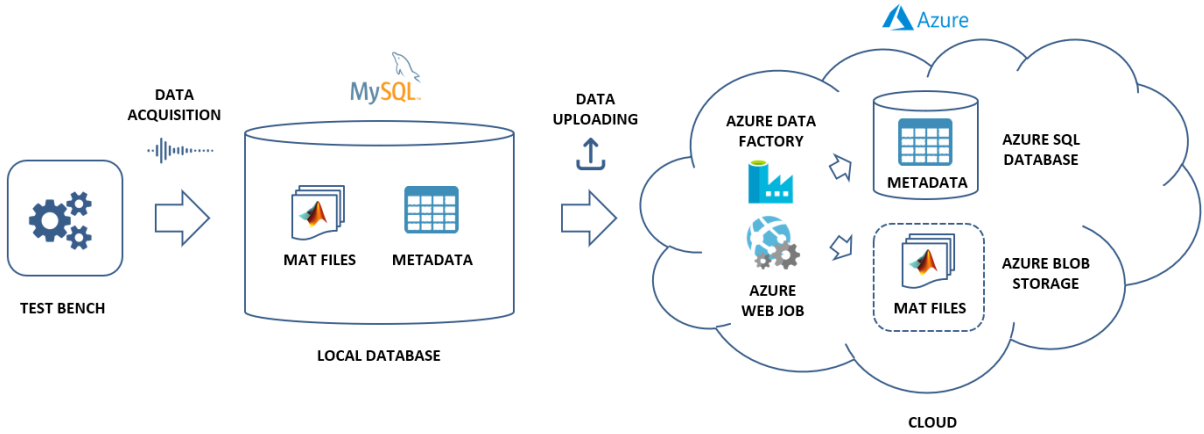


Figure 15: Scheduled task for uploading data on the cloud

Test results with electrical units are saved into Matlab MAT files, which are embedded in database table records. Metadata information, on the other hand, is registered directly in database tables.

A scheduled task (an Azure WebJob running a complementary Azure Data Factory task) runs once a day and performs the following tasks:

- Matlab MAT files are extracted from the MySQL database and copied on an Azure blob storage due to the potentially large volume of data produced during the experiments. The original files are deleted from the MySQL instance to save space on the local disk.
- MySQL database tables (without the embedded MAT files) are synchronised with an Azure SQL Database instance. This task is performed by an Azure Data Factory pipeline. This will help the development of an analytical platform on the cloud.

The Azure blob storage files and the Azure SQL Database instance data will be used as the core infrastructure during the preprocessing and analysis stages.

4.1 Test Bench Database Schema

The test bench has been designed in such a way that can be used for different actuators and experiments. This means that the local database schema included on the platform has to be flexible enought to handle this variability.

The highest level information concepts on this schema are explained below. These concepts have a corresponding database table, which will be used to filter test results during the preprocessing phase:

- Test campaign.

  Experiments and configurations are organised into campaigns in order to group the information related to a particular project or client.

- Test type.

  It is a sequence of steps defining the purpose of the experiment, for example:

  - To start the system.
  - To apply a constant force on the EMA.
  - To make a displacement at constant speed.
  - To release the force on the EMA.
  - To capture some signals during the previous steps.
  - To stop the system.

- Test.

  It is an instance of a test type with explicit values to perform the different steps. Continuing with the example described above, the test should define the following parameters:

  - The value of the constant force to be applied.
  - The length of the displacement to be made.
  - The value of the constant speed during the displacement.

- Execution.

  It contains the results of a test, with information such as:

  - Start and end execution timestamps.
  - All the values registered during the execution.

- Result.

  It is a particular outcome of an execution (more than one result is possible per execution). It can be a piece of data coming from the Ingesys or Compaq acquistion systems, or metadata saved on the database describing the context of the execution.

The entity-relationship diagram presented below shows how these core concepts relate to each other, together with other tables showing other entities. These tables will be used to filter the results that are going to be processed (results belonging to the EMA fatigue test).

The columns used in the query to get the results during the preprocessing stage are highlighted in the picture (this query is presented later in the document):

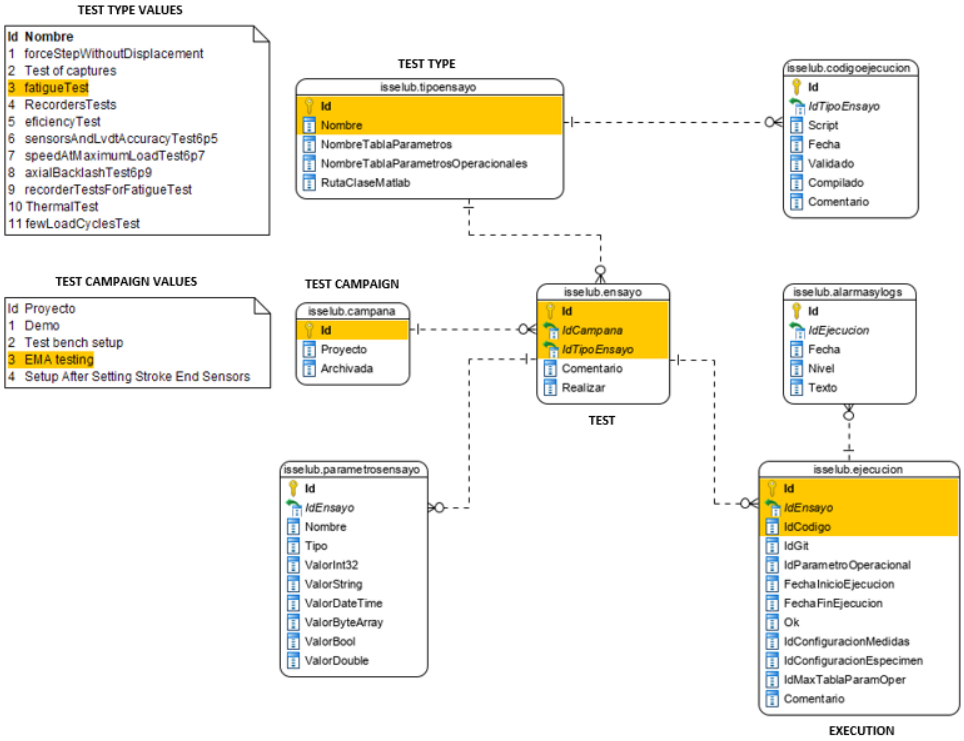


Figure 16: Entity-relationship diagram for the core application concepts

The results and metadata for a specific test execution are stored in the database in a series of entities related to the automation system controlling the test bench. These entities can be divided into three groups from the point of view of their role in this automation platform (input, output, signaling): measurement elements, actuation elements and internal signals.

The entities belonging to these groups can be seen in the following picture (there is a table in the database for each of these entities):

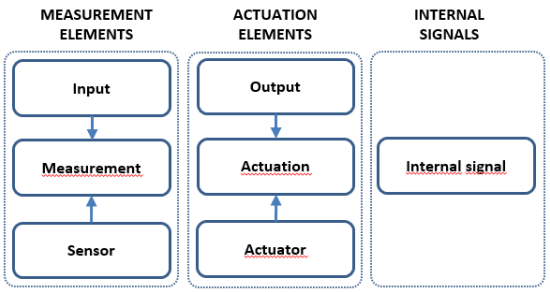


Figure 17: Groups of entities in the database for storing signal metadata

During the experiments, signals with electrical units are stored in the MAT files associated with alphanumeric identifiers representing inputs, outputs or internal signal (i.e., some of the entities presented in the figure above. The description of all these entities is provided a little later).

For example, samples coming from the accelerometer measuring Y-axis vibrations are stored associated with the alfanumeric identifier "INvalue1201AIO2Ts0p00002s", which represents a specific input in the automation environment.

These alphanumeric symbols are not useful for the analysis. Instead, the name of the signal is needed, which in the previous example is "Accelerometer_Y". In addition to this, some metadata (polynomial coefficients) is also required for the conversion from electrical to physical units.

All this information can be obtained through a query to the tables representing these entities in the automation system.

For a better understanding of the process from a programming point of view, these concepts are explained below:

- Sensor: it represents a sensor device and its associated calibration information. Each time a sensor is calibrated, a new sensor registry is created on the database.
- Measurement: this concept is used to postprocess the information coming from a particular sensor through a specific PLC input. Data is transformed based on the sensor’s calibration information, and on a linear transformation equation provided by the measurement.
- Input: it is an analog or digital input on the acquistion equipment (Ingesys or CompaqDAC) for signals coming from a sensor device.

The actuation elements are explained below:

- Actuator: it is a component in the automation system for moving and controlling a mechanism (not to be confused by the EMA itself). It is equivalent to a sensor in the measurement element group.
- Actuation: similar to a measurement, but for actuator outputs.
- Output: it represents a PLC analog or digital output.

Finally, internal signals are represented by a single concept with the same name. It is used to store information related to signalling in the automation system, or calculated data.

The tables and columns involved in the generation of the query obtaining the name and metadata of the signals (polynomial coefficients) are highlighted in the diagram depicted below (this query will be shown later in the document):

Figure 18: ER diagram for measurements, actuation elements and internal signals

4.2 Test Result Types

The following image shows the different result types generated from the two acquisition systems:

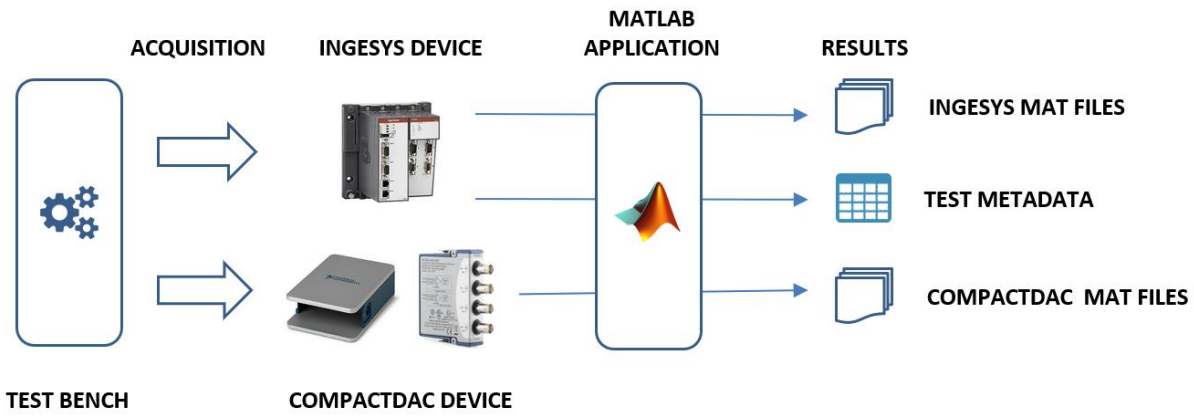


Figure 19: Types of results

These types of results are described in the following sections.

MAT files

The Matlab application running on the test bench workstation generates a specific MAT file for the CompactDAC and Ingesys devices each time a data acquisition is made. These files will be referred to as “CompactDAC” and “Ingesys”, respectively, in the document.

These MAT files store a group of signals with electrical units from inputs, outputs and internal signals in the automation system. Signal data has to be extracted from these files, transformed, and serialized in a convenient format, so that features can be created later for the analysis stage.

Ingesys MAT files have signals with three different samplings (namely 2000, 100000 and 20 elements per signal, respectively). The internal structure of these files is the following:

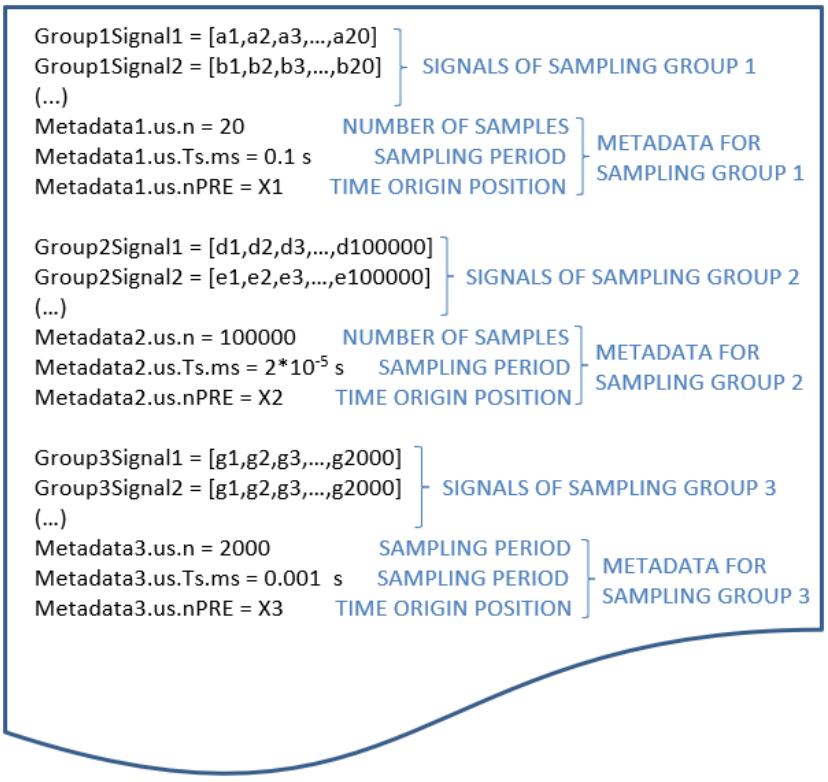


Figure 20: Internal structure of Ingesys MAT files

In the figure above, “GroupXSignalY” and “MetadataX” are placeholders for input, output, and internal signal identifiers.

Ingesys signals have three sampling groups, with sampling periods of 0.1, $2*10^{-5}$ and 0.001 seconds, respectively.

There are three special metadata parameters for the signals belonging to each sampling group, which are used to store the number of samples, the sampling period and the time origin position (the meaning of this value is explained later). These parameters have special name endings (“.us.n”, “.us.Ts.ms”, “.us.nPRE”).

This is an excerpt of a real Ingesys file shown with RStudio, where the signals belonging to a specific sampling group are highlighted, together with their metadata:

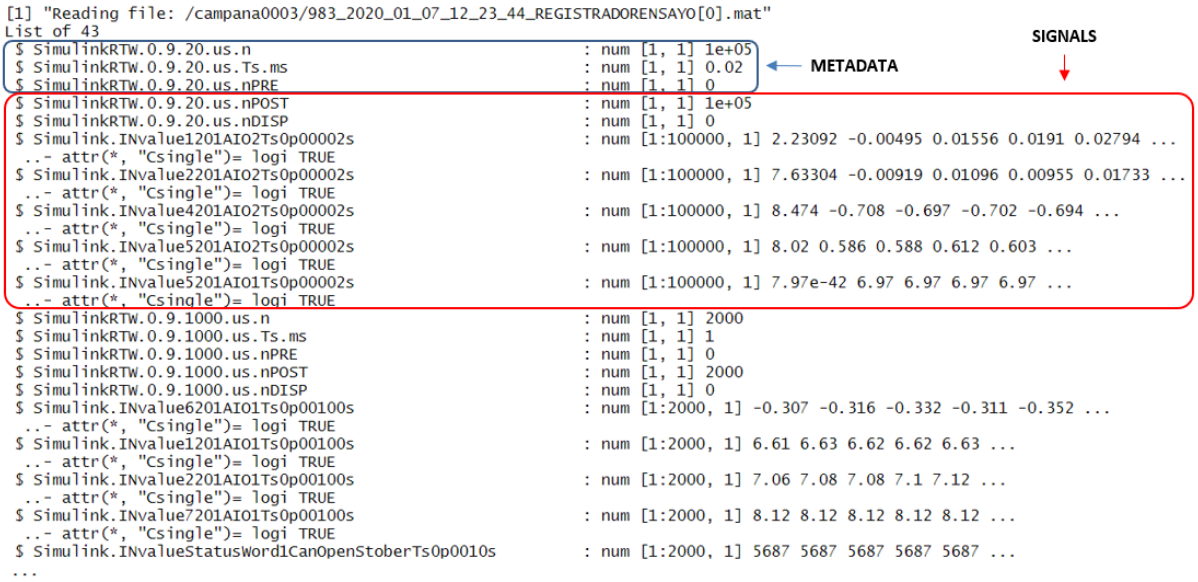


Figure 21: Partial view of a real Ingesys file with RStudio

CompactDAC files have only one sampling group (4000000 samples per signal). Parameter “Time” will be used to obtain the sampling period by subtracting two consecutive timestamps. The method to obtain the final relative timestamps associated to the samples is explained later.

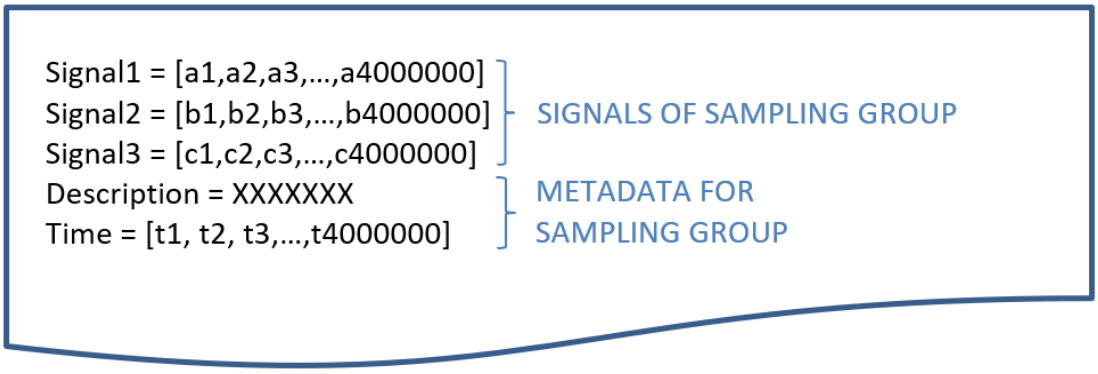


Figure 22: Internal structure of CompactDAC files

This is a real CompactDAC file showed with RStudio:

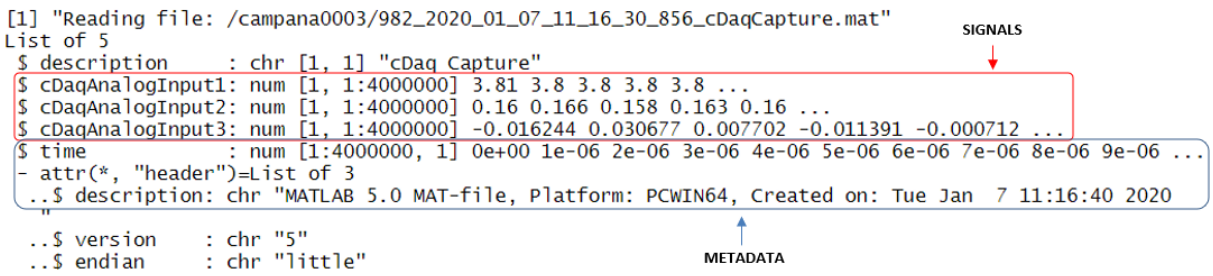


Figure 23: View of a real CompactDAC file with RStudio

The timestamp when the MAT file was created is included in the filename for both Ingesys and CompactDAC, with the pattern “yyyy_MM_dd_hh_mm_ss”. Here are some examples:

- Ingesys: 2020_03_01_14_58_05_REGISTRADORENSAYO[0].mat
- CompactDAC: 2020_03_01_13_49_56_683_cDaqCapture.mat

Test metadata

In addition to MAT files, there are other variables that are sampled periodically through the Ingesys equipment, and this information is saved on a specific database table (“ContextData”) as test metadata.

Some columns of this table are presented on the following picture, showing different records captured at different moments in time belonging to a specific test result (there are more records than can be seen on the image for the highlighted result):

Results Messages

	Id	IdResultado	Fecha	word0valueF201DI1	word1value0201DI1	word0valueB201DI1	word0valueC201DI1	word0valueD201DI1	word0valueE201DI1	word1value2201DI1	word1value3201DI1	value0201AI1	valu
75519	75519	3205	2020-05-06 11:52:51	1	1	0	1	1	1	1	1	14,2186803817749	9,51
75520	75520	3205	2020-05-06 11:53:02	1	1	0	1	1	1	1	1	14,2176837921143	9,51
75521	75521	3207	2020-05-07 06:51:06	1	1	0	1	1	1	1	1	14,2176837921143	9,62
75522	75522	3207	2020-05-07 06:52:14	1	1	0	1	1	1	1	1	14,2166872024536	9,58
75523	75523	3207	2020-05-07 06:52:29	1	1	0	1	1	1	1	1	14,2186803817749	9,57
75524	75524	3207	2020-05-07 06:52:44	1	1	0	1	1	1	1	1	14,2057285308838	9,56
75525	75525	3207	2020-05-07 06:52:58	1	1	0	1	1	1	1	1	14,2166872024536	9,56
75526	75526	3207	2020-05-07 06:53:09	1	1	0	1	1	1	1	1	14,2156915664673	9,56
75527	75527	3207	2020-05-07 06:53:20	1	1	0	1	1	1	1	1	14,2156915664673	9,56
75528	75528	3207	2020-05-07 06:53:31	1	1	0	1	1	1	1	1	14,2156915664673	9,54
75529	75529	3207	2020-05-07 06:53:42	1	1	0	1	1	1	1	1	14,211706161499	9,53

Figure 24: Example of a database table test result

In contrast to test metadata, Ingesys and CompactDAC MAT file sampling is done at a very high frequency, and the time during which sampling is made is restricted to a few seconds only.

The following picture shows a view of the acquisition sequence for the different types of results. Database results are captured between longer periods of time than MAT files (the proportion

of samples between both types of results shown is this image is not the actual ratio. It is just for representation's sake):

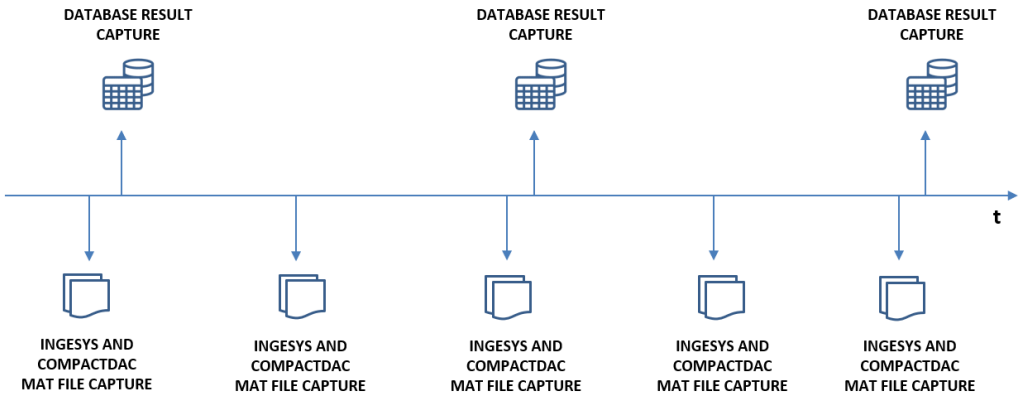


Figure 25: Data capture overview for the different types of results

5 Methodology of Work

The following sections explain the details of the steps described in the section devoted to the data processing approach:

- Signal Preprocessing
- Feature Selection and Extraction
- Feature Analysis
- Analysis Automation

5.1 Signal Preprocessing

The steps that are taken on this stage are the following:

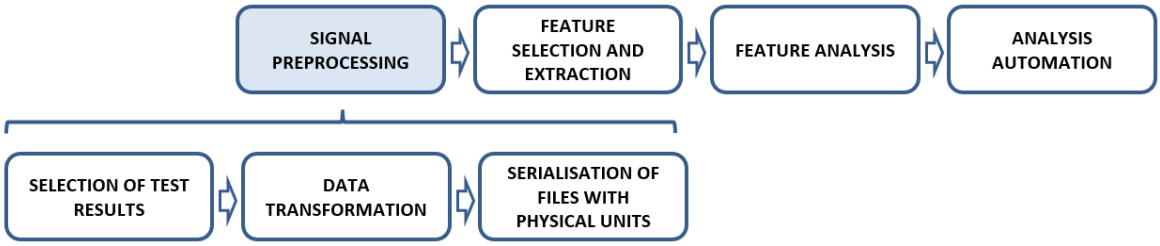


Figure 26: Steps during the signal preprocessing phase

The outcome of this stage for each type of result can be seen here:

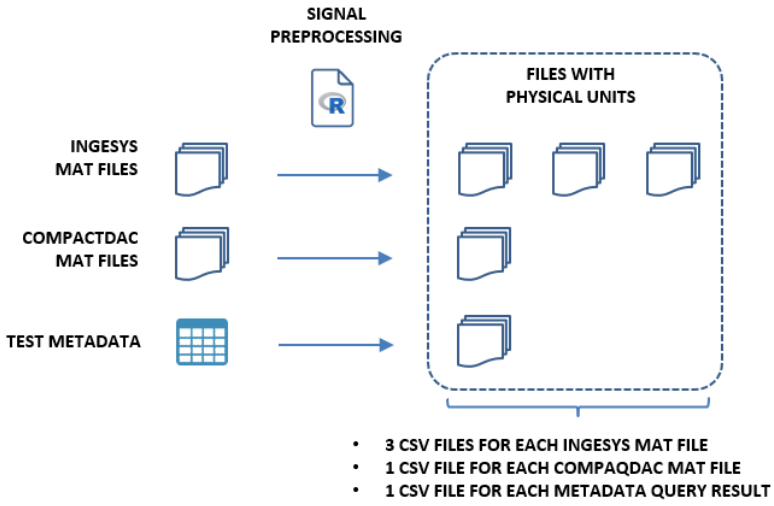


Figure 27: Outcome of the preprocessing phase

The output (a series of files, as can be seen in the previous picture) is going to be generated by a program written in R (see annex).

The details of the steps during the preprocessing phase (performed by the R program) are provided in the following sections.

5.1.1 Selection of Test Results

A SQL query will obtain the fatigue test results, from among the other tests that are also stored in the database. The details of this query can be seen in the annex.

The “RODBC” package has the functionality to connect and send the query to the Azure SQL Database instance.

Next, the MAT files corresponding to those results have to be located and obtained from the Azure blob storage. The “AzureStor” package provides the necessary methods to accomplish this task.

This is the location of the MAT files on the Azure blob storage container (the image has been taken from the Azure Storage Explorer utility):

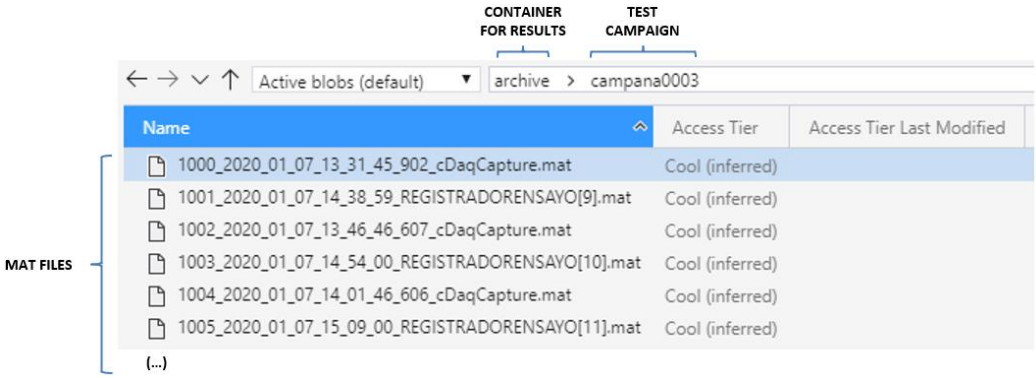


Figure 28: MAT files location on the Azure blob storage

Then, each MAT file has to be parsed and loaded into the R environment, and this is done through the “R.matlab” package methods.

5.1.2 Data Transformation

The goal during this stage is to extract the signals and metadata information contained in MAT files, which are compressed in a binary format, and create a set of R dataframes where signals are arranged in columns, and their values are converted to physical units. The name of each signal, which has to be read from the database, will be the name of the columns. Also, an extra dataframe column will be created with the timestamp for each sample of data. Therefore, signals with a common sampling rate must be placed in a different dataframe.

On each MAT file, signal data (in electrical units) are associated with alphanumeric strings representing memory positions on the acquisition equipments, which are input/output data channels for sensors, actuators, or internal signals.

Those alphanumeric strings have to be read from the files, so that a query to the database can be made to obtain the name of these signals and their metadata. This query can be seen in the annex of this document.

The following picture highlights one of these alphanumeric strings, corresponding to an input in the CompactDAC system. This input is the specific channel in the CompactDAC system where the values of signal “Acoustic Emission RMS” are read. The name of this signal (and the metadata for the conversion to physical units) is obtained through the query mentioned above, using the alphanumeric string as a filter:

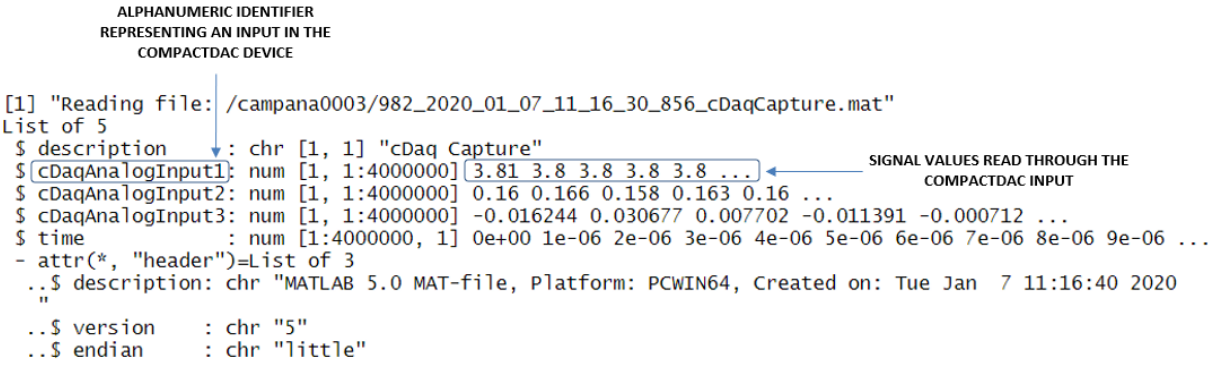


Figure 29: Example of an alphanumeric identifier in a MAT file

For each MAT file, signals with the same sampling frequency are grouped together in the same R dataframe, creating one column for each signal. There are 3 sampling periods in Ingesys files so 3 dataframes are created here, and only one sampling period in Ingesys, resulting in one dataframe. The column names in the dataframes are generated with the names of the signals.

Next, each value of every dataframe column is transformed with linear equations, whose coefficients are stored in the database as metadata with the format “[a1, a0]”, meaning the following polynomial:

$$f(x) = a1x + a0$$

Two potential transformations can be applied (one after the other) to CompactDAC and Ingesys data. The first equation represents potential calibration adjustments in the sensor or actuator, and the second one allows the conversion from electrical to physical units:

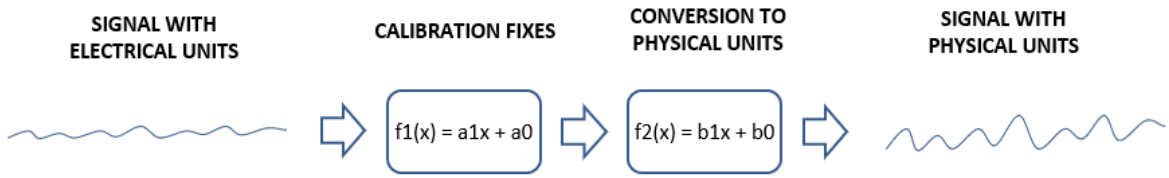


Figure 30: Signal transformation from electrical to physical units

Once transformations are done, the next step is the creation of a relative timestamp column for each R dataframe. These relative values represent the offset in seconds for every signal sample with respect to the absolute timestamp of the file (which is located in the filename).

Creating relative timestamps (in milliseconds) is more convenient than absolute timestamps, as the sampling frequency is very high.

The timestamp vector to be generated for each dataframe has the following structure:

$$-N*T, \ldots, -3*T, -2*T, -1*T, 0, 1*T, 2*T, \ldots, M*T$$

where T is the sampling period corresponding to the sampling group, and the position where time = 0 is the synchronization point between different sampling groups and acquisition equipments.

This is an example of a timestamp vector, where t = 0 is on the fourth position, T is the sampling period, and elements "vXY" are the values of signals:

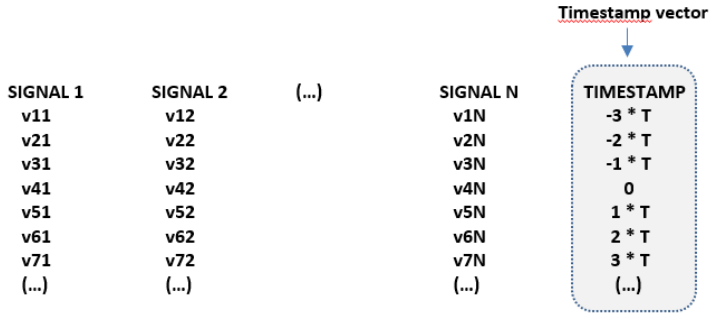


Figure 31: Example of a relative timestamp vector in a dataframe created from a MAT file

The process for obtaining the position where time = 0 is different in Ingesys and CompactDAC MAT files. In the case of Ingesys, it is a two-step process:

- Synchronization between sampling groups of the same Ingesys file.

  For each sampling group, the metadata parameter ending in ".us.nPRE" has to be found. This value provides the position where time = 0 for the corresponding sampling group signals.

  This means that the timestamp vector for each sampling group has to be build taking into account this value, as well as the sampling period (provided by the parameter ending in ".us.Ts.ms") and the number of samples (read in the metadata variable ending in ".us.n").

  The result is that the different sampling groups "shift" between each other on the time axis to adjust to a common time = 0 (this can be seen graphically in the following picture).

- Sychronization with respect to the CompactDAC device.

  The transition from 0 to 5 in signal "Synchronization_signal_for_NI" (which can be found in the Ingesys file) is the synchronization point with CompactDAC.

The timestamp vectors for all the sampling groups in the Ingesys file have to be synchronised with the CompactDAC signals based on this transition point, as can be seen in the figure below:

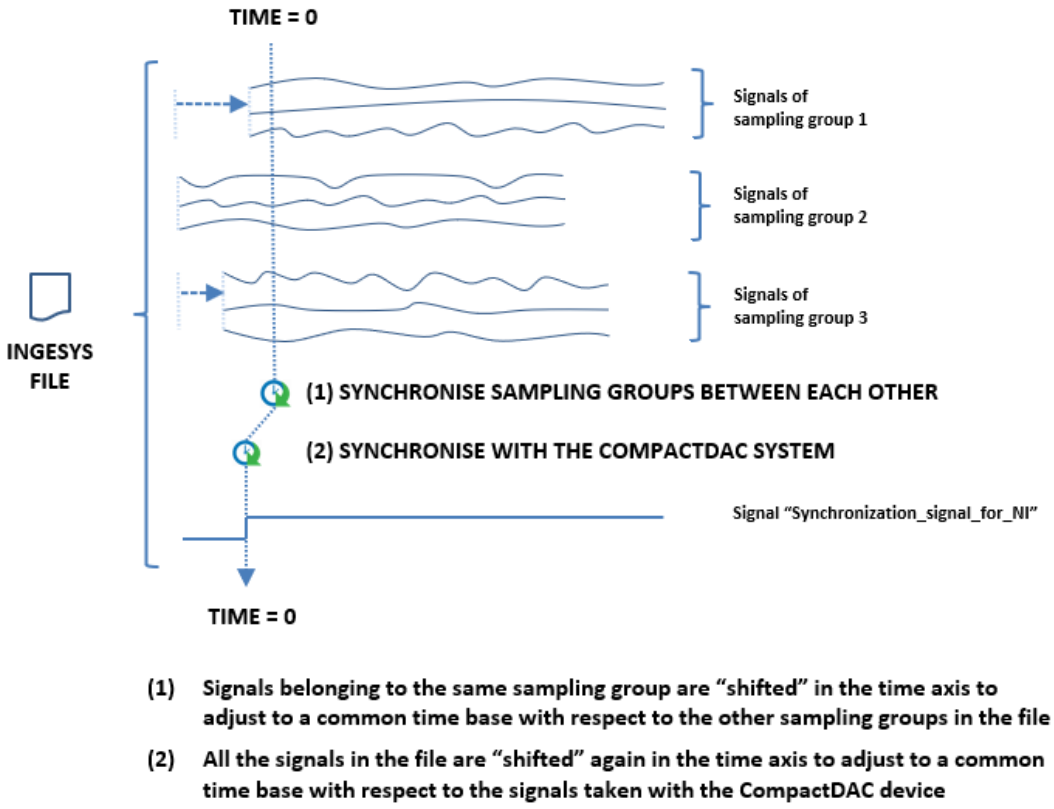


Figure 32: Synchronization process for Ingesys file signals

For a CompactDAC MAT file, the process only takes one step. The point where signal "Synchronization_signal_from_Ingesys" (which can be located in the CompactDAC file) goes from 0 to 5 marks the place where the timestamp must be 0 for every CompactDAC signal in the timestamp vector to be generated:

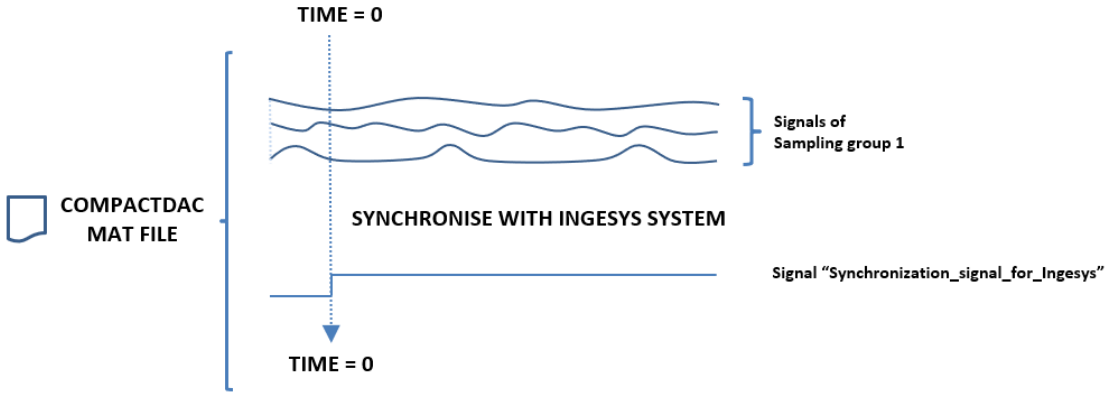


Figure 33: Synchronization process for CompactDAC file signals

5.1.3 Serialisation of Files with Physical Units

Once data is separated by sampling groups, transformed, and timestamp vectors are generated, a file with physical units for each sampling group (i.e., for each R dataframe created) can be serialised in csv files. Here, 3 csv files must be created for each Ingesys MAT file, 1 csv file for each CompactDAC file, and 1 csv file for the metadata.

The filename of each resulting file will have the following pattern:

{ResultId}_{yyyy}_{MM}_{dd}_{hh}_{mm}_{ss}_{ResultType}_c{XXXX}_n{YYYY}.csv

where:

- {ResultId}: it is the result identifier.
- {yyyy}_{MM}_{dd}_{hh}_{mm}_{ss}: for Ingesys and CompactDAC, it is the datetime of the original file. For metadata, it is the datetime of the first result record.
- {ResultType}: it is a string for each of the three possible result types: “Ingesys”, “CompactDAC” or “ContextData” (metadata).
- {XXXX}: number of the EMA cycles at the beginning of the file.
- {YYYY}: number of samples contained in the file.

Once the files are serialised, they are uploaded on the Azure blob storage under a specific location:

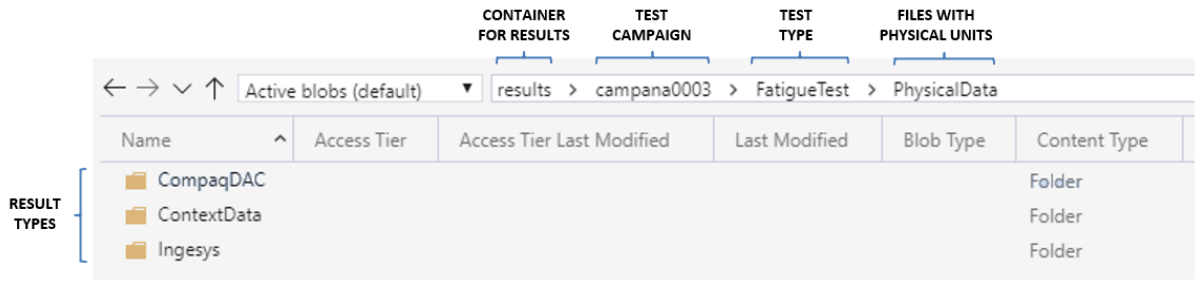


Figure 34: Folders for generated signals with physical units on the Azure blob storage

5.2 Feature Selection and Extraction

The steps that must be taken on this stage are the following:

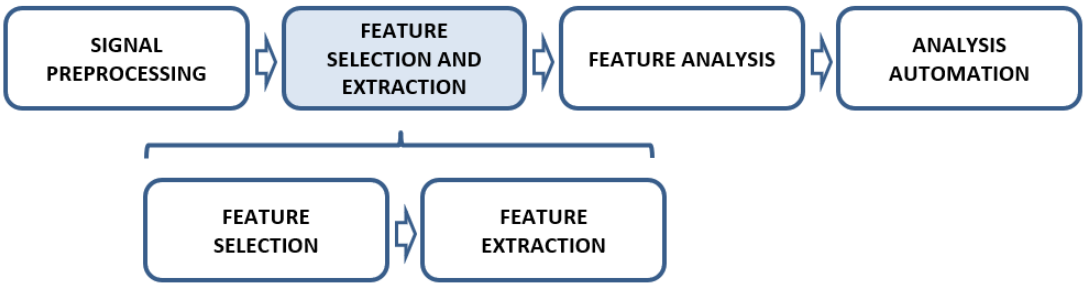


Figure 35: Feature selection and extraction steps

The outcome of this stage is the following:

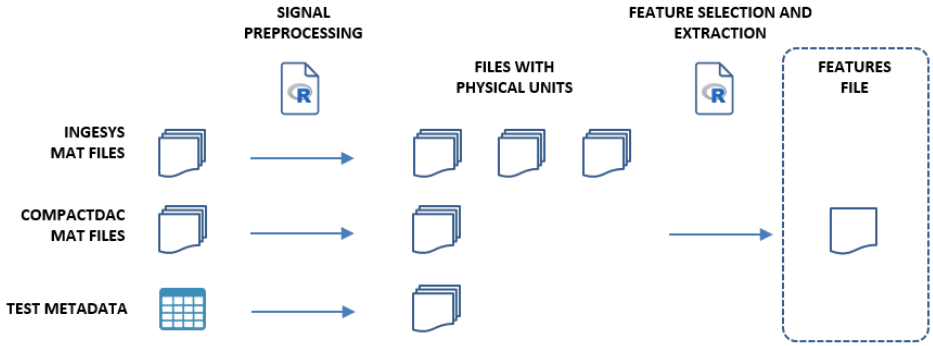


Figure 36: Outcome of the feature selection and extraction phase

Feature generation is done through a different piece of R code (see annex) to decouple this process from data pre-processing. Thus, new features can be developed and tested independently from the same pre-processing outcome.

5.2.1 Feature Selection

Some typical calculations performed in the time domain to obtain signal features are described below (Lei, 2016):

- Mean value, Root Mean Square (RMS), Peak value.

  These statistical values can be good fault indicators as they can be amplified by the presence of a mechanical fault in the EMA, and this change in the amplitude and energy can be proportional to the severity of the fault.

- Kurtosis value, Crest factor, Clearance factor, Impulse factor.

  These calculations could be used for detecting the beginning of a fault, as they are more related to peaks in the signal.

In addition to these statistical values, other aggregations are also obtained. The complete list of features to be calculated is shown in the following table:

Table 1: Feature calculations

FEATURE	EXPRESSION	ABBREVIATION (IN FEATURE NAMES)
Maximum	$X_{max} = \max(x(n))$	Max
Minimum	$X_{min} = \min(x(n))$	Min
Peak value	$X_{pv} = \max\|x(n)\|$	Pv
Peak to peak value	$X_{ppv} = \|X_{max} - X_{min}\|$	Pk2Pk
Root Mean Square (RMS)	$X_{rms} = \sqrt{\frac{\sum_{n=1}^{N}(x(n))^2}{N}}$	Rms
Kurtosis	$X_k = \frac{\sum_{n=1}^{N}(x(n) - X_m)^4}{(N-1)X_{sd}^4}$ where X_{sd} *is the standard deviation:* $X_{sd} = \sqrt{\frac{\sum_{n=1}^{N}(x(n) - X_m)^2}{N-1}}$ and X_m is the mean value: $X_m = \frac{\sum_{n=1}^{N} x(n)}{N}$	Ku
Crest factor	$X_{Crf} = \frac{X_{pv}}{X_{rms}}$	Crf
Clearance factor	$X_{Clf} = \frac{X_{pv}}{\left(\frac{\sum_{n=1}^{N}\sqrt{\|x(n)\|}}{N}\right)^2}$	Clf
Impulse factor	$X_{Imf} = \frac{X_{pv}}{\frac{1}{N}\sum_{n=1}^{N}\|x(n)\|}$	Imf
Shape factor	$X_{Shf} = \frac{X_{rms}}{\frac{1}{N}\sum_{n=1}^{N}\|x(n)\|}$	Shf
Median	$X_{Me} = \begin{cases} \frac{(N+1)^{th}}{2} term\ when\ N\ is\ odd \\ \frac{N^{th}}{2} + \left(\frac{N^{th}}{2} + 1\right) \\ \frac{}{2} term\ when\ N\ is\ even \end{cases}$	Median

The abbreviation strings shown in the table above are used when generating the name of the features, as is presented later in the following section.

As was mentioned when describing the data processing approach, the reason why several features are generated is that they complement to each other from different aspects.

Kurtosis will be calculated using the R “moments” package. The rest of the features will be obtained through R base functions.

5.2.2 Feature Extraction

After examining the results generated during the tests, it has been decided that the signals where features are going to be extracted come exclusively from the Ingesys device. This is because acoustic information generated through the CompactDAC acquisition device has been discarded due to a recurring bug in the software controlling the test bench.

This bug has caused missing information in the database for most of the samplings (therefore, only partial information is available), and it is only until recently that this problem has been fixed.

The signals where features are extracted for the fatigue test are the following:

Table 2: Signals of interest for feature extraction

SIGNAL	DESCRIPTION
Accelerometer_X	Signal from X-Axis accelerometer. Acquired at 10 KHz.
Accelerometer_Y	Signal from Y-Axis accelerometer. Acquired at 10 KHz.
Accelerometer_Z	Signal from Z-Axis accelerometer. Acquired at 10 KHz.
Cylinder_force_filtered	Filtered force applied by the hydraulic cylinder. Acquired at 10 KHz
EMA_Force	Force provided by the motor converter. Acquired at 10 KHz
Motor_current_1	U channel motor current. Acquired at 50 KHz.
Motor_current_2	V channel motor current. Acquired at 50 KHz.
Stober_Speed_Meassurement	Speed measurement from encoder

The strategy for feature extraction is based on signal “EMA_position_set-point”, which is the commanded set-point for the EMA (i.e., the command for the theoretical position of the EMA during the test).

The idea is to identify certain types of segments on this signal, and "project" those segments on the other signals of interest (listed in the previous table) for feature generation. This process of segment projection will be presented in a figure later.

There are two types of segments on “EMA_position_set-point” signals that are going to be identified in this process:

- “Rising-edge to falling-edge” (“RE-FE”) segments: an interval with a starting transition from low to high values, and an ending transition from high to low values.
- “Falling-edge to raising-edge” (“FE-RE”) segments: an interval with a starting transition from high to low values, and an ending transition from low to high values.

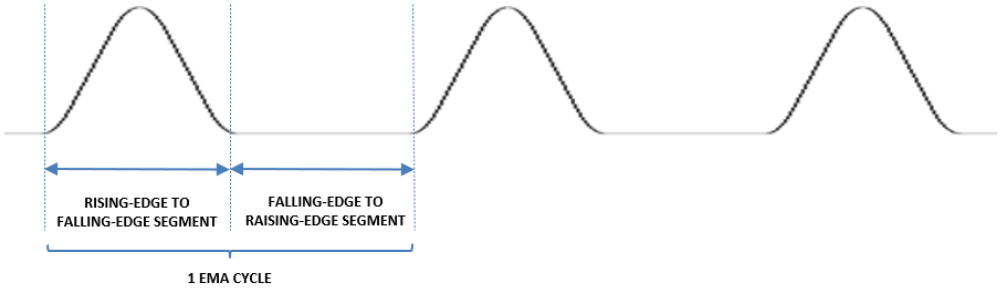


Figure 37: Segment types on an “EMA_position_set-point” signal

As can be seen in the above picture, two adjacent “RE-FE” and “FE-RE” segments make a complete EMA cycle. This cycle is the basic movement unit during the EMA fatigue test, which repeats over and over again, and this movement is controlled by this commanding signal.

The reason for distinguising both kind of segments (“RE-FE” and “FE-RE”) is because their nature is different. RE-FE segments represent transient operations, where the EMA is commanded to move with a certain trajectory and acceleration, while FE-RE segments represent steady operations, i.e., the EMA must remain static in the same position for a while.

Potential issues in the EMA will manifest themselves diffently in both parts of the signals, so in order to be consistent, the same set of features will be calculated separately in both types of segments.

In any case, the goal is the same: to detect deviations from the ideal trajectory in the signals.

There are many causes for these deviations. A typical example is what is known as “jerk”, which is the rate of change in acceleration with time. Jerk is generally caused by inertia due to internal friction or damaged components (like problems in the ball-screw surface, for instance).

The following image shows an example of symptoms related to jerk in a signal:

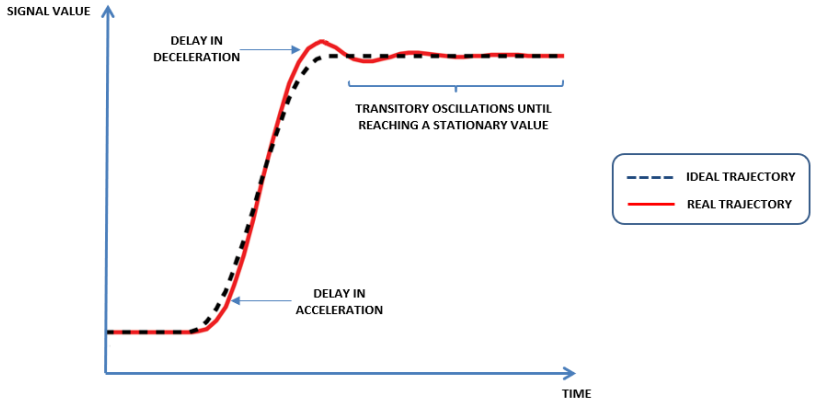


Figure 38: Example of jerk in a signal

Due to some technical issues on the test bench, signal “EMA_position_set-point” do not always create correct movement commands, so it is necessary to distinguish between valid and invalid segments. Only valid segments will be projected on signals for feature generation.

The following figure shows an example of correct and incorrect movement commands in signal “EMA_position_set-point”. The picture also shows the projection of valid segments on signal “Cylinder_force_filtered” (this projection mechanism is similar for all the other signals):

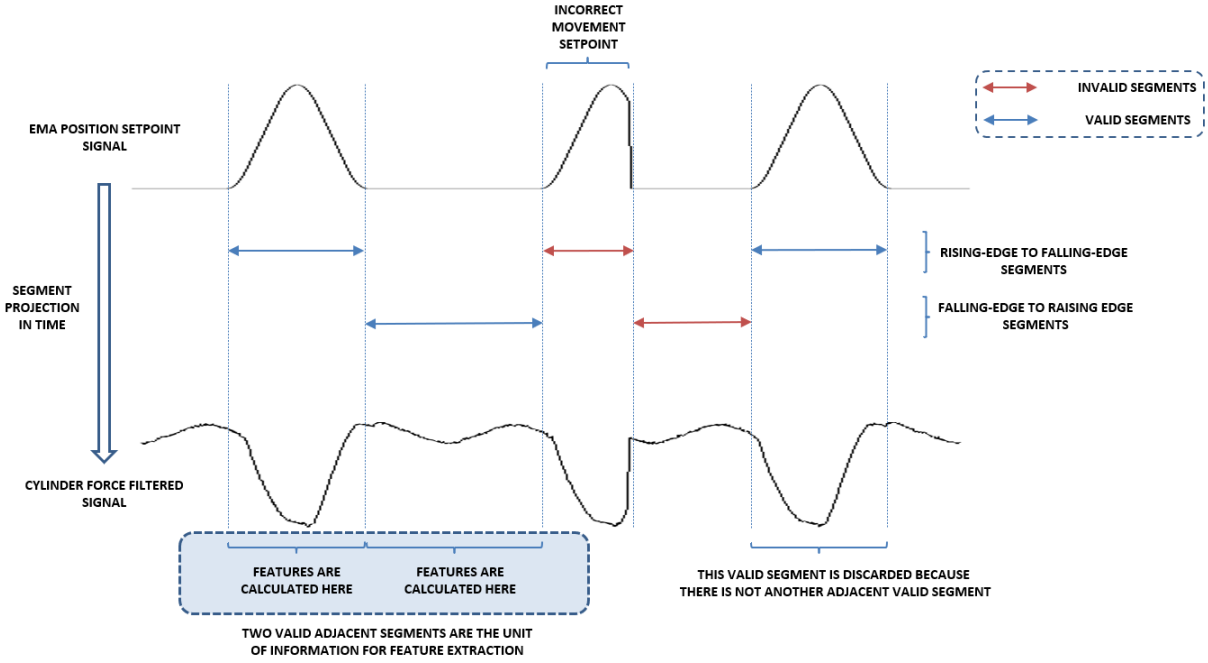


Figure 39: Segment projection on a signal of interest for feature extraction

As can be seen in the previous figure, only features for two valid adjacent RE-FE and FE-RE segments (or viceversa) in a file will be calculated, and the same set of features will be obtained separately.

Although these features will be generated separately for both adjacent segments, they will be placed in the same row within the features file to be generated (in csv format) as a result of the feature extraction step, thus forming a consistent unit of analysis. The R program to be developed during this phase will do this task.

To distinguish between valid and invalid movement commands in signal “EMA_position_set-point”, a simple time length criterion will be applied, with thresholds that have been determined empirically after examining the different measurements:

- RE-FE valid segments > 0.3339 seconds
- FE-RE valid segments > 0.41 seconds

The name of every feature in the features file has the following pattern:

{FeatureType}_{SegmentType}_{SignalName}

where:

- {FeatureType}: it is the abbreviation of the type of feature as indicated in the section devoted to feature selection.
- {SegmentType}: type of segment where the feature is extracted.
    - “RF”: Raising-Edge to Falling-Edge segment.
    - “FR”: Falling-Edge to Raising-Edge segment.
- {SignalName}: name of the signal from which the feature is generated. The names of the signals are indicated in the table at the beginning of this section.

The complete list of feature names (176) can be found in the annex.

The features file also contains metadata information providing context for these features. This metadata is the following:

- “ResultId”: identifier of the test result which is the source of the information.
- “Cycles”: number of cycles of the EMA when the result was obtained. This information comes from the metadata files.
- “Datetime”: absolute timestamp of the file from which the features where extracted.
- “InitialTimestamp”: offset with respect to the value “Datetime” corresponding to the initial point of the two segments from which the features were generated.

- "FinalTimestamp": offset with respect to the value "Datetime" corresponding to the final point of the two segments from which the features were generated.

The structure of the features file is the following (4 columns for metadata, 176 columns for features):

ResultId	Cycles	Datetime	InitialTimestamp	FinalTimestamp	Feature1	Feature2	(...)	FeatureN
v11	v12	v13	v14	v15	v16	v17		v1M
v21	v22	v23	v24	v25	v26	v27		v2M
v31	v32	v33	v34	v35	v36	v37		v3M
v41	v42	v43	v44	v45	v46	v47		v4M
v51	v52	v53	v54	v55	v56	v57		v5M
v61	v62	v63	v64	v65	v66	v67		v6M
v71	v72	v73	v74	v75	v76	v77		v7M
(...)	(...)	(...)	(...)	(...)	(...)	(...)		v8M
METADATA					FEATURES			

Figure 40: Structure of the features file

Once the features file is created, it will be uploaded on Azure by the R program under a specific location:

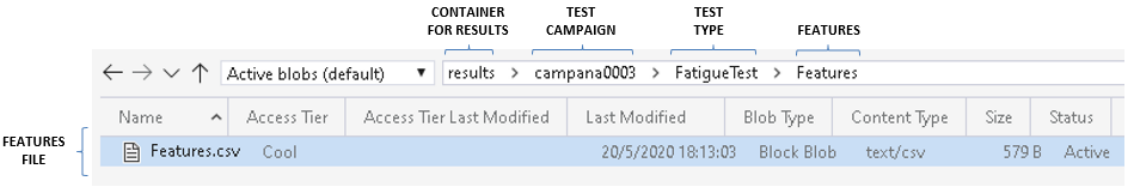


Figure 41: Generated features file on the Azure blob storage

5.3 Feature Analysis

Here are the steps for the analysis stage:

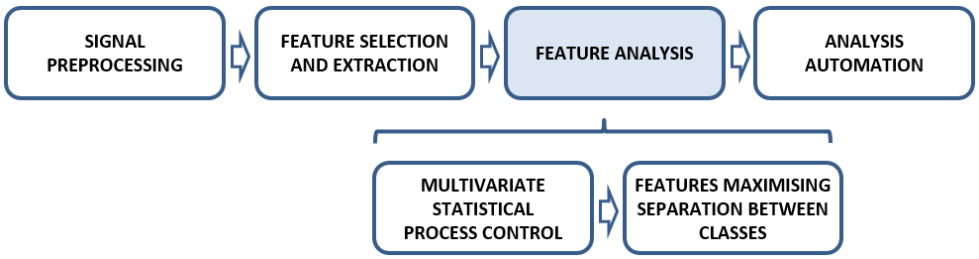


Figure 42: Feature analysis steps

The output of this phase is the following:

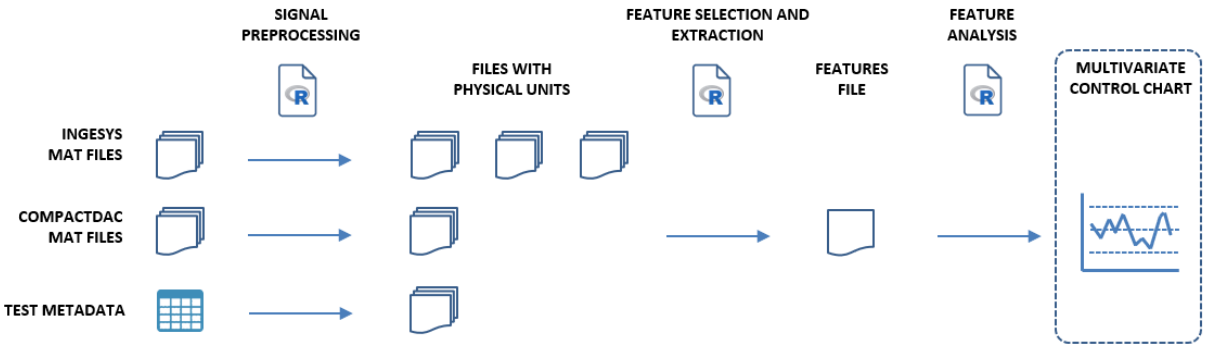


Figure 43: Outcome of the feature analysis phase

Feature analysis is performed through a specific R program that can be seen in the annex.

5.3.1 Multivariate Statistical Process Control

Given that several measures are avalailable for each sample, the approach taken to evaluate the EMA degradation involves the use of a Statistical Process Control (SPC) method where several features are monitored simultaneously to be inside some control limits.

In this scenario, the use of a multivariate technique is preferred over univariate methods (where variables are monitored individually) as we want to account for correlation between measures.

Therefore, a multivariate control chart will be developed with the combination of the different features generated in the previous step.

Among the different multivariate techniques, the Hotelling's control chart is a classic method, allowing observations to be plotted as a single statistical value on a chart.

The application of this technique involves two phases:

- Phase I: a control chart is used to ensure that the "nominal" process is in control. For this purpose, a reference (training) dataset must be created for obtaining a single-dimensional statistical series to be checked against some calculated upper and lower control limits.
- Phase-II: a control chart is used for checking whether the rest of the process is in statistical control. Here, a test dataset is used, from which the single-dimensional statistical series is generated, using control limits created with the reference dataset.

Basically, the goal of these two phases is to ensure that the previous and future process, respectively, are in control.

In Hotelling's control charts, data can be used in subgroups or individual observations, and calculations are different in both cases. In this scenario, data are individual observations (i.e, size of feature samples is 1), and the statistical Q value equation in this case is the following (NIST, 2020):

$$Q_j = (x - \bar{x})' S^{-1} (x - \bar{x})$$

where:

- Q_j: it is the statistical value to be calculated
- $\bar{x}$: it is the sample mean vector of the observations
- S^{-1}: it is the covariance matrix of the observations
- $(x - \bar{x})'$: it is the transpose of the expression $(x - \bar{x})$

In the case of individual observations, the upper control limit (UCL) and lower control limit (LCL) for the reference dataset in phase one is the following (NIST, 2020):

$$UCL = \frac{(m-1)^2}{m} B_{1-\alpha/2;p/2,(m-p-1)/2}$$

$$LCL = \frac{(m-1)^2}{m} B_{\alpha/2;p/2,(m-p-1)/2}$$

where:

- m: it is the number of preliminary samples
- B: it is the percent point function of the Beta distribution (corresponding to R-function “qbeta”)
- α: it is the significance level (typically set to 0.05 or 0.01)

On the other hand, the Q_j values for the test dataset in phase two are calculated with the same expression shown previously, but the terms $\bar{x}$ and S^{-1} are computed with the reference (historical) dataset.

Regarding the control limits for the test dataset in phase two, the equations to be used are the following (NIST, 2020):

$$UCL = \frac{p(m+1)(m-1)}{m^2 - mp} F_{1-\alpha/2;p,m-p}$$

$$LCL = \frac{p(m+1)(m-1)}{m^2 - mp} F_{\alpha/2;p,m-p}$$

where:

- p: it is the number of variables (type of features)
- F: it is the expression corresponding to the F-test (R-function “qf”)

Once the theory has been introduced, the following sub-sections explain the details of the implementation of the multivariate control chart:

- Feature dimensionality reduction.
- Phase I of Hotelling’s Multivariate Control Chart.
- Phase II of Hotelling’s Multivariate Control Chart.

Feature Dimensionality Reduction

The first step to generate the chart is to split the data into two datasets:

- Reference dataset: it is a subset of the features where the EMA is estimated to be in control. This "in-control" status will be validated in phase one of the Hotelling's control chart methodology.
- Test dataset: it is the remaining part of the features. This is the dataset that is going to be verified for a potential out-of-control situation.

Once data is split into a training and test dataset, a feature dimensionality reduction will be perform to later facilitate the analytical task. This will also minimise potential computation overflows in calculations related to the Hotelling's control chart, and decrease the influence of irrelevant features.

Principal component analysis (PCA) will be used to perform this task, and the process will be the following:

- First, PCA will be applied on the reference dataset, and principal components (PCs) will be calculated. All features have to be standarised, as PCA is quite sensitive regarding the variance of data.
- After examining the results, a decision will be made about the number of PCs to be reduced, while preserving as much information as possible.
- Lastly, the outcome of PCA on the reference dataset will be projected on the test dataset. The same number of PCs will be obtained, and data will be standarised at the same time.

Phase I of Hotelling's Multivariate Control Chart

Next, phase one of Hotelling's control chart methodology will be implemented. The reference data will be used here, and the actions to be done are the following:

- Calculation of the Q statistical values, together with the upper and lower control limits of the chart for the reference dataset. In order to do this, the equations presented previously will be applied.
- Removal of data outside the control limits.
- Recalculation of Q and reference control limits with the remaining data.
- Verify that the process is in control.
- Calculation of the control limits that will be used for the test dataset (here, the reference data will be used too).

Phase II of Hotelling's Multivariate Control Chart

Then, it is the turn of phase two of Hotelling's control chart methodology, and the goal here is to verify whether the test dataset has reached an out-of-control status.

The process is as follows:

- Calculation of the Q statistical values for the test data. The control limits for these values were obtained in the previous step.
- Verification of the Q values against the limits to evaluate if the process is in control.

5.3.2 Features Maximising the Separation between Classes

Once the control chart is created, the Linear Discriminant Analysis (LDA) algorithm will be used to obtain those features that maximise the separation between the two categories representing the status of the EMA.

Linear discriminant analysis (LDA) is a supervised machine learning technique that uses linear combinations of variables for the separation of two or more categories. Therefore, it is typically used for classification problems, and also for dimensionality reduction.

In this scenario, there are only two categories ("non-degraded" and "degraded"), so only one linear combination of variables (one "linear discriminant") is generated ("LD1").

Given that this is a supervised method, a training dataset has to be generated with an extra categorical variable representing the EMA status. Thus, some of the samples in the dataset must be marked with a “non-degraded” label, and the rest with a “degraded” mark.

The samples with the “non-degraded” label will be taken from the reference dataset created during the phase I of the Hotelling’s method. On the other hand, the samples with the “degraded” status will be obtained starting from the first point in the chart where the process is considered out of control, and the number of elements will be the same as in the case of the “non-degraded” label.

Before using this dataset as the input to train the LDA model, a standarization of the features must be done, and then the algorithm is run.

The output model will contain the single linear discriminant (LD1), and those features having the largest coefficients (in absolute value) will provide clues for explaining the change in the EMA status.

5.4 Analysis Automation

The last stage in the analysis process has the following steps:

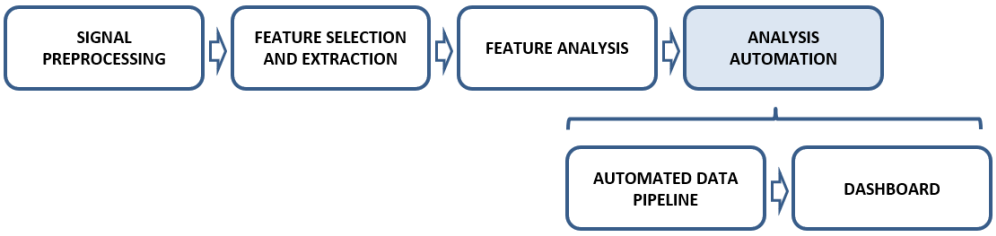


Figure 44: Analysis automation steps

The result of this step is a dashboard containing the multivariate control chart, which is updated periodically through an infrastructure created on the cloud:

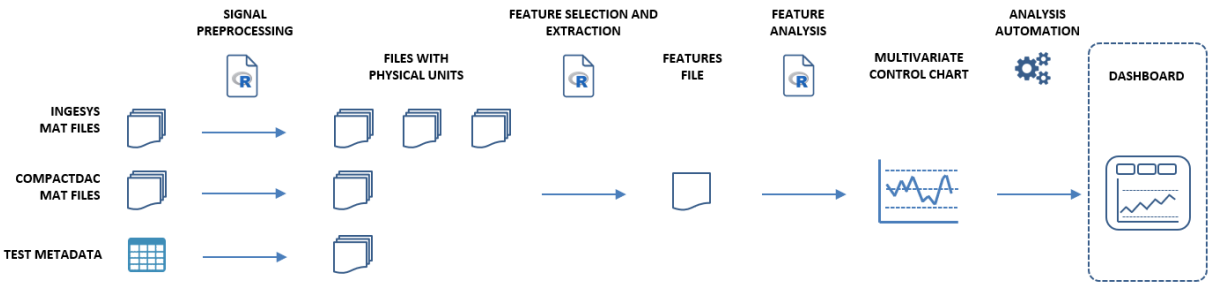


Figure 45: Outcome of the analysis automation phase

5.4.1 Automated Data Pipeline

The goal of this step is the periodic execution of the R code developed in the previous phases (preprocessing, feature selection and extraction, and analysis), so that features coming from new tests are appended to the features file located on the Azure repository.

This mechanism will be created on the cloud environment to take advantage of the database and file repository that are updated there on a daily basis. Also, for a better efficiency, the R code will be slightly modified so that only new data is processed on each run.

5.4.2 Dashboard

A Power BI dashboard hosted on the cloud is the final destination of the automated data processing pipeline. The idea is to provide a visualization tool that allows the monitoring and evaluation of the EMA degradation.

6 Description of the Contribution

6.1 Analysis Results

Before explaining the analytical process in depth, some remarks have to be made to contextualise the testing scenario.

At the time of writing this document, the fatigue test has not yet been fully completed due to different problems in the test bench. Some of these problems have been the following:

- Electrical noise from nearby machines affecting the test bench operation.
- Cooling problems in the hydraulic system.
- Updates in the software controlling the testbed due to bug fixes and changes in the behaviour.

All these problems have led to an irregular temporal distribution of samples due to interruptions in the experiments.

Even if the fatigue test has not yet been completed, the analysis can still be performed, not only to check if there is some trend in the data during the tests, but because there has been a major failure in the test bench that caused the experiments to stop for a long period of time until some components were replaced.

Therefore, even if it was the test rig itself that had the problem and not the EMA, the implemented method can be tested to see if it is effective.

As different components were introduced in the test bench once it was fixed, the dataset has been divided into two parts, one with samples (features) until the failure, and the other one containing samples obtained with the new components.

Few days after the new components were introduced in the system, the test bench started to experience problems again. After some time with these problems, it was decided to stop the experiments to proceed with further investigations. When this document is being written, the test bench is still stopped.

Thus, two different analyses have been performed, and the following sections describe their results.

6.1.1 Anaysis until the Test Bench Failure

One important issue to consider when conducting the analysis has been the potential influence of the numerous stops and starts in the test bench as a result of the different problems experienced during testing.

After a long-lasting stop in the bench, even if there is a short period of warming-up and verification before resuming the experiments, the first samples might be affected by a transitory state due to mechanical inertia, temperature differences, etc., until the process can be considered stable and uniform.

Thus, the initial samples taken during the next hour after each one of these stops (i.e., when the bench was idle for more than 1 hour) have been removed from the dataset.

Once this removal is done, the data is divided into a reference and a test dataset. The reference dataset (where the actuator is estimated to be in a non-degraded status) is created with a 10% of the initial features, and the rest is assigned to the test dataset.

Next, a principal component analysis (PCA) is implemented on the reference dataset to obtain a dimensionality reduction that will facilitate the computations for the Hotelling's multivariate control chart, otherwise there would be a data overflown in these calculations.

The following chart shows the variance explained by the the different principal components (PCs) generated, ordered from highest to lowest:

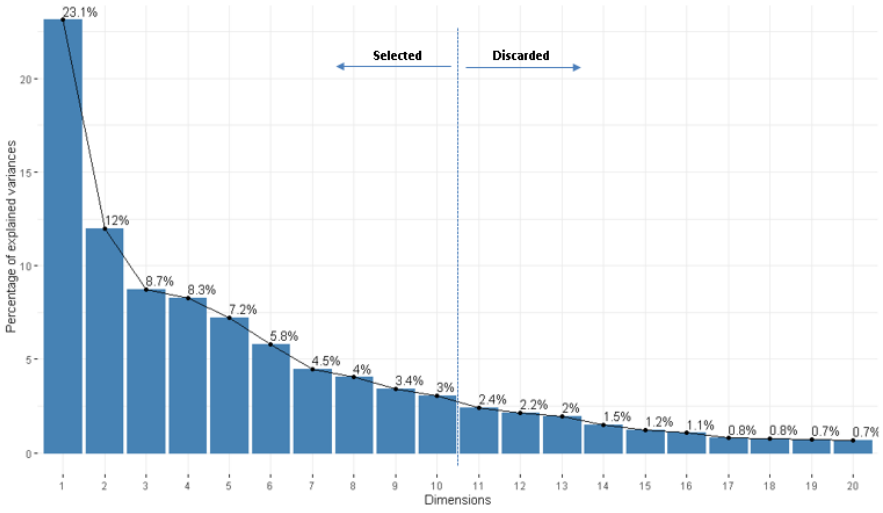


Figure 46: Principal components in the reference dataset during the first analysis

Once the number of selected PCs is established (10), the PCs for the testing dataset are obtained and filtered too.

The next step is to apply the Hotelling's procedure. The equations have been implemented in R taking into account the contents of NIST (2020).

The following charts show the Q values calculated for the reference dataset corresponding to phase I of Hotelling's methodology. Two charts are presented, one of them showing the Q values by sample index, and the other showing Q by date and time. Previously, an outlier removal has been made, as was stated previously in the section describing the multivariate statistical process control:

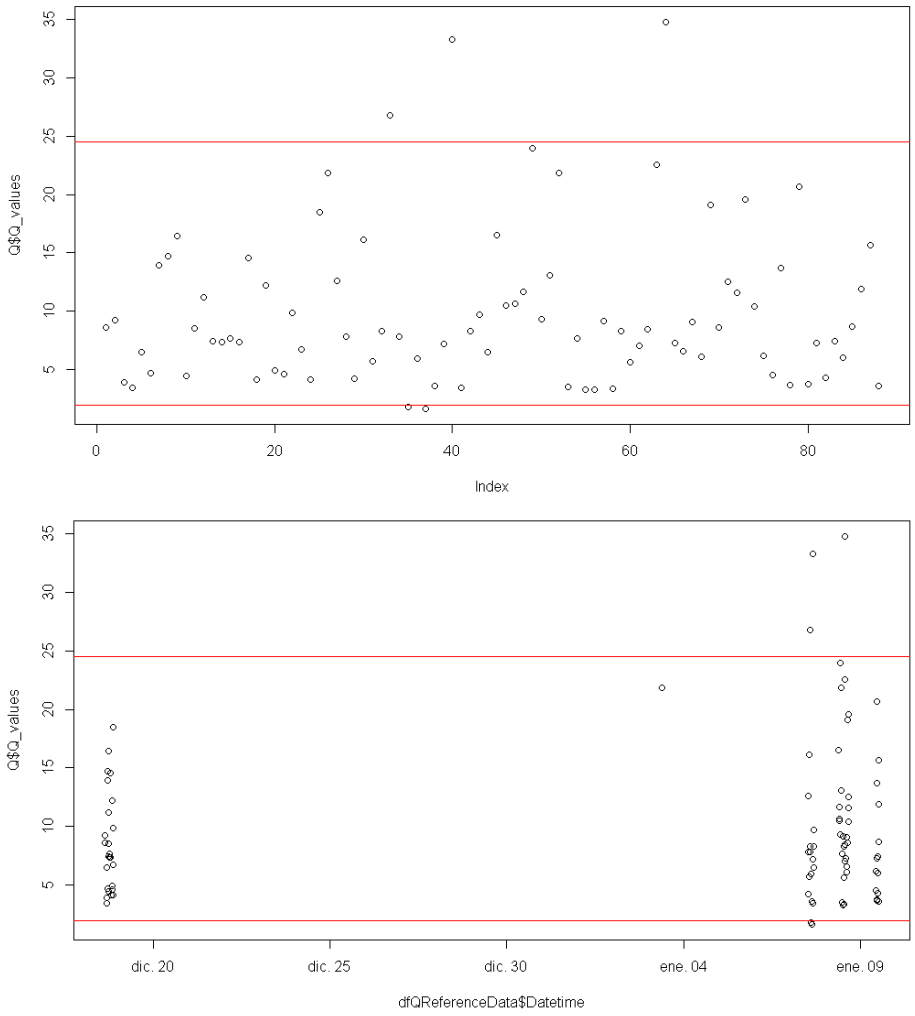


Figure 47: Q values for the reference dataset during the first analysis

As can be seen in the first chart, there are two points barely below the lower control limit, and three values located above the upper control limit. As there are few values, and they are not located next to each other on the x axis, it can be concluded that the process in the reference dataset is in control.

To check if the removal of data after long-lasting stops in the bench had any influence in this result, the same charts are generated with this information included:

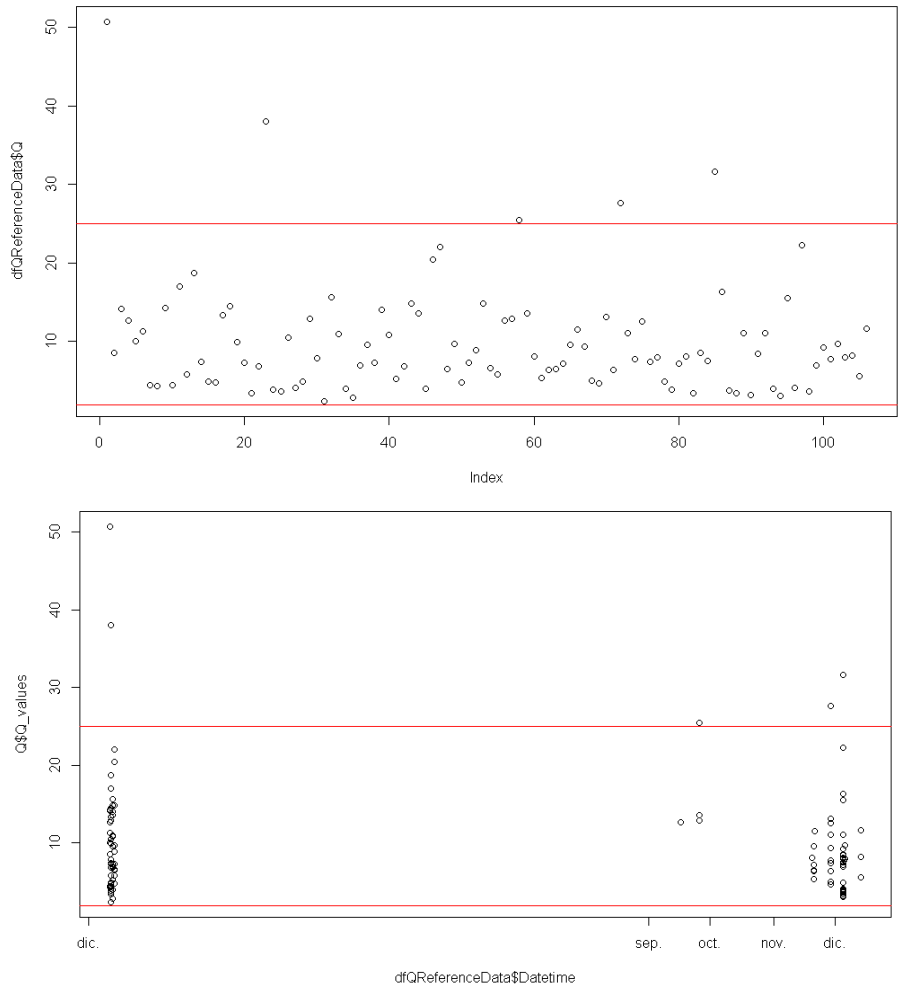


Figure 48: Q values for the reference dataset with bench stops during the first analysis

As can be seen, there are more points above the upper limit, particularly at the beginning, and no point below the lower limit, but the process seems to be in control. In any case, the rest of the analysis has been done removing the 1-hour window of data after long-term stops.

Next, the Q statistical values are calculated for the test dataset during the Hotelling's phase II, and merged with the previous values obtained for the reference dataset. The idea is to represent all the information in a single chart, using the control limits calculated for the test dataset.

Before presenting the chart, outliers are removed, i.e., points more than 3 times above and below the interquartile range. Same as before, two charts are presented, one of them showing Q by sample index, and the other showing Q by date and time:

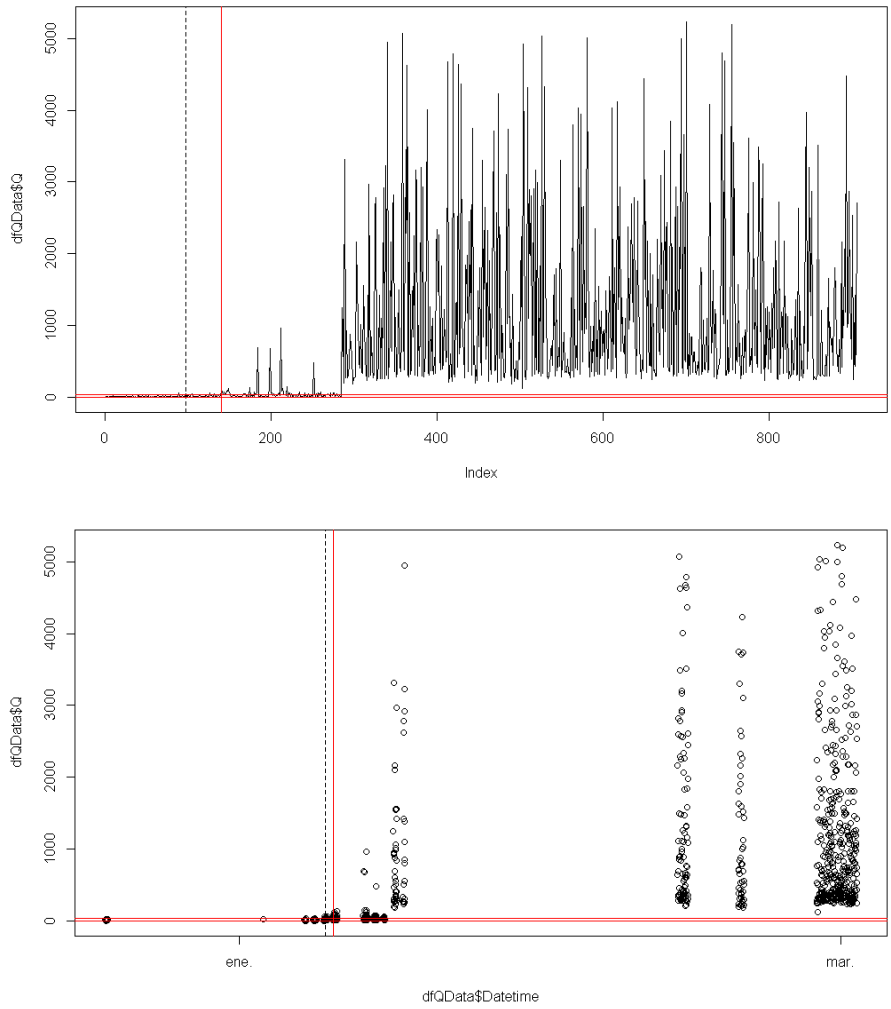


Figure 49: Q values for the reference and test datasets during the first analysis

In the above charts, the upper and lower control limits are represented very close to each other (red horizontal lines) due to the high values shown on the y axis. The dotted vertical line represents the border between the reference and the test datasets.

The red vertical line represents the first point in the series where a critical status condition has been reached, namely, the first of five consecutive points above the upper control limit.

Around this period of time (the time corresponding to this first alert), it was observed on the bench that the EMA electrical motor was having difficulty in coping with the force exerted by the hydraulic system. This was noticed because alarms were raised on the software controlling the test bench.

This condition got worse over time, and at some point the hydraulic system's force setpoint had to be decreased on the bench configuration to continue with the experiments.

From the first critical point onwards, as can be seen on the first chart, the Q parameter reaches some peaks until it finally goes clearly above the upper limit. After that, the process is clearly out of control until the testbed finally broke down.

The following chart shows a zoom on the area between the start of the test dataset and the point where the value of Q goes out of control:

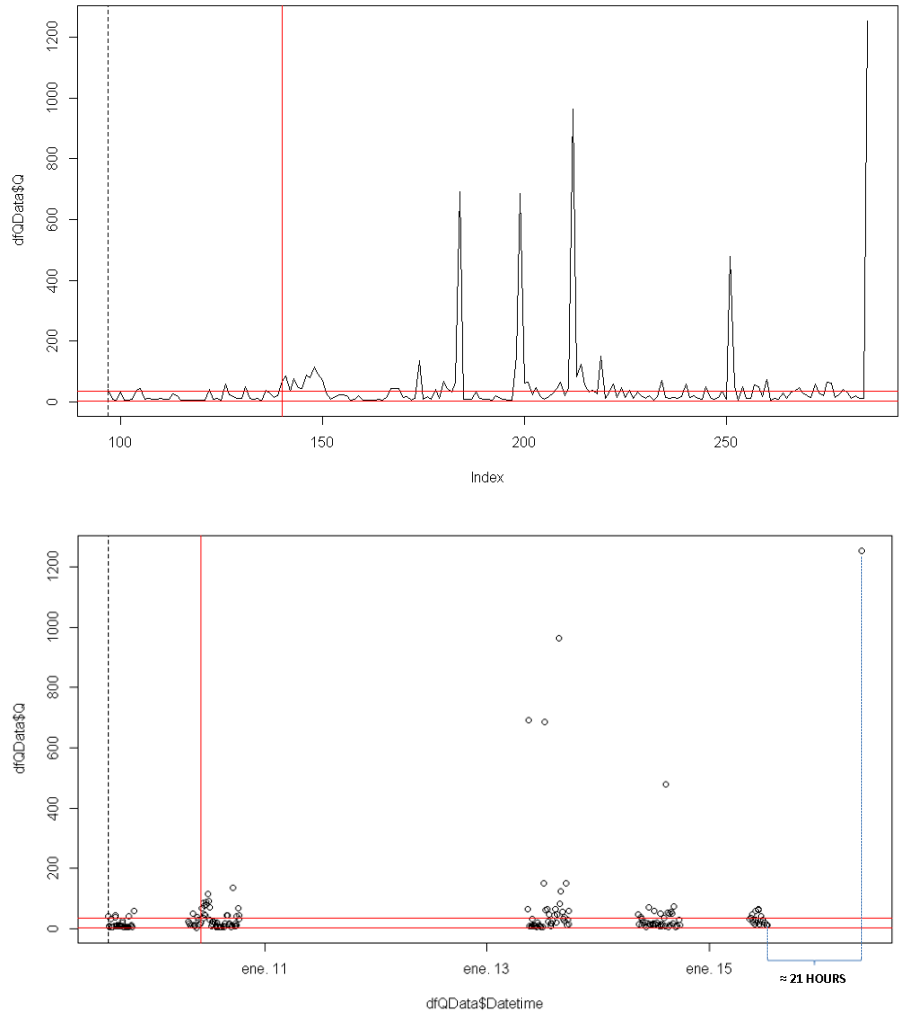


Figure 50: Partial view of Q values for the reference dataset during the first analysis

As can be seen on the second chart, there is a time gap of almost 21 hours between the last point, which is located well beyond the upper control limit, and the previous one. This sharp change in the shape of the chart may suggest that some kind modification or adjustment was made in the test rig during this stop (it has not been identified), which might cause the Q value condition to get worse.

To complete the analysis, the LDA algoritm is run on the data to obtain the linear combination of the features which best separates the two classes representing the “non-degraded” and “degraded” status.

The whole reference dataset will be used for the “non-degraded” label (96 samples). The same number of elements will be taken from the testing dataset for the “degraded” label, starting from the first critical point onwards.

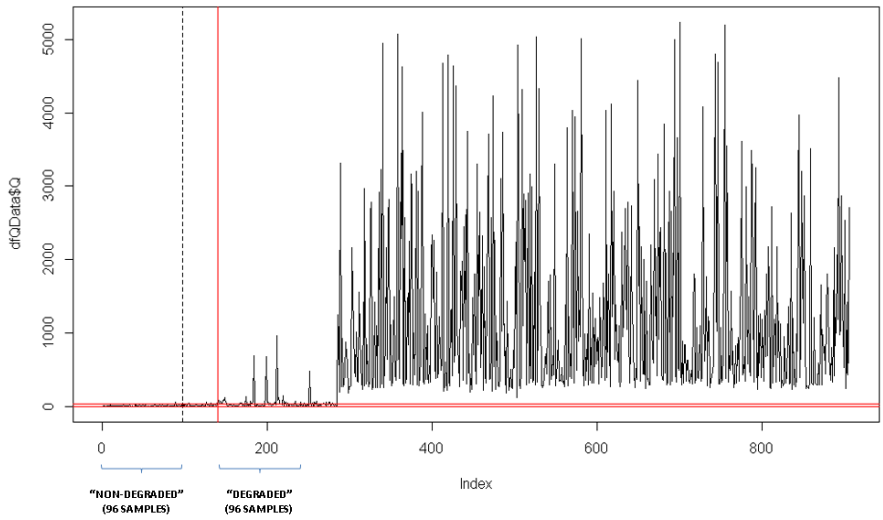


Figure 51: Dataset selection for the LDA algorithm during the first analysis

Once the LDA analsys is completed, a cross-validation is performed to check the accuracy of the LDA model. The resulting confusion matrix is the following:

Table 3: Confusion matrix for the first LDA analysis

	degraded	non-degraded
degraded	75	14
non-degraded	23	83

The accuracy of the model is 80,2%. This results could be acceptable, and it means that the LDA algorithm can distinguish both categories.

Next, the features which best explain the separation between the non-degraded and the degraded status are obtained. To do this, the higher absolute value coefficients of linear discriminants for the LDA analysis are selected.

Here is the list with the 10 first values and their corresponding features:

Table 4: Most relevant features in the change of status for LDA analysis 1

FEATURE	LINEAR DISCRIMINANT COEFFICIENT
Imf_RF_Cylinder_force_filtered	833,2042099
Rms_RF_Cylinder_force_filtered	496,8922696
Pv_RF_Cylinder_force_filtered	459,4645094
Max_RF_Cylinder_force_filtered	459,4645094
Max_FR_Cylinder_force_filtered	432,9189787
Pv_FR_Cylinder_force_filtered	432,9189787
Clf_RF_Cylinder_force_filtered	418,2629639
Rms_FR_Cylinder_force_filtered	417,1285701
Crf_RF_Cylinder_force_filtered	384,4684808
Crf_FR_Cylinder_force_filtered	320,8108582

As can be seen on the table, all features come from signal “Cylinder_force_filtered”, which is the force applied by the hydraulic cylinder (which is later filtered to remove the noise).

The relevancy of this signal between the non-degraded and degraded status is consistent with the fact that the test bench was experiencing a problem, which led to the final breakdown of the system.

On the other hand, the feature with the highest relevance (well above the others) has been calculated with the impulse factor, which is good at detecting the beginning of faults (as was explained in the feature selection section).

This is coherent with the Q values going up and down the upper control limit repeatedly in the segment used for the “degraded” label during the LDA analysis (as can be observed in the last figure), before going completely out of control later in the series.

This is the representation of feature “Imf_RF_Cylinder_force_filtered”, taking into account the non-degraded and degraded segments used for the LDA analysis (the vertical dotted line separates both parts of the signal):

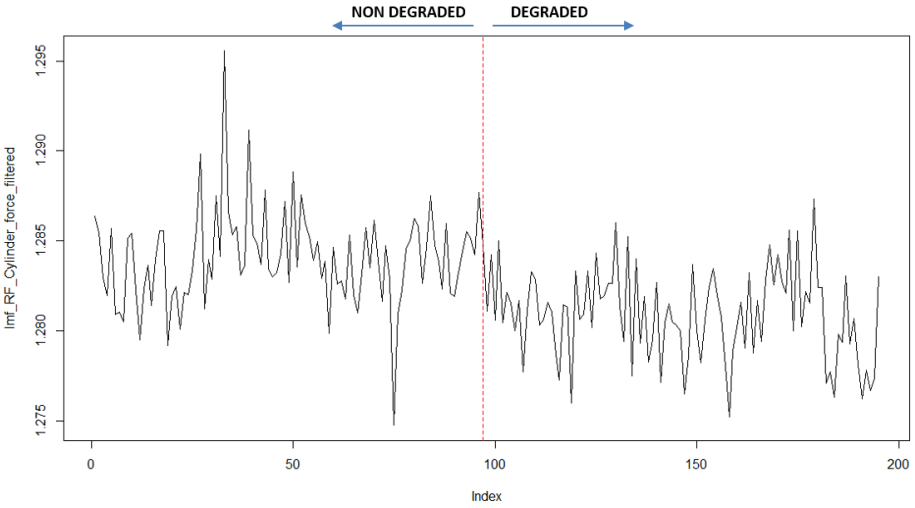


Figure 52: Evolution of feature “Imf_RF_Cylinder_force_filtered” in the first analysis

To have a better view of the distribution of the values, this is the density plot of feature “Imf_RF_Cylinder_force_filtered” for both categories (the dotted lines represents the mean of the distributions):

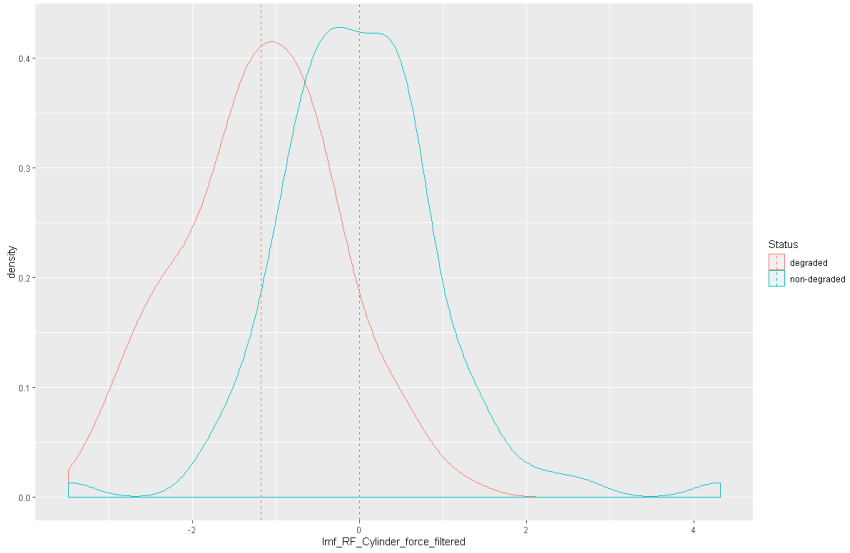


Figure 53: Density plot for feature “Imf_RF_Cylinder_force_filtered” in the first analysis

As can be seen, both series follow a normal distribution, and the degraded subset has "shifted" with respect to the degraded part. This means that the LDA algorithm has a chance to distinguish both categories.

6.1.2 Anaysis after the Test Bench Failure

Once the new components were introduce on the bench, a different analysis is made repeating the same steps explained in the last section.

The PCs calculated for the reference dataset can be seen in the following picture:

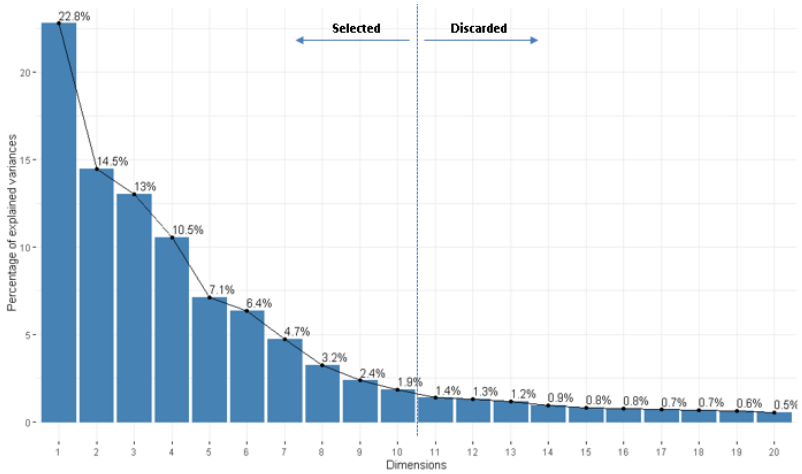
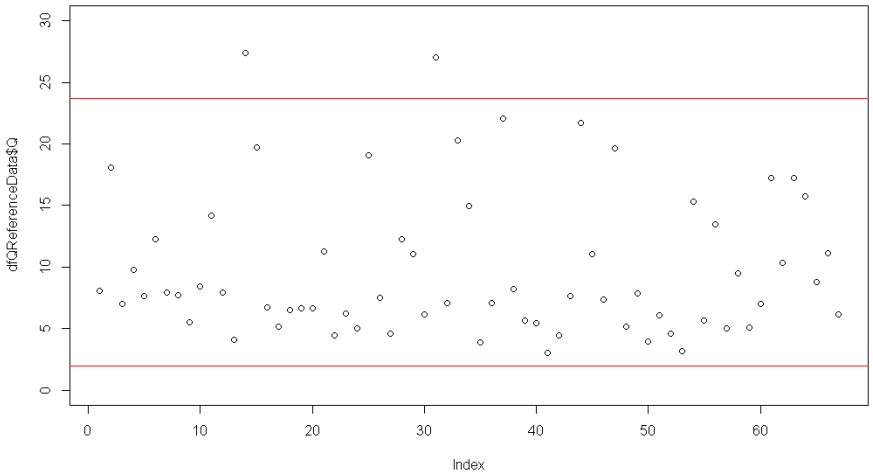


Figure 54: Principal components in the reference dataset during the second analysis

Same as before, 10 PCs are selected, and the PCs for the testing dataset are also obtained.

These are the Q values calculated for the reference dataset during phase I of Hotelling's procedure (both by sample index and by time):



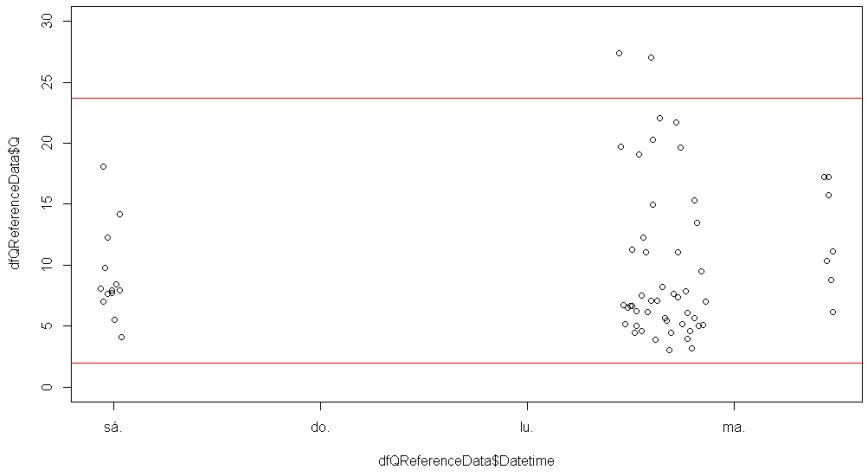


Figure 55: Q values for the reference dataset during the second analysis

Having a look at the figures above, only two non-contiguous points are above the upper limit so the process can be considered to be in control.

The charts showing the Q values for the reference and test database are as follows:

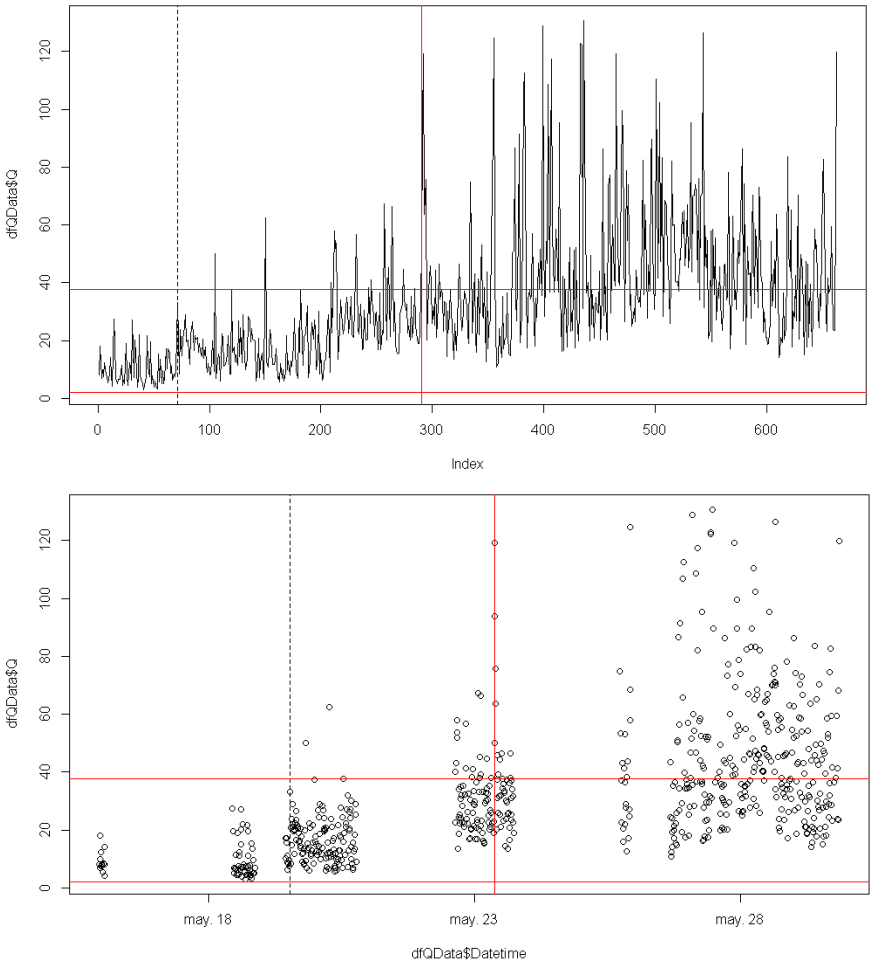


Figure 56: Q values for the reference and test datasets during the second analysis

As can be seen on the pictures, the Q parameter increases its value until a first critical point is reached. This is consistent with what has been observed in the bench, because the EMA has been experiencing problems again with the force applied by the hydraulic system.

After dealing for a while with this problem, it has been decided to stop the experiments to investigate if the EMA has something to do with this or not.

Next, in the same way as in the first analysis, the LDA algorithm is executed. All rows from the reference dataset are used for the “non-degraded” label (70 samples), and the same number of rows are taken from the testing dataset for the “degraded” status, starting from the first alert point onwards.

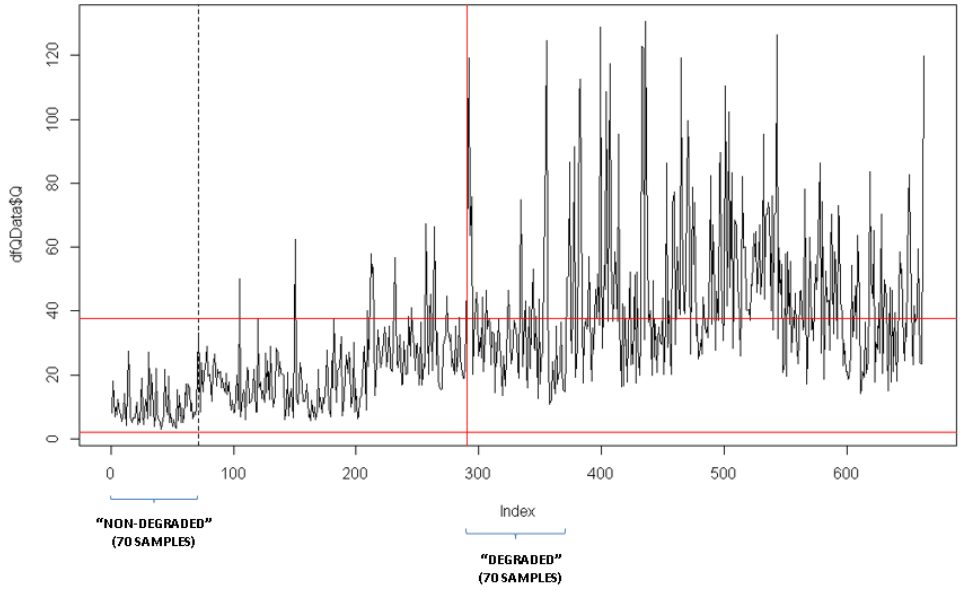


Figure 57: Dataset selection for the LDA algorithm during the second analysis

The confusión matrix resulting from the cross-validation for the calculated LDA model is the following:

Table 5: Confusion matrix for the second LDA analysis

	degraded	non-degraded
degraded	33	35
non-degraded	33	30

Here, the accuracy of the model is 48,09%. This results is pretty bad, meaning that the LDA algorithm is not working in this scenario.

The most relevant feature obtained from the LD1 linear discriminant is “Clf_FR_Accelerometer_Z”, whose density plot is the following:

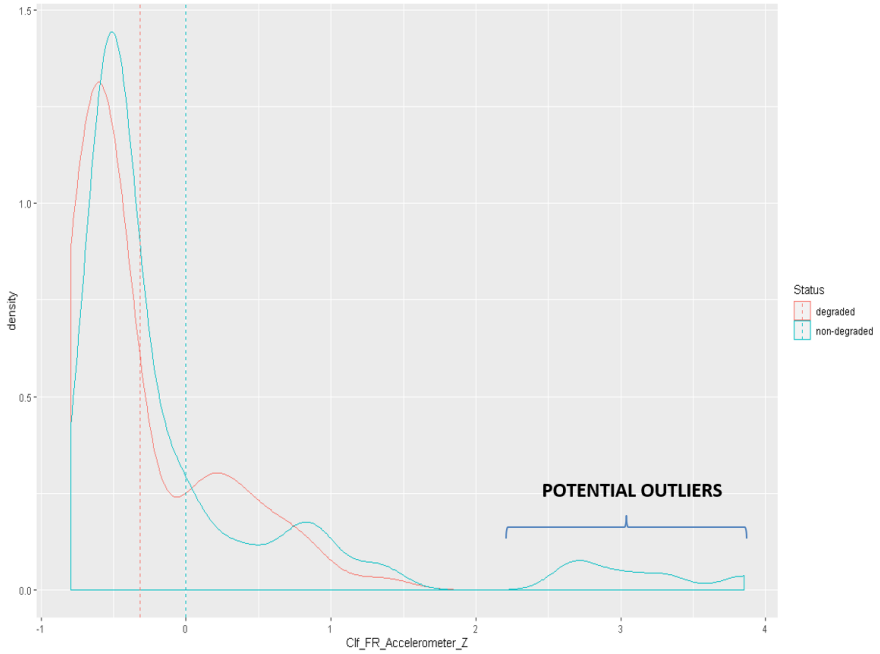


Figure 58: Density plot for feature “Clf_RF_Cylinder_force_filtered” in the second analysis

The values shown in the picture above do not follow a normal distribution, and both sets have a high degree of overlapping. Interestingly, the non-degraded series shows a subset of values to the right of the chart that are not present in the degraded part.

This might mean that these values could be outliers, which were not removed during the phase one of the Hotelling’s procedure. If this is the case, they could be affecting the LDA analysis.

After removing this set of values from all the features in the dataset, and executing the LDA algorithm again, the resulting most relevant feature is “Imf_RF_Accelerometer_Z”. The density plot for this feature is the following:

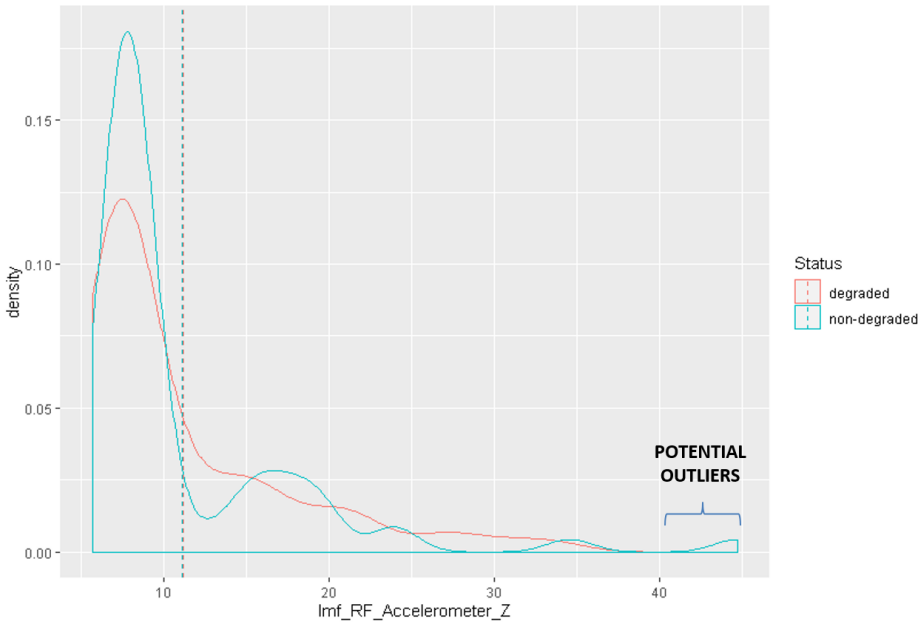


Figure 59: Density plot for feature "Imf_RF_Accelerometer_Z" in the second analysis

Again, there is a subset of values in the non-degraded series that are not present in the degraded part (to the right of the above chart). These values are removed again from the dataset, and the LDA algorithm is recalculated. After this, the most relevant feature is "Shf_FR_Motor_current_1", whose density plot is the following:

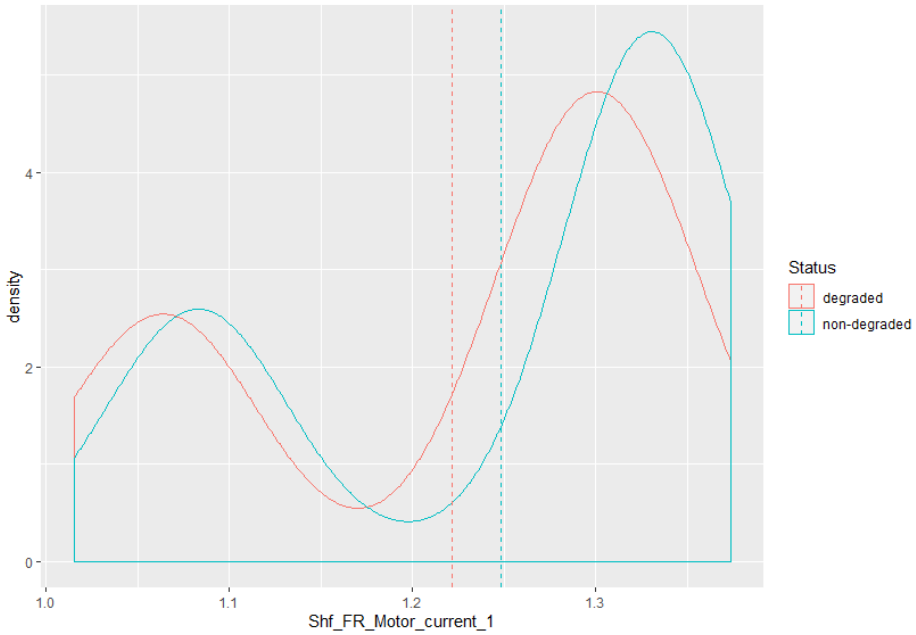


Figure 60: Density plot for feature "Shf_FR_Motor_current_1" in the second analysis

From the figure shown above, it is clear that the LDA method cannot distinguish between the two categories, and therefore the origin of this problem cannot be explained by this algorithm.

As an attempt to explain this second analysis, the possible presence of outliers in the non-degraded part of the dataset could suggest that the system might already be somehow degraded after the new components were replaced. This could be confirmed by the fact that the first critical point in the Hotelling's chart was reached few days after the new components were replaced, when the bench started the tests again.

Even if the origin of this problem cannot be explained by the LDA analysis, the multivariate chart detected that something wrong was happening in the system, and this was confirmed by the alarms raised in the automation environment.

In any case, the experiments have been stopped and the EMA is being revised to see if it is damaged.

For the shake of completeness, the list with the 10 first features with the most relevancy in the change between the non-degraded and degraded status is provided in the following table:

Table 6: Most relevant features in the change of status for LDA analysis 2

FEATURE	LINEAR DISCRIMINANT COEFFICIENT
Shf_FR_Motor_current_1	324,90675
Imf_RF_Accelerometer_Z	299,74202
Clf_FR_Motor_current_2	291,90643
Crf_FR_Accelerometer_X	262,67104
Crf_RF_Accelerometer_Z	235,01531
Crf_FR_Stober_Speed_Meassurement	229,54174
Imf_FR_Stober_Speed_Meassurement	207,44236
Imf_FR_Motor_current_2	191,65877
Pv_RF_Accelerometer_Y	172,38230
Clf_FR_Motor_current_1	169,84171

Also, the representation of feature “Shf_FR_Motor_current_1” is provided, with the non-degraded and degraded segments used for the LDA analysis:

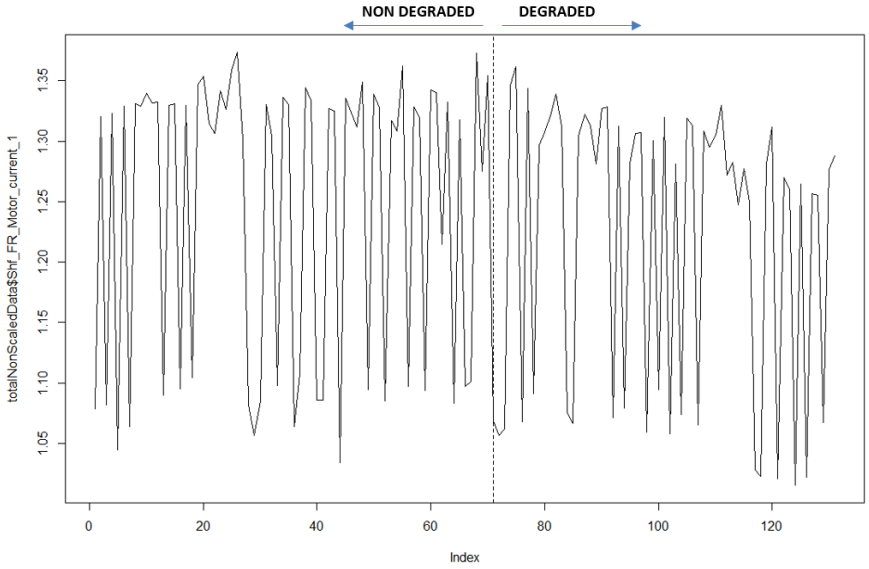


Figure 61: Evolution of feature "Shf_FR_Motor_current_1" in the second analysis

6.1.3 Summary Figures for the Analyses

These tables show the main figures from the analysis stage:

Table 7: Global figures and analysis settings

DESCRIPTION	VALUE
Total number of samples	1853
Number of selected principal components	10
Percentage of data for the reference dataset in Hotelling's phase I	10%
Percentage of data for the test dataset in Hotelling's phase II	90%
Alpha value in Hotelling's calculations	0.005
Time window removed after long-term stops in the bench	1 hour
Long-term stop in the bench	>= 1hour
Number of consecutive points above the upper control limit for triggering an alarm	5

Table 8: Specific figures for the first analysis

FIRST ANALYSIS	VALUE
Total number of samples	968
Number of samples in the reference dataset	96
Number of samples in the test dataset	872
Maximum number of EMA cycles	1574998

FIRST ANALYSIS	VALUE
Upper control limit in the multivariate chart	34.58
Lower control limit in the multivariate chart	1.98
Date and time of the first sample	2019-12-18 15:08:59
Date and time of the first sample	2020-03-02 15:01:49
Date and time of the first alert	2020-01-10 10:22:07

Table 9: Specific figures for the second analysis

SECOND ANALYSIS	VALUE
Total number of samples (features	708
Number of samples in the reference dataset	70
Number of samples in the test dataset	638
Maximum number of EMA cycles	2336304
Upper control limit in the multivariate chart	37.65
Lower control limit in the multivariate chart	2.04
Date and time of the first sample	2020-05-15 23:48:48
Date and time of the last sample	2020-05-29 21:34:57
Date and time of the first alert	2020-05-23 08:57:18

6.2 Automated Cloud Platform

Once the Hotelling's multivariate chart has been developed, the cloud infrastructure has been extended to automatically generate periodic (daily) updates on the chart with the new tests. The results are shown on a dashboard built on the cloud using Power BI, so that the status of the EMA can be monitored until the experiments are fully completed.

The whole data processing pipeline (from the data source located on premises until the final output on the cloud) can be seen in the following diagram:

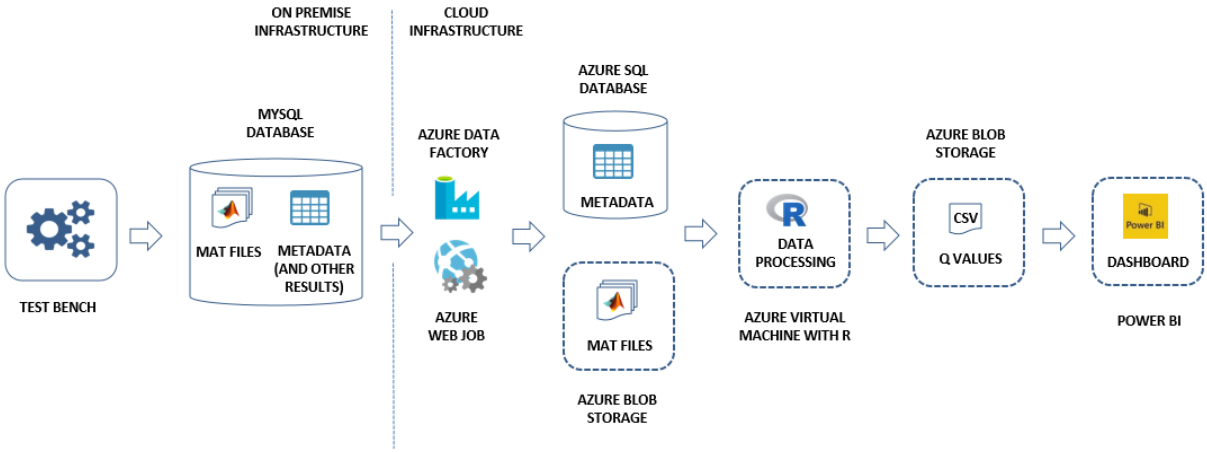


Figure 62: Automated cloud platform

The pieces of R code developed for the preprocessing, feature extraction and analysis stages are executed on a daily basis by means of an Azure virtual machine with an Ubuntu operating system (version 18.04.2 LTS), containing the latest R base installation (version 4.0.1), together with all the R packages used by the code.

The programs have been slightly modified so that only new tests are considered for each execution, and they are scheduled to run through the built-in linux cron utility.

The final result of each execution is a CSV file containing the Hotelling's Q values from the analysis stage, containing all the new tests performed over the last day.

The file is updated on an Azure blob storage container, and it is the input for a Power BI dashboard, which has been designed with the Power BI Desktop utility.

Finally, this dashboard has been uploaded to the cloud service, which periodically imports the new content from the features file located on the Azure Blob Storage.

Two reports have been created in the dashboard, one for each of the two analyses:

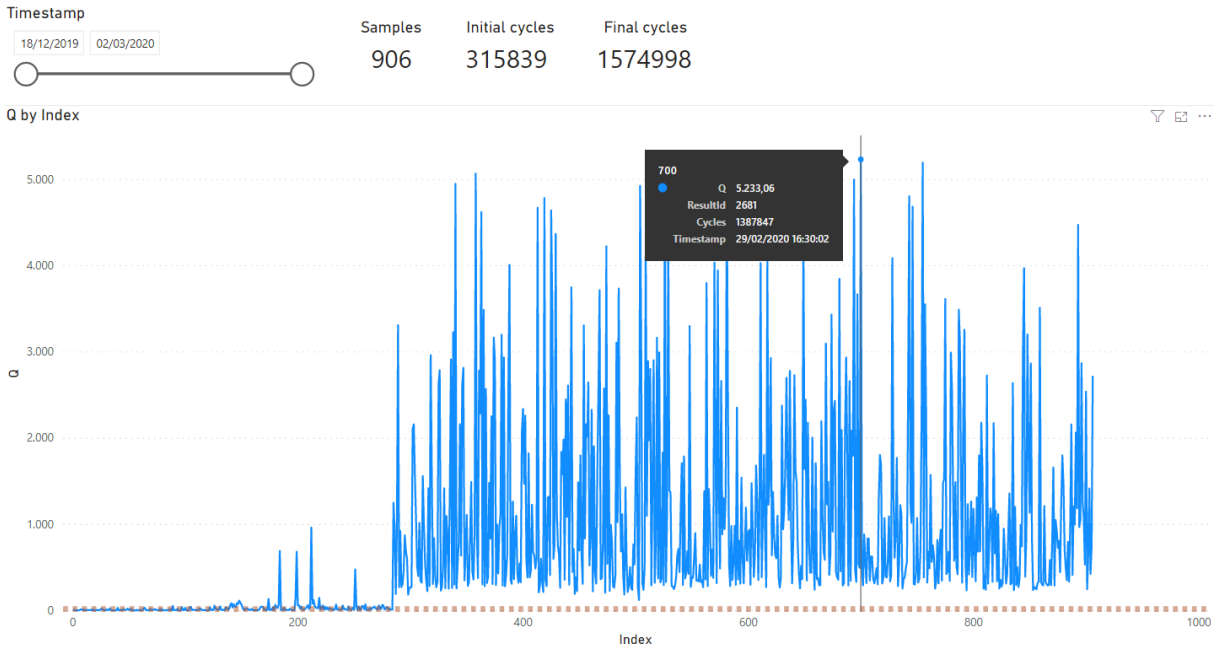


Figure 63: Power BI report for the first analysis

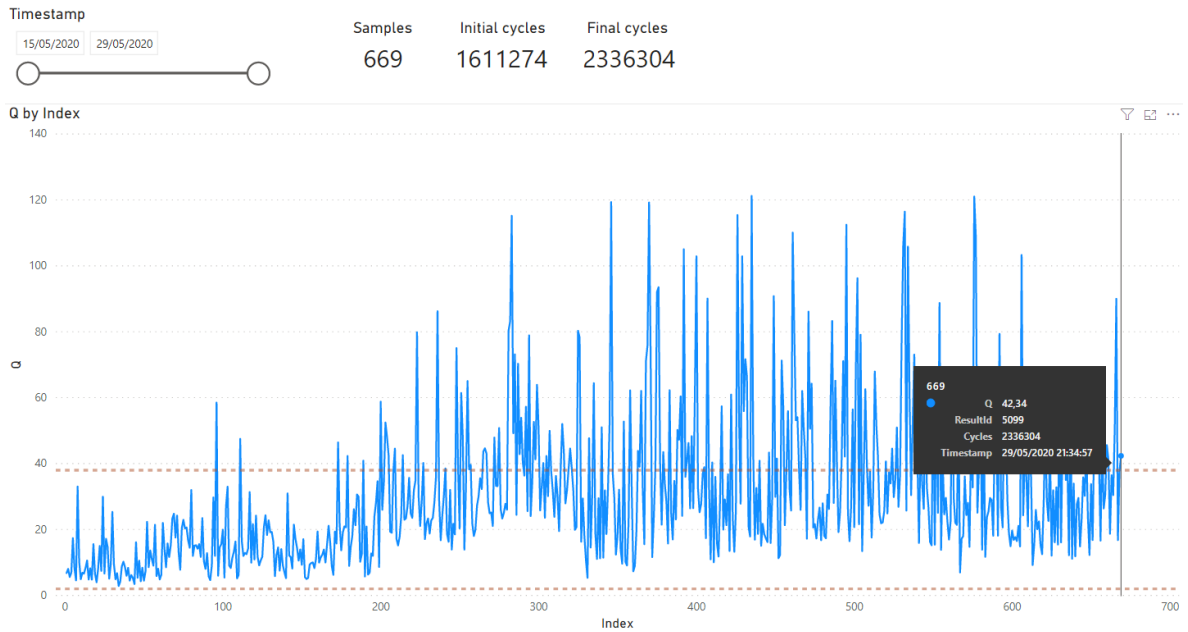


Figure 64: Power BI report for the second analysis

Both reports share the same structure and the same elements:

- A line chart with the upper and lower control limits representing the EMA's safe operation area. The sampling index has been used on the x axis instead of time, because it allows to see the values as a continuum, without the temporal gaps in the data due to the different stops in the bench.

- A slicer to set the date range.
- The number of samples, and the initial and final values of the EMA cycles.
- A tooltip to see extra details of any data point on the chart.

All the different scheduled tasks running on the cloud platform have to be triggered sequentially at the right time for the data processing pipeline to work, until the final visualisation on the dashboard.

Here is a summary of these tasks, together with the time of day when they are executed:

Table 10: Scheduled tasks on the automated cloud platform

TASK	OUTPUT	TIME
Data uploading from the local MySQL database to the cloud	SQL Database synchronisation Extraction of MAT files to the Azure blob storage	00:00
Pre-processing of MAT files with electrical units	Files with physical units	01:00
Feature generation	Updated features file	02:00
Feature analysis	File with Q values	03:00
Power BI dashboard update	Power BI dashboard updated contents	04:00

7 Conclusions and Further Work

7.1 Conclusions

The EMA testing process that is currently taking place on the test rig is not finished yet at the time of writing this document. Only a little over a quarter of the estimated activity has been performed so far, due to a series of issues in the hardware and software controlling the experiments, as was explained in the analysis section.

Being an experimental system, different adjustments and fixes had to be made as the project was progressing, which is something that can be considered normal to some extent.

Currently, the test bench is stopped, and a review is underway on the EMA to determine if it is damaged before continuing the testing.

Despite all these issues, and even if the fatigue test has not been completed until the end, which was the main goal in the initial approach, the information system implemented in this work has proven to be effective in detecting the two main problems that have ocurred during the tests.

The first major problem started when it was noticed that EMA electrical motor was having difficulties with the force exerted by the hydraulic system. This problem got worse over time, and eventually led to a breakdown in the test bench.

The multivariate chart detected that something wrong was happening, as reflected in the increasing number of peaks above the upper limit, once the first critical alert was raised, until reaching some point where the process got out of control.

Even if this failure had nothing to do with the EMA itself, the fact that was detected in the chart it is good proof that the implemented method is reliable, and the calculated features were well chosen. It only remains to be seen whether the added information coming from the acoustic sensors can improved this detection.

The LDA algorithm worked well in the first analysis, pointing to the test bench as the source of the problem.

The second major problem started some days after the new components were replaced, with the EMA experiencing difficulties again with the force applied by the hydraulic system.

This issue was also detected on the chart, with a raising trend in the Q statistical value towards the upper control limit, until a first critical point was reached. After this, the process on the chart

remained up and down around this limit, until it was decided to stop the experiments to determine if this problem had something to do with the EMA, and to rule out any damage.

The LDA method could not explain the source of the problem during the second analysis. Therefore, it would be interesting to use other types of algorithms to get a better result.

Once the experiment activity is resumed again, the automated data processing platform implemented on the cloud will contribute to the monitoring of the testing process, providing regular updates in the Power BI dashboard.

This means that the technitians dealing with the experimental setup have a useful tool to periodically evaluate the status of the EMA, without having to rely on a data scientist to generate the chart.

This automated platform will complement the alarms coming from the test bench itself, and will help in making decisions to prevent any problem that may occur to the EMA during the experiments, as was the case with the second main issue described above, which is something important since the EMA prototype is an expensive system.

The cloud has proven to be a very convenient environment to create this platform and to deal with the large amount of data generated in the process, providing the necessary processing and analytical infrastructure for developing a big data pipeline, without having to use to the ever scarce resources available in corporate servers.

7.2 Further Work

The results presented in this document with the fatigue test could be further extended to create a fully-fledged health monitoring system, which could be incorporated into the actuator.

Once the fatigue test is completed, the resulting dataset could be the basis of more analyses to generate predictive models for estimating the remaining useful life (RUL) of the EMA, or for more studies in areas like energy efficiency, or vibration analysis, for example. In order to do this, it would be desirable to run more experiments with similar EMAs for consistency and repeatability.

Another line of work could be the development of methods for fault detection and isolation (FDI), where the specific type of fault is identified, and procedures for fault detection and diagnosis (FDD), where an estimation of severity of such fault is also provided.

In order to achieve this goal, the results obtained in this work could be combined with the concept of "fingerprint" implemented in the different condition tests performed on the EMA.

These condition tests are registered on the system through the same information infrastructure (database and MAT files), so the amount of work devoted in this work on the preprocessing stage can be leveraged here.

The goal would be to generate condition features associated with specific functionalities in the EMA. These features would have limits where the EMA operation is considered safe, and those limits would have to be obtained during the experiments.

Thus, these safe operating limits could be introduced in a brand new EMA's health monitoring system as nominal values. In addition, those values could be further adjusted for each EMA with the information coming from new condition tests performed during their lifetime, through the addition of an optimization module incorporated into the health monitoring system.

8 Bibliography

ACTUATION 2015. (2016). *Modular Electro Mechanical Actuators for ACARE 2020 Aircraft and Helicopters*. https://cordis.europa.eu/project/id/284915/reporting

Balaban, E., Saxena, A., Narasimhan, S., Roychoudhury, I., Koopmans, M., Ott, C., & Goebel, K. (2015). Prognostic Health-Management System Development for Electromechanical Actuators. *J. Aerosp. Inf. Syst.* https://doi.org/10.2514/1.I010171

Bennett, J. W. (2010). *Fault Tolerant Electromechanical Actuators for Aircraft* [Doctoral dissertation]. Newcastle University.

Chirico, A., & Kolodziej, J. (2014). A Data-Driven Methodology for Fault Detection in Electromechanical Actuators. *Journal of Dynamic Systems, Measurement, and Control*, *136*, 041025. https://doi.org/10.1115/1.4026835

Dalla Vedova, M., Berri, P., & Maggiore, P. (2016). A Smart Electromechanical Actuator Monitor for New Model-Based Prognostic Algorithms. *International Journal of Mechanics and Control*, *17*, 59-66.

Dalla Vedova, M., Lauria, D., Maggiore, P., & Pace, L. (2016). *Electromechanical actuators affected by multiple failures: A simulated-annealing-based fault identification algorithm*. *10*, 219-226.

Di Rito, G., & Schettini, F. (2018). Health monitoring of electromechanical flight actuators via position-tracking predictive models. *Advances in Mechanical Engineering*, *10*(4), 1687814018768146. https://doi.org/10.1177/1687814018768146

Ferreiro, S., Konde, E., Fernández, S., & Prado, A. (2016). INDUSTRY 4.0: Predictive Intelligent Maintenance for Production Equipment. *European Conference of the Prognostics and Health Management Society*, (pp. 1-8).

Giangrande, P., Madonna, V., Sala, G., Kladas, A., Gerada, C., & Galea, M. (2018, octubre 21). *Design and Testing of PMSM for Aerospace EMA Applications*. https://doi.org/10.1109/IECON.2018.8591318

Hussain, Y. M., Burrow, S., Henson, L., & Keogh, P. (2018). A Review of Techniques to Mitigate Jamming in Electromechanical Actuators for Safety Critical Applications. *International Journal of Prognostics and Health Management*, *9(9), 1-11*, 12.

Ismail, M. A. A., Balaban, E., & Spangenberg, H. (2016). Fault detection and classification for flight control electromechanical actuators. *2016 IEEE Aerospace Conference*, 1-10. https://doi.org/10.1109/AERO.2016.7500784

Isturiz, A., Aizpurua, J. I., Hernández, F. E., Iturrospe, A., Muxika, E., & Viñals, J. (2016). A data-driven health assessment method for electromechanical actuation systems. En I. Eballard & A. Bregon (Eds.), *Proceedings of the Third European Conference of the Prognostics and Health Management Society 2016* (pp. 686-694). PHM Society. https://strathprints.strath.ac.uk/56911/

Isturiz, A., Vinals, J., Abete, J. M., & Iturrospe, A. (2012). Health monitoring strategy for electromechanical actuator systems and components. Screw backlash and fatigue estimation. *Recent Advances in Aerospace Actuation Systems and Components, 5*. https://www.researchgate.net/publication/258927355_HEALTH_MONITORING_STRATEGY_FOR_ELECTROMECHANICAL_ACTUATOR_SYSTEMS_AND_COMPONENTS_SCREW_BACKLASH_AND_FATIGUE_ESTIMATION

Isturiz, A., Viñals, J., Fernandez, S., Basagoiti, R., Torre Arnanz, E. de la, & Novo, J. (2010). Development of an aeronautical electromechanical actuator with real time health monitoring capability. *Proceedings of the 4th International Conference on Recent Advances in Aerospace Actuation Systems and Components, 2010 (R3ASC)*. 4th International Conference on Recent Advances in Aerospace Actuation Systems and

Components, 2010 (R3ASC), Toulouse (France). http://congres.insa-toulouse.fr/R3ASC/index.html

Jones, R. I. (2002). The more electric aircraft—Assessing the benefits. *Proceedings of the Institution of Mechanical Engineers, 2002, Part G: 259-269*. https://bv.unir.net:2257/docview/213210175?pq-origsite=summon

Lei, Y. (2016). *Intelligent Fault Diagnosis and Remaining Useful Life Prediction of Rotating Machinery*. Butterworth-Heinemann.

Li, J., Yu, Z., Huang, Y., & Li, Z. (2016). *A review of electromechanical actuation system for more electric aircraft*. 490-497. https://doi.org/10.1109/AUS.2016.7748100

López de Calle, K., Ferreiro, S., Arnaiz, A., & Sierra, B. (2019). Dynamic condition monitoring method based on dimensionality reduction techniques for data-limited industrial environments. *Computers in Industry*, *112*, 103114. https://doi.org/10.1016/j.compind.2019.07.004

Maggiore, P., Dalla Vedova, M., Pace, L., & Desando, A. (2014, julio 10). *Definition of parametric methods for fault analysis applied to an electromechanical servomechanism affected by multiple failures*.

NIST. (2020). *Hotelling Multivariate Control Chart*. National Institute of Standards and Technology. https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/hotell.htm

Pang, J., Liu, D., Peng, Y., & Peng, X. (2018). Optimize the Coverage Probability of Prediction Interval for Anomaly Detection of Sensor-Based Monitoring Series. *Sensors*, *18*(4), 967. https://doi.org/10.3390/s18040967

Qiao, G., Liu, G., Shi, Z., Wang, Y., Ma, S., & Teik, C. L. (2018). A review of electromechanical actuators for More/All Electric aircraft systems. *Proceedings of the Institution of*

Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 232(22), 4128-4151. https://doi.org/10.1177/0954406217749869

Rosero, J. A., Ortega, J. A., Aldabas, E., & Romeral, L. (2007). Moving towards a more electric aircraft. *IEEE Aerospace and Electronic Systems Magazine*, *22*(3), 3-9. https://doi.org/10.1109/MAES.2007.340500

Ruiz-Carcel, C., & Starr, A. (2018). Data-Based Detection and Diagnosis of Faults in Linear Actuators. *IEEE Transactions on Instrumentation and Measurement*, *67*(9), 2035-2047. https://doi.org/10.1109/TIM.2018.2814067

Todeschi, M., & Baxerres, L. (2015). Health Monitoring for the Flight Control EMAs. *IFAC-PapersOnLine*, *48*(21), 186-193. https://doi.org/10.1016/j.ifacol.2015.09.526

van der Linden, F. L. J., Dreyer, N., & Dorkel, A. (2016). EMA Health Monitoring: An overview. *Recent Advances in Aerospace Actuation Systems and Components*, 21-26. https://elib.dlr.de/104367/

Yang, J., Guo, Y., & Zhao, W. (2019). Long short-term memory neural network based fault detection and isolation for electro-mechanical actuators. *Neurocomputing*, *360*, 85-96. https://doi.org/10.1016/j.neucom.2019.06.029

Zhang, Y., Liu, D., Yu, J., Peng, Y., & Peng, X. (2017). EMA remaining useful life prediction with weighted bagging GPR algorithm. *Microelectronics Reliability*, *75*, 253-263. https://doi.org/10.1016/j.microrel.2017.03.021

Zhang, Y., Liu, L., Peng, Y., & Liu, D. (2018). An Electro-Mechanical Actuator Motor Voltage Estimation Method with a Feature-Aided Kalman Filter. *Sensors*, *18*(12), 4190. https://doi.org/10.3390/s18124190

9 Annexes

9.1 SQL Queries in the Preprocessing Phase

The database schema which is the base for following queries can be seen in the section devoted to the exploratory analysis of the result data in the document.

9.1.1 SQL Query for Test Results

Below, a Transact-SQL query is presented obtaining the results which are the input for the preprocessing stage. These results correspond to fatigue test executions in the EMA test campaign.

A query similar to this is created dynamically in the R code (without the T-SQL variables "@testType" and "@campaignId", which are shown here so that the query can be executed directly on the database for demonstration purposes).

```
int = 3  -- Fatigue test
DECLARE @campaignId int = 3 -- EMA testing campaign

SELECT [Id]
      ,[IdEjecucion]
      ,[NombreFichero]
      ,[Exportar]
      ,[NombreTablaDeBaseDeDatos]
FROM [isselub].[resultados]
WHERE IdEjecucion IN
         (
         SELECT [Id]
         FROM [isselub].[ejecucion]
         WHERE IdEnsayo IN
                  (SELECT [Id]
                  FROM [isselub].[ensayo]
                  WHERE IdTipoEnsayo = @testType
                           and IdCampana = @campaignId))
```

9.1.2 SQL Query for Signal Metadata

The following query shows a Transact-SQL code that returning the name of the signal ("SignalName") for a particular input ("cDaqAnalogInput3") from a acquisition equipment (CompactDAC).

Also, the polynomial coefficients for this signal are returned in this query, related to calibration fixes ("SignalCoefficients1") and the conversion to physical units ("SignalCoefficients2").

This query can be used not only for inputs, but for outputs and internal signals as well.

Same as the previous section, this query is created dynamically in the R code used for the preprocessing stage.

```
-- An example of an input identifier
DECLARE @variable nvarchar(400) = 'cDaqAnalogInput3'

SELECT top 1 [NombreCastellano] AS SignalName,
                          [Calibracion] AS SignalCoefficients1,
                          [Ecuacion] AS SignalCoefficients2
FROM
(
SELECT [NombreCastellano],
       [Calibracion],
           [Ecuacion]
FROM [isselub].[medida]
        INNER JOIN [isselub].[sensor] ON [isselub].[medida].[IdSensor] = [isselub].[sensor].[Id]
WHERE IdInput IN
         (SELECT TOP 1 [Id]
          FROM
             (SELECT [Id], LEN([NombreVbleRegistrada]) AS LongitudNombreVbleRegistrada
              FROM [isselub].[input]
              WHERE PATINDEX('%' + NombreVbleRegistrada + '%', @variable) > 0) AS T1
          ORDER BY LongitudNombreVbleRegistrada DESC)
UNION

SELECT [NombreCastellano],
       [Calibracion],
           [Ecuacion]
FROM [isselub].[actuacion]
        INNER JOIN [isselub].[actuador] ON [isselub].[actuacion].[IdActuador] = [isselub].[actuador].[Id]
WHERE IdOutput IN
         (SELECT TOP 1 [Id]
          FROM
             (SELECT [Id], LEN([NombreVbleRegistrada]) AS LongitudNombreVbleRegistrada
              FROM [isselub].[output]
              WHERE PATINDEX('%' + NombreVbleRegistrada + '%', @variable) > 0) AS T2
                   ORDER BY LongitudNombreVbleRegistrada DESC)
UNION

SELECT TOP 1 [NombreCastellano],
                          '[1,0]' AS [Calibracion],
                          [Ecuacion]
FROM
(
         SELECT [NombreCastellano], LEN([NombreVbleRegistrada]) AS LongitudNombreVbleRegistrada, [Ecuacion]
         FROM [isselub].[senalinterna]
         WHERE PATINDEX('%' + NombreVbleRegistrada + '%', @variable) > 0
) AS T3
ORDER BY LongitudNombreVbleRegistrada DESC

UNION

SELECT [NombreCastellano],
       COALESCE ([Calibracion], '[1,0]') AS [Calibracion],
           [Ecuacion]
FROM [isselub].[medida]
         LEFT OUTER JOIN [isselub].[sensor] ON [isselub].[medida].[IdSensor] = [isselub].[sensor].[Id]
```

```
WHERE PATINDEX('%' + NombreVbleRegistrada + '%', @variable) > 0
) AS T4
```

9.2 Signal Preprocessing Code

Preprocessing code is executed with function “processResults”. The code with the call to this function is located at the end of the script.

```
# Instalación de paquetes
# install.packages("RODBC")
# install.packages("R.matlab")
# install.packages("AzureStor")

# Carga de paquetes en el entorno de R
library(RODBC)
library(R.matlab)
library(AzureStor)

# Variable que indica si estamos ejecutando el script en Azure de forma automatizada
ubuntuVirtualMachine <- FALSE

# Función que obtiene una conexión a la base de datos de Azure
getAzureSQLDatabaseServerConnection <- function() {

        # Base de datos SQL Database en Azure
        AzureSQLDatabaseServer <- "sqlisselub.database.windows.net"
        AzureSQLDatabaseUser <- "adminsql"
        AzureSQLDatabasePassword <- "Tekniker2016@"
        AzureSQLDatabase <- "configuration"
        if(ubuntuVirtualMachine == TRUE) {
                # Driver ODBC para la virtualización de Ubuntu
                AzureSQLDatabaseDriver <- "ODBC Driver 17 for SQL Server"
        }
        else {
                # Driver ODBC para la virtualización de Windows
                AzureSQLDatabaseDriver <- "SQL Server"
        }

        AzureSQLDatabaseConnectionString <- paste0(
                "Driver=", AzureSQLDatabaseDriver,
                ";Server=", AzureSQLDatabaseServer,
                ";Database=", AzureSQLDatabase,
                ";Uid=", AzureSQLDatabaseUser,
                ";Pwd=", AzureSQLDatabasePassword)

        # Conexión con la base de datos
        result <- odbcDriverConnect(AzureSQLDatabaseConnectionString)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003".
# Deben generarse cuatro dígitos para el número de la campaña.
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
# Resultado: cadena de caracteres con el nombre de la campaña de tests
# Se genera siempre cuatro dígitos para el número de la campaña (se completa con ceros).
# Por ejemplo, para la campaña 3 se crea la siguiente cadena: "campana0003"
# ------------------------------------------------------------------------------------------------------------

getTestCampaignName <- function(testCampaignId) {

        # Creamos una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003"
        # El número de dígitos en la cadenaserá siempre de cuatro
        campaignNumberString = as.character(testCampaignId)
        campaignNumberString = paste("campana", formatC(testCampaignId, width=4, flag="0"), sep="")

        return(campaignNumberString)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los datos de un archivo y los carga en un dataframe
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileName: nombre del archivo
#  - azureContainerName: nombre del contenedor de Azure donde está el archivo
# Resultado: dataframe con señales físicas correspondientes al resultado
```

```
# ------------------------------------------------------------------------------------------------------------

getFileData <- function(testCampaignId, testTypeId, fileName, azureContainerName) {

        # Contenedor de Azure donde están los archivos con datos físicos que hay que procesar
        blobStorageContainer <- getAzureContainer(testCampaignId, testTypeId, azureContainerName)


        # Archivo temporal para guardar los datos en local
        tmpfile <- tempfile()

        result <- tryCatch(
        {
                        # Descargamos el archivo de Azure
                        print(paste0("Downloading file: ", fileName))

                        storage_download(blobStorageContainer, fileName, tmpfile, overwrite=TRUE)

                        # Leemos los datos en un dataframe
                        print(paste0("Reading file: ", fileName))

                        result <- read.csv(tmpfile, header = TRUE, sep = ",", dec = ".")
                },
        error=function(cond) {

                        # Se ha producido un error
                        print(paste0("Cannot download file: ", fileName))

            result <- NULL
        },
        finally={

                        # Eliminamos el archivo temporal
                        file.remove(tmpfile)

                }
        )

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el Id de resultado a partir del cual se van a procesar los datos.
# Para ello se obtiene el mayor Id de resultado de los posibles descriptores que existan en Azure
# La consulta se realiza en la base de datos SQL Database de Azure donde se almacena la información.
# Parámetros:
# - testCampaignId: identificador de campaña de ensayos
# - testTypeId: identificador de tipo de ensayo
# - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: dataframe con los registros de los resultados del tipo de ensayo y campaña especificados
# ------------------------------------------------------------------------------------------------------------

getResultIdToStartFrom <- function(testCampaignId, testTypeId, processNewResultsOnly) {

        ResultId <- 0
        if(processNewResultsOnly == TRUE) {

                featuresFileName = paste0(getTestCampaignName(testCampaignId),
                                                                "/", getTestTypeName(testTypeId),
                                                                "/Features/features.csv")
                dfFeatures <- getFileData(testCampaignId, testTypeId, featuresFileName, "results")
                if(!is.null(dfFeatures) == TRUE){
                        ResultId <- max(dfFeatures$ResultId)
                }
        }

        return(ResultId)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los resultados correspondientes a un tipo de ensayo en una campaña.
# La consulta se realiza en la base de datos SQL Database de Azure donde se almacena la información.
# Parámetros:
# - testCampaignId: identificador de campaña de ensayos
# - testTypeId: identificador de tipo de ensayo
# - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: dataframe con los registros de los resultados del tipo de ensayo y campaña especificados
# ------------------------------------------------------------------------------------------------------------

getResults <- function(testCampaignId, testTypeId, processNewResultsOnly) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Obtenemos el Id de resultado a partir del cual se van a procesar los datos.
        # Hay dos posibilidades:
```

```
        # * Procesar todos los resultados (processNewResultsOnly == FALSE)
        # * Procesar sólo los nuevos resultados (processNewResultsOnly == FALSE)
        #   Para ello, se comprueba el último Id de resultado procesado en los descriptores.
        ResultId <- getResultIdToStartFrom(testCampaignId, testTypeId, processNewResultsOnly)

        # Consulta
        sqlQuery <- paste0("SELECT * ",
                           "FROM ",
                           "( ",
                                               "SELECT [Id], ",
                                               "              (CASE  WHEN  [NombreFichero]  IS  NOT  NULL  AND
LEN([NombreFichero]) > 0 ",
                                               "                THEN '/', getTestCampaignName(testCampaignId), "/' + (CAST([Id] AS
nvarchar) + '_' +  [NombreFichero]) ",
                                               "                ELSE '' ",
                                               "        END) AS [NombreFichero], ",
                                               "       [NombreTablaDeBaseDeDatos] ",
                                               "FROM [isselub].[resultados] ",
                                               "WHERE IdEjecucion IN ",
                                               "   (SELECT [Id] ",
                                               "    FROM [isselub].[ejecucion] ",
                                               "    WHERE IdEnsayo IN ",
                                               "        (SELECT [Id] ",
                                               "         FROM [isselub].[ensayo] ",
                                               "         WHERE IdTipoEnsayo = ", as.character(testTypeId),
                                               "             AND IdCampana = ", as.character(testCampaignId),
"))",
                                               ") AS T1 ",
                                               "WHERE [Id] > ", ResultId)

                                               #"WHERE                    NombreFichero                    LIKE
'%2020_01_07_13_31_45_902_cDaqCapture.mat%'")
                                               #"WHERE                    NombreFichero                    LIKE
'%2018_12_11_12_48_47_REGISTRADORENSAYO%'")
                                               #"WHERE                    NombreFichero                    LIKE
'%2020_01_07_14_38_59_REGISTRADORENSAYO%'")
                                               #"WHERE [Id] = 983")

                                               #"WHERE                    NombreFichero                    LIKE
'%2020_01_07_13_31_45_902_cDaqCapture.mat%' ",
                                               #"                          OR        NombreFichero        LIKE
'%2020_01_07_14_38_59_REGISTRADORENSAYO%' ",
                                               #"   OR [Id] = 981")
                                               #"WHERE [Id] > 955 ORDER BY [Id] ASC")
                                               #"WHERE [Id] = 423")
                                               #"WHERE [Id] > 5075 ORDER BY [Id] ASC")


        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, sqlQuery)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        return(dfQueryResult)
}

# ------------------------------------------------------------------------------------------------------------
# Comprueba si en un vector de cadenas de caracteres (nombres de variables) hay alguna que acaba con una
subcadena
# Parámetros:
# - resultFileVariables: vector de cadenas de caracteres que contiene nombres de variables de un archivo de
resultados
# - ending: cadena de caracteres que hay que buscar al final de cada cadena (nombre de variable)
# Resultado:
#       TRUE: Hay al menos una cadena de caracteres (nombre de variable) del vector que acaba con la subcadena
especificada
#   FALSE: no hay ninguna cadena de caracteres que cumpla lo anterior
# ------------------------------------------------------------------------------------------------------------

checkStringEndingInResultVariables <- function(resultFileVariables, endingString) {

        result = any(grepl(endingString, resultFileVariables))

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Devuelve los distintos grupos de muestreo de un archivo de resultados de Ingesys
# Por grupo de muestreo se entiende un grupo de tres datos:
#   * Periodo de muestreo
#   * Número de puntos
#   * Índice del punto que es el origen de la base de tiempos (el primer punto tiene índice 0)
# Parámetros:
# - resultFileVariables: vector de cadenas de caracteres que contiene nombres de variables de un archivo de
resultados
```

```
# - resultFileData: lista con el contenido total de un archivo de resultados (cada elemento de la lista:
variable = matriz)
# Resultado: Se genera una lista de elementos. Cada elemento es a su vez una lista, con tres datos:
#       * SamplingPeriod: Periodo de muestreo
#       * NumberOfPoints: Número de puntos
#       * PointsAtTimeOrigin: Punto origen de la base de tiempos
# Cada uno de estos tres datos se corresponde con:  raíz_variable + terminación específica, por ejemplo:
#       * "SimulinkRTW.0.9.100000.us.n"
#       * "SimulinkRTW.0.9.100000.us.Ts.ms"
#       * "SimulinkRTW.0.9.100000.us.nPRE"
# En un mismo archivo de resultados puede haber varios grupos de este tipo de variables. Otro ejemplo de
grupo:
#       * "SimulinkRTW.0.9.20.us.n"
#       * "SimulinkRTW.0.9.20.us.Ts.ms"
#   * "SimulinkRTW.0.9.20.us.nPRE"
#  Se ve que el último grupo que tiene una raiz ("SimulinkRTW.0.9.20") distinta a la del anterior
("SimulinkRTW.0.9.100000").
# Hay que ir buscando en el archivo estos grupos e ir anotando sus valores en la lista final.
# ------------------------------------------------------------------------------------------------------

getIngesysResultFileSamplingGroups <- function(resultFileVariables, resultFileData) {

        result = list()

        # Pasos en el bucle:
        # 1) Se busca una variable que acabe en ".us.Ts.ms"
        # 2) Se obtiene la raíz de la variable sin la terminación ".us.Ts.ms"
        # 3) Se obtienen los valores mediante dicha raíz y los siguientes sufijos:
        #         raíz + ".us.Ts.ms": periodo de muestreo en milisegundos (lo multiplicamos por 0.001 para
convertirlo a segundos)
        #         raíz + ".us.n": número de puntos
        #         raíz + ".us.nPRE": punto que es el origen de tiempo
        # 4) Se guardan los tres datos en una lista
        # 5) Se vuelve al paso 1) hasta que no haya más variables

        for(v in 1:length(resultFileVariables)) {

                # Se comprueba si la variable acaba en ".us.Ts.ms"
                if (checkStringEndingInResultVariables(resultFileVariables[v], ".us.Ts.ms$") == TRUE) {

                        # Se obtiene la raíz de la variable sin la terminación ".us.Ts.ms$"
                        variableRoot = sub(".us.Ts.ms", "", resultFileVariables[v])

                        # Se obtienen los valores
                        dataGroup = list()

                        # Periodo de muestreo (segundos)
                        resultFileVariable = resultFileVariables[v]
                        if(!is.null(resultFileVariable) && !is.na(resultFileVariable)) {

                                dataGroup$SamplingPeriod                                                =
as.numeric(resultFileData[resultFileVariable][1]) * 1E-3
                        }

                        # Número de puntos
                        resultFileVariable = paste0(variableRoot, ".us.n")
                        if(!is.null(resultFileVariable) && !is.na(resultFileVariable)) {

                                dataGroup$NumberOfPoints                                                =
as.integer(resultFileData[resultFileVariable][1])
                        }

                        # Origen de tiempo
                        resultFileVariable = paste0(variableRoot, ".us.nPRE")
                        if(!is.null(resultFileVariable) && !is.na(resultFileVariable)) {

                                dataGroup$PointsAtTimeOrigin                                            =
as.integer(resultFileData[resultFileVariable][1])
                        }

                        # Agregamos la lista a los resultados
                        result <- append(result, list(dataGroup))
                }
        }

        return(result)
}

# ------------------------------------------------------------------------------------------------------
# Devuelve los distintos grupos de muestreo (normalmente sólo habrá uno) de un archivo de resultados de
CompaqDAC.
# Por grupo de muestreo se entiende un grupo de tres datos:
#   * Periodo de muestreo
#   * Número de puntos
#   * Índice del punto que es el origen de la base de tiempos (el primer punto tiene índice 0)
```

```
# En un archivo de CompaqDAC se obtendrán los dos primeros valores, el índice del punto que es el origen de
la base de tiempos se pone a NULL.
# Se devuelven tres datos para unificar el procesamiento posterior con los archivos de Ingesys.
# En dicho procesamiento posterior, se sabrá que es un archivo de CompaqDAC ya que el tercer dato vale NULL.
# Parámetros:
# - resultFileVariables: vector de cadenas de caracteres que contiene nombres de variables de un archivo de
resultados
# - resultFileData: lista con el contenido total de un archivo de resultados (cada elemento de la lista:
variable = matriz)
# Resultado: Se genera una lista de un único elemento, que es a su vez una lista, con tres datos:
#       * SamplingPeriod: Periodo de muestreo (se pone a NULL)
#       * NumberOfPoints: Número de puntos
#       * PointsAtTimeOrigin: Punto origen de la base de tiempos (se pone a NULL)
# Esto se hace así para que el procesamiento sea similar a Ingesys
# ------------------------------------------------------------------------------------------------------------

getCompaqDacResultFileSamplingGroups <- function(resultFileData) {

        result = list()

        # Pasos en el bucle:
        # 1) Se busca una variable cuyo nombre sea "time"
        # 2) Se obtiene la longitud del vector de sus datos.
        #    Este valor será el número de puntos del único grupo de muestreo del archivo.
        # 3) Se obtiene el periodo de muestreo restando los valores entre dos puntos consecutivos del vector
"time" (los dos primeros)

        # Se obtienen los valores
        dataGroup = list()

        # Obtención de los datos del grupo de muestreo.
        # Para obtener el periodo de muestreo, hay que restar los valores entre los dos primeros puntos del
vector "time"
        # (utilizamos la función abs para devolver siempre un valor positivo, aunque no debería hacer falta)
        dataGroup$SamplingPeriod <- abs(resultFileData[["time"]][2] - resultFileData[["time"]][1])
        dataGroup$NumberOfPoints <- length(resultFileData[["time"]])
        dataGroup$PointsAtTimeOrigin <- NULL

        # Agregamos la lista a los resultados
        result <- append(result, list(dataGroup))

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Busca el índice de grupo de muestreo al que pertenece una variable de un archivo de resultados.
# El índice es la posición del grupo de muestreo en la lista de grupos de muestreos del archivo de resultados.
# Parámetros:
# - samplingGroups: lista de grupos de muestreos del archivo de resultados
# - resultFileData: datos del archivo de resultados
# - resultFileVariable: variable de un archivo de resultados
# Resultado: índice de grupo de muestreo al que pertenece la variable de un archivo de resultados
# ------------------------------------------------------------------------------------------------------------

getResultFileSamplingGroupIndexForVariable <- function(samplingGroups, resultFileData, resultFileVariable) {

    result = NULL

    # Recorremos los distintos grupos de muestreos
    for(sg in 1:length(samplingGroups)) {

        # Se comprueba si el número de puntos del grupo de muestreo coincide con el número de datos de la
variable
        samplingGroup = samplingGroups[[sg]]
        numberOfPointsOfVariable = length(resultFileData[[resultFileVariable]])
        if(samplingGroup$NumberOfPoints == numberOfPointsOfVariable) {

            # Anotamos el índice del grupo y salimos del bucle
            result = sg
            break
        }
    }

    return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Busca el nombre de una señal a partir del nombre de una variable de un archivo o tabla de resultados.
# El nombre de la señal se obtendrá en la base de datos en función de la naturaleza de la variable, que puede
ser:
#       * Una medida de sensor (tabla "medida")
#       * Una actuación (tabla "actuacion")
#       * Una señal interna (tabla "senalinterna")
#   * Una señal acústica (a partir de la tabla "medida")
# Parámetros:
# - resultVariable: nombre de variable de un archivo o tabla de resultados
```

```
# Resultado: dataframe con los siguientes valores:
# - SignalName: nombre de la señal.
# - SignalCoefficients1: cadena de caracteres con el formato "[Coeficiente1;Coeficiente2]"
# - SignalCoefficients2: cadena de caracteres con el formato "[Coeficiente1;Coeficiente2]"
#   Es una cadena de caracteres tendrá el siguiente formato: "[Coeficiente1;Coeficiente2]"
# Los coeficientes permitirán pasar los datos de la variable a valores físicos.
# Se aplicarán al valor de la variable de la misma forma que se hace en la función polyval de Matlab.
# (primero se aplicará SignalCoefficients1 y después SignalCoefficients2)
# Nota: no hay un valor explícito de SignalCoefficients1 para las señales internas en base de datos.
#       Para las señales internas, se utilizará '[1;0]' para SignalCoefficients1, que no cambiará el valor
del dato
# ------------------------------------------------------------------------------------------------------

getSignalMetadata <- function(resultVariable) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        # La variable se corresponderá a uno de estos tres conceptos: medida, actuación o señal interna.
        # Para evitar ir haciendo consultas secuenciales hasta encontrar el nombre, hacemos una única consulta.
        # Cada parte entre las instrucciones UNION se corresponderá a uno de estos conceptos:
        #       * Medida,
        #   * Actuación,
    #   * Señal interna
        #   * Señal acústica
        # Los tres primeros conceptos se encuentran en archivos de Ingesys. El último en archivos de CompaqDAC.
        # Nota: el ORDER BY + TOP 1 de las consultas interiores se pone para devol
        query <- paste0("SELECT TOP 1 [NombreCastellano] AS SignalName, ",
                                        "             [Calibracion] AS SignalCoefficients1, ",
                                        "             [Ecuacion] AS SignalCoefficients2 ",
                                        "FROM ",
                                        "  (SELECT [NombreCastellano], ",
                                        "          [Calibracion], ",
                                        "          [Ecuacion] ",
                                        "   FROM [isselub].[medida] ",
                                        "                        INNER    JOIN    [isselub].[sensor]    ON
[isselub].[medida].[IdSensor] = [isselub].[sensor].[Id] ",
                                        "   WHERE IdInput IN ",
                                        "       (SELECT TOP 1 [Id] ",
                                        "        FROM ",
                                        "           (SELECT [Id], ",
                                        "                                    LEN([NombreVbleRegistrada])  AS
LongitudNombreVbleRegistrada ",
                                        "            FROM [isselub].[input] ",
                                        "            WHERE PATINDEX('%' + NombreVbleRegistrada + '%', '",
resultVariable, "') > 0) AS T1 ",
                                        "        ORDER BY LongitudNombreVbleRegistrada DESC) ",
                                        "   UNION ",
                                        "   SELECT [NombreCastellano],  ",
                                        "          [Calibracion], ",
                                        "          [Ecuacion] ",
                                        "   FROM [isselub].[actuacion] ",
                                        "                        INNER   JOIN    [isselub].[actuador]    ON
[isselub].[actuacion].[IdActuador] = [isselub].[actuador].[Id] ",
                                        "   WHERE [IdOutput] IN ",
                                        "       (SELECT TOP 1 [Id] ",
                                        "        FROM ",
                                        "           (SELECT [Id], ",
                                        "                                    LEN([NombreVbleRegistrada])  AS
LongitudNombreVbleRegistrada ",
                                        "            FROM [isselub].[output] ",
                                        "            WHERE PATINDEX('%' + NombreVbleRegistrada + '%', '",
resultVariable, "') > 0) AS T1 ",
                                        "        ORDER BY LongitudNombreVbleRegistrada DESC) ",
                                        "   UNION ",
                                        "   SELECT TOP 1 [NombreCastellano], ",
                                        "                '[1,0]' AS [Calibracion], ",
                                        "                [Ecuacion] ",
                                        "   FROM ",
                                        "      (SELECT [NombreCastellano], ",
                                        "              [Ecuacion], ",
                                        "                                    LEN([NombreVbleRegistrada])  AS
LongitudNombreVbleRegistrada ",
                                        "       FROM [isselub].[senalinterna] ",
                                        "          WHERE PATINDEX('%' + NombreVbleRegistrada + '%', '",
resultVariable, "') > 0) AS T3 ",
                                        "   ORDER BY LongitudNombreVbleRegistrada DESC ",
                                        "   UNION ",
                                        "   SELECT [NombreCastellano], ",
                                        "          COALESCE ([Calibracion], '[1,0]') AS [Calibracion], ",
                                        "          [Ecuacion] ",
                                        "   FROM [isselub].[medida] ",
                                        "                    LEFT   OUTER   JOIN   [isselub].[sensor]   ON
[isselub].[medida].[IdSensor] = [isselub].[sensor].[Id] ",
```

```
                                        "      WHERE  PATINDEX('%'  +  NombreVbleRegistrada  +  '%',  '",
resultVariable, "') > 0 ",
                                        ") AS T4")

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        return(dfQueryResult)
}

# ------------------------------------------------------------------------------------------------------------
# Función auxiliar para utilizar con sapply.
# Se utilizará en el proceso de obtención de los valores físicos.
# ------------------------------------------------------------------------------------------------------------

applySignalCoefficients <- function(value, coefficientItem1, coefficientItem2) {

        # Si los coeficientes son coefficientItem1 = 1 y coefficientItem2 = 0, no hacemos nada.
        # Esto lo hacemos ya que estos valores no afectan a "value", y evitamos las operaciones ya que "value"
puede ser muy grande
        if(!is.null(value) && !is.na(value) && !(coefficientItem1 == 1 && coefficientItem2 == 0)) {
                # Cálculo similar a la función polyval de Matlab
                (value * coefficientItem1) + coefficientItem2
        }
        else {
                value
        }
}

# ------------------------------------------------------------------------------------------------------------
# Evalua la expresión de un coefiente de una señal.
# Parámetros:
# - signalCoefficient: cadena de caracteres que representa un coefiente que se aplica a una señal.
#   Puede ser una cadena que representa un valor numérico o bien una expresión matemática.
#   Un ejemplo de cadena que representa una expresión matemática: "-4*400/16"
# - resultOnError: resultado numérico por defecto si hay un error en la evaluación del coeficiente
# Resultado: valor numérico del coeficiente
# ------------------------------------------------------------------------------------------------------------

evaluateSignalCoefficientItem<- function(coefficientItem, resultOnError) {

        result <- tryCatch(
        {
                        # Comprobamos que haya algún dato en el coeficiente
                        if(is.null(coefficientItem) || is.na(coefficientItem) || nchar(coefficientItem)==
0) {

                                return(resultOnError)
                        }
                        else {

                                # Evaluación de la expresión del coeficiente
                                eval(parse(text=coefficientItem))
                        }
        },
        error=function(cond) {

                        # Se ha producido un error
            return(resultOnError)
        },
        warning=function(cond) {

                        # Se ha producido algún problema
            return(resultOnError)
        },
        finally={}
    )

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
#   Obtiene los items de los coeficientes de una cadena de caracteres con el formato
"[Coeficiente1;Coeficiente2]".
# Parámetros:
# - signalCoefficients: cadena de caracteres con el formato "[Coeficiente1;Coeficiente2]"
# Resultado: los datos que se devuelven son el valor numérico de "Coeficiente1" y "Coeficiente2".
# El resultado se devuelve en un vector de dos elementos de datos numéricos
# ------------------------------------------------------------------------------------------------------------

getSignalCoefficientItems <- function(signalCoefficients) {

        # Por defecto, indicamos un valor que no afecte al valor de la señal
        result = c(1,0)
```

```
        if(!is.null(signalCoefficients) && !is.na(signalCoefficients)) {

                # Se eliminan espacios en blanco al comienzo y al final
                coefficientItems <- trimws(signalCoefficients)

                # Comprobación de que el primer caracter es '[' y el último ']'
                if    (substring(coefficientItems,    1,    1)    ==    "["    &&    substr(coefficientItems,
nchar(coefficientItems), nchar(coefficientItems)) == "]" ) {

                        # Se eliminan los caracteres iniciales y finales
                        coefficientItems <- gsub('^.|.$', '', coefficientItems)
                        coefficientItems <- strsplit(coefficientItems, ";")

                        # Se obtienen las cadenas de caracteres de los coeficientes
                        coefficientItem1 <- (coefficientItems[[1]][1])
                        coefficientItem2 <- (coefficientItems[[1]][2])

                        # Evaluación de la expresión del primer coeficiente
                        # Si hay error, devolvemos 1 para no alterar el valor del dato de la señal
                        coefficientItem1 <- evaluateSignalCoefficientItem(coefficientItem1, 1)

                        # Evaluación de la expresión del segundo coeficiente
                        # Si hay error, devolvemos 0 para no alterar el valor del dato de la señal
                        coefficientItem2 <- evaluateSignalCoefficientItem(coefficientItem2, 0)

                        result = c(coefficientItem1, coefficientItem2)
                }
        }

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Modifica caracteres especiales en el nombre de una señal:
# - Los espacios en blanco " "se sustituyen por guiones bajos "_"
# - Los paréntesis "(" y ")" se eliminan
# Esto para evitar posteriores problemas en la gestión de los datos.
# Parámetros:
#  - signalName: nombre de una señal
# Resultado: nombre de la señal con las modificaciones
# ------------------------------------------------------------------------------------------------------------

modifySignalNameSpecialCharacters <- function(signalName) {

        result <- signalName

        # Se eliminan espacios en blanco al comienzo y al final
        result <- trimws(result)

        # Reemplazamos caracteres en los nombres de las señales para evitar posteriores problemas:
        # - Los espacios en blanco " "se sustituyen por guiones bajos "_"
        # - Los paréntesis "(" y ")" se eliminan
        result <- gsub(" ", "_", result)
        result <- gsub("\\(", "", result)
        result <- gsub(")", "", result)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Agrega los datos de una señal a un dataframe que contiene señales que se han obtenido del mismo archivo o
tabla.
# Todas las señales del dataframe tendrán la misma frecuencia de muestreo.
# Parámetros:
#  - dfResults: dataframe donde se va a agregar los datos de la señal
#  - signalName: nombre de la señal que se va a agregar
#  - signalData: datos de la señal que se va a agregar
# Resultado: nombre de la señal con las modificaciones
# ------------------------------------------------------------------------------------------------------------

addSignalDataToFinalResults <- function(dfResults, signalName, signalData) {

        # Agregamos la columna con los datos
        if(is.null(dfResults)) {

                # El dataframe está vacío.
                # Agregamos los datos y dejamos que R ponga un nombre predeterminado a la nueva columna
                dfResults = data.frame(signalData)

                # Reemplazamos el nombre de la única columna por el nombre de la señal
                names(dfResults) <- signalName
        }
        else {

                # Ya había columnas en el dataset.
```

```
                # Se agrega la nueva columna y los datos
                dfResults[,signalName] <- signalData
        }

        return(dfResults)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los valores físicos para los datos de una variable de resultados (Ingesys, CompaqDAC o tabla de
base de datos)
# Parámetros:
# - resultVariableData: vector con los datos de una variable de resultados (Ingesys, CompaqDAC o tabla de
base de datos)
# - signalCoefficients1: cadena de caracteres con el formato "[Coeficiente1;Coeficiente2]"
# - signalCoefficients2: cadena de caracteres con el formato "[Coeficiente1;Coeficiente2]"
# Los coeficientes permitirán pasar los datos de la variable a valores físicos.
# Se aplicarán al valor de la variable de la misma forma que se hace en la función polyval de Matlab.
# Primero se aplicará SignalCoefficients1 y después SignalCoefficients2.
# Resultado: vector de valores numéricos con los datos transformados
# ------------------------------------------------------------------------------------------------------------

getSignalData <- function(resultVariableData, signalCoefficients1, signalCoefficients2) {

        # Se aplica el primer coeficiente al dato
        signalCoefficientItems = getSignalCoefficientItems(signalCoefficients1)
        result     =     sapply(resultVariableData,     applySignalCoefficients,     coefficientItem1     =
signalCoefficientItems[1], coefficientItem2 = signalCoefficientItems[2])

        # Se aplica el segundo coeficiente al dato
        signalCoefficientItems = getSignalCoefficientItems(signalCoefficients2)
        result  =  sapply(result,  applySignalCoefficients,  coefficientItem1  =  signalCoefficientItems[1],
coefficientItem2 = signalCoefficientItems[2])

        return (result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene la fecha-hora a partir del nombre de un archivo de resultados de Ingesys
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#        Ejemplo de nombre:
#                    /campana0003/2018_12_11_09_03_35_REGISTRADORENSAYO[6].mat
# Resultado: fecha-hora del archivo de resultados de Ingesys
# ------------------------------------------------------------------------------------------------------------

getResultFileTimestamp <- function(resultFileName){

        # Ejemplo de nombre de archivo: "983_2020_01_07_12_23_44_REGISTRADORENSAYO[0].mat"
        data <- strsplit(resultFileName, "_")[[1]]
        year <- data[2]
        month <- data[3]
        day <- data[4]
        hour <- data[5]
        minute <- data[6]
        second <- data[7]

        result <- paste0(year, "_", month, "_", day, "_", hour, "_", minute, "_", second)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene un vector de timestamps relativos de una longitud determinada, para un punto de sincronización y
un periodo de tiempo.
# Parámetros:
# - numberElements: número de elementos del vector de timestamp que hay que generar.
# - samplingPeriod: periodo de muestreo para generar el timestamp
# - position: posición del vector cuyo tiempo relativo vale 0
# Resultado: vector con la información de timestamps relativos
# ------------------------------------------------------------------------------------------------------------

getTimestampDataFromSamplingPeriodAndPosition <- function(numberElements, samplingPeriod, position) {

        if(position < 1 || position > numberElements) {

                # Esto no debería ocurrir, pero para evitar errores se toma como tiempo cero la primera
posición
                print("!!!!! Error en la posición de sincronización")
                position <- 1
        }

        # Los valores posteriores al punto (incluido el punto) tendrán los siguientes timestamps:
        #               0, 1*T, 2*T, 3*T, ....
        numberPositiveElementsIncludingZero <- numberElements - position + 1
```

```
        positiveAndZeroTimestampVector <- seq(from = 0, by = samplingPeriod, length.out =
numberPositiveElementsIncludingZero)

        # Los valores anteriores al punto tendrán los siguientes timestamps:
        #                 ..., -3*T, -2*T, -1*T [, 0]
        numberNegativeElements <- position - 1
        start = -1 * numberNegativeElements * samplingPeriod
        negativeTimestampVector <- seq(from = start, by = samplingPeriod, length.out = numberNegativeElements)

        # Unimos los valores negativos y positivos en un único vector
        result <- c(negativeTimestampVector, positiveAndZeroTimestampVector)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene la información de timestamp para un grupo de muestreo.
# Parámetros:
# - sincronizationSignal: señal de sincronización que se va a evaluar para encontrar la transición que nos
indique el origen de tiempo
# - samplingPeriod: periodo de muestreo para generar el timestamp
# - threshold: valor que nos indicará el umbral de la transición que nos indique el origen de tiempo
# Resultado: vector con la información de timestamp para un grupo de muestreo
# ------------------------------------------------------------------------------------------------------------

getSamplingGroupTimestampData <- function(sincronizationSignal, samplingPeriod, threshold) {

        # Vector con los timestamps que se van a generar
        result <- c()

        # Número de elementos del vector que hay que generar
        numberElements <- length(sincronizationSignal)

        # Comprobamos si hay elementos por encima del valor umbral
        numberElementsAboveThreshold <- sum(sincronizationSignal > threshold)

        if(numberElementsAboveThreshold > 0) {

                # En algún punto de la señal se supera el umbral

                # Buscamos en la señal de sincronización la posición del primer valor que supere el umbral.

                position <- min(which(sincronizationSignal > threshold))

                # Se obtiene un vector de timestamps relativos de una longitud determinada, para el punto de
sincronización y el periodo de tiempo
                result <- getTimestampDataFromSamplingPeriodAndPosition(numberElements, samplingPeriod,
position)
        }
        else {
                # En ningún punto de la señal se supera el umbral
                # Se genera el vector de tiempos desde el inicio
                result <- seq(from = 0, by = samplingPeriod, length.out = numberElements)
        }

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Procesa un archivo de resultados (Ingesys o CompaqDAC)
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#        Ejemplo de nombre:
#                        /campana0003/2018_12_11_09_03_35_REGISTRADORENSAYO[6].mat
# - resultFileVariables: vector con los nombres de las variables de un archivo de resultados de Ingesys
# - resultFileData: lista con el contenido total de un archivo de resultados (cada elemento de la lista:
variable = matriz)
# - samplingGroups: lista de grupos de muestreos del archivo de resultados
# - variablesNotToProcess: vector con variables (o sufijos de variables) que no hay que procesar (de las que
se han pasado en resultFileVariables)
# Resultado: lista con dataframes de señales físicas
# ------------------------------------------------------------------------------------------------------------

processResultFileBySamplingGroups <- function(resultFileName,

resultFileVariables,

resultFileData,

samplingGroups,

resultFileVariablesNotToProcess) {
```

```
        # Se va a crear una lista de dataframes, donde cada dataframe guarda los valores físicos de las
variables que tienen el mismo muestreo.
        # En la siguiente instrucción, a pesar de que pone "vector", se genera una lista.
        samplingGroupsDataframeList <- vector(mode = "list", length = length(samplingGroups))

        # Bucle de procesamiento de variables
        for(v in 1:length(resultFileVariables)) {

                resultFileVariable <- resultFileVariables[v]

                # Comprobamos que el nombre de la variable no termine en alguno de los que se han pasado por
parámetro a la función.
                # Si es así, no procesamos la variable.
                processVariable <- TRUE
                for (ev in 1:length(resultFileVariablesNotToProcess)) {

                        if(checkStringEndingInResultVariables(resultFileVariable,
resultFileVariablesNotToProcess[ev]) == TRUE) {

                                # No hay que procesar la variable
                                processVariable <- FALSE
                                break
                        }
                }

                # Procesamos la variable a no ser que sea una de las que no haya que considerar
                if(processVariable == TRUE) {

                        # Se obtiene el índice de grupo de muestreo al que pertenece la variable.

                        samplingGroupIndex  <-  getResultFileSamplingGroupIndexForVariable(samplingGroups,
resultFileData, resultFileVariable)

                        if(!is.null(samplingGroupIndex)) {

                                #  Obtenemos los metadatos de la variable (nombre y coeficientes de
conversión)
                                signalMetadata <- getSignalMetadata(resultFileVariable)
                                signalName <- as.character(signalMetadata$SignalName)

                                signalCoefficients1  <-  as.character(signalMetadata$SignalCoefficients1)

                                signalCoefficients2  <-  as.character(signalMetadata$SignalCoefficients2)


                                # Obtenemos los datos físicos de la señal correspondiente a la variable.
                                # Hay que pasar un vector con los datos de la variable
                                signalData <- getSignalData(resultFileData[[resultFileVariable]],


signalCoefficients1,

        signalCoefficients2)

                                # Agregamos los datos de la señal como una nueva columna del dataframe que
contiene los datos de su grupo de muestreo
                                dfResults <- samplingGroupsDataframeList[[samplingGroupIndex]]

                                dfResults <- addSignalDataToFinalResults(dfResults,

                 modifySignalNameSpecialCharacters(signalName),

                 signalData)

                                samplingGroupsDataframeList[[samplingGroupIndex]] <- dfResults
                        }
                }
        }

        # Bucle para agregar una columna extra a cada dataframe de grupo de muestreo con los timestamps de
los datos.
        for(sg in 1:length(samplingGroups)) {

                # Se obtiene el grupo de muestreo
        samplingGroup <- samplingGroups[[sg]]
        numberOfPoints <- samplingGroup$NumberOfPoints
        samplingPeriod <- samplingGroup$SamplingPeriod
                pointsAtTimeOrigin <- samplingGroup$PointsAtTimeOrigin

                # Se genera el vector de timestamps relativos
                if (is.null(pointsAtTimeOrigin) == FALSE) {

                        # Es un archivo de Ingesys

                        # La posición del punto de sincronización en el archivo de matlab está referenciado
respecto a 0.
```

```
                        # Sumamos una unidad a este valor, ya que las posiciones en los vectores de R están
referenciadas respecto a 1.
                        position <- pointsAtTimeOrigin + 1

                        # Se obtiene la señal de timestamp a partir del punto de sincronización.
                        #  Los valores posteriores al punto (incluido el punto) tendrán los siguientes
timestamps:
                        #               0, 1*T, 2*T, 3*T, ....
                        # Los valores anteriores al punto tendrán los siguientes timestamps:
                        #               ..., -3*T, -2*T, -1*T [, 0]
                        # Esta operación hará que las señales del Ingesys se sincronicen entre sí.
                        #  Faltará sincronizar las señales con respecto a CompaqDac, lo cual se hace más
adelante
                        timestampData   <-   getTimestampDataFromSamplingPeriodAndPosition(numberOfPoints,
samplingPeriod, position)
                }
                else {
                        # Es un archivo de CompactDAC.

                        #  La señal de sincronización en la señal física es aquella cuyo nombre es
"Synchronization_signal_from_Ingesys".
                        # (Atención: este nombre no es el original de la base de datos, ya que los espacios
en blanco se han sustituido por '_')
                        dfResults <- samplingGroupsDataframeList[[sg]]
                        sincronizationSignal <- dfResults[,"Synchronization_signal_from_Ingesys"]

                        # Se obtiene la señal de timestamp a partir de la señal de sincronización.
                        # La posición cuyo timestamp vale 0 es aquella en donde la señal de sincronización
supera un valor umbral.
                        #  Los valores posteriores al punto (incluido el punto) tendrán los siguientes
timestamps:
                        #               0, 1*T, 2*T, 3*T, ....
                        # Los valores anteriores al punto tendrán los siguientes timestamps:
                        #               ..., -3*T, -2*T, -1*T [, 0]
                        threshold <- 2.5
                        timestampData        <-        getSamplingGroupTimestampData(sincronizationSignal,
samplingPeriod, threshold)
                }

                # Agregamos la columna con los timestamps relativos        al   dataframe   del   grupo   de
muestreo.
                # Buscamos el dataframe cuyo número de datos coincida con el valor de elementos del grupo de
muestreo
                for(d in 1:length(samplingGroupsDataframeList)) {
                        if(nrow(samplingGroupsDataframeList[[d]]) == numberOfPoints) {
                                samplingGroupsDataframeList[[d]]                                        <-
addSignalDataToFinalResults(samplingGroupsDataframeList[[d]], "Timestamp", timestampData)
                        }
                }
        }

        return(samplingGroupsDataframeList)
}

# ------------------------------------------------------------------------------------------------------------
#   Desplaza   los   timestamps   de   los   grupos   de   muestreo   de   Ingesys   por   medio   de   la   señal
"Synchronization_signal_for_NI".
# Pasos:
#       * Obtener los datos de la señal "Synchronization_signal_for_NI".
#       * Buscar en la señal de sincronización la posición del primer valor que supere un umbral (2.5)
#       * Obtener el valor de timestamp correspondiente a esa posición
#       * Restamos el valor de timestamp anterior de los datos de todos los vectores de timestamp de los
grupos de sicronización
# Parámetros:
# - samplingGroupsDataframeList: lista con los dataframes de datos físicos de un archivo de Ingesys
# Resultado: lista con dataframes de señales físicas con el timestamp sincronizado con respecto a CompaqDAC
# ------------------------------------------------------------------------------------------------------------

synchroniseIngesysTimestampDataWithCompaqDac <- function(samplingGroupsDataframeList) {

        timestampOffsetWithRespectToCompacDac <- 0
        threshold <- 2.5

        # Bucle para encontrar la señal "Synchronization_signal_for_NI".
        #   El   objetivo   del   bucle   es   encontrar   el   timestamp   de   sincronización   (valor   de
timestampOffsetWithRespectToCompacDac)
        for(sg in 1:length(samplingGroupsDataframeList)) {

                # Se obtienen los datos de un dataframe de resultados
                dfSamplingGroupResults <- samplingGroupsDataframeList[[sg]]

                # La señal "Synchronization_signal_for_NI" está en el dataframe que tiene 2000 datos
                if(nrow(dfSamplingGroupResults) == 2000) {

                        # Obtenemos los datos de la señal "Synchronization_signal_for_NI" y el vector de
timestamp de su grupo de muestreo
```

```
                        sincronizationSignal <- dfSamplingGroupResults[,"Synchronization_signal_for_NI"]
                        timestampData <- dfSamplingGroupResults[,"Timestamp"]

                        # Comprobamos si hay elementos por encima de un valor umbral

                        numberElementsAboveThreshold <- sum(sincronizationSignal > threshold)
                        if(numberElementsAboveThreshold > 0) {

                                # Buscamos en la señal de sincronización la posición del primer valor que
supere el umbral.
                                position <- min(which(sincronizationSignal > threshold))

                                # Obtenemos el valor de timestamp correspondiente a esa posición en el
vector de timestamp del grupo
                                # Este valor se tiene que restar en todos los timestamps de los grupos de
sincronización
                                timestampOffsetWithRespectToCompacDac <- timestampData[position]
                        }

                        # Salimos de bucle
                        break
                }
        }

        # Sincronizamos las señales de Ingesys con respecto a las de CompacDac.
        # Para esto, restamos el timestamp obtenido antes de los diferentes grupos de sicronización
        if (timestampOffsetWithRespectToCompacDac != 0) {

                for(sg in 1:length(samplingGroupsDataframeList)) {

                        # Se obtiene el vector de timestamp del dataframe del grupo de muestreo
                        timestampData <- samplingGroupsDataframeList[[sg]][,"Timestamp"]

                        # Se restan todos los valores con el valor de timestamp de sincronización

                        timestampData <- timestampData - timestampOffsetWithRespectToCompacDac

                        # Guardamos los cambios del timestamp en el grupo de muestreo
                        samplingGroupsDataframeList[[sg]][,"Timestamp"] <- timestampData
                }
        }

        return(samplingGroupsDataframeList)
}

# ------------------------------------------------------------------------------------------------------------
# Procesa un archivo de resultados de Ingesys
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#         Ejemplo de nombre:
#                         /campana0003/2018_12_11_09_03_35_REGISTRADORENSAYO[6].mat
# - resultFileVariables: vector con los nombres de las variables de un archivo de resultados de Ingesys
# - resultFileData: lista con el contenido total de un archivo de resultados (cada elemento de la lista:
variable = matriz)
# Resultado: lista con dataframes de señales físicas
# ------------------------------------------------------------------------------------------------------------

processIngesysResultFile <- function(resultFileName, resultFileVariables, resultFileData) {

        # Se obtienen los distintos grupos de muestreo del archivo de Ingesys
        samplingGroups <- getIngesysResultFileSamplingGroups(resultFileVariables, resultFileData)

        # Creamos un vector con el nombre de las variables que no se van a procesar en un archivo de Ingesys.
        # Ponemos el carácter '$' al final para indicar que es la parte final de nombre variable
        resultFileVariablesNotToProcess <- c(".us.Ts.ms$", ".us.n$", ".us.nPRE$")

        # Procesamos el archivo de resultados
        result <- processResultFileBySamplingGroups(resultFileName,

resultFileVariables,

        resultFileData,

        samplingGroups,

        resultFileVariablesNotToProcess)

        # Queda desplazar los timestamps de los grupos de muestreo por medio de la señal
"Synchronization_signal_for_NI".
        result <- synchroniseIngesysTimestampDataWithCompaqDac(result)

        return(result)
}
```

```
# ------------------------------------------------------------------------------------------------------------
# Procesa un archivo de resultados de CompaqDAC
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#         Ejemplo de nombre:
#                         1000_2020_01_07_13_31_45_902_cDaqCapture.mat
# - resultFileVariables: vector con los nombres de las variables de un archivo de resultados de Ingesys
# - resultFileData: lista con el contenido total de un archivo de resultados (cada elemento de la lista:
variable = matriz)
# Resultado: señales físicas del archivo
# ------------------------------------------------------------------------------------------------------------

processCompactDacResultFile <- function(resultFileName, resultFileVariables, resultFileData) {

        # Se obtiene el único grupo de muestreo del archivo de CompaqDAC
        samplingGroups <- getCompaqDacResultFileSamplingGroups(resultFileData)

        # Creamos un vector con el nombre de las variables que no se van a procesar en un archivo de CompaqDAC.

        resultFileVariablesNotToProcess <- c("description", "time")

        # Procesamos el archivo de resultados
        result <- processResultFileBySamplingGroups(resultFileName,

resultFileVariables,

        resultFileData,

        samplingGroups,

        resultFileVariablesNotToProcess)
        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el contenedor de Azure donde se almacenan los archivos con datos físicos
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - azureContainerName: nombre del contenedor de Azure
# Resultado: contenedor de Azure
# ------------------------------------------------------------------------------------------------------------

getAzureContainer <- function(testCampaignId, testTypeId, azureContainerName) {

        # Url a la carpeta donde están los archivos
        blobStorageContainerUrl <-paste0("https://stoisselubfzg.blob.core.windows.net/", azureContainerName
,"/")

        # Clave del blob storage
        blobStorageKey                                                                                  <-
"9FGKaoD/BAsrz6Vmfx74oxD/jZBVIfVg8S1wSPJwqJ3H2tgg6A1bk5fWUAolowSR32a75IgZ9lTBvg0exrwSlw=="

        # Obtenemos el contenedor
        blobStorageContainer <- blob_container(blobStorageContainerUrl, key=blobStorageKey)

        return(blobStorageContainer)
}

# ------------------------------------------------------------------------------------------------------------
# Procesa un archivo de resultados (Ingesys o CompaqDAC)
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#         Ejemplo de nombre:
#                         /campana0003/1000_2020_01_07_13_31_45_902_cDaqCapture.mat
# Resultado: señales físicas correspondientes al resultado
# ------------------------------------------------------------------------------------------------------------

processFileResult <- function(resultFileName) {

        result <- list()

        # Obtención del contenedor del blob storage de Azure donde están almacenados los archivos
        blobStorageContainer <- getAzureContainer(testCampaignId, testTypeId, "archive")

        # Lectura de los datos del archivo de resultados
        tryCatch(
        {
                        # Descargamos el archivo de Azure
                        print(paste0("Downloading file: ", resultFileName))
                        resultFile <- storage_download(blobStorageContainer, resultFileName, dest=NULL)
```

```
                        # Obtenemos los datos del archivo en una lista
                        print(paste0("Reading file: ", resultFileName))
                        resultFileData <- readMat(resultFile)
        },
        error=function(cond) {

                        # Se ha producido un error
                        print(paste0("!!!!!!", "Problems when downloading or reading file: ",
resultFileName))
            return(NULL)
        },
        finally={}
    )

        # Obtención de los nombres de las variables de resultados.
        resultFileVariables <- names(resultFileData)

        # Comprobamos si es un archivo de Ingesys.
        # Los archivos de Ingesys tienen varibles que acaban en "_us_Ts_ms$"
        print(paste0("Processing file: ", resultFileName))
        isIngesysMatFile = checkStringEndingInResultVariables(resultFileVariables, ".us.Ts.ms$")
        if (isIngesysMatFile) {

                # Se procesa el archivo de Ingesys
                result <- processIngesysResultFile(resultFileName, resultFileVariables, resultFileData)
        }
        else
        {
                isCompaqDacMatFile = checkStringEndingInResultVariables(resultFileVariables, "time$")
                if(isCompaqDacMatFile) {

                        # Se procesa el archivo de CompactDAC
                        result <- processCompactDacResultFile(resultFileName, resultFileVariables,
resultFileData)
                }
        }

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Lee los datos de una tabla de resultados correspondientes a un identificador de resultados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - resultDbTableName: nombre de la tabla donde están los resultados
#  - resultId: identificador del resultado
# Resultado: dataset con los campos de los datos de resultado
# ------------------------------------------------------------------------------------------------------------

getResulDbTableData <- function(testCampaignId, resultDbTableName, resultId) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT * ",
                                        "FROM ", getTestCampaignName(testCampaignId), ".",
resultDbTableName, " ",
                                        "WHERE IdResultado = ", as.character(resultId))

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        return(dfQueryResult)
}

# ------------------------------------------------------------------------------------------------------------
# Procesa los datos de un resultado que está guardado en una tabla de la base de datos
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - resultDbTableName: nombre de la tabla de resultados
#  - resultId: identificador del resultado en la base de datos
# Resultado: lista con un dataframe que contiene las señales físicas.
# Se genera una lista para que el procesamiento posterior del dataframe sea igual que en el del caso de
Ingesys y CompaqDAC
# (en Ingesys hay varios grupos de muestreos, cada uno de los cuales se guarda en un dataframe distinto)
# ------------------------------------------------------------------------------------------------------------

processDbTableResult <- function(testCampaignId, resultDbTableName, resultId) {

        dfResults <- NULL

        # Obtenemos los datos de la tabla
        print(paste0("Reading result ", resultId, " from table ", resultDbTableName))
```

```
        dfDbTableData <- getResulDbTableData(testCampaignId, resultDbTableName, resultId)

        # Nombres de variables de la tabla
        resultVariableNames <- names(dfDbTableData)

        # Bucle para agregar las señales
        print(paste0("Processing result ", resultId, " from table ", "resultDbTableName"))
        for(v in 1:length(resultVariableNames)) {

                resultVariable = resultVariableNames[v]

                if(checkStringEndingInResultVariables(resultVariable, "Id$") == FALSE &&
                                checkStringEndingInResultVariables(resultVariable,   "IdResultado$")    ==
FALSE &&
                                        checkStringEndingInResultVariables(resultVariable,  "Fecha$")   ==
FALSE) {

                        # Para cada variable, hay que obtener los siguientes datos:
                        # * Nombre de la señal
                        # * Datos de la variable
                        # * Datos de timestamp

                        signalMetadata <- getSignalMetadata(resultVariable)

                        signalName <- as.character(signalMetadata$SignalName)
                        signalCoefficients1 <- as.character(signalMetadata$SignalCoefficients1)

                        signalCoefficients2 <- as.character(signalMetadata$SignalCoefficients2)

                        # Obtenemos los datos físicos de la señal correspondiente a la variable.
                        # Hay que pasar un vector con los datos de la variable

                        signalData <- getSignalData(dfDbTableData[,v],
                                                                                signalCoefficients1,
                                                                                signalCoefficients2)

                        # Agregamos la columna con los datos
                        dfResults = addSignalDataToFinalResults(dfResults,

        modifySignalNameSpecialCharacters(signalName),

        signalData)
                }
        }

        # Bucle para agregar las fechas-horas de los datos
        for(v in 1:length(resultVariableNames)) {

                resultVariable = resultVariableNames[v]

                if(checkStringEndingInResultVariables(resultVariable, "Fecha$") == TRUE) {

                        # Agregamos la columna con las fechas-horas
                        dfResults       =       addSignalDataToFinalResults(dfResults,       "Timestamp",
as.character(dfDbTableData[,v]))
                }
        }

        # Se genera una lista para que el procesamiento posterior del dataframe sea igual que en el del caso
de Ingesys y CompaqDAC
        return(list(dfResults))
}

# ------------------------------------------------------------------------------------------------------------
# Carga los datos físicos de un resultado a un archivo CSV en un contenedor de Azure
# Parámetros:
#  - resultData: Lista con datos del resultado con las señales físicas y los timestamp. La lista puede tener
uno o más dataframes.
#  - resultFileName: nombre del archivo de resultados
#  - tempFolder: carpeta temporal local para crear los archivos con los datos físicos
# Resultado: serialización del resultado (cada uno de los posibles dataframes) en un contenedor de Azure
# ------------------------------------------------------------------------------------------------------------

createResultPhysicalFileAndUploadOnAzure <- function(resultData, resultFileName, tempFolder) {

        # Si no existe el directorio temporal, se crea
        dir.create(tempFolder, showWarnings = FALSE)

        # Comprobaciones en los datos
        if(length(resultData) != 0) {

                # Bucle para serializar los dataframes del resultado
                for (df in 1:length(resultData)) {

                        print(paste0("Creating file: ", resultFileName))
```

```
                        # Obtenemos el dataframe que se va a serializar a csv
                        dfResult <- resultData[[df]]

                        # Obtenemos el número de filas del dataframe y las indicamos al final del nombre del
archivo
                        resultFileNameEnding = paste0("_n", nrow(dfResult), ".csv")
                        resultFileNameWithNumberOfRows = sub('.csv$', resultFileNameEnding, resultFileName)

                        # Creamos el archivo en la ubicación temporal
                        completeFileName <- file.path(tempFolder, resultFileNameWithNumberOfRows)

                        # Creamos las posibles subcarpetas del archivo
                        fileFolder <- dirname(completeFileName)
                        dir.create(fileFolder, recursive = TRUE)

                        # Creamos el archivo en la ubicación temporal
                        write.csv(dfResult, completeFileName, row.names = FALSE)

                        # Subimos el archivo a Azure

                        print(paste0("Uploading file on Azure: ", resultFileName))

                        if(ubuntuVirtualMachine == TRUE) {

                                cmd = paste0("/home/adminssh/isselub/azcopy copy \"",
                                                                tempFolder, "/*\" ",

"\"https://stoisselubfzg.blob.core.windows.net/results?sv=2019-02-02&ss=b&srt=sco&sp=rwdlac&se=2040-04-
27T19:17:18Z&st=2020-04-27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\"     --
recursive")
                                system(cmd)
                        }
                        else {

                                cmd = paste0("azcopy copy \"",
                                                                tempFolder, "/*\" ",

"\"https://stoisselubfzg.blob.core.windows.net/results?sv=2019-02-02&ss=b&srt=sco&sp=rwdlac&se=2040-04-
27T19:17:18Z&st=2020-04-27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\"     --
recursive")
                                system("cmd.exe", input = cmd)
                        }

                        print(paste0("File uploaded: ", resultFileName))

                        # Borramos la carpeta temporal y todo su contenido
                        unlink(tempFolder, recursive = TRUE)
                }
        }
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene la fecha y número de ciclos de una tabla de resultados correspondientes a un identificador de
resultados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - resultDbTableName: nombre de la tabla donde están los resultados
#  - resultId: identificador del resultado
# Resultado: lista con la cadena de caracteres modificada para la fecha y el número de ciclos
# ------------------------------------------------------------------------------------------------------------

getResultDbTableTimestampAndCycleNumber <- function(testCampaignId, resultDbTableName, resultId) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT TOP 1 Fecha AS Timestamp, ",
                                        "                                positionProfileAbsolutePeriodCounter
As CycleNumber ",
                                        "FROM      ",      getTestCampaignName(testCampaignId),      ".",
resultDbTableName, " ",
                                        "WHERE IdResultado = ", as.character(resultId), " ",
                                        "ORDER BY Id ASC")

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Obtenemos la fecha y el número de ciclos.
        # Para la fecha se hacen los siguientes cambios:
        # - Se sustituye el espacio en blanco por un guión bajo
        # - Se sustituye los dos puntos por un guión bajo
        # - Se sustituyen los guiones normales por guiones bajos
        # Ejemplo: "2020-01-07 11:15:53" -> "2020-01-07_11-15-53"
```

```
        timestamp <- as.character(dfQueryResult[1,"Timestamp"])
        timestamp <- gsub(" ", "_", timestamp)
        timestamp <- gsub(":", "_", timestamp)
        timestamp <- gsub("-", "_", timestamp)
        cycleNumber = as.integer(dfQueryResult[1,"CycleNumber"])

        result = list(Timestamp = timestamp, CycleNumber = cycleNumber)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el número de ciclos de un archivo de resultados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - resultFileName: nombre del archivo de resultados
#  - resultId: identificador del resultado
# Resultado: Número de ciclos (entero)
# ------------------------------------------------------------------------------------------------------------

getResultFileCycleNumber <- function(testCampaignId, resultFileName, resultId) {

        # Obtenemos la fecha del nombre del archivo
        timestamp = getResultFileTimestamp(resultFileName)

        # Ejemplo de formato que se ha obtenido con la consulta anterior: "2020_01_07_11_16_30"
        # A partir de la cadena anterior, hay que obtener lo siguiente: "2020-01-07 11:16:30"
        data <- strsplit(timestamp, "_")[[1]]
        year <- data[1]
        month <- data[2]
        day <- data[3]
        hour <- data[4]
        minute <- data[5]
        second <- data[6]
        timestamp <- paste0(year, "-", month, "-", day, " ", hour, ":", minute, ":", second)

        # Obtenemos la tabla de la base de datos donde se va a obtener la información del número de ciclos
        resultDbTableName <- getResultFileCycleNumberDbTable(resultId, resultFileName)

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT TOP 1 positionProfileAbsolutePeriodCounter As CycleNumber ",
                                        "FROM      ",      getTestCampaignName(testCampaignId),      ".",
resultDbTableName, " ",
                                        "WHERE Fecha <= '", timestamp, "' ",
                                        "ORDER BY Fecha DESC")

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Obtenemos el número de ciclos
        result = as.integer(dfQueryResult[1,"CycleNumber"])

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene la tabla de base de datos de donde se obtendrá la información de número de ciclos para un archivo
de resultados
# Parámetros:
#  - resultId: identificador del resultado
#  - resultFileName: nombre del archivo de resultados
# Resultado: nombre de la tabla de base de datos (cadena de caracteres)
# ------------------------------------------------------------------------------------------------------------

getResultFileCycleNumberDbTable <- function(resultId, resultFileName) {

        # Obtenemos la fecha del nombre del archivo.
        timestamp = getResultFileTimestamp(resultFileName)

        # Ejemplo de formato que se ha obtenido con la consulta anterior: "2020_01_07_11_16_30"
        # A partir de la cadena anterior, hay que obtener lo siguiente: "2020-01-07 11:16:30"
        data <- strsplit(timestamp, "_")[[1]]
        year <- data[1]
        month <- data[2]
        day <- data[3]
        hour <- data[4]
        minute <- data[5]
        second <- data[6]
        timestamp <- paste0(year, "-", month, "-", day, " ", hour, ":", minute, ":", second)

        # Conexión a la base de datos
```

```
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT TOP 1 NombreTablaDeBaseDeDatos As CycleNumberResultDbTableName ",
                                    "FROM [isselub].[resultados] ",
                                    "WHERE Id <= ", resultId, " ",
                                    "               AND NombreTablaDeBaseDeDatos IS NOT NULL ",
                                    "               AND LEN(NombreTablaDeBaseDeDatos) > 0 ",
                                    "ORDER BY Id DESC")

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Obtenemos el número de ciclos
        result = as.character(dfQueryResult[1,"CycleNumberResultDbTableName"])

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el nombre para un archivo de datos físicos correspondientes a un resultado de tabla de base de datos
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - resultDbTableName: nombre de la tabla donde están los resultados
#  - resultId: identificador del resultado
#  - physicalDataFolderName: nombre de la carpeta donde se guardarán los datos físicos
# Resultado: señales físicas de todos los resultados correspondientes a una campaña y tipo de test
# ------------------------------------------------------------------------------------------------------------

getPhysicalFileNameForDbResult <- function(testCampaignId, testTypeId, resultDbTableName, resultId,
physicalDataFolderName) {

        # Obtenemos la fecha-hora y el número de ciclos
        timestampAndCycleNumber = getResultDbTableTimestampAndCycleNumber(testCampaignId, resultDbTableName,
resultId)
        timestamp = timestampAndCycleNumber$Timestamp
        cycleNumber = timestampAndCycleNumber$CycleNumber

        # Obtenemos el nombre del archivo final
        result <- paste0(getTestCampaignName(testCampaignId), "/",
                                    getTestTypeName(testTypeId), "/",
                                    physicalDataFolderName, "/",
                                    resultDbTableName, "/",
                                    resultId, "_",
                                    timestamp, "_",
                                    resultDbTableName, "_",
                                    "c", cycleNumber, "_", ".csv")

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el nombre para un archivo de datos físicos correspondientes a un archivo de resultados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - resultFileName: nombre del archivo de resultados.
#  - resultData: lista con los dataframes de los datos físicos.
#  - resultId: identificador del resultado
#  - physicalDataFolderName: nombre de la carpeta donde se guardarán los datos físicos
# Resultado: señales físicas de todos los resultados correspondientes a una campaña y tipo de test
# ------------------------------------------------------------------------------------------------------------

getPhysicalFileNameForResultFile <- function(testCampaignId,
                                                                                                testTypeId,

resultFileName,
                                                                                                resultData,
                                                                                                resultId,


physicalDataFolderName) {

        # Nombre del tipo de archivo ("Ingesys" o "CompaqDAC").
        # Si los datos tienen un sólo dataframe, es un resultado de CompaqDAC. Si tienen más, es de Ingesys
        resultFileTypeName <- ""

        if (length(resultData) > 1) {
                resultFileTypeName <- "Ingesys"
        }
        else if (length(resultData) == 1) {
                resultFileTypeName <- "CompaqDAC"
        }
```

```
        # Obtenemos el número de ciclos del archivo
        cycleNumber = getResultFileCycleNumber(testCampaignId, resultFileName, resultId)

        # Obtenemos la fecha-hora a partir del nombre del archivo de resultados
        # Ejemplo de nombre de archivo: "983_2020_01_07_12_23_44_REGISTRADORENSAYO[0].mat"
        timestamp = getResultFileTimestamp(resultFileName)

        # Obtenemos el nombre del archivo final
        result <- paste0(getTestCampaignName(testCampaignId), "/",
                                        getTestTypeName(testTypeId), "/",
                                        physicalDataFolderName, "/",
                                        resultFileTypeName, "/",
                                        resultId, "_",
                                        timestamp, "_",
                                        resultFileTypeName, "_",
                                        "c", cycleNumber, ".csv")

        return(result)
}

# ------------------------------------------------------------------------------------------------------------------
# Obtiene el texto de un tipo de ensayo
# Parámetros:
#  - testTypeId: tipo de test
# Resultado: texto correspondiente al tipo de ensayo
# ------------------------------------------------------------------------------------------------------------------

getTestTypeName <- function(testTypeId) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT Nombre as TestName ",
                                        "FROM [isselub].[tiposensayo] ",
                                        "WHERE Id = ", testTypeId)

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Modificaciones que hacemos en el nombre:
        # - Se eliminan espacios en blanco al comienzo y al final
        # - Se sustituye los espacios en blanco por un guión bajo
        # - Se pone en mayúscula la primera letra
        result = as.character(dfQueryResult[1,1])
        result <- trimws(result)
        result <- gsub(" ", "_", result)
        result <- paste(toupper(substring(result, 1,1)), substring(result, 2), sep="", collapse=" ")

        return(result)
}

# ------------------------------------------------------------------------------------------------------------------
# Obtiene los datos físicos correspondientes a una campaña y tipo de test
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - tempFolder: carpeta temporal para generar los archivos con datos físicos antes de subirlos a Azure
# - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: señales físicas de todos los resultados correspondientes a una campaña y tipo de test
# ------------------------------------------------------------------------------------------------------------------

processResults <- function(testCampaignId, testTypeId, tempFolder, processNewResultsOnly) {

        # Nombre de la carpeta donde se guardarán los datos físicos
        physicalDataFolderName <- "PhysicalData"

        # Obtenemos los resultados correspondientes a una campaña y tipo de test
        dfResults <- getResults(testCampaignId, testTypeId, processNewResultsOnly)
        numberOfResults <- nrow(dfResults)

        # Comprobamos si hay resultados
        if(is.na(numberOfResults) || numberOfResults == 0) {
                # No hay resultados para procesar
                return()
        }

        # Procesamiento de los resultados
        for(r in 1:numberOfResults) {

                # Obtenemos la información inicial del resultado
                resultId <- as.integer(dfResults[r,"Id"])
                resultFileName <- as.character(dfResults[r, "NombreFichero"])
```

```
                resultDbTableName <- as.character(dfResults[r, "NombreTablaDeBaseDeDatos"])

                # En función del tipo de resultado realizamos el procesamiento adecuado
                if(!is.null(resultFileName) && !is.na(resultFileName) && nchar(resultFileName) > 0) {

                        # Es un archivo de Ingesys o CompaqDAC

                        # Procesamos el resultado
                        resultData <- processFileResult(resultFileName)

                        if(!is.null(resultData) == TRUE) {

                                # Nombre del resultado
                                resultFileName <- getPhysicalFileNameForResultFile(testCampaignId,

                                   testTypeId,

                                   resultFileName,

                                   resultData,

                                   resultId,

                                   physicalDataFolderName)
                                # Serialización de los datos
                                createResultPhysicalFileAndUploadOnAzure(resultData,       resultFileName,
tempFolder)
                        }
                }
                else      if      (!is.null(resultDbTableName)      &&      !is.na(resultDbTableName)      &&
nchar(resultDbTableName) > 0) {

                        # Es un resultado que está en una tabla de la base de datos

                        # Ponemos la primera letra de la tabla en mayúscula
                        resultDbTableName     <-     paste(toupper(substring(resultDbTableName,     1,1)),
substring(resultDbTableName, 2), sep="", collapse=" ")

                        # Procesamos el resultado
                        resultData  <-  processDbTableResult(testCampaignId,  resultDbTableName,  resultId)


                        # Nombre del resultado
                        resultFileName <- getPhysicalFileNameForDbResult(testCampaignId,

                         testTypeId,

                         resultDbTableName,

                         resultId,

                         physicalDataFolderName)
                        # Serialización de los datos
                        createResultPhysicalFileAndUploadOnAzure(resultData,  resultFileName,  tempFolder)

                }
        }
}

# ------------------------------------------------------------------------------------
# Genera los archivos físicos
# ------------------------------------------------------------------------------------

processData <- function() {

        testCampaignId <- 3
        testTypeId <- 3
        processNewResultsOnly <- TRUE

        if(ubuntuVirtualMachine == FALSE) {

                tempFolder <- "D:/Users/rgonzalez/Documents/Datos/temp/TFM/Archivos/temp"
        }
        else {

                tempFolder <- "/home/adminssh/isselub/temp"
        }

        processResults(testCampaignId, testTypeId, tempFolder, processNewResultsOnly)
        print("Preprocesing finished")
}

processData()
```

9.3 Feature Extraction Code

Features are extracted by executing function “processPhysicalFileResults”. The call to this function can be found at the end of this script.

```
 # Instalación de paquetes
# install.packages("RODBC")
# install.packages("AzureStor")
# install.packages('moments')

# Carga de paquetes en el entorno de R
library(RODBC)
library(AzureStor)
library(moments)

# Variable que indica si estamos ejecutando el script en Azure de forma automatizada
ubuntuVirtualMachine <- FALSE

# ------------------------------------------------------------------------------------------------------------
# Obtiene la fecha-hora a partir del nombre de un archivo de resultados de Ingesys.
# Formato: "yyy-MM-dd hh:mm:ss"
# Parámetros:
# - resultFileName: nombre del archivo de resultados.
#     Es el nombre completo de su blob, que está en el contenedor "archive" del blob storage de Azure de
Isselub.
#         Ejemplo de nombre:
#                         /campana0003/2018_12_11_09_03_35_REGISTRADORENSAYO[6].mat
# Resultado: fecha-hora del archivo de resultados de Ingesys
# ------------------------------------------------------------------------------------------------------------

getDateTimeForFeature <- function(resultFileName){

        #               Ejemplo               de               nombre               de               archivo:
"/campana0003/FatigueTest/PhysicalData/Ingesys/196_2018_12_11_08_03_31_Ingesys_c18961_n2000.csv"
        data <- strsplit(resultFileName, "_")[[1]]
        year <- data[2]
        month <- data[3]
        day <- data[4]
        hour <- data[5]
        minute <- data[6]
        second <- data[7]

        result <- paste0(year, "-", month, "-", day, " ", hour, ":", minute, ":", second)

        return(result)
}

# Función que obtiene una conexión a la base de datos de Azure
getAzureSQLDatabaseServerConnection <- function() {

        # Base de datos SQL Database en Azure
        AzureSQLDatabaseServer <- "sqlisselub.database.windows.net"
        AzureSQLDatabaseUser <- "adminsql"
        AzureSQLDatabasePassword <- "Tekniker2016@"
        AzureSQLDatabase <- "configuration"
        if(ubuntuVirtualMachine == TRUE) {
                # Driver ODBC para la virtualización de Ubuntu
                AzureSQLDatabaseDriver <- "ODBC Driver 17 for SQL Server"
        }
        else {
                # Driver ODBC para la virtualización de Windows
                AzureSQLDatabaseDriver <- "SQL Server"
        }
        AzureSQLDatabaseConnectionString <- paste0(
                "Driver=", AzureSQLDatabaseDriver,
                ";Server=", AzureSQLDatabaseServer,
                ";Database=", AzureSQLDatabase,
                ";Uid=", AzureSQLDatabaseUser,
                ";Pwd=", AzureSQLDatabasePassword)

        # Conexión con la base de datos
        result <- odbcDriverConnect(AzureSQLDatabaseConnectionString)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003".
# Deben generarse cuatro dígitos para el número de la campaña.
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
# Resultado: cadena de caracteres con el nombre de la campaña de tests
```

```
# Se genera siempre cuatro dígitos para el número de la campaña (se completa con ceros).
# Por ejemplo, para la campaña 3 se crea la siguiente cadena: "campana0003"
# ------------------------------------------------------------------------------------------------------------

getTestCampaignName <- function(testCampaignId) {

        # Creamos una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003"
        # El número de dígitos en la cadenaserá siempre de cuatro
        campaignNumberString = as.character(testCampaignId)
        campaignNumberString = paste("campana", formatC(testCampaignId, width=4, flag="0"), sep="")

        return(campaignNumberString)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el texto de un tipo de ensayo
# Parámetros:
#  - testTypeId: tipo de test
# Resultado: texto correspondiente al tipo de ensayo
# ------------------------------------------------------------------------------------------------------------

getTestTypeName <- function(testTypeId) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT Nombre as TestName ",
                                    "FROM [isselub].[tiposensayo] ",
                                    "WHERE Id = ", testTypeId)

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Modificaciones que hacemos en el nombre:
        # - Se eliminan espacios en blanco al comienzo y al final
        # - Se sustituye los espacios en blanco por un guión bajo
        # - Se pone en mayúscula la primera letra
        result = as.character(dfQueryResult[1,1])
        result <- trimws(result)
        result <- gsub(" ", "_", result)
        result <- paste(toupper(substring(result, 1,1)), substring(result, 2), sep="", collapse=" ")

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el contenedor de Azure donde están los archivos con datos físicos que hay que procesar
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
# Resultado: contenedor de Azure
# ------------------------------------------------------------------------------------------------------------

getAzureContainer <- function(testCampaignId, testTypeId) {

        # Url a la carpeta donde están los archivos
        blobStorageContainerUrl <-paste0("https://stoisselubfzg.blob.core.windows.net/results/")

        # Clave del blob storage
        blobStorageKey                                                                                  <-
"9FGKaoD/BAsrz6Vmfx74oxD/jZBVIfVg8S1wSPJwqJ3H2tgg6A1bk5fWUAolowSR32a75IgZ91TBvg0exrwSlw=="

        # Obtenemos el contenedor
        blobStorageContainer <- blob_container(blobStorageContainerUrl, key=blobStorageKey)

        return(blobStorageContainer)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los datos de un archivo de resultados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileName: nombre del archivo
# Resultado: dataframe con señales físicas correspondientes al resultado
# ------------------------------------------------------------------------------------------------------------

getFileData <- function(testCampaignId, testTypeId, fileName) {

        # Contenedor de Azure donde están los archivos con datos físicos que hay que procesar
        blobStorageContainer <- getAzureContainer(testCampaignId, testTypeId)

        # Archivo temporal para guardar los datos en local
```

```
        tmpfile <- tempfile()

        result <- tryCatch(
        {
                        # Descargamos el archivo de Azure
                        print(paste0("Downloading file: ", fileName))

                        storage_download(blobStorageContainer, fileName, tmpfile, overwrite=TRUE)

                        # Leemos los datos en un dataframe
                        print(paste0("Reading file: ", fileName))

                        result <- read.csv(tmpfile, header = TRUE, sep = ",", dec = ".")
        },
        error=function(cond) {

                        # Se ha producido un error
                        print(paste0("Cannot download file: ", fileName))

            result <- NULL
        },
        finally={

                        # Eliminamos el archivo temporal
                        file.remove(tmpfile)

                }
        )

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el Id de resultado del nombre de un archivo
#  - resultFileName: nombre del archivo
# Resultado: Id de resultado (valor entero)
# ------------------------------------------------------------------------------------------------------------

getResultIdFromFileName <- function(resultFileName) {

        as.integer(strsplit(basename(resultFileName), "_")[[1]][1])

}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el Id de resultado a partir del cual se van a procesar los datos.
# Para ello se obtiene el mayor Id de resultado de los posibles descriptores que existan en Azure
# La consulta se realiza en la base de datos SQL Database de Azure donde se almacena la información.
# Parámetros:
# - testCampaignId: identificador de campaña de ensayos
# - testTypeId: identificador de tipo de ensayo
# - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: dataframe con los registros de los resultados del tipo de ensayo y campaña especificados
# ------------------------------------------------------------------------------------------------------------

getResultIdToStartFrom <- function(testCampaignId, testTypeId, processNewResultsOnly) {

        ResultId <- 0
        if(processNewResultsOnly == TRUE) {

                featuresFileName = paste0(getTestCampaignName(testCampaignId),
                                                        "/", getTestTypeName(testTypeId),
                                                        "/Features/features.csv")
                dfFeatures <- getFileData(testCampaignId, testTypeId, featuresFileName)
                if(!is.null(dfFeatures) == TRUE){
                        ResultId <- max(dfFeatures$ResultId)
                }
        }

        return(ResultId)
}


# ------------------------------------------------------------------------------------------------------------
# Obtiene los nombres de los archivos de datos que cuyo nombre acaba en una determinada cadena
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileEndingString: cadena final de los archivos (por ejemplo "2000.csv$")
#  - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: vector con los nombres de archivos
# ------------------------------------------------------------------------------------------------------------

getIngesysFileNames <- function(testCampaignId, testTypeId, fileEndingString, processNewResultsOnly) {

        # Obtenemos los nombres de todos los archivos
        resultFileNames <- list_blobs(getAzureContainer(testCampaignId, testTypeId), info = c("name"))
```

```
        # Filtramos los nombres que están en la carpeta de los archivos de datos físicos

        # Cadena para filtrar los archivos
        fileStartingString <- paste0("^",
                                                                getTestCampaignName(testCampaignId),
"/",
                                                                getTestTypeName(testTypeId),
"/PhysicalData/Ingesys")

        # Filtramos los archivos por la cadena inicial
        resultFileNames <- resultFileNames[grepl(fileStartingString, resultFileNames) == TRUE]

        # Filtramos los nombres de los archivos con la terminación indicada en el argumento de la función
        result <- resultFileNames[grepl(fileEndingString, resultFileNames) == TRUE]

        # Código para eliminar aquellos archivos que tengan el mismo número de ciclos
        cycles <- gsub(".*_c(.+)_.*", "\\1", result)
        dfResult      <-      data.frame(ResultId=sapply(result,      getResultIdFromFileName),File=result,
Cycles=gsub(".*_c(.+)_.*", "\\1", cycles))
        dfResult = dfResult[!duplicated(dfResult$Cycles),]

        # Obtenemos el máximo resultado procesado del posible archivo de descriptores situado en Azure.
        # Si no existe dicho archivo, se empezará desde cero.
        # Nos quedamos con los archivos cuyo resultado sea mayor que dicho valor.
        ResultId <- getResultIdToStartFrom(testCampaignId, testTypeId, processNewResultsOnly)
        dfResult <- dfResult[dfResult$ResultId > ResultId,]

        # Ordenamos los datos alfabéticamente por Id de resultado
        result <- dfResult[order(dfResult$ResultId),]

        # Extraemos el vector con el nombre del archivo
        result <- as.vector(dfResult[["File"]])

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los segmentos de tiempo para los que hay que generar los descriptores en los archivos de señales
físicas relacionados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - resultFileName: nombre del archivo  de señales físicas en donde se van a obtener los segmentos.
#               Para ello se usará la señal "EMA_position_set-point".
#  - cycles: número de ciclos del actuador en el momento de registrar el archivo en el sistema
# Resultado: segmentos de para buscar descriptores en los archivos de señales físicas relacionados
# ------------------------------------------------------------------------------------------------------------

getFeatureSegments <- function(testCampaignId, testTypeId, resultFileName, cycles) {

        # Dataframe que almacenará los segmentos para los descriptores
        dfFeatureSegments  <-  data.frame(InitialTimestamp  =  numeric(0),  FinalTimestamp  =  numeric(0),
SegmentType = integer(0), Cycles = integer(0))

        # Variable auxiliar para encontrar los puntos de comienzo de subida (raisingEdge) y de bajada
(fallingEdge)
        threshold <- 50.00001

        # Tiempos mínimos de los dos tipos de segmentos para ser considerados válidos
        risigEdgeToFallingEdgeMinimunSegmentTime <- 0.3339
        fallingEdgeToRaisingEdgeMinimunSegmentTime <- 0.41

        # Descargamos el archivo para generar los segmentos
        resultFileData <- getFileData(testCampaignId, testTypeId, resultFileName)
        if(is.null(resultFileData) == TRUE) {
                # Se ha producido un problema en la lectura del archivo
                return(NULL)
        }

        # Obtenemos los vectores de la señal "EMA_position_set-point" y timestamp
        print(paste0("Creating segments from file: ", resultFileName))
        timestamp <- resultFileData$Timestamp
        emaPositionSetPoint <- resultFileData$EMA_position_set.point

        # Se crea un vector de emaPositionSetPoint desplazado a la izquierda en una posición
        data <- emaPositionSetPoint[2:length(emaPositionSetPoint)]
        shiftedEmaPositionSetPoint <- c(data, NA)

        # Posiciones de los puntos de comienzo de subida y bajada
        raisingEdgePositions  <-  which(emaPositionSetPoint  <  threshold  &  shiftedEmaPositionSetPoint  >
threshold)
        fallingEdgePositions  <-  which(emaPositionSetPoint  >  threshold  &  shiftedEmaPositionSetPoint  <
threshold)

        # Dataframe con las posiciones RE
```

```
        positions1 <- data.frame("Position" = raisingEdgePositions, "Type" = rep(c(1), times =
length(raisingEdgePositions)))

        # Dataframe con las posiciones FE
        positions2 <- data.frame("Position" = fallingEdgePositions, "Type" = rep(c(0), times =
length(fallingEdgePositions)))

        # Juntamos los dos dataframes
        positions <- rbind(positions1,positions2)

        # Ordenamos el dataframe
        positions <- positions[with(positions, order(Position)),]

        # Bucle de obtención de segmentos para descriptores.
        # Para cada segmento anotamos lo siguiente:
        # - Timestamp inicial
        # - Timestamp final
        # - Tipo de segmento (1: RisingEdge-FallingEdge; 0: FallingEdge-RisingEdge)
        # - Índice de ciclo (relativo al archivo)
        p <- 1
        cycleIncremented <- FALSE
        while (p < nrow(positions)) {

                # El objetivo es almacenar parejas de sementos RE-FE y FE-RE que estén adyacentes y que sean
válidos.

                # Variable para almacenar de forma temporal la información del primer segmento y segundo
segmento
                firstSegmentData <- NULL
                secondSegmentData <- NULL

                # Primer segmento (siempre que haya un segundo segmento)
                # Restamos una unidad al número máximo, así nos aseguramos de que hay un segundo segmento.
                if ((p+1) < nrow(positions)) {

                        initialSegmentPosition <- positions[p,1]
                        finalSegmentPosition <- positions[p+1,1]

                        # Tiempo mínimo y tiempo real del segmento
                        segmentInitialValueIsRaising <- ifelse(positions[p,2] == 1, TRUE, FALSE)

                        minimumSegmentTime <- ifelse(segmentInitialValueIsRaising,

risigEdgeToFallingEdgeMinimunSegmentTime,

fallingEdgeToRaisingEdgeMinimunSegmentTime)
                        segmentTime <- timestamp[finalSegmentPosition] - timestamp[initialSegmentPosition]


                        # El tiempo del segmento debe superar el valor mínimo
                        if(segmentTime > minimumSegmentTime) {

                                # Guardamos el segmento de forma temporal
                                firstSegmentData           <-           list(InitialTimestamp           =
timestamp[initialSegmentPosition],

                                             FinalTimestamp = timestamp[finalSegmentPosition],

                                             SegmentType = 1*segmentInitialValueIsRaising,

                                             Cycles = cycles)
                        }
                }

                # Segundo segmento (siempre que exista)
                # Además, descartamos el segundo segmento si el primer segmento no es válido
                if((p+1) < nrow(positions) && !is.null(firstSegmentData)) {

                        # Para obtener el valor final del segmento, aumentamos en una unidad la posición
                        initialSegmentPosition <- positions[p+1,1]
                        finalSegmentPosition <- positions[p+2,1]

                        # Tiempo mínimo y tiempo real del segmento.
                        # El tiempo mínimo será al revés que para el primer segmento.
                        # Por ello, utilizamos la operación "not" en la variable segmentInitialValueIsRaising
                        segmentInitialValueIsRaising <- ifelse(positions[p+1,2] == 1, TRUE, FALSE)

                        minimumSegmentTime <- ifelse(segmentInitialValueIsRaising,

risigEdgeToFallingEdgeMinimunSegmentTime,

fallingEdgeToRaisingEdgeMinimunSegmentTime)
                        segmentTime <- timestamp[finalSegmentPosition] - timestamp[initialSegmentPosition]


                        # El tiempo del segmento debe superar el valor mínimo
```

```
                        if(segmentTime > minimumSegmentTime) {

                                # Primero, anotamos el primer segmento
                                dfFeatureSegments[nrow(dfFeatureSegments) + 1,] <- firstSegmentData

                                # Anotamos el segundo segmento
                                secondSegmentData           <-           list(InitialTimestamp           =
timestamp[initialSegmentPosition],
                                                                                                        FinalTimestamp      =
timestamp[finalSegmentPosition],
                                                                                                        SegmentType         =
1*(segmentInitialValueIsRaising),
                                                                                                        Cycles = cycles)
                                dfFeatureSegments[nrow(dfFeatureSegments) + 1,] <- secondSegmentData
                        }
                }

                # Incremento del puntero de la posición
                if(!is.null(firstSegmentData) && !is.null(secondSegmentData)) {
                        # Los dos segmentos son válidos

                        # Incremento del puntero en las posiciones
                        p <- p + 2
                }
                else if(!is.null(firstSegmentData) && is.null(secondSegmentData)) {
                        # El primer segmento es válido, el segundo no es válido

                        # Incremento del puntero en las posiciones
                        p <- p + 2
                }
                else if(is.null(firstSegmentData)) {
                        # El primer segmento es inválido

                        # Incremento del puntero en las posiciones
                        p <- p + 1
                }
                else
                {
                        # En teoría, nunca se debería dar este caso

                        # Incremento del puntero en las posiciones
                        p <- p + 2
                }

                # Aumentamos el número de ciclos
                cycles <- cycles + 1
        }

        return(dfFeatureSegments)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los descriptores para una señal y un segmento de dicha señal
# Parámetros:
#  - featureList: lista para guardar los datos de los descriptores
#  - featureName: nombre del descriptor
#  - featureValue: valor del descriptor
# Resultado: generación de descriptores para el archivo
# ------------------------------------------------------------------------------------------------------------

addFeatureValue <- function(featureList,
                                                        featureName,
                                                        featureValue) {

        # Obtenemos el vector con los datos del descriptor
        featureData <- featureList[[featureName]]
        if(is.null(featureData)) {
                # Guardamos el primer valor del descriptor.
                featureList[featureName] <- c(featureValue)
        }
        else {
                # El descriptor ya tenía valores. Agregamos el nuevo valor
                featureData <- c(featureData, featureValue)
                featureList[[featureName]] <- featureData
        }

        return(featureList)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los descriptores para una señal y un segmento de dicha señal
# Parámetros:
#  - featureList: lista para guardar los datos de los descriptores
#  - signalData: dataframe con los datos de la señal. Contiene dos columnas:
#                * Columna con los datos de la señal cuyo nombre se indica en el argumento "signalName"
#                * Columna "Timestamp" con los datos de timestamp de la señal
```

```
#  - signalName: nombre de la señal (es el nombre columna del dataframe con estos datos)
#  - segment: segmento en donde se van a extraer descriptores
# Resultado: generación de descriptores para el archivo
# ------------------------------------------------------------------------------------------------------------

getSignalFeaturesForSegment <- function(featureList, signalData, signalName, segment) {

        # Obtenemos el segmento de la señal en donde se van a calcular los descriptores
        initialTimestamp <- segment$InitialTimestamp
        finalTimestamp <- segment$FinalTimestamp
        signalDataSegment <- signalData[signalData$Timestamp >= initialTimestamp & signalData$Timestamp <=
finalTimestamp,][[signalName]]

        # Los nombres de los descriptores que se generarán serán distintos en función del tipo de segmento.
        # Indicamos en el nombre del descriptor el tipo:
        #         1: RisingEdge-FallingEdge; 0: FallingEdge-RisingEdge

        # RMS
        rmsValue <- sqrt(sum(signalDataSegment ^2) / length(signalDataSegment));
        featureName      <-  paste0("Rms_",  ifelse(segment$SegmentType  ==  1,"RF","FR"),  "_",  signalName)

        featureList <- addFeatureValue(featureList, featureName, rmsValue)

        # Mediana
        medianValue <- median(signalDataSegment)
        featureName      <- paste0("Median_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, medianValue)

        # Peak to Peak
        peakToPeakValue <- abs(max(signalDataSegment) - min(signalDataSegment))
        featureName      <- paste0("Pk2Pk_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, peakToPeakValue)

        # Peak value
        peakValue <- max(abs(signalDataSegment))
        featureName      <- paste0("Pv_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, peakValue)

        # Kurtosis
        kurtosisValue <- kurtosis(signalDataSegment)
        featureName      <- paste0("Ku_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, kurtosisValue)

        # Crest factor
        crestFactorValue <- peakValue / rmsValue
        featureName      <- paste0("Crf_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, crestFactorValue)

        # Clearance factor
        clearanceFactorValue <- peakValue / ((mean(sqrt(abs(signalDataSegment))))^2)
        featureName      <- paste0("Clf_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, clearanceFactorValue)

        # Impulse factor
        impulseFactorValue <- peakValue / (mean(abs(signalDataSegment)))
        featureName      <- paste0("Imf_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, impulseFactorValue)

        # Shape factor
        shapeFactorValue <- rmsValue / (mean(abs(signalDataSegment)))
        featureName      <- paste0("Shf_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, shapeFactorValue)

        # Máximo
        maxValue <- max(signalDataSegment)
        featureName      <- paste0("Max_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, maxValue)

        # Mínimo
        minValue <- min(signalDataSegment)
        featureName      <- paste0("Min_", ifelse(segment$SegmentType == 1,"RF","FR"), "_", signalName)
        featureList <- addFeatureValue(featureList, featureName, minValue)

        return(featureList)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los descriptores de un archivo a partir de los segmentos indicados
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - featureList: lista para guardar los datos de los descriptores
#  - signalNames: vector con los nombres de las señales para las que se van a generar descriptores
#  - resultFileName: nombre del archivo  de señales físicas en donde se van a obtener los segmentos.
#  - featureSegments: dataframe con los segmentos donde se van a extraer descriptores
# Resultado: generación de descriptores para el archivo
```

```
# -------------------------------------------------------------------------------------------------------------

getFeaturesForFile <- function(testCampaignId,
                                                                testTypeId,
                                                                featureList,
                                                                signalNames,
                                                                resultFileName,
                                                                featureSegments) {

        # Obtenemos los datos del archivo
        resultFileData <- getFileData(testCampaignId, testTypeId, resultFileName)
        if(is.null(resultFileData) == TRUE) {
                # Se ha producido algún error en la lectura de los datos del archivo.
                # Devolvemos como resultado la misma lista de datos recibida en la función
                return(featureList)
        }

        print(paste0("Creating descriptors for file: ", resultFileName))

        # Fecha-hora del archivo
        fileDatetime <- getDateTimeForFeature(resultFileName)

        # Id de resultado del archivo
        # Por ejemplo, en siguiente archivo:
        #
        "/campana0003/FatigueTest/PhysicalData/Ingesys/196_2018_12_11_08_03_31_Ingesys_c18961_n2000.csv"
        # el Id de resultado sería 196
        resultId <- as.integer(strsplit(basename(resultFileName), "_")[[1]][1])

        # Número de ciclos del archivo. Se obtienen del nombre del archivo. Ejemplo de nombre:
        #
        "/campana0003/FatigueTest/PhysicalData/Ingesys/196_2018_12_11_08_03_31_Ingesys_c18961_n2000.csv"
        # El número de ciclos está entre la subcadena "_c" y "_"
        # Forma de obtener el dato: gsub(".*STR1(.+)STR2.*", "\\1", resultFileName)
        # donde STR1= "_c" y STR2="_"
        fileCycleNumber <- gsub(".*_c(.+)_.*", "\\1", resultFileName)

        # Columnas del archivo (nombres de las señales)
        resultFileDataSignalNames <- colnames(resultFileData)

        # Buscamos cada una de las señales en el archivo y si se encuentra se generan los descriptores
        for(sn in 1:length(signalNames)) {

                if(signalNames[sn] %in% resultFileDataSignalNames) {

                        # Obtenemos los datos de la señal junto con el timestamp en un dataframe

                        signalData <- resultFileData[,c(signalNames[sn],"Timestamp")]

                        # Bucle para guardar los descriptores
                        # Siempre habrá un segmento RF y otro FR con el mismo número de ciclos
                        # El objetivo final es crear una única fila para los descriptores que se generen
para dichos segmentos.
                        #  Por tanto, los descriptores compartirán los mismos metadatos, que serán los
siguientes:
                        # - ResultId: Id de resultado
                        # - Cycles: número de ciclos
                        # - Datetime: fecha en la que se capturó el archivo en donde están los segmentos
                        #  - InitialTimestamp: timestamp inicial del segmento que esté antes (es decir,
timestamp inicial menor de los dos segmentos)
                        #  - FinalTimestamp: timestamp final del segmento que esté después (es decir,
timestamp final mayor de entre los dos segmentos)

                        fs <- 1
                        while(fs < nrow(featureSegments)) {

                                # Ciclos del segmento
                                cycles <- featureSegments[fs,]$Cycles

                                # Timestamp inicial de la pareja de segmentos
                                initialTimestamp <- featureSegments[fs,]$InitialTimestamp

                                # El timestamp final se obtiene en el siguiente segmento
                                finalTimestamp <- NULL

                                # Generamos los descriptores de la señal para el segmento
                            featureList <- getSignalFeaturesForSegment(featureList,

                    signalData,

                    signalNames[sn],

                    featureSegments[fs,])

                                # Siguiente segmento
                                fs <- fs + 1
```

```
                                # Comprobación de que existe el segundo segmento (siempre debería ser así).
                                # También comprobamos que el número de ciclos es el mismo (lo cual debería
ser siempre así también)
                                if(fs <= nrow(featureSegments) && cycles == featureSegments[fs,]$Cycles) {

                                        # Timestamp final de la pareja de segmentos
                                        finalTimestamp <- featureSegments[fs,]$FinalTimestamp

                                        # Generamos los descriptores de la señal para el segmento
                                        featureList <- getSignalFeaturesForSegment(featureList,

                                signalData,

                                signalNames[sn],

                                featureSegments[fs,])
                                }

                                # Agregamos los metadatos para los descriptores obtenidos en una pareja RF-
FR (o FR-RF)
                                # Esto lo hacemos siempre que no se haya agregado antes esta información.
                                # Es decir, lo agregamos siempre que no se haya procesado antes la pareja
de segmentos.
                                # Esto será así cuando se haya procesado antes algún archivo relacionado,
o bien los archivos de otra señal.
                                # La comprobación que hay que hacer es que no se haya insertado antes el
dato del número de ciclos.
                                insertMetadata <- FALSE
                                if(is.null(featureList[["Cycles"]]) == TRUE) {

                                        insertMetadata <- TRUE
                                }
                                if(insertMetadata == FALSE) {
                                        if(cycles %in% featureList[["Cycles"]] == FALSE) {
                                                insertMetadata <- TRUE
                                        }
                                }
                                if(insertMetadata == TRUE) {

                                        # Agregamos los metadatos de los descriptores

                                        featureList <- addFeatureValue(featureList, "ResultId", resultId)

                                        featureList  <-  addFeatureValue(featureList,  "Cycles",  cycles)

                                        featureList    <-    addFeatureValue(featureList,    "Datetime",
fileDatetime)
                                        featureList  <-  addFeatureValue(featureList,  "InitialTimestamp",
initialTimestamp)
                                        featureList   <-   addFeatureValue(featureList,   "FinalTimestamp",
finalTimestamp)
                                }

                                # Siguiente segmento
                                fs <- fs + 1
                        }
                }
        }

        return(featureList)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene los descriptores a partir de las señales físicas
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - signalNames: vector con los nombres de las señales para las que se van a generar descriptores
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
#  - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE) -----------
-------------------------------------------------------------------------------------------

getFeatures <- function(testCampaignId, testTypeId, signalNames, tempFolder, processNewResultsOnly) {

        # Lista para guardar los datos de los descriptores.
        # Se utiliza una lista en vez de un dataframe para ir agregando los datos de forma incremental.
        # La lista tendrá dos tipos de elementos:
        # - Elementos para guardar metadatos de los descriptores
        # - Elementos para guardar descriptores
        # Los elementos que guardan metadatos serán los siguientes:
        # - ResultId: identificador del resultado que ha dado origen a la "fila" de datos de descriptores
        # - Cycles: número de ciclos correspondiente a una "fila" de datos de descriptores
        # - Datetime: fecha de creación del archivo que correspondiente a la "fila" de datos de descriptores
        #  - InitialTimestamp: timestamp relativo del inicio de una pareja de segmentos RF-FR (o FR-RF)
adyacentes
```

```
        # - FinalTimestamp: timestamp relativo del final de una pareja de segmentos RF-FR (o FR-RF) adyacentes
        # En una fila se guardan los descriptores de todas las señales de interés para una pareja de segmentos
RF-FR (o FR-RF) adyacentes
        # En la siguiente instrucción, a pesar de que pone "vector", se genera una lista.
        # Creamos inicialmente los elementos de los metadatos para que aparezcan antes que los elementos de
los descriptores
        featureList    <-    list(ResultId=NULL,    Cycles=NULL,    Datetime=NULL,    InitialTimestamp=NULL,
FinalTimestamp=NULL)

        # Obtenemos los nombres de los archivos que contienen las variables "EMA_position_set-point".
        # Estas variables están en los archivos de Ingesys que tienen 2000 muestras (los nombres acaban en
"2000.csv")
        print("Reading files for segment creation")
        resultFileNames <- getIngesysFileNames(testCampaignId,
                                                                                testTypeId,
                                                                                "2000.csv$",

processNewResultsOnly)

        if(length(resultFileNames) > 0) {
                # Bucle para generar el dataframe de segmentos para los descriptores
                for(f in 1:length(resultFileNames)) {

                        # Nombre del archivo que contiene la señal "EMA_position_set-point"
                        resultFileName <- resultFileNames[f]

                        # Se obtienen los números de ciclos a partir del nombre del archivo. Ejemplo de
nombre de archivo:
                        #
        "/campana0003/FatigueTest/PhysicalData/Ingesys/196_2018_12_11_08_03_31_Ingesys_c18961_n2000.csv"
                        # El número de ciclos está entre la subcadena "_c" y "_"
                        # Forma de obtener el dato: gsub(".*STR1(.+)STR2.*", "\\1", resultFileName)
                        # donde STR1= "_c" y STR2="_"
                        cycles <- gsub(".*_c(.+)_.*", "\\1", resultFileName)
                        cycles <- as.integer(cycles)

                        # Obtenemos los segmentos de las señales para los que se van a obtener descriptores

                        dfFeatureSegments <- getFeatureSegments(testCampaignId, testTypeId, resultFileName,
cycles)

                        if(!is.null(dfFeatureSegments)) {
                                if(nrow(dfFeatureSegments) > 0) {

                                        # Obtenemos los descriptores en ciertas señales físicas de los
archivos relacionados.
                                        # Los archivos relacionados son los siguientes:
                                        # - El mismo archivo que se ha utilizado para obtener los segmentos
de los descriptores (2000 muestras)
                                        # - El archivo con la misma raíz en su nombre que tiene 100000
muestras
                                        # Ejemplo:
                                        # - "196_2018_12_11_08_03_31_Ingesys_c18961_n2000.csv"
                                        # - "196_2018_12_11_08_03_31_Ingesys_c18961_n100000.csv"
                                        featureList <- getFeaturesForFile(testCampaignId,

          testTypeId,

          featureList,

          signalNames,

          resultFileName,

          dfFeatureSegments)


                                        relatedResultFileName    <-    sub('2000.csv$',    "100000.csv",
resultFileName)
                                        featureList <- getFeaturesForFile(testCampaignId,

          testTypeId,

          featureList,

          signalNames,

          relatedResultFileName,

          dfFeatureSegments)

                                }
                        }
                }
        }
        else {
```

```
                # Ponemos a null la lista
                featureList <- NULL
        }

        return(featureList)
}

# ------------------------------------------------------------------------------------------------------------
# Crea el archivo de descriptores y lo carga en Azure
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileName: nombre del archivo de los descriptores
#  - features: dataframe con los descriptores
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
# Resultado: carga de los descriptores en Azure
# ------------------------------------------------------------------------------------------------------------

createFeatureFileAndUploadOnAzure <- function(testCampaignId, testTypeId, fileName, features, tempFolder) {

        # Ruta relativa y nombre del archivo del descriptor
        descriptorFileName <- paste0("/", getTestCampaignName(testCampaignId),
                                                                "/",       getTestTypeName(testTypeId),

                                                                "/Features",

                                                                "/", fileName, ".csv")

        # Si no existe el directorio temporal, se crea
        dir.create(tempFolder, showWarnings = FALSE)

        # Comprobaciones en los datos
        if(nrow(features) != 0) {

                print(paste0("Creating descriptor file: ", descriptorFileName))

                # Creamos el archivo en la ubicación temporal
                completeFileName <- file.path(tempFolder, descriptorFileName)

                # Creamos las subcarpetas del archivo
                fileFolder <- dirname(completeFileName)
                dir.create(fileFolder, recursive = TRUE)

                # Creamos el archivo en la ubicación temporal
                write.csv(features, completeFileName, row.names = FALSE)

                # Subimos el archivo a Azure
                print(paste0("Uploading file on Azure: ", descriptorFileName))

                if(ubuntuVirtualMachine == TRUE) {

                        cmd = paste0("/home/adminssh/isselub/azcopy copy \"",
                                                tempFolder, "/*\" ",

"\"https://stoisselubfzg.blob.core.windows.net/results?sv=2019-02-02&ss=b&srt=sco&sp=rwdlac&se=2040-04-
27T19:17:18Z&st=2020-04-27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\"     --
recursive")
                        system(cmd)
                }
                else {

                        cmd = paste0("azcopy copy \"",
                                                tempFolder, "/*\" ",

"\"https://stoisselubfzg.blob.core.windows.net/results?sv=2019-02-02&ss=b&srt=sco&sp=rwdlac&se=2040-04-
27T19:17:18Z&st=2020-04-27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\"     --
recursive")
                        system("cmd.exe", input = cmd)
                }

                print(paste0("File uploaded: ", descriptorFileName))

                # Borramos la carpeta temporal y todo su contenido
                unlink(tempFolder, recursive = TRUE)
        }
}

# ------------------------------------------------------------------------------------------------------------
# Crea el dataframe con los de descriptores a partir de la lista generada durante el proceso
# Parámetros:
#  - features: dataframe con los descriptores
# Resultado: carga de los descriptores en Azure
# ------------------------------------------------------------------------------------------------------------

createFeatureDataframe <- function(featureList) {
```

```
        # Obtenemos los nombres de los descriptores
        featureNames <- names(featureList)

        # Bucle para generar el dataframe con los descriptores
        dfFeatures <- NULL
        for(d in 1:length(featureNames)) {

                if(is.null(dfFeatures)) {

                        # Cargamos el vector del primer descriptor
                        dfFeatures <- data.frame(featureList[[d]], stringsAsFactors=FALSE)

                        # Reemplazamos el nombre de la única columna por el nombre del descriptor
                        names(dfFeatures) <- featureNames[d]
                }
                else {

                        # Agregamos el descriptor al dataframe
                        dfFeatures[,featureNames[d]] <- featureList[[d]]
                }
        }

        return(dfFeatures)
}

# ------------------------------------------------------------------------------------------------------------
# Procesa los archivos de señales físicas
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
# - processNewResultsOnly: procesar sólo nuevos resultados (TRUE) o bien procesar todos (FALSE)
# Resultado: carga en azure del archivo con los descriptores
# ------------------------------------------------------------------------------------------------------------

processPhysicalFileResults <- function(testCampaignId,
                                                                                testTypeId,
                                                                                tempFolder,
                                                                                processNewResultsOnly) {

        featuresFileName = "features"

        # Dataframes cuyos datos se van a generar y serializar
        df_RE_FE_Features <- NULL
        df_FE_RE_Features <- NULL
        df_All_Features <- NULL

        # Señales a partir de las cuales se van a obtener descriptores
        signalNames <- c("Cylinder_force_filtered",
                                        "Accelerometer_X",
                                        "Accelerometer_Y",
                                        "Accelerometer_Z",
                                        "EMA_Force",
                                        "Motor_current_1",
                                        "Motor_current_2",
                                        "Stober_Speed_Meassurement")

        # Obtenemos una lista con los descriptores que se han creado
        featureList <- getFeatures(testCampaignId,
                                                                testTypeId,
                                                                signalNames,
                                                                tempFolder,
                                                                processNewResultsOnly)

        if(!is.null(featureList)) {

                # Obtenemos el dataframe de los descriptores a partir de la lista
                dfFeatures <- createFeatureDataframe(featureList)

                # Comprobamos si es una actualización incremental de descriptores.
                # Si es así, hay que unir los nuevos descriptores con los que estén en la nube.
                if(processNewResultsOnly == TRUE) {
                        # Obtenemos los descriptores que estén en la nube y creamos un dataframe unificado
                        fileName = paste0(getTestCampaignName(testCampaignId),
                                                                        "/", getTestTypeName(testTypeId),
                                                                        "/Features/features.csv")
                        dfPreviousFeatures <- getFileData(testCampaignId, testTypeId, fileName)
                        if(!is.null(dfPreviousFeatures) == TRUE) {
                                dfFeatures <- rbind(dfPreviousFeatures, dfFeatures)
                        }
                }

                # Serializamos los descriptores en Azure
                createFeatureFileAndUploadOnAzure(testCampaignId, testTypeId, featuresFileName, dfFeatures,
tempFolder)
        }
```

```
}

# ------------------------------------------------------------------------------------
# Genera los descriptores
# ------------------------------------------------------------------------------------

processData <- function() {

        testCampaignId <- 3
        testTypeId <- 3
        processNewResultsOnly <- TRUE

        if(ubuntuVirtualMachine == FALSE) {

                tempFolder <- "D:/Users/rgonzalez/Documents/Datos/temp/TFM/Archivos/temp"
        }
        else {

                tempFolder <- "/home/adminssh/isselub/temp"
        }

        processPhysicalFileResults(testCampaignId, testCampaignId, tempFolder, processNewResultsOnly)
        print("Feature extraction finished")
}

processData()
```

9.4 Feature Analysis Code

Function “processData” can be run to perform the analysis. The call to this function can be found at the end of this script.

```
# install.packages("RODBC")
# install.packages("AzureStor")
# install.packages("factoextra")

# Carga de paquetes en el entorno de R
library(RODBC)
library(AzureStor)
#library(factoextra)
library(MASS)
library(ggplot2)
library(plyr)

# Variable que indica si estamos ejecutando el script en Azure de forma automatizada
ubuntuVirtualMachine <- FALSE

# ------------------------------------------------------------------------------------------------------------------
# Obtiene los límites superior e inferior de referencia y de prueba del gráfico de Hotelling (fase I de
Hotelling)
# Parámetros:
#  - x: dataset de referencia para los que se va a calcular los límites
#  - alpha: valor de significación que se quiere utilizar para las fórmulas
# Resultado: límites superior e inferior del gráfico, junto con otros valores calculados y utilizados en las
fórmulas
# ------------------------------------------------------------------------------------------------------------------

Q_limits <- function(x, alpha){

        # Dimensiones (número de filas y columnas) del dataset de prueba
        m <- nrow(x)
        p <- ncol(x)

        # Media para cada columna de los datos de referencia
        x_hat_m <- apply(x,2,mean)

        # Covarianza del dataset de prueba
        Sm <- cov(x)

        # Valor estadístico Q para el dataset de referencia
        Q_history  <- apply(x, MARGIN = 1, FUN = Qj, x_hat_m, Sm )

        # Límites superior e inferior para el dataset de referencia
        UCL <- (m-1)^2/m*qbeta(1-(alpha/2), p/2, (m-p-1)/2)
        LCL <- (m-1)^2/m*qbeta(alpha/2, p/2, (m-p-1)/2)

        # Límites superior e inferior para el dataset de prueba
        p_UCL <- p*(m+1)*(m-1)/(m^2-m*p)*qf(1-alpha/2, p, m-p)
        p_LCL <- p*(m+1)*(m-1)/(m^2-m*p)*qf(alpha/2, p, m-p)
```

```
        return(list(Q_values = Q_history, UCL = UCL, LCL = LCL, p_UCL = p_UCL, p_LCL = p_LCL, x_hat = x_hat_m,
S = Sm))
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el valor estadístico Q para el dataset de prueba (fase II de Hotelling)
# Parámetros:
#  - x_instance: dataset de prueba
#  - x_hat_m: media de cada columna del dataset de prueba
#  - Sm: covarianza del dataset de referencia
# Resultado: vector con el valor estadístico Q para el dataset de prueba
# ------------------------------------------------------------------------------------------------------------

Qj <- function(x_instance, x_hat_m, Sm){

        x_instance <- as.matrix(x_instance)

        Q <- t(x_instance - x_hat_m) %*% solve(Sm) %*% ((x_instance - x_hat_m))

        return(Q)
}

# Función que obtiene una conexión a la base de datos de Azure
getAzureSQLDatabaseServerConnection <- function() {

        # Base de datos SQL Database en Azure
        AzureSQLDatabaseServer <- "sqlisselub.database.windows.net"
        AzureSQLDatabaseUser <- "adminsql"
        AzureSQLDatabasePassword <- "Tekniker2016@"
        AzureSQLDatabase <- "configuration"
        if(ubuntuVirtualMachine == TRUE) {
                # Driver ODBC para la virtualización de Ubuntu
                AzureSQLDatabaseDriver <- "ODBC Driver 17 for SQL Server"
        }
        else {
                # Driver ODBC para la virtualización de Windows
                AzureSQLDatabaseDriver <- "SQL Server"
        }
        AzureSQLDatabaseConnectionString <- paste0(
                "Driver=", AzureSQLDatabaseDriver,
                ";Server=", AzureSQLDatabaseServer,
                ";Database=", AzureSQLDatabase,
                ";Uid=", AzureSQLDatabaseUser,
                ";Pwd=", AzureSQLDatabasePassword)

        # Conexión con la base de datos
        result <- odbcDriverConnect(AzureSQLDatabaseConnectionString)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el contenedor de Azure donde está el archivo de descriptores
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
# Resultado: contenedor de Azure
# ------------------------------------------------------------------------------------------------------------

getAzureContainer <- function(testCampaignId, testTypeId) {

        # Url a la carpeta donde están los archivos
        blobStorageContainerUrl <-paste0("https://stoisselubfzg.blob.core.windows.net/results/")

        # Clave del blob storage
        blobStorageKey                                                                                  <-
"9FGKaoD/BAsrz6Vmfx74oxD/jZBVIfVg8S1wSPJwqJ3H2tgg6A1bk5fWUAolowSR32a75IgZ91TBvg0exrwSlw=="

        # Obtenemos el contenedor
        blobStorageContainer <- blob_container(blobStorageContainerUrl, key=blobStorageKey)

        return(blobStorageContainer)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003".
# Deben generarse cuatro dígitos para el número de la campaña.
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
# Resultado: cadena de caracteres con el nombre de la campaña de tests
# Se genera siempre cuatro dígitos para el número de la campaña (se completa con ceros).
# Por ejemplo, para la campaña 3 se crea la siguiente cadena: "campana0003"
# ------------------------------------------------------------------------------------------------------------

getTestCampaignName <- function(testCampaignId) {
```

```
        # Creamos una cadena de caracteres con el nombre de la campaña de tests, por ejemplo "campana0003"
        # El número de dígitos en la cadenaserá siempre de cuatro
        campaignNumberString = as.character(testCampaignId)
        campaignNumberString = paste("campana", formatC(testCampaignId, width=4, flag="0"), sep="")

        return(campaignNumberString)
}

# ------------------------------------------------------------------------------------------------------------------
# Obtiene el texto de un tipo de ensayo
# Parámetros:
#  - testTypeId: tipo de test
# Resultado: texto correspondiente al tipo de ensayo
# ------------------------------------------------------------------------------------------------------------------

getTestTypeName <- function(testTypeId) {

        # Conexión a la base de datos
        AzureSQLDatabaseConnection <- getAzureSQLDatabaseServerConnection()

        # Consulta
        query <- paste0("SELECT Nombre as TestName ",
                                        "FROM [isselub].[tiposensayo] ",
                                        "WHERE Id = ", testTypeId)

        dfQueryResult <- sqlQuery(AzureSQLDatabaseConnection, query)

        # Cerramos la conexión a la base de datos
        close(AzureSQLDatabaseConnection)

        # Modificaciones que hacemos en el nombre:
        # - Se eliminan espacios en blanco al comienzo y al final
        # - Se sustituye los espacios en blanco por un guión bajo
        # - Se pone en mayúscula la primera letra
        result = as.character(dfQueryResult[1,1])
        result <- trimws(result)
        result <- gsub(" ", "_", result)
        result <- paste(toupper(substring(result, 1,1)), substring(result, 2), sep="", collapse=" ")

        return(result)
}

# ------------------------------------------------------------------------------------------------------------------
# Obtiene los datos del archivo de descriptores
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileName: nombre del archivo de descriptores
# Resultado: dataframe con señales físicas correspondientes al resultado
# ------------------------------------------------------------------------------------------------------------------

getFileData <- function(testCampaignId, testTypeId, fileName) {

        # Contenedor de Azure donde está el archivo
        blobStorageContainer <- getAzureContainer(testCampaignId, testTypeId)

        # Archivo temporal para guardar los datos en local
        tmpfile <- tempfile()

        # Descargamos el archivo de Azure
        print(paste0("Downloading file: ", fileName))
        storage_download(blobStorageContainer, fileName, tmpfile, overwrite=TRUE)

        # Leemos los datos en un dataframe
        print(paste0("Reading file: ", fileName))
        result <- read.csv(tmpfile, header = TRUE, sep = ",", dec = ".")

        # Eliminamos el archivo temporal
        file.remove(tmpfile)

        return(result)
}

# ------------------------------------------------------------------------------------------------------------------
# Elimina una ventana de descriptores después de un reinicio del banco de ensayos
# Parámetros:
#  - featuresData: dataframe con los descriptores
#  - benchStopTimeInSeconds: tiempo de parada (en segundos) del banco
#  - timeWindowInSecondsToRemoveInBenchRestarts: ventana de tiempo (en segundos) de descriptores
#                que se eliminarán después de un reinicio del banco de ensayos
# Resultado: dataframe filtrado de descriptores
# ------------------------------------------------------------------------------------------------------------------

removeFeaturesAfterBenchRestarts <- function(featuresData,
```

```
benchStopTimeInSeconds,

timeWindowInSecondsToRemoveInBenchRestarts) {

	# ------------------------------------------------------------------------
	# Se eliminan los datos en una franja después de una parada en el banco superior a un tiempo.
	# Esto se hace así para esperar a que el funcionamiento del banco se estabilice
	# ------------------------------------------------------------------------

	# Se crea un vector con la columna "Datetime" desplazado a la derecha en una posición.
	# El primer valor del nuevo vector coincide con el primer valor del vector "Datetime".
	# Esto se hace así para que el primer valor del vector diferencia que se va a construir sea cero.
	data <- featuresData$Datetime[1:(length(featuresData$Datetime)-1)]
	shiftedDatetime <- c(featuresData$Datetime[1],data)

	# Se crea un vector con las diferencias (segundos) entre elementos contiguos en "Datetime"
	differenceDatetime <- featuresData$Datetime - shiftedDatetime

	# Identificamos las posiciones en donde se ha vuelvo a arrancar el banco.
	# Son aquellos puntos cuya diferencia de tiempo es mayor de un periodo.
	stopPoints <- which(differenceDatetime > benchStopTimeInSeconds)

	# Bucle para eliminar elementos.
	# Se recorren los puntos de arranque desde atrás hacia el principio.
	# Para cada punto, se eliminan los elementos dentro del margen seleccionado (dos horas)
	p <- length(stopPoints)
	while(p > 0) {

		# Seleccionamos el siguiente elemento al punto de arranque
		s <- p + 1

		# Tiempo acumulado de diferencia con respecto al punto inicial de arranque
		accumulatedDatetime <- 0

		# Vamos recorriendo los puntos hasta que la diferencia de tiempo con respecto
		# al punto de arranque sea mayor que un determinado valor
		while(accumulatedDatetime <= timeWindowInSecondsToRemoveInBenchRestarts && s <=
length(featuresData$Datetime)) {

			# Agregamos la diferencia de tiempo
			accumulatedDatetime <- accumulatedDatetime + differenceDatetime[s]

			# Siguiente elemento
			s <- s + 1
		}

		# Eliminamos los datos de la franja de tiempo
		featuresData <- featuresData[-(p:(s-1)),]

		# Siguiente punto (vamos hacia atrás)
		p <- p - 1
	}

	return(featuresData)
}

# ------------------------------------------------------------------------------------------------------------
# Elimina los datos extremos del dataframe del estadístico Q
# Parámetros:
#  - dfQData: dataframe con los datos del estadístico Q total (referencia y prueba)
#  - numberOfIqrTimesForOutlierRemovalInQ: Número de veces por el que hay que multiplicar el rango
#       intercuartílico para eliminar los datos extremos
# Resultado: dataframe filtrado de datos extremos
# ------------------------------------------------------------------------------------------------------------

removeQDataOutliers <- function(dfQData, numberOfIqrTimesForOutlierRemovalInQ) {

	# Rango intercuartílico
	lowerq = quantile(dfQData$Q)[2]
	upperq = quantile(dfQData$Q)[4]
	iqr <- upperq - lowerq

	# Límites para el filtrado de datos extremos
	outlierUpperThreshold <- (iqr * numberOfIqrTimesForOutlierRemovalInQ) + upperq
	outlierLowerThreshold <- lowerq - (iqr * numberOfIqrTimesForOutlierRemovalInQ)

	# Eliminamos los datos extremos
	dfQData <- dfQData[dfQData$Q < outlierUpperThreshold & dfQData$Q > outlierLowerThreshold,]

	return(dfQData)
}

# ------------------------------------------------------------------------------------------------------------
# Obtiene el punto crítico en el estadístico Q donde se considera que cambia el estado del actuador.
```

```
# Este punto será aquél para el cual se han producido un número determinado de puntos por encima del límite
superior
# Parámetros:
#  - dfQData: dataframe con los datos del estadístico Q total (referencia y prueba)
#  - upperQLimit: límite superior de Hotelling
#  - repetitionsAboveThresholdsForCriticalPoint: número de datos consecutivos de Q por encima del
#    valor límite para determinar el punto crítico
# Resultado: vector con todos los puntos en donde hay un número mínimo de datos que pasan por encima del valor
límite
# ------------------------------------------------------------------------------------------------------------

getQCriticalPoints <- function(dfQData, upperQLimit, repetitionsAboveThresholdsForCriticalPoint) {

        # Posición del punto crítico
        criticalPoints <- -1

        # A partir del estadístico Q, obtenemos un vector con las siguientes características:
        # * Valor de 1 para los datos por encima del valor del límite
        # * Valor de 0 para los datos sean iguales o que estén por debajo del valor del límite
        data <- ifelse(dfQData$Q > upperQLimit, 1, 0)

        # Generamos los grupos con datos consecutivos
        runs = rle(data)

        # Buscamos grupos con repeticiones mayores de un valor
        runsOfInterest      =      which(runs$values      ==      TRUE      &      runs$lengths      >=
repetitionsAboveThresholdsForCriticalPoint)
        if(length(runsOfInterest) > 0) {

                runs.lengths.cumsum = cumsum(runs$lengths)

                newindex = ifelse(runsOfInterest>1, runsOfInterest-1, 0)
                criticalPoints = runs.lengths.cumsum[newindex] + 1
                if (0 %in% newindex){
                        criticalPoints = c(1,criticalPoints)
                }
        }

        return(criticalPoints)
}

# ------------------------------------------------------------------------------------------------------------
# Crea el archivo con los datos del estadístico Q y lo carga en Azure
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - fileName: nombre del archivo de los descriptores
#  - features: dataframe con los descriptores
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
#  - azureContainerName: nombre del contenedor de Azure a donde se suben los datos
# Resultado: carga de los descriptores en Azure
# ------------------------------------------------------------------------------------------------------------

createAnalysisResultsFileAndUploadOnAzure  <-  function(testCampaignId,  testTypeId,  fileName,  fileData,
tempFolder, azureContainerName) {

        # Si no existe el directorio temporal, se crea
        dir.create(tempFolder, showWarnings = FALSE)

        # Comprobaciones en los datos
        if(nrow(fileData) != 0) {

                print(paste0("Creating descriptor file: ", fileName))

                # Creamos el archivo en la ubicación temporal
                completeFileName <- file.path(tempFolder, fileName)

                # Creamos las subcarpetas del archivo
                fileFolder <- dirname(completeFileName)
                dir.create(fileFolder, recursive = TRUE)

                # Creamos el archivo en la ubicación temporal
                write.csv(fileData, completeFileName, row.names = FALSE)

                # Subimos el archivo a Azure
                print(paste0("Uploading file on Azure: ", fileName))

                if(ubuntuVirtualMachine == TRUE) {

                        cmd = paste0("/home/adminssh/isselub/azcopy copy \"",
                                                tempFolder, "/*\" ",
                                                "\"https://stoisselubfzg.blob.core.windows.net/",
                                                azureContainerName,                         "?sv=2019-02-
02&ss=b&srt=sco&sp=rwdlac&se=2040-04-27T19:17:18Z&st=2020-04-
27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\" --recursive")
```

```
                        system(cmd)
                }
                else {

                        cmd = paste0("azcopy copy \"",
                                                tempFolder, "/*\" ",
                                                "\"https://stoisselubfzg.blob.core.windows.net/",
                                                azureContainerName,                         "?sv=2019-02-
02&ss=b&srt=sco&sp=rwdlac&se=2040-04-27T19:17:18Z&st=2020-04-
27T11:17:18Z&spr=https&sig=YGtjD%2BGpdRhfHi1mgf9hdOVSS4pOfalg1tGAWkIs4Sk%3D\" --recursive")

                        system("cmd.exe", input = cmd)
                }

                print(paste0("File uploaded: ", fileName))

                # Borramos la carpeta temporal y todo su contenido
                unlink(tempFolder, recursive = TRUE)
        }
}

# ------------------------------------------------------------------------------------------------------------
# Realiza el análisis de los descriptores entre dos valores de ciclos determinados.
# Sirve para poder separar los dos análisis realizados en el trabajo.
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
#  - initialCycles: nº de ciclos inicial para realizar el análisis
#  - finalCycles: nº de ciclos final para realizar el análisis
#  - ldaAnalysis: variable booleana para hacer analisis LDA (TRUE) o no (FALSE)
#  - analysisResultsFileName: nombre del archivo con los resultados que se subirán en Azure
#  - analysisFeaturesFileName: nombre del archivo con los descriptores a partir de los cuales se obtienen los
valores de Q
#  - ldaFileName: nombre del archivo RDS para serializar el modelo LDA
#  - azureContainerName: nombre del contenedor de Azure a donde se suben los datos
# Resultado: carga en azure del archivo con los descriptores
# ------------------------------------------------------------------------------------------------------------

analyseFeaturesBetweenCycles <- function(testCampaignId,
                                                                                        testTypeId,
                                                                                        tempFolder,
                                                                                        initialCycles,
                                                                                        finalCycles,
                                                                                        ldaAnalysis,

analysisResultsFileName,

analysisFeaturesFileName,
                                                                                        ldaFileName,
                                                                                        azureContainerName) {

        # Datos de configuración
        percentageOfInitialDataForTraining <- 0.1
        retainedPCs <- 10
        repetitionsAboveThresholdsForCriticalPoint <- 5
        alphaValue <- 0.005
        removeOutliersInQ <- TRUE
        numberOfIqrTimesForOutlierRemovalInQ <- 3
        removeBenchRestartsInitialData <- TRUE
        benchStopTimeInSeconds <- 3600
        timeWindowInSecondsToRemoveInBenchRestarts <- 3600
        plotByPoints <- T
        plotByDatetime <- F
        plotByCycles <- F

        print(analysisResultsFileName)
        print(analysisFeaturesFileName)


        # ---------------------------------------------------------------------
        # Descargamos el archivo de descriptores
        # ---------------------------------------------------------------------

        featuresFileName = paste0(getTestCampaignName(testCampaignId),
                                                        "/", getTestTypeName(testTypeId),
                                                        "/Features/features.csv")
        featuresData <- getFileData(testCampaignId, testTypeId, featuresFileName)

        # Ordenación de los datos
        featuresData <- featuresData[order(featuresData$Cycles),]

        # Filtrado de datos      de ciclos.
        # Esto se hace para distinguir los dos análisis que se van a realizar
        if(is.null(finalCycles) == TRUE) {
                featuresData <- featuresData[which(featuresData$Cycles >= initialCycles),]
```

```
        }
        else {
                featuresData     <-     featuresData[which(featuresData$Cycles     >=     initialCycles     &
featuresData$Cycles <= finalCycles),]
        }

        # ---------------------------------------------------------------------
        # Transformaciones en el archivo de descriptores
        # ---------------------------------------------------------------------

        # Se eliminan las posibles filas que tengan NA
        featuresData <- featuresData[ complete.cases(featuresData),]

        # Transformamos la columna "Datetime" a POSIXct
        featuresData$Datetime <- as.POSIXct(featuresData$Datetime)

        # Eliminamos las ventanas de descriptores después de los reinicios del banco de ensayos
        if(removeBenchRestartsInitialData == TRUE) {
                featuresData <- removeFeaturesAfterBenchRestarts(featuresData,
                                                                                                benchStopTimeInSeconds,

        timeWindowInSecondsToRemoveInBenchRestarts)
        }

        # ---------------------------------------------------------------------
        # Selección del dataset de referencia y el dataset de pruebas
        # ---------------------------------------------------------------------

        # Columnas de los datos que no son descriptores
        nonfeaturesNames <- c("ResultId","Cycles","Datetime","InitialTimestamp","FinalTimestamp")

        trainEnd <- as.integer(nrow(featuresData) * percentageOfInitialDataForTraining)
        trainRowVector <- c(1:trainEnd)
        testStart <- trainEnd + 1
        testRowVector <- c(testStart:nrow(featuresData))

        # Dataset de referencia y dataset de pruebas
        trainData <- featuresData[trainRowVector, !(names(featuresData) %in% nonfeaturesNames)]
        testData <- featuresData[testRowVector, !(names(featuresData) %in% nonfeaturesNames)]


        print(paste0("N° de datos del dataset de referencia: ", nrow(trainData)))
        print(paste0("N° de datos del dataset de prueba: ", nrow(testData)))
        print(paste0("N° de datos totales: ", nrow(featuresData)))

        # ---------------------------------------------------------------------
        # Cálculo de las componentes principales a partir de los datos de referencia.
        # Además, se calcula el escalado y el centrado de los datos
        # ---------------------------------------------------------------------

        pcaTrainData <- prcomp(trainData, center = TRUE, scale. = TRUE)

        # Variabilidad de las componentes principales
        # Hay que decidir con qué cantidad de componentes principales nos vamos a quedar
        #get_eigenvalue(pcaTrainData)
        #fviz_screeplot(pcaTrainData, addlabels = TRUE, ncp = 20)

        # ---------------------------------------------------------------------------
        # Dataset de prueba y de test con las componentes principales seleccionadas
        # ---------------------------------------------------------------------------

        # Selección del conjunto de componentes principales
        trainData <- pcaTrainData$x[, 1:retainedPCs]

        # Proyectamos los datos de test usando predict y el objeto pca
        # Esta acción realizará también el escalado y centrado.
        # Por último, nos quedamos con las componentes principales seleccionadas
        testData <- predict(pcaTrainData, newdata = testData)[ , 1:retainedPCs]

        # ------------------------------------------------
        # Contribución de los descriptores a las componentes
        # ------------------------------------------------

        # Contribución de los descriptores a las componentes principales
        # Dimensión 1 y dimensión 2
        #fviz_contrib(pcaTrainData, choice = "var", axes = 1, top = 10)
        #fviz_contrib(pcaTrainData, choice = "var", axes = 2, top = 10)

        # ------------------------------------------------
        # Hotelling - Fase 1
        # ------------------------------------------------

        # Cálculo del estadístico Q, a partir de un valor de alfa
        Q <- Q_limits(trainData, alpha = alphaValue)

        # Fase 1:
```

```
        #   - Cálculo del estadístico Q para los datos de referencia, a partir de un valor de alfa
        #       - Cálculo de los límites.
        #   - Eliminación de datos extremos
        #   - Cálculo de nuevo de los límites de los datos de referencia

        # Mostramos un gráfico con los valores Q para los datos de referencia, junto con los límites del
estadístico
        #plot(Q$Q_values)
        #abline(h = Q$UCL, col = 'red')
        #abline(h = Q$LCL, col = 'red')

        # Obtenemos los datos extremos (puntos por encima de límite superior)
        outliers <- Q$Q_values > Q$UCL

        # Eliminamos los datos extremos de los datos de entrenamiento
        trainData <- trainData[!outliers,]

        # Cálculo de nuevo de Q sin los datos extremos
        Q <- Q_limits(trainData, alpha = alphaValue)

        # Dataset con Q de referencia y metadatos
        datetime <- featuresData$Datetime
        datetime <- datetime[trainRowVector]
        datetime <- datetime[!outliers]
        cycles <- featuresData$Cycles
        cycles <- cycles[trainRowVector]
        cycles <- cycles[!outliers]
        results <- featuresData$ResultId
        results <- results[trainRowVector]
        results <- results[!outliers]
        dfQReferenceData <- data.frame(Q=Q$Q_values, Datetime=datetime, Cycles=cycles, ResultId=results)

        # Mostramos de nuevo la información de referencia sin los datos extremos
        if(ubuntuVirtualMachine == FALSE) {
                if(plotByPoints) {
                        plot(dfQReferenceData$Q)
                }
                else if(plotByDatetime) {
                        plot(y=dfQReferenceData$Q, x=dfQReferenceData$Datetime)
                }
                else if(plotByCycles){
                        plot(y=dfQReferenceData$Q, x=dfQReferenceData$Cycles)
                }
                abline(h = Q$UCL, col = 'red')
                abline(h = Q$LCL, col = 'red')
        }
        # ------------------------------------------------
        # Hotelling - Fase 2
        # ------------------------------------------------

        # Cálculo del valor Q en los datos de prueba
        Q_test <- apply(testData, 1, FUN = Qj, Q$x_hat, Q$S)

        # Dataset con Q de prueba y metadatos
        datetime <- featuresData$Datetime
        datetime <- datetime[testRowVector]
        cycles <- featuresData$Cycles
        cycles <- cycles[testRowVector]
        results <- featuresData$ResultId
        results <- results[testRowVector]
        dfQTestData <- data.frame(Q=Q_test, Datetime=datetime, Cycles=cycles, ResultId=results)

        # ------------------------------------------------
        # Hotelling - Gráfico
        # ------------------------------------------------

        # Creamos un único gráfico con el estadístico Q para los datos de referencia y de prueba
        dfQData <- rbind(dfQReferenceData,dfQTestData)

        # Eliminamos los datos extremos
        if(removeOutliersInQ == TRUE) {
                dfQData <- removeQDataOutliers(dfQData, numberOfIqrTimesForOutlierRemovalInQ)
        }

        # Obtención del punto crítico en el estadístico Q
        criticalPoints <- getQCriticalPoints(dfQData,
                                                                                Q$p_UCL,

repetitionsAboveThresholdsForCriticalPoint)
        # Dibujamos el gráfico
        if(ubuntuVirtualMachine == FALSE) {

                if(plotByPoints) {
                        plot(dfQData$Q, type="l")
                        abline(v = testStart, lty = 2)
```

```
                        if(length(criticalPoints) > 0) {
                                abline(v = criticalPoints[1], col = 'red')
                        }
                }
                else if(plotByDatetime) {
                        plot(y=dfQData$Q, x=dfQData$Datetime)
                        abline(v = dfQData$Datetime[testStart], lty = 2)

                        if(length(criticalPoints) > 0) {
                                abline(v = dfQData$Datetime[criticalPoints[1]], col = 'red')
                        }
                }
                else if(plotByCycles){
                        plot(y=dfQData$Q, x=dfQData$Cycles)
                        abline(v = dfQData$Cycles[testStart], lty = 2)

                        if(length(criticalPoints) > 0) {
                                abline(v = dfQData$Cycles[criticalPoints[1]], col = 'red')
                        }
                }

                if(plotByPoints || plotByDatetime || plotByCycles) {
                        abline(h = Q$p_UCL, col = 'red')
                        abline(h = Q$p_LCL, col = 'red')

                        print(paste0("Límite superior: ", Q$p_UCL))
                        print(paste0("Límite inferior: ", Q$p_LCL))
                }
        }

        if(length(criticalPoints) > 0) {

                print(paste0("Primer punto crítico: ", criticalPoints[1]))
                print(paste0("Número de puntos críticos: ", length(criticalPoints)))
                print(paste0("Fecha del primer punto crítico: ", dfQData$Datetime[criticalPoints[1]]))
                print(paste0("Ciclos del primer punto crítico: ", dfQData$Cycles[criticalPoints[1]]))

        }

        # ------------------------------------------------
        # LDA
        # ------------------------------------------------

        if(ldaAnalysis == TRUE) {

                # Filtramos los descriptores con los datos del dataframe de Q, uniendo por "Cycles".
                # Finalmente, eliminamos las columnas no necesarias para el entrenamiento
                #toRemove <- c("ResultId","Datetime","InitialTimestamp","FinalTimestamp")
                featuresData <- featuresData[order(featuresData$Cycles),]
                totalData   <-   featuresData[,   !(names(featuresData)   %in%   c("ResultId","Datetime",
"InitialTimestamp", "FinalTimestamp"))]
                totalData <- merge(dfQData, totalData, by=c("Cycles"))
                totalData <- totalData[, !(names(totalData) %in% c("Datetime","Cycles", "Q", "ResultId"))]

                # Datos no degradados.
                # Una vez seleccionados los escalamos y forzamos la conversión de scale a dataframe.

                nonDegradedData <- totalData[(1:testStart),]
                # Código para filtrar datos de outliers en el segundo análisis
                #nonDegradedData <- nonDegradedData[nonDegradedData$Clf_FR_Accelerometer_Z< 38,]
                #nonDegradedData <- nonDegradedData[nonDegradedData$Imf_RF_Accelerometer_Z< 40,]

                nonDegradedNonScaledData <- nonDegradedData
                nonDegradedData <- scale(nonDegradedData, center = TRUE, scale = TRUE)

                # Datos degradados.
                degradedData  <-  totalData[(criticalPoints[1]:(criticalPoints[1]+nrow(nonDegradedData))),]

                degradedNonScaledData <- degradedData
                degradedData <- scale(degradedData, center = attr(nonDegradedData,'scaled:center'), scale =
attr(nonDegradedData,'scaled:scale'))
                degradedData <- as.data.frame(degradedData)

                # Agregamos columna de estado
                nonDegradedData <- as.data.frame(nonDegradedData)
                nonDegradedData['Status'] = "non-degraded"
                nonDegradedNonScaledData <- as.data.frame(nonDegradedNonScaledData)
                nonDegradedNonScaledData['Status'] = "non-degraded"
                degradedData <- as.data.frame(degradedData)
                degradedData['Status'] = "degraded"
                degradedNonScaledData <- as.data.frame(degradedNonScaledData)
                degradedNonScaledData['Status'] = "degraded"

                # Dataset de entrenamiento para LDA
                totalData <- rbind(nonDegradedData, degradedData)
                totalData <- as.data.frame(totalData)
```

```
                totalNonScaledData <- rbind(nonDegradedNonScaledData, degradedNonScaledData)

                totalNonScaledData <- as.data.frame(totalNonScaledData)

                # Ejecutamos LDA
                View(totalData)
                browser()
                formulaLda <- formula("Status~.")
                model <- lda(formulaLda, data=totalData)

                #Código para ejecutar cross-validation
                #model <- lda(formulaLda, data=totalData, CV=TRUE)
                #table(model$class, totalData$Status)

                # Obtenemos los coeficientes LD1 del analysis LDA
                ldaResults <- abs(model$scaling)
                ldaResults <- ldaResults[order(ldaResults, decreasing = TRUE),]


                # Mostramos la parte no degradada y degradada de señal con más cambio en LDA

                #plot(totalNonScaledData$Shf_FR_Motor_current_1, type="l")
                #abline(v = testStart, lty = 2)
                #df1                    <-                    data.frame(Status=totalNonScaledData$Status,
Shf_FR_Motor_current_1=totalNonScaledData$Shf_FR_Motor_current_1)
                #mu <- ddply(df1, "Status", summarise, grp.mean=mean(Shf_FR_Motor_current_1))

                #p<-ggplot(df1, aes(x=Shf_FR_Motor_current_1, color=Status)) +
                #  geom_density()+
                #  geom_vline(data=mu, aes(xintercept=grp.mean, color=Status),
                #                         linetype="dashed")
                #p
        }

        # ------------------------------------------------
        # Generación de archivo csv con datos de Q
        # ------------------------------------------------

        # Vector con todas las posiciones a FALSE
        alarms <- logical(nrow(dfQData))

        # Ponemos a TRUE si es una posición de alarma
        if(length(criticalPoints) > 0) {
                alarms[criticalPoints] <- TRUE
        }

        # Subimos el archivo con los datos del estadístico Q a Azure
        createAnalysisResultsFileAndUploadOnAzure(testCampaignId,
                                                                                                testTypeId,

analysisResultsFileName,

cbind(Index=c(1:nrow(dfQData)),dfQData,Alarms=alarms),
                                                                                                tempFolder,

azureContainerName)

        # Filtramos los descriptores correspondientes a los valores del estadístico Q
        featuresData <- featuresData[featuresData$Cycles %in% dfQData$Cycles, ]

        # Subimos los datos a Azure
        createAnalysisResultsFileAndUploadOnAzure(testCampaignId,
                                                                                                testTypeId,

analysisFeaturesFileName,

cbind(Index=c(1:nrow(dfQData)), featuresData, Alarms=alarms),
                                                                                                tempFolder,

azureContainerName)
}

# ------------------------------------------------------------------------------------------------------------
# Realiza el análisis de los descriptores
# Parámetros:
#  - testCampaignId: identificador de campaña de ensayos
#  - testTypeId: tipo de test
#  - tempFolder: carpeta temporal para generar el archivo con los descriptores antes de subirlo a Azure
# Resultado: carga en azure del archivo con los descriptores
# ------------------------------------------------------------------------------------------------------------

analyseFeatures <- function(testCampaignId, testTypeId, tempFolder) {

        # Primer análisis (antes del fallo del banco)
        initialCycles <- 16562
```

```
        finalCycles <- 1574998
        analysisResultsFileName <- "analysis1.csv"
        analysisFeaturesFileName <- "features1.csv"
        ldaFileName <- "lda1.rds"
        ldaAnalysis <- TRUE
        azureContainerName <- "analysis1"
        analyseFeaturesBetweenCycles(testCampaignId, testTypeId, tempFolder, initialCycles, finalCycles,
ldaAnalysis, analysisResultsFileName, analysisFeaturesFileName, ldaFileName, azureContainerName)

        # Segundo análisis (después del fallo del banco)
        initialCycles <- 1576492
        finalCycles <- NULL
        analysisResultsFileName <- "analysis2.csv"
        analysisFeaturesFileName <- "features2.csv"
        ldaFileName <- "lda2.rds"
        ldaAnalysis <- TRUE
        azureContainerName <- "analysis2"
        #analyseFeaturesBetweenCycles(testCampaignId, testTypeId, tempFolder, initialCycles, finalCycles,
ldaAnalysis, analysisResultsFileName, analysisFeaturesFileName, ldaFileName, azureContainerName)
}

# ------------------------------------------------------------------------------------------
# Genera la serie del gráfico de Hotelling
# ------------------------------------------------------------------------------------------

processData <- function() {

        testCampaignId <- 3
        testTypeId <- 3

        if(ubuntuVirtualMachine == FALSE) {

                tempFolder <- "D:/Users/rgonzalez/Documents/Datos/temp/TFM/Archivos/temp"
        }
        else {

                tempFolder <- "/home/adminssh/isselub/temp"
        }

        analyseFeatures(testCampaignId, testTypeId, tempFolder)
        print("Feature analysis finished")
}

processData()
```

9.5 Complete List of Features

A total of 176 types of features have been generated during the process. Here is the list of these features grouped by the type of signal:

Table 11: Complete list of features

SIGNAL	FEATURES Raising Edge - Falling Edge	FEATURES Falling Edge – Raising Edge
Accelerometer_X	Rms_RF_Accelerometer_X	Rms_FR_Accelerometer_X
	Median_RF_Accelerometer_X	Median_FR_Accelerometer_X
	Pk2Pk_RF_Accelerometer_X	Pk2Pk_FR_Accelerometer_X
	Pv_RF_Accelerometer_X	Pv_FR_Accelerometer_X
	Ku_RF_Accelerometer_X	Ku_FR_Accelerometer_X
	Crf_RF_Accelerometer_X	Crf_FR_Accelerometer_X
	Clf_RF_Accelerometer_X	Clf_FR_Accelerometer_X
	Imf_RF_Accelerometer_X	Imf_FR_Accelerometer_X
	Shf_RF_Accelerometer_X	Shf_FR_Accelerometer_X
	Max_RF_Accelerometer_X	Max_FR_Accelerometer_X
	Min_RF_Accelerometer_X	Min_FR_Accelerometer_X

SIGNAL	FEATURES Raising Edge - Falling Edge	FEATURES Falling Edge – Raising Edge
Accelerometer_Y	Rms_RF_Accelerometer_Y Median_RF_Accelerometer_Y Pk2Pk_RF_Accelerometer_Y Pv_RF_Accelerometer_Y Ku_RF_Accelerometer_Y Crf_RF_Accelerometer_Y Clf_RF_Accelerometer_Y Imf_RF_Accelerometer_Y Shf_RF_Accelerometer_Y Max_RF_Accelerometer_Y Min_RF_Accelerometer_Y	Rms_FR_Accelerometer_Y Median_FR_Accelerometer_Y Pk2Pk_FR_Accelerometer_Y Pv_FR_Accelerometer_Y Ku_FR_Accelerometer_Y Crf_FR_Accelerometer_Y Clf_FR_Accelerometer_Y Imf_FR_Accelerometer_Y Shf_FR_Accelerometer_Y Max_FR_Accelerometer_Y Min_FR_Accelerometer_Y
Accelerometer_Z	Rms_RF_Accelerometer_Z Median_RF_Accelerometer_Z Pk2Pk_RF_Accelerometer_Z Pv_RF_Accelerometer_Z Ku_RF_Accelerometer_Z Crf_RF_Accelerometer_Z Clf_RF_Accelerometer_Z Imf_RF_Accelerometer_Z Shf_RF_Accelerometer_Z Max_RF_Accelerometer_Z Min_RF_Accelerometer_Z	Rms_FR_Accelerometer_Z Median_FR_Accelerometer_Z Pk2Pk_FR_Accelerometer_Z Pv_FR_Accelerometer_Z Ku_FR_Accelerometer_Z Crf_FR_Accelerometer_Z Clf_FR_Accelerometer_Z Imf_FR_Accelerometer_Z Shf_FR_Accelerometer_Z Max_FR_Accelerometer_Z Min_FR_Accelerometer_Z
Cylinder_force_filtered	Rms_RF_Cylinder_force_filtered, Median_RF_Cylinder_force_filtered, Pk2Pk_RF_Cylinder_force_filtered, Pv_RF_Cylinder_force_filtered, Ku_RF_Cylinder_force_filtered, Crf_RF_Cylinder_force_filtered, Clf_RF_Cylinder_force_filtered Imf_RF_Cylinder_force_filtered Shf_RF_Cylinder_force_filtered Max_RF_Cylinder_force_filtered Min_RF_Cylinder_force_filtered	Rms_FR_Cylinder_force_filtered Median_FR_Cylinder_force_filtered Pk2Pk_FR_Cylinder_force_filtered Pv_FR_Cylinder_force_filtered Ku_FR_Cylinder_force_filtered Crf_FR_Cylinder_force_filtered Clf_FR_Cylinder_force_filtered Imf_FR_Cylinder_force_filtered Shf_FR_Cylinder_force_filtered Max_FR_Cylinder_force_filtered Min_FR_Cylinder_force_filtered
EMA_Force	Rms_RF_EMA_Force Median_RF_EMA_Force Pk2Pk_RF_EMA_Force Pv_RF_EMA_Force Ku_RF_EMA_Force Crf_RF_EMA_Force Clf_RF_EMA_Force Imf_RF_EMA_Force Shf_RF_EMA_Force Max_RF_EMA_Force Min_RF_EMA_Force	Rms_FR_EMA_Force Median_FR_EMA_Force Pk2Pk_FR_EMA_Force Pv_FR_EMA_Force Ku_FR_EMA_Force Crf_FR_EMA_Force Clf_FR_EMA_Force Imf_FR_EMA_Force Shf_FR_EMA_Force Max_FR_EMA_Force Min_FR_EMA_Force

SIGNAL	FEATURES Raising Edge - Falling Edge	FEATURES Falling Edge – Raising Edge
Motor_current_1	Rms_RF_Motor_current_1 Median_RF_Motor_current_1 Pk2Pk_RF_Motor_current_1 Pv_RF_Motor_current_1 Ku_RF_Motor_current_1 Crf_RF_Motor_current_1 Clf_RF_Motor_current_1 Imf_RF_Motor_current_1 Shf_RF_Motor_current_1 Max_RF_Motor_current_1 Min_RF_Motor_current_1	Rms_FR_Motor_current_1 Median_FR_Motor_current_1 Pk2Pk_FR_Motor_current_1 Pv_FR_Motor_current_1 Ku_FR_Motor_current_1 Crf_FR_Motor_current_1 Clf_FR_Motor_current_1 Imf_FR_Motor_current_1 Shf_FR_Motor_current_1 Max_FR_Motor_current_1 Min_FR_Motor_current_1
Motor_current_2	Rms_RF_Motor_current_2 Median_RF_Motor_current_2 Pk2Pk_RF_Motor_current_2 Pv_RF_Motor_current_2 Ku_RF_Motor_current_2 Crf_RF_Motor_current_2 Clf_RF_Motor_current_2 Imf_RF_Motor_current_2 Shf_RF_Motor_current_2 Max_RF_Motor_current_2 Min_RF_Motor_current_2	Rms_FR_Motor_current_2 Median_FR_Motor_current_2 Pk2Pk_FR_Motor_current_2 Pv_FR_Motor_current_2 Ku_FR_Motor_current_2 Crf_FR_Motor_current_2 Clf_FR_Motor_current_2 Imf_FR_Motor_current_2 Shf_FR_Motor_current_2 Max_FR_Motor_current_2 Min_FR_Motor_current_2
Stober_Speed_Meassurement	Rms_RF_Stober_Speed_Meassurement Median_RF_Stober_Speed_Meassurement Pk2Pk_RF_Stober_Speed_Meassurement Pv_RF_Stober_Speed_Meassurement Ku_RF_Stober_Speed_Meassurement Crf_RF_Stober_Speed_Meassurement Clf_RF_Stober_Speed_Meassurement Imf_RF_Stober_Speed_Meassurement Shf_RF_Stober_Speed_Meassurement Max_RF_Stober_Speed_Meassurement Min_RF_Stober_Speed_Meassurement	Rms_FR_Stober_Speed_Meassurement Median_FR_Stober_Speed_Meassurement Pk2Pk_FR_Stober_Speed_Meassurement Pv_FR_Stober_Speed_Meassurement Ku_FR_Stober_Speed_Meassurement Crf_FR_Stober_Speed_Meassurement Clf_FR_Stober_Speed_Meassurement Imf_FR_Stober_Speed_Meassurement Shf_FR_Stober_Speed_Meassurement Max_FR_Stober_Speed_Meassurement Min_FR_Stober_Speed_Meassurement