

POLITECNICO DI TORINO

Corso di laurea magistrale in
Mechatronic Engineering

Master Thesis

*Solenoid Valve Features Detection
for Predictive Maintenance*



Supervisors:

Doc. Barth Christian
Prof. Rizzo Alessandro

Candidate:

Berti Carlo Antonio

Academic year: 2019/2020

Abstract

In this elaborate a system for measuring, storing and processing voltage and current quantities in a solenoid valve has been developed. This system will be used in a bigger project that has the goal of doing a sensor-less monitoring of solenoid valves in order to do predictive maintenance on these components.

In the implementation, a microcontroller was used for performing the measurements and the LwIP library was used for a lightweight implementation of a UDP/IP communication for exporting the data in almost-real-time. Through the use of a Raspberry Pi, a first data manipulation of the data is performed and a secure communication with the AWS platform has been implemented. Using Amazon Cloud-based applications the data are stored and finally processed. The result of the complete data processing is the computation of the linked flux in the coil of the valve and then the detection the features that characterize the shape of the transients of the flux and the current during a complete switching cycle. The collected and processed quantities in this system are obtained from the valve just using the microcontroller and simple electronics without adding any sensors.

The system has been used for collecting data during a first endurance test on a solenoid valve. From the results analysis it was possible to observe how the detected features evolved during the valve aging. Some evaluations of the temperature effects on the measurements and of possible improvement of the accuracy of the detection are done.

The topics presented in this thesis are useful to understand which information are obtainable from a solenoid valve without the usage of additional sensors and how to retrieve, process and store them. This information can be used for doing predictive maintenance on those components by finding a relation between the evolution of the detected features and the decline of the health status of the valve itself.

Table of contents

1	Introduction.....	1
1.1	Introduction to Predictive Maintenance	1
1.2	Motivations.....	2
1.3	Overview on the complete project.....	2
1.4	Thesis task.....	4
2	Background.....	5
2.1	Electromagnetic quantities in a solenoid valve	5
2.1.1	Electromagnetism principles	5
2.1.1	Solenoid valve and Weber-Ampère characteristic.....	9
2.2	Statistics concepts.....	12
2.2.1	Standard deviation	12
2.2.2	Linear regression	12
2.3	Computer communication protocols	15
2.3.1	TCP/IP	15
2.3.2	Security.....	17
2.3.3	LwIP stack.....	19
2.3.4	MQTT concepts.....	21
2.4	Amazon Web Services	22
2.4.1	Cloud computing concepts	22
2.4.2	AWS API.....	23
2.4.3	AWS Security concepts.....	24
3	Setup and procedures	25
3.1	Architecture overview	25
3.2	FESTO VYKA valve.....	26
3.3	Embedded system for measurements and data generation	27
3.3.1	Electronic circuit.....	28
3.3.2	Microcontroller software task.....	29
3.3.3	Microcontroller communication	29
3.3.4	Board routine	34
3.4	Raspberry Pi	35
3.4.1	Flux computation.....	35
3.4.2	Node Red.....	38
3.5	Amazon Web Services setup.....	43

3.5.1	AWS IoT Core.....	43
3.5.2	AWS Simple Storage Service (S3).....	45
3.5.3	AWS Lambda.....	45
3.5.4	Dynamo DB.....	48
3.6	Key features and detection procedure.....	49
3.6.1	Detected features.....	49
3.6.2	Detection procedure.....	50
4	Endurance test results.....	53
4.1	Introduction.....	53
4.1.1	VYKA valve a-priori information and working condition.....	53
4.1.2	W-A aging evolution types.....	53
4.1.3	Endurance test regions.....	54
4.2	W-A evolution overview.....	55
4.3	Detected features evolution.....	55
5	Discussion.....	59
5.1	Temperature effect.....	59
5.1.1	Temperature dependencies.....	59
5.1.2	Temperature and aging effects comparison.....	60
5.1.3	Temperature effect compensation.....	61
5.2	Detection accuracy limitations.....	64
5.2.1	Flux and current coordinates.....	64
5.2.2	Quantization effect.....	65
5.2.3	Possible detection improvement.....	66
6	Summary.....	67
7	References.....	68
8	Appendix A: hardware setup for LwIP.....	70
9	Appendix B: eNet.c.....	73
10	Appendix C: Lambda Role settings.....	75
11	Appendix D: lambda_handler.....	76
12	Appendix E: System scaling.....	77

List of figures

Figure 1.1: classification of maintenance techniques [3]	1
Figure 1.2: Flux-Current (a) and Current-Time (b) plots during a complete switching cycles	3
Figure 1.3: overall project steps	3
Figure 2.1: magnetic field generated by a current flowing in a linear conductor.....	5
Figure 2.2: magnetic flux generated by a current flowing in a solenoid	6
Figure 2.3: first magnetization curve for a ferromagnetic material [4]	6
Figure 2.4: hysteresis cycle shape [4].....	7
Figure 2.5 : magnetic circuit (a), fringing effect (b) [4]	7
Figure 2.6: electromotive force sign convention [4]	8
Figure 2.7: 2/2 Solenoid valve elements	9
Figure 2.8: solenoid valve working principle.....	9
Figure 2.9: Current transient during an ON switch	10
Figure 2.10:hysteresis cycles when the plunger is blocked in two extreme positions [5].....	10
Figure 2.11: Weber-Ampère characteristic of a solenoid valve during a complete switching cycle [6].....	11
Figure 2.12: current transient during a complete switching cycle.....	11
Figure 2.13: linear regression [21]	13
Figure 2.14: correlation coefficient sign [21].....	14
Figure 2.15: TCP/IP stack	16
Figure 2.16: Demultiplexing [7].....	17
Figure 2.17: Asymmetric keys encryption [7].....	18
Figure 2.18: LwIP APIs comparison	19
Figure 2.19: PBUF_RAM structure [9].....	20
Figure 2.20: publish/subscribe paradigm and broker role	21
Figure 2.21: cloud computing layers [13]	22
Figure 2.22: AWS API interface tools [15].....	23
Figure 3.1: Architecture block-diagram	26
Figure 3.2 : VYKA valve cross section.....	27
Figure 3.3: embedded system (microcontroller + PCB) for data generation.....	27
Figure 3.4: electrical circuit for voltage and current measurements and valve control.....	28
Figure 3.5 Current and voltage transient during a complete ON-OFF cycle.....	29
Figure 3.6: protocol suite recap scheme	32
Figure 3.7: package structure.....	34
Figure 3.8: Board tasks routine.....	34

Figure 3.9: Kirchhoff's voltage law for flux computation	35
Figure 3.10: range for internal resistance computation	36
Figure 3.11: Diode ON region.....	37
Figure 3.12: Weber-Ampère characteristic for the VYKA valve.....	37
Figure 3.13: Node Red main flow	38
Figure 3.14: UDP in node settings.....	38
Figure 3.15: Data Elaboration sub-flow	39
Figure 3.16: JSON structure of the test data after the elaboration.....	39
Figure 3.17: To AWS Cloud node configuration	41
Figure 3.18: switch counter sanity check sub-flow	41
Figure 3.19: watchdog subflow	42
Figure 3.20: UI real time plot subflow	42
Figure 3.21: IoT Core console.....	43
Figure 3.22: IoT device certificates generation	44
Figure 3.23: IoT device policy permissions	44
Figure 3.24: IoT rule settings	45
Figure 3.25: AWS Lambda home.....	46
Figure 3.26: peak points in W-A characteristic.....	49
Figure 3.27: path lengths during the switches	49
Figure 3.28: time delays in the switching.....	50
Figure 3.29: list of characterizing features	50
Figure 4.1: W-A aging examples.....	53
Figure 4.2: average internal resistance evolution	54
Figure 4.3: W-A aging overview.....	55
Figure 4.4: current (P1) evolution	56
Figure 4.5: current (P5) evolution	56
Figure 4.6:W-A area evolution.....	57
Figure 4.7: current variation during OFF switching evolution.....	58
Figure 5.1: resistance evolution during a thermal transient.....	59
Figure 5.2: correlation diagram current(P1) - resistance	60
Figure 5.3: comparison between aging and temperature effects	61
Figure 5.4: switch OFF time delay complete evolution	63
Figure 5.5: switch OFF time delay before and after temperature effect compensation.....	63
Figure 5.6: flux and current detection accuracy	64
Figure 5.7: quantization effect (ON delay evolution).....	65
Figure 5.8: Flux variation during the On switch	65

Figure 8.1: Clock configuration.....	70
Figure 8.2: fsl_debug_console configuration	71
Figure 8.3: add Lwip libraries to the project path	72
Figure 10.1: IAM console.....	75

List of tables

Table 3.1: LwIP lines of code analysis [9]	30
Table 3.2: test ID mapping	34
Table 4.1 : regression parameters for aging effect	56
Table 5.1: linear regression parameters for temperature correlation	60
Table 5.2: regression for aging modelling after temperature compensation	62

1 Introduction

1.1 Introduction to Predictive Maintenance

The European Standard defines maintenance as the combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function [1].

Maintenance represents one of the biggest parts of the total operating costs of all production plants. Ineffective maintenance management affects the possibility to have high quality final products and be competitive in the world market [2]. One of the main reasons for the ineffective management is the lack of data to effectively quantify and plan the actual need for maintenance operations. Maintenance scheduling has been based for many years on statistical trend data or on real failure of the equipment [2].

The maintenance management techniques can be classified into three types: corrective, preventive and predictive maintenance [3].

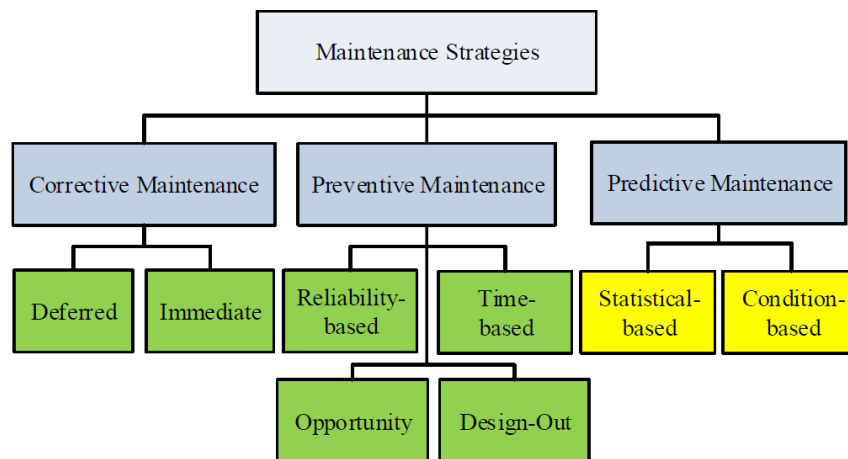


Figure 1.1: classification of maintenance techniques [3]

The logic of *corrective* (or run-to-failure) maintenance is the simplest one: maintenance operations are performed only when a failure occurs. It is a reactive management and it is the most expensive approach. The high cost of this technique is due to the need of having ready to use a high number of spare components (that lead also to high inventory costs), high machine downtime and low production availability [2].

In the *preventive* approach, maintenance tasks are based on hours of operation of the component or machine. In general, the repairs or rebuilds are scheduled based on mean-time-to-failure (MTTF) statistics for components that need a complete replacement or mean-time-between-failures (MTBF) statistics for components that need only repairing actions. The problem with this approach is that a device or machine is assumed to degrade in a time range that is typical of its classification. In other words, the mode of operation and the system specific variables are not fully taken into account although they affect the normal life duration of the components. The general result is a scheduling of maintenance that brings either to unnecessary repairs or to unwanted failures.[2]

The central idea of *predictive* maintenance (PdM) is to monitor the actual operating conditions of the equipment and use the obtained data to ensure maximum interval between repairs possible and minimize the number and the cost of unscheduled interventions due to unexpected failures. Predictive maintenance is a condition driven preventive maintenance program: it uses direct monitoring of the equipment to determine the real MTTF or loss of efficiency instead of relying on average life statistics.[2]

PdM presents some advantages if compared with other strategies: the system that requires maintenance is shut down only when it is strictly necessary; the time and the costs of the maintenance is reduced; the availability and reliability of the equipment is increased [3]. In general PdM has some drawback: the cost of the equipment used for monitoring the condition of the machines is often high; the skills level and/or the computational power needed to correctly and accurately interpret the monitoring data is high as well [1].

PdM is not always the best maintenance approach possible. Some equipment can have failures that may be more cost-effective if managed with traditional maintenance techniques. An accurate evaluation should be done when deciding the best approach for a system or machine.

1.2 Motivations

Festo is a German company founded in 1925 by Albert Fezer and Gottlieb Stoll. During the years Festo has grown as a global leader company in the development of pneumatic and electromechanical solutions for factory and process automation. Its portfolio offers a wide choice of products and services from complete systems to individual components such as valves (terminals, pilot, solenoid, etc), linear and turn actuators, compressed air preparation and supply components.

Solenoid valves are one of the most commonly used components in the industrial automation and in particular in the field of process automation in which they are used mainly for fluid flow control. The role of those components is relevant in many applications and for this reason doing predictive maintenance on solenoid valves has become a relevant objective for the producers.

As mentioned in the previous paragraph, one of the drawbacks of the predictive maintenance technique is the cost of the equipment (mainly sensors) used for the continuous monitoring of the device condition. Festo is carrying on research to add functionalities to solenoid valves without the usage of additional sensors. Among the different projects involved, one has the objective of evaluate the possibility of doing predictive maintenance on those components using the coil current and voltage as the monitoring quantities.

1.3 Overview on the complete project

This thesis is part of a project that has the objective of doing predictive maintenance on solenoid valves without the usage of additional sensors for the device monitoring. To better understand the motivations of this thesis, in this part is given an overview on the steps of the overall project.

First, the quantities that are directly measurable from a solenoid valve are the current and the voltage drop across the coil. Starting from those two measurements, it is possible to compute the magnetic flux linked to the solenoid winding. Measuring those quantities during a complete switching cycle (switch ON and OFF) of the valve, characteristic shapes are obtained (Figure 1.2). In the next chapters a detailed explanation of these two plots will be provided. For now, it is important to say that the objective of the project is to find a relation between the change of the shape of the transients in Figure 1.2 and the real decline of the status of the valve.

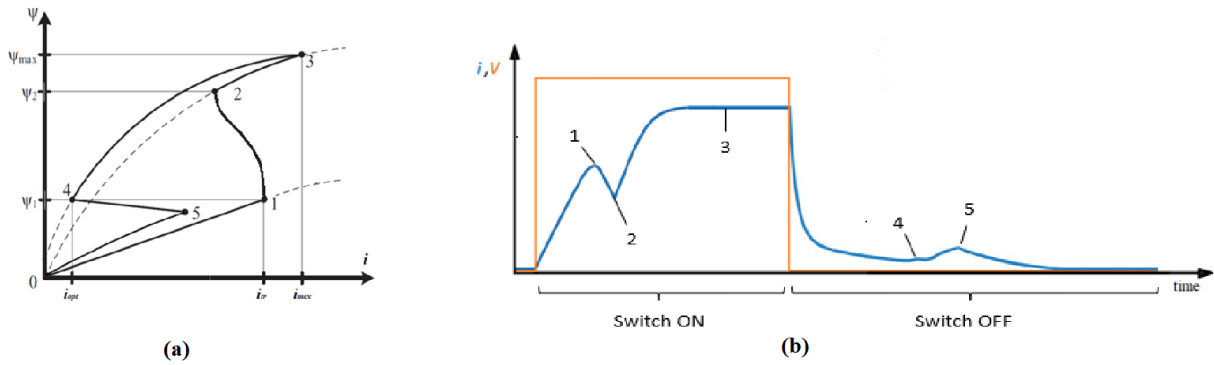


Figure 1.2: Flux-Current (a) and Current-Time (b) plots during a complete switching cycles

In general, the real conditions of the valve can be estimated through the execution of test procedure mainly regulated by company internal standards. Example of those test procedures are leakage test and the mobility test. The output of the tests are measurements sets (pressure drop across two channels of the valve, minimum static current able to move the plunger, etc.) that have to be collected and interpreted.

Traditionally, performing those test procedures requires the presence of an operator that executes the steps of the test and collects the measurements.

The first step in this project is to develop a test bench that automatically performs the test procedures and generates the data. To do this a microcontroller has to be used for controlling the sequence of the tests execution. Due to the limited resources available on a microcontroller, the generated data must be exported from it in almost real time. After that, the data will be stored and pre-processed through the usage of Cloud-based applications.

Once the setup is completed, it will be used to perform the tests during the complete aging of several valves. In this way the data collected contain the information about the evolution of the valve condition and about the “free quantities” (Figure 1.2) during the valve aging.

Once the data are collected during the endurance tests, they will be analysed using machine learning techniques in order to find the relation between the decline of the valve condition and the changes of the shape of the current and magnetic flux transients (Figure 1.2). If a relation is successfully found, a model based on the free quantities’ transients can be built and used for estimating the real working condition of the valve and doing predictive maintenance on those components.

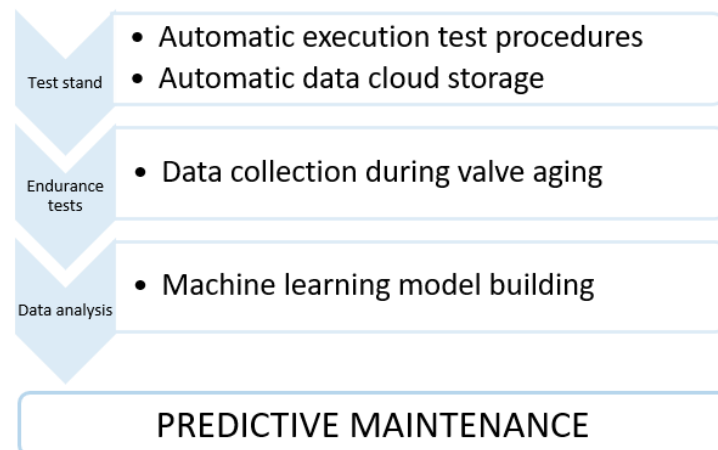


Figure 1.3: overall project steps

1.4 Thesis task

Due to the complexity of the project, a number of thesis have been offered from Festo each one with the objective of giving a contribution in the overall project. This thesis is part of the initial step is focused on the development of the architecture for the generated data exporting, storage and pre-processing.

A fundamental step for reaching the predictive maintenance objective is being able to collect the data generated from the test bench, pre-process and store them for the final analysis. It is important to evaluate in this early stage of the project how is possible to keep track of the evolution of features that characterize the shape of the free information transients obtainable from the solenoid valve ideally using low computing resources.

The objective of this elaborate is to develop a system able to export the data generated from the microcontroller in almost real time a securely send the data on the Amazon Cloud platform. Using AWS, store the results in a consistent way and detect the relevant features from every stored transient curves of the type shown in Figure 1.2.

The developed architecture will be tested running a first endurance test on a solenoid valve to evaluate the robustness of the system and the accuracy of the detection.

It is important to point out that although the architecture is used just for the collection and processing of the free information, the other test procedures can be easily implemented starting from the developed system.

2 Background

2.1 Electromagnetic quantities in a solenoid valve

In this chapter it is explained the meaning the Weber-Ampère characteristic for a solenoid valve starting from the basic magnetic field principles.

2.1.1 Electromagnetism principles

In this first section is provided a concise explanation of the electromagnetism principles needed to understand the relation among current and magnetic flux in a magnetic circuit.

2.1.1.1 Magnetic field generation and Ampère's law

A current flowing in a conductor produces a magnetic field, which can be visualized as a series of circular field lines around the wire and with intensity proportional to the inverse of the square of the distance from the conductor. Figure 2.1 shows the magnetic field lines generated by a current flowing in a linear conductor.

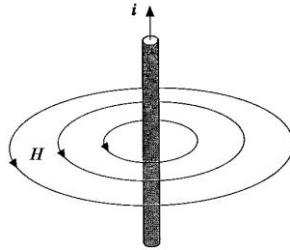


Figure 2.1: magnetic field generated by a current flowing in a linear conductor

The Ampère's law states that the line integral of the magnetic field intensity H about any closed path equals the net current enclosed by that path.

$$\oint H \cdot dl = I_{en} \quad (1)$$

If the enclosed current is generated by a coil of N turns in which is flowing a current i , then it is possible to rewrite the enclosed current as

$$I_{en} = \sum_j^{N'} i_j = i N' \quad (2)$$

With N' number of turns enclosed in the integration path, and the product $i N'$ is referred to as *magneto motive force* since is responsible for the generation of the magnetic field. The magnetic field intensity H produced by a solenoid is maximum on its central axis (Figure 2.2)

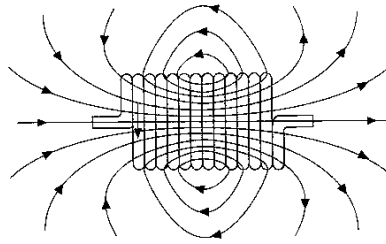


Figure 2.2: magnetic flux generated by a current flowing in a solenoid

2.1.1.2 Magnetic properties of the materials

The magnetic field intensity is in a sense an indication of the effort that the current is putting to generate it. The magnetic flux produced by the coil depends also on the material in which the field is acting. The relation between the magnetic field intensity H and the *magnetic flux density* B produced in the material is given by

$$B = \mu H \quad (3)$$

Where μ is the *magnetic permeability* of the material.

The permeability of a modern ferromagnetic material is $10^3 \div 10^4$ times bigger than the permeability of the air, but it is not constant. In Figure 2.3 the magnetic flux density B is plotted versus the magnetic field intensity H . This curve is called *magnetising curve* and is characteristic of the material. The slope of this curve is the magnetic permeability. In the initial unsaturated region of the curve the magnetic permeability is almost constant, while increasing the magnetic field intensity H the curve flattens out and presents a saturated region.

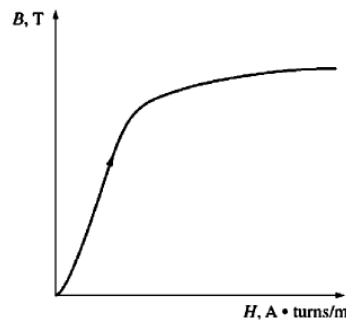


Figure 2.3: first magnetization curve for a ferromagnetic material [4]

2.1.1.3 Magnetic hysteresis

Consider again a solenoid with a ferromagnetic core. If in the winding is flowing an alternate current with a constant peak to peak amplitude, the field H and the induction B are alternate as well. Looking at Figure 2.4 when the current increases the magnetising path is *deb*, while when the current decreases the demagnetising path is *bcd*. The amount of magnetic induction B present in the core material depends on the magnetic field intensity H , but also on the previous history of the induction in the core. The complete path *bcdeb* is referred to as *hysteresis loop*. It can be shown that the area enclosed in an hysteresis loop is directly proportional to the energy loss in that cycle. This energy is used to orientate the magnetic field in the domains of the material. The first time the material is magnetised, the first magnetisation curve is *m* and corresponds to the one represented in Figure 2.4.

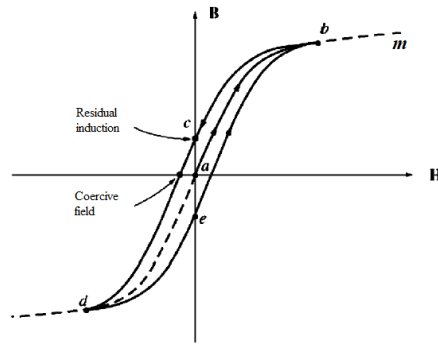


Figure 2.4: hysteresis cycle shape [4]

2.1.1.4 Magnetic flux and magnetic circuits

Given an area A , the magnetic flux through it can be computed as

$$\phi = \int_A B \cdot dA \quad (4)$$

If the magnetic induction is constant over the area A , then the flux can be easily computed as

$$\phi = B A \quad (5)$$

In Figure 2.5-a is represented a magnetic circuit in which a winding of N turns is responsible of the production of the magnetic field. It is possible to assume that all the magnetic flux produced will remain inside the core since the core is made by ferromagnetic material (neglecting the magnetic leakages), and that the air gap length l_a is short enough to avoid the fringing effect of the magnetic field (Figure 2.5-b). In this way it is possible to consider the core as a flux tube having a constant section s . Moreover, assuming a constant magnetic induction in the ferromagnetic material, the flux can be computed using (previous formula).

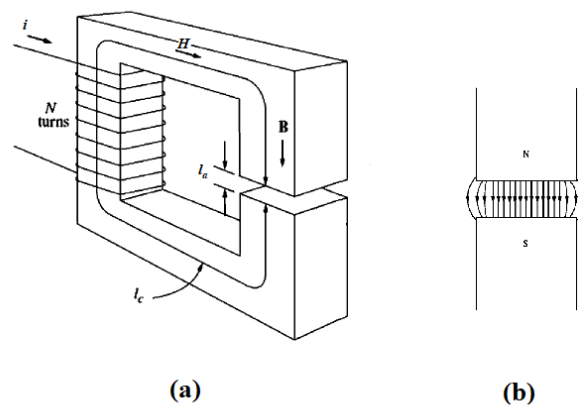


Figure 2.5 [4]: magnetic circuit (a), fringing effect (b)

Using $l = l_a + l_c$ as the path of integration for the Ampère's law(1.1), rewriting the enclosed current using (1.2) and considering constant the magnetic field intensity inside the material, the (1.1) becomes

$$H_c l_c + H_a l_a = N i \quad (6)$$

Combining now the (6) with the (3) and the (5)

$$\phi \left[\frac{l_c}{\mu_c S} + \frac{l_a}{\mu_a S} \right] = N i \quad (7)$$

Introducing the concept of *reluctance*

$$\phi [R_c + R_a] = \phi R_{tot} = N i \quad (8)$$

The (8) is called *Hopkinson law*.

It is important to notice in the (7) that for a given magnetomotive force, the flux is strongly affected by the presence of the air gap in the magnetic circuit.

2.1.1.5 Lenz's law

If in the magnetic circuit of Figure 2.5-a is circulating a flux ϕ , then this flux is linking the coil N times. The product

$$\psi = N \phi \quad (9)$$

is called *linked flux* or *flux linkage* of the coil.

The *Lenz's law* states that across the coil is generated an electromotive force that is directly proportional to the rate of change of the flux linkage

$$e_{ind} = - \frac{d\psi}{dt} \quad (10)$$

The sign of the induced electromotive force is such that it tends to generate a current that opposes to the flux variation (see Figure 2.6)

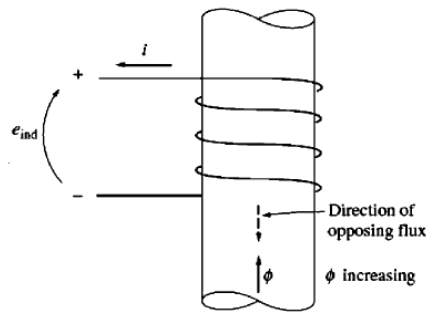


Figure 2.6: electromotive force sign convention [4]

It is possible to rewrite the (10)

$$e_{ind} = \frac{d}{dt} \left(\frac{N^2 i}{R_{tot}} \right) = \frac{d}{dt} (L i) \quad (11)$$

Where L is the *self inductance* of the solenoid.

2.1.1 Solenoid valve and Weber-Ampère characteristic

This section is dedicated to the explanation of the working principles of a solenoid valve and the meaning of the Weber-Ampère characteristic for those devices.

2.1.1.1 Solenoid valve

A valve is a device used for controlling a fluid flow by opening and closing completely or partially a passage. A solenoid valve is a valve controlled by an electromagnet. In Figure 2.7 is represented a cross section of a 2 ways 2 position monostable solenoid valve.

In rest condition (coil not energized) the plunger, made of ferromagnetic material with rubber seal, is pressed against the valve seat by a spring and the medium pressure.

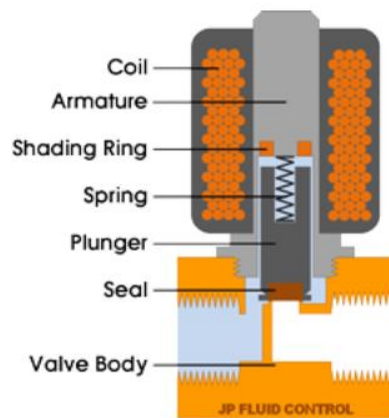


Figure 2.7: 2/2 Solenoid valve elements

When the coil is energized a current starts to flow inside it and a magnetic field is created (Figure 2.8-a). The plunger is magnetized and pulled toward the centre of the coil. In this way the orifice is opened and the medium can flow through it (Figure 2.8-b). As soon as the coil is de energized the flux starts to decrease until the valve closes going back to the rest position.

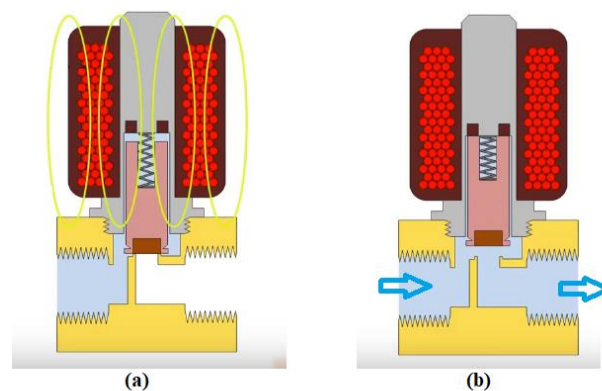


Figure 2.8: solenoid valve working principle

- (a) the coil is energized and the plunger is magnetized,
- (b) the magnetized plunger is pulled towards the armature opening the passage for the fluid flow

From the magnetic circuit point of view the air gap length used for the plunger movement is the biggest source of variation of the reluctance. The movement of the plunger causes a variation of the self inductance of the coil and this phenomenon can be observed in the current transient during the switching of the valve (Figure 2.9).

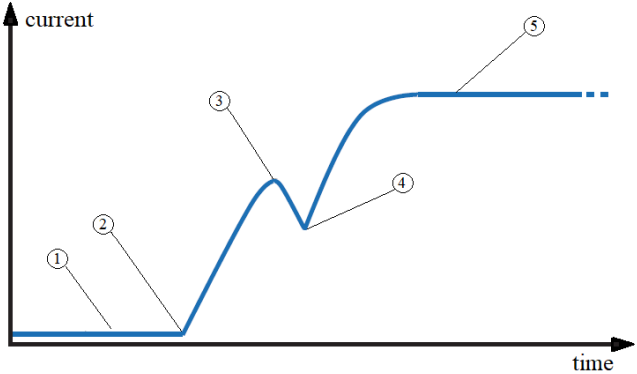


Figure 2.9: Current transient during an ON switch

1: Solenoid OFF (no current), 2:solenoid turns on and the current starts to raise, 3: current of peak (drop when the plunger strokes), 4: current of valley (plunger has moved completely), 5:current has reached the steady state

2.1.1.2 Weber – Ampère characteristic

The Weber Ampère characteristic of the solenoid is built plotting the linked flux in the solenoid with respect to the current flowing in the winding. To better understand the shape of this curve it is explained in two steps.

- Blocked plunger**

To better understand the shape of this curve let’s consider the hysteresis loop when the plunger is blocked in the extreme positions (valve completely open or close). For each of these two configurations applying a positive square wave as a voltage input for the coil the resulting flux-current curves are shown in Figure 2.10.

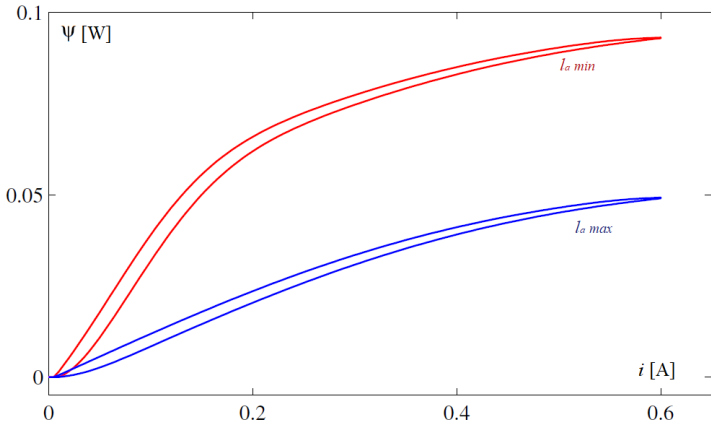


Figure 2.10:hysteresis cycles when the plunger is blocked in two extreme positions [5]

The blue curve corresponds to the closed valve: the air gap in the magnetic circuit is the highest ($I_a max$) and the self inductance of the solenoid is the minimum possible. For the valve in open configuration (red curve) the reluctance of the circuit is minimum and the produced flux is higher than the one in the previous case with the same flowing current.

- **Free plunger**

Letting the plunger free to move during the energizing cycle, the flux-current (i.e. Weber-Ampère characteristic) can be seen in Figure 2.11.

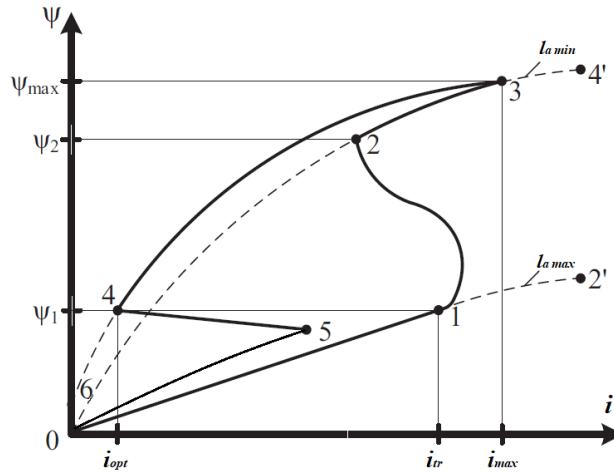


Figure 2.11: Weber-Ampère characteristic of a solenoid valve during a complete switching cycle [6]

Looking at Figure 2.11, when the valve is in rest configuration both the current and the flux are null (point 0). After the winding is turned on, the current increases as well as the flux linked to the coil until the point 1 is reached. At this point the plunger starts the movement, the air gap length decreases, the self inductance increases and the current in the winding has a drop. This transient continues until the plunger is attached to the core (point 2). After that the current continues to increase again until it reaches the steady state (see also figure 2.9) and the flux has the same behaviour until point 3.

When the coil is de-energized the current drops until it reaches the value i_{opt} (point 4) and the plunger starts the movement to return in the initial position (point 5), then the current reaches zero value (point 6). Notice that the path 5-6-1 in figure 2.11 is moving on the curve corresponding to the minimum inductance and maximum air gap length in figure 2.10, while the path 2-3-4 in figure 2.11 belongs to the red cycle in figure 2.10.

It is also important to point out that the Weber-Ampère characteristic and the current transient are just two projection of the same 3D curve on two different planes (current-flux and time-current). In Figure 2.12 it can be seen that the same key points corresponding to the beginning and the end of the switch can be found also in this type of curve

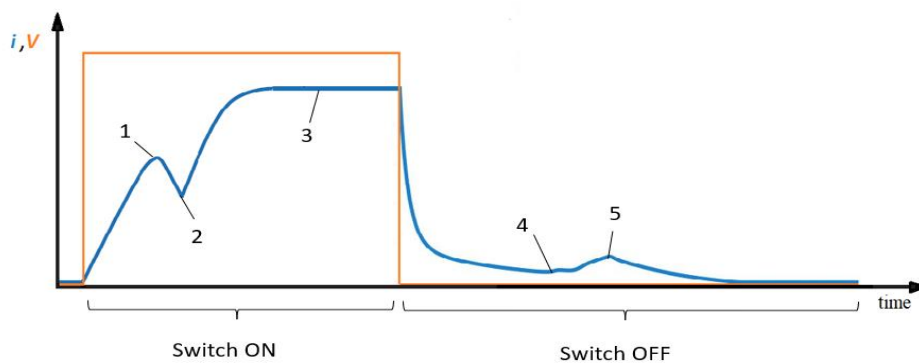


Figure 2.12: current transient during a complete switching cycle

2.2 Statistics concepts

In this section are discussed two concepts of statistics used in this thesis. In particular the linear regression will be used in the results analysis to try to model and compensate the effect of the temperature on the value of the detected features.

2.2.1 Standard deviation

A very important characteristic of any data set is the measure of the spread of the data. Some sets of data are characterized by a concentration of the values near the mean, while in others the data are more spread out from the average. One of the most common measure of the variation of the data value is the standard deviation. This number measures how far the data are in average from their mean value.

If x is a value in the data set, then the difference between x and the set average is called *deviation* of x . In a dataset every measurement has its deviation from the mean. For a sampled data, the deviation is represented with the notation $x - \bar{x}$ (where \bar{x} is the mean value in the data set) [21].

To calculate the standard deviation, it is necessary to calculate the variance first. The variance is the average of the squares of the deviations. The symbol used to represent this quantity for a sample is s^2 . The standard deviation s is the square root of the sample variance [21].

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}} \quad (12)$$

2.2.2 Linear regression

It is often necessary to know if two or more numeric variables are related and, if a relationship is present, determine what and how strong it is. In statistical modelling a regression analysis is a set of processes for estimation the relationship between a dependent variable and one or more independent variables [21].

The simplest form of regression is the linear one. This involves data that fit a line in two dimensions. A linear regression for two variables is based in a linear equation of the form:

$$y = ax + b \quad (13.1)$$

Where y is the dependent variable, x is the independent variable, a and b are constant numbers.

The equation 13.1 can be used as a model can be used to describe the relationship between each dependent variable real point y_i and the correspondent value of the independent variable real point x_i

$$y_i = ax_i + b \quad (13.2)$$

When the dependent variable dataset is obtained through measurements, the deviation of each measurement $y_{i\text{ meas}}$ from the real value y_i can be taken into account by adding to equation 13.2 a random error ε_i

$$y_{i\text{ meas}} = ax_i + b + \varepsilon_i \quad (13.3)$$

In equation 13.3 the measurements of the independent variable are assumed to be without error.

If were possible to obtain measurements of the two variables without errors, two points (x,y) would be enough to compute the line equation of the type 13.1 that passes through those points. Since this is an ideal case, method to find the equation of the line that best approximates the relation between two variables have to be used.

2.2.2.1 Least square estimation

The objective of a linear regression is to find the equation of the straight line that best fits the data set. One of the possible approaches is to use a least square regression line.

Let's consider the diagram in Figure 2.13. The points that belong to the dataset are in the form (x,y) while the points of the fit line are in the form (x,\hat{y}) . The \hat{y} are the estimated values, they are obtained using the equation of the regression line and in general are different from the measured y .

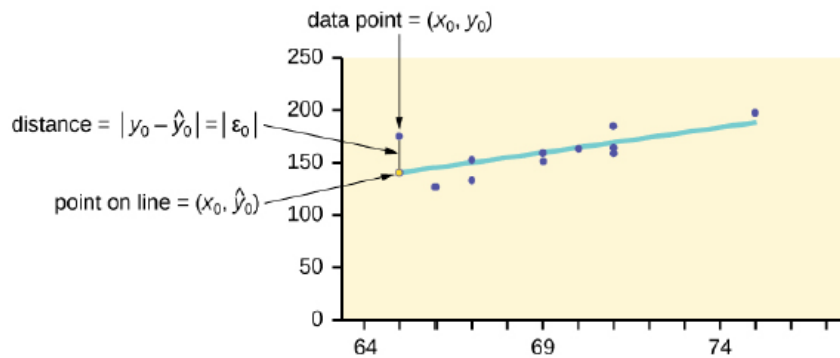


Figure 2.13: linear regression [21]

The term $\epsilon_0 = y_0 - \hat{y}_0$ is called *residual* and its absolute value measures the difference between the actual data point and the computed one for a given value of the independent variable. Geometrically it is the vertical distance between the point (x_0, \hat{y}_0) and the fit line.

For each data point it is possible to calculate the residuals $\epsilon_i = y_i - \hat{y}_i$ for $i = 1, \dots, n$ and then their sum $\epsilon_1 + \dots + \epsilon_n = \sum \epsilon_i$ called *sum of squared errors* (SSE).

Using calculus, the values of a and b are determined so that SSE is the minimum possible. The computed values are

$$a = y - bx, \quad b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})} \quad (13)$$

Where \bar{x} and \bar{y} are the means of the independent and dependent sample values respectively. The best fit line always passes through the point (\bar{x}, \bar{y}) [21].

After this the point on the line of best fit are determined using the equation $\hat{y} = ax + b$

The slope b describes how the changes of the two variables are related.

To evaluate how well the line of best fit predicts the dependent variable values the correlation coefficient r can be used. This number gives a measure of strength and direction of the linear relation between two variables. It is calculated as

$$r = \frac{n \sum(xy) - (\sum x)(\sum y)}{\sqrt{[n \sum x^2 - (\sum x)^2][n \sum y^2 - (\sum y)^2]}} \quad (14)$$

Where n is the number of data points.

The value of r is always between -1 and 1. The higher is its absolute value the stronger is the linear relationship between the two variables. The sign of r is the same of b in the fit line equation.

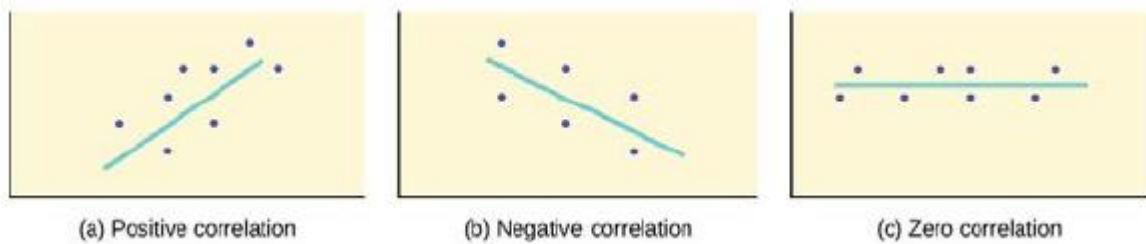


Figure 2.14: correlation coefficient sign [21]

The square of the correlation coefficient is called coefficient of determination and it has an interpretation in the context of data: when expressed in percentage, it represents the percent of variation of the dependent variable that can be explained by the variation of the independent variable using the regression line.

2.2.2.2 Models with more than a single independent variable

Most models use more than a single independent variable to describe the behaviour of a dependent variable. A linear additive model can be used extending the one described in equation 13.3

$$y_{i\text{ meas}} = a_1x_{i1} + a_2x_{i2} + \dots + a_nx_{in} + b + \varepsilon_i = \hat{y}_i + \varepsilon_i \quad (13.4)$$

The normal least square method can be applied to find the best fit equation for a multivariable linear additive relation, but the complexity of the computation increases exponentially with the number of independent variables [22].

In assuming this model are neglected the possible combined effects between two independent variables.

2.3 Computer communication protocols

An important part of the thesis was focused on the implementation of communication between devices and between device and cloud. In this section are presented the basics notions of the TCP/IP protocol, some concepts of communication security and an overview on the MQTT protocol. In the following paragraphs is also given an introduction on the LwIP library, used for the communication between the microcontroller and the Raspberry.

2.3.1 TCP/IP

Since the LwIP library gives the possibility of implementing a lightweight version of the TCP/IP protocol stack, some concepts are presented here to better understand the choices made for the final implementation.

2.3.1.1 Communication protocol

A network protocol is a set of common rules and conventions used for an effective communication between computers. A collection of related protocols is called protocol suite. The design that specifies how different protocols interact each other and how the different tasks to be accomplished are assigned is called architecture or reference model for the protocol suite [7].

The TCP/IP protocol suite is the most used to implement computers communication. It forms the basis for the global Internet, a wide area network (WAN) that spans the entire globe.

2.3.1.2 Packets, connection and datagram

In communication there are basically two types of service: the connection oriented and the connectionless. In the first one a connection has to be established before starting the communication. Establishing a connection means that a virtual circuit is made between the two communication end points. After that, the messages or the information can be sent and then the connection is released. In the connectionless service each message is sent independently from source to destination and the order of messages sent can be different from the order received.

One important concept is the idea of packet switching: “chunks” (*packets*) of the information to deliver are carried through the network in a certain sense independently.

A *datagram* is a special type of packet in which all the characteristic information on the sender and the final receiver are contained inside the packet itself. This brings the possibility of building a connectionless network and eliminates the need for a signalling protocol.

2.3.1.3 Errors and flow control

There are situations in which data in the network are damaged or lost. Dealing with such errors is called error control. Usually, if a small number of bits is damaged, some method can be used to detect and repair the error when the data is received. When a more severe damage occurs, could be necessary to retransmit the entire packet. This error control method is suitable for application that require error free delivery of the data.

An alternative to this approach a service called best-effort delivery can be adopted. In this case the network does not spend much effort to ensure that the packets are delivered without errors. Some type of errors can be detected and, in this case, the errant datagram is simply discarded without further actions.

When a best-effort delivery works effectively, could happen that the sender produces data at a rate higher than the receiver ability to process them. In best effort IP networks, limiting the sending rate is done through flow control mechanisms that work outside the network.

2.3.1.4 Layering

The usual approach used to implement protocol suites is called layering. With this approach each layer is responsible for a different aspect of the communication.

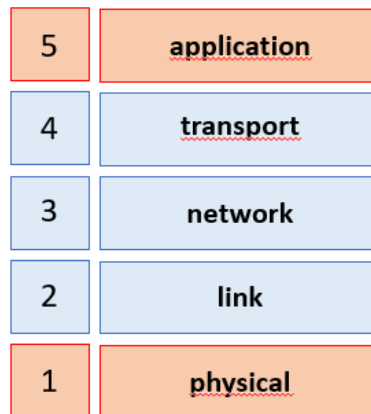


Figure 2.15: TCP/IP stack

In figure is represented in blue the layering inspired by the ARPANET reference model, which has been adopted by the TCP/IP suite. In red the two layers with which the protocol interfaces. The lowest layer represents the physical mean used to move the digital information. Portions of the Ethernet and Wireless standards are here. On top of the stack there is the application layer. Every application in general implements its own protocol and this is the most visible to the user.

Starting from the bottom, at layer 2, the first is an “unofficial layer”. It is a specialized protocol used with Ipv4 and only with multi-access link-layer protocols (Ethernet and Wi-Fi) to convert the address used by the IP layer and the one used by the link layer [7].

The layer 3 is represented the network layer protocol (or IP) used for the TCP/IP suite. The protocol data unit (PDU) that the IP layer sends to the link layer is called datagram. In some cases, the datagram is called packet when the context is clear. In this chapter the term IP is used to refer to both IP version 4 (IPv4) and version 6 (IPv6). Since the IP packets are datagrams, each one contains the addresses (called IP addressed) of the sender and the receiver. The main difference between IPv4 and IPv6 is the size of the IP address. The first makes usage of a 32 bits long address while the second uses a 128 bits address. The process of determining where each datagram should be sent by using the destination address and sending it to the next step (not necessarily the final receiver) is called forwarding.

At layer 4 the most common transport layer protocols are the transmission control protocol (TCP) and the user datagram protocol (UDP). The first deals with problem such as packets reordering, loss duplications and operates in a connection-oriented way. Conversely, the UDP provides little more features than what the IP does. This protocol imposes no rate or flow control.

The TCP provides a reliable delivery of data between two actors. It divides the data received from the application layer into appropriately sized packets for the network layer, it acknowledges received packets and sets timeouts for the other end acknowledgments.

The UDP provides a much simpler service for the application layer. The packet is delivered, but there is not guarantee that it reaches the destination. Any type of reliability must be added by the application layer.

2.3.1.5 Multiplexing and demultiplexing

One of the advantages of a layered architecture is the possibility of performing protocol multiplexing. This allows different types of protocols to coexist on the same infrastructure and multiple instances of the same protocol to work at the same time. Multiplexing can work at different layers and uses a sort of identifier for determining which protocol is used or which pieces of information belong together.

When an object called protocol data unit (PDU) is passed from a layer to a lower one, it is said to be encapsulated by the lower layer. In this passage the identifier (header) is added to the PDU to correctly identify the protocol use during the demultiplexing. In general, each layers generates its own header that is used for multiplexing when the data is sent and for demultiplexing when the data is received. As an example, to better explain this concept, let' explain how demultiplexing works from the link layer using Ethernet as an example (Figure 2.16)

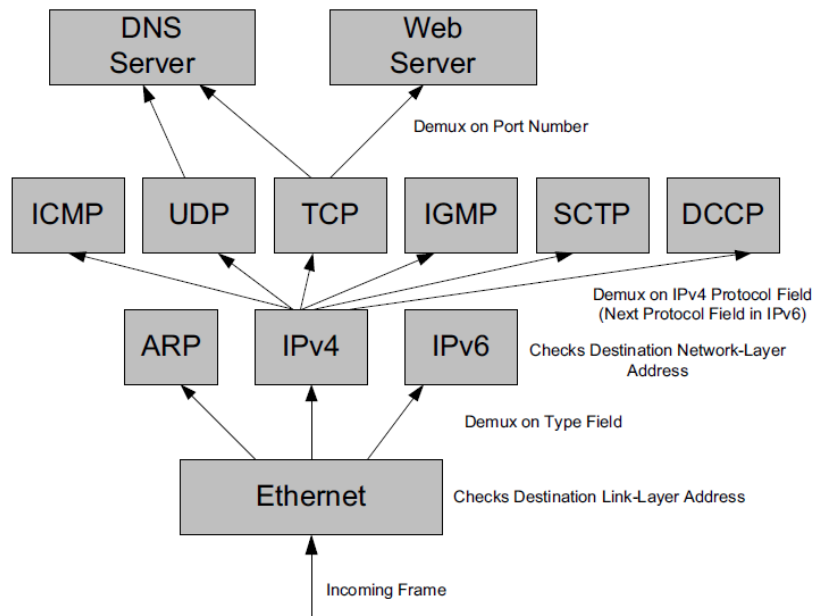


Figure 2.16: Demultiplexing [7]

An Ethernet frame when is received contains a 48-bit destination address (also called MAC- Media Access Control- address) and a 16-bit field called Ethernet type. If the MAC address matches the receiving system's one, the Ethernet type field value is used to select which network-layer protocol should be used to process the frame [7].

If the frame contains an IP datagram, the Ethernet header is removed, and the remaining bytes are passed to the IP for processing. If the address matches one the receiver's one and the datagram contains no errors in the header, the 8-bit IPv4 protocol field is used to determine which protocol use next.

After that the resulting datagram (reassembled from fragments if needed) is passed to the transport layer for processing. At this stage most of the protocols use port number for demultiplexing to the right receiving application [7].

Port numbers are 16 bits positive integers. Each IP address has a 65536 associated port numbers for each transport protocols that uses them. They are used for determining the correct receiving application.

2.3.2 Security

When two devices communicate, there are mainly three properties of information that are desirable from a security point of view:

- *Confidentiality*: the information is made available only to intended users
- *Integrity*: information must not be modified in unauthorized way before it is received
- *Availability*: information is available when needed

Some mechanism is needed to allow information to be sent through unsecured channels. With cryptography it is possible to achieve the three properties mentioned above. The two most important cryptographic algorithms are called symmetric key and public (or asymmetric) key ciphers [7].

In both the algorithms a clear text message is processed to produce cipher text (scrambled). The same clear text in input processed with different keys produces different outputs. In symmetric cryptosystems the encryption and decryption keys and algorithms are the same. In asymmetric

cryptosystems each party is provided with a pair of keys: one public and one private that are mathematically related. The first one is available to any party that needs to send a message to the owner and is used to encrypt the message, while the second is used to decrypt the received message by the owner of the key.

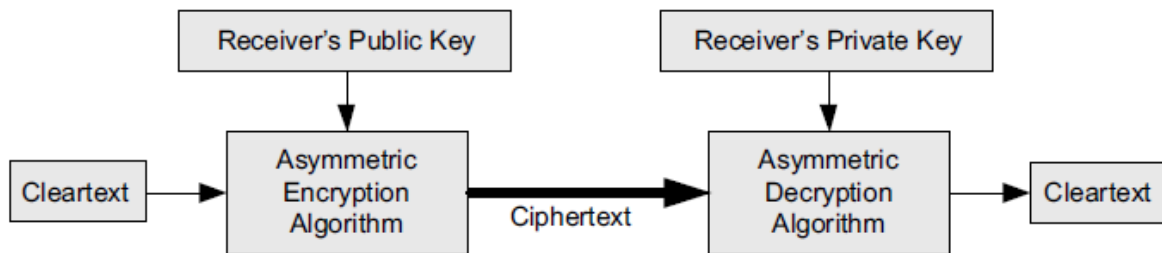


Figure 2.17: Asymmetric keys encryption [7]

2.3.2.1 Certificates authorities (CAs), PKIs and X.509

One of the challenges with public keys cryptosystems is to determine the correct public keys associated to an identity. The public key certificates are used to bind an identity to a particular public key. This approach involves the usage of a public key infrastructure (PKI) responsible for managing key pairs and associated certificates. It works with a collection of certificates authorities (CAs) that are entities used to manage and verify the bindings between keys and identities. CA employs a hierarchical signing scheme: a public key may be signed using a parent key that is signed using a grandparent key and so on. At the last step of validation, CA has one or more root certificates from which many subordinates certificates depend for trust.

Among different types of certificates, the most interesting si based on the ITU-T X.509 standard. The certificates may be stored and exchanged in a number of different encoding formats. The X.509 certificate in PEM (Base64 encoded) format is the default one for many Internet applications.

2.3.2.2 Security protocols and layering

Protocols involving cryptography can work at a different layer level in the protocol stack.

For example security implemented at the network layer protect information flowing between hosts, security at the transport layer protects process-to-process communication and security at the application layer protects information handled by applications.

The Transport Security Layer (TLS) operates just above the transport layer. It is one of the most popular because it can be implemented within or just under the application, while other security protocols (example Ipsec) require capability to operate within the operating system and protocol implementation. The TLS operates over a stream-oriented protocol (TCP) and provides confidentiality and data integrity based on a number of cryptographic protocols that use certificates provided by a PKI.

A detailed usage of the TLS protocol is besides the scope of this thesis. In this chapter were provided the basic concepts to understand the implementation of the secure communication between two actors in the developed architecture.

2.3.3 LwIP stack

LwIP is a light-weight implementation of the TCP/IP protocol stack initially developed by Adam Dunkels at the Swedish institute of Computer Science. The main focus of the LwIP stack is to reduce the memory usage and the code size while still having all the functionalities of the full TCP/IP. This make LwIP very suitable for small devices with limited resources such as embedded systems. Moreover, it is designed to operate with or without an OS and support for threads [8].

LwIP modularity provides a wide variety of protocols that can be removed for reducing the code size and complexity. Particularly relevant protocols in the LwIP stack are IPv4, IPv6 and ARP for the network and link layer; TCP and UDP for the transport layer; DHCP, AUTOIP and PPP as high-level protocols.

2.3.3.1 LwIP APIs

LwIP provides three types of application programming interfaces (APIs) to interact with the TCP/IP stack:

- Raw/Native (low level for callback style programs)
- Netconn (high level “sequential”)
- Socket (high level “sequential”)

The Raw API is an event-driven API used without an operating system.it implements a zero copy send and receive and it is used by the core stack for the interaction between the various protocols. This is the only API available when running LwIP without an operating system[10].

The two sequential APIs provide a way for sequential programs to use the LwIP stack. Since the TCP/IP stack is event-based, the TCP/IP code and the application must reside in different threads.

Figure 2.18 shows the main differences between the two types of APIs. Using the Raw one the developer has more control on the functionalities and size of the program, but this approach adds a little more of complexity in the development. On the other hand using the sequential APIs is simpler thanks to the higher level of abstraction, but an operating system is needed to manage multithreading and this requires a larger amount of memory.

	RAW API	Netconn / Socket API
RTOS	No need	Need
Control based on	Pcb	socket
Calling methods	Callback	Close to the windows or Linux socket APIs
Structure	Core APIs	Higher level APIs
Application	<ul style="list-style-type: none"> ➤ Lower memory devices ➤ Application without RTOS ➤ Developers has more control 	<ul style="list-style-type: none"> ➤ Higher memory devices ➤ Porting of protocols or application coming from Linux/windows
complexity	++	+
Memory	--	+

Figure 2.18: LwIP APIs comparison

2.3.3.2 LwIP packet buffers

The *pbuf* is an LwIP internal representation of a packet, designed for the need of minimal stack [9].

The *pbuf* structure supports dynamic memory allocation to hold temporarily the packet contents and static memory storage. Pbufs can be linked together in a chain so that a packet can be distributed over different pbufs. The structure can be of three types: PBUF_RAM, PBUF_POOL and PBUF_ROM.

The RAM type has the packet data stored in memory managed by the pbuf subsystem, while that is not true for the ROM type. The POOL type consists of fixed size pbufs allocated from a pool of fixed size pbufs.

The POOL type is mainly used for incoming data, while RAM and ROM types are used for outgoing packets. In particular the RAM types are suitable for data that are dynamically generated by the application. Figure 2.19 shows the structure of a PBUF_RAM. It consists of two pointers (next and payload), two length fields, a flag, a reference count and fields for protocol headers. The *next* field is a pointer to the next pbuf in case of chain. The *payload* is a pointer to the start of the data in the pbuf. The *len* and *tot_len* contain respectively the length of the payload and the length of all the payloads of the pbuf chain.

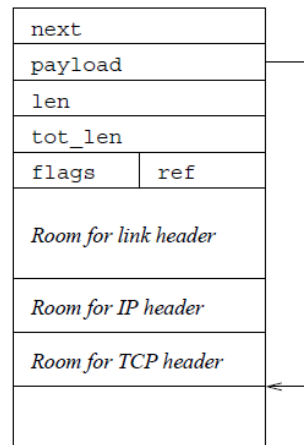


Figure 2.19: PBUF_RAM structure [9]

2.3.4 MQTT concepts

Telemetry technology gives the possibility to monitor devices from distance while reducing the cost of building applications.

MQTT (Message Queuing Telemetry Transport) is a lightweight and simple messaging protocol based on the TCP/IP stack. It provides telemetry technology and It is ideal for constrained environments in which there is bandwidths limitation, latency or when are used devices having limited processing and memory capabilities [20].

MQTT is able to solve these environmental problems while attempting to ensure reliability and delivery. This makes the MQTT protocol particularly suitable for machine-to-machine (M2M) communication. The basic concepts on which this protocol is based are the following [20].

- *Publish/Subscribe paradigm and broker*

The paradigm on which is build the communication consists in publishing messages and subscribing to topics (pub-sub model). Clients can subscribe to topics and receive whatever message published on those topics. Clients can also publish messages on topics making them available to all subscribed clients. In MQTT there is not direct connection between publisher and subscriber and messages are published to a central broker. The role of the broker consists in filtering the received messages based on topics and then distribute them to subscribers.

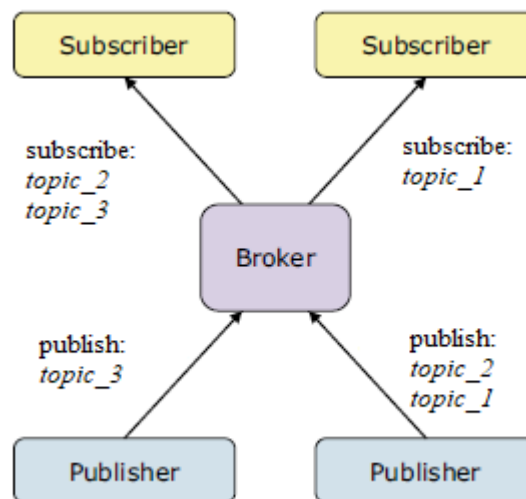


Figure 2.20: publish/subscribe paradigm and broker role

- *Topics and subscription*
Topics in MQTT can be thought as a subject area. Each client can subscribe explicitly to a topic or can use wildcard designators to receive messages from a variety of related topics.
- *Quality of service level*
The quality of service (QoS) defines the level of effort by the service to ensure that the message is delivered effectively. the MQTT protocol defines three possible levels. The higher is the level the more reliable is the service, but the higher is the consumed bandwidth and latency.
- *Retained messages*
The server (broker) keeps messages even after sending them to all subscribed clients. If a new subscription occurs, the retained messages are sent to the new subscribed.

2.4 Amazon Web Services

An important part of the system developed in this thesis was built using web services provided by Amazon. In this section is given a brief introduction on Cloud computing and on relevant concepts and terminology used to explain the implementation of the application using Amazon Web services.

2.4.1 Cloud computing concepts

Cloud computing can be defined as a style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet. This technology brings many advantages such as cost and time saving, high availability and scalability [13].

2.4.1.1 Layers of cloud computing

The services provided by a cloud can be presented in general as a layered cloud computing architecture. The most general division is the one shown Figure 2.21, but a more detailed division can be done. The three layers presented here are:

- **Software as a Service (SaaS)**
Also called “on-demand software” as end users have ready-to-use software that can run remotely. The main benefits of this service is no installation, maintenance, update and other costs. The payment for SaaS are based on the effective usage of the software and usually include technical support [14].
- **Platform as a Service (PaaS)**
Provides access to cloud-based platforms such as operating systems, instruments for software development and testing or database management systems. In this case the payment can be based on the usage time or for the volume of processed information [14].
- **Infrastructure as a Service (IaaS)**
Provides access to computing resources. The provider manages administration and hardware issues, while the customer has to manage all the settings of the operating system and application [14].

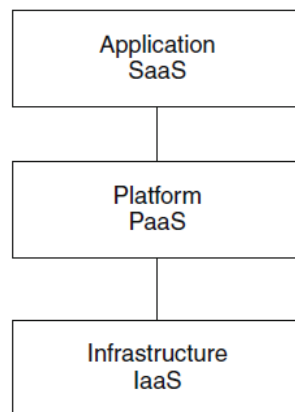


Figure 2.21: cloud computing layers [13]

2.4.1.2 Cloud deployment models

Depending on the final users target it is possible to classify the cloud into three categories [13]

- **Private cloud**
The computing is done over private network. Private or internal clouds are built for the usage of a single client or organization that has the complete control over security, quality of service and data. They can be built and managed by company IT organization or by cloud providers [13].
- **Public cloud**
In Public (or external) cloud the resources are dynamically accessible over the Internet through web services and applications by public customers [13]. Compared with the private cloud, the public one has the big advantage of being less expensive (in general pay-per-use billing), but a more complex considerations about security have to be done since it works over public networks.
- **Hybrid cloud**
Hybrid cloud is composed by multiple public and private cloud modules. This introduces the complexity of determining how to distribute the application across the two types [13].

2.4.2 AWS API

AWS is a cloud platform that provides a wide number of services at different levels (2.5.1.1). The services can be used as blocks for building any type of application on the cloud with a complete external or hybrid implementation extending private implementations.

The way for programmers to interact and communicate with the Amazon services is through the API. For a simple interaction with AWS APIs there are four main mechanisms [15]:

- **AWS management console**
A graphical web interface that provides the most intuitive way of interaction with services and computing resources. Through the main page of the AWS management console it is possible to access to all the other services console for instantiate and setting them [15].
- **SDK/CLI**
The SDK (studio development kit) and CLI (command line interface) allow programmers to directly call AWS APIs when developing an application that requires this direct access. The idea behind those two mechanisms is to provide a simple programmatic interface to a set of functions that easily allow the interaction with the AWS API [15].
- **Third-party tools**
Some company have built tools that incorporate AWS by simplifying it or extending it. In those tools just some or all the AWS services can be accessed and in general are integrated with other services not provided by Amazon [15].

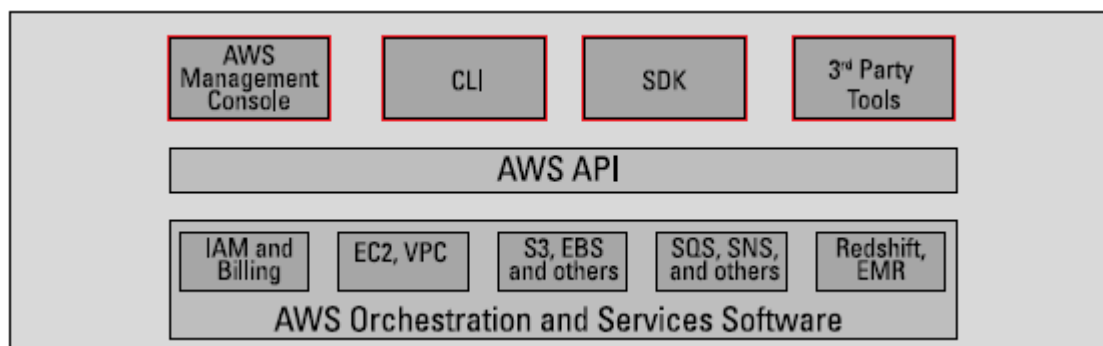


Figure 2.22: AWS API interface tools [15]

2.4.3 AWS Security concepts

In cloud computing a crucial aspect is related to the identification of the user that makes API requests and the determination of his permissions. In this paragraph some of the used concepts and terminology about Identity and Access management used on AWS are given.

- **Resource [16]**
In AWS a resource is an object that exists within a service. Examples of resources can be an amazons S3 bucket or an Amazon Dynamo DB table. Every service defines a set of actions that can be performed on each resources through API requests (for example, allowed requests on a Dynamo DB table can be delete, put, get an item) .
- **Principal [16]**
Is a user or an application that makes requests for actions or operations on an AWS resource
- **Request [16]**
When a principal tries to use an API, it sends a request to AWS. A request includes the following information:
 - *Action or operation* that the principal wants to perform
 - *Resources* upon which the operation is performed
 - *Principal* that sends the request
 - *Resource data* to identify the resource requested
- **Authentication [16]**
A principal must be authenticated (his *identity* must be verified) using credentials to send a request to AWS.
- **Authorization [16]**
After the authentication, another check is done to verify if the principal has the permission to *access* to the required resource. To do this AWS makes use of policies associated to the identity that makes the request.
Amazon Identity and Access Management (IAM) is a service that helps to control the access to the AWS resources focusing on the Authentication and Authorization phases.
- **(IAM) Identities [16]**
In AWS IAM Identities are created to provide authentication for people and processes. There are three types of identities:
 - *Users*: are entities that represent the person or the service that interacts with AWS services. A primary use of IAM users is to provide people the ability to sign into the AWS Management Console for using the AWS services. A user in AWS consists of a name, a password to sign in, and up to two keys for the API usage [16].
 - *Roles*: are like users, but they do not have any credential statically associated to them. A role makes use of temporary credentials that have a more restricted set of permissions than normal users. The role is intended to be assumable by anyone that needs it, but they are always used when an AWS service or an application needs to access to another AWS service's resource. In this way the credentials (IAM username and password) are not embedded in the application that makes the request [16].
 - *Groups*: is a collection of IAM users. Groups are useful to specify the permissions for a collection of users. Users can be easily added and removed from groups and they inherit the groups permission or lose them when they enter or exit from a group [16].
- **Policies [16]**
When a principal makes a request in AWS, checks are done to verify whether it is authenticated and authorized. The management of the authorizations is done by creating policies and attaching them to IAM identities. Policies are JSON documents containing the permissions for the associated identity and they are used to determine if a request is accepted or denied.

3 Setup and procedures

In this chapter the setup developed for the data collection during the endurance test is explained in detail. The structure used for the presentation of the implemented system is meant to follow the steps in the processing of each dataset generated by the microcontroller. The objective of this explanation is to show the steps both in the implementation and in the data processing and to allow the replication of this setup for further development.

3.1 Architecture overview

To better understand the next paragraphs, in this section a high-level view of the overall implementation is given. In Figure 3.1 is shown a block diagram of the implementation highlighting the tools and the steps used to process the data generated from the test procedures.

The first two rows of blocks on top of the scheme represent the valves under test that are physically connected to the microcontroller. The latter is responsible for the execution of the different test procedures periodically during the valve aging.

Every time a test procedure has completed, the measurement data are sent to the Raspberry using a lightweight UDP/IP protocol implementation over an Ethernet connection.

The packets received on the Raspberry are manipulated using the software Node Red. In this phase the data are converted to the right unit of measure, organized in json data format for successive cloud processing, the W-A characteristic is computed and finally they are sent to the AWS cloud. In this last step the used communication protocol is the MQTT with a transport layer security (TLS).

The packets are received in the AWS cloud through the IoT Core service. Every time a packet is received on the IoT Core service, it is stored in an AWS S3 bucket as a json file. Every file stored in the bucket contains the measurement data and metadata in output from a test procedure execution in a key-value structure. The S3 buckets are used for the first level of storage and contain the raw data.

Every time a new file is added to an S3 bucket, the Lambda function execution is triggered. As a result of this, the characteristic features of each test current and flux transients are detected and stored in a Dynamo DB table as an item.

Each item in a Dynamo DB table contains the features that characterize a single W-A curve.

As it is shown in Figure 3.1, the data of each valve are always separated during their processing. In particular in each step implemented in the architecture, every valve is associated to a different instance of the same service (flow, topic, bucket and table). The only common instance of a service is the lambda function. This is done because of the simplicity in keeping the data separated in this phase and the automatic scalability of this service.

In the next paragraphs a detailed explanation of the implementation is given following the order of data processing described in Figure 3.1. The material and the implementation are presented in a step by step order.

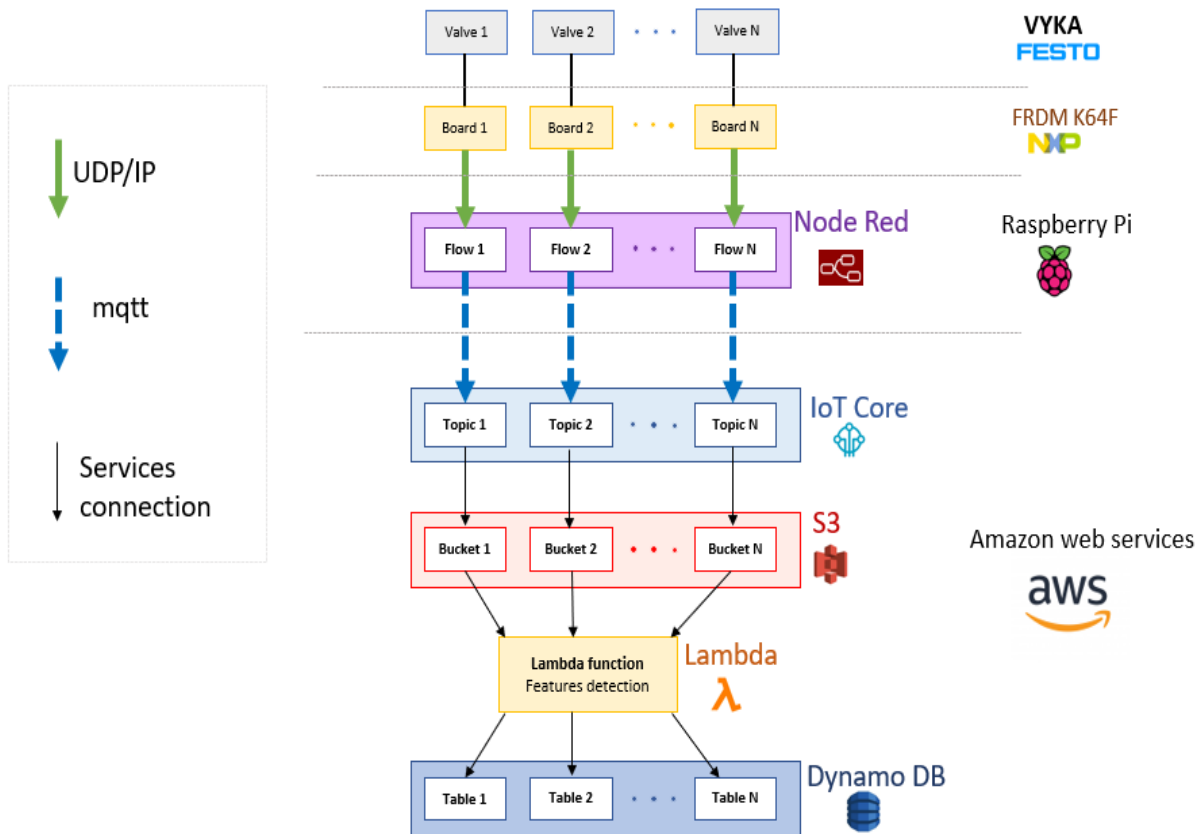


Figure 3.1: Architecture block-diagram

3.2 FESTO VYKA valve

The device under test is a 3/2 medium separated solenoid valve produced by Festo. It can be used for dosing or continuous flow applications in the variants 3/2 or 2/2, NO or NC.

The cross section of the valve is shown in Figure 3.2. The two connections among the three channels (NC-COM and NO-COM) are closed by the poppets (8) by pushing on the diaphragm (10). The closure of a passage and the opening of the other is determined by the resultant momentum applied to the rotating bar (6) around the joint (7). When the valve is in rest state the plunger (3) is pulled by the torsional spring (5) against the rotating bar (6). In this configuration the poppet above the NC channel closes the passage NC-COM. When the coil is energized the electromagnet pulls the plunger in contact with it. Due to the difference of stiffness between the two linear springs (9), the rotating bar pushes the poppet above the NO channel closing the NO-COM passage. When one of the two passages is closed the other is open thanks to the linear springs.

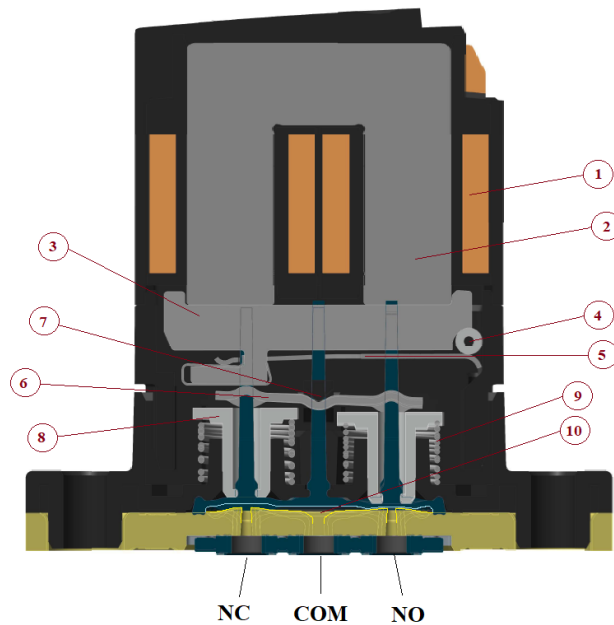


Figure 3.2 : VYKA valve cross section

1: copper windings, 2:electromagnet core, 3:plunger, 4:rotational joint,
 5: torsional spring, 6:rotating bar, 7:rotational joint, 8:poppet,
 9:linear spring, 10:diaphragm (blue) and channel (between blue and yellow)

3.3 Embedded system for measurements and data generation

The system responsible for controlling the valve and performing the measurement cycles is composed by a microcontroller and an electronic circuit on a printed circuit board (PCB) connected to the board through the pins. The usage of the electronic circuit is necessary for separating the power system (24V) from the digital part on the microcontroller (5V maximum) and for adding functionalities needed for the execution of the test procedures. Figure 3.3 shows the complete system for valve aging and data generation.

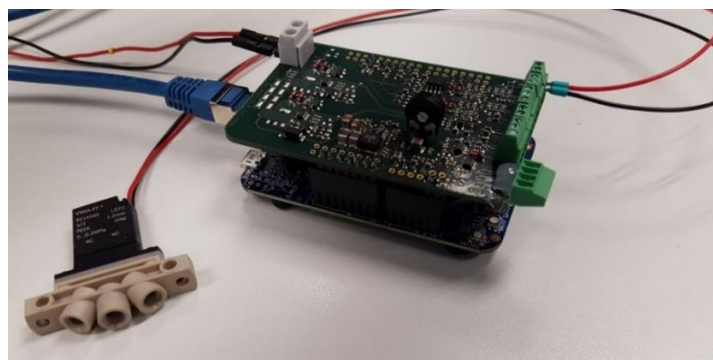


Figure 3.3: embedded system (microcontroller + PCB) for data generation

3.3.1 Electronic circuit

In this section is presented the part of the electronic circuit and the components of the microcontroller used for performing voltage and current measurements and for switching the valve.

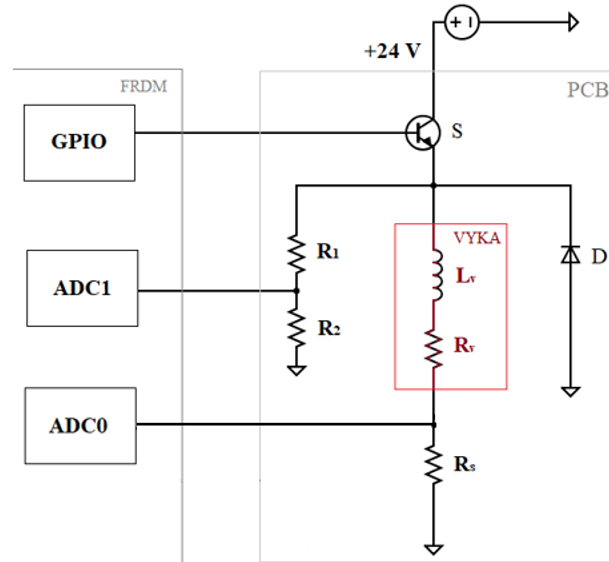


Figure 3.4: electrical circuit for voltage and current measurements and valve control

In Figure 3.3 the scheme of the electronic circuit (in the grey rectangle on the right) and the microcontroller components (in the grey rectangle on the left).

The solenoid valve (red rectangle in Figure 3.4) is modelled as a series of an internal inductance L_V and an internal resistance R_V .

The measurements of the voltage and current in the valve coil are performed using the microcontroller Analog to Digital Converters (ADC0 and ADC1). The two ADCs can convert only input voltages ranging from 0-3.3 V. For this reason, the resistances R_1 , R_2 , R_S are used. The resistances R_1 and R_2 are used to scale the voltage from almost 24V to the allowed input range for the ADC1. The shunt resistance R_S is used to indirectly measure the current flowing in the coil by measure the voltage drop across the resistance itself.

The transistor S is used as an electrical switch to drive the valve. The switch is controlled through the GPIO pin of the microcontroller.

A free-wheeling diode D is used to allow the current flowing in the valve during the switch off transient. In fact, in a solenoid the current can variate only in a continuous way since it is a state variable. When the switch S opens to de-energize the coil the current safely flows through the diode D without damaging the switch S.

3.3.2 Microcontroller software task

For the objective of this thesis, the software deployed on the microcontroller is responsible for the execution of two tasks: age the valve and perform measurements during a switching cycle. The two tasks are implemented at the software level as functions called “states”.

- STATE 1 (task: *valve aging*)

The valve switches at 17 Hz and no measurements are performed. This working condition is used to age the valve. The switching frequency is the maximum possible in order to allow the mechanical part of the valve to have a complete dynamic evolution (i.e. in Figure 3.2 the plunger and the rotating bar complete the movement closing and opening the two passages NO-COM and NC-COM in each cycle)

- STATE 2 (task: *voltage and current measurement during a switching cycle*)

This state consist in the execution of a test procedure. The voltage and current are measured using the two ADC (Figure 3.4) during both the switch ON and switch OFF transients. During each of the two transients, the two ADCs are triggered every 7 μ s until 2000 values are collected both for voltage and current. The sampling period of 7 μ s is the minimum usable without drive the ACD to instability. The amount of points collected is the minimum that, with the chosen sampling period, allows to sample the complete current transient during the OFF switching.

In Figure 3.5 voltage and current during a complete cycle are shown.

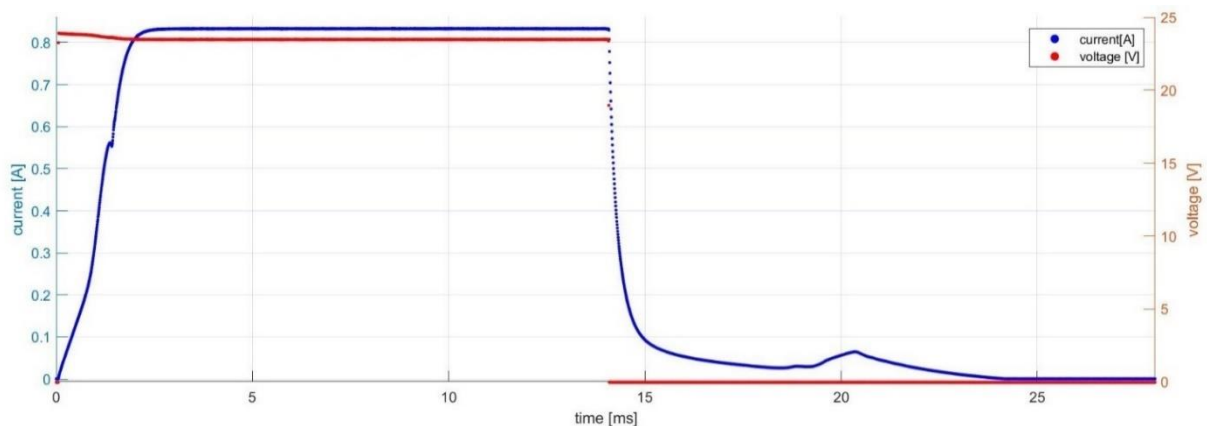


Figure 3.5 Current and voltage transient during a complete ON-OFF cycle

3.3.3 Microcontroller communication

At the end of each test procedure the measurement performed have to be sent from the microcontroller to the Raspberry. This is necessary because the limited memory of the microcontroller is not suitable for storing the data.

3.3.3.1 Motivations of the choice

Among the possible implementations of communication between the two devices (USB, Wireless, Ethernet) UDP/IP protocol over an Ethernet physical connection has been chosen and implemented using the LwIP stack. The motivation of this choice are the following:

- With respect to a serial (USB) communication, an Ethernet communication gives more flexibility. No new drivers are required for the Ethernet and the UDP/IPv4 protocol is suitable for connecting different networks. This means that using a company internal network (intranet) the data can be generated in a location and be sent to the Raspberry in another part of the building.
- A UDP transport layer has been preferred to a TCP one mainly for microcontroller resources limitations. Analysing the code of LwIP with respect to the number of lines in the source code, it can be seen that TCP is much larger than the other protocol implementations [9]. As shown in Table 3.1 the TCP contribution is as large as the API and support functions together. When the same analysis is done on the compiled object code for processors with different architectures, the TCP remains the biggest resources consumer (around 50% of the total memory used by the complete LwIP code) [9].

Module	Lines of code	Relative size
TCP	1076	42%
Support functions	554	21%
API	523	20%
IP	189	7%
UDP	149	6%
ICMP	87	3%
Total	2578	100%

Table 3.1: LwIP lines of code analysis [9]

On one hand the UDP has the advantage of being much lighter than the TCP protocol in terms of memory and computational power used, but on the other hand it brings some drawbacks:

- UDP protocol does not provide flow control. This means that an evaluation of the rate of packets sending should be done at the application level. The maximum rate of packets sent is imposed by the ability of the Raspberry to correctly process the received data. This aspect should be taken into account for the later stages of the project when multiple microcontrollers will send data to a single Raspberry.
- UDP does not provide retransmission functionalities, so if the packet is damaged or lost it is not retransmitted. Nevertheless, with the simple system used in this thesis, this problem never occurred during the monitoring done during the development phase

3.3.3.2 Implementation

The software development on the microcontroller was built without the usage of an operating system (OS bare metal). For this reason the only choice possible among the LwIP APIs was the Raw one.

The Raw API is meant for a call-back style application (request received → application execution → response sent), while the communication between the microcontroller and the Raspberry is unidirectional (data generated from the microcontroller are sent to the Raspberry at the end of every test procedure). At the application level, the event used to trigger the sending process is the termination of a test procedure.

The implementation of the communication procedure has been developed in two functions:

- `udp_enet_init()`: invoked at the beginning of the application execution to setup the LwIP stack and initialize the hardware needed for the communication
- `udp_enet_send()`: used to send the data generated by the measurement cycle

The code of those two functions is written in a file called `eNetCom.c` that is imported (with the relative header file `eNetCom.h`) in the software project in order to use the functions in the application. The two complete files are reported in Appendix B.

In order to use the udp functions, some setting at hardware level and library importing should be done. The application developed for the microcontroller makes usage of Processor Expert for the low level hardware initialization. The steps needed for implementing those basics setup are reported in Appendix A.

3.3.3.3 udp_enet_init()

The sequence of function composing the `udp_enet_init()` is developed by modifying the initialization steps used in the `udpecho_example` in the NXP Software Tools library. The implemented steps are the following:

- 1- declare the variables and structures used for the communication

```
// lwip/udp variables
uint16_t port = 1500; // Raspberry open port for sending data
uint16_t valve_ID = 1; // set the ID here
struct udp_pcb * pcb; // protocol control block structure for udp protocol
struct netif fsl_netif0; // network interface structure
ip_addr_t fsl_netif0_ipaddr, fsl_netif0_netmask, fsl_netif0_gw, Raspi_address; // used IP addresses
```

- 2- Calling the `app_low_level_init()` to initialize timer, clock used for the Ethernet port communication. This function is also defined in the `NetCom.c` file.

```
static void app_low_level_init(void)
{
    // Open UART module for debug
    hardware_init();

    // Open ENET clock gate
    CLOCK_SYS_EnableEnetClock(0);
    // Select PTP timer outclk
    CLOCK_SYS_SetEnetTimeStampSrc(0, kClockTimeSrcOsc0erClk);

    // Disable the mpu
    MPU_BWR_CESR_VLD(MPU, 0);
}
```

- 3- Calling the `lwip_init()` the lwip stack is initialized with all of its subsystems. This function is defined in the lwip library.
- 4- Define the IP addresses of the Raspberry and the microcontroller, the network mask and the default gateway address. The first three are used in the initialization of the network interface of the microcontroller (see point 5). To initialize a network interface a default gateway address is required. In the described application, the network accessible by the microcontroller is composed only by the microcontroller(s) and a Raspberry not connected with external networks. For this reason the gateway address should never be useful. Using the network mask 255.255.255.0 means that the defined address for the microcontroller must have the first three octets equal to the Raspberry's one. Anyway the Raspberry IP address is used also as the default gateway one.

```
// IPv4 addresses (32 bits)
IP4_ADDR(&fsl_netif0_ipaddr, 169,254,186,155); // microcontroller static IP address
IP4_ADDR(&fsl_netif0_netmask, 255,255,255,0); // Network mask
IP4_ADDR(&fsl_netif0_gw, 169,254,186,153); // Default gateway address = Raspi_address
IP4_ADDR(&Raspi_address, 169,254,186,153); // Raspberry IP address
```

- 5- Initialize the network interface using the IP addressed defined before. In this application a single network interface is sufficient for the application communication. The last two parameters passed to the `netif_add()` function are obligatory choices when using the Ethernet physical layer. After the initialization, the network interface is set as the default one and enabled.


```

netif_add(&fsl_netif0, &fsl_netif0_ipaddr, &fsl_netif0_netmask, &fsl_netif0_gw, NULL,
ethernetif_init, ethernet_input); //Add a network interface to the list of lwIP netifs
netif_set_default(&fsl_netif0); // Set the network interface as the default network interface
netif_set_up(&fsl_netif0); // Bring the interface up, available for processing traffic

```

- 6- Initialize the pcb (protocol control block) for udp protocol. It contains the information required and used at transport layer level. After a sanity check, the remote address and port in the pcb structure are set as the Raspberry's.

```

// get new pcb
pcb = udp_new();

if (pcb == NULL) {
LWIP_DEBUGF(UDP_DEBUG, ("udp_new failed!\r\n"));
return -1;
}
//set the remote address as the Raspberry's one
udp_connect(pcb, &Raspi_address, port);

return 0;

```

In Figure 3.6 is represented the final implemented stack with the functions used to build it.

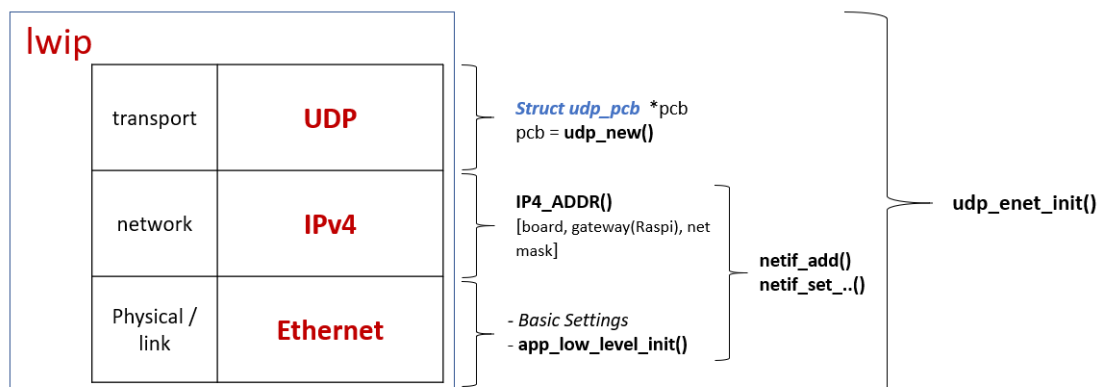


Figure 3.6: protocol suite recap scheme

3.3.3.4 udp_enet_send()

After the `udp_enet_init()` is executed, it is possible to invoke this function in the main application. `udp_enet_send()` takes 2 input arguments:

- the pointer to the first element of the array `data[]`. This array contains the packet to send composed by the data in output from the measurement cycles plus some metadata. (Figure 3.7)
- the size of the array `data[]`

In the `udp_enet_send()` are just executed the steps required to send a packet. In particular:

- 1- a new pointer to a packet buffer (pbuf) structure is declared. The allocation is done using a RAM type buffer suitable for continuously update the payload. The `PBUF_TRANSPORT` argument is used to include spare rooms for the transport layer header in the pbuf structure (Figure 2.19), `data_size` argument is the size of the packet to send (`data[]` array).

```

struct pbuf *new_p; //packet buffers
new_p = pbuf_alloc(PBUF_TRANSPORT, data_size , PBUF_RAM);

```

2- The packet we want to send in the “payload” attribute of the pbuf and the pcket is sent using the `udp_send` function provided by the RAW LwIP API. This function takes as agument the pbuf containing the data and the `pcb` structure containing the remote address of the Raspberry. After the message is sent, the pbuf structure is dereferenced and deallocated using the `pbuf_free` function from the LwIP RAW API.

```

if(new_p != NULL){
    memcpy(new_p->payload, data1, data_size);
    err = udp_send(pcb, new_p);
    pbuf_free(new_p);
}

```

3.3.3.5 Application packet preparation

Every data set resulting from the test procedures must be associated to other three quantities:

- Switch counter: a 32 bits integer that indicates the number of switches performed by the valve until that test. Every time the GPIO (Figure 3.4) complete a driving cycle 0-1-0, this variable is incremented by one. This is an indication of the age of the valve.
- Valve ID: a 16 bits integer used to identify the valve that generated the data
- Test ID: a 16 bits integer used to identify the test procedure performed. This is used to interpret the data meaning when they are read in NodeRed on the Raspberry.

This numbers are concatenated in a single array with the data from the test procedure at the application level.

```

uint16_t *array_concat(uint16_t sw_count_LSB, uint16_t sw_count_MSB, uint16_t valveID, uint16_t testID,
    uint16_t measure1[], uint16_t measure2[], size_t ADC_buf_size){

    int tot_size = (4 + ADC_buf_size)*sizeof(uint16_t);
    char *p = malloc(tot_size);

    memcpy(p, &sw_count_LSB, sizeof(uint16_t));
    memcpy(p + sizeof(uint16_t), &sw_count_MSB, sizeof(uint16_t));
    memcpy(p + sizeof(uint16_t)*2, &valveID, sizeof(uint16_t));
    memcpy(p + sizeof(uint16_t)*3, &testID, sizeof(uint16_t));
    memcpy(p + sizeof(uint16_t)*4, measure1, ADC_buf_size*sizeof(uint16_t));
    memcpy(p + sizeof(uint16_t)*(4 + ADC_buf_size), measure2, ADC_buf_size*sizeof(uint16_t));

    return p;
}

```

The result is the UDP packet to be sent to the Raspberry. In Figure 3.7 is represented the package map for the voltage and current measurement cycle.

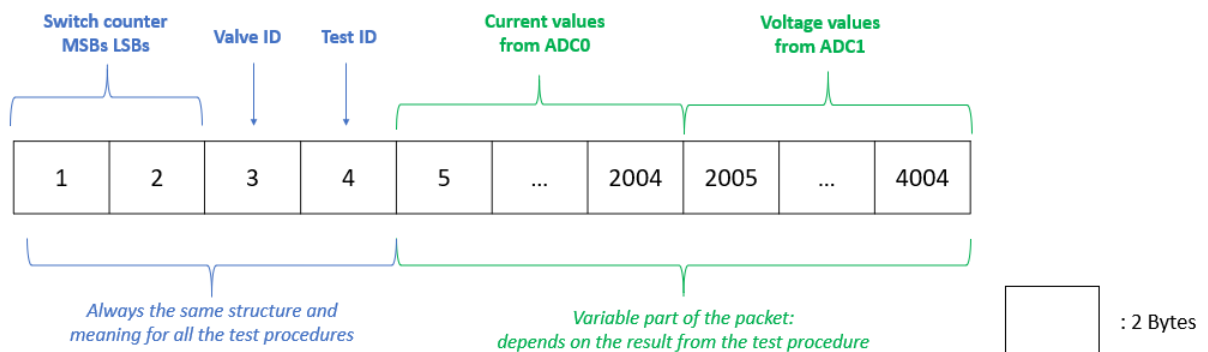


Figure 3.7: package structure

Although in this thesis only the voltage and current measurement cycle is used, a map for the test ID usage is already prepared and implemented both on the microcontroller and on the Raspberry (see 3.4.2.3). Table 3.2: test ID mapping shows the test ID mapping in which the first two rows, Rising and Falling, refer to the switch on and off transients respectively.

Current Voltage Rising curve	1
Current Voltage Falling curve	0
Current Acceleration Rising curve	11
Current Acceleration Falling curve	10
PWM Sweep	20
Leakage test	30

Table 3.2: test ID mapping

3.3.4 Board routine

During the whole endurance test, the software running on the microcontroller performs always the same sequence of actions. It keeps the valve in state 1 for 60 seconds and then performs the measurement cycle switching to state 2. At the end of the execution of the test the board sends the results to the Raspberry and goes back to state 1 restarting the cycle. In state 2 it was necessary to split the measurement cycle in two parts because the memory available was not enough to store even temporarily all the measurement points of a complete ON-OFF transient.

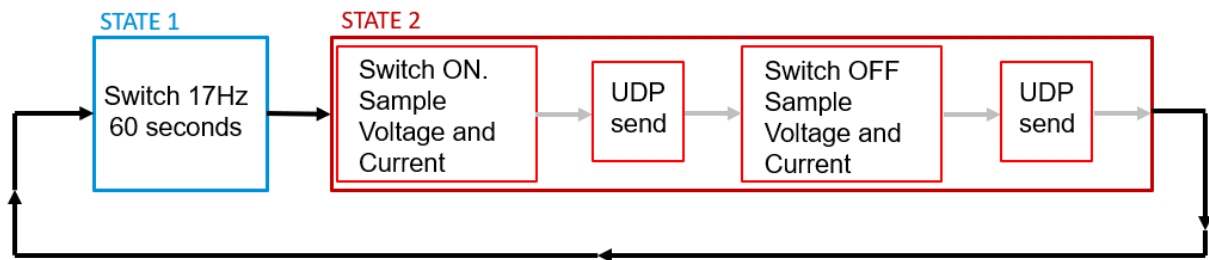


Figure 3.8: Board tasks routine

3.4 Raspberry Pi

All the data sent by the microcontroller pass through the Raspberry Pi before being stored and processed in the cloud. This is done for two reasons:

1. A pre-processing of the data from the test procedure is needed. In general, the data in output from the microcontroller are contained in a buffer of numbers that are not yet expressed in the right unit of measure and they are not readable without knowing the package mapping. At this stage, the also flux information is computed starting from the transient measurement to minimize the computational resources used on the cloud
2. A change the communication protocol and a security layer are also for the cloud communication through the Internet.

These two tasks are performed using Node Red.

3.4.1 Flux computation

In this section is explained how the magnetic flux linked to the solenoid is computed starting from the current and voltage measurements and knowing the electronic circuit parameters. This explanation is done at this point since the computation of the flux is performed on the Raspberry.

To explain how the linked flux in the coil is computed let's consider the portion of the circuit in the PBC (Figure 3.4) to which the valve is connected Figure 3.9.

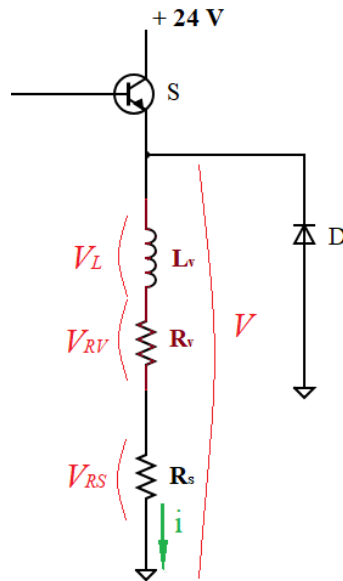


Figure 3.9: Kirchhoff's voltage law for flux computation

Writing the Kirchhoff's Voltage law

$$V = V_L + V_{RV} + V_{RS} \quad (15)$$

Where V and V_{RS} are measured quantities

From the Lenz's law (equation 10) the linked flux in a solenoid can be computed as

$$\psi(t) = \int e_{ind} dt + \psi_0 \quad (16)$$

where $e_{ind} = V_L$ induced voltage across the internal inductance of the valve and ψ_0 is the linked flux at the beginning of the switching transient. ψ_0 is assumed null neglecting the residual induction of the

ferromagnetic material. This can be done because the final objective of this experiment is to detect the variations in the quantities and not the absolute values.

When the switch on transient has completed, according to the equation (11), the induced voltage across the inductance is null. In this phase there is no variation of the inductance (the plunger does not move anymore) and of the current that is in steady state. The (12) becomes

$$V = R_V \cdot i + R_S \cdot i \quad (17)$$

where R_S is the only unknown and therefore can be computed. To have a good estimation of the internal resistance of the coil of the valve, it is computed using the average of the last 100 values of current and voltage measurements in the switch on transient (Figure 3.10 highlighted in yellow and green). In this way the effect of the noise of the measurement is attenuated.

$$R_V = \frac{\sum_{i=1901}^{2000} V_i}{\sum_{j=1901}^{2000} i_j} - R_S \quad (18)$$

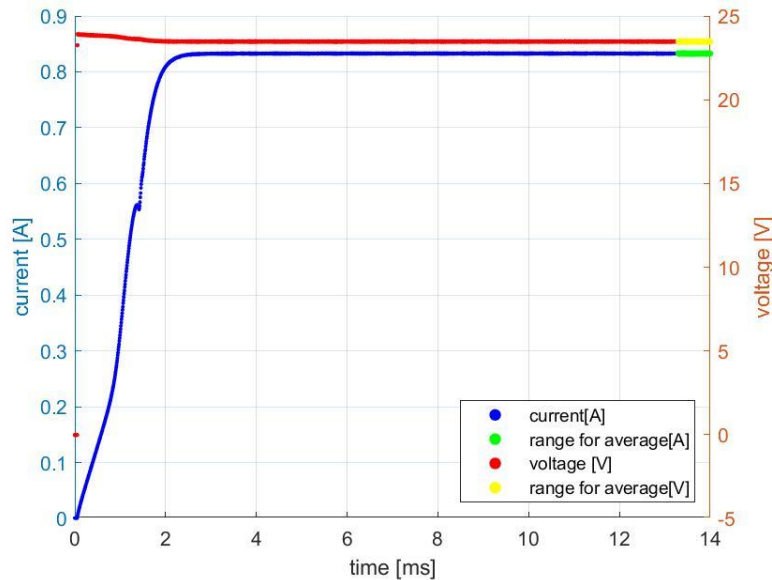


Figure 3.10: range for internal resistance computation

After the estimation of the internal resistance the flux array elements are computed as follows

$$\begin{aligned} V_L(k) &= V(k) - i(k)[R_V + R_S] \\ \psi(k) &= \psi(k-1) + V_L(k) \cdot t_s \quad k = 1, \dots, 2000 \\ \psi(0) &= \psi_0 \end{aligned} \quad (19)$$

where k is the index of the measurements array. The initial flux is assumed $\psi_0 = 0$ for the switch on transient, while for the switch off transient is assumed equal to the last value computed during the switch on transient.

It is important to point out again that with the used electronic circuit (Figure 3.4) it is possible to measure only positive values of the voltage and the current. During the switch off transient the measure voltage V (Figure 3.9) measured by the microcontroller is always equal to zero as it can be seen in Figure 3.11

(red line). Actually, during the off switching transient, the current flows through the free-wheeling diode causing a voltage drop across this component. The voltage V should be negative and equal to 0.89 V. This contribution is manually added in the region between the voltage is switched off and the current reaches zero (Figure 3.11 Diode ON region) because the electronic circuit is not able to measure negative voltages. The results of this are acceptable since the curve closes at the origin at the end of a switching cycle without residual flux.

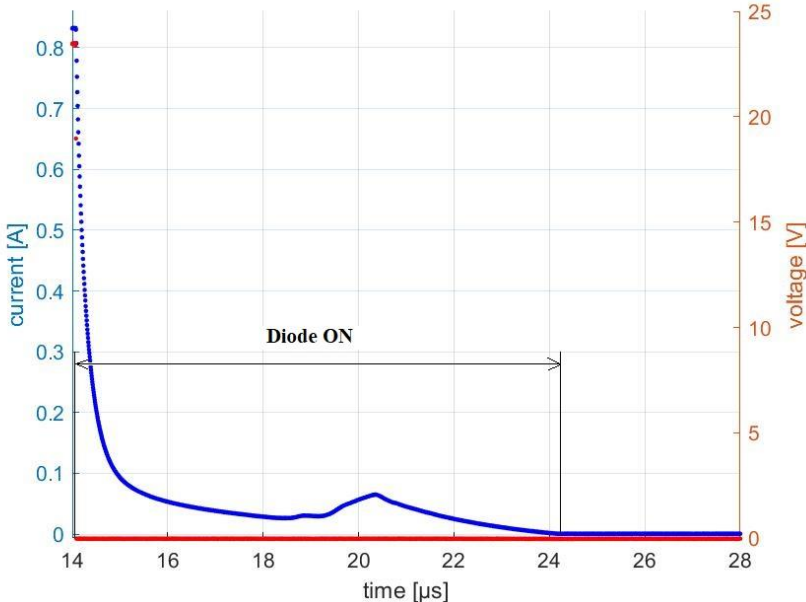


Figure 3.11: Diode ON region

The computed flux can be plotted against the current for both the on and off switching transients. The obtained curve is shown in Figure 3.12.

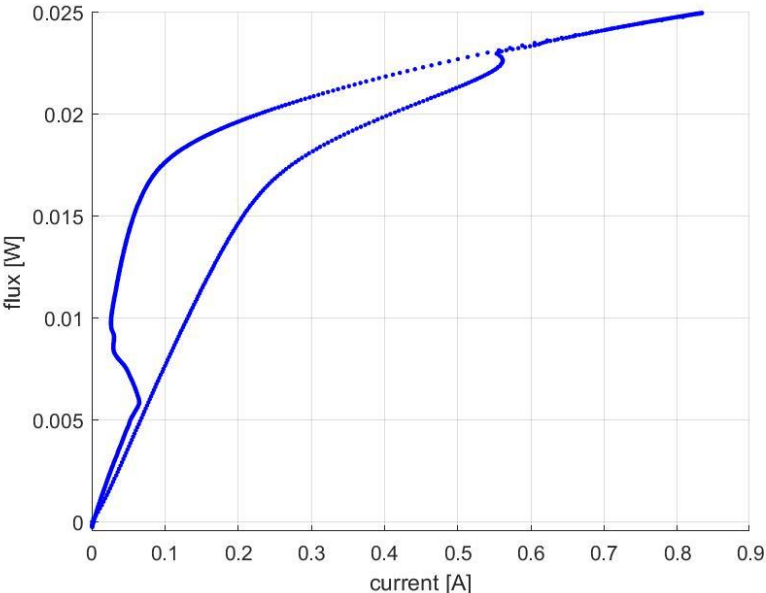


Figure 3.12: Weber-Ampère characteristic for the VYKA valve

3.4.2 Node Red

Node Red is a Node.js-based tool for building Internet of Things applications with a focus on simplifying the connection of code blocks (‘nodes’) to carry out tasks. The combination of nodes (usually with structure input-process-output) wired together makes up a flow. Node Red provides a browser-based editor that allows the development and deployment at runtime. To build the flows JavaScript function can be created for customized nodes and build in library block can be used for a wide range of tasks [12].

In this section is described the flow developed to accomplish the tasks presented at the beginning of section 3.4. A general explanation is given to show all the implemented functionalities, with a focus on the most important features. For completeness, in Attachment A the file containing the complete flow JSON code is provided. To view the complete flow this file has to be imported in Node Red.

3.4.2.1 Main flow

The main flow created for this architecture is shown in Figure 3.13. It composed by seven nodes (5 sub-flows and two built-in blocks for the communication input and output). Three of those blocks are used to accomplish the main tasks of the Raspberry:

- “UDP from K64F board” node used to receive the packets from the microcontroller
- “Data Elaboration” sub-flow used to process the received data
- “To AWS Cloud” used for sending the elaborated data to the Cloud.

The other sub-flows are mainly used to add robustness functionalities to the system.

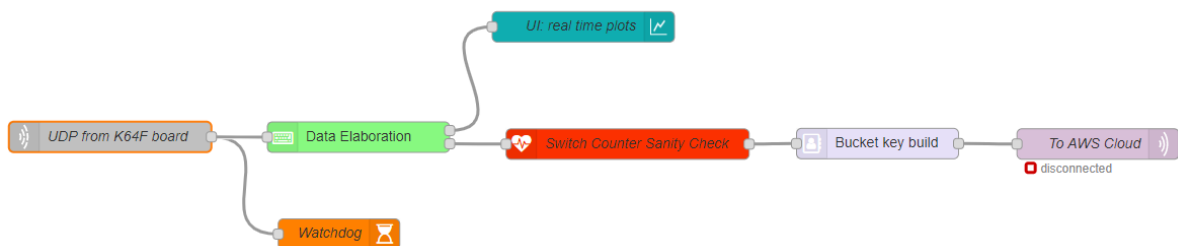


Figure 3.13: Node Red main flow

3.4.2.2 UDP from K64F board node

Using the built-in “UDP IN” block in Node Red a port is opened for receiving the data from the microcontroller. The setup of this node is very simple as it is shown in Figure 3.14. It is needed just to specify what the node should listen for (udp or multicast messages), the port and the network protocol and the type of output (buffer or string). With these settings, the received packet is automatically read as a sequence of 8bits integers, so the output buffer does not contain the right numbers yet.

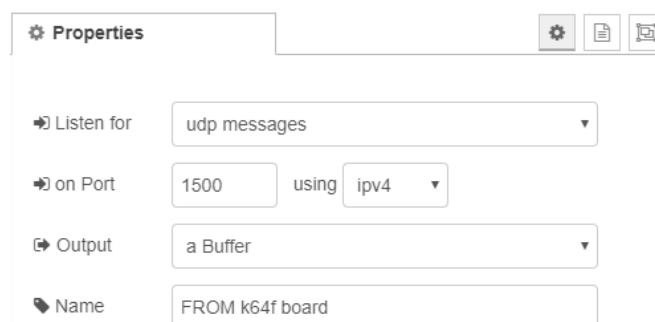


Figure 3.14: UDP in node settings

3.4.2.3 Data Elaboration sub-flow

In this sub-flow are performed all the data manipulation: change of the units of measure, flux computation and payload formatting. Following the flow process from left to right in Figure 3.15 the action executed are the following:

- 1- In the first node the received buffer re interprets the buffer received according to the data types send by the microcontroller (see Figure 3.7): the first 4 Bytes as 32 bits integer followed by 16 bits integers.
- 2- The switch block uses the test ID number to select the branch with the right processing sequence. For the Current-Voltage measurement cycles processing is used the lowest branch in Figure 3.15
- 3- The data are converted in the right unit of measure (Current, Voltage, Sampling period)
- 4- The average resistance is computed using equation (18)

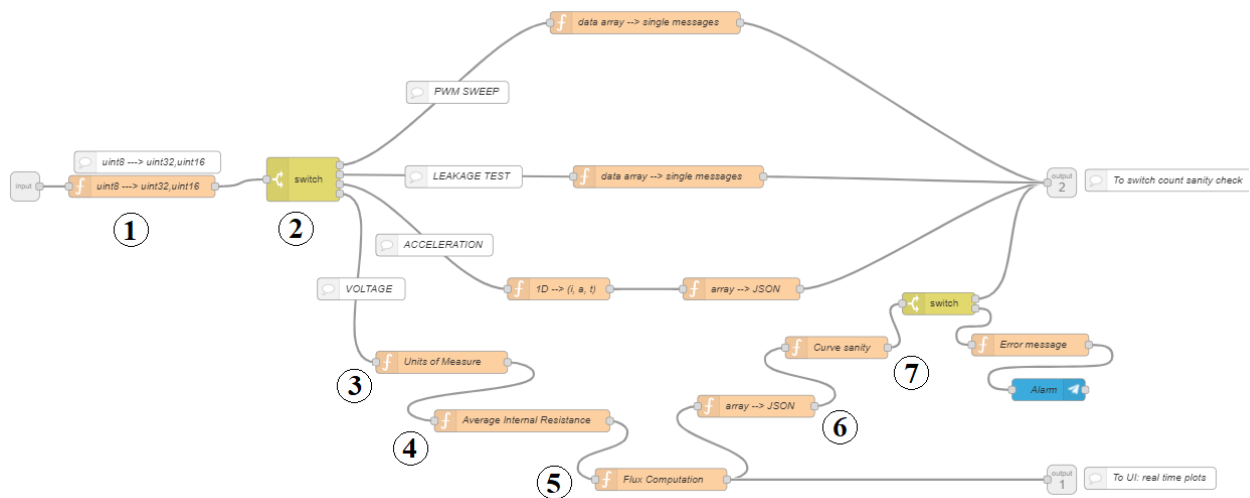


Figure 3.15: Data Elaboration sub-flow

- 5- The magnetic flux during the switching transient is computed using equation (19)
- 6- The payload is organized in a JSON data structure (Figure 3.16: The Current, Voltage and Flux variable are vectors of 2000 values).
- 7- A sanity check is performed and if not passed the entire payload is discarded and an alarm message is sent on a telegram chat in order to trigger an intervention. The check is performed evaluation if the maximum value of the flux in the array is lower than a threshold. If that is true the main cause is a non correct turning off of the system: in this case the microcontroller would send wrong measurement sets.

```

{
  "valveID": 1,
  "SwitchCount": 2398012,
  "testID": 1,
  "ResAve": 26.5786,
  "Current": Current,
  "Voltage": Voltage,
  "timestamp": 1583049305892,
  "Flux": Flux,
  "Tsampling": 0.000007
}

```

Figure 3.16: JSON structure of the test data after the elaboration

It is important to point out that the two branches of the complete switching cycle are processed and sent to the cloud separately. This was necessary because the payload that is possible to send to the AWS platform has a maximum size when using AWS IoT Core (service used for receiving messages).

3.4.2.4 To AWS Cloud node

This node is obtained by setting the built in “mqtt out” node in order to establish a secure connection with AWS IoT Core service.

The configuration of this node consists of three steps:

- 1- At the first level (Figure 3.17-a) the topic has to be inserted. In this application the topic has a basic hierarchical structure to facilitate the scalability of the system. The Quality of service has to be zero for the communication with AWS IoT Core.
- 2- The second step consists in the server configuration (Figure 3.17-b). The address and port of the AWS broker are obtained directly from the AWS IoT Core console during the service configuration.
- 3- To configure the TLS, enable its usage (Figure 3.17-c) and attach the certificates and the private key obtained from the IoT Core console. The first certificate is used for the device authentication when the connection is established, while the second (CA root) is used for the server authentication. The private key is used for payload encryption.

In paragraph 3.5.1 will be explained how to obtain and activate those certificates used in TLS configuration. Before that phase it would not be possible to establish a connection with the AWS platform.

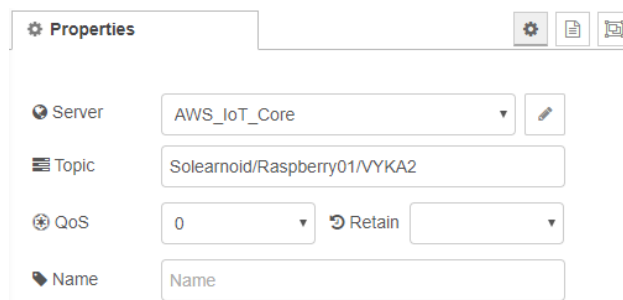


Figure 3.17-a shows the configuration for the MQTT node. The 'Server' dropdown is set to 'AWS_IoT_Core'. The 'Topic' field contains 'Solearnoid/Raspberry01/VYKA2'. The 'QoS' dropdown is set to '0', and the 'Retain' checkbox is checked. The 'Name' field contains 'Name'.

(a)

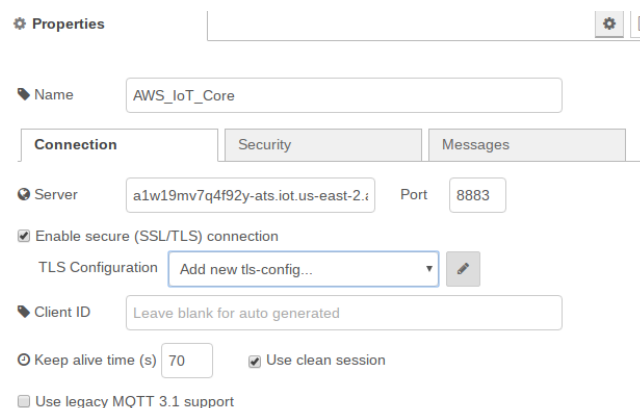


Figure 3.17-b shows the configuration for the MQTT node, specifically the 'Security' tab. The 'Name' field contains 'AWS_IoT_Core'. The 'Connection' tab is selected. The 'Server' field contains 'a1w19mv7q4f92y-ats.iot.us-east-2.i' and the 'Port' field contains '8883'. The 'Enable secure (SSL/TLS) connection' checkbox is checked. The 'TLS Configuration' dropdown is set to 'Add new tls-config...'. The 'Client ID' field contains 'Leave blank for auto generated'. The 'Keep alive time (s)' field contains '70' and the 'Use clean session' checkbox is checked. The 'Use legacy MQTT 3.1 support' checkbox is unchecked.

(b)

Properties

Use key and certificates from local files

Certificate

Private Key

Passphrase

CA Certificate

Verify server certificate

Server Name

Name

(c)

Figure 3.17: To AWS Cloud node configuration

3.4.2.5 Switch Counter Sanity Check sub-flow

In case of reboot of the microcontroller, the switch counter variable would be reinitialized. The sub-flow in Figure 3.18 has been implemented to always have a consistent switch counter variable. For every received packet, the contained switch counter is compared with the previous value received stored in a local file. If the received value is lower than the previous one, it is corrected otherwise it is left as it is. After that, the file is updated with the correct value and the payload proceeds to the next sub-flow.

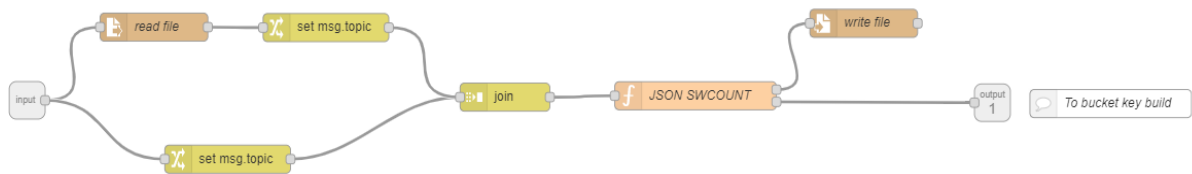


Figure 3.18: switch counter sanity check sub-flow

3.4.2.6 Bucket key build node

Every JSON containing the data from the test procedure, will be stored in a file on the cloud in the next step

The bucket key is used as identifier for the file. It is create by concatenating the switch counter with the test procedure ID. In this way it is possible to identify the test procedure from the last two digits and the age of the valve from the remaining part of the number.

3.4.2.7 Watchdog sub-flow

The software running on the microcontroller has unsolved problems that lead to periodic faults that block the execution of the running program. Every time the microcontroller program is blocked, it has to be restarted. This sub-flow has been implemented to accomplish this task automatically and send a Telegram message in case the restart does not work.

The microcontroller, in normal working condition, sends a UDP packet to the Raspberry every 60 seconds.

Every time a packet is received on the raspberry two count down timers (purple nodes in Figure 3.19) are reset.

In the upper branch, if the timer hits the zero, after 62 seconds, it resets the microcontroller by turning off the power supply and turning it on after 0.1 seconds.

In the lower branch, if the timer hits zero, after 123 seconds, it sends a Telegram error message that indicates that this restart mechanism did not work.

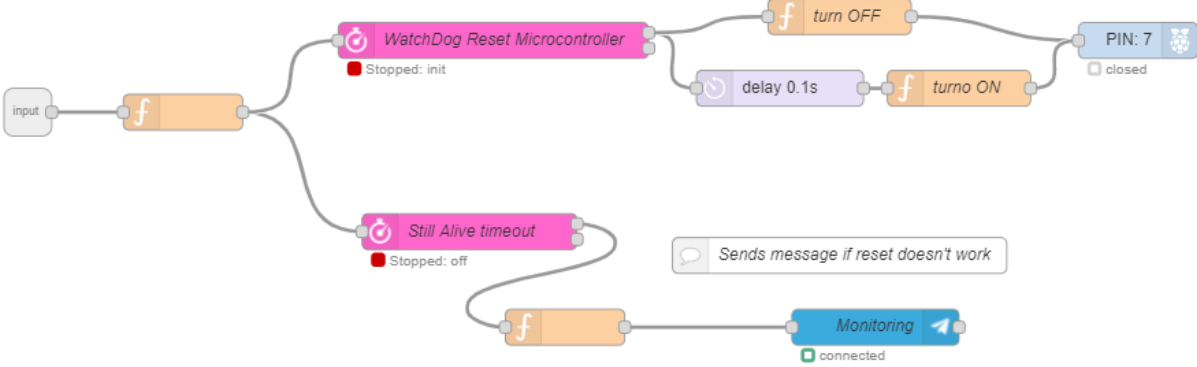


Figure 3.19: watchdog subflow

3.4.2.8 UI: real-time plot sub-flow

Once the flux is computed in the Data Elaboration sub-flow, the payload is sent also to the UI: real time plot sub-flow. Here the current transients and the complete Weber Ampère characteristic are plotted in the user interface page. This becomes particularly useful to have an immediate feedback during the tuning of the microcontroller parameters (sampling time of the ADC, length of the arrays containing the measurements from the test, voltage drop across the diode, range of average for the internal resistance computation).

The first part of the flow (until the join node in Figure 3.20) waits for both the raising and falling branch of the full switching cycle. In this way this sub-flow provides a view of the complete cycle and not just half at a time. The first block in each branch selects the data for the plots and the remaining parts of the branches sets the dynamic scatter plots that will be accessible from the user interface page of Node Red. The output graphs are basically the same shown in Figure 3.5 and Figure 3.12.

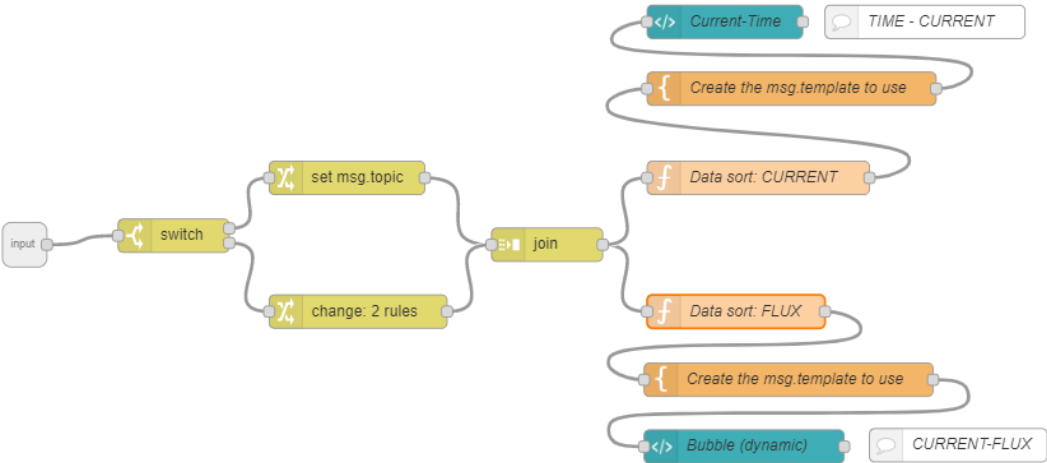


Figure 3.20: UI real time plot subflow

3.5 Amazon Web Services setup

In this section are presented the setups done for each of the used Amazon web services in the building of the cloud part of the architecture. Also, in this explanation, the services are presented using the same order in which they are used during the data processing.

3.5.1 AWS IoT Core

The first service used on the AWS is the IoT Core. It provides a secure and bi-directional communication between devices (Raspberry in this case) and the AWS platform. For setting this service, was used the IoT Console (Figure 3.21) accessible from the AWS Management Console. Using IoT Core two tasks have been implemented:

- 1- Enable a secure communication between Node Red and the AWS IoT
- 2- Connect AWS IoT to another AWS service (AWS S3) for storing the received data

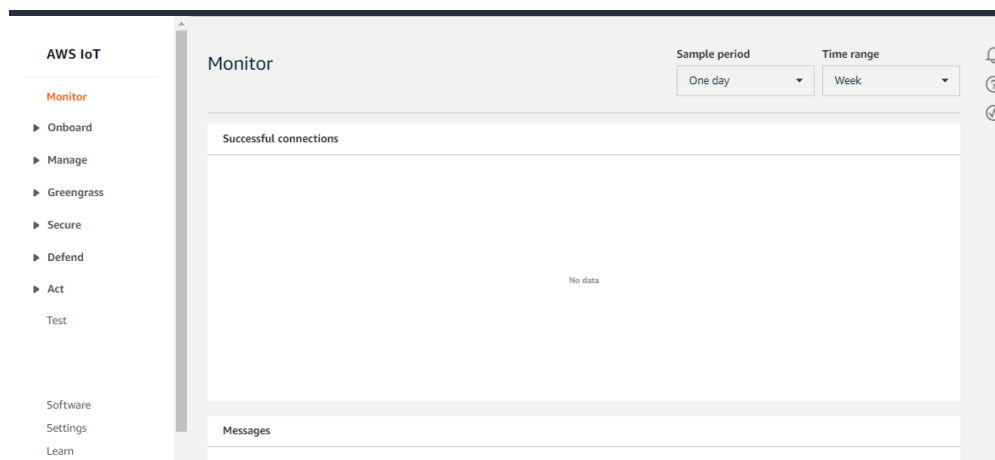


Figure 3.21: IoT Core console

3.5.1.1 Register the Raspberry on AWS IoT Core

To allow the communication with AWS IoT it is necessary to register a device on the service, generate and attach to it certificates and policies for the authentication and authorization (see 2.4.3).

To register a new device the guided procedure proposed in the Manage>Things>Create has been followed. During this procedure it is possible to generate automatically the X.509 certificate for the device authentication, the private key for payload encryption and download the CA root certificate for server authentication (Figure 3.22). These certificates, have to be stored in the raspberry and attached to the MQTT output node (0).

After that, a policy can be created from the Secure>policy>create section in the IoT Core console section. In creating the policy, one must specify which actions the device can perform. Since the raspberry needs just to connect and publish messages on the AWS platform these two actions are specified, while for the topic a wild card is used since the device should be allowed to scale the system by adding valves and related topics (Figure 3.23).

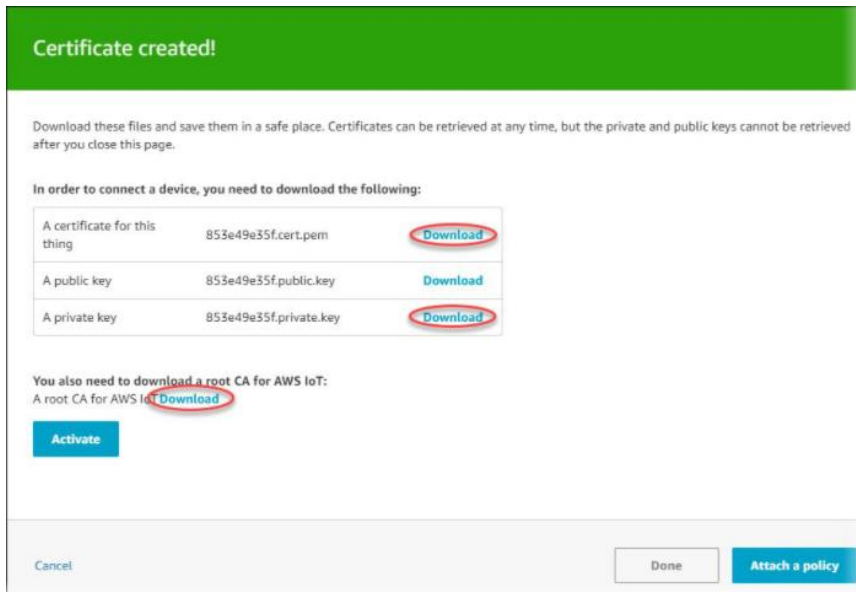


Figure 3.22: IoT device certificates generation



Figure 3.23: IoT device policy permissions

After both the certificates and the policy are generated they have to be attached to the device. To do this in the Secure>Certificate section, select the certificate just generated and from the drop-down menu choose Attach policy and select the just generated one. From the same drop-down menu attach the thing (device) just registered.

With these settings done the Raspberry is allowed to connect to the AWS IoT platform and publish messages on whatever topic, but no action is executed by the AWS service (i.e. the message is not processed or stored).

3.5.1.2 Set an AWS IoT rule

Rules give a registered device the ability to interact with other AWS services. They are analysed and actions are performed based on MQTT topics stream [17].

To set a rule from the IoT console (Figure 3.21) go into the Act>Rule>Create section. The first step in the procedure is to use an SQL statement to filter the received messages on an MQTT topic. In this system the entire message will be stored, so the statement consists in selecting the whole payload from a specific topic.

```
1 SELECT * FROM 'Solearnoid/Raspberry01/VYKA1'
```

Also, during this procedure, the action that AWS IoT performs when executing the rule has to be selected. In the case of study the entire message is stored in an S3 bucket previously created (see 3.5.2) In the final step the attribute of the payload used for naming the stored object has to be selected (bucket key structure at paragraph 3.4.2.6). In this phase it is also necessary to enable the rule by creating a IAM role that AWS IoT can use when performs requests to AWS S3. This step is done during the same setup

procedure: an existing role can be used or a new one can be created and saved. When another valve is added to the system the same role can be used for setting this passage.

The screenshot shows the 'Configure action' interface in the AWS IoT console. At the top, there is a blue header with the text 'Configure action'. Below this, a red icon and the text 'Store a message in an Amazon S3 bucket' are displayed. A descriptive line states: 'This action will write the message to a file in a S3 bucket.' The configuration section includes two main fields: '*S3 bucket' and '*Key'. The '*S3 bucket' field is a dropdown menu with 'solearnoid-vyka-valve1' selected, accompanied by a 'Create a new resource' button. The '*Key' field contains the placeholder text '\$(bucketkey)'. Below these fields, a section titled 'Choose or create a role to grant AWS IoT access to perform this action.' shows a dropdown menu with 'Raspi2_Acton_Role' selected and a 'Policy Attached' indicator. To the right of this dropdown are 'Create Role' and 'Select' buttons.

Figure 3.24: IoT rule settings

With this implementation of the IoT service the Raspberry can send the messages over a secure connection and every time a message is received on the AWS platform, the rule stores the message in an S3 bucket. To keep the data from different valves separated during the whole process, it is important to set a new rule for each topic (uniquely associated to a single valve) and store the message in a dedicated S3 bucket.

3.5.2 AWS Simple Storage Service (S3)

AWS S3 allows users to store and retrieve object of almost all types at anytime. Amazon S3 stores data as objects within buckets. Buckets are basically containers for objects with a controlled access.

In the developed architecture this service is used for storing the raw data (each transient data is contained in a file with structure shown in Figure 3.16). An S3 bucket is created for each running valve to store the data generated from the embedded system separately. No particular settings are done at this level since the communication with the other services is not implemented at this point.

Using the S3 console in the Buckets>Create section it is possible to create a bucket in few clicks. In this procedure all the default settings are used. The “block *all* the public accesses” setting was not modified. In this way only authorized and authenticated principals can access to the objects contained in the bucket.

3.5.3 AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. This service executes the code only when needed and scales automatically, from a few requests per day to thousands per second [18]. AWS Lambda runs the code on high availability infrastructures and performs all the administration of the computing resource including operating system maintenance, capacity provisioning and automatic scaling [18]. The code is executed in response to events such as changes in an S3 bucket or http requests [18].

The role of this service is to run a code that detects the characterizing features of the flux and current transients. This code is executed every time a new object is put in an S3 bucket and, as a result of the execution, a new Dynamo DB Item containing the detected features is created and written in the appropriate table.

In this section is explained how the service is configured and connected to AWS S3 and DynamoDB, while the list of the relevant features and the algorithm used for the detection will be explained in a separate paragraph (3.6.2).

3.5.3.1 Function setup

A new function has been created from the Lambda console choosing Python 3 as a programming language. Before developing the code, two sets are done:

- *Add a trigger event*: from the function home page (Figure 3.25), in the *configuration-Designer* section, the event that will trigger the function execution has to be chosen. Selecting *+Add trigger* it is possible to choose the service (S3), the name of the bucket and the type of event (*all created objects* in this case)
- *Execution role settings*: the AWS role that the function will assume at every execution has to be customized for the intended operation. As default a role is created with basic execution permissions. In the Permission section of the Lambda home page it is possible to see the name of the role created automatically. In this application the function needs permissions for getting objects from an S3 bucket and for putting Items in a Dynamo DB table. The customization of this role must be done using AWS IAM service. For readability reasons the setting procedure of the role is explained in Appendix C

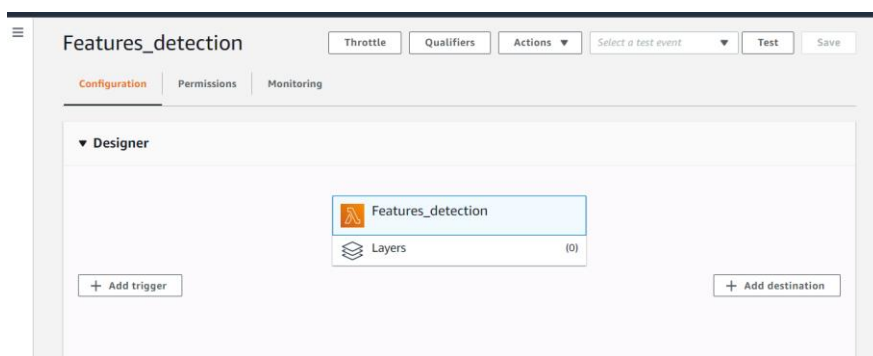


Figure 3.25: AWS Lambda home

3.5.3.2 lambda_handler function

The code executed in response to the triggering event is contained in the *lambda_handler* function.

At every execution, two parameters are automatically passed to the handler function:

- *event*: contains the data of the event that triggered the function execution and is in Python *dict* type. From this parameter are retrievable the name of the bucket and the object key of the last element saved in S3 (triggering event).
- *context*: contains information about the invocation, function and execution environment (amount of memory allocated for the function, amount of milliseconds left before execution timeout, etc).

When using Python as a programming language, the library *boto3* has to be used for interacting with the other AWS services. It is an SDK that provides an easy to use, object-oriented API, as well as low level access to AWS services [19].

To interact with AWS S3 and Dynamo DB, two instances from *boto3* are used

```
s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
```

The procedure implemented in the *lambda_handler* is composed by the following steps:

- 1- from the 'event' parameter the name of the bucket and the object key are obtained. These parameters are used to retrieve the object just stored in the bucket by doing a get request to S3. The object retrieved and stored in a 'content' variable is the same file sent from the Raspberry (Figure 3.16)

```

#_____Get the object from the event and check the bucket
bucket = event['Records'][0]['s3']['bucket']['name']
key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

try:
    response = s3.get_object(Bucket=bucket, Key=key)
    content = response["Body"].read().decode("utf-8")
    content = json.loads(content)
except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. check "event" parameters'.format(key, bucket))
    raise e

```

- 2- For the features detection, it is necessary to have the complete ON-OFF transient, but they are sent and stored in the bucket separately. When an OFF transient is received, the correspondent ON transient is retrieved. The key of the ON part can be computed knowing the structure used for it (paragraph 3.4.2.6)

```

#_____Check test id.. if it is an OFF transient...

if content["testID"]== 0: # falling_branch (switch off transient)
    falling = content
    key = str(int(key)-99) # see bucket primary key composition
# ... retrieve the correspondent ON transient previously stored
try:
    response = s3.get_object(Bucket=bucket, Key=key) # get rising branch (ON transient)
    rising = response["Body"].read().decode("utf-8")
    rising = json.loads(rising)
except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. On transient not found'.format(key, bucket))
    raise e

```

- 3- at this point the two structures containing the transient for a single switching cycle are processed using the *Dynamo_DB_Psi_I* function for the features detection (see paragraph 3.6.2). The result (*new_Item*) is then stored in a Dynamo DB table doing a put request. The table name is composed according to the naming convention used in this architecture (see paragraph 3.5.4)

```

#_____Create the new item

newItem = Dynamo_DB_Psi_I(rising, falling)
Table_name = "SolearnoidValve" + str(falling["valveID"])

#_____Put the new item in the table in Dynamo DB.

try:
    Table = dynamodb.Table(Table_name)
    Table.put_item(Item = newItem)

except Exception as e:
    print(e)
    raise e

```

In Appendix D is reported the full code of the *lambda_handler*.

3.5.4 Dynamo DB

Amazon Dynamo DB is fully managed noSQL database service. The reason for this choice is mainly related to the flexibility of a schema-less database that allows to choose the attributes of each items almost independently and also after the creation of the table. In this implementation this service has been used only for final storage of the results of the detection performed with the Lambda function. The only setting needed at this level is to crate a table for each running valve with the appropriate name. As a convention the name of the table composed be “SolearnoidValve” + valve ID in a single string. In this way no modification to the Lambda code is needed during the scaling of the system. The primary key and sort key were defined as the SwitchCounter and Test ID variables respectively. All the item attributes are presented in the next paragraph.

A new table can be easily created using the Dynamo DB console and no more settings have to be implemented in this phase since the communication with the Lambda service is implemented in it.

3.6 Key features and detection procedure

In this section are presented the features chosen to characterize the current and flux transients during a complete switching cycle. Also, the detection procedure implemented in the Lambda service will be explained.

3.6.1 Detected features

In this first system development, the features chosen as the most meaningful to characterize the shape of the W-A characteristic are the following:

- 1 The five *peak points* shown in Figure 3.26. Each point P_i corresponds to a couple of coordinates (i_i, ψ_i) . In particular P_1 and P_2 correspond respectively to the start and the end of the plunger movement during the on switching cycle. P_4 and P_5 are related to the off switching. The position of those points can give indications on the status of some components of the valve. For example about the internal spring stiffness (flux coordinate of P_1 is related to the force needed to move the plunger against the spring force) or status of the coil windings (if the same steady state current (P_3) generates a lower flux, a short circuit in the coil is possible).

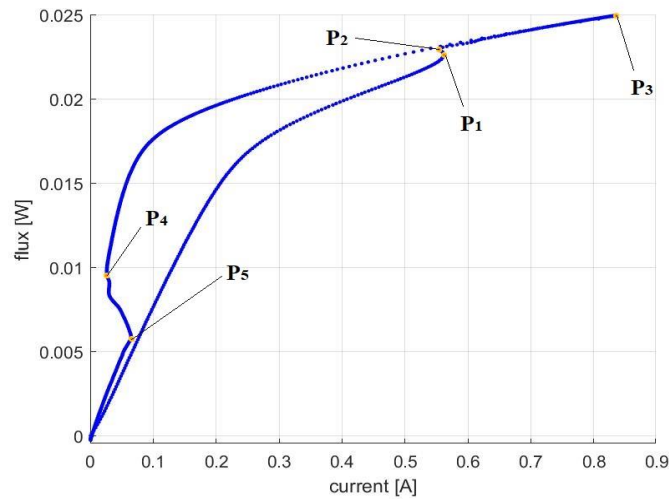


Figure 3.26: peak points in W-A characteristic

- 2 The *variations* of current and linked flux during the on and off switching. These quantities are related to the mechanical movement of the plunger

$$\begin{aligned} \Delta i_{ON} &= i_{P2} - i_{P1} & \Delta i_{OFF} &= i_{P5} - i_{P4} \\ \Delta \psi_{ON} &= \psi_{P2} - \psi_{P1} & \Delta \psi_{OFF} &= \psi_{P5} - \psi_{P4} \end{aligned}$$

- 3 The *path lengths* of the curve between the peak points P_1 - P_2 (l_{ON}) and P_4 - P_5 (l_{OFF}).

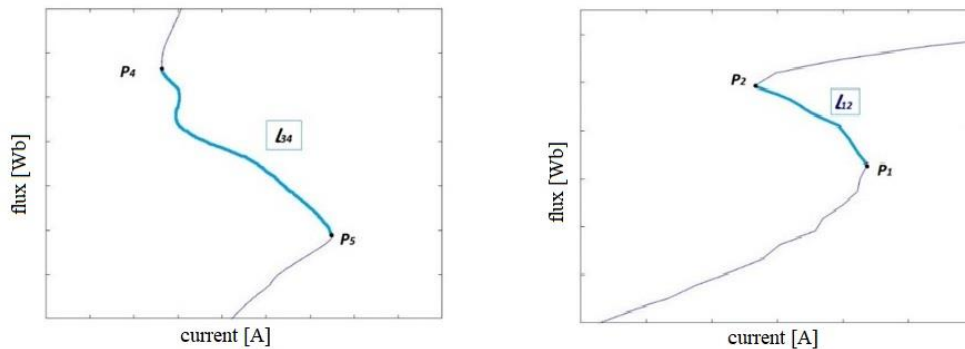


Figure 3.27: path lengths during the switches

- 4 The *area* enclosed in the curve: this quantity is related to the energy dissipated by the valve due to hysteresis effects and mechanical friction.
- 5 The *time delay* between the input voltage is applied and the plunger completes its movement (Figure 3.28). Also these quantities are related to the status of the internal components of the valve (coil windings and spring stiffness)

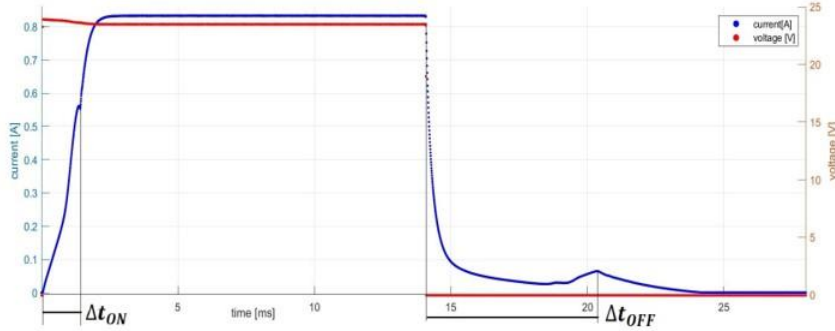


Figure 3.28: time delays in the switching

- 6 The *current standard deviation* computed in the steady state region of the ON switching current transient. This quantity gives an indication on the average amplitude of the noise in the current signal.

Other quantities such as the *internal resistance* of the coil, the *switch counter* and the metadata such as the valve and test ID and the timestamp are taken directly from the JSON file stored in the S3 bucket. The complete list of the features is reported in Figure 3.29 and it represents the attributes of each item in Dynamo DB

$$\{ i_{P1}, i_{P2}, i_{P3}, i_{P4}, i_{P5}, \psi_{P1}, \psi_{P2}, \psi_{P3}, \psi_{P4}, \psi_{P5}, \Delta i_{ON}, \Delta i_{OFF}, \Delta \psi_{ON}, \Delta \psi_{OFF}, l_{ON}, l_{OFF}, A, \Delta t_{ON}, \Delta t_{OFF}, \sigma_{curr}, R_v, timestamp, switchcounter, valve ID, test ID \}$$

Figure 3.29: list of characterizing features

3.6.2 Detection procedure

The approach used for the detection of the features presented in the previous paragraph is a simplistic one and involves the direct usage of the measurement dataset without further processing techniques such as filtering or curve fittings. The reasons for that are mainly related to the AWS Lambda limitation.: the execution time of a function and the importable modules for code development are limited. As a consequence, no complex algorithms have been developed for the detection.

1 points P_1, P_2, P_4, P_5

The detection and computation of the majority of the features depends on the detection quality of the points P_1, P_2, P_4, P_5 (see Figure 3.26).

The first step to detect those points is to find the index of the peaks in the current transient since the flux transient does not present local maxima or minima around the switching area.

To detect the local maximum point in the current transient, the array containing the current measurement is scanned (Code 3.1 line 159) and each value is compared with the previous values and the following values to check if it is a local peak (Code 3.1 lines 165 to 168).

This check is done only for values of the current between the 1% and the 95% of the maximum (Code 3.1 line 163). This is done to avoid the peak detection in the steady state regions in which the peaks

are due to the noise effect. In Code 3.1 is shown the code for the maximum detection, but the approach used for the minimum detection is the same.

```

159     for i in range(window_size , len(Current)-window_size):
160         new_max = True
161
162         # if the current value is between 1% and 95% of the maximum value
163         if (Current[i] > current_min*max(Current) & (Current[i]<current_max*max(Current):
164             #compare the current value in position i with the others in the window
165             for j in range (-window_size, window_size+1):
166                 if Current[i] < Current[i+j]:
167                     new_max = False
168                     break

```

Code 3.1: local maximum detection in current array

The number of values with which every current point is compared is set using the *window_size* variable (Code 3.1) and it changes for the ON and OFF transients. The ON transient is much faster than the OFF and this leads to a lower density of points around the switching region for the ON branch. With a trial and error procedure the two windows have been set to 40 for the OFF branch and 5 for the ON branch.

Once the peaks have been detected, their indexes in the current arrays are used to retrieve the flux coordinate in flux array. In this way the coordinates of the points P₁, P₂, P₄, P₅ are determined.

2 Time delays

The index of the peaks of current correspondent to the end of the switching (P₂ and P₅) are used to compute the time delays. Knowing the sampling time it is possible to compute the time between the input voltage is applied and the switch has completed by multiplying the number of samples between those two events by the sampling period (Code 3.2).

```

278     if (branch["testID"]==1): # Rising branch
279         for i in range(0,len(Voltage)):
280             if(Voltage[i]>1):
281                 input_start_index = i
282                 break
283         switching_delay = (end_switch_index - input_start_index) * sampling_period

```

Code 3.2: switching time computation

3 Paths length

Those quantities are computed as the sum of the distance between each couple of consecutive points on the W-A curve between points P₁-P₂ and P₄-P₅. In Code 3.3 line 233 and 234 is shown the code used for the computation of the switch ON path length knowing the index of point1 and point2.

```

231     length = 0
232     if(Branch["testID"] == 1):
233         for i in range(point1, point2 +1):
234             length += sqrt( ((Flux[i+1]-Flux[i])**2) + ((Current[i+1] - Current[i])**2) )

```

Code 3.3: path length

4 Area

The area enclosed in the cycle is computed as the difference of the area subtended by the OFF and the ON branch. The value of the area of the two transients is computed using a discrete trapezoidal approximation. Code 3.4 shows how the area subtended by a single branch is computed.

```
125     for i in range(0,len(Flux)-1):
126
127         Dy = Flux[i+1] - Flux[i]
128         y = Flux[i] + Dy/2
129         Dx = Current[i+1] - Current[i]
130         dA = y*Dx
131         A += dA
```

Code 3.4: Area computation

5 Current standard deviation

Is computed using the equation (12) in which the average value of the current is computed in the steady state part of the switch on transient

4 Endurance test results

4.1 Introduction

4.1.1 VYKA valve a-priori information and working condition

Before showing the results of the endurance test it is necessary to do some considerations.

It was possible to have some a-priori information about the valve degradation from the datasheets from Festo. It is guaranteed a correct working of the valves until it reaches 10 million of cycles in rated conditions. In this working conditions the most probable cause of failure of the valve is due to a weakening of the torsional spring that pulls the plunger back in position (Figure 3.2 (5)). This leads to an increase of the fluid leakages (NC-COM) over 1 ml/h that is threshold over which the valve is considered in failed condition.

During the endurance test the valve is aged while working in the following condition:

- No fluid supply is connected to the valve ports and this does not affect the health condition of the valve
- The valve is aged switching at 17 Hz using an input voltage with a low duty cycle (22%). This is done in order to have an average current in the coil that is below the admitted value for a service S1 (coil always energized). On the other hand, the maximum switching frequency allowed in rated condition is 9 Hz when the valve works in a cool ambient ($T < 20^{\circ}\text{C}$). This should bring to a faster decline of the torsional spring condition.
- The valve is kept in the same position and orientation during the endurance test. this is done to avoid that changes of the pose of the valve lead to variations in the detected features values.

The valve worked in the described conditions until it reached 12 million of cycles to be sure that the component is in failure condition.

4.1.2 W-A aging evolution types

From the modifications of the W-A characteristic shape during the aging it is possible to have information about the health status of the valve. In general, depending on the type of shape change, it is also possible to read what kind of failure is happening in the valve. As an example, in Figure 4.1 are shown two types of aging.

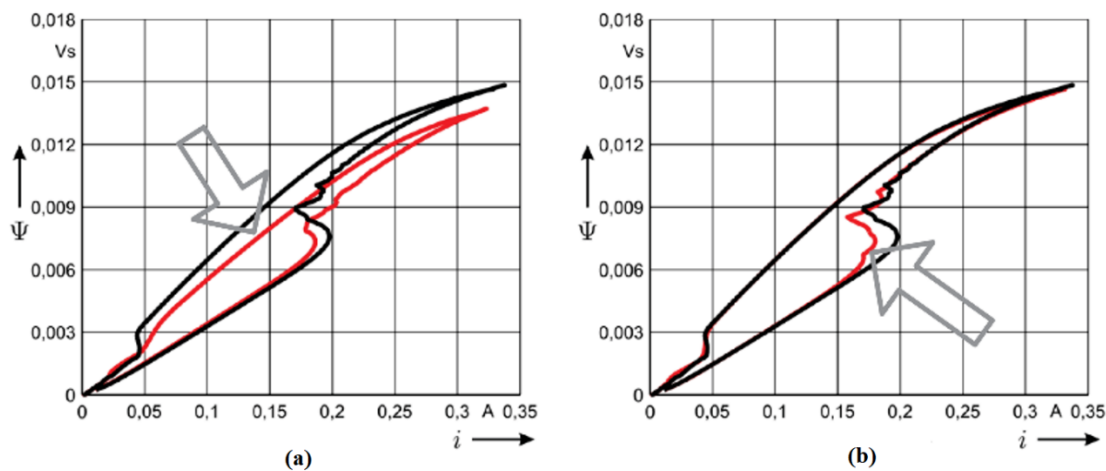


Figure 4.1: W-A aging examples
(a): coil short circuit (b): spring weakening

In the first one (a) the cause of failure is the aging of the insulating material in the coil, while the other components are still properly working. In this scenario the flux needed for moving the plunger is the same needed for a healthy valve since the spring is not aged. Due to the short circuit in the coil, to produce that amount of flux it is needed a higher amount of current than the one needed in healthy condition. Moreover, the short circuit leads to a lower number of turns in the coil windings. This means a lower self inductance of the solenoid (equation (11)). This leads to a change in the magnetization curves in “blocked plunger” condition (paragraph 2.1.7.1)

The second type of aging represented in Figure 4.1 (b) as associated to an aging of the spring that brings back in position the plunger when the coil is de-energized. In this case the amount of flux needed to win the force generated by the spring is lower than the one needed for a new valve. The self inductance of the coil remains the same since the electrical part of the valve is still working properly. In this scenario the start and the end of the switch happen at a lower value of both the current and the flux coordinates. Through These two examples, it can be understood how in principle different changes of the W-A shape mean different types of failure.

4.1.3 Endurance test regions

After the start of the endurance test still some modifications to the overall system for data generation, processing and collection were needed. In particular the system was successfully stabilized after 2.3 million of cycles performed by the valve. In Figure 4.2 is plotted the value of the internal resistance with respect to the number of cycles performed. The average resistance is used as an indication of the temperature of the coil. This plot is useful to do some considerations provide some explanation about the shapes of the plotted results since all the feature will be presented with respect to the number of performed switches.

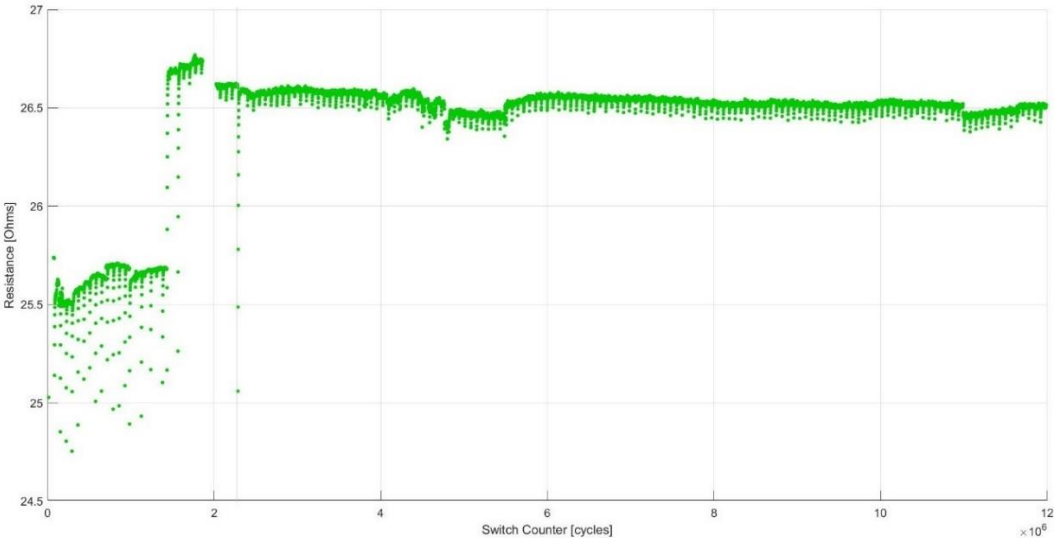


Figure 4.2: average internal resistance evolution

It is possible to distinguish two regions in the plot. Before the 2.2 millions cycles (grey vertical line in Figure 4.2) it was necessary to stop the data collection many times to solve technical issues. Every time a reboot was performed, it caused a thermal transient due to the cooling of the valve during the interruption. Every time the system was started the temperature, and so the internal resistance, increases until it reached a steady state.

After the stabilization of the system the internal resistance stabilized as well, and we can consider this stable region as an almost constant temperature region.

The first region is not useful to evaluate the aging effect on the detected values, but will be used to show the temperature effect on them. On the contrary, the almost constant temperature region can be used to show the aging effect without the temperature contribution. The assumption done in this analysis is that

there is not combination effect between the temperature and the age on the measured value of the features.

4.2 W-A evolution overview

First of all, a high-level view on how the complete characteristic changes during the endurance test is given. In Figure 4.3 are plotted curves that represent full switching cycles at different values of the number of performed switches, but at the same value of temperature (not all of them belong to the constant temperature region).

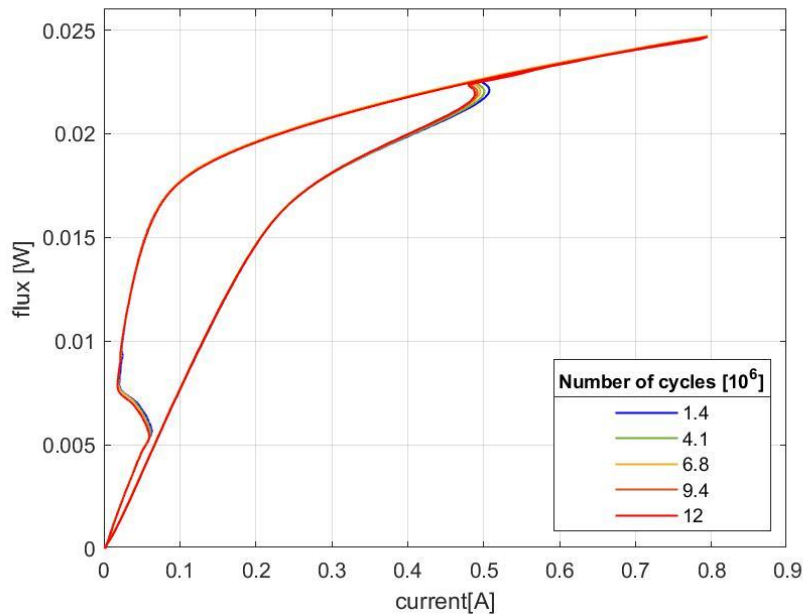


Figure 4.3: W-A aging overview

As it can be seen the type of shape change during the aging is similar to the one presented in Figure 4.1 (b). this means that the working condition of the valve are made the stiffness of the torsional spring decrease.

Although the variation of the shape is well visible from a high-level point of view, not all the detected features have been measured with the same accuracy during the endurance test.

4.3 Detected features evolution

In this paragraph are presented the evolution of these detected values during the aging of the valve.

The majority part of the features presents a linear relation with the number of performed cycles in the whole constant temperature region or in a subregion. A linear regression has been done to each of the features that show this behaviour in a defined range and the resulting coefficient of determination has been used to evaluate the strength of the best fit line (see paragraph 2.2.2).

Among all the detected features, the ones that presented a clear variation during the type of aging described in paragraph 4.1.1 are the coordinates of the points P_1 , P_2 , P_4 and P_5 (Figure 3.26) and the time delays (Figure 3.28). As an example of how the evolution of these quantities looks like, in Figure 4.4 and Figure 4.5 are reported the plots of the current coordinate of points P_1 and P_5 (respectively linear behaviour in the whole region and in a subregion)

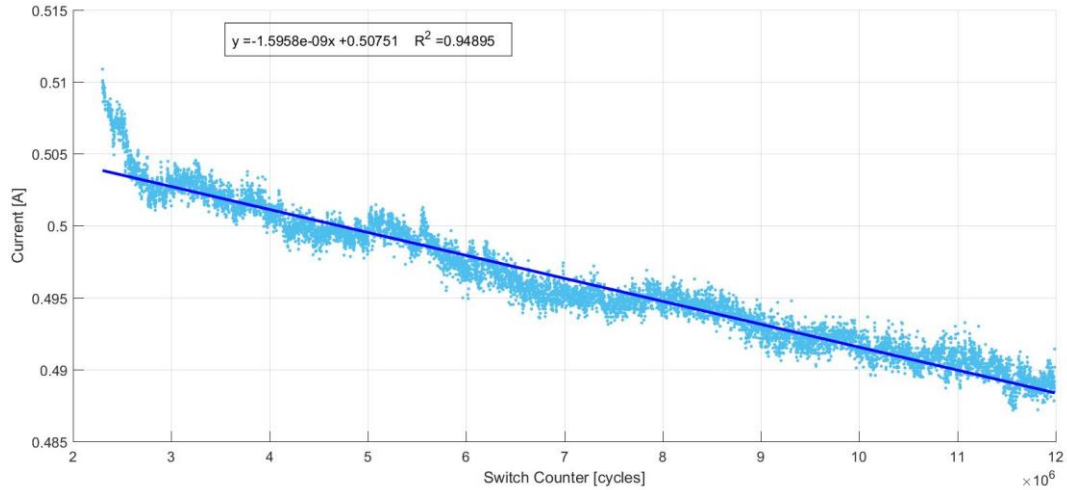


Figure 4.4: current (P1) evolution

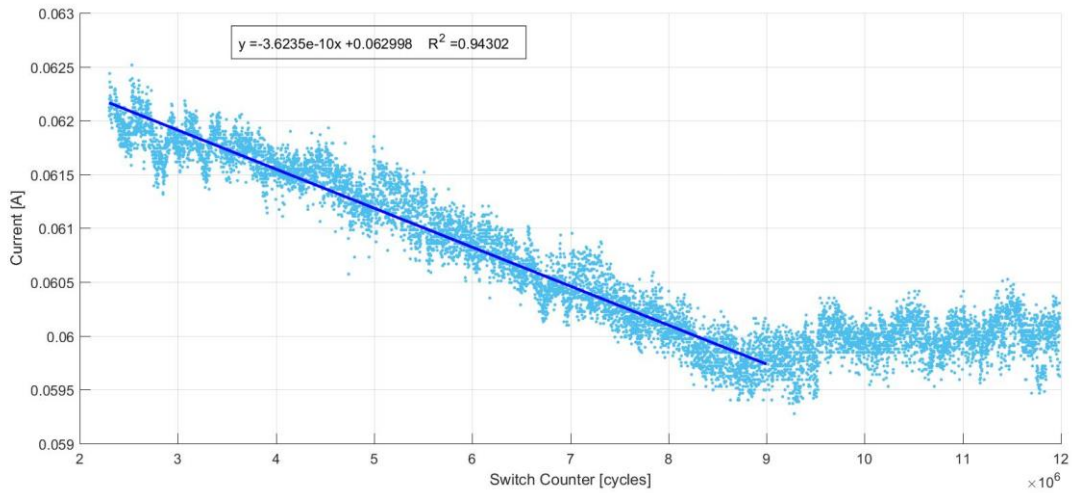


Figure 4.5: current (P5) evolution

In the Table 4.1 : regression parameters for aging effect are contained the results of the regressions (parameters described in paragraph 2.2.2) for the values of the coordinates and time delays mentioned before and their range of validity.

feature	a	b	R ²	Range (10 ⁶ cycles)
Current (P1)	-1.596 10 ⁻⁹	0.508	0.949	2.3 - 12
Current (P2)	-1.504 10 ⁻⁹	0.498	0.946	2.3 - 12
Current (P4)	-2.322 10 ⁻¹⁰	0.020	0.752	2.3 - 9
Current (P5)	-3.624 10 ⁻¹⁰	0.063	0.943	2.3 - 9
Flux(P1)	-1.866 10 ⁻¹¹	0.022	0.587	2.3 - 12
Flux(P2)	-1.614 10 ⁻¹¹	0.023	0.663	2.3 - 12
Flux(P5)	-2.721 10 ⁻¹¹	0.006	0.617	2.3 - 9
ON delay	-2.611 10 ⁻¹²	0.0013	0.840	2.3 - 12
OFF delay	2.077 10 ⁻¹¹	0.007	0.780	2.3 - 9

Table 4.1 : regression parameters for aging effect

Looking at the coefficients of determination it is easy to see that the current coordinates are detected with a higher accuracy than the flux ones. The motivation of this will be explained in section 5.2. This is the same reason for which the point P4 presented an acceptable evolution in its current coordinate while the flux coordinate did not show a clear trend (high spreading of the data and R^2 less than 5%). The low order of magnitude of the parameters a and b are due to the units of measure used (Ampères, Webers, seconds, cycles).

The features that present a good linear trend in the subregion (between 2.3 and 9 millions of cycles), always stabilize their value after the 9 millions cycles as it happens in Figure 4.5. This happens mainly to the quantities relates to the OFF part of the cycle (P4, P5 and OFF delay).

The features not mentioned in Table 4.1 can be separated in two categories of behaviour. In the first one the detected quantities show a weak linear correlation with the number of performed cycles. Nevertheless, the high spread of the detected values leads to low values of the coefficient of determination (R^2 below 20%). Belong to this group the area of the cycle, the switch on path length and the variation of the current coordinate during the switch on. As an example for this group, in Figure 4.6 is reported the evolution of the area during the endurance test.

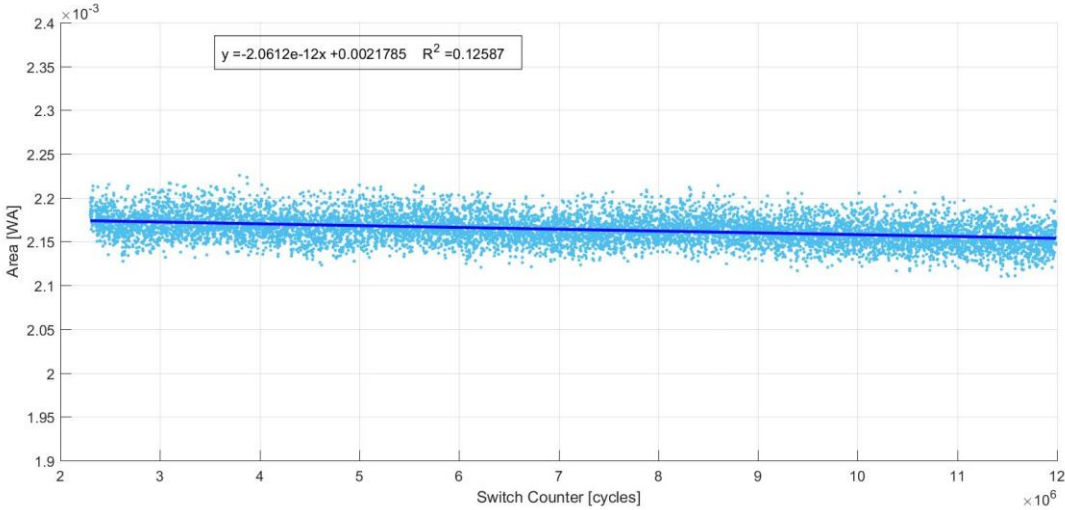


Figure 4.6:W-A area evolution

The remaining features that characterize the W-A characteristic don't show a correlation with the number of cycles performed and their behaviour can not be approximated and described with a straight line nor with other types of simple regressions (i.e. polynomial regressions). Belong to this group the variations of flux during the switches, the variation of current and the length of the path during the off switch and the coordinates of the point P3. As an example for this category in Figure 4.7 is shown the evolution of the variation of the current during the switch off

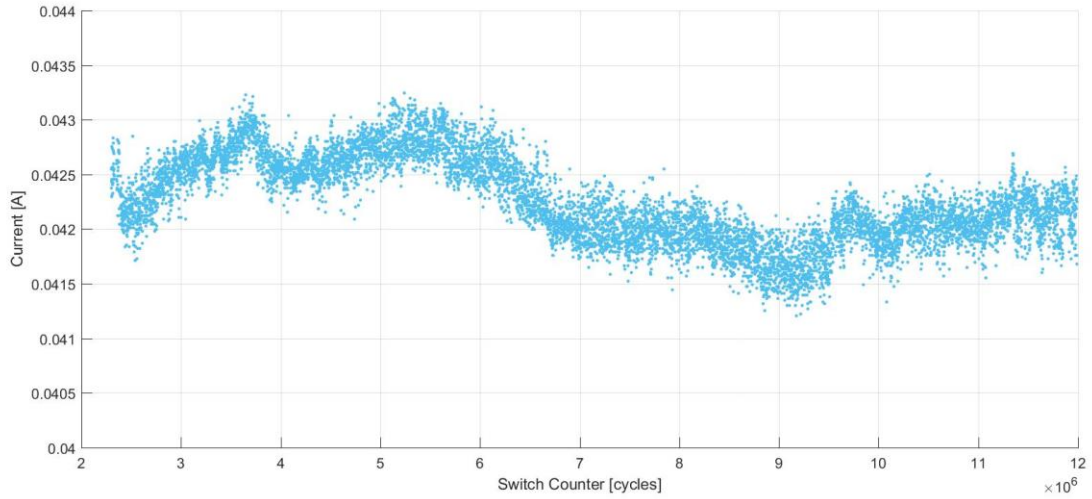


Figure 4.7: current variation during OFF switching evolution

In this chapter the correlation between aging and features values was described as a linear one when possible. It is important to remark that this behaviour is valid only after 2.3 millions of cycles. Before this point the quantities present a variation heavily different from a linear one. The cause of this is not certain and could be due to inconsistency coming from the setup changes, due to the thermal transients that occurred at every reboot of the system, to a change of aging influence at the begin of the experiment or due to other unknown factors.

5 Discussion

In this chapter are discussed the factors that influence the values detected with the presented setup. In particular, the discussion is focused on how the temperature modifies a part of the features and how the system can be improved to increase the accuracy of the detection.

5.1 Temperature effect

In the previous chapter the presented results were the ones belonging to the almost constant temperature region (see paragraph 4.1.3). In that part of the experiment the setup was stable, and no big thermal transient occurred over the time.

In the beginning region of the experiment (before 2.3 millions of cycle performed (Figure 4.2)) the collected data are not fully consistent due to a number of changes in the setup. Nevertheless, this region gave the possibility to see how the coil temperature affects the values of the detected features.

In this section id discussed the relation with the temperature of the different quantities and then is presented the result of a first attempt of compensation using the linear models obtained from the first point.

5.1.1 Temperature dependencies

In order to evaluate the correlation with the temperature of the features, the variation of the different quantities are analysed during different thermal transients. In each of those transient the amount of cycles performed is low (order of 10^4 cycles) if compared with the life of the valve. In Figure 5.1 is shown a thermal transient described through the internal resistance evolution.

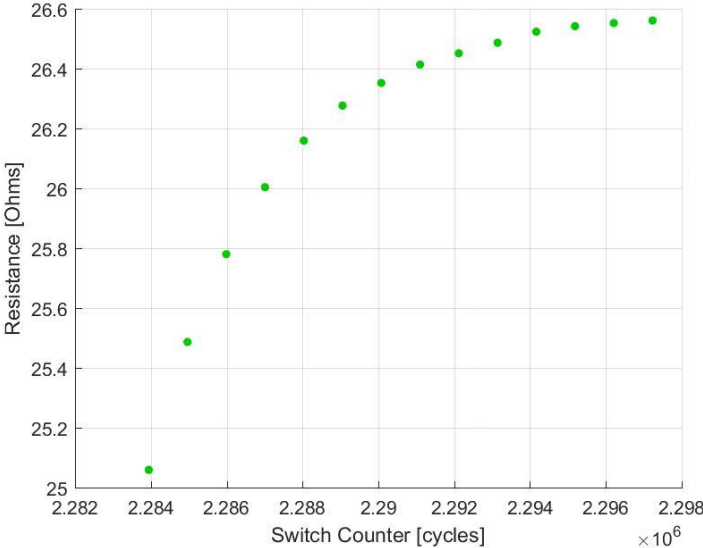


Figure 5.1: resistance evolution during a thermal transient

Inside these ranges of switch counts the correlation between detected values and temperature has been evaluated using linear regressions. This types of regression was again suitable to evaluate the dependencies due to the limited range of resistance (temperature) variation.

In figure is shown the correlation diagram between the current coordinate of the point P1 and the internal resistance during the transient shown in Figure 5.1

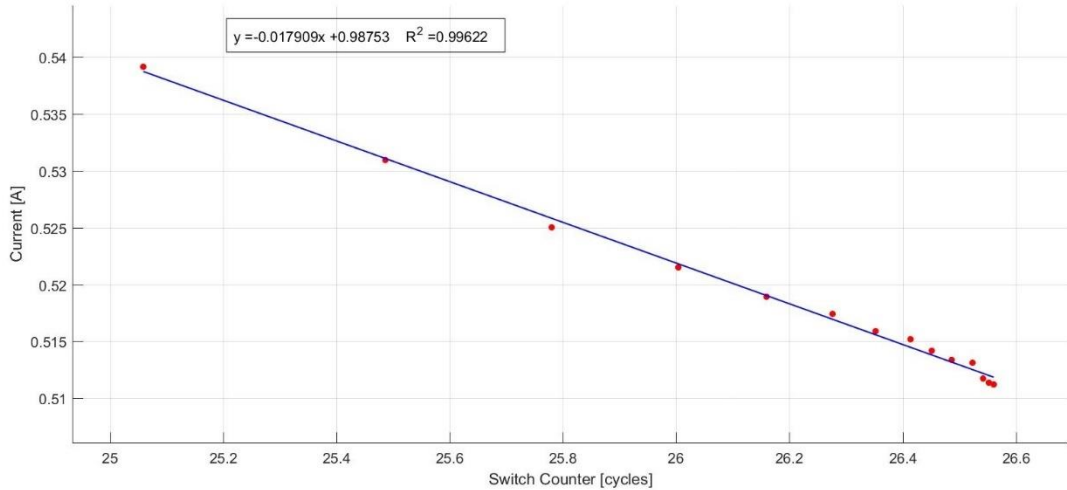


Figure 5.2: correlation diagram current(P1) - resistance

A linear regression for all the features has been performed in different transient regions. Overall for those features that show a relation with the temperature variations, the results of the linear regression changed significantly with the switch count. Moreover, as explained in the previous chapter, the system detection is not fully reliable in the first part of the experiment. For these two reasons the regression results from the last thermal transient (Figure 5.1) have been assumed as the most correct. In fact, during the last transient the system was already stabilized and was able to produce reliable results.

feature	a	b	R ²
Current (P1)	-1.791 10 ⁻²	9.875 10 ⁻¹	0.996
Current (P2)	-1.817 10 ⁻²	9.834 10 ⁻¹	0.996
Current (P3)	-2.776 10 ⁻²	1.531 10 ⁻¹	0.999
Flux (P1)	-1.417 10 ⁻⁴	2.600 10 ⁻²	0.958
Flux (P2)	1.741 10 ⁻⁴	2.718 10 ⁻²	0.960
Flux (P3)	1.596 10 ⁻⁴	2.896 10 ⁻²	0.964
ON delay	-4.713 10 ⁻⁶	1.447 10 ⁻³	0.72
OFF delay	-1.187 10 ⁻⁴	9.87 10 ⁻³	0.943

Table 5.1: linear regression parameters for temperature correlation

5.1.2 Temperature and aging effects comparison

From the previous paragraphs we know that both the temperature and the number of switches have an effect on the value of the detected features.

Let's look at the tables 5.1 and 4.1 and in particular let's focus on the features present in both two the tables (current and flux of points P1, P2 and time delays). The quantities under analysis are the ones that show a clear linear evolution during the aging, but also a strong correlation with temperature variations. To compare the magnitude of the two effects it is necessary to look at the slope coefficients of the best fit line ("a" column). It can be seen how for each of the quantities under analysis the variation of one Ohm of the internal resistance leads to a variation in the value of the feature that is of the same order of magnitude of the variation caused by 10 millions of cycles (total age of the valve). The only exception among these features is the on delay that is more robust to temperature variations.

In Figure 5.3 are reported the details of the current transient around the switching regions. Figure 5.3 (a) and (b) represent how the shape of the off switching region changes with aging and temperature respectively. Figure 5.3 (c) and (d) show the same comparison for the on switching region.

Observing the two plots for the off switching we can see how the time delay (time coordinate of the maximum point of the curves) changes in a comparable way with the temperature and the aging. On the other hand the current coordinates of the peaks P4 and P5 are almost not affected by temperature variations. For the switch on transient the situation is the dual one: the two current peaks are strongly affected by temperature variations, while the time delay (time coordinate of the local minimum) is more robust to them.

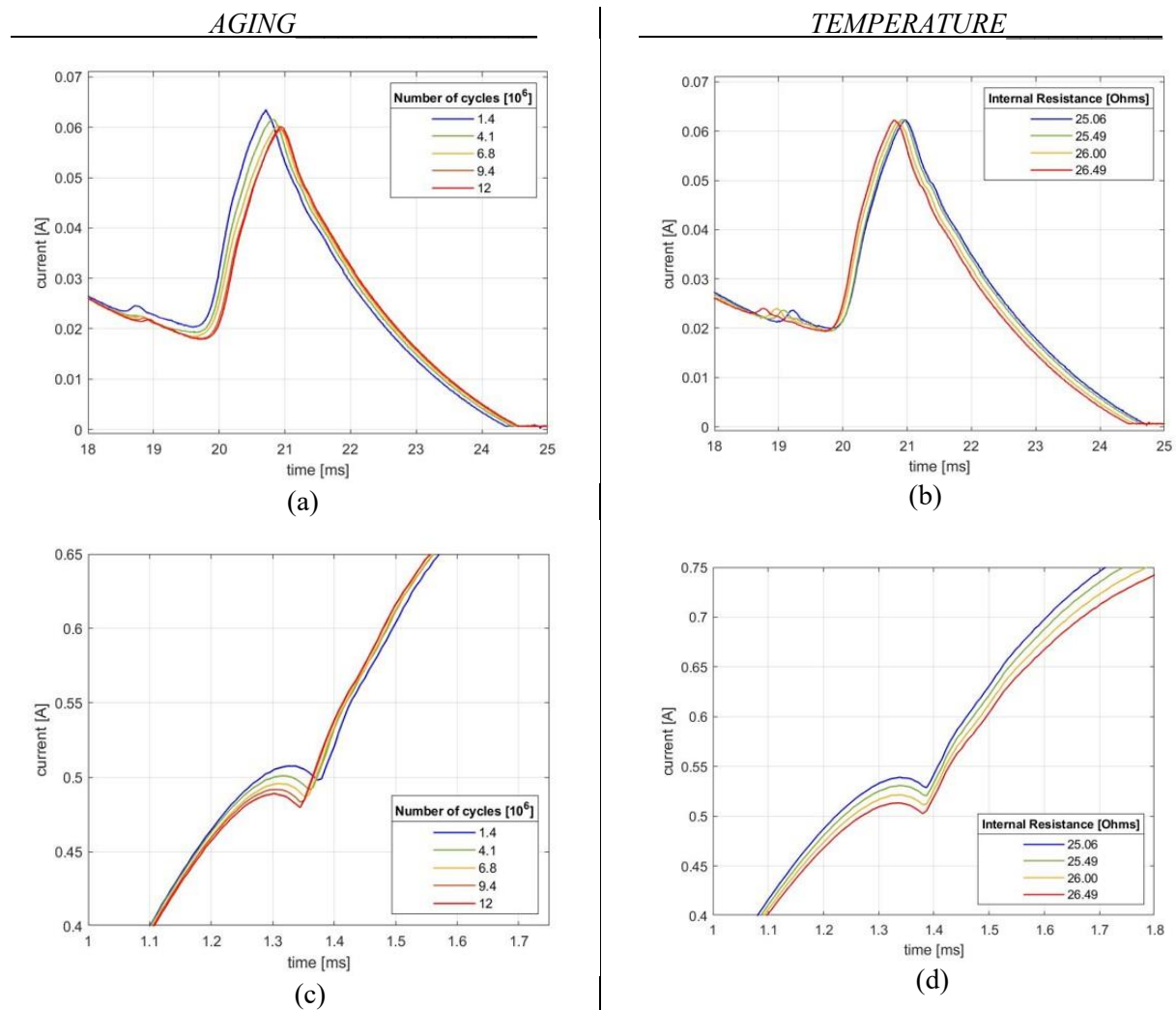


Figure 5.3: comparison between aging and temperature effects

5.1.3 Temperature effect compensation

The detected features values will be used to estimate the residual life of the valve by finding a relationship between their variation and the real health status decline of the valve itself. For this reason it is important that the measured variation of the features is due only to the aging of the valve and not to temperature variations. For this reason it is necessary to be able to compensate the temperature effect on the measured values.

In this thesis a first attempt of compensation is done by using the models obtained from the linear regression reported in Table 5.1: linear regression parameters for temperature correlation.

The compensation of the temperature contribution has been performed by neglecting the possible combination effect with the aging. In general a quantity Q that depends on temperature and aging can be describe with a function of the type

$$Q(T, age) = q_0 + q_1(T) + q_2(age) + q_3(T \cdot age) + \varepsilon \quad (20)$$

Considering $q_3 = 0$ we are assuming that the model of Q is a simple linear additive one. In this thesis the internal resistance of the valve has been used as an indicator of the internal temperature. The used model for each quantity Q is

$$Q = q_0 + q_1 \cdot R_V + q_2 \cdot (\text{switch count}) + \varepsilon \quad (21)$$

In which q_1 is the slope coefficient of the best fit line obtained from temperature correlation (Table 5.1: linear regression parameters for temperature correlation and q_2 is the slope of the line coming from the regression used to model the aging effect (Table 4.1).

To remove the resistance contribution, all the measurements of the quantity Q have been projected at the same value of resistance $R_{Vreference}$. The projection has been done in the following way

$$Q(R_{Vreference}) = Q(R_{Vi}) + q_1 \cdot (R_{Vreference} - R_{Vi}) \quad (22)$$

Where $Q(R_{Vreference})$ is the value projected at the reference temperature, $Q(R_{Vi})$ is the quantity measured at R_{Vi} .

The compensation in equation (22) has been applied to the quantities listed in Table 5.1. After that, the linear regression used to model the aging effect has been performed again to evaluate if the temperature compensation brought some improvements. The results of the new regression are resumed in Table 5.2.

feature	a	b	R ²
Current (P1)	-1.730 10 ⁻⁹	0.509	0.953
Current (P2)	-1.640 10 ⁻⁹	0.499	0.950
Flux (P1)	-1.971 10 ⁻¹¹	0.022	0.609
Flux (P2)	-1.722 10 ⁻¹¹	0.023	0.689
ON delay	-2.646 10 ⁻¹²	0.0013	0.844
OFF delay	1.986 10 ⁻¹¹	0.007	0.765

Table 5.2: regression for aging modelling after temperature compensation

Comparing the results between the first and the second regression (Table 4.1 and Table 5.2 respectively) it can be observed, by looking at the coefficients of determination, that the accuracy of the models don't increase significantly. This is mainly because the region of compensation is also the region at almost constant temperature.

Anyway, performing the compensation in the complete aging range (from 0 to 12 millions of cycles) of the features listed in Table 5.2, a qualitative improvement can be observed. As an example of this improvement, in Figure 5.4 and Figure 5.5 are shown the complete evolution of the switch off time delay respectively before and after the temperature compensation.

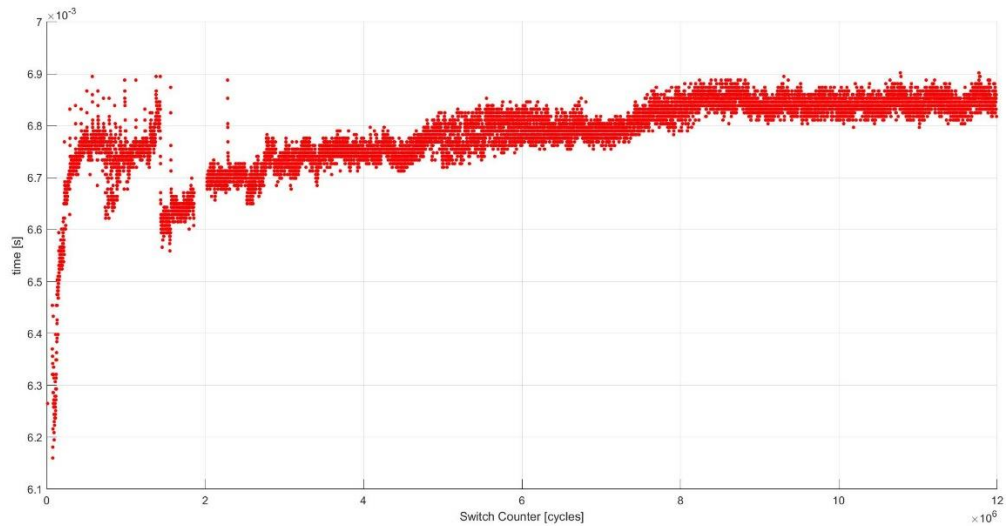


Figure 5.4: switch OFF time delay complete evolution

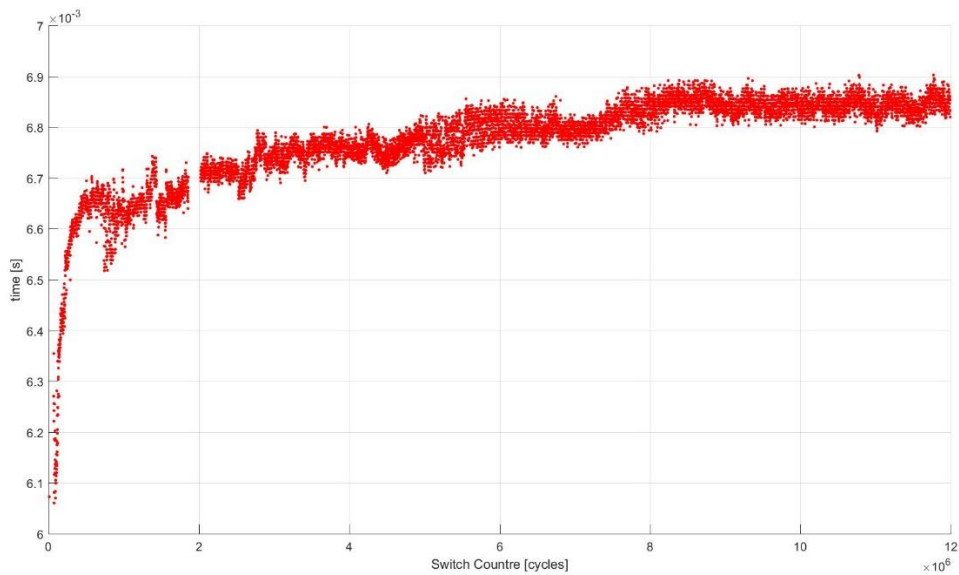


Figure 5.5: switch OFF time delay before and after temperature effect compensation

Although the linear models of the temperature effects on the features gave overall good results, they can not be validated in a suitable way due to a lack of reliable datasets for validation. Moreover, the models for temperature effects should be able to normalize the values in a wider range of variations. In fact, the parameters obtained from linear regression are suitable for compensation in the range of temperature in which the models have been built.

For these reasons other experiments must be done with the objective of building a model that can be used for temperature compensation of the detected features.

5.2 Detection accuracy limitations

In this section are presented the limitations of the used procedure for the feature detection and which are the possible improvements for increasing the accuracy.

5.2.1 Flux and current coordinates

As explained in paragraph 3.6.2, the approach used for the features detection was based on finding the local maximum and minimum point around the switching area in the current transient. The information used for the detection is the very same one measured from the microcontroller (no signal processing or curve fitting techniques are applied). Once these points are detected the flux and time coordinate associated to the point are used for the point definition. This approach leads to a lower accuracy in the detection flux and time information.

In Figure 5.6 is represented a detail of the on switching region of the W-A characteristic. The blue smooth line represents the real curve, while the orange one represents the information available from the system. The real peak of the curve is in the position of the blue point and the detected one is the red point. It can be seen how on one hand the detection accuracy of the current quantity can be in general acceptable, while the detection accuracy of the flux coordinate is in general lower.

Moreover, the noise in the signal, brings some randomness in the detected peak position index around the real peak.

This limitation can be seen in the results of the regression in Table 4.1. The coefficients of determination of the regressions done on flux quantities are much lower than the ones associated to the current quantities. In fact, the values of detected flux coordinates are much more spread than the current one. The same limitation applies to the time quantities (ON and OFF delays) since they are detected using the same approach.

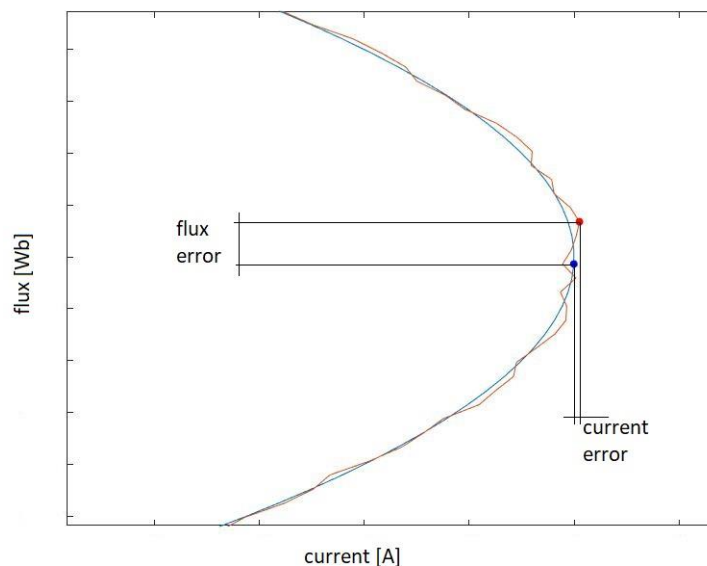


Figure 5.6: flux and current detection accuracy

5.2.2 Quantization effect

As mentioned in the previous paragraph the value of the detected time delay is affected by the low accuracy of the detection algorithm. The accuracy of these values is further decreased by a quantization effect caused by the sampling time used for triggering the ADC. Considering how the time delays are computed (3.5.3.2), their values can be only multiples of the sampling period. This effect becomes evident in particular looking at the ON time delay evolution since the overall variation of this quantity during the endurance test is comparable with the quantization step Figure 5.7: quantization effect (ON delay evolution).

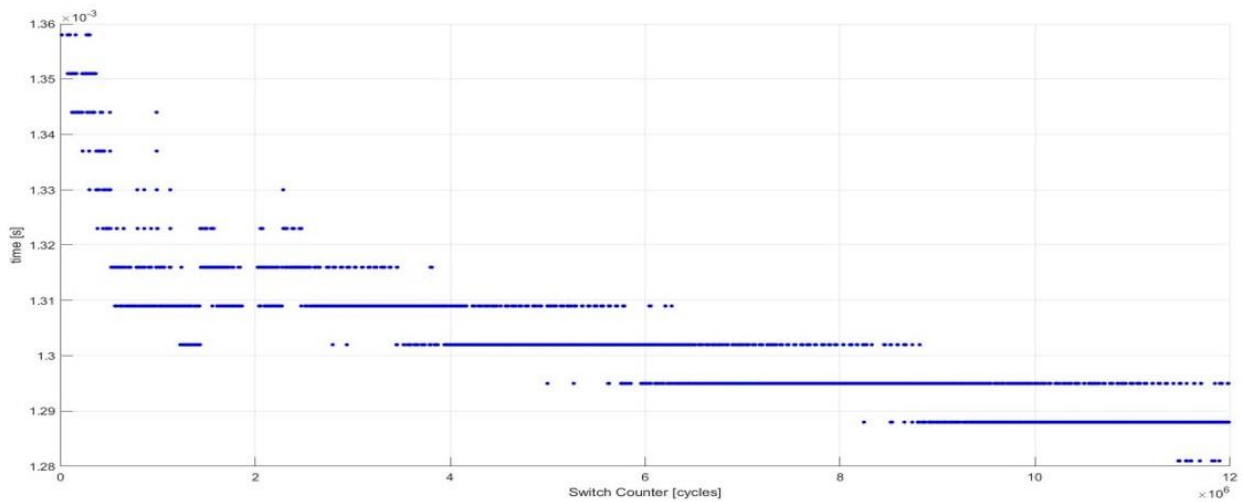


Figure 5.7: quantization effect (ON delay evolution)

Another particular quantization effect can be observed in the variation of the flux during the switch ON (Figure 5.8). This effect is due to the low accuracy effect explained in the previous paragraph.

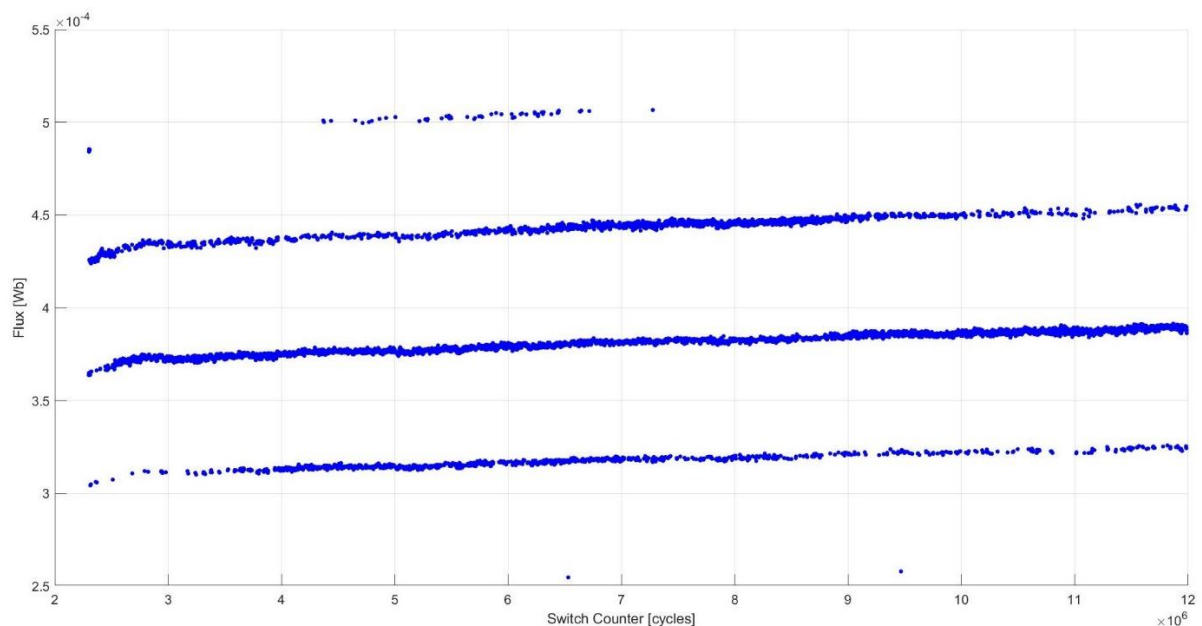


Figure 5.8: Flux variation during the On switch

5.2.3 Possible detection improvement

If during the further development of the complete project will turn out that this system is not accurate enough a possible solution could be using *curve fittings techniques*.

Curve fitting is a process of building a “smooth” curve that best approximate the set of measurement data points. Once the fitting is performed, the detection can be done on the obtained smooth curve that in principle should remove the noise and quantization effect explained in the previous section.

The drawback of this approach is the computational power required. As explained in paragraph 3.6.2, the reason why the implemented detection algorithm was a simplistic one are related to the limitation of the Lambda service (importable libraries and execution timeout). For this reason implementing a curve fitting technique would probably require a change of the architecture using another computing service (such as AWS EC2).

Moreover, an increase of the complexity of the detection algorithm would probably increase the computational power needed for the final monitoring system. This could make a difference on the hardware needed in the predictive maintenance implementations, that in the worst case could need the usage of cloud computing.

6 Summary

During this thesis was successfully implemented a system able to store and process data generated from a microcontroller during a solenoid valve monitoring. In particular, the implementation consists in:

1. A functionality on a microcontroller able to export the generated data in almost real time. This implementation is a light-weight version of the UDP/IP protocol and makes usage of an Ethernet connection.
2. A secure connection with AWS Cloud platform through the usage of a Raspberry Pi using Node Red. On this component is also implemented a pre-processing of the data generated by the test stand for a more efficient manipulation on the Cloud.
3. An architecture on Amazon Web Services platform that provides:
 - a. A processing of the generated Current and flux transient measurements. As a result of this, some the features that characterize those transients are successfully detected with a good accuracy
 - b. Two levels of storage for the generated data: in AWS S3 are stored the raw data generated during measurement cycles and test procedures; in AWS Dynamo DB are stored the results of the features detection.

The system has been tested during an endurance test of a solenoid valve. During this test, the microcontroller generated data continuously. The implemented detection algorithm worked accurately with some of the monitored quantities: current peaks during the switching transients and input/output time delays. With this setup those quantities are good candidates for being used in the monitoring of the valve status for doing predictive maintenance.

The analysis of the data collected during the endurance test shew that the temperature of the internal coil of the valve strongly affect the value of some of the detected features. An initial attempt to model and compensate this effect was performed with some good results. Nevertheless, other test are needed to be able to compensate the temperature effect in a wider range of variation and also to validate the built models.

7 References

- [1] Ke-Sheng Wang, Zhe Li, Jørgen Braaten, Quan Yu, “interpretation and compensation of backlash error in machine centers fo intelligent predictive maintenance”, *Advances in Manufacturing*, 2015, 3(2): 97-104, [online]: <http://html.rhhz.net/AIM/html/127.htm#R-8>
- [2] R. Keith Mobley, “An introduction to predictive maintenance”, 2 ed., Butterworth-Heinemann, 2002
- [3] Zhe Li, Kesheng Wang, Yafei He, “Industry 4.0 – Potentials for predictive maintenance”, *International workshop of advanced manufacturing*, 2016
- [4] Stephen J. Chapman, “Electric Machinery Fundamentals”, 4 ed., McGraw-Hill, 2005
- [5] M. Ruderman and A. Gadyuchko, "Phenomenological modeling and measurement of proportional solenoid with stroke-dependent magnetic hysteresis characteristics", 2013 IEEE International Conference on Mechatronics (ICM), Vicenza, 2013, pp. 180-185
- [6] A. Reznitchenko, A. Rekov and E. Shantaev, "Development and Research of Device for Measuring Weber-ampere Characteristics of Electrical Products with Ferromagnetic Magnetic," 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), Sochi, Russia, 2019, pp. 1-5
- [7] Kevin R. Fall, W. Richard Stevens, “TCP/IP illustrates”, v.1, ed.2, Addison-Wesley, 2012
- [8] LwIP Wiki, LwIP application developer manual, [online]: https://lwip.fandom.com/wiki/LwIP_Application_Developers_Manual, 2020
- [9] Adam Dunkels, “Design and implementation of the LwIP TCP/IP stack”, Swedish Institute of computer science, 2001
- [10] LwIP 2.0.2, [online]: https://www.nongnu.org/lwip/2_0_x/index.html, 2020
- [11] Marek Neuzil, “Processor Expert UDP+DHCP/LwIP demo application for FRDM-K64F target board”, NXP Community, 2015, [online]: <https://community.nxp.com/docs/DOC-275614>
- [12] Node Red official page, [online]: <https://nodered.org/>, 2020
- [13] Borko Furth, Armando Escalante, “Handbook of cloud computing”, ed.1, Springer, 2010
- [14] Vladimir Fedak, “Cloud computing architecture: layers of cloud computing”, 2019, [online]: <https://medium.com/@FedakV/cloud-computing-architecture-layers-of-cloud-pyramid-b2e70f877c91>
- [15] Golden Bernard, “AWS for dummies”, John Wiley & Sons, 2013
- [16] AWS IAM User Guide, [online]: <https://docs.aws.amazon.com/IAM/latest/UserGuide/intro-structure.html>, 2020
- [17] AWS IoT User Guide, [online]: <https://docs.aws.amazon.com/iot/latest/developerguide>
- [18] AWS Lambda User Guide, [online]: <https://docs.aws.amazon.com/lambda/latest/dg>

- [19] boto3 documentation, [online]: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>
- [20] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam, Rong Xiang, “Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry, ed.1, 2012
- [21] OpenStax College, “Introductory statistics”, ed.1, v.1, Textbook Equity Edition, 2013
- [22] John O. Rawlings, Stastry G. Pantula, David A. Dickey, “Applied regression analysis: a research tool”, ed.2, Springer, 1998

8 Appendix A: hardware setup for LwIP

In this section are reported the hardware settings needed to use the LwIP library on the NXP FRDM K64F when using Processor Expert for the hardware configuration.

Before implementing the solution for the used architecture, some configurations have to be done [11].

1. Clock configuration (Figure 8.1)

- System Oscillator 0 > External oscillator at 50 MHz
- New Clock configuration 6 > PEE
- Core Clock 100 MHz, Bus Clock 50 MHz, External Bus Clock 25 MHz and Flash Clock 25 MHz

Name	Value	Details
Component name	clockMan1	
Component version	1.2.0	
▲ Clock management		
▲ Clock settings		
▲ Clock sources		
Internal oscillator		
RTC oscillator	Enabled	
▲ System oscillator 0	Enabled	
▲ Clock source	External reference clock	
▲ Clock input pin		
Pin name	RME_RXCLK	EXTAL0/PTA18/FTM0_FLT2/FTM...
Clock frequency [MHz]	50.0	50 MHz
▲ Clock configurations	7	
Clock configuration 0	clockMan1_InitConfig0	
Clock configuration 1	clockMan1_InitConfig1	
Clock configuration 2	clockMan1_InitConfig2	
Clock configuration 3	clockMan1_InitConfig3	
Clock configuration 4	clockMan1_InitConfig4	
Clock configuration 5	clockMan1_InitConfig5	
▲ Clock configuration 6	clockMan1_InitConfig6	
Description	Ethernet 25MHz EXTAL0 external oscillator	
Internal reference clock		
▲ External reference clock		
OSC0ERCLK clock	Enabled	
OSC0ERCLK in stop	Disabled	
OSC0ERCLK clock [MHz]	50.0	50 MHz
ERCLK32K clock source	Auto select	RTC oscillator
ERCLK32K clock [kHz]	0.032768	0.032768 MHz
▲ MCG settings		
MCG mode	PEE	
MCG output clock	PLL clock	
MCG output [MHz]	100.0	100 MHz
MCG external ref. clock source	System oscillator 0	
MCG external ref. clock [MHz]	50.0	50 MHz
Clock monitor	Disabled	
▲ FLL settings		
FLL module	Disabled	
FLL output [MHz]	0.0	0 MHz; FLL is disabled.
MCGFFCLK clock [kHz]	32.5520833	32.552083333333 kHz
▲ Reference clock source	External clock	
Reference clock divider	Auto select	1536
FLL reference clock [kHz]	32.5520833	32.552083333333 kHz

Figure 8.1: Clock configuration

2. Processor Expert components

- fsl_enet component added into the PEx project
- fsl_debug_console component added into the PEx project. It provides debug output and it is configured as shown in Figure 8.2

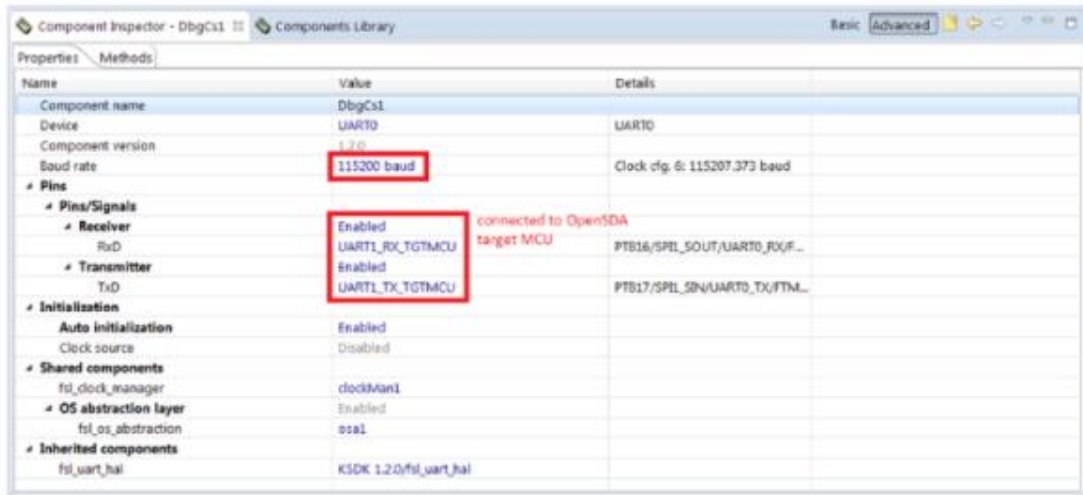


Figure 8.2: fsl_debug_console configuration

3. Add the Lwip libraries to the project path

For compilation and linking of the project are necessary also following LwIP library:

```
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\port"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\port\arch"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include\ipv4"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include\ipv4\lwip"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include\lwip"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include\netif"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include\posix"  
"${PROJECT_KSDK_PATH}\middleware\tcpip\lwip\src\include"
```

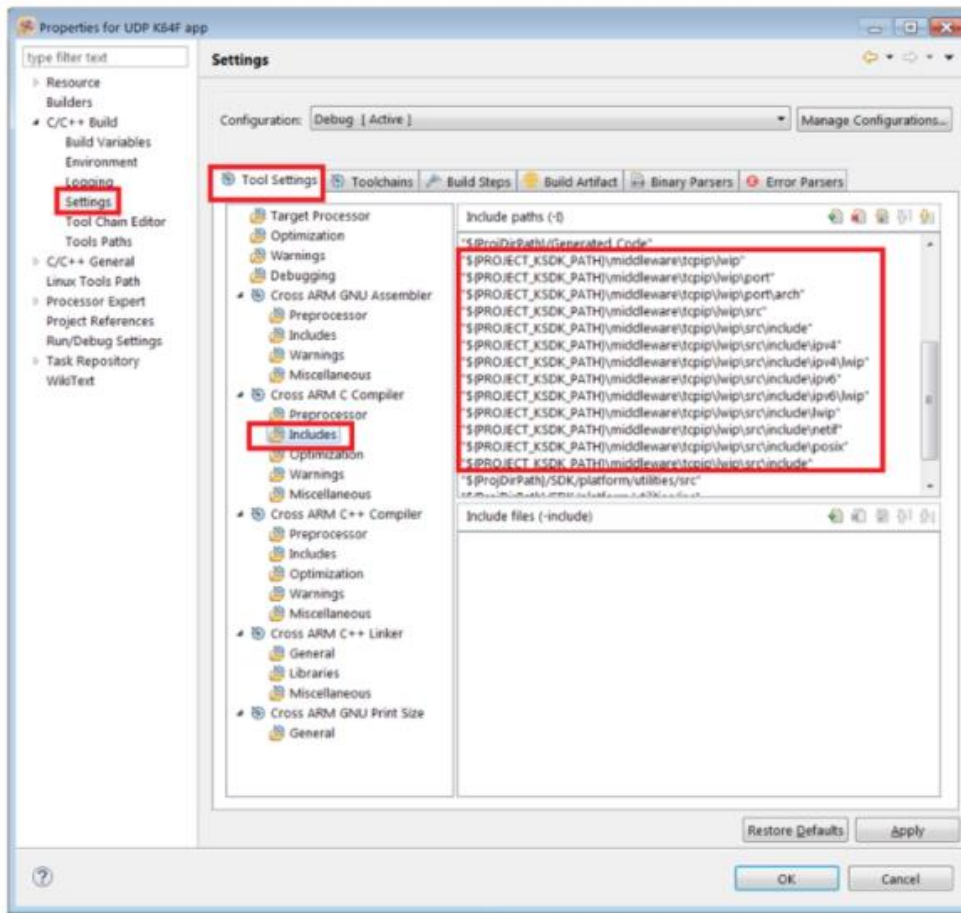



Figure 8.3: add Lwip libraries to the project path

In this phase it is important to import just the modules needed for the stack implementation and, in the imported modules, remove the #include dependencies on the removed modules. If this is not done the whole LwIP library is imported, compiled and built.

9 Appendix B: eNet.c

```
#include "eNetCom.h"
#include "lwip/opt.h"
#include <string.h>
// Standard C Included Files
#include <stdio.h>
// lwip Included Files
#include "lwip/udp.h"
#include "lwip/debug.h"
#include "netif/etharp.h"
#include "lwip/init.h"
// SDK Included Files
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "ethernetif.h"
#include "board.h"

// lwip/udp global variables
uint16_t port = 1500; // Raspberry open port for sending data
uint16_t valve_ID = 1; // set the ID here
struct udp_pcb *pcb; // protocol control block structure for udp protocol
struct netif fsl_netif0; // network interface structure
ip_addr_t fsl_netif0_ipaddr, fsl_netif0_netmask, fsl_netif0_gw, Raspi_address; // used IP addresses

// initializations for the udp communication (called once before starting the tests scheduling)
int udp_enet_init(void) {

    app_low_level_init();
    lwip_init();

    // IPv4 addresses (32 bits)
    IP4_ADDR(&fsl_netif0_ipaddr, 169,254,186,155); // microcontroller static IP address
    IP4_ADDR(&fsl_netif0_netmask, 255,255,255,0); // Network mask
    IP4_ADDR(&fsl_netif0_gw, 169,254,186,153); // Default gateway address = Raspi_address
    IP4_ADDR(&Raspi_address, 169,254,186,153); // Raspberry IP address

    netif_add(&fsl_netif0, &fsl_netif0_ipaddr, &fsl_netif0_netmask, &fsl_netif0_gw, NULL,
              ethernetif_init, ethernet_input); //Add a network interface to the list of lwIP netifs
    netif_set_default(&fsl_netif0); // Set the network interface as the default network interface
    netif_set_up(&fsl_netif0); // Bring the interface up, available for processing traffic

    // get new pcb
    pcb = udp_new();

    if (pcb == NULL) {
        LWIP_DEBUGF(UDP_DEBUG, ("udp_new failed!\r\n"));
        return -1;
    }

    //set the remote address as the Raspberry's one
    udp_connect(pcb, &Raspi_address, port);

    return 0;
}

void app_low_level_init(void)
{
    // Open uart module for debug
    hardware_init();

    // Open ENET clock gate
    CLOCK_SYS_EnableEnetClock(0);
    // Select PTP timer outclk
    CLOCK_SYS_SetEnetTimeStampSrc(0, kClockTimeSrcOsc0erClk);

    // Disable the mpu
    MPU_BWR_CESR_VLD(MPU, 0);
}
```

```

err_t udp_enet_send(uint16_t data1[], size_t size ){

    int data_size = (size + 4)*2; // total size of the payload = [ sizeADCbuf + 4(header) ] *
sizeof(uint16)

    struct pbuf *new_p; //packet buffers

    new_p = pbuf_alloc(PBUF_TRANSPORT, data_size , PBUF_RAM);

    if(new_p != NULL){
        memcpy(new_p->payload, data1, data_size);
        udp_send(pcb, new_p);
        pbuf_free(new_p);
    }

    OSA_TimeDelay(5);
    return err;
}

```

10 Appendix C: Lambda Role settings

In the presented architecture, the implemented Lambda service needs the permission to access to the S3 bucket objects and the Dynamo DB tables. The role crated automatically during the creation of the lambda function gives just basic execution permissions to the service. To add other permission, other policies must be attached to the created role using the AWS IAM service.

To do this, in the IAM console, navigate in the Roles section (Figure 10.1) and select the role with the name found in the “permissions” section of the Lambda service console (paragraph 3.5.3.1).

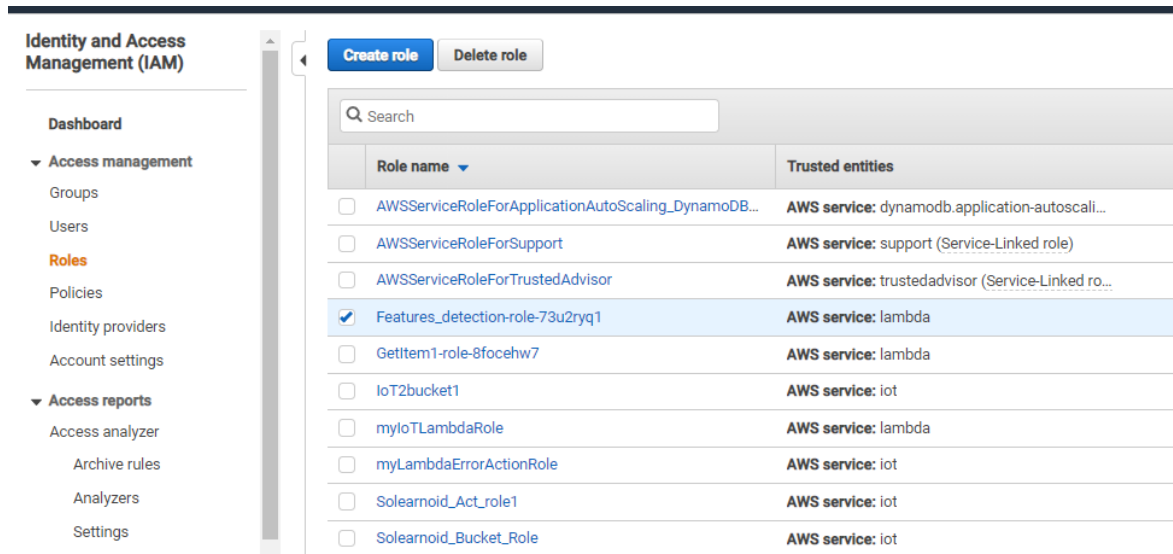


Figure 10.1: IAM console

In the next window, select “attach policy” and search for the “AmazonS3ReadOnlyAccess” and “AmazonDynamoDBFullAccess” and then “attach policy”. Doing this procedure the policies associated to the role give more permission than what is really needed by the Lambda function (the service can perform any action on the dynamo Db tables instead of just writing items in it). Anyway, if the function is carefully used, this over-dimensioned permission does not create any problem. Moreover, the advantage of this approach is that the role does not need further modifications during the scaling of the system.

Alternatively, to give the minimum permissions to the lambda function, a policy can be manually created instead of choosing one of the ready-to-use policies.

11 Appendix D: lambda_handler

```
import json
import urllib.parse
import boto3
import sys
from Psi_I_process import *

s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

def lambda_handler(event, context):

    # _____ Get the object from the event and check the bucket
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        content = response["Body"].read().decode("utf-8")
        content = json.loads(content)
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. check "event" parameters'.format(key,
bucket))
        raise e

    # _____ Check test id.. if it is an OFF transient...

    if content["testID"]== 0: # falling_branch (switch off transient)
        falling = content
        key = str(int(key)-99) # see bucket primary key composition
    # ... retrieve the correspondent ON transient previously stored
    try:
        response = s3.get_object(Bucket=bucket, Key=key) # get rising branch (ON transient)
        rising = response["Body"].read().decode("utf-8")
        rising = json.loads(rising)
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. On transient not foun'.format(key,
bucket))
        raise e

    # _____ Create the new item

    newItem = Dynamo_DB_Psi_I(rising, falling)
    Table_name = "SolearnoidValve" + str(falling["valveID"])

    # Put the new item in the table in Dynamo DB.

    try:
        Table = dynamodb.Table(Table_name)
        Table.put_item(Item = newItem)

    except Exception as e:
        print(e)
        raise e

    else:
        pass

return
```

12 Appendix E: System scaling

When a test on a new valve has to be started, some settings have to be done at different levels for the correct working of the system:

- The software running on the microcontroller needs just the modification of the valveID and the port number variables of the Raspberry (see next point).
- In Node Red, the flow can be duplicated with the following modification for each valve:
 - In the first node used to receive the packets from the microcontroller, a free port must be chosen (3.4.2.2).
 - For the “switch counter sanity check” sub-flow a must be used dedicated file for keeping track of the switch counter. In Attachment A the “valve.txt” must be duplicated and renamed. The path of this new file must be added in the “write file” and “read file” in the sub-flow (3.4.2.5)
 - In the “to AWS cloud” node set the dedicated topic on which the valve will publish the payloads. (Figure 3.17: To AWS Cloud node configuration -a)
- On AWS IoT Core a new rule should be created to associate the new topic to the new bucket (paragraph 3.5.1.2).
- An S3 bucket must be created for containing the new valve data (paragraph 3.5.2)
- In the Lambda service the update of the new S3 bucket should be added as a triggering event (paragraph 3.5.3.1)
- A new AWS Dynamo DB table must be created to collect the results of the detection (paragraph 3.5.4)

In case the number of valves connected to a single Raspberry generate a too high throughput to be processed, it could be necessary to use another Raspberry for further scale the system. In this case, besides the setting described before, it is also needed to register the new device on AWS IoT core (paragraph 3.5.1.1) and configure completely the “to AWS Clod” node in Node Red (paragraph 3.4.2.4) using the new certificates and key generated during the device registration.