



**Politecnico
di Torino**

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Meccanica

A.A. 2025/2026

Structural analysis with data-driven Machine Learning for components with native sensors

Relatore:

Giorgio De Pasquale

Candidato:

Giacomo Maria Allora

Luglio 2026

A mio padre.

Index

Figure Index	6
Table Index	9
List of Used Abbreviations	10
1 Introduction.....	11
1.1 Background	11
1.2 Objective	13
1.3 Why Machine Learning.....	13
References	14
2 State of the Art.....	15
2.1 Applications of multifunctional materials	15
2.1.1 Materials and Manufacturing Methods	15
2.1.2 Analytical and Numerical Models	19
2.1.3 Experimental Tests and Characterization	25
2.2 Machine Learning Methods Applied to SHM.....	30
2.2.1 Supervised Algorithms.....	32
2.2.2 Unsupervised Algorithms	32
2.2.3 Deep Learning Algorithms.....	32
2.2.4 Application Examples	33
References	36
3 Component and Experimental Tests	39
3.1 Component	39
3.1.1 Geometry.....	39
3.1.2 Materials.....	41
3.1.3 Process	42
3.1.4 Alternative Specimens	44
3.2 Test Bench and Data Acquisition.....	45
3.3 Datasets	50
3.3.1 Crack Propagation Dataset.....	50
3.3.2 Single Segment Characterization Dataset	54
3.3.3 Alternative Dataset.....	55
4 First Approach: Boundaries	57
4.1 Algorithm Explanation	57

4.1.1	Data Management	57
4.1.2	Training.....	59
4.1.3	Test.....	60
4.1.4	Example	62
4.2	Results	63
4.3	Remarks.....	83
5	Second Approach: Medians	86
5.1	Algorithm Explanation	86
5.1.1	Training.....	86
5.1.2	Test.....	88
5.1.3	Example	90
5.2	Results	92
5.3	Remarks.....	106
6	Third Approach: Gaussians	108
6.1	Algorithm Explanation	108
6.1.1	Training.....	108
6.1.2	Test.....	109
6.1.3	Example	114
6.2	Results	116
6.3	Remarks.....	120
6.4	Graphical Comparison between the First Three Approaches.....	122
7	Fourth Approach: Ideal Model	125
7.1	Algorithm Explanation	125
7.2	Results and Issues.....	129
8	Final Tool.....	132
8.1	Explanation.....	132
8.2	Example.....	135
8.3	Results	137
8.4	Remarks.....	139
9	Conclusions.....	142
9.1	Research Path	142
9.1.1	Initial Problem.....	142
9.1.2	Proposed Technological Solution	142
9.1.3	Working Hypotheses.....	143
9.1.4	Execution of the Work and Verification of Results	144

9.1.5	Final Comment.....	146
9.2	Future Perspectives.....	146
9.2.1	Applications and Provided Advantages	146
9.2.2	Issues to be Further Investigated.....	146
10	Appendix: Matlab Scripts	148
10.1	Build_training_and_test_table.m	149
10.2	Build_boundaries_table.m.....	152
10.3	Build_medians_table.m.....	153
10.4	Build_gaussians_table.m.....	156
10.5	Build_confusion_matrix_broken_branches_boundaries.m.....	159
10.6	Build_confusion_matrix_broken_branches_medians.m.....	161
10.7	Build_confusion_matrix_broken_branches_gaussians.m.....	163
10.8	Final_tool.m	165
10.9	split_test.m	172
10.10	boundaries.m	174
10.11	find_boundaries.m.....	176
10.12	median_func.m.....	177
10.13	mean_medians.m.....	178
10.14	find_medians.m.....	179
10.15	gaussian.m.....	181
10.16	find_gaussians.m.....	183
10.17	Build_ideal_medians_table.m	184

Figure Index

Figure 1: Percolation threshold.	17
Figure 2: Dependence of the conductivity on the conductive filler content.	21
Figure 3: Schematic of a resistor model with a random distribution of CNTs.	25
Figure 4: Modeling of tunneling resistance in the resistor network.	25
Figure 5: Predicted and measured DC-conductivity in terms of tensile strain.	27
Figure 6: Electrical response during loading and unloading.....	28
Figure 7: Reversibility of the electrical response.....	29
Figure 8: Typical components of SHM.....	30
Figure 9: Three common learning algorithms in SHM.....	31
Figure 10: Experimental setup of the ML application example.....	34
Figure 11: Flowchart of the ML application example.	35
Figure 12: Results visualization in the ML application example.....	35
Figure 13: Performances of the ML application example.....	36
Figure 14: Component geometry.	39
Figure 15: R1 circuit.	40
Figure 16: R2 circuit.	40
Figure 17: Pause after layer 2 to insert electrodes.	43
Figure 18: Finite prototype with jumpers.	44
Figure 19: Alternative specimen.	45
Figure 20: Test bench, front view.	46
Figure 21: Test bench, overall view.....	47
Figure 22: Test bench, top view.....	47
Figure 23: Voltage divider.	48
Figure 24: Electrical scheme of an arm of the sensor.	50
Figure 25: ID configurations, CP dataset.....	51
Figure 26: Test in time domain.	52
Figure 27: Segmentation of a test.	53
Figure 28: Test in the displacement-resistance plane.	53
Figure 29: ID1 to ID10 configurations, SSC dataset.	54
Figure 30: ID11 to ID15 configurations, SSC dataset.	54
Figure 31: Alternative test bench.....	55
Figure 32: Example of the region of a family.	60
Figure 33: Angle classification knowing the number of broken branches.	62
Figure 34: Confusion matrix for angle, resistance 1 – Broken branches known, load known.	65
Figure 35: Confusion matrix for angle, resistance 2 – Broken branches known, load known.	65
Figure 36: Confusion matrix for angle, resistance 1 – Broken branches known, load unknown.	66
Figure 37: Confusion matrix for angle, resistance 2 – Broken branches known, load unknown.	66
Figure 38: Confusion matrix for angle, resistance 1 – Broken branches unknown, load known.	67
Figure 39: Confusion matrix for angle, resistance 2 – Broken branches unknown, load known.	67
Figure 40: Confusion matrix for angle, resistance 1 – Broken branches unknown, load unknown.	68
Figure 41: Confusion matrix for angle, resistance 2 – Broken branches unknown, load unknown.	68
Figure 42: Confusion matrix for broken branches, resistance 1 – Angle known, load known.....	69
Figure 43: Confusion matrix for broken branches, resistance 2 – Angle known, load known.....	69
Figure 44: Confusion matrix for broken branches, resistance 1 – Angle known, load unknown.....	70

Figure 45: Confusion matrix for broken branches, resistance 2 – Angle known, load known.....	70
Figure 46: Confusion matrix for broken branches, resistance 1 – Angle unknown, load known.....	71
Figure 47: Confusion matrix for broken branches, resistance 2 – Angle unknown, load known.....	71
Figure 48: Confusion matrix for broken branches, resistance 1 – Angle unknown, load unknown..	72
Figure 49: Confusion matrix for broken branches, resistance 2 – Angle unknown, load unknown..	72
Figure 50: Confusion matrix for load, resistance 1 – Angle known, broken branches known.....	73
Figure 51: Confusion matrix for load, resistance 2 – Angle known, broken branches known.....	73
Figure 52: Confusion matrix for load, resistance 1 – Angle known, broken branches unknown.....	74
Figure 53: Confusion matrix for load, resistance 2 – Angle known, broken branches unknown.....	74
Figure 54: Confusion matrix for load, resistance 1 – Angle unknown, broken branches known.....	75
Figure 55: Confusion matrix for load, resistance 2 – Angle unknown, broken branches known.....	75
Figure 56: Confusion matrix for load, resistance 1 – Angle unknown, broken branches unknown..	76
Figure 57: Confusion matrix for load, resistance 2 – Angle unknown, broken branches unknown..	76
Figure 58: Regions of the plane based on broken branches; angle and load unknown.	78
Figure 59: Regions of the plane based on angles; broken branches and load unknown.....	79
Figure 60: Regions of the plane based on load; angle and broken branches unknown.	79
Figure 61: Regions of the plane based on angles; B1, load unknown.	80
Figure 62: Comparison between scores of the angle families.	81
Figure 63: Regions of the plane based on load; B1, angle unknown.....	82
Figure 64: Comparison between true median and mean of the medians.	89
Figure 65: Broken branches classification without knowing the other variables.	90
Figure 66: Confusion matrix for angle, resistance 1 – Broken branches known, load known.	92
Figure 67: Confusion matrix for angle, resistance 2 – Broken branches known, load known.	92
Figure 68: Confusion matrix for angle, resistance 1 – Broken branches known, load unknown.	93
Figure 69: Confusion matrix for angle, resistance 2 – Broken branches known, load unknown.	93
Figure 70: Confusion matrix for angle, resistance 1 – Broken branches unknown, load known.	94
Figure 71: Confusion matrix for angle, resistance 2 – Broken branches unknown, load known.	94
Figure 72: Confusion matrix for angle, resistance 1 – Broken branches unknown, load unknown. .	95
Figure 73: Confusion matrix for angle, resistance 2 – Broken branches unknown, load unknown. .	95
Figure 74: Confusion matrix for broken branches, resistance 1 – Angle known, load known.....	96
Figure 75: Confusion matrix for broken branches, resistance 2 – Angle known, load known.....	96
Figure 76: Confusion matrix for broken branches, resistance 1 – Angle known, load unknown.....	97
Figure 77: Confusion matrix for broken branches, resistance 2 – Angle known, load unknown.....	97
Figure 78: Confusion matrix for broken branches, resistance 1 – Angle unknown, load known.....	98
Figure 79: Confusion matrix for broken branches, resistance 2 – Angle unknown, load known.....	98
Figure 80: Confusion matrix for broken branches, resistance 1 – Angle unknown, load unknown..	99
Figure 81: Confusion matrix for broken branches, resistance 2 – Angle unknown, load unknown..	99
Figure 82: Confusion matrix for load, resistance 1 – Angle known, broken branches known.....	100
Figure 83: Confusion matrix for load, resistance 2 – Angle known, broken branches known.....	100
Figure 84: Confusion matrix for load, resistance 1 – Angle known, broken branches unknown....	101
Figure 85: Confusion matrix for load, resistance 2 – Angle known, broken branches unknown....	101
Figure 86: Confusion matrix for load, resistance 1 – Angle unknown, broken branches known....	102
Figure 87: Confusion matrix for load, resistance 2 – Angle unknown, broken branches known....	102
Figure 88: Confusion matrix for load, resistance 1 – Angle unknown, broken branches unknown.	103

Figure 89: Confusion matrix for load, resistance 2 – Angle unknown, broken branches unknown.	103
Figure 90: Angle classification without knowing the broken branches.	104
Figure 91: Combination of four PDFs with the same ID.	111
Figure 92: Combination of PDFs with the same number of broken branches.	112
Figure 93: Combination of two PDFs with different number of broken branches.	113
Figure 94: Gaussian distribution and test for a family.	114
Figure 95: Gaussian distribution and test for a family – 3D plot.	115
Figure 96: Classification example using the third approach.	116
Figure 97: Confusion matrix for angle, resistance 1.	117
Figure 98: Confusion matrix for angle, resistance 2.	117
Figure 99: Confusion matrix for broken branches, resistance 1.	118
Figure 100: Confusion matrix for broken branches, resistance 2.	118
Figure 101: Confusion matrix for load, resistance 1.	119
Figure 102: Confusion matrix for load, resistance 2.	119
Figure 103: Graphical comparison between the first three approaches.	123
Figure 104: Sensor scheme.	125
Figure 105: Electrical scheme of the sensor.	128
Figure 106: Comparison between ideal model and experimental tests.	129
Figure 107: Hysteresis evidence in time domain.	130
Figure 108: Confusion matrix for angle.	138
Figure 109: Confusion matrix for broken branches.	138
Figure 110: Confusion matrix for load.	139

Table Index

Table 1: Fili Technical Data Sheet, table A.	41
Table 2: Fili Technical Data Sheet, table B.	41
Table 3: Process parameters.....	42
Table 4: Training table structure.	58
Table 5: Test table structure.....	58
Table 6: Accuracy of the first approach.....	77
Table 7: Accuracy of the second approach compared with the first one.	105
Table 8: Accuracy of the third approach compared with the others.	120
Table 9: Accuray values of all the approaches.	136
Table 10: Results of an example using the final tool.....	136

List of Used Abbreviations

<i>Abbreviation</i>	<i>Definition</i>
<i>AI</i>	Artificial Intelligence
<i>AM</i>	Additive Manufacturing
<i>CNT</i>	Carbon Nanotube
<i>CP</i>	Crack Propagation
<i>CPC</i>	Conductive Polymer Composite
<i>DL</i>	Deep Learning
<i>EG</i>	Expanded Graphite
<i>eVTOL</i>	Electric Vertical Take-Off and Landing
<i>FDM</i>	Fused Deposition Modeling
<i>GF</i>	Gauge Factor
<i>GNP</i>	Graphene Nanoplatelet
<i>HDPE</i>	High-Density Polyethylene
<i>HS</i>	Hyper-Sphere
<i>IoT</i>	Internet of Things
<i>ML</i>	Machine Learning
<i>MM</i>	Multiple Model
<i>PLA</i>	Polylactic Acid
<i>PMMA</i>	Poly(Methyl Methacrylate)
<i>SHM</i>	Structural Health Monitoring
<i>SSC</i>	Single Segment Characterization
<i>TDS</i>	Technical Data Sheet
<i>TPU</i>	Thermoplastic Polyurethane

1 Introduction

1.1 Background

In recent decades, the research field regarding Structural Health Monitoring (SHM) has grown significantly and is undergoing a major innovation. Traditional SHM techniques predominantly rely on external sensors that adhere to the surface of external components, such as electrical resistance strain gauges or piezoelectric transducers. However, this approach presents several challenges when applied to aircrafts:

- the mass of the sensor, although small, adds dead weight to the aircraft;
- the required wiring introduces structural complexity, further increasing the aircraft weight;
- external sensors are directly exposed to the environment and suffer the effects of humidity, temperature fluctuations, potential impacts, and so on. This poses a risk of not only altering data acquisition but also causing the detachment of the adhesive interface.

To overcome these limitations, the new frontier of research is the high integration. It consists of embedding the sensors seamlessly within the components themselves. Consequently, the component and the sensor are no longer distinct elements but constitute a single entity: a sensorized (or smart) component. In this manner, the mass added by the sensor is significantly reduced, a portion of the cabling is no longer necessary, and the sensor is no longer directly exposed to the environment.

Furthermore, high integration does not merely represent a solution to these issues, but it also provides an additional advantage: the proximity to the measurement point. Surface-mounted sensors cannot detect internal stress states or damage propagation within the element. In contrast, when the sensor is located inside the component, it enables a distance from the measurement point that can even be null.

Another important advantage of a highly integrated sensorized component is that its health status can be monitored much more effectively, thereby avoiding the replacement of parts that are still usable and consequently improving sustainability.

One of the most used techniques to achieve this is Additive Manufacturing (AM). By using different materials, the component and the sensor can be produced simultaneously during the same process, using a distinct material for each.

The material constituting the sensor is a critical aspect of the process. Recent innovations have led to multifunctional materials. These are materials that simultaneously perform two functions:

- a structural function, bearing a portion of the loads affecting the component;
- a conductive function, acting as transducers.

The type of material most frequently used for this purpose is the Conductive Polymer Composite (CPC). It is composed of a polymeric matrix that performs the structural function, and conductive inclusions that perform the conductive function.

Since CPC filaments suitable for Fused Deposition Modeling (FDM) are now commercially available, this technique is confirmed to be particularly suitable to manufacture sensorized components. Moreover, it is highly versatile and allows for the optimization of both the geometry of the structural component and the circuit constituting the internal sensor [1].

However, FDM is a process with lower repeatability compared to other more industrialized methods, and it is subject to several unpredictable phenomena, such as the anisotropy, the presence of localized voids, and the random distribution of conductive inclusions within the polymeric matrix [2].

Furthermore, it must be considered that each material is affected differently by these phenomena. Therefore, to really understand its behaviour, in-depth studies and experimental testing are required. This investigation was conducted on the CPC material named "Fili", which was used to manufacture our specimens [3].

This material was selected because of its electrical properties. The same authors of study [3] had previously observed that materials like Fili (a graphite-filled TPU-based CPC) “exhibit a strong linear piezoresistive response, while carbon-black-filled CPCs show lower strain sensitivity” [4].

As we can see, the carbon-based fillers are widely used for this kind of applications. The study [5] offers an overview about this kind of materials and their use in FDM: it highlights that particular attention must be paid to the process because it changes the physical properties of the material; in particular, temperature plays a fundamental role. The aforementioned study [3] also focused on this aspect. For further information, see the state of the art at paragraph 2.1.1.

1.2 Objective

Now that we have defined the research field in which we operate and all its challenges, it is possible to proceed with the presentation of the thesis objective.

The Smart Structures and Systems Laboratory of Politecnico di Torino is currently researching in the field of CPCs and high-integration SHM. One of the ongoing projects is the design of a sensorized landing gear for an eVTOL (electric Vertical Take-Off and Landing). To manufacture it, the intention is to exploit FDM technology using a CPC material. Naturally, to design such a component, it is necessary to first manufacture and characterize prototypes. Therefore, the characterization of the behaviour and the properties of the sensorized prototype of the landing gear constitutes the primary objective of this thesis.

Since the design of new prototypes occurs concurrently with the data analysis obtained from the initial prototypes, the characterization does not consist only in studying the behaviour of a definitive component: it has evolved into a mutual partnership with the laboratory staff for the progressive improvement of the various manufactured prototypes.

Consequently, the objective of the thesis is divided into two main aspects:

- the design of the mechanical component, particularly the electrical circuit constituting the sensor;
- the characterization of the sensor, studying what information can be extracted from it and which experimental tests are required to obtain it.

1.3 Why Machine Learning

The sensor must be characterized, and to do so, it was decided to exploit Machine Learning algorithms instead of more traditional methods. This decision is driven by two reasons:

- at the moment, the studied component is a prototype. An overly refined characterization of the sensor would not be applicable to the definitive component and would be useless. This is the risk encountered when using fitting curves or excessively complex ideal models. Machine Learning algorithms, conversely, treat the component as a black box while maintaining a good

degree of generality. Therefore, if trained on proper data, they are applicable to other components as well, including the definitive one;

- the phenomena mentioned at the end of paragraph 1.1 are unpredictable and difficult to model. Once again, treating the problem as a black box provides an effective solution to take account of them without requiring an exhaustive study of their physics.

Since an ideal model of the circuit constituting the sensor is easy to implement, it was built; however, as anticipated, the Machine Learning approaches demonstrated higher performances.

References

The introduction references are reported here to facilitate consultation.

1. Stano, G., Pavone, A., Jafor, M. A., Matalgah, K., Percoco, G., & Fleck, T. J. (2024). *Enhancing the sensitivity of 3D printed sensors via ironing and void reduction*. Virtual and Physical Prototyping, 19(1), e2331153.
2. Hassan, M. S., Zaman, S., Dantzler, J. Z., Leyva, D. H., Mahmud, M. S., Ramirez, J. M., & Lin, Y. (2023). *3D printed integrated sensors: From fabrication to applications – A review*. Nanomaterials, 13(24), 3148.
3. Ursi, F., & De Pasquale, G. (2025). *Comparative characterization of FDM structures with electrically-conductive sensing elements under static, dynamic and thermal loads*. Nature Scientific Reports, 15, 26877.
4. De Pasquale, G., & Ursi, F. (2025). *Characterization of electrically conductive polymers integrated in additively manufactured parts for sensing*. 11th ECCOMAS Thematic Conference on Smart Structures and Materials (SMART 2025), Linz, Austria.
5. Wei Q, Yang R, Zhao X, Zhou J, An Y, Yang S. (2023). *Fused deposition modeling of carbon-reinforced polymer matrix composites: A comprehensive review*. Polym Compos., 44(9), 5313-5345.

2 State of the Art

2.1 Applications of multifunctional materials

As described in the previous chapter, multifunctional materials represent a frontier of research in the field of mechanical engineering. Their name derives from the ability to perform multiple functions, but our research focuses on those that perform the load-bearing function (thanks to the mechanical properties of the insulating matrix) and the self-sensing one (thanks to the piezoresistivity generated by the conductive material).

2.1.1 Materials and Manufacturing Methods

Carbon-based inclusions

Part of the research naturally focuses on the raw material, and thus on composite materials. In particular, several studies focus on the conductive particles. For example, Carbon Nanotubes (CNTs) and Graphene Nanoplatelets (GNPs) are used. This is because carbon, in certain forms, effectively presents both properties required by this type of multifunctional materials: it is particularly strong and has excellent conductivity.

A study in particular [1] investigated the volumetric fraction of CNT inclusions necessary to create a continuous conductive network within a cement matrix. Then, the same study seeks solutions to avoid CNT agglomeration due to Van der Waals forces. Solutions such as the use of surfactants and ultrasound are presented to obtain a homogeneous dispersion of the particles. Since our work falls within the scope of aircraft rather than buildings, this study is of our interest not so much for the composite obtained, but because it highlights one of the main problems of this type of materials: the unpredictability given by the random distribution of conductive inclusions within the matrix. This problem is common to all multifunctional materials of our type, especially when Fused Deposition Modeling technique is used.

Regarding GNPs, there are numerous studies seeing their application for strain gauges, achieving considerable performance both in terms of Gauge Factor (paragraph 2.1.2) and measured strain range. Cataldi et al. in 2018 wrote a review about this topic, also evaluating the effects that the graphene structure has on the performance [2].

The article also refers to an application example for SHM in the aeronautical field [19]. This study is particularly interesting because:

- it belongs to the same field of application as this thesis;
- it mentions the hysteresis phenomenon, a problem identified in this work and which does not occur with GNPs;
- it highlights the importance of the reversibility of the piezoresistive phenomenon, which makes damage and strain detection possible. This is a crucial point for this type of material: it will be shown in the present work how hysteresis and plasticity can actually be a problem for the characterization of the sensor, particularly for its modelling.

However, unlike this thesis, the authors use the multifunctional material to create a component that is sensorized throughout its entire volume. This is because the multifunctional material is used to entirely manufacture the specimen rather than a strain gauge within itself. Naturally, the advantage of monitoring the part throughout its entire volume is evident, but this can only be done if the non-conductive matrix (in this case made of epoxy resin) possesses mechanical properties suitable for the loads to be supported. Otherwise, as in our work, most of the component is composed of a material with high mechanical performance (such as steel) and only its most stressed part is sensorized using a strain gauge. To do this, techniques such as Additive Manufacturing, which will be discussed later, are necessary.

There are also various other types of inclusions, based on the different forms of carbon: expanded graphite, carbon black, etc. The book chapter cited in source [4] deeply illustrates the percolation threshold (paragraph 2.1.2) and various chemical and physical processes to increase the material's conductivity. Composites with Expanded Graphite (EG) and Graphene Nanoplatelets (GNPs) are compared. Among the main results, it concludes that “EG reinforced composites that preserve the porous structure of EG showed a higher electrical conductivity and poorer mechanical properties than GNP reinforced nanocomposites. The percolation threshold was much lower for GNP nanocomposites than EG composites”. Furthermore, “Chemical functionalization of GNP improves the electrical and mechanical properties of the nanocomposites due to the enhanced interfacial adhesion between GNP and polymer”.

Figure 1 visually shows some results of the study [4] to better understand the meaning of the percolation threshold. The graph contains the experimental conductivity values (σ) of the studied composite as the filler percentage (f_v) within the matrix varies.

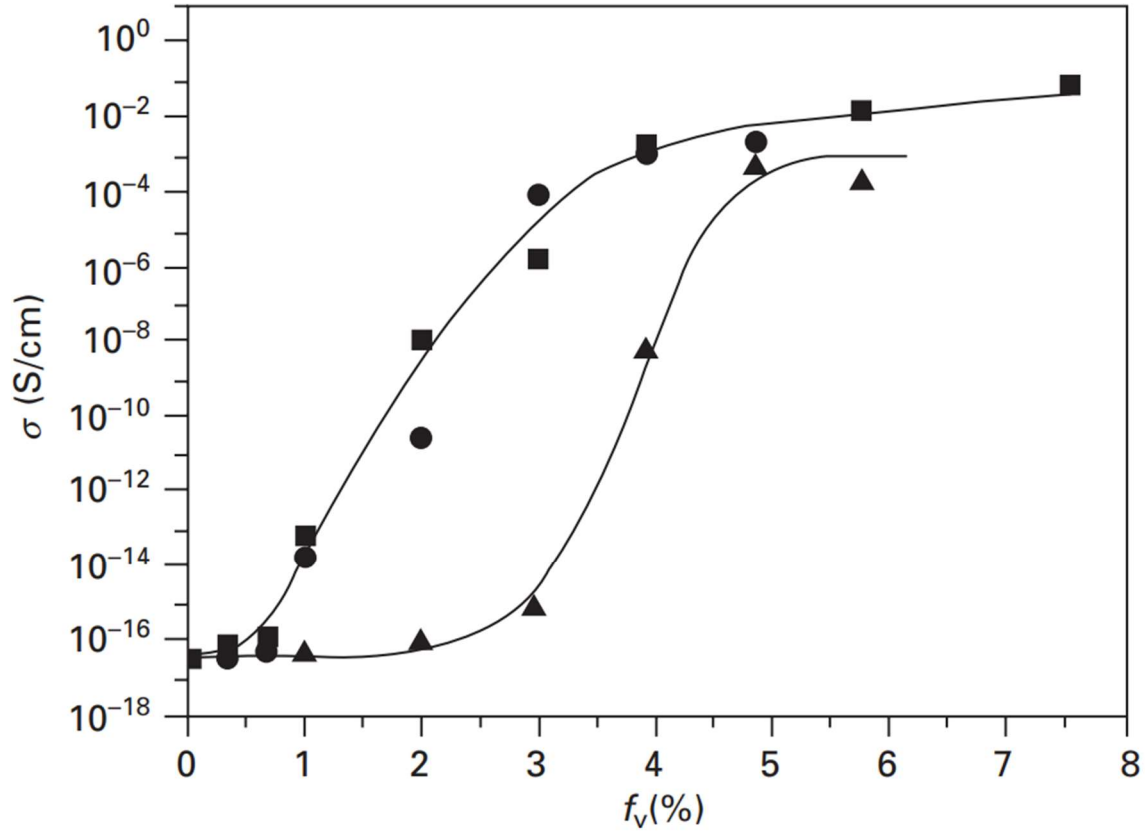


Figure 1: Percolation threshold.

Context of the figure: Electrical conductivity (σ) versus volume fraction of expanded graphite (f_v) for Expanded Graphite/Polypropylene composites prepared by two methods: melt blending (▲); solution intercalation (■). Repeated experiment of solution intercalation (●) [4, citing 7].

It is confirmed, therefore, that the main parameter to keep under control during the manufacturing phase (or to manipulate through chemical and physical processes) is the percentage of conductive particles within the matrix, but their size and shape must also be particularly taken into consideration.

Conductive Polymer Composites

Among the multifunctional materials that perform load-bearing and conductive functions is the category of Conductive Polymer Composites (CPCs). As mentioned in the previous paragraphs, the component designed in this thesis is natively sensorized through a strain gauge made of CPC (with graphite inclusions). Consequently, an in-depth analysis of this type of material is presented below. In particular, study [5] provides an excellent overview of the matrices and fillers currently used, as well as several present and future applications.

High-performance CPCs represent the material basis for organic electronics. “These not only include their conventional device uses in antistatic coatings, microwave absorbers, transparent electrodes, organic solar cells, organic field effect transistors, organic electroluminescence devices, electrochromic devices, thermoelectric devices, supercapacitors, sensors, and so on, but also bring great opportunity in emerging fields like flexible electronics, wearable electronics, biomedical electronics, etc.”

The regulation of mechanical properties is a widely open research field. It varies basing on the morphology of the material and the specific requirements. Here are some examples:

- hydrogels (bioelectronics, sensors): research focuses on increasing tensile strength and elongation. The same applies to fibers (wearable electronics);
- aerogels (interference shielding, wearable devices): research focuses on reduced density, high porosity, and compressibility;
- films (most common morphology): research focuses on high strength, surface hardness, and strong adhesion to the substrate;
- elastomers (strain sensors): research focuses on increasing tensile strength, elongation, and fracture energy.

The polymers mostly used as matrices are poly(vinyl alcohol) (PVA) and polydimethylsiloxane (PDMS). The review identifies Intrinsically Conductive Polymers (ICP) and MXene (which has a graphene-like 2D structure) as the most promising conductive fillers. It is interesting to note that using ICPs as fillers would result in a fully polymeric compound.

Regarding the manufacturing process, it depends on the type of material to be obtained. For example, hydrogel is mainly built through physical/chemical crosslinking and molding methods, films are processed through drop-coating and spin-coating. Scalability also influences the required process. For example, in-situ polymerization and freeze-drying are suitable for large-scale production but have poor repeatability, while additive printing and screen printing are highly precise but poorly scalable industrially.

The same article concludes by highlighting the current challenges in this research field: “the synergy of electrical related functionality and mechanical properties will become more and more important to promote the environmental stability, operational reliability, and multi-functional integration capability of CPCs in growing practical applications.”

2.1.2 Analytical and Numerical Models

The fundamental quantity for evaluating a strain gauge as a whole is the Gauge Factor (GF). It is defined by the formula:

$$GF = \frac{\Delta R/R_0}{\varepsilon}$$

Where:

- ΔR is the resistance difference recorded for a certain strain;
- R_0 is the initial resistance value;
- ε is the material strain.

However, it derives from the structure of the material and the phenomena involving it. Therefore, the GF can only be an evaluation parameter, but research must be much more in-depth.

The authors of the work [3] studied the influence of the amount of conductive particles on the measured resistance and the phenomena generating a resistance difference: the variation of conductive paths caused by the tunneling effect and the contact between particles. In particular, they highlighted that “flexural strain makes possible the creation of new electrically conductive paths

meaning an apparent lower sensitivity while strain produced in a tensile test provokes the breakage of the tunnel or contact based conductive paths”.

Therefore, the models in the literature focus on the microscopic scale, evaluating the relative distance of the conductive particles as mechanical stress varies, also considering phenomena such as quantum tunneling [6]. Study [7] proposes a subdivision of the research field to explain the electrical conduction mechanism:

- how to form conductive paths: theories are cited such as “the percolation theory, the effective medium theory, the microstructure theory, and the thermodynamic theory”;
- how conduction occurs after the formation of the paths: theories are cited such as “contact conducting, the tunneling effect, dielectric breakdown, and field emission”.

Actually, all the theories cited for the microscopic scale study, in particular those relating to conductive paths, date back several decades. This shows how composite materials of this kind have been studied for a long time: recent manufacturing techniques allow for the production of components, for the study of which the results of such research are taken again.

Percolation theory

The percolation theory refers to the homonymous phenomenon. Within a composite, the percolation threshold is the minimum amount of conductive particles to be inserted within the insulating matrix to create a continuous interconnected network (i.e., the conductive path). When this network is formed, the material suddenly changes behaviour, transforming from an insulator to an electrical conductor. Study [8] investigated this phenomenon in a short carbon fiber-filled high-density polyethylene (HDPE)/poly(methyl methacrylate) (PMMA) polymer blend. The composite composed of an HDPE matrix (insulating) and carbon fiber particles (conductive) was mixed with PMMA (insulating). Therefore, we can talk about two percolation thresholds: a first one to make the HDPE conductive, a second one to make the mixture of PMMA and conductive HDPE conductive.

The study concludes that the amount of carbon fibers has an influence on the second percolation threshold. A carbon fiber content of 1.5 phr is sufficient to have a percolation threshold of 20% by weight of HDPE in PMMA. Instead, increasing the amount of conductive inclusions to 2.5 phr also raises the second percolation threshold to 30% by weight (see Figure 2). This result is counterintuitive: one would expect an increase in conductive particles to lead to a decrease in the

second percolation threshold. Actually, the quantity of inclusions modifies their shape and influences the creation of the connection network. This highlights the difficulty in deriving generally valid rules for these materials: each of them must be studied in depth to understand which phenomena come into play in the formation of conductive paths.

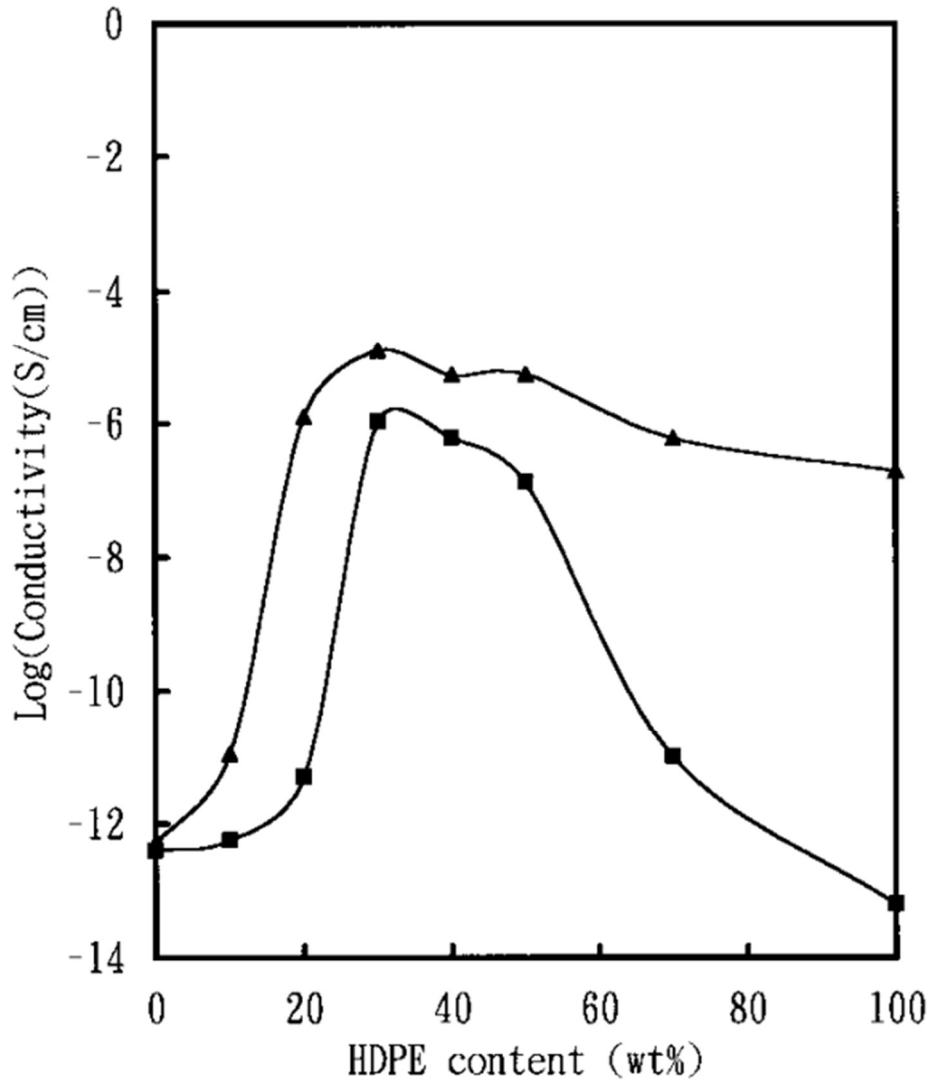


Figure 2: Dependence of the conductivity on the conductive filler content.

Context of the figure: Dependence of electrical conductivity on HDPE content for HDPE/PMMA blends filled with (▲) 2.5 and (■) 1.5 phr Vapor-Grown Carbon Fiber molded at 190°C for 15 min. [8].

The percolation equation at the basis of the homonymous theory describes the conductivity value of a composite material with an insulating matrix and conductive filler. It depends on:

- f'_c : the critical fraction, relative to bond, site, volume or area. It corresponds to the fraction of the conductive component at which the particles form the conductive path for the first time;
- f' : the fraction of the conductive component;
- A : a proportionality constant typical of the composite;
- t : a constant exponent typical of the composite.

Therefore, the equation to calculate the conductivity σ of the composite is:

$$\sigma = A(f' - f'_c)^t$$

It should strictly be valid only when f is close to f'_c , but often its range of validity covers the entire range $f'_c \leq f' \leq 1$ [9].

Effective medium theory

A study was conducted decades ago on modelling via the effective medium theory [9]. “McLachlan has introduced a more general effective-media (GEM) equation for the conductivity of binary media. The GEM equation [...] has the mathematical form of the percolation equation when the ratio of the conductivities of the two components is infinitely large. The GEM equation has, as does the percolation equation, two parameters, a critical volume (area) fraction f_c and an exponent t (in this case related to the shape of the grains of the components and f_c)”. Therefore, this study focuses on the formulation of equations that allow describing the conductivity of dual media; such modelling is also validated by experimental evidence. In particular, the General Effective-Media (GEM) equation is the following.

$$\frac{f(\Sigma_l - \Sigma_m)}{\Sigma_l + [f_c/(1 + f_c)]\Sigma_m} + \frac{(1 - f)(\Sigma_h - \Sigma_m)}{\Sigma_h + [f_c/(1 - f_c)]\Sigma_m} = 0$$

Where:

- f is the volume (or area) fraction of the insulating matrix;
- $f_c = 1 - f'_c$ (see f'_c in the previous formula);
- $\Sigma_i = \sigma_i^{1/t}$, with $i = l, h, m$;
- σ_l is the conductivity of the insulation matrix;
- σ_h is the conductivity of the conductive inclusions;
- $\sigma_m(f)$ is the conductivity of the medium itself;
- t is a constant exponent.

Microstructure theory

Regarding microstructure theory, work [10] represents an interesting alternative to percolation theory and effective medium theory (which is based on the first one). The abstract explains that “existing models generally can be divided into percolation theories and non-percolation theories. The basis of the percolation theory is the fact that the conductivity of the composite increases dramatically at a certain fiber concentration called the percolation threshold. [...] Non-percolation theories include terms, which account for microstructural data such as fiber orientation, length, and packing arrangement. A comparison of experimental data with predictions from the various models reveals that only the percolation theory is able to accurately model the conductive behavior of an actual composite”. Despite this, the study proposes two new alternative models, based on microstructures, to predict the volume resistivity of a composite. “The first model relates resistivity to the concentration and orientation of the fibers, while assuming perfect fiber-fiber contact. [...] The second model accounts for the effect of fiber-fiber contact and fiber length on composite resistivity”.

The attention of study [10] to the microstructure of the composite is supported by study [11], which states: “It will be shown here that statistical, geometric or thermodynamic models explain the conductivity behaviour of specific mixtures on the basis of insufficient assumptions. However, the conductivity seems to be predictable with the help of structure-oriented models.”

Fiber orientation and tunneling effect

In a previously cited article [6], the authors proposed a numerical model to study the piezoresistivity of a polymer nanocomposite with multi-walled carbon nanotube (MWNT) fillers. The nanotubes are assumed to be randomly distributed within the epoxy matrix. This model is interesting because it takes into account both effects allowing the passage of current: the orientation of the fibers (and therefore the contact between them) and the tunneling effect (when they are very close but contact is absent). Furthermore, it is important to note that the model's results agree with the experimental ones. Therefore, once the conductive network is formed the contributions of the individual resistances present within it can be evaluated:

- the resistance of a single CNT in contact with two other CNTs: “for a CNT with two contacting points i and j with neighboring CNTs, the conductance g_{ij} between i and j (the inverse of resistance R_{ij}) can be evaluated as

$$g_{ij} = \sigma_{CNT} * \frac{S_{CNT}}{l_{ij}}$$

where l_{ij} is the length between the points i and j & σ_{CNT} and S_{CNT} are electrical conductivity and cross-sectional area of the CNT, respectively” (see Figure 3 [6]);

- the resistance given by the tunneling effect involving two neighboring CNTs without contact: “the tunneling resistance between two neighboring CNTs can be approximately estimated as

$$R_{tunnel} = \frac{V}{AJ} = \frac{h^2 d}{A e^2 \sqrt{2m\lambda}} \exp\left(\frac{4\pi d}{h} \sqrt{2m\lambda}\right)$$

where J is the tunneling current density, V the electrical potential difference, e the quantum of electricity, m the mass of electron, h the Planck's constant, d the distance between CNTs, k the height of barrier (for epoxy, 0.5–2.5 eV), and A the cross-sectional area of tunnel (the cross-sectional area of the CNT is used here as an approximation)” (see Figure 4 [6]).

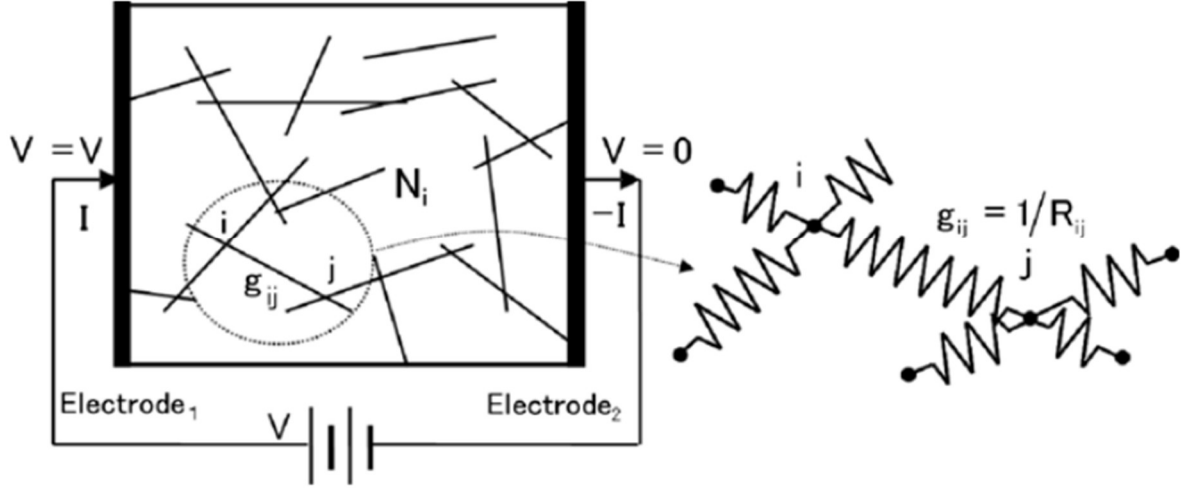


Figure 3: Schematic of a resistor model with a random distribution of CNTs.

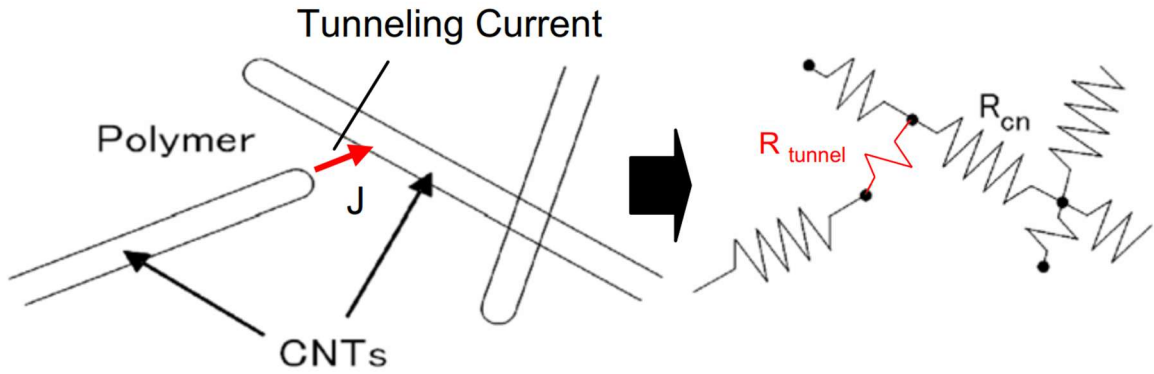


Figure 4: Modeling of tunneling resistance in the resistor network.

2.1.3 Experimental Tests and Characterization

The standard experimental campaigns for sensor characterization found in the literature generally involve cyclic tension/compression tests in the elastic range, to evaluate the linearity of the GF, the repeatability of the electrical signal and the absence of hysteresis.

An example of this can be seen in [12], which proposes a new three-dimensional percolation theory considering three factors and their dependence on strain: nonlinear elasticity, Poisson's ratio, and

percolation threshold. The phenomenon driving this theory is the strain-dependent filler reallocation. The two fundamental equations of the three-dimensional percolation theory are the following:

$$\sigma_{1 \text{ or } 2} = A(V_{f1 \text{ or } f2} - V_c)^t$$

$$V_{f1 \text{ or } f2} = \frac{V_{filler}}{V_{1 \text{ or } 2}}$$

Where:

- index *1 or 2* describes initial or deformed state, respectively;
- σ is the electrical conductivity of the composite material;
- A is a proportionality constant related to the electrical conductivity of conductive fillers;
- V_f is the volume fraction of conductive fillers;
- V_{filler} is the volume of conductive fillers;
- V_1 or V_2 is the volume of the conductive composites at initial and deformed state, respectively;
- V_c is the percolation threshold (constant) which indicates the volume fraction where a rapid increase of conductivity is observed;
- t is a constant fitting exponent.

The novelty of the new theory proposed by the article lies in not considering V_c constant but dependent on strain, through Poisson's ratio and nonlinear elasticity. Beyond the details regarding this research, it is interesting to note how experimental tests are performed to obtain graphs on a plane very similar to the one widely used in this thesis. On the horizontal axis, the engineering strain is reported instead of the absolute displacement, on the vertical axis, the conductivity is reported instead of the overall resistance. But, ultimately, the concept of a variable conductivity based on strain is what is shown in these types of graphs. Figure 5 [12] shows an example. An analogy with what will be discussed in this thesis can be noted. In the experimental tests that we performed, the resistance with respect to

the strain presents a minimum; similarly, in this article the conductivity, which inherently contains the reciprocal of the resistance, presents a maximum.

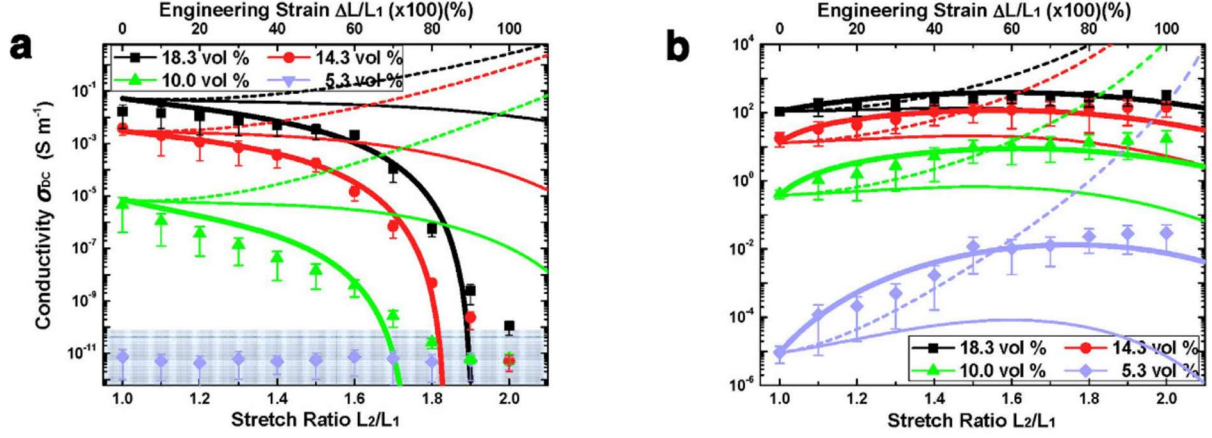


Figure 5: Predicted and measured DC-conductivity in terms of tensile strain.

Context of the figure: Predicted and measured DC-conductivity in terms of tensile strain for (a) Isotropically distributed Conductive Composites and (b) Anisotropically aligned Conductive Composites. Symbols indicate measured results. Dashed lines indicate the results of typical three-dimensional percolation theory (with constant Poisson's ratio and constant percolation threshold). Thin solid lines indicate the results of modified three-dimensional percolation theory (with strain-dependent Poisson's ratio and constant percolation threshold). Thick solid lines indicate the results of proposed three-dimensional percolation theory (with strain-dependent Poisson's ratio and strain-dependent percolation threshold) [12].

The authors of the article [13] performed tests of the same type to show the reversibility of the composite used with graphene filler. Unlike in the previous article [12], here the tension/compression cycles are repeated not only to obtain a larger data population, but also to show the absence of hysteresis within a certain number of tests. The graphs obtained are shown in Figure 6 and Figure 7 [13].

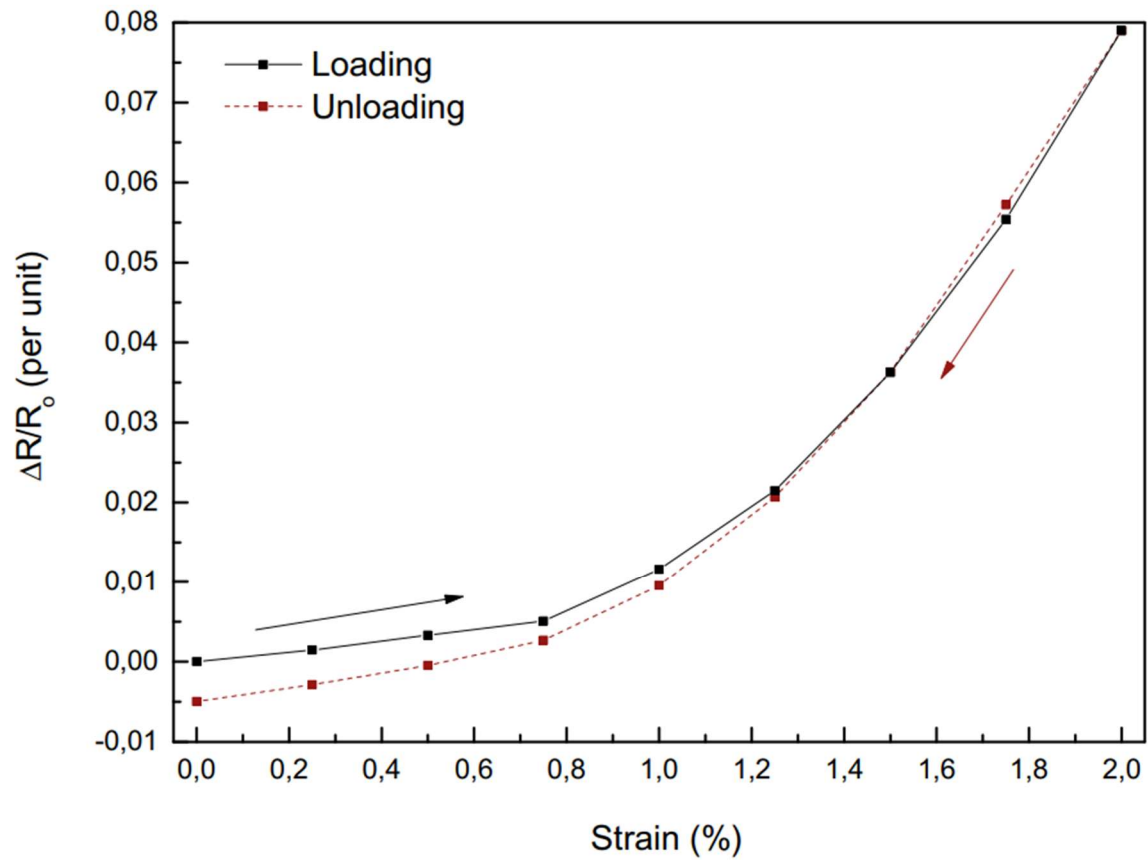


Figure 6: Electrical response during loading and unloading.

Context of the figure: Electrical response of Graphene Nanoplatelets/epoxy nanocomposites during loading and unloading under flexural test [13].

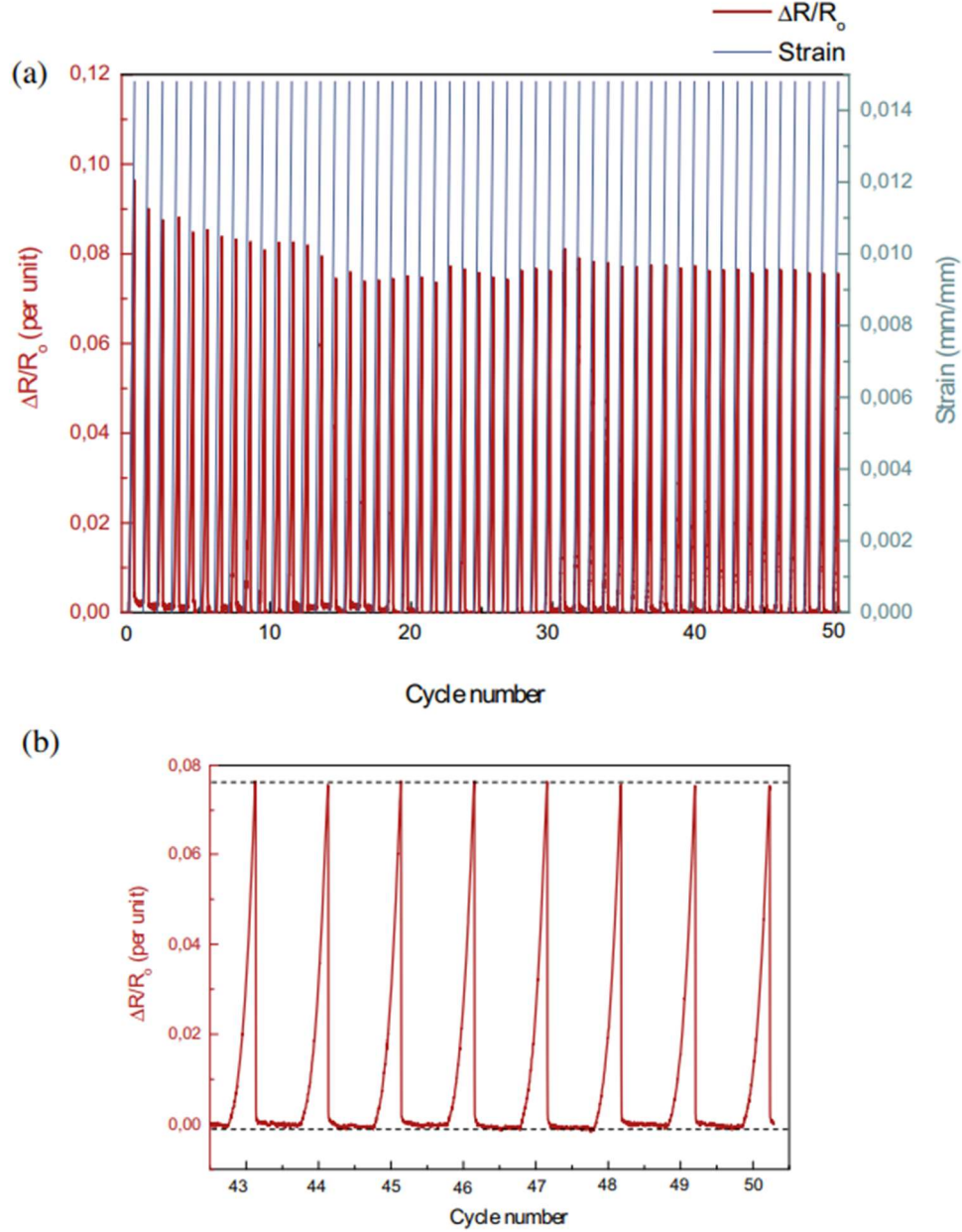


Figure 7: Reversibility of the electrical response.

Context of the figure: Reversibility of the electrical response during (a) 50 strain cycles under flexural test and (b) detail of the last 8 cycles [13].

The typology of cyclic tension/compression tests in the elastic range is the one used also in this thesis.

2.2 Machine Learning Methods Applied to SHM

The enormous amount of data (Big Data) generated by multifunctional materials in SHM systems requires processing tools capable of extracting significant features and automating the diagnosis. As can be seen in Figure 8 [14], which depicts the typical operating and information flow scheme of an SHM, Machine Learning algorithms are the most suitable tool for this purpose. In fact, they are particularly high performing in the four tasks contained in the lower semicircle of this figure: data processing and control, data management, structural health evaluation and decision making.

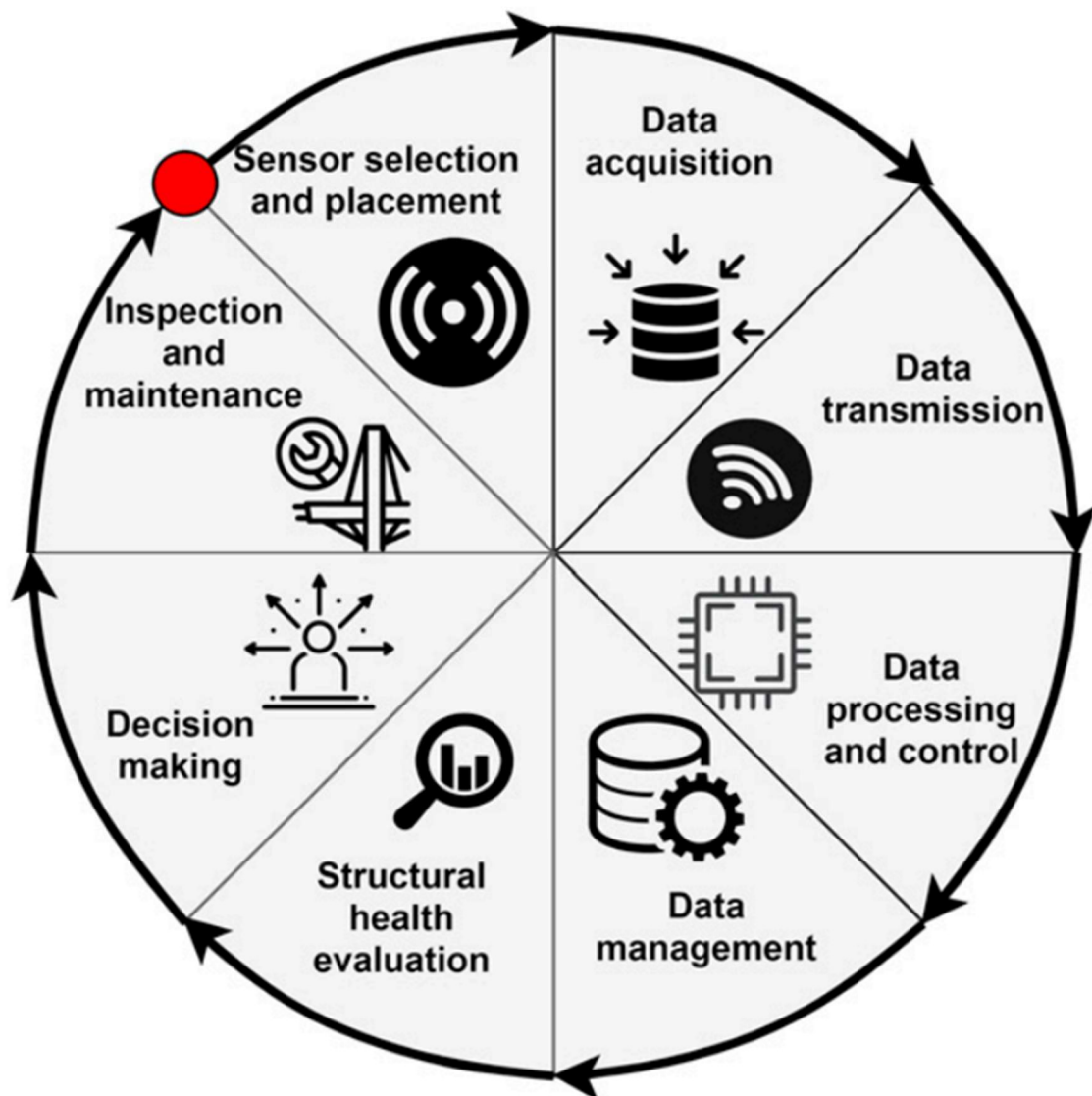


Figure 8: Typical components of SHM.

ML algorithms are usually divided into three main families:

- supervised algorithms;
- unsupervised algorithms;
- Deep Learning (DL) algorithms.

Malekloo et al. in 2022 [14] wrote an in-depth review that provides an excellent overview of the current state of the art. The authors also propose a hybrid category (semi-supervised algorithms), but it is not particularly detailed: these methods combine techniques of supervised and unsupervised algorithms together when there is a partial lack of information. A scheme of the ML algorithms exploited in the context of SHM could be the one proposed in Figure 9 [14].

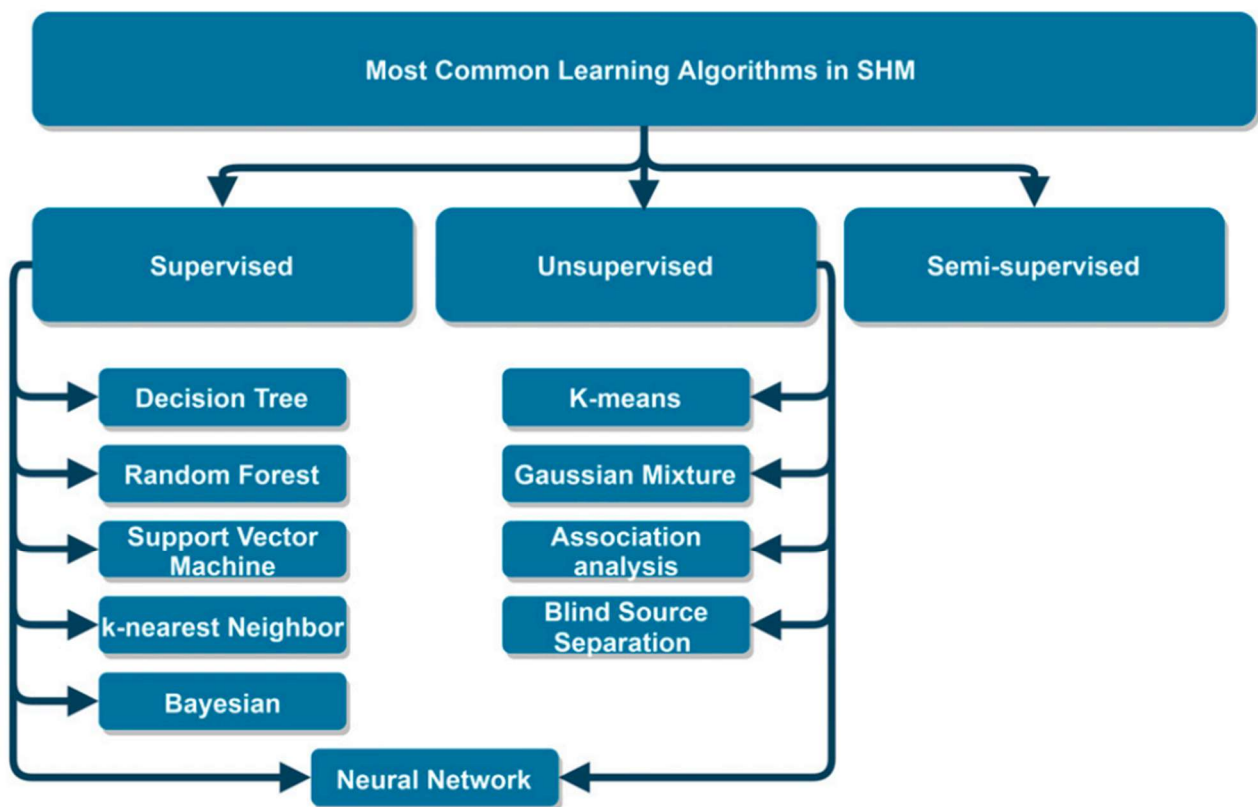


Figure 9: Three common learning algorithms in SHM.

2.2.1 Supervised Algorithms

Supervised algorithms are widely used and deal with the classification of variables. During the training phase, the algorithm learns to recognize the label to be associated with each record (classification), thanks to a dataset containing labelled data. Alternatively, it can seek a regression function to divide the data into categories (again, these must be already labelled). Algorithms such as Decision Tree, Random Forest, and K-Nearest Neighbors belong to this family. They are generally explainable, meaning that it is possible to track how the results are obtained results. They adapt very well to all discrete classifications in the engineering field, such as the distinction between defective and non-defective components. The approaches used in this thesis are supervised classification algorithms.

2.2.2 Unsupervised Algorithms

Unsupervised algorithms work with unlabelled data, searching for patterns and anomalies. Algorithms such as Principal Component Analysis and K-Means Clustering belong to this family. They generally require a larger amount of data compared to supervised ones, and they are less explainable. They are particularly suitable for finding defects and anomalies within parts or a set of parts.

2.2.3 Deep Learning Algorithms

Deep Learning algorithms work on massive amounts of data and are poorly explainable. The most famous of these is the Convolutional Neural Network. They can process highly complex data and perform both classification and pattern recognition tasks. They have great performances, but sometimes their low explainability is an obstacle or it is not possible to collect enough data to implement them.

In general, ML algorithms are employed for their advantages, attempting to mitigate their disadvantages. On one hand, they are capable of processing a large amount of data even in the presence of background noise; on the other hand, it is necessary that the collected data are of high quality. On one hand, they allow extracting information from components with complex geometries that are difficult to model; on the other hand, their low explainability could lead to incorrect conclusions without knowing what caused them. Some studies have investigated ways to overcome this problem by forcing neural networks to use constitutive equations of continuum mechanics, obtaining the Physics-Informed Neural Networks [15].

It is very important to highlight the significance of using Machine Learning in this study. Indeed, the material used is suitable for FDM, but its filler material is graphite rather than graphene or other materials more suitable for characterization. Therefore, this material has all the inherent advantages of FDM (in particular the high versatility and the possibility of creating complex geometries) but it presents irreversible phenomena that make it less suitable for characterization, compared to materials based, for example, on graphene [3]. Thus, the challenge of this thesis lies precisely in the fact that the sensor to be characterized is composed of a CPC material that presents irreversible phenomena such as plasticization and hysteresis. This is where ML methods come into play: as will be discussed in subsequent chapters, they allow to extract information from experimental datasets even when the physics of the phenomenon is not completely known. Thus, through this work and other similar research projects, it will be possible to increasingly better characterize self-sensing components even when made with innovative and still little studied materials.

2.2.4 Application Examples

In this thesis, ML is applied for the SHM of a drone. First of all, we have to pay attention to an ambiguity that arises when talking about SHM applied to drones. We can refer to the technologies inherent to drones with the term UAV: Unmanned Aerial Vehicle. Malekloo et al. in their article [14] wrote a chapter on “UAV-assisted SHM”, but this is a slightly different field from the application of this thesis. “UAV-assisted SHM” refers to the use of drones to monitor the state of health of other objects, particularly bridges, since drones can easily reach locations that are difficult to access. Therefore, the state of health of the drone is not monitored, but monitoring is done through the drone itself [16].

In the research described in articles [17] and [18], ML algorithms are applied to a case very similar to the one of this thesis. “The development and experimental validation of a Machine Learning (ML) vibration based Structural Health Monitoring (SHM) prototype system for composite aerostructures operating under uncertainty is presented.”

However, unlike this thesis, here SHM is based not only on strains (detected via strain gauges) but also on vibrations (detected via accelerometers). Furthermore, the strain gauges are made with a composite of carbon fibers in epoxy resin (a material very similar to the one used in our project), but they are bonded on the surface and not manufactured internally. Figure 10 shows the experimental setup [17].

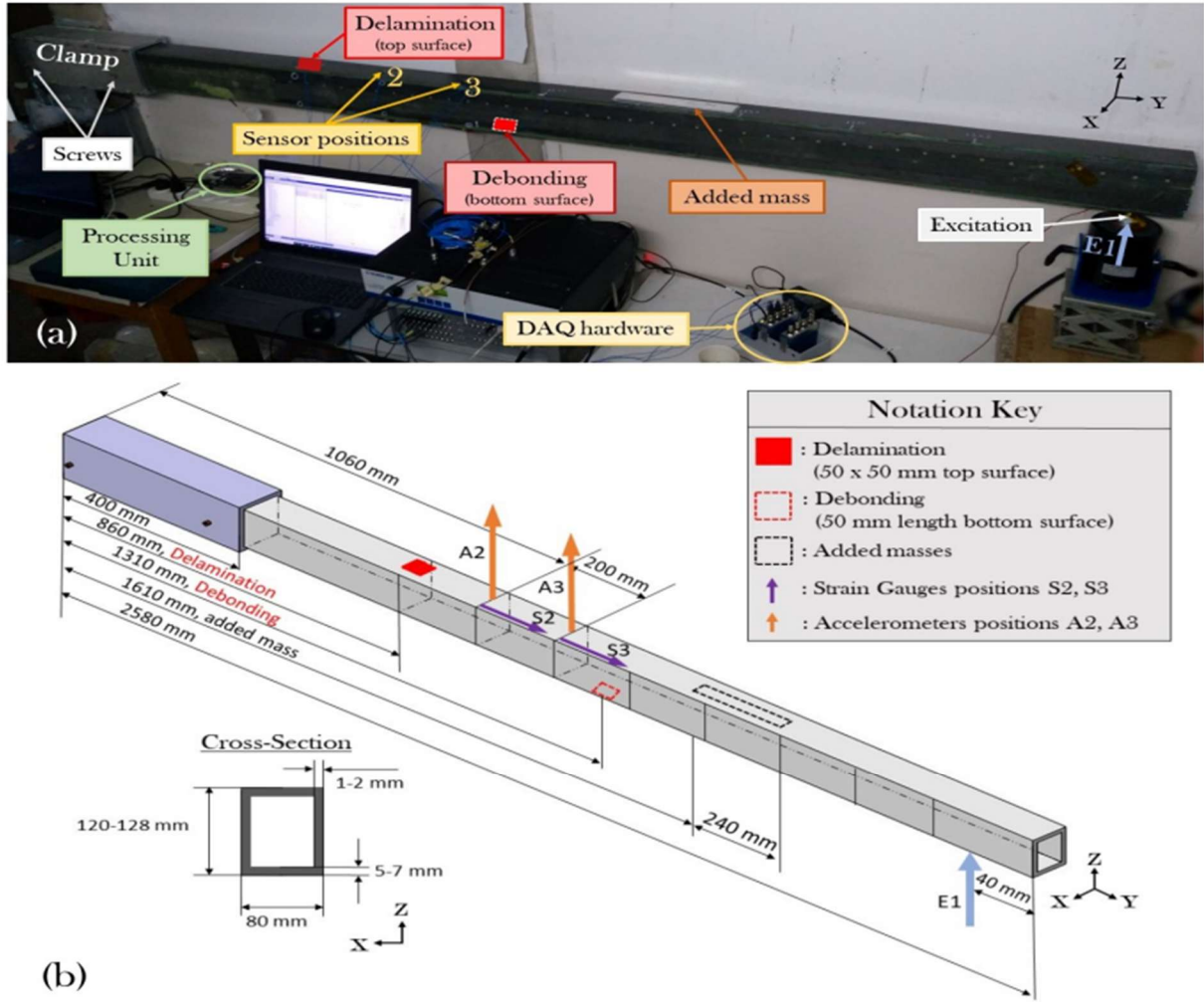


Figure 10: Experimental setup of the ML application example.

Context of the figure: Experimental setup: (a) Photo, (b) schematic representation of the aerostructure and the clamping at the left end [17].

Different ML methods and different models are adopted to create a tool capable of performing damage detection. To validate both the ML algorithms and the models, numerous experimental tests were performed, introducing three uncertainty factors through the variation of excitation force, boundary conditions, and mass.

In the training phase, it is possible to choose which methods to adopt. In particular, the choice of the Healthy Subspace representation has a strong influence: a Multiple Model (MM) or a varying radii Hyper-Sphere (HS) type. The process is represented in Figure 11.

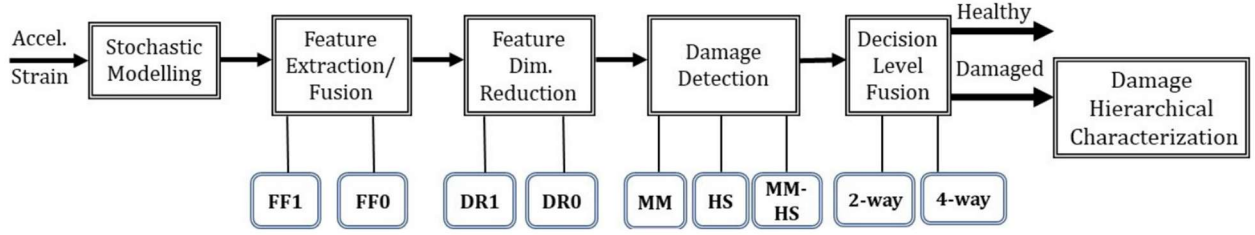


Figure 11: Flowchart of the ML application example.

Context of the figure: Flowchart of the SHM methodology including its Baseline and Inspection Phases and the available options. Baseline Phase: FF1 if feature fusion is selected (else FF0); DR1 if feature dimensionality reduction is selected (else DR0); Diagnostic algorithm selection as MM, or HS, or both MM-HS. Inspection Phase: Selection of Decision Level Fusion as 2-way using the FF1 option (concatenated feature) and MM or HS, or 4-way using the FF0 option (two types of features) and both MM and HS methods. [18].

The experimental data are therefore represented through the MM and HS methods. Due to their nature, the representation provided by these methods is free of units of measurement but, even without knowing them, it is possible to intuit from the figure that the final evaluation to determine if the component is damaged is based on a threshold value (see Figure 12 [18]).

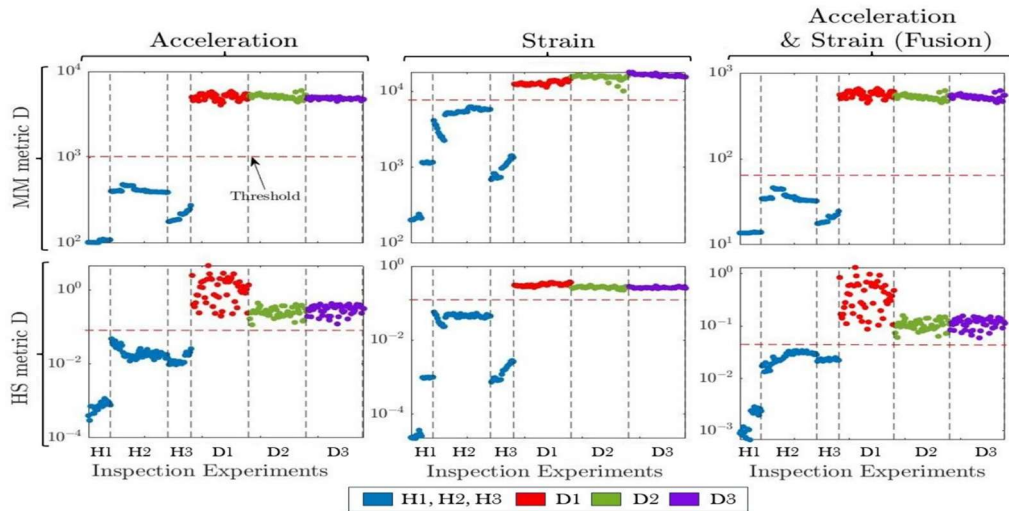


Figure 12: Results visualization in the ML application example.

Context of the figure: Damage detection results in terms of the distance metric D. Six diagnostic procedures are implemented: (a), (b), (c) MM based using acceleration, strain, and acceleration-strain signals; (d), (e), (f) HS based counterparts. Conditions: Healthy (H1), Healthy with smaller mass added (H2), Healthy with bigger mass added (H3), Damaged (D1), Damaged with smaller mass added (D2), Damaged with bigger mass added (D3). [17].

By making the different choices, excellent accuracy values are obtained in every case, collected in Figure 13.

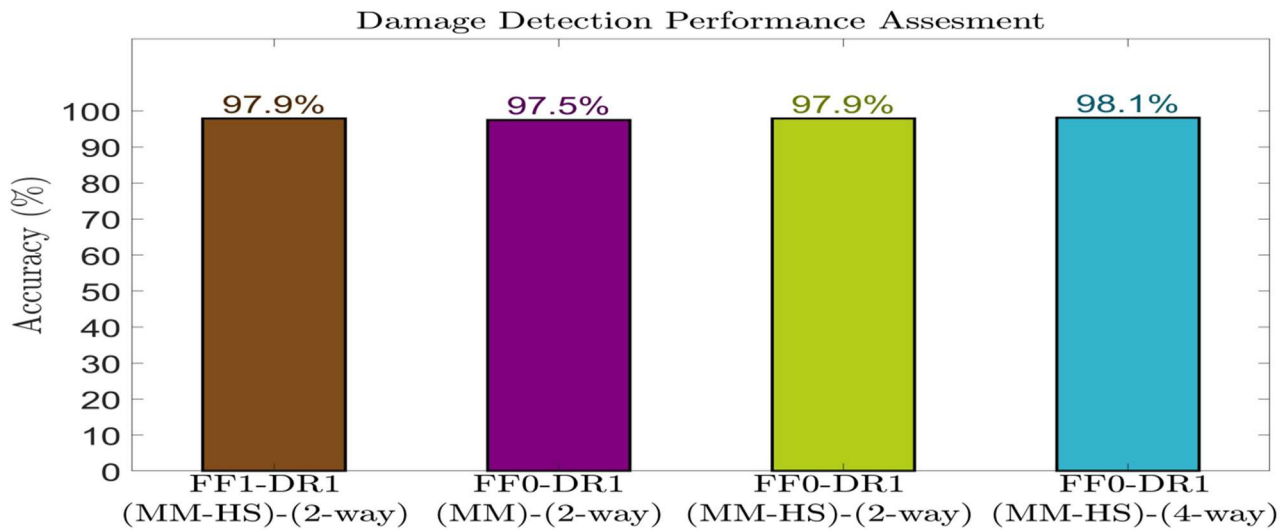


Figure 13: Performances of the ML application example.

Context of the figure: Damage detection results with respect to the selected SHM methodology options (see Figure 11). [18].

References

Since there are no further citations to articles in the next chapters, the state-of-the-art references are reported here to facilitate consultation.

1. D'Alessandro, A., Rallini, M., Ubertini, F., Materazzi, A. L., & Torre, L. (2016). *Scalable self-*

- sensing carbon nanotubes cement-matrix composites for applications in structural health monitoring*. Cement and Concrete Composites, 68, 23-34.
2. Cataldi, P., Athanassiou, A., & Bayer, I. S. (2018). *Graphene nanoplatelets-based advanced materials and recent progress in sustainable applications*. Applied Sciences, 8(9), 1438.
 3. Moriche, R., Sánchez, M., Jiménez-Suárez, A., Prolongo, S. G., & Ureña, A. (2016). *Strain monitoring mechanisms of sensors based on the addition of graphene nanoplatelets into an epoxy matrix*. Composites Science and Technology, 123, 65-70.
 4. Geng, Y., Li, J., & Kim, J. K. (2010). *Synthesis and electrical conducting behavior of graphite nanoplatelet/polymer nanocomposites*. In Physical Properties and Applications of Polymer Nanocomposites (pp. 315-346). Woodhead Publishing.
 5. Zhu, L., Chen, S., Zhou, M., An, S. Y., Liang, L. S., Shen, Y. L., & Xue, Z. X. (2024). *Regulation of mechanical properties of conductive polymer composites*. Chinese Journal of Polymer Science, 42(12), 1855-1880.
 6. Hu, N., Karube, Y., Yan, C., Yuan, Z., Yin, G., & Fukunaga, H. (2008). *Tunneling effect in a polymer/carbon nanotube nanocomposite strain sensor*. Acta Materialia, 56(13), 2929-2936.
 7. Shen J W, Chen X M, and Huang W Y (2003). *Structure and electrical properties of grafted polypropylene/graphite nanocomposites prepared by solution intercalation*. J Appl Polym Sci, 88, 1864–1869.
 8. Zhang, C., Yi, X.-S., Yui, H., Asai, S. and Sumita, M. (1998), *Morphology and electrical properties of short carbon fiber-filled polymer blends: High-density polyethylene/poly(methyl methacrylate)*. J. Appl. Polym. Sci., 69: 1813-1819.
 9. McLachlan, D. S. (1988). *Measurement and analysis of a model dual-conductivity medium using a generalised effective-medium theory*. Journal of Physics C: Solid State Physics, 21(8), 1521-1532.
 10. Weber, M., & Kamal, M. R. (1997). *Estimation of the volume resistivity of electrically conductive composites*. Polymer composites, 18(6), 711-725.
 11. Lux F (1993), *Models proposed to explain the electrical conductivity of mixtures made of conductive and insulating materials*. J Mater Sci, 28, 285–301.

12. Kim, S., Choi, S., Oh, E., Byun, J., Kim, H., Lee, B. & Hong, Y. (2016). *Revisit to three-dimensional percolation theory: Accurate analysis for highly stretchable conductive composite materials*. Scientific Reports (Nature Publisher Group), 6, 34632.
13. Moriche, R.; Sánchez, M.; Prolongo, S.G.; Jiménez-Suárez, A.; Ureña, A. *Reversible phenomena and failure localization in self-monitoring gnp/epoxy nanocomposites*. Compos. Struct. 2016, 136, 101–105.
14. Malekloo, A., Ozer, E., AlHamaydeh, M., & Girolami, M. (2022). *Machine learning and structural health monitoring overview with emerging technology and high-dimensional data source highlights*. Structural Health Monitoring, 21(4), 1906-1955.
15. Worden, K., & Dulieu-Barton, J. M. (2020). *Machine learning at the interface of structural health monitoring and non-destructive evaluation*. Philosophical Transactions of the Royal Society A, 378(2182), 20190581.
16. Perry BJ, Guo Y, Atadero R, et al. *Streamlined bridge inspection system utilizing unmanned aerial vehicles (UAVs) and machine learning*. Measurement 2020; 164: 108048.
17. Spiliotopoulos, P. E., Fera, F. T., Saramantas, I. E., Sakellariou, J. S., Fassois, S. D., Ofir, Y., & Spandonidis, C. (2024). *Development and experimental validation of a prototype system for machine learning based SHM in composite aerostructures*. Journal of Physics: Conference Series, 2692(1), 012025.
18. Saramantas, I. E., Spiliotopoulos, P. E., Fera, F. T., Bourdalos, D. M., Sakellariou, J. S., Fassois, S. D. & Spandonidis, C. (2024, February). *A vibration-based Machine Learning type Structural Health Monitoring methodology for populations of composite aerostructures under uncertainty*. In Journal of Physics: Conference Series (Vol. 2692, No. 1, p. 012023). IOP Publishing.
19. Usha Pawar, Shivaji G. Chavan, Kiran Suresh Bhole, Mansing Rathod, Dipali Bhole, Ankit D. Oza, Manoj Kumar, Manish Gupta, Satbir S. Sehgal & Manoj Kumar (2024). *Modeling and analysis of multi-functional self-healing material using Runge-Kutta Method for investigation of aircraft wing structure*. Advances in Materials and Processing Technologies, 10:4, 3724-3740.

3 Component and Experimental Tests

3.1 Component

As previously mentioned, the component under test is a specimen of an eVTOL aircraft landing gear.

3.1.1 Geometry

The specimen occupies an overall size of 140.80 x 100 x 79.33 mm. It consists of a square base, which represents a portion of the aircraft fuselage, and an outgoing curved beam connecting to the skid.

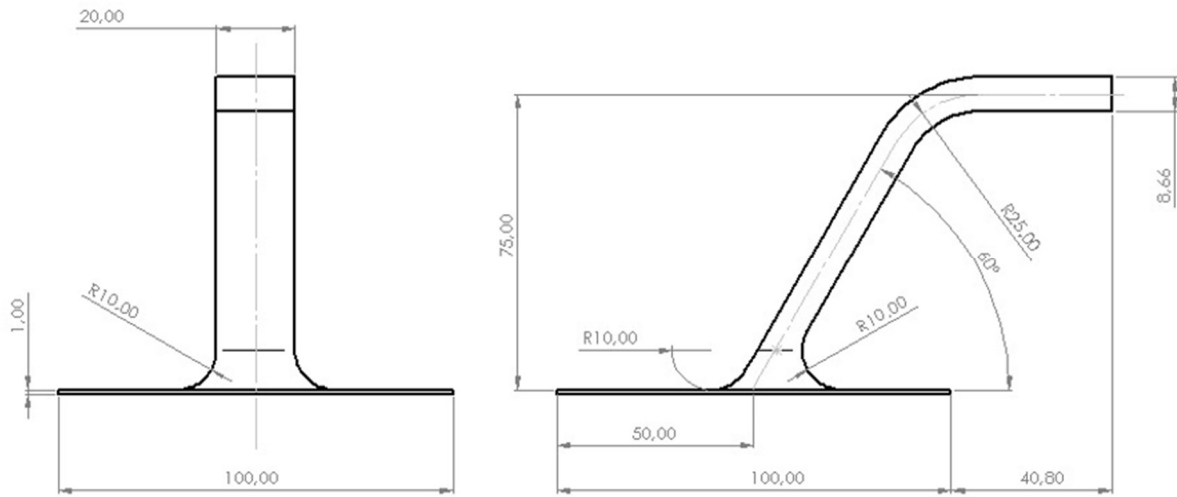


Figure 14: Component geometry.

The two electrical circuits (R1 and R2) constituting the strain gauges constituting the sensor are embedded within the base. Each circuit (arm) consists of two symmetrical parts. Each part is formed by five branches of different lengths ending with a copper electrode, which allows the electrical circuit to extend outside the component. Along the axis of symmetry of each arm, there is a circuit segment that connects all the branches together; this segment also terminates with a copper electrode. The electrodes will then be connected in different ways basing on the experimental campaign (see paragraph 3.3).

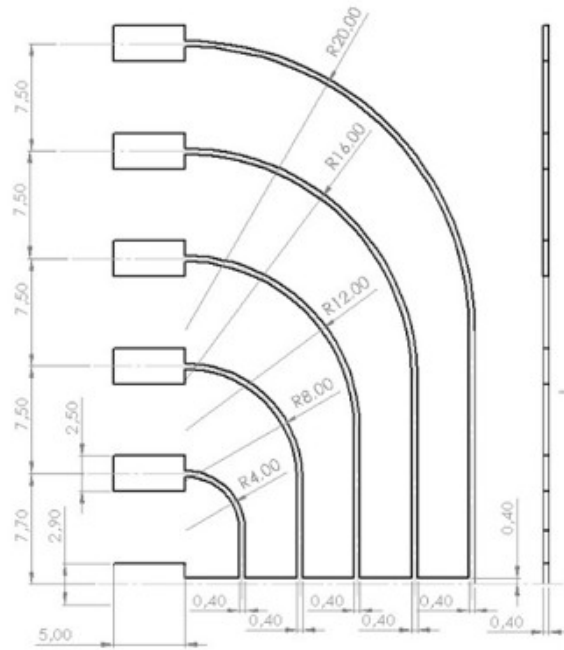


Figure 15: R1 circuit.

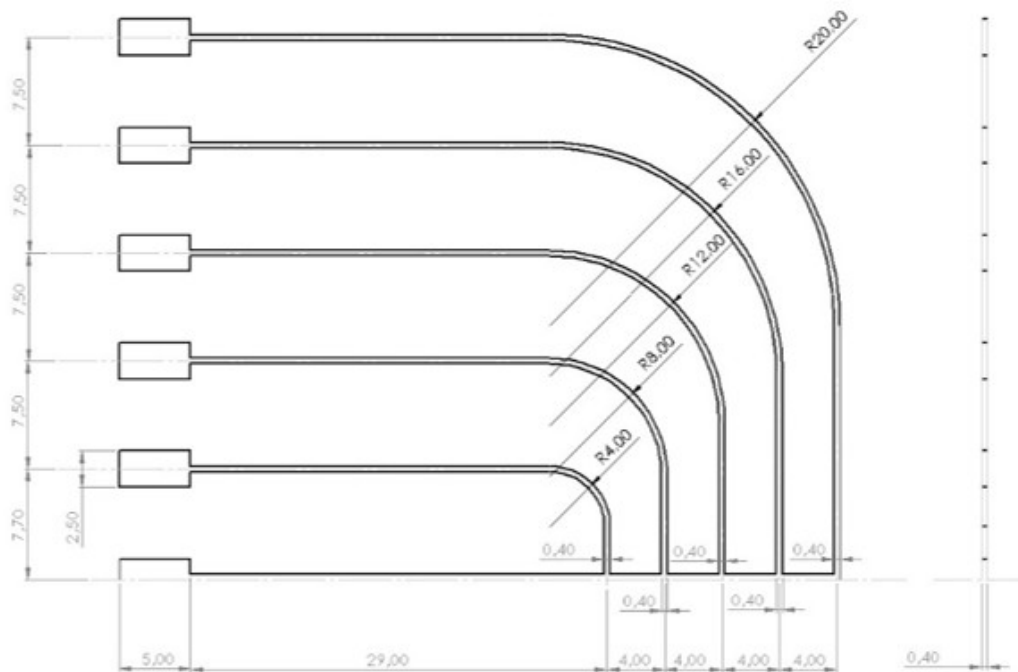


Figure 16: R2 circuit.

3.1.2 Materials

The specimen is manufactured via Fused Deposition Modeling (FDM). The main body is composed of Polylactic Acid (PLA), commercially available as Ultrafuse PLA PRO1 from BASF. The conductive circuit is made of a material commercialized by the Aimplas Technological Institute of Plastics under the name Fili. The Fili filament is a TPU (Thermoplastic Polyurethane) based flexible material with a 27.44 $\Omega\cdot\text{cm}$ resistivity. The inclusions are made of graphite: Aimplas declares that thanks to them the conductivity is slightly lower than materials with graphene particles, but higher than carbon based conductive filaments (carbon black, carbon fibers, etc.). The filament itself is flexible and does not break when handled. When extruded it shows no signs of retraction and it maintains a homogeneous diameter.

Here are reported the tables contained in the Technical Data Sheet of Fili by Aimplas.

		Standard	Unit	Typical Value
MECHANICAL PROPERTIES	Tensile Strength. Break	ISO 527	MPa	9,56
	Tensile Modulus	ISO 527	MPa	641
	Tensile Elongation. Break	ISO 527	%	7,4
	Flexural Strength	ISO 178	MPa	16.9
	Flexural Modulus	ISO 178	MPa	826
ELECTRICAL PROPERTIES	Volume Resistivity	Probe 10x4x25,9	$\Omega\text{ cm}$	27,44

Table 1: Fili Technical Data Sheet, table A.

NOZZLE DIAMETER	0.4 mm
PRINTING TEMPERATURE	250° C/260° C
TABLE TEMPERATURE	No
CHAMBER TEMPERATURE	No
PRINTING SPEED	40 mm/s
FILLING SPEED	40 mm/s
MATERIAL FLOW	100%
FILL DENSITY	100%
LAYER HEIGHT/QUALITY	0.2 mm
ADHESIVE/ADHESIVE TYPE	SKIRT

Table 2: Fili Technical Data Sheet, table B.

3.1.3 Process

The printing parameters are collected in the table below.

Parameter	Value
Material extruder 1 (E1)	Fili
Material extruder 2 (E2)	PLA-PRO1
Print core type	AA 0.4
Nozzle temperature (°C)	260 (E1) 220 (E2)
Bed temperature (°C)	65
Layer height (mm)	0.2
Layer width (mm)	0.4
Infill pattern	Grid
Infill density (%)	100
Printing speed (mm/s)	30 (E1) 70 (E2)
Pause at height (layer n°)	2

Table 3: Process parameters.

The AA print core type is suitable for extrusion with the chosen materials and for the required detail quality. For the same reasons, a diameter of 0.4 mm was chosen, which is the minimum possible according to the Technical Data Sheet (TDS).

The extrusion temperatures chosen are the maximum within the range indicated by the manufacturers. This ensures the best extrusion efficiency for the nozzle (any discontinuities in the raster would compromise the functionality of the sensor).

The bed temperature is a critical parameter for the successful realization of the first layers, particularly if, as in this case, they are voluminous. Setting it at 65°C ensures good adhesion of the first layer.

The layer height is the one recommended by the TDS for the Fili material. Its thickness derives from the nozzle diameter.

The grid infill pattern is the one that guarantees the best mechanical properties; also setting its density to 100 % has the same effect. In other words, the part is completely solid inside.

The printing speeds are chosen in accordance with the TDS of the materials. The speed for Fili was lower to ensure a robust and consistent raster. On the other hand, the speed for PLA was chosen to optimize the printing time.

The pause after printing the second layer (i.e., the first with conductive material) is necessary to insert the copper electrodes (see Figure 17).

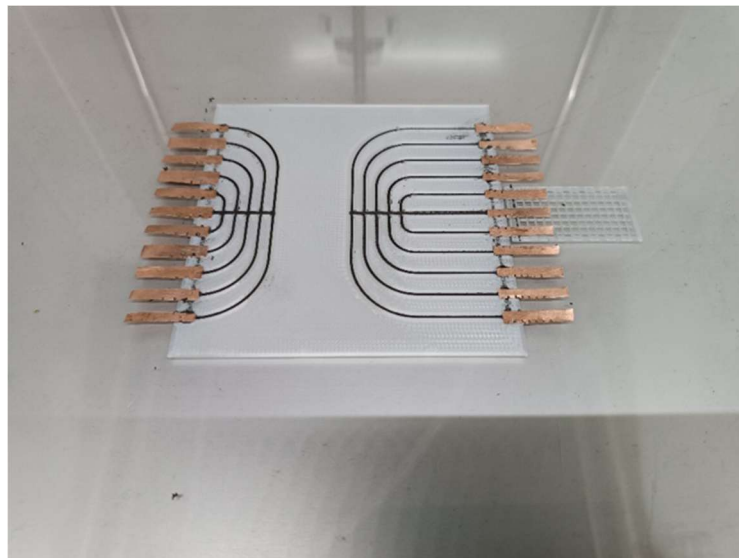


Figure 17: Pause after layer 2 to insert electrodes.

Afterward, printing continues until the specimen is completely built. When the process is ended, the male jumpers are connected to the copper electrodes, obtaining the complete specimen shown in Figure 18.

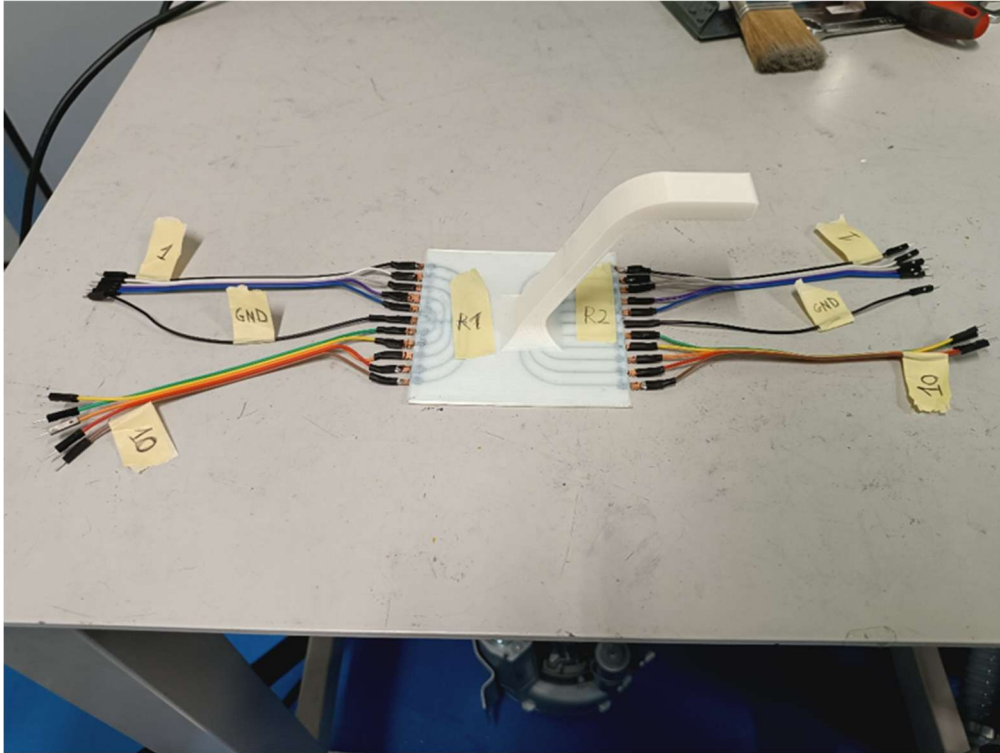


Figure 18: Finite prototype with jumpers.

3.1.4 Alternative Specimens

The use of jumpers allows to obtain an electrical circuit in which certain branches can be disconnected. Since one of the objectives of the sensor under study is precisely the identification of the number of broken branches inside it (to monitor its health status), their failure can be simulated simply by disconnecting the jumpers.

This was not the case for the first specimens manufactured. They were without jumpers, so the breaking of the branches had to occur physically, irreversibly compromising the specimen. Alternatively, several different specimens had been built, each with a different number of branches. Aside from the printing defects that can be noticed, an example of this alternative specimen configuration, which was later discarded, is shown in Figure 19.

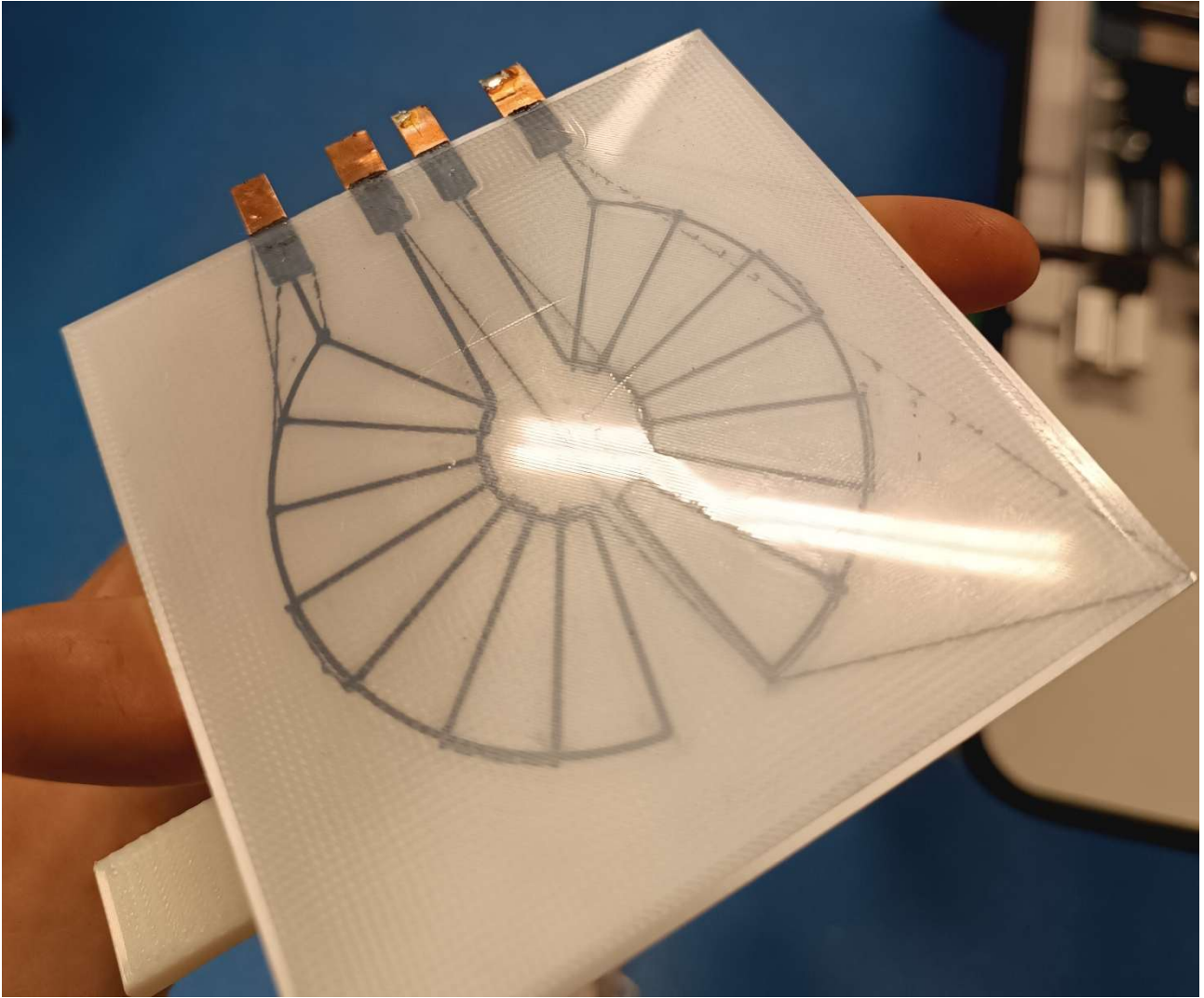


Figure 19: Alternative specimen.

3.2 Test Bench and Data Acquisition

The objective of the sensor is to monitor the health status of the component by identifying the strain gauge branches that are still intact, with reference to a fracture front propagation in real operative conditions. Secondly, the aim is to detect, as accurately as possible, the dynamics of landing and take-off, as well as the orientation of applied forces during landing.

Therefore, it is necessary to conduct numerous experimental tests to observe the behaviour of the specimen in different cases. Using these acquisitions, the sensor can be characterized so that it can monitor the mentioned quantities during operation.

Hence, the experimental tests consist of a series of loading/unloading cycles on the specimen, imposing a known displacement on its free end and measuring the equivalent resistance values in the two strain gauges. The resistances are measured between the central branch of each strain gauge and the parallel combination of all the other connected branches. The angle of the applied force can be varied, and it is measured using a protractor.

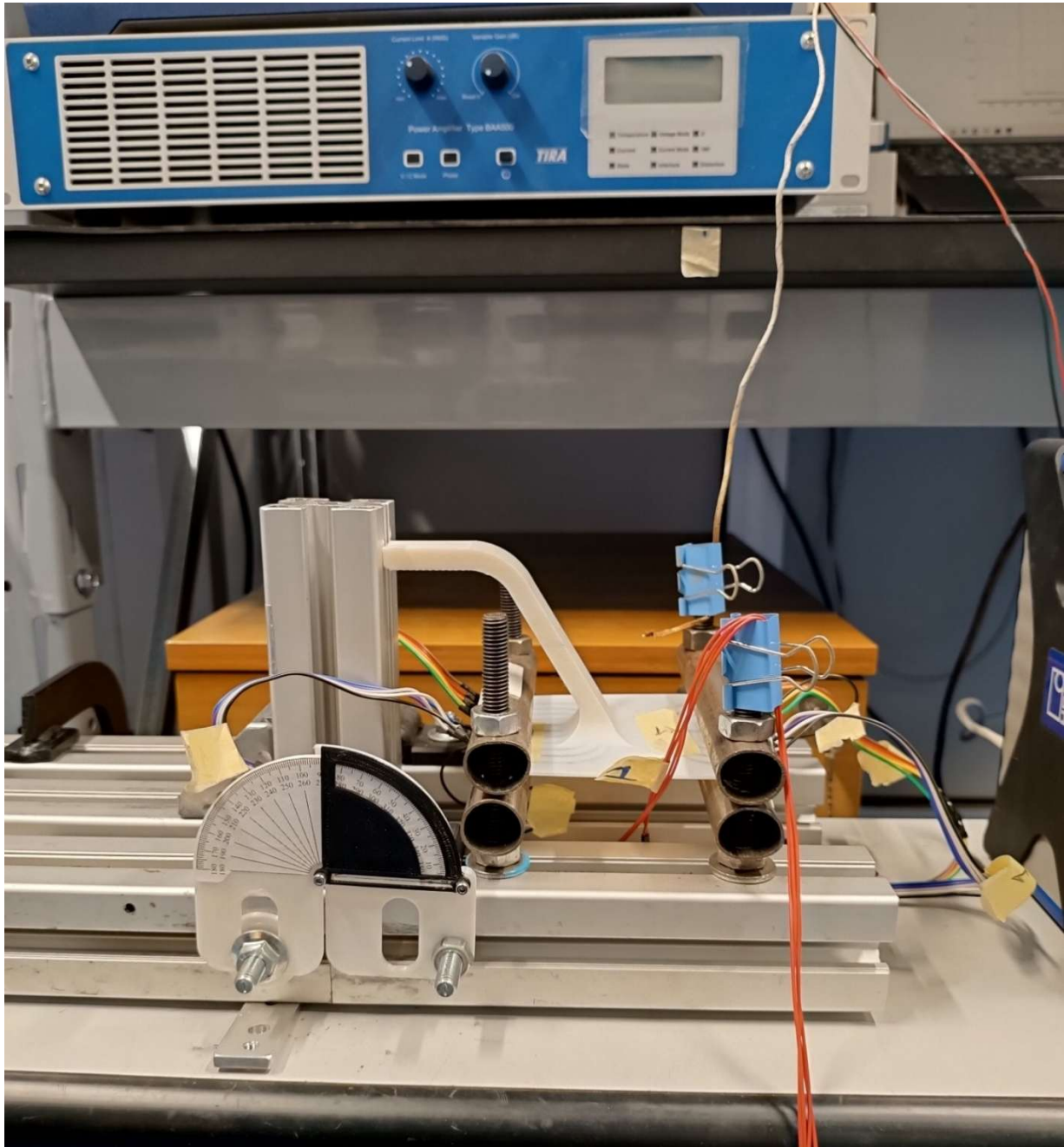


Figure 20: Test bench, front view.

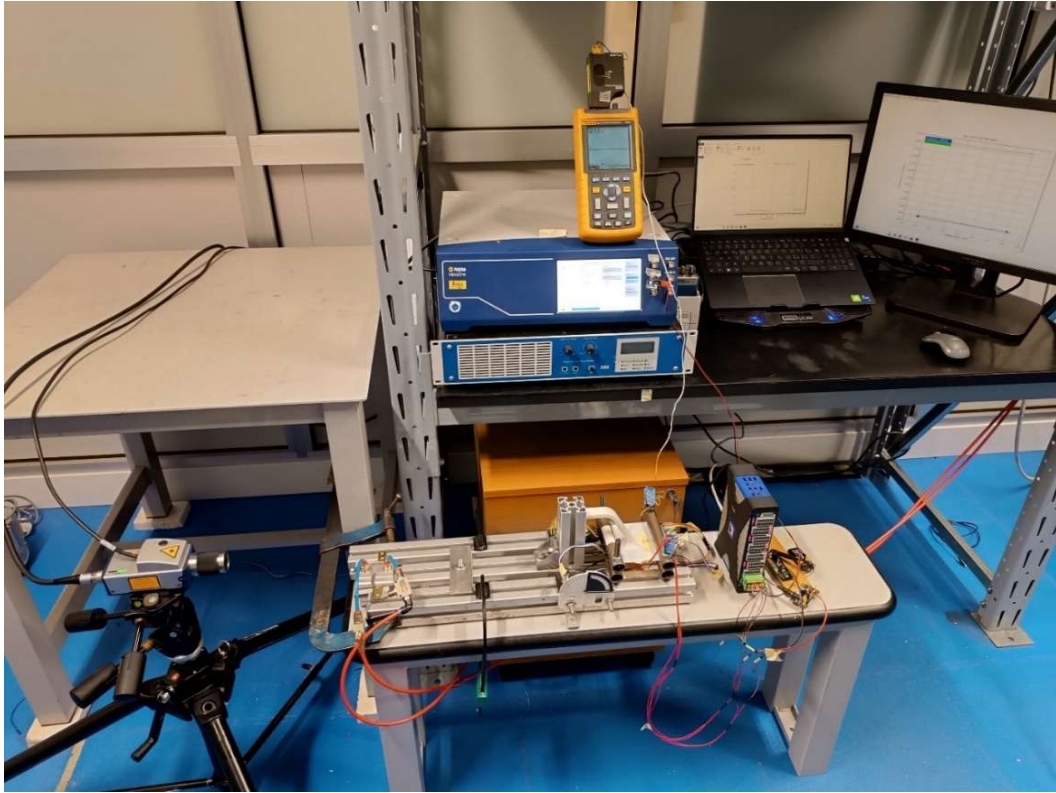


Figure 21: Test bench, overall view.

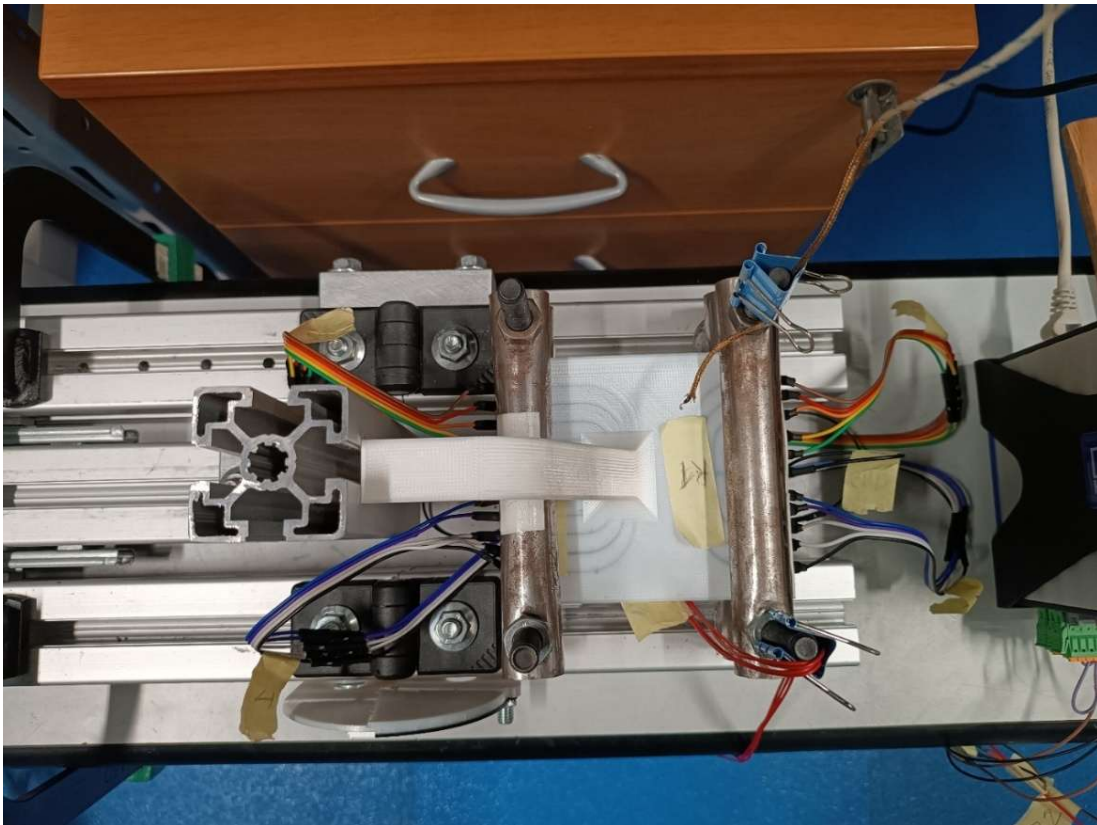


Figure 22: Test bench, top view.

The displacement is imposed via a mobile crosshead and it is recorded using a VibrONE by Polytec vibrometer. The maximum displacement range is set to 10 mm. This is because, thanks to a mechanical stopping system, the mobile crosshead, driven by a pneumatic actuator, is able to move repeatably between the rest position and a maximum nominal displacement of 3 mm. Through the connection to the console, the signal is converted into voltage using a sensitivity of 5 mm/V, in this way it is made available to the data acquisition system.

The electrical resistance of the two sensor parts (R_1 and R_2) is measured independently via voltage dividers powered by a total potential difference of 3.3 V obtained through a 9 V battery via a step-down converter. The electrical resistance in CPCs is typically elevated, sometimes in the $k\Omega$ range. The systems capable of measuring electrical resistances are often unable to reach these orders of magnitude. Then, voltage divider is used to measure the electrical resistance (see Figure 23).

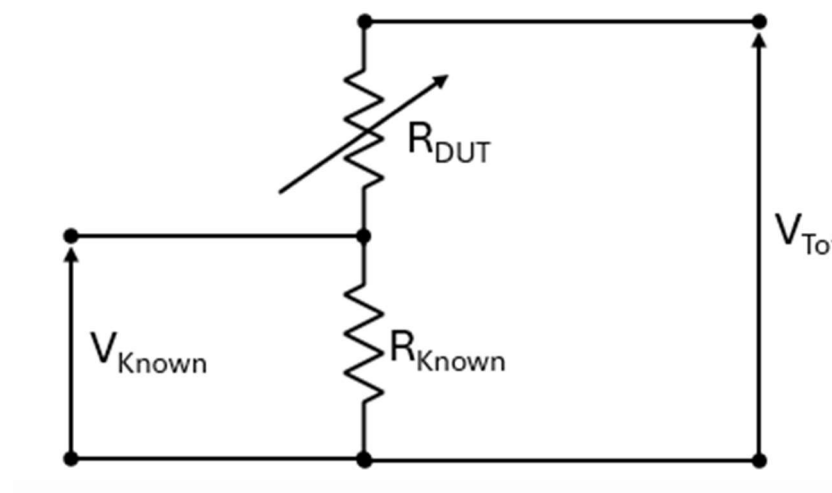


Figure 23: Voltage divider.

- V_{Tot} : the 3.3 V voltage given by the battery.
- V_{Known} : the voltage measured.
- R_{DUT} : the resistance of the Device Under Test (i.e. the sensor R_1 or R_2), to be measured.
- R_{Known} : a known resistor with a nominal value of 100 $k\Omega$.

Modelling the divider as an ideal circuit, we can assume that a current (I) flows through the resistances, but not in the left horizontal branches (the voltmeter is ideal). Through Ohm's first law we obtain:

$$\begin{cases} V_{Tot} = I * (R_{Known} + R_{DUT}) \\ V_{Known} = I * R_{Known} \end{cases}$$

$$\Rightarrow R_{DUT} = R_{Known} * \left(\frac{V_{Tot}}{V_{Known}} - 1 \right)$$

Thus, measuring a low voltage (V_{Known}) we can measure an elevated resistance (R_{DUT}).

The temperature is measured at the beginning of each test with a K-type thermocouple using a 123 Industrial Multimeter by FLUKE. It will not be used in any of the approaches, but it could have provided interesting data in the search for potential noise deriving from external sources.

Data synchronization is performed via a QuantumX MX1615B DAQ and managed through CatmanEasy DAQ software by HBK. Signals are sampled at 300 Hz, given the dynamic nature of the piston action (the movement duration is roughly 0.05 s).

The actuator is operated by a 3-way, 2-position manual lever pneumatic valve. For each test, 10 excitation cycles are applied via manual control.

As previously described, damage propagation is simulated through the selective and programmed disconnection of the jumper connectors by which each branch is linked.

The acquisitions were performed with 2 or 3 successive iterations (depending on the experimental campaign). The damage simulation is carried out at different angles to simulate different landing conditions.

3.3 Datasets

Following what was illustrated in the previous section, the planned iterations were conducted, building two different datasets:

- Crack Propagation dataset (CP dataset), whose objective is to simulate the propagation of a crack along the component by progressively disconnecting the various jumpers to indicate the failure of the different branches;
- Single Segment Characterization dataset (SSC dataset), in which the configurations of connected jumpers allow the characterization of every single segment of the circuit.

3.3.1 Crack Propagation Dataset

In this dataset, the goal is to highlight how the failure of each branch causes an increase in the equivalent resistance measured between the parallel combination of all intact branches and the end of the central segment (which constitutes the ground). This phenomenon is exploited to monitor the structural health status of the component. The electrical scheme is represented in Figure 24, where the jumpers are indicated as switches.

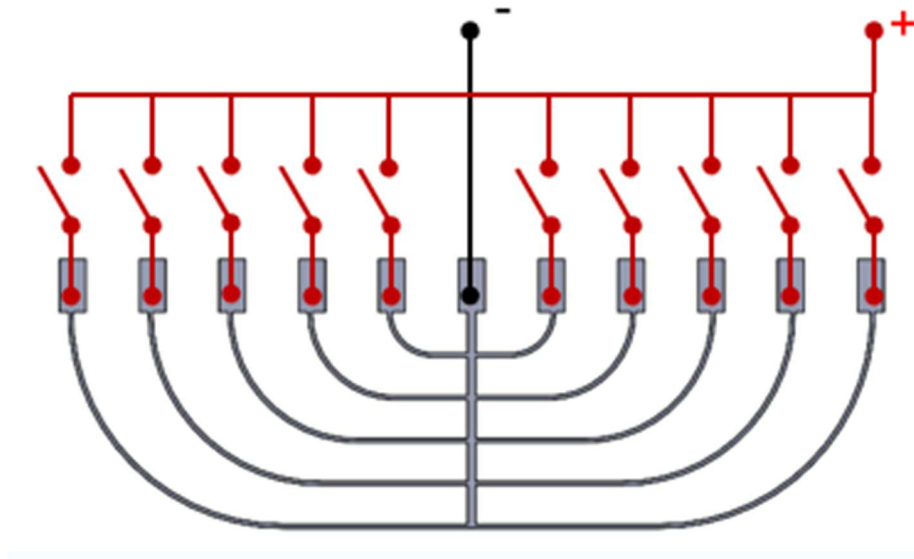


Figure 24: Electrical scheme of an arm of the sensor.

The measured resistance is the sum of the contributes given by all the segments involved in the connection between negative and positive poles. To simulate crack propagation, resistance values are acquired with different configurations (IDs) of connected jumpers. The first acquisition (ID1) is performed with all jumpers connected. Assuming that the crack propagates from the outside inward, ID2 corresponds to the configuration where the outermost branch of the circuit breaks (thus, its jumper is disconnected). Since the two equivalent resistances are measured independently, the same branches are disconnected in both.

Continuing in this way, for ID3 the first branch remains disconnected, and also the second one is disconnected. This proceeds up to ID10, where only the innermost branch is considered intact, on the side opposite to the crack nucleation site. Note that it is meaningless to measure resistance if all the branches are broken, as it would result in a null value because the circuit would be open. For the same reason, it is assumed that the crack does not damage the central ground connection branch: otherwise, a constantly null resistance would be measured. In a two-dimensional plane, this crack behaviour would not be possible, but, since the component has its own thickness, the described scenario can be assumed.

Proceeding with ID11, the last remaining jumper is disconnected while the first one that had been disconnected is reconnected. Then, progressively from ID12 to ID20, the jumpers are reconnected in the same order in which they were disconnected. In this way, in reverse order, IDs 11 to 20 simulate the propagation of a crack nucleated from the opposite side compared to IDs 1 to 10. Note that, due to the symmetry of the component, IDs referring to symmetrical situations (hence, IDs whose sum equals 21) are ideally expected to lead to identical values of equivalent resistance. Figure 25 illustrates what has just been described.

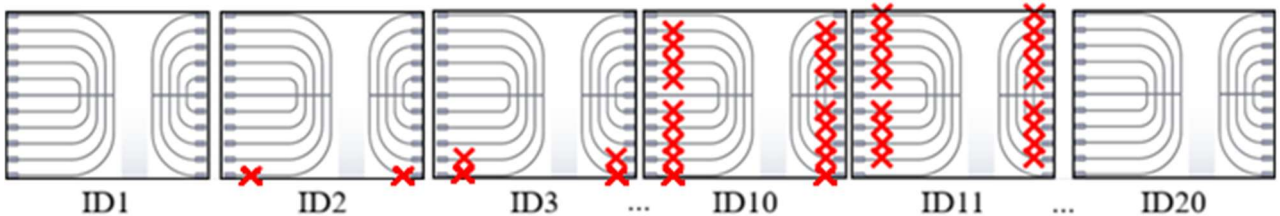


Figure 25: ID configurations, CP dataset.

For each ID, 10 loading/unloading cycles are applied, as described in the previous section. Three iterations of this test are performed, and the entire process is repeated for stress angles of 0° and 15° . In this way, 1200 cycles are obtained to characterize the specimen (this number resulting from the combination of 10 cycles, 3 iterations, 20 IDs, and 2 angles). An example of a test in time domain is shown in Figure 26.

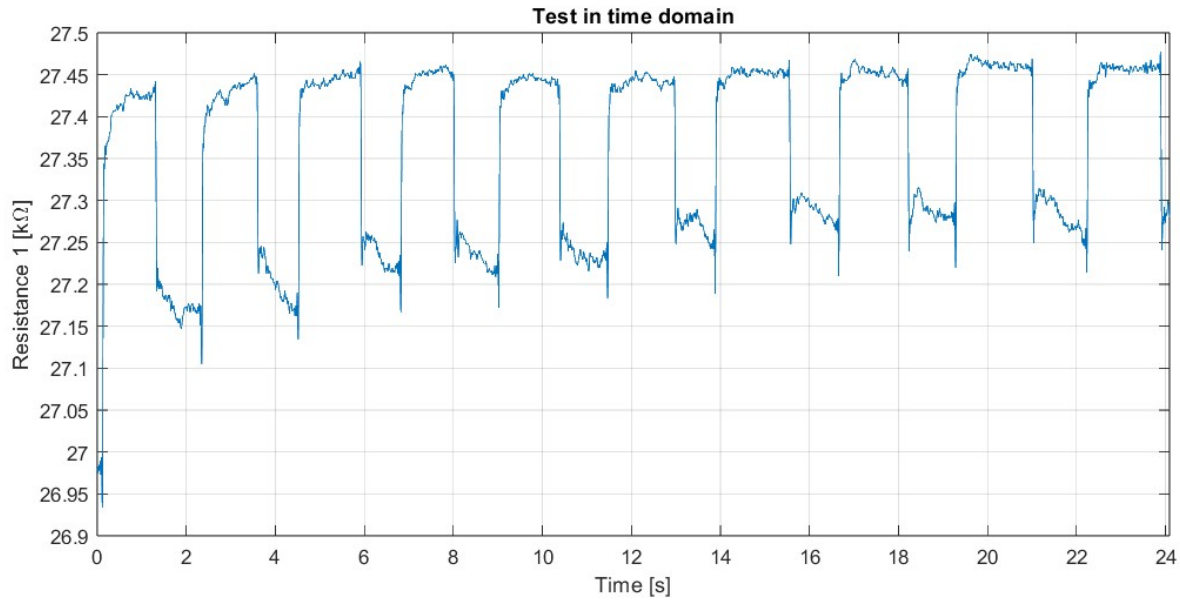


Figure 26: Test in time domain.

The data were classified and organized within a Matlab structure and subjected to post-processing:

- identification and removal of anomalous peaks attributable to noise;
- noise reduction in the resistance signals using a digital low-pass Butterworth filter;
- signal segmentation, imposing the first actuation of the piston as the start, and the last return of the piston to its rest position as the end.

Figure 27 graphically shows the segmentation applied to a test in time domain. To see how it is implemented, see the script *split_test.m* in paragraph 10.9.

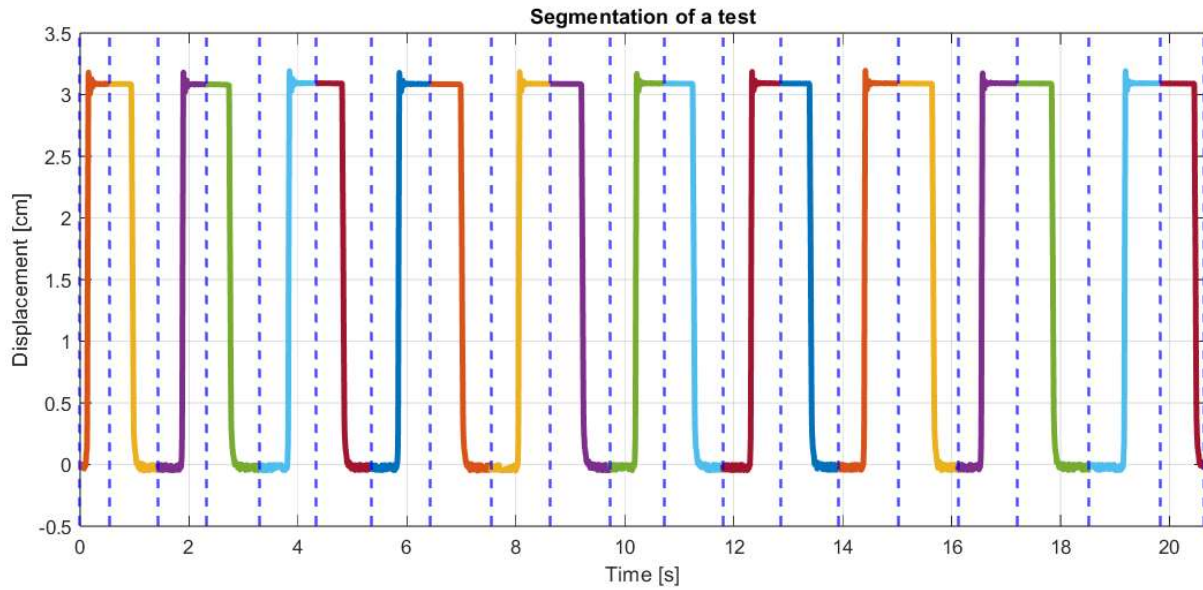


Figure 27: Segmentation of a test.

By observing the resistance values as a function of displacement, we can notice that the relationship is not biunivocal (see Figure 28). Therefore, to characterize the sensor, it is not sufficient to associate a displacement value with each resistance value. This necessitates the use of more complex approaches, which constitute the core body of this thesis.

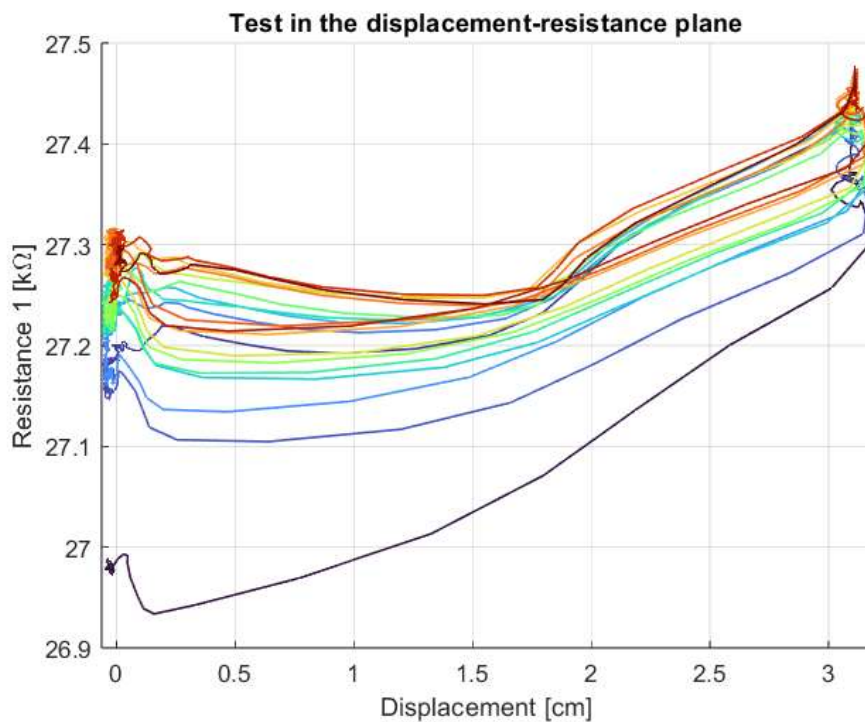


Figure 28: Test in the displacement-resistance plane.

3.3.2 Single Segment Characterization Dataset

For the characterization of the individual segments constituting the circuit, a second experimental campaign was conducted simultaneously with the first one. The procedure is the same: 10 loading/unloading cycles via imposed displacement, 3 iterations, and 2 possible stress angles. The only difference lies in the configurations of the disconnected jumpers. For the characterization of individual segments to be possible and unique, 15 configurations are required, identified by IDs.

Using the same scheme adopted in Figure 24, but without the connection between all the numbered pins, the first 10 configurations (ID1 to ID10) measure the resistance obtained by connecting the ground with a single pin. As in the first dataset, since the resistances of the two arms are measured separately, the same pins are disconnected in both.

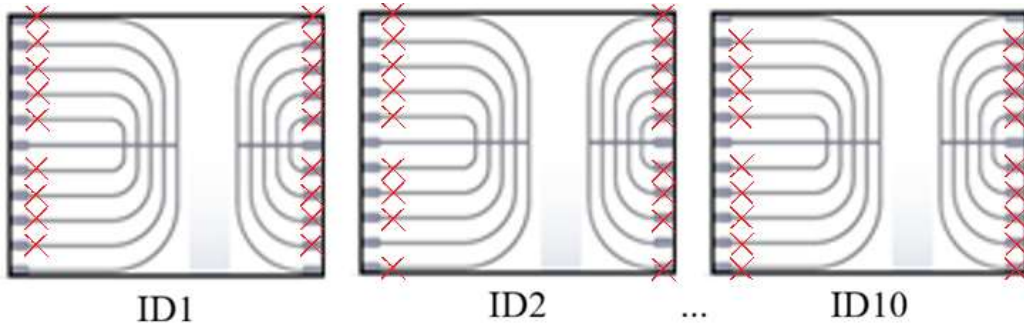


Figure 29: ID1 to ID10 configurations, SSC dataset.

The last 5 configurations (ID11 to ID15) measure the resistance obtained by connecting two symmetrical pins (i.e. two pins whose sum gives 11). Hence, the ground and all the pins are disconnected except for a symmetrical pair.

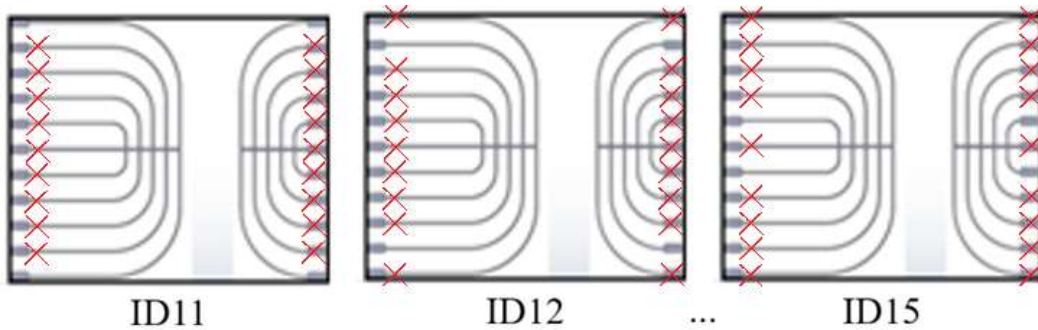


Figure 30: ID11 to ID15 configurations, SSC dataset.

The reasoning that leads to these configurations is described in paragraph 7.1, where this dataset is employed.

3.3.3 Alternative Dataset

The first dataset obtained in chronological order was constructed differently, as the entire test bench and the specimens were different. The displacement was imposed manually by turning a hand crank that moved the mobile crosshead. The specimens in use were not equipped with jumpers: they were the alternative type described in paragraph 3.1.4. Therefore, to simulate a different number of broken branches, the entire specimen was replaced with one having a different number of branches. This introduced greater uncertainty into the tests, which was accentuated by the use of a less performing data acquisition system.

Another remarkable difference lays in the fact that the tests were performed for a greater number of stress angle values: 0° , 15° , 30° , 45° , and 60° .

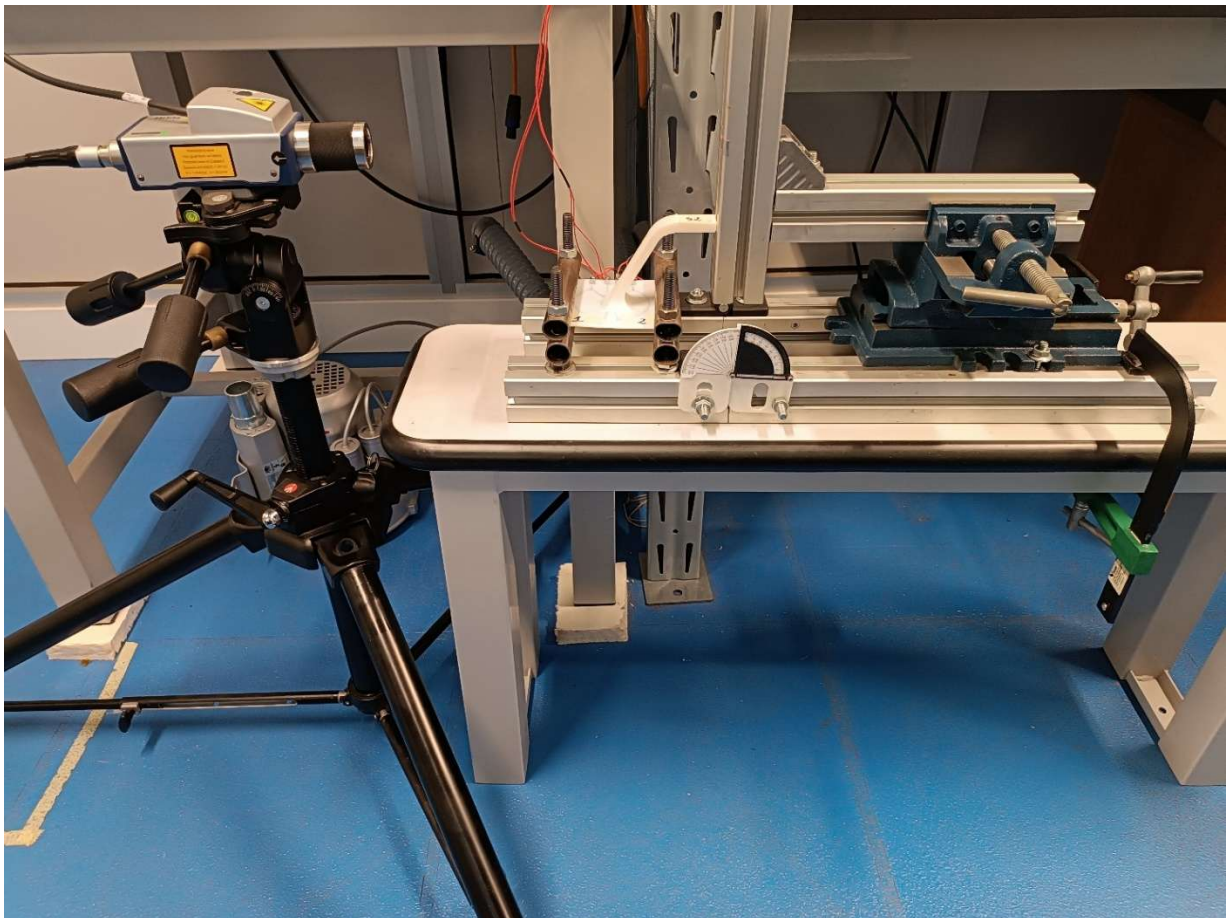


Figure 31: Alternative test bench.

Overall, this first acquired dataset proved to be significantly less reliable than the subsequent ones; therefore, it was put aside. However, it proved to be particularly interesting for approaching the problem for the first time, highlighting potential irreversibility phenomena and laying the groundwork for choosing the setup for the following experimental campaigns.

4 First Approach: Boundaries

4.1 Algorithm Explanation

4.1.1 Data Management

The goal of the algorithm is to earn the greatest possible amount of information from the experimental dataset obtained to characterize the sensor. This algorithm was originally based on the first obtained dataset (the alternative one, see paragraph 3.3.3): since it was the least reliable, the algorithm has been applied also to the Crack Propagation dataset. Here is again the structure of this experimental dataset.

- Angle 0°
 - ID 1
 - n 1: [Time; Displacement; Resistance 1; Resistance 2]
 - n 2: [Time; Displacement; Resistance 1; Resistance 2]
 - n 3: [Time; Displacement; Resistance 1; Resistance 2]
 - ...
 - ID 20
 - n 1: [Time; Displacement; Resistance 1; Resistance 2]
 - n 2: [Time; Displacement; Resistance 1; Resistance 2]
 - n 3: [Time; Displacement; Resistance 1; Resistance 2]
- Angle 15°
 - ID 1
 - n 1: [Time; Displacement; Resistance 1; Resistance 2]
 - n 2: [Time; Displacement; Resistance 1; Resistance 2]
 - n 3: [Time; Displacement; Resistance 1; Resistance 2]
 - ...
 - ID 20
 - n 1: [Time; Displacement; Resistance 1; Resistance 2]
 - n 2: [Time; Displacement; Resistance 1; Resistance 2]
 - n 3: [Time; Displacement; Resistance 1; Resistance 2]

As explained in paragraph 3.3, each test n consists of a series of 10 loading/unloading cycles by imposing a determined displacement and coming back to the rest position. Each load or unload phase is followed by a few seconds keeping a constant position. Then, to manage the data in a better way, each test n has been divided into 20 sub-tests: 10 for the loading phase and 10 for the unloading one. So, now we can appreciate a new parameter for a test: the loading phase. It is assigned value 0 for the loading phase and value 1 for the unloading one. Every Machine Learning algorithm requires also that all the data are divided into a training part and a test part: here the validation part is not required thanks to the nature of this algorithm. Thus, the tests have been randomly divided keeping 80 % of them for the training and 20 % for the test. To ensure that each combination of angle, ID and loading/unloading phase would be tested, the randomness is applied only for picking 20 % of each combination of these three parameters. In other words, for each combination of angle, ID and loading/unloading phase there are 30 tests (10 for each sequence n): 6 of them (20 % of 30) is randomly picked and kept for the test part. Finally, the dataset has been reorganized into two tables (training and test) basing on the three main variables characterizing each test: angle, ID and loading/unloading phase. To simplify the reading, from now on the loading/unloading phase variable will be called just “load”. Here is the structure of the two tables: the training one and the test one.

	Angle 0° - Load 0	Angle 0° - Load 1	Angle 15° - Load 0	Angle 15° - Load 1
ID 1	24 tests	24 tests	24 tests	24 tests
ID 2	24 tests	24 tests	24 tests	24 tests
...	...	...	...	...
ID 20	24 tests	24 tests	24 tests	24 tests

Table 4: Training table structure.

	Angle 0° - Load 0	Angle 0° - Load 1	Angle 15° - Load 0	Angle 15° - Load 1
ID 1	6 tests	6 tests	6 tests	6 tests
ID 2	6 tests	6 tests	6 tests	6 tests
...	...	...	...	...
ID 20	6 tests	6 tests	6 tests	6 tests

Table 5: Test table structure.

Now that has been explained how the data are managed, we can see how this algorithm works. To do that, we have to introduce a notation for some aspects.

Stress angle, broken branches (indicated by the ID) and load are called “**variables**”.

The set of all tests for a certain combination of stress angle, broken branches and load is called “**family**”; when all the variables are known, a family coincides with the cell of a table (training or test).

For the definition of the resistances, the displacements or other features, see paragraph 3.

Since there are three main variables, the idea for the first approach is to design an algorithm that can recognize one or more of them for a certain profile of the resistances 1 and 2 with respect to the displacement. Now can be clear why the data were reorganized in the previously explained way. The algorithm will be trained using the resistance profiles for each family, in order to recognize the family to whom a new test profile belongs. In other words, the first approach is a classification algorithm that works on three levels, one for each variable.

4.1.2 Training

The training is given by the following steps.

- Choose which one of the two measured resistances will be used for the classification. The test could be repeated for the other resistance, but only one resistance at the time can be used.
- Build a family set for each combination of stress angle, broken branches and load: only the tests contained in the training table can be used in this step. See the Matlab script *Build_training_and_test_table.m* in the appendix for the details.
- In the displacement-resistance plane, for each family build the boundaries of the region that contains all the profiles of that family. The boundaries are two: the upper one and the lower one. The first one contains the highest value of all the resistance profiles for each displacement value; the second one contains the lowest value of all the resistance profiles for each displacement value. To have continuous boundaries, use interpolation between the experimental points. For each family, search the maximum and the minimum values of displacement measured in all the tests and build an array with 200 equally spaced points between them. Then, interpolate the resistance value for each record using these points and finally obtain the two boundaries of the resistance

with respect to the displacement. Every boundary computed in this way is stored in a table. See the Matlab scripts *Build_boundaries_table.m* and *boundaries.m* in the appendix for the details.

Figure 32 shows an example of region: the one referred to the family with no broken branches (ID1 and ID20), stress angle of 0° in unloading phase (load = 0), using resistance 1. Looking at the graph, it is possible to see the influence of ambience and time on the tests. Each colour represents a single test performed in a different instant of time during the sampling weeks: the fact that they are clearly distinct introduces the topic of the irreversibility that will be covered in the next chapters.

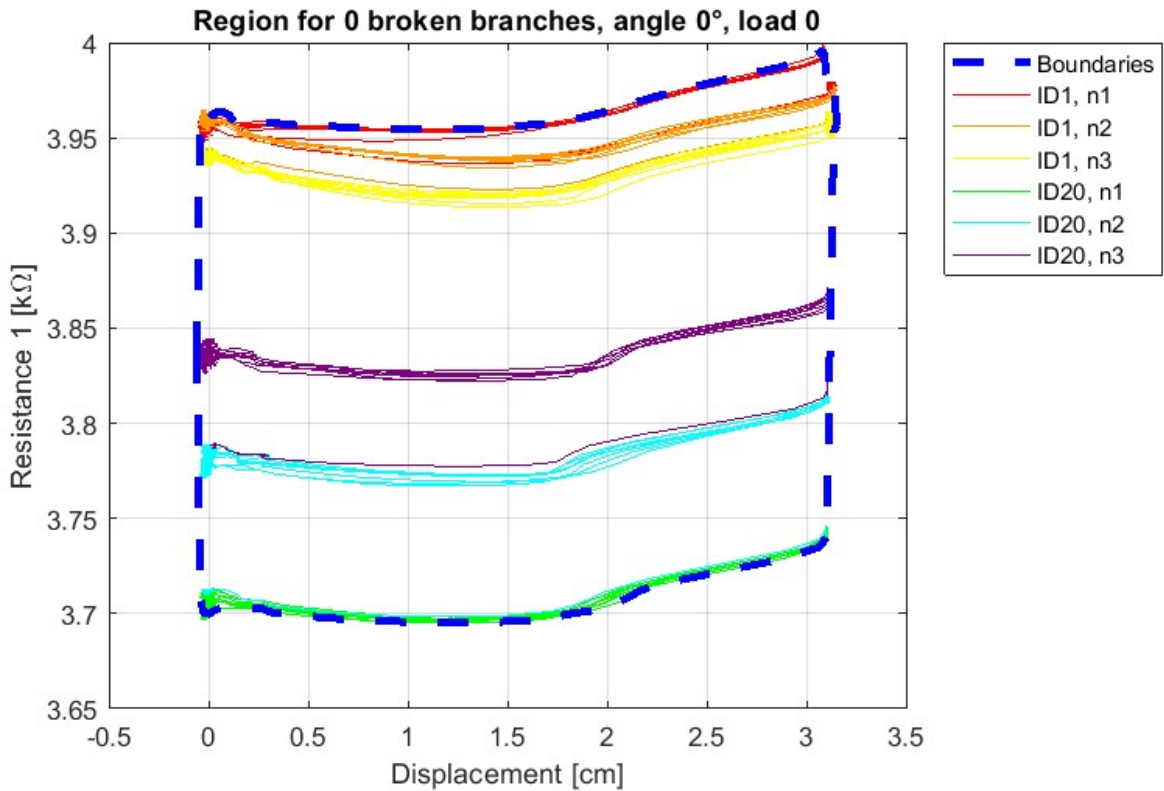


Figure 32: Example of the region of a family.

4.1.3 Test

- Define which variables are known and which variable we want to classify. Indeed, we could assume that we want to classify a certain variable but also the other variables are not known, or maybe just one of them is known. However, only one variable at the time can be classified.

- Reorganize the families in this way:
 - if a variable is assumed to be known, discard all the families with a different value for that variable;
 - if a variable we do not want to classify is assumed to be unknown, group all the families with the same value for all the other variables and compute the total boundaries of this new family.

For example, assume that we want to classify the broken branches (ID). We know that the given test profile has a stress angle 0° but we do not know if it refers to a loading or unloading phase. So, we must discard the useless families with different angles (third and fourth columns in Table 4) and group together the families with same ID and angle, but different load condition (for each row, the first and the second columns in Table 4). In this way, we obtain 20 families: (angle 0° , load 0/1, ID1), (angle 0° , load 0/1, ID2), ... , (angle 0° , load 0/1, ID20).

- Once the boundaries have been built, we can use the test profiles to classify the chosen variable. Given the test displacement-resistance profile, we compute a score to estimate the probability of belonging to each family. For each point of the test profile, if it falls within a family region, it is added to the family score the following value:

$$0.5 + 0.5 * \frac{\text{distance from the mean value of the region for that displacement}}{\text{half the width of the region for that displacement}}$$

- The family with the highest score is the chosen family.

How was this score designed? It is arbitrary, but obviously it is not casual. It derives by the combination of three ideas:

- the highest possible score assigned to a single point is 1;
- just the fact of being inside a region gives a part of score to the point;
- the score is lower along the boundaries and higher in the central part of the region.

Thus, the designed score consists of a 0.5 step given by being inside the region and an additive score that grows linearly from 0 in the boundaries to 0.5 in the mean value of the region for that displacement.

4.1.4 Example

Here is an example that shows how the algorithm works. The most reliable dataset is the one described in the paragraph about data management (paragraph 4.1.1), but it contains only two values for the stress angle (0° and 15°). So, despite the alternative dataset is less reliable, here is shown an example belonging to that dataset because it contains five values for the stress angle (0° , 15° , 30° , 45° and 60°); hence, the number of the families is most appreciable.

So, in this example we want to classify the stress angle of a test assuming that we know the exact number of broken branches, but we do not know which is the load phase. We arbitrarily choose to use resistance 2 for the classification. We collect the displacement-resistance profiles for each family; we keep only the ones relating to the right number of broken branches; we group the families with same stress angle and different load; we build the upper and the lower boundaries and we obtain the solid lines, defining a region for each possible angle. The dot line shows the experimental points of the test.

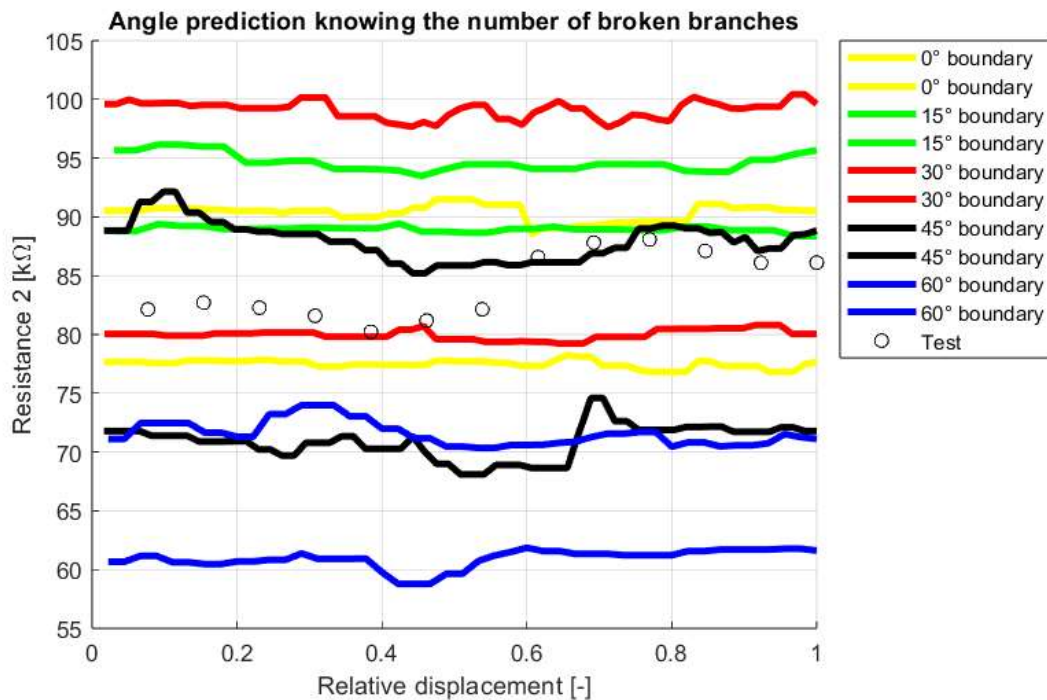


Figure 33: Angle classification knowing the number of broken branches.

As we can see, all the test points are contained in the 0° region, between the yellow boundaries, and some of them are also quite close to the medium value of the region for each displacement. All the points are contained also in the 30° region (the red one), but they are closer to the lower boundary.

Almost all the points belong also to the 45° region (the black one), but again they are close to the upper boundary. None of the other families contains any points inside its region. All in all, we expect the 0° family to have the highest score, the 30° and 45° families to have medium-high scores and the other families to have a null score. These are the scores computed by the algorithm:

0° angle: 10.1889

15° angle: 0

30° angle: 9.2974

45° angle: 8.3628

60° angle: 0

Estimated angle: 0°

Real angle: 0°

We have just seen the strength of this algorithm: despite some points belonging to the wrong regions, the closeness to the central part of the right region allows to assign the right angle to the test.

4.2 Results

To evaluate the algorithm, we can do some qualitative considerations and some other quantitative. In this paragraph, we will see the second ones.

For how the algorithm is built, the test will concern the classification of one variable at the time. To see when it works better, it is useful to test the classification for all the variables and assuming every combination of known and unknown variables. For each of these conditions, we will use the entire test table making the algorithm classify the desired variable and comparing the classification with the real value.

A good tool to show the results of this test is the confusion matrix. The rows of this matrix indicate the possible real values of the classified variable; the columns indicate the classified ones. The result of each test is collected in the matrix adding 1 to the cell whose row and column correspond, respectively, to its real and classified values. So, the ideal confusion matrix is a diagonal matrix, where classified and real values are equal for each test.

See the Matlab scripts *Build_confusion_matrix_broken_branches_boundaries.m* and *find_boundaries.m* in the appendix for the details.

A proper parameter to evaluate a classification algorithm is the accuracy. It is computed as follows:

$$Accuracy = \frac{\text{correct predictions}}{\text{total predictions}}$$

The following figures show all the possible confusion matrices and their accuracy. The amount of these matrices is 24; this number derives by the combination of all the possible conditions obtainable from the three classifiable variables, the two remaining variables to assume known or unknown, and the two resistances to use. We recall that an ideal confusion matrix is a diagonal one, where real and classified values are equal for each test. Thus, the more tests are placed outside the diagonal, the less ideal the confusion matrix is.

The used dataset is the Crack Propagation dataset. Hence, there are only two possible values for the stress angle: 0° and 15°.

Note that the classified values are labelled as “predicted”: in the ML context, prediction is a typical used word, so it has been kept.

Stress angle classification

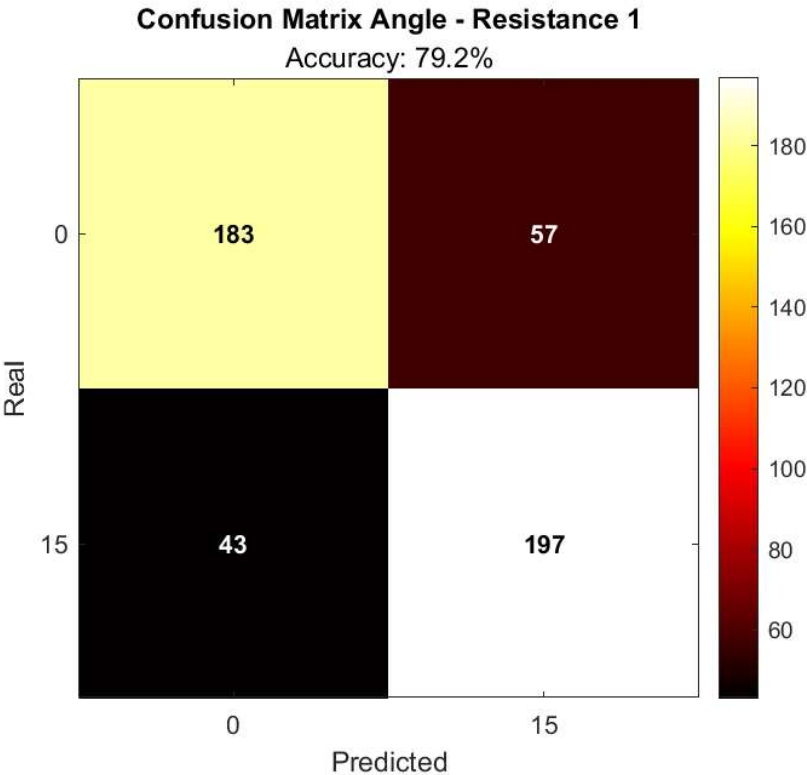


Figure 34: Confusion matrix for angle, resistance 1 – Broken branches known, load known.

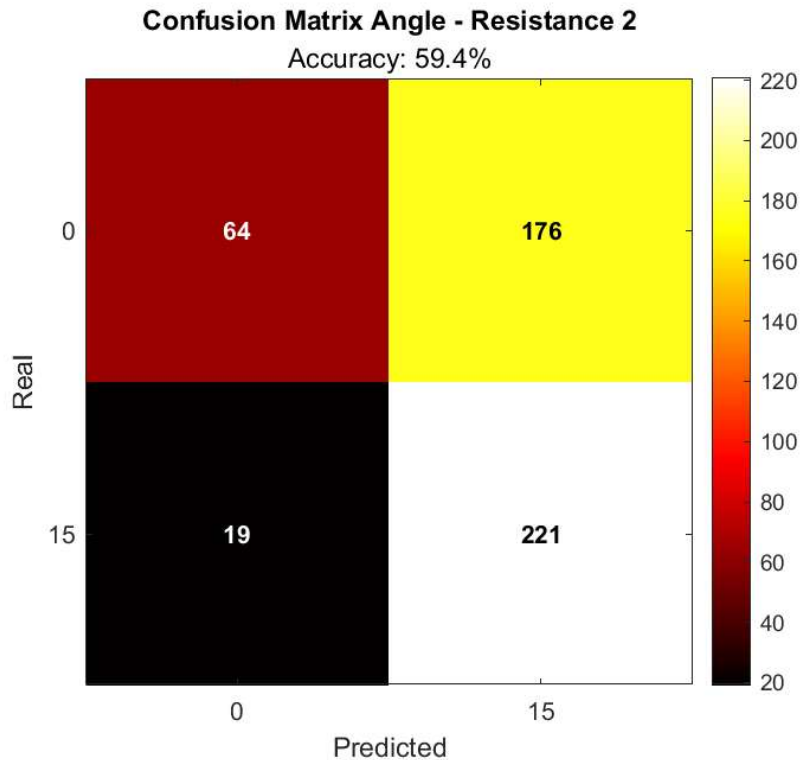


Figure 35: Confusion matrix for angle, resistance 2 – Broken branches known, load known.

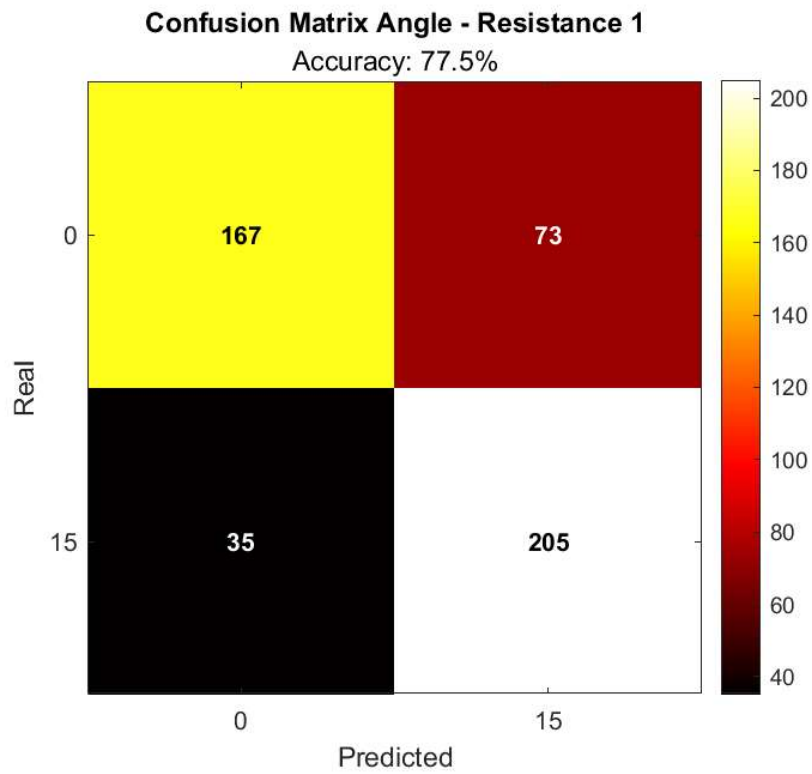


Figure 36: Confusion matrix for angle, resistance 1 – Broken branches known, load unknown.

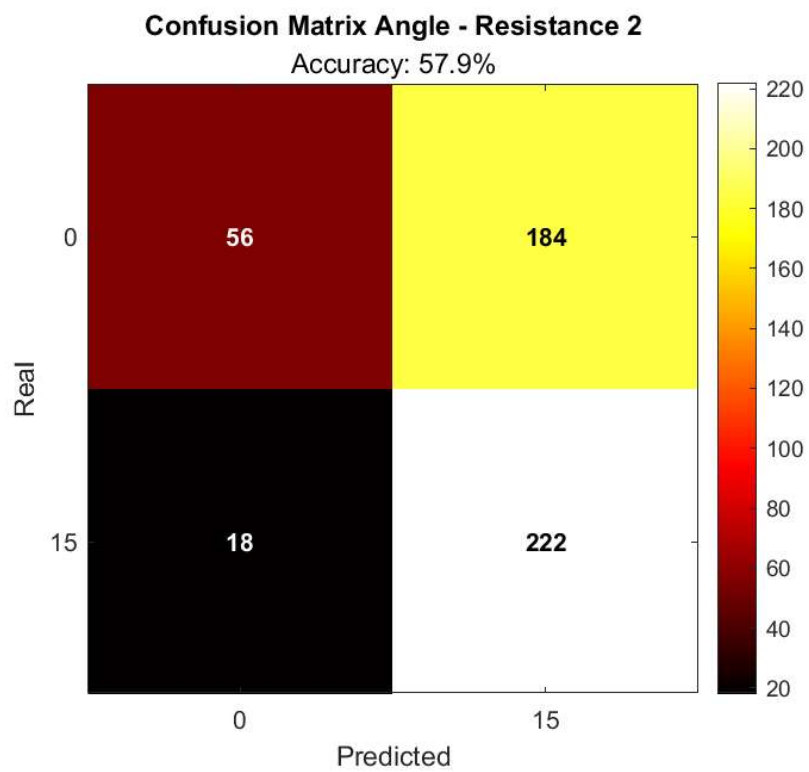


Figure 37: Confusion matrix for angle, resistance 2 – Broken branches known, load unknown.

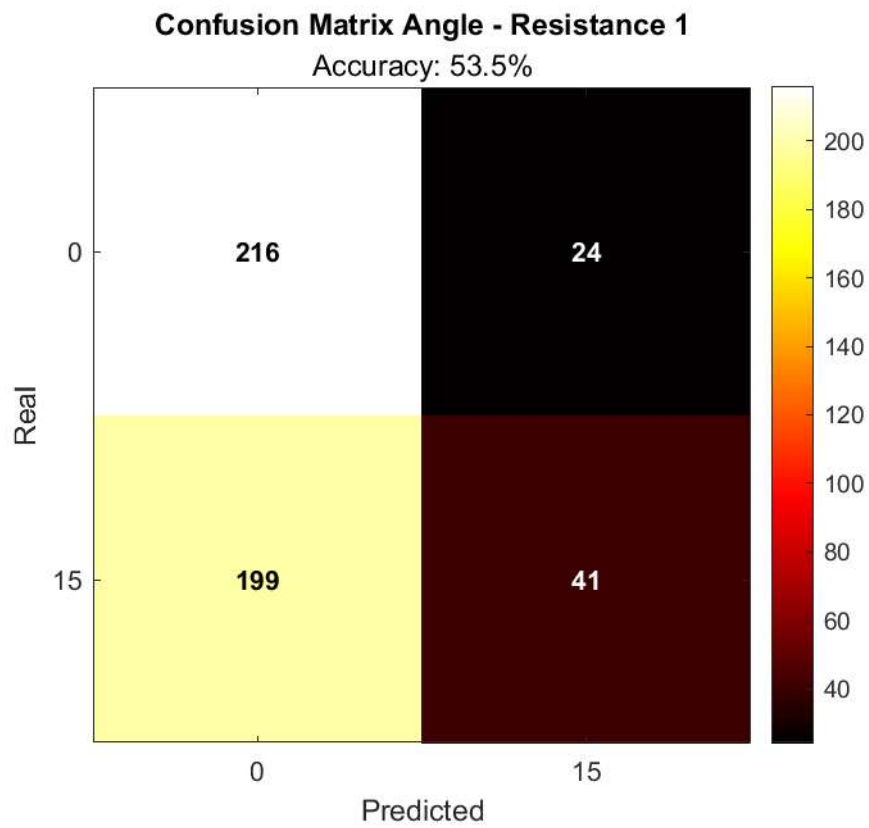


Figure 38: Confusion matrix for angle, resistance 1 – Broken branches unknown, load known.

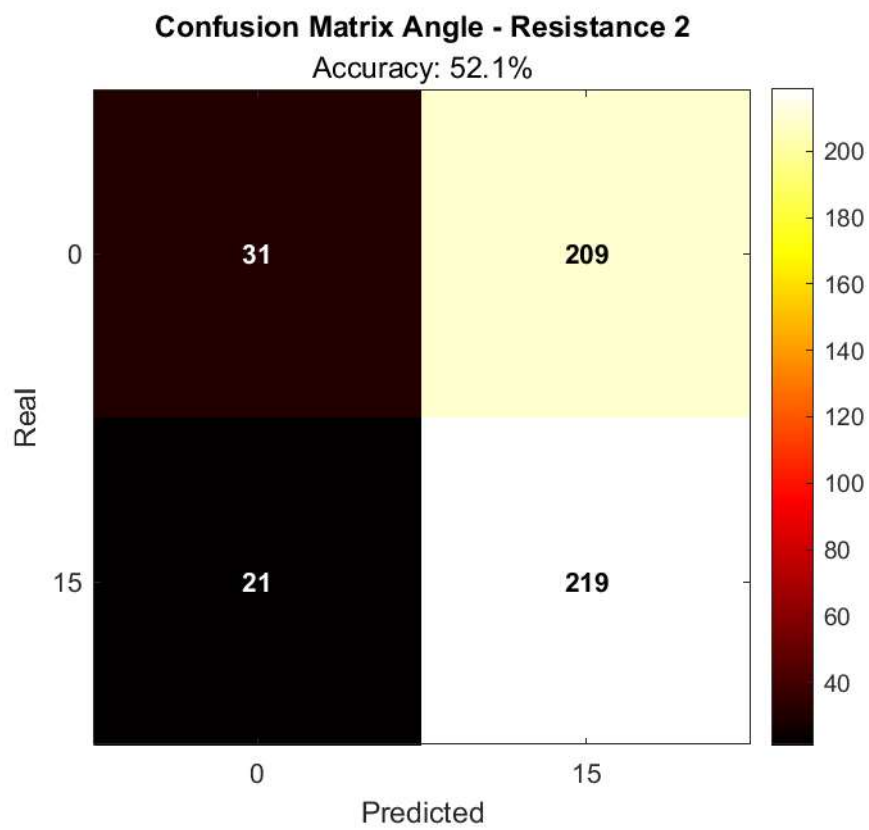


Figure 39: Confusion matrix for angle, resistance 2 – Broken branches unknown, load known.

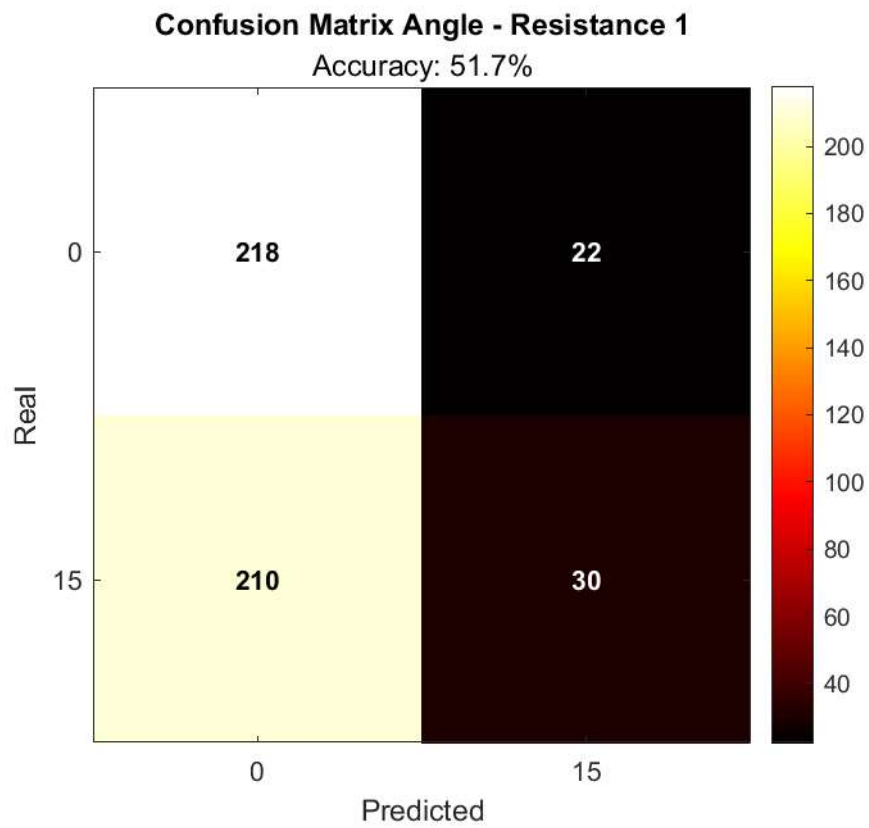


Figure 40: Confusion matrix for angle, resistance 1 – Broken branches unknown, load unknown.

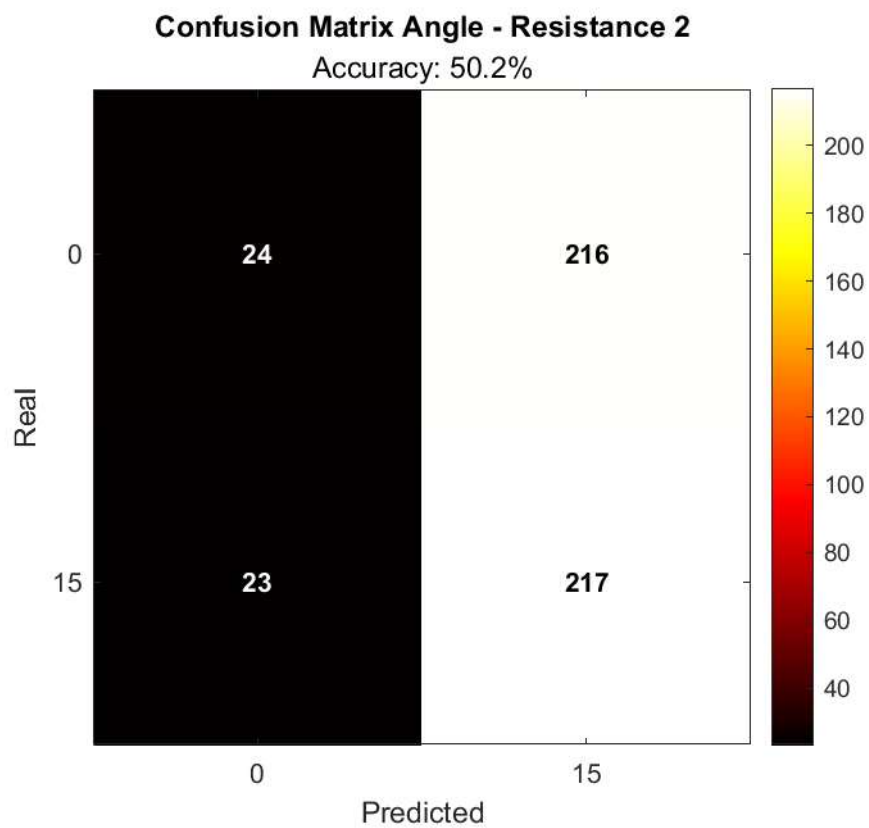


Figure 41: Confusion matrix for angle, resistance 2 – Broken branches unknown, load unknown.

Broken branches classification

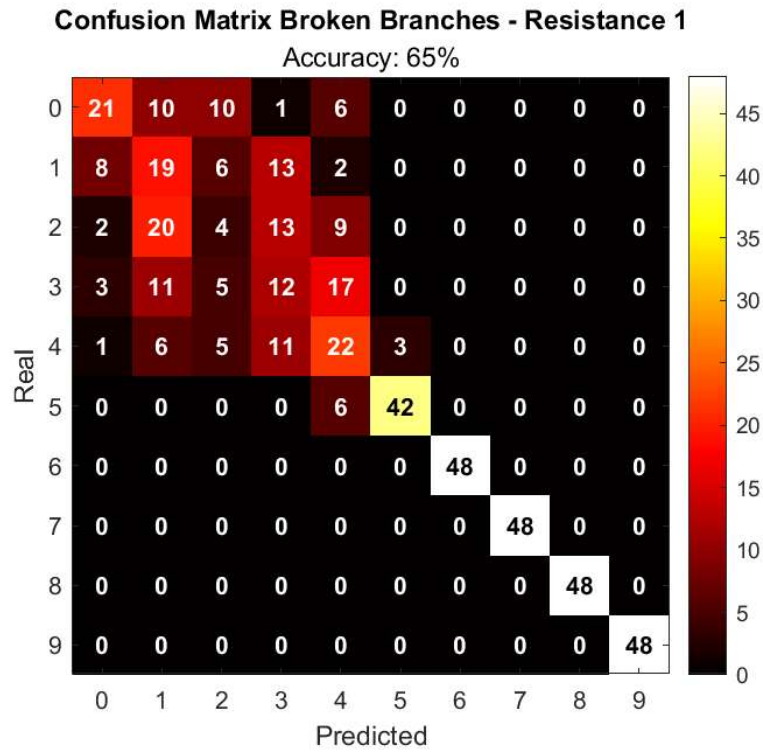


Figure 42: Confusion matrix for broken branches, resistance 1 – Angle known, load known.

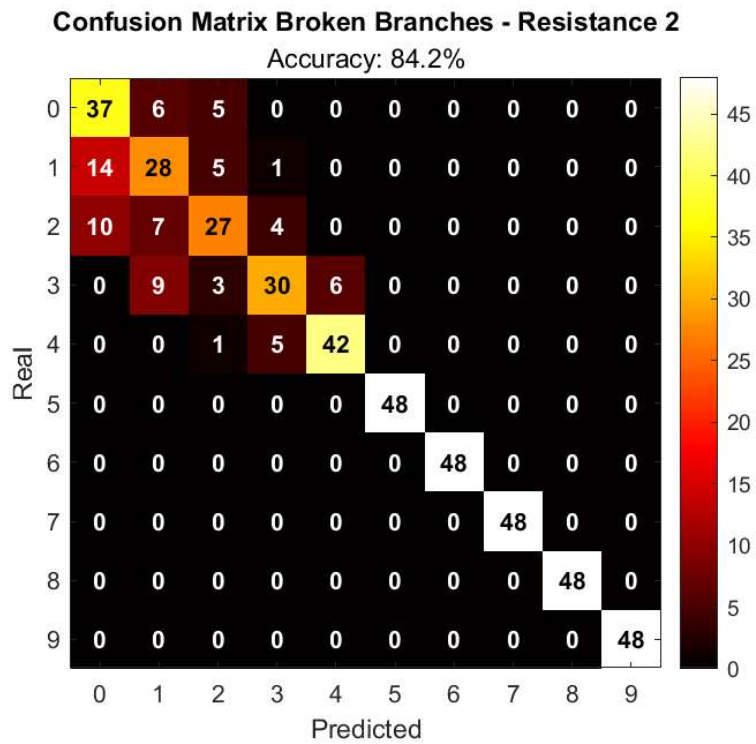


Figure 43: Confusion matrix for broken branches, resistance 2 – Angle known, load known.

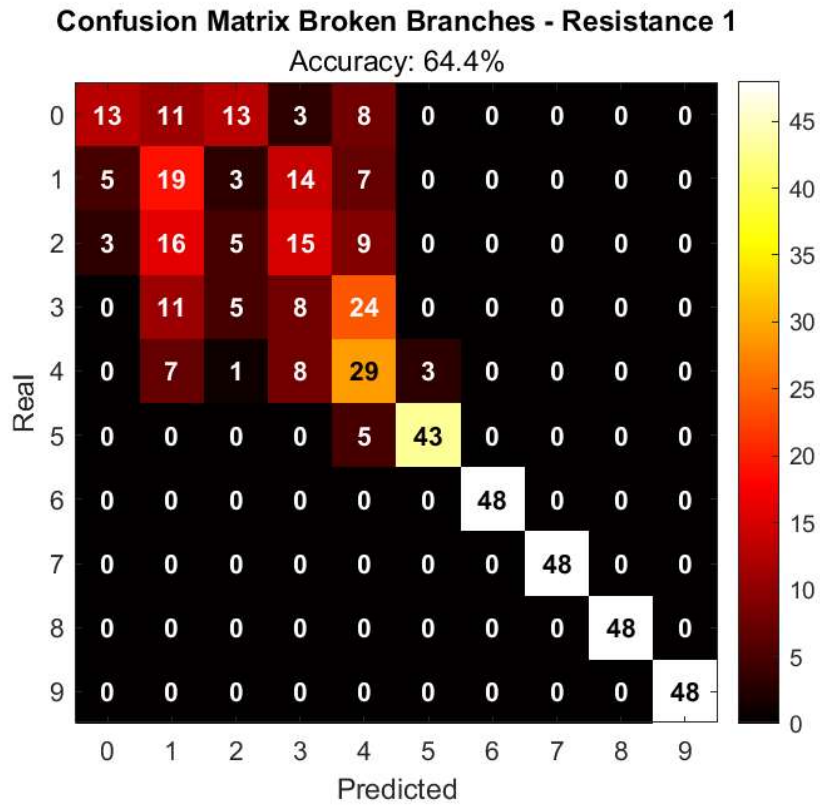


Figure 44: Confusion matrix for broken branches, resistance 1 – Angle known, load unknown.

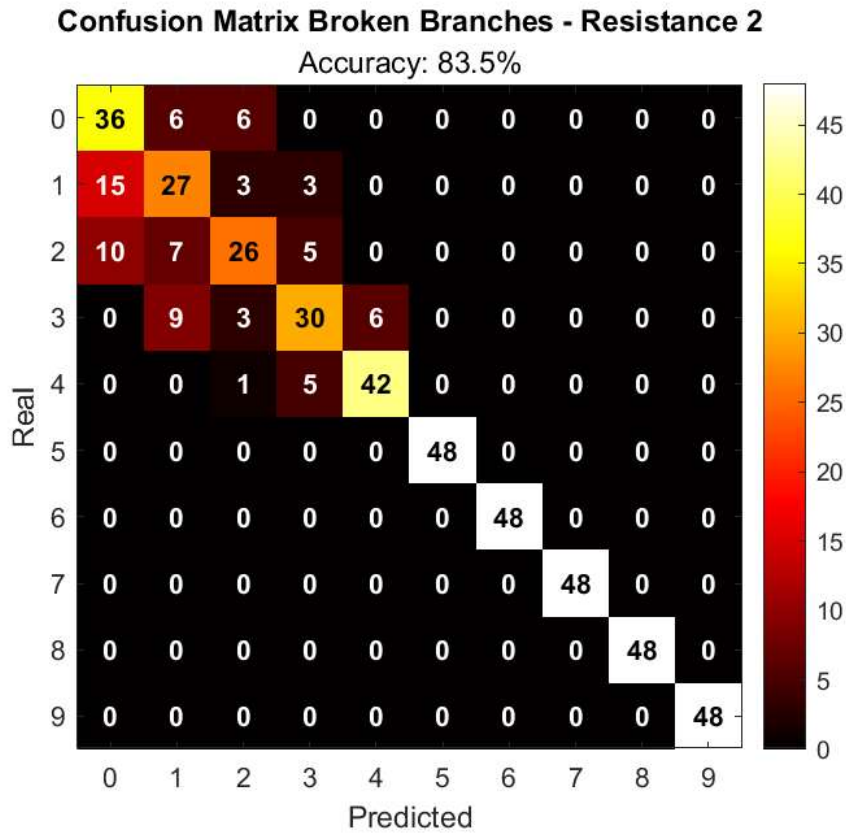


Figure 45: Confusion matrix for broken branches, resistance 2 – Angle known, load known.

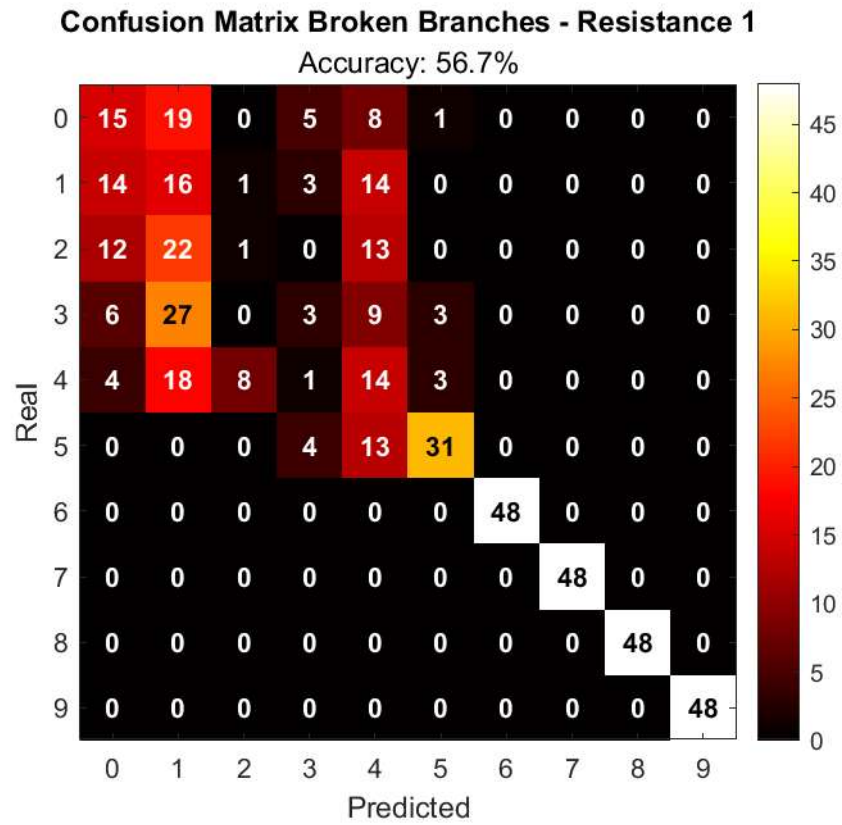


Figure 46: Confusion matrix for broken branches, resistance 1 – Angle unknown, load known.

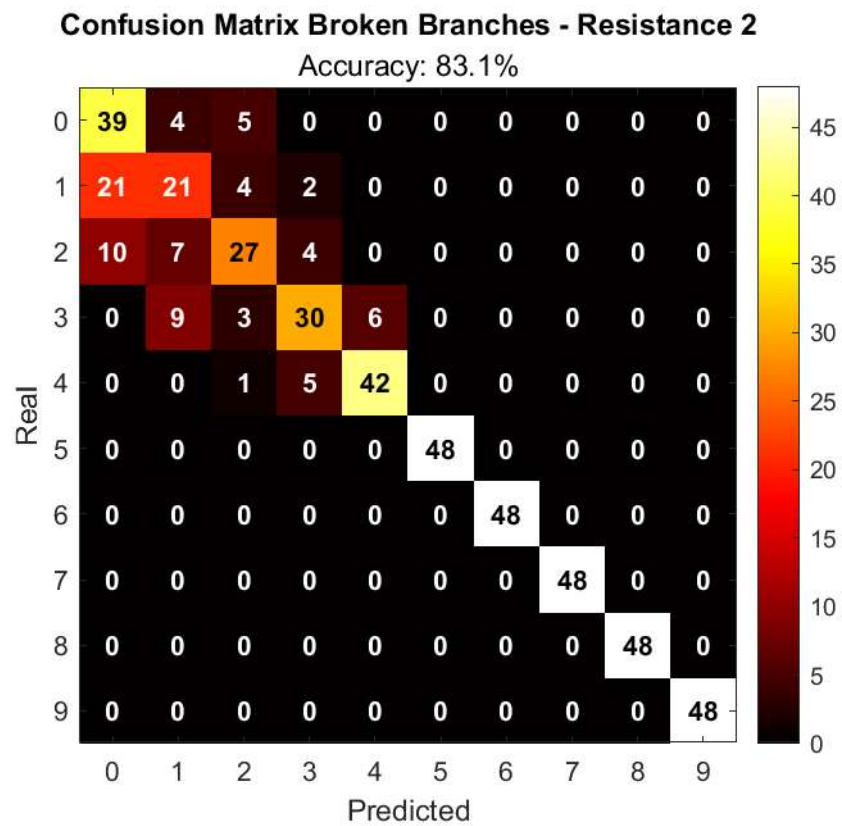


Figure 47: Confusion matrix for broken branches, resistance 2 – Angle unknown, load known.

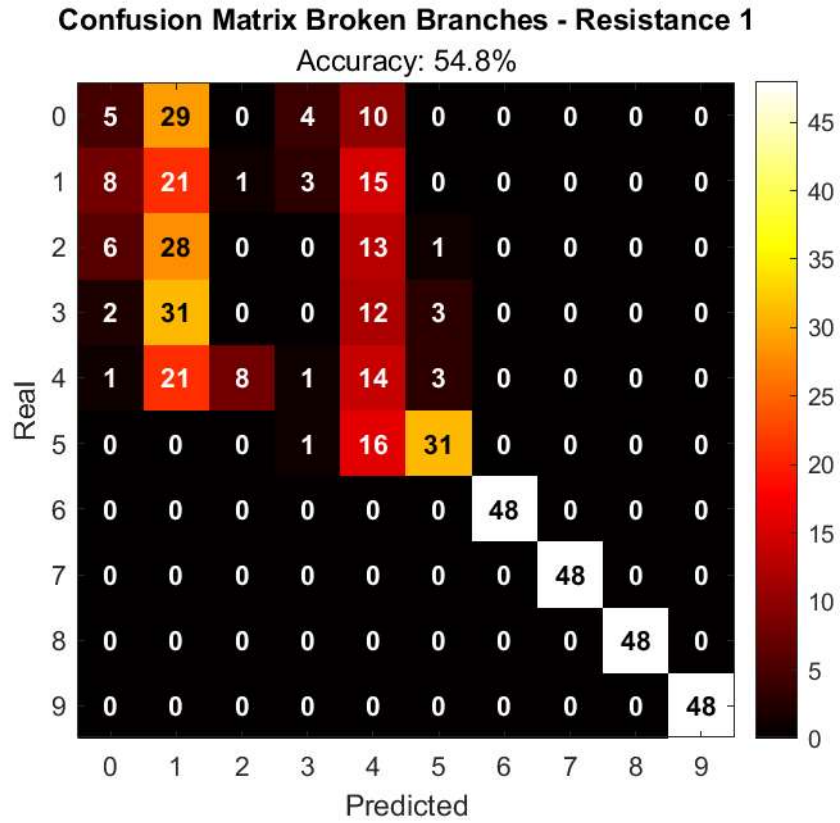


Figure 48: Confusion matrix for broken branches, resistance 1 – Angle unknown, load unknown.

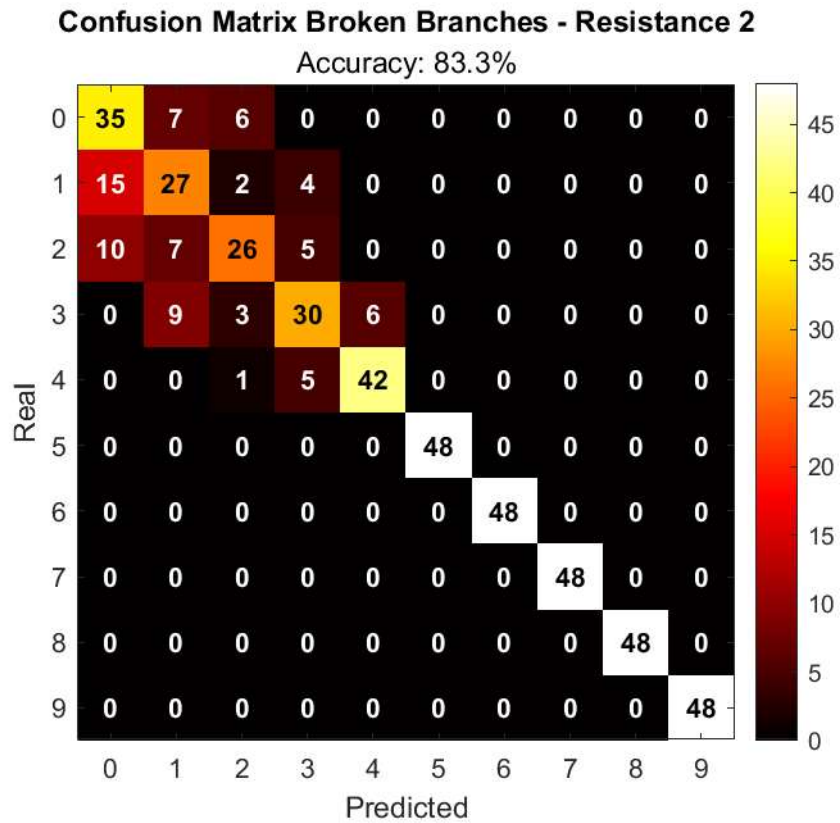


Figure 49: Confusion matrix for broken branches, resistance 2 – Angle unknown, load unknown.

Load classification

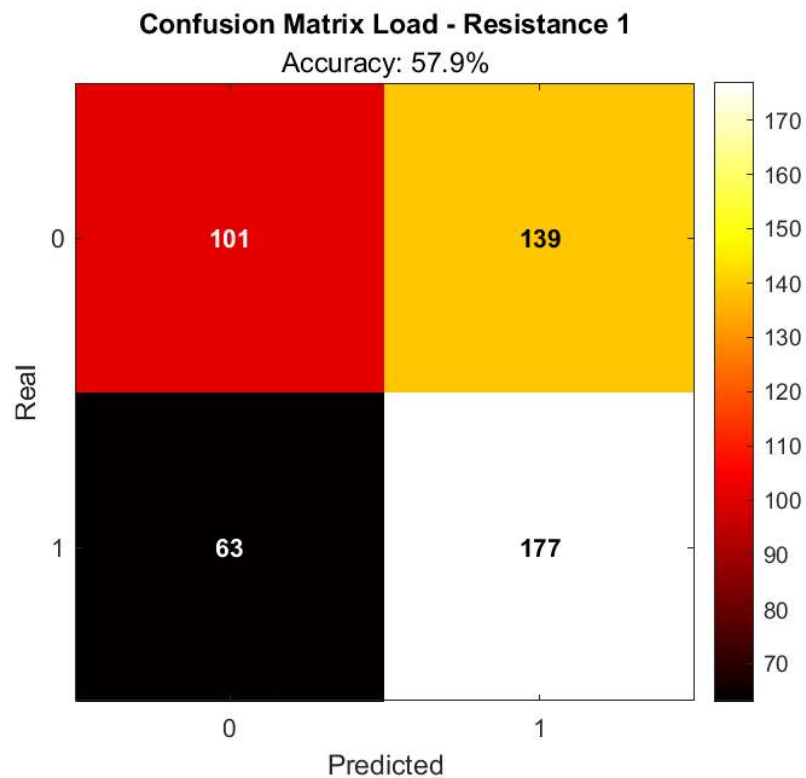


Figure 50: Confusion matrix for load, resistance 1 – Angle known, broken branches known.

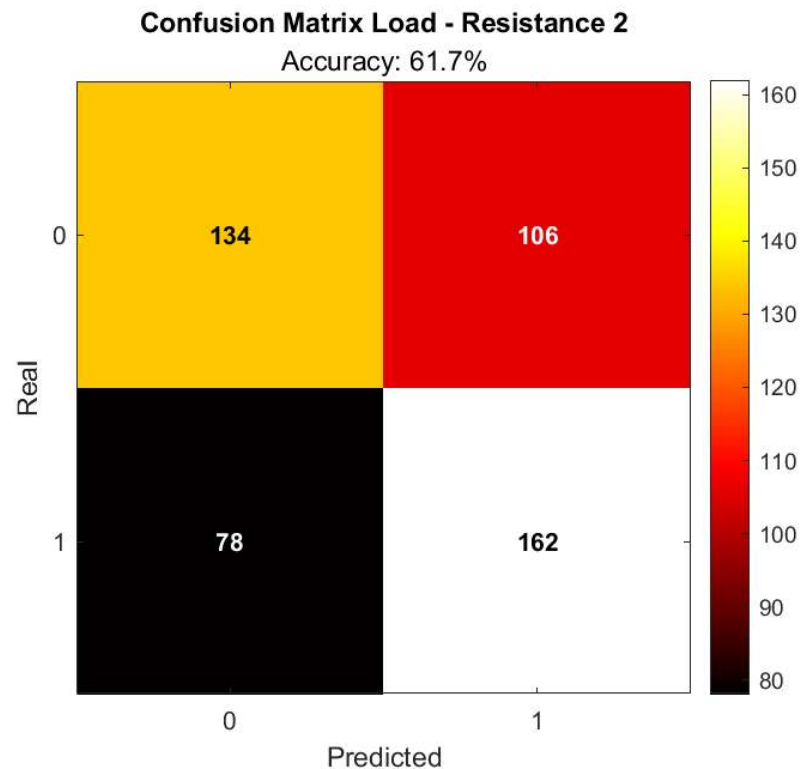


Figure 51: Confusion matrix for load, resistance 2 – Angle known, broken branches known.

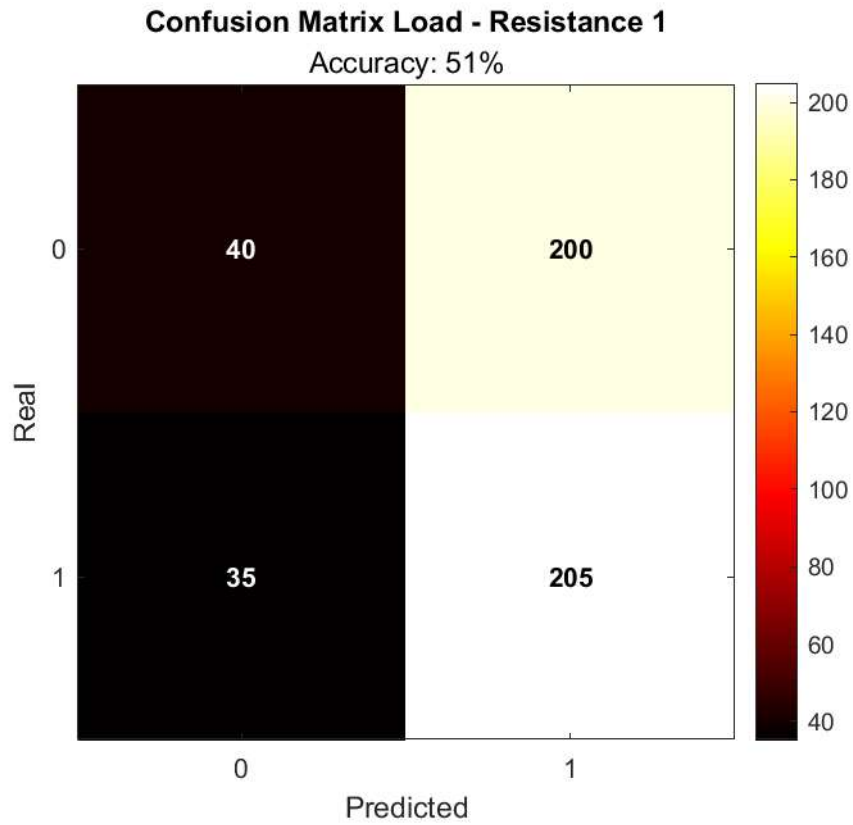


Figure 52: Confusion matrix for load, resistance 1 – Angle known, broken branches unknown.

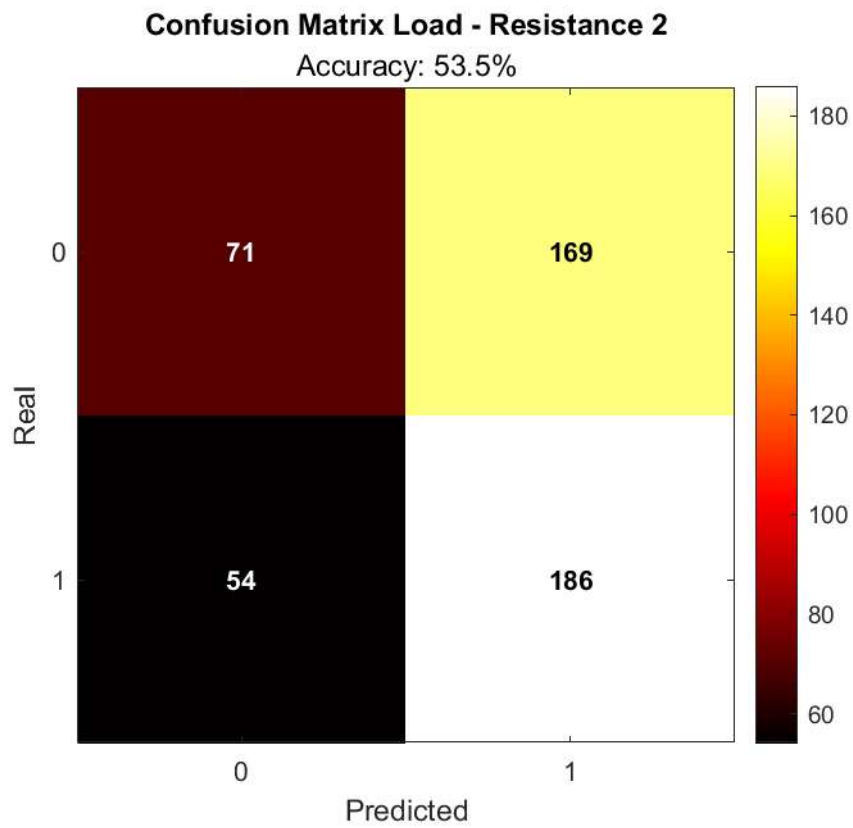


Figure 53: Confusion matrix for load, resistance 2 – Angle known, broken branches unknown.

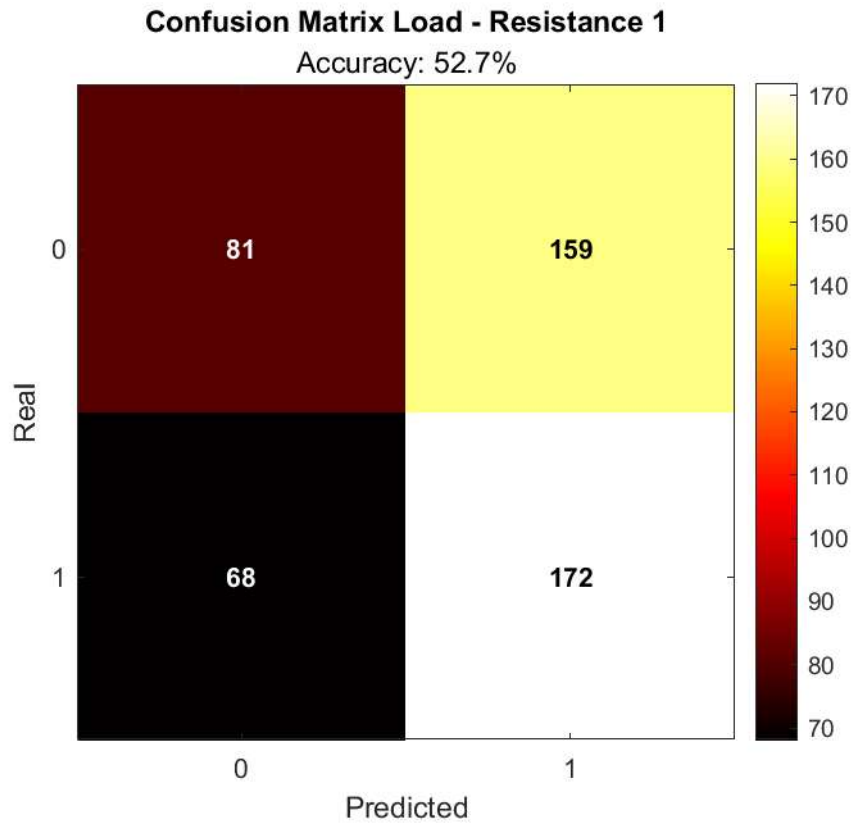


Figure 54: Confusion matrix for load, resistance 1 – Angle unknown, broken branches known.

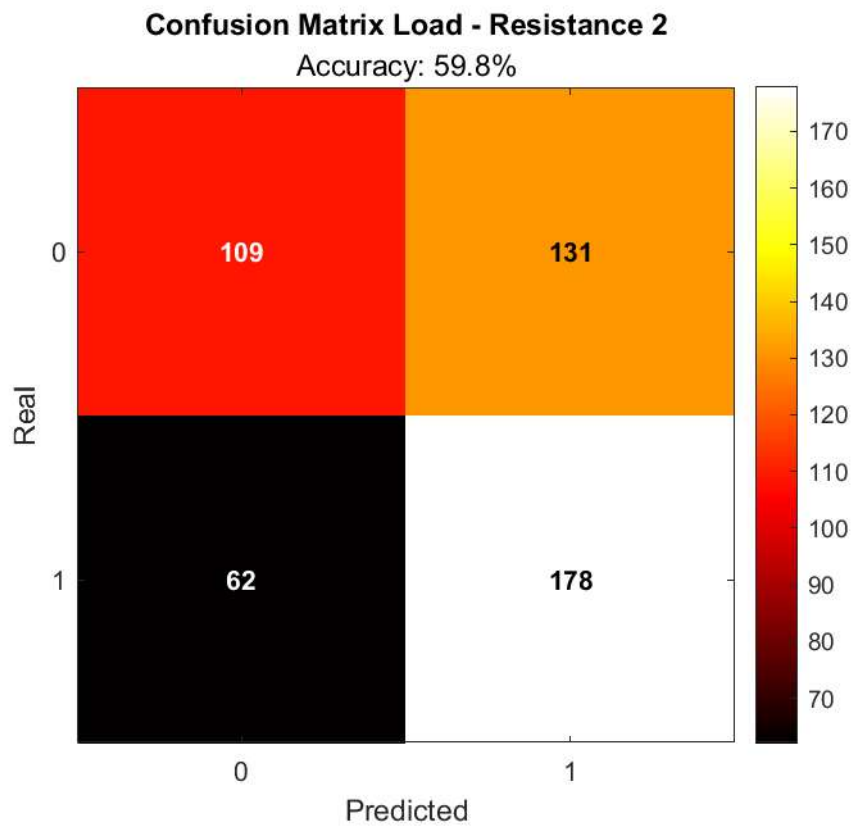


Figure 55: Confusion matrix for load, resistance 2 – Angle unknown, broken branches known.

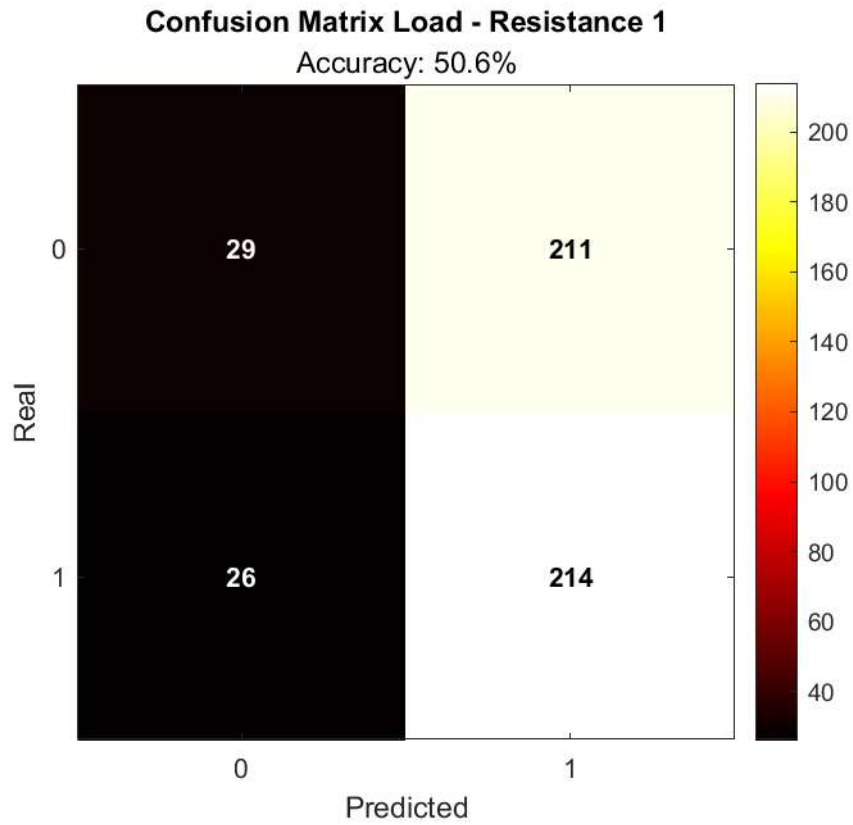


Figure 56: Confusion matrix for load, resistance 1 – Angle unknown, broken branches unknown.

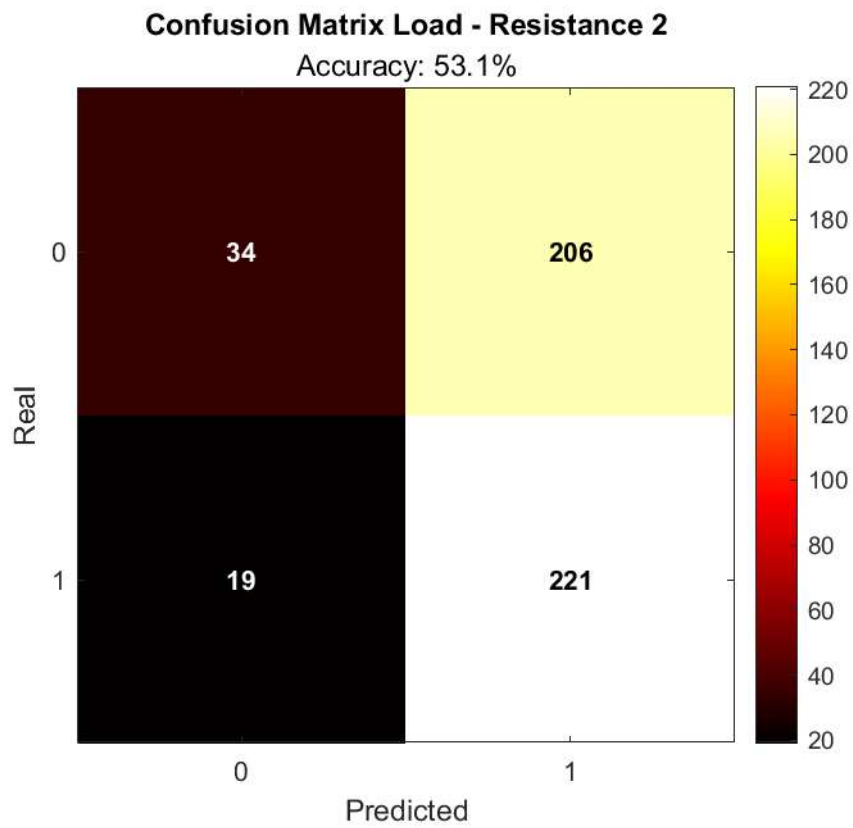


Figure 57: Confusion matrix for load, resistance 2 – Angle unknown, broken branches unknown.

In Table 6 are collected all the accuracy values for an overview.

Classified variable	Angle known?	Broken branches known?	Load known?	Resistance used	Accuracy [%]
Angle	-	Yes	Yes	1	79.2
				2	59.4
			No	1	77.5
				2	57.9
		No	Yes	1	53.5
				2	52.1
			No	1	51.7
				2	50.2
Broken branches	Yes	-	Yes	1	65.0
				2	84.2
			No	1	64.4
				2	83.5
	No		Yes	1	56.7
				2	83.1
			No	1	54.8
				2	83.3
Load	Yes	-	Yes	1	57.9
				2	61.7
			No	1	51.0
				2	53.5
	No		Yes	1	52.7
				2	59.8
			No	1	50.6
				2	53.1

Table 6: Accuracy of the first approach.

Reading the table and analysing at the matrices, it is possible to draw some conclusions.

In general, the algorithm classifies the number of broken branches better than the other variables, and the load worse than the other variables. This is clear if we look at the boundaries of each variable when the others are unknown (figures below): most regions related to the broken branches are well separated in the plane, while this is not true for the angle and the load. To understand why this happens, we have to think about the electric circuit that constitutes the sensor. Breaking a branch has a strong impact on the equivalent resistance in each instant of time, especially if the number of remained branches is small. This effect is evident in all the confusion matrices where the classified variable is the number of broken branches: the lower right part, referred to the highest number of broken branches, is always a perfect diagonal matrix. Instead, the stress angle has a lower effect on the equivalent resistance; for this reason, the accuracy for the classification of this variable is generally lower. Finally, the load has an effect just on the slope direction of the test, but this slope is small with respect to the total value of the resistance, so the accuracy is low.

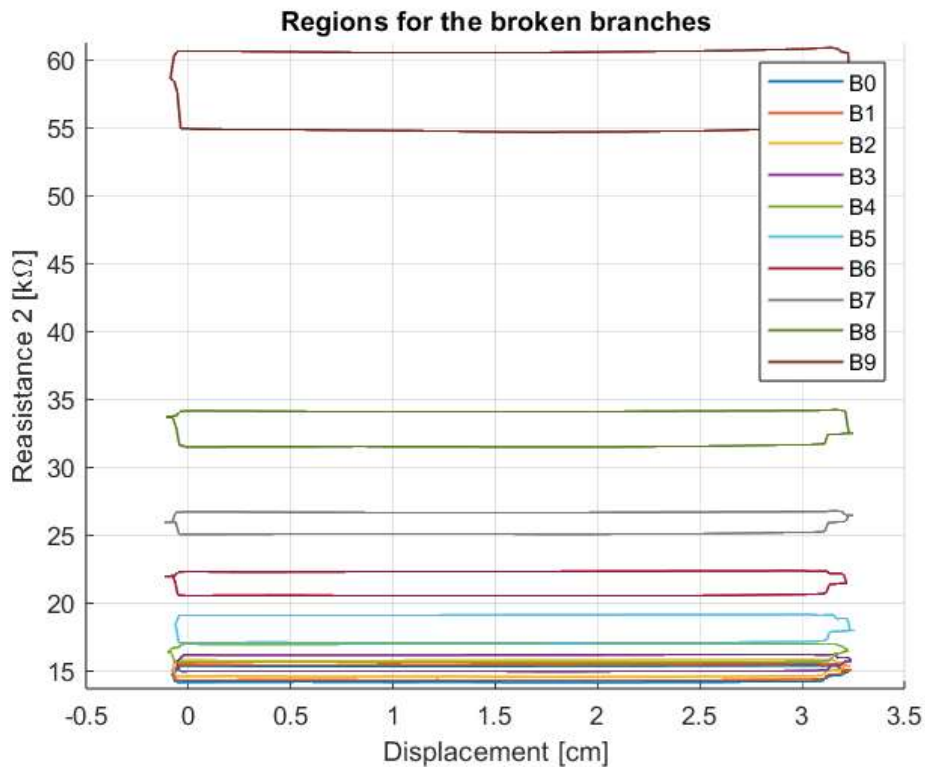


Figure 58: Regions of the plane based on broken branches; angle and load unknown.

Notation: “Bn” stays for the configuration with n broken branches.

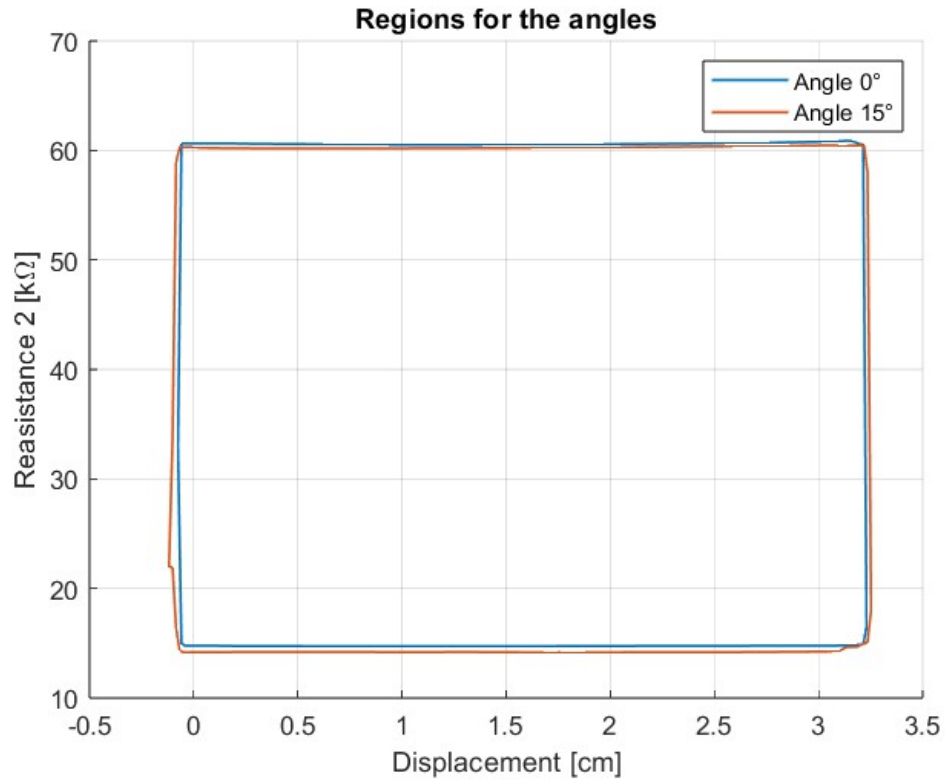


Figure 59: Regions of the plane based on angles; broken branches and load unknown.

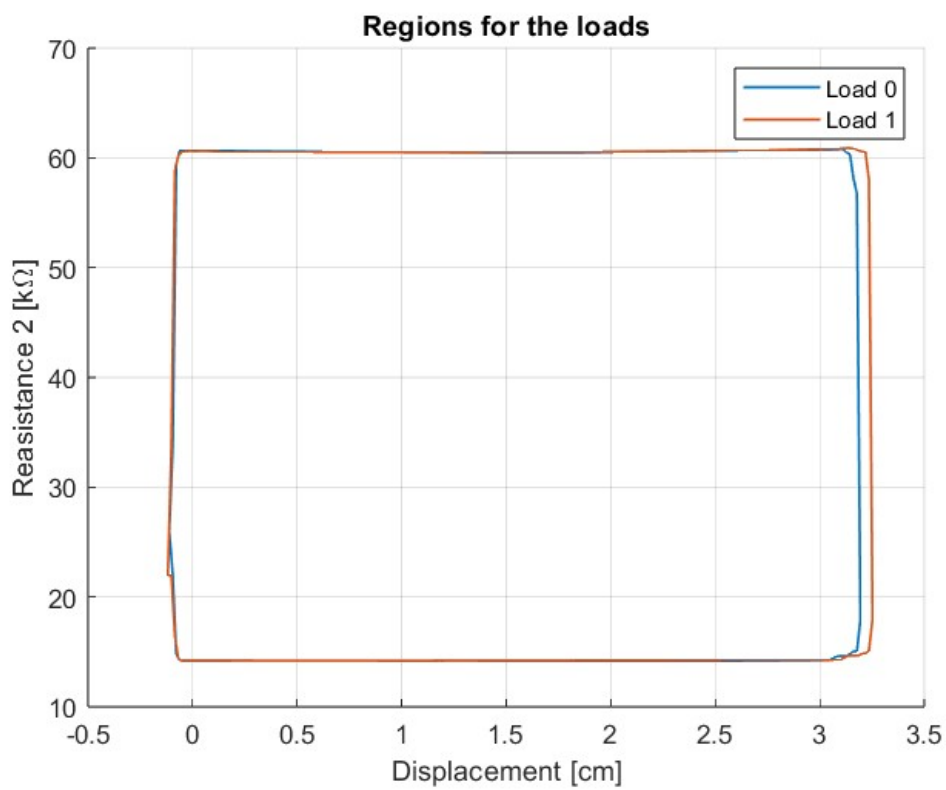


Figure 60: Regions of the plane based on load; angle and broken branches unknown.

It is also possible to make some considerations about some confusion matrices. Knowing that the number of broken branches is the variable with the strongest influence, it is clear that the matrices referred to the classification of angle or load when this variable is known are very different from the ones when it is unknown. This is because if the number of broken branches is unknown, the algorithm builds regions that are too large and they do not allow to appreciate the resistance variation given by a different stress angle. The opposite happens when this variable is known, as we can see in Figure 61 that shows the regions based on the angles when there is one broken branch and the load is unknown.

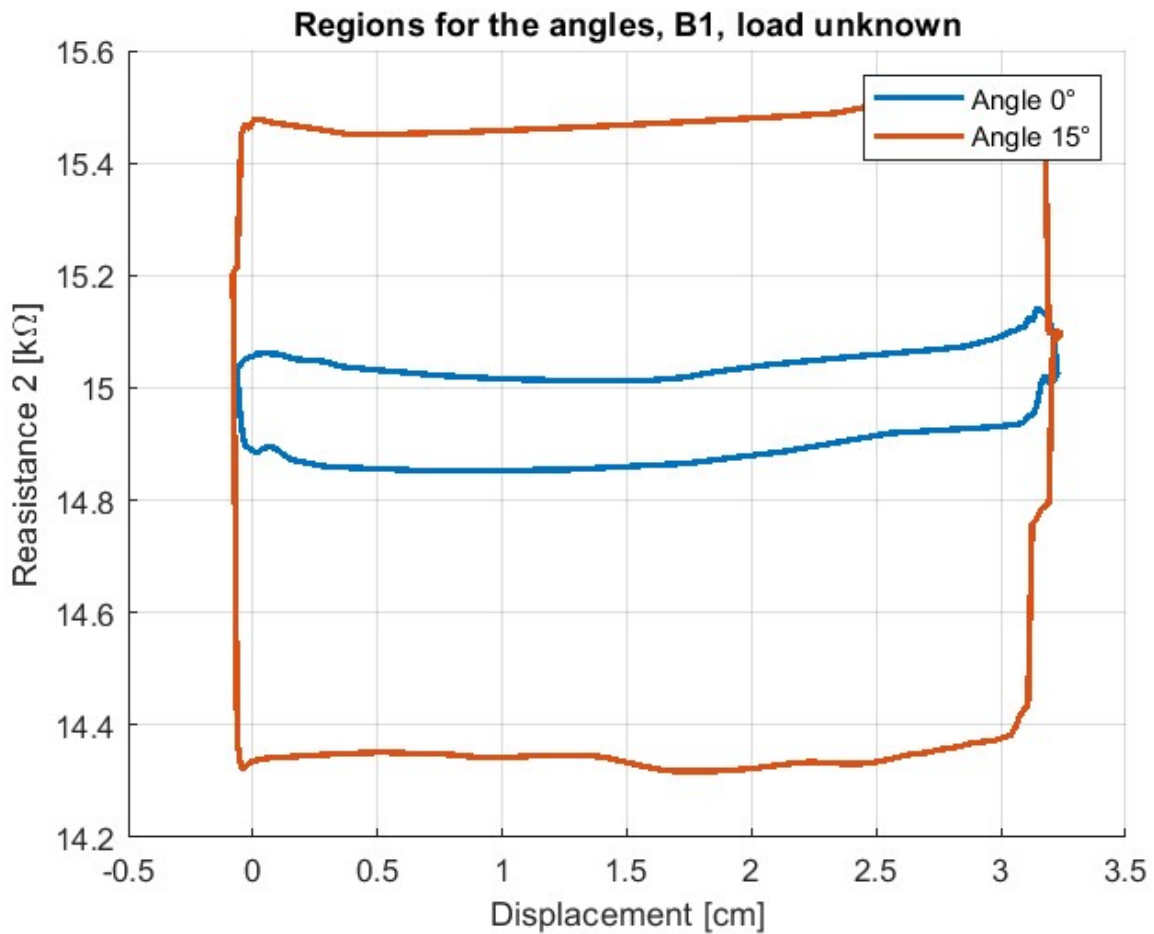


Figure 61: Regions of the plane based on angles; B1, load unknown.

This graph does not show only that the difference between the angle regions is appreciable only when the broken branches are known, but it allows to understand why the classification is still not good in this case (see the confusion matrices for the angle when the broken branches are known, from Figure

34 to Figure 37). We can clearly see that the variation of the stress angle does not imply an evident difference in the medium value of the equivalent resistance, but it just increases its range of values. Hence, when we give to the model a test whose stress angle is 0° , it lies in both the regions and it gets a high score in both, because it is close to the centre part of both the regions. As a result, the algorithm gives in output classified variable equal to 15° in many cases and the accuracy is lower than the one referred to the classification of the broken branches. Figure 62 shows the scores of these family as functions of the resistance, when the displacement is a half of its maximum. It clarifies that an extremely small range of resistance values gives a higher score for the 0° family.

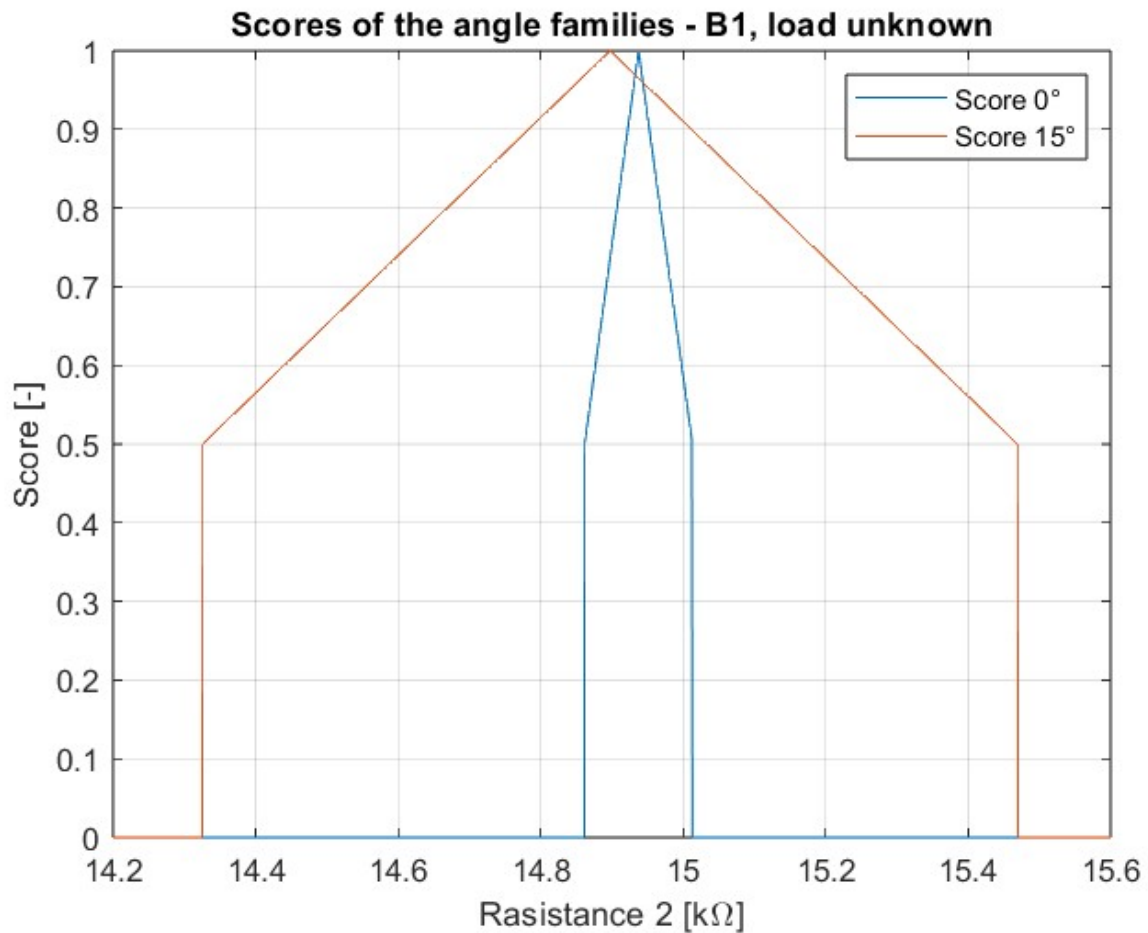


Figure 62: Comparison between scores of the angle families.

The same applies to the classification of the load. In particular, a different load implies a different slope for the resistance, but not a different range of values, as we can see in Figure 63. For this reason, this algorithm is not well designed for the classification of this variable.

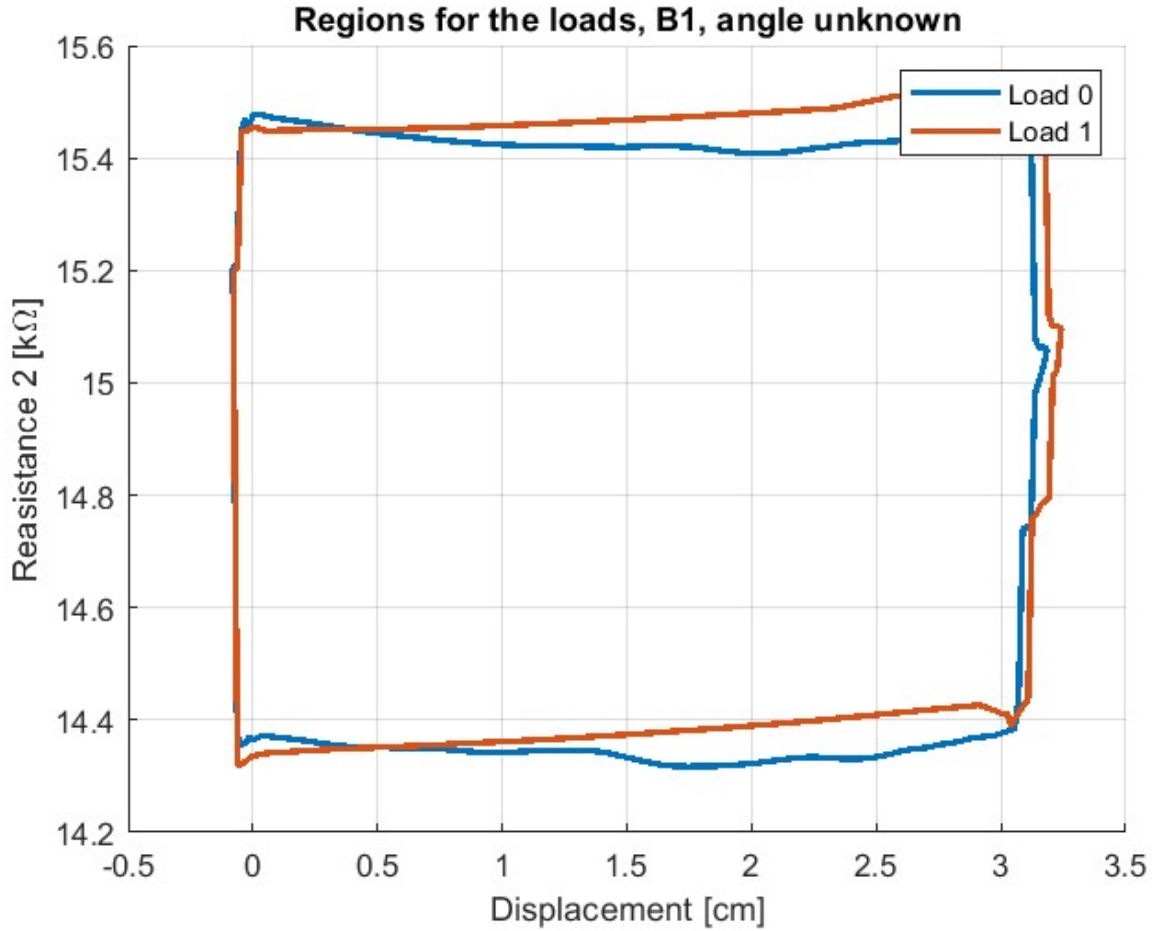


Figure 63: Regions of the plane based on load; B1, angle unknown.

In conclusion, using this algorithm for the classification of all the variables, one at the time, the best choice is to classify first the broken branches, then the stress angle and finally the load. This method allows to reach the following accuracies:

- accuracy for the broken branches (resistance 2): 83.3 %;
- accuracy for the stress angle (resistance 1): 77.5 %;
- accuracy for the load (resistance 2): 61.7 %.

4.3 Remarks

In the previous paragraph, we saw some quantitative results, but it is possible to evaluate the algorithm also through some qualitative features; in particular, using six evaluation parameters typically used for the Machine Learning algorithms. Since in these parameters there is also the accuracy, here it is resumed also if it was already discussed in the previous section. The six parameters are the following:

- Accuracy: quality of the classification;
- Interpretability: model interpretability and compactness;
- Robustness: reaction to noise and missing data;
- Incrementality: model update in presence of newly labelled records;
- Efficiency: model building time and classification time;
- Scalability: training set size and attribute number.

Here, the word “model” does not stay for the ideal electric model built and discussed in this thesis, but it refers to the boundaries and all other possible information known to the algorithm before the classification. The word “attribute”, instead, refers to the kind of variables used for the classification like resistances and displacement.

Accuracy

There are different values of accuracy depending on the variable to classify: they assume values between 60 % and 85 %. The accuracy is satisfactory for the classification of the broken branches, is medium for the angle, and is low for the load.

Interpretability

The model is highly interpretable; as we have seen, all the calculations that lead to a certain classification can be clearly shown and the result is 100 % explainable. It is also compact: all data is contained in a table whose size is modest compared to other common classification algorithms.

Robustness

This a key feature for evaluating this model: it represents its weak point. Indeed, when the boundaries of a region are built, an outlier test is enough to lose all the information about one of the boundaries making it unreliable and non-representative of the main part of the family. The missing data could be

an issue, too. For example, in the alternative dataset the problems encountered during the experimental tests with 4 active branches resulted in a lack of information for stress angles greater than 30° . Hence, it is impossible to distinguish if a test is referred to that set of variables.

Incrementality

The incrementality of the model is medium, because every newly labelled record could lead to different results that are difficult to evaluate:

- it could be included within all the possible regions it belongs to. In this case, none of the family borders would change and the new information is unused;
- it could influence some family borders. In this case, if the region enlargement is representative of the reality, the incrementality is well spent; otherwise, it makes the model less reliable.

Anyway, the model can be easily updated with new data, but it is not automatic and a human intervention is needed to update the table containing the family borders.

Efficiency

The efficiency of the model is excellent: both building the boundaries (model building) and the classification require about a second using a common computer.

Scalability

The model has a good scalability. It does not require a large amount of experimental data compared to other common classification algorithms and the attributes are only six: displacement, resistances 1 and 2, stress angle, broken branches and load.

All in all, the first approach gives some good results, but it has also some critical aspects. They are here resumed.

Strengths

- The **classification** is **reliable**, especially for the number of broken branches, that is the most interesting variable for what concerns the state of health of the component.
- The **training requires a small number of records**, compared with other typical Machine Learning algorithms.

- Both the training and the classification require a **small amount of time**.

Weaknesses

- The **score** for the classification is **arbitrary**. This is not necessarily a defect, but it can involve a loss of generality for the algorithm. If we search the best arbitrary score for this single dataset, it can improve the performance in this case, but it introduces a sort of overfitting that does not allow to apply this model to other datasets.
- The algorithm is **highly sensitive to outliers** (see the part about robustness).
- **Increasing the dataset may not refine the model** (see the part about incrementality). Once again, it could be seen as an advantage, because the algorithm does not require a large amount of records for a good classification. But it is also a limit: once a certain accuracy has been reached, it is almost impossible to improve it.

The second approach is designed to fix the weaknesses of the first approach maintaining its strengths.

5 Second Approach: Medians

5.1 Algorithm Explanation

The second approach is based on the first one, but it is designed to fix the issues that could income using it. Hence, the main idea is to substitute the boundaries of each region with a median of the family and compute the distance of each test point from it instead of using the arbitrary score. Therefore, all the weaknesses of the first approach could be improved.

- The score is no more arbitrary, because it is based on the geometric distance between points.
- The sensitivity to outliers is highly decreased thanks to the use of the median.
- Increasing the dataset could not change the medians of the family and so it could not refine the model, but it gives robustness to the value of the computed median, confirming its value or changing it if a large number of records tend to move it in the same direction.

5.1.1 Training

The dataset used is the Crack Propagation dataset (see paragraph 3.3.1). It is managed the same way as the first approach (see paragraph 4.1.1), obtaining a training table and a test table with a set of tests for each possible combination of stress angle, ID and load.

The training also has the same structure, with a few differences. It follows the steps below.

- Choose which one of the two measured resistances will be used for the classification. The test could be repeated for the other resistance, but only one resistance at the time can be used.
- Build a family set for each combination of stress angle, ID and load: only the records contained in the training table can be used in this step.
- For each family, compute the median curve of the resistance with respect to the displacement. Each test has its own values of displacement between the two configurations: loaded and unloaded. We need a median that can be used for new experimental tests: a curve with a proper number of points. So, to compute it, for each family we search the maximum and the minimum values of displacement measured in all the records, and we build an array with 200 equally spaced points between them. Then, we interpolate the resistance value for each record using these points and we can finally obtain the median value of the resistance with respect to the displacement. Every median curve computed in this way is stored in a table.

See the Matlab scripts

Build_medians_table.m and *median_func.m* in the appendix for the details.

5.1.2 Test

- Define which variables are known and which variable we want to classify. Like in the first approach, only one variable at the time can be classified.
- Reorganize the families in this way:
 - if a variable is assumed to be known, discard all the families with a different value for that variable;
 - if a variable we do not want to classify is assumed to be unknown, group all the families with the same value for all the other variables and compute the mean of all their medians. This part of the step has to be better explained. The ideal way for building a median of all the families grouped is to sort all the records contained in the family sets and compute the total median. But in this way the model, instead of being a table containing only the computed medians, would consist in all the training records labelled by angle, broken branches and load. Hence, the median calculation would be repeated for each test, basing on which variables are known or unknown. Since this method would highly increase the computational cost, a Machine Learning thinking leads to build the medians of the grouped families as a mean of all the medians. In this way, the records can be discarded after the medians have been computed and the number of calculations decreases. Moreover, thanks to the fact that each family has the same number of records, the real median value and the mean of the medians are sufficiently close. This is evident in Figure 64, where are shown the results for grouping the families with ID9 and all the different angles (0° and 15°) and loads (0 and 1). On the displacement-resistance plane we can see all the tests of all the families, the total median and the mean of the four medians related to the four families. See the Matlab function *mean_medians.m* in the appendix for the details.

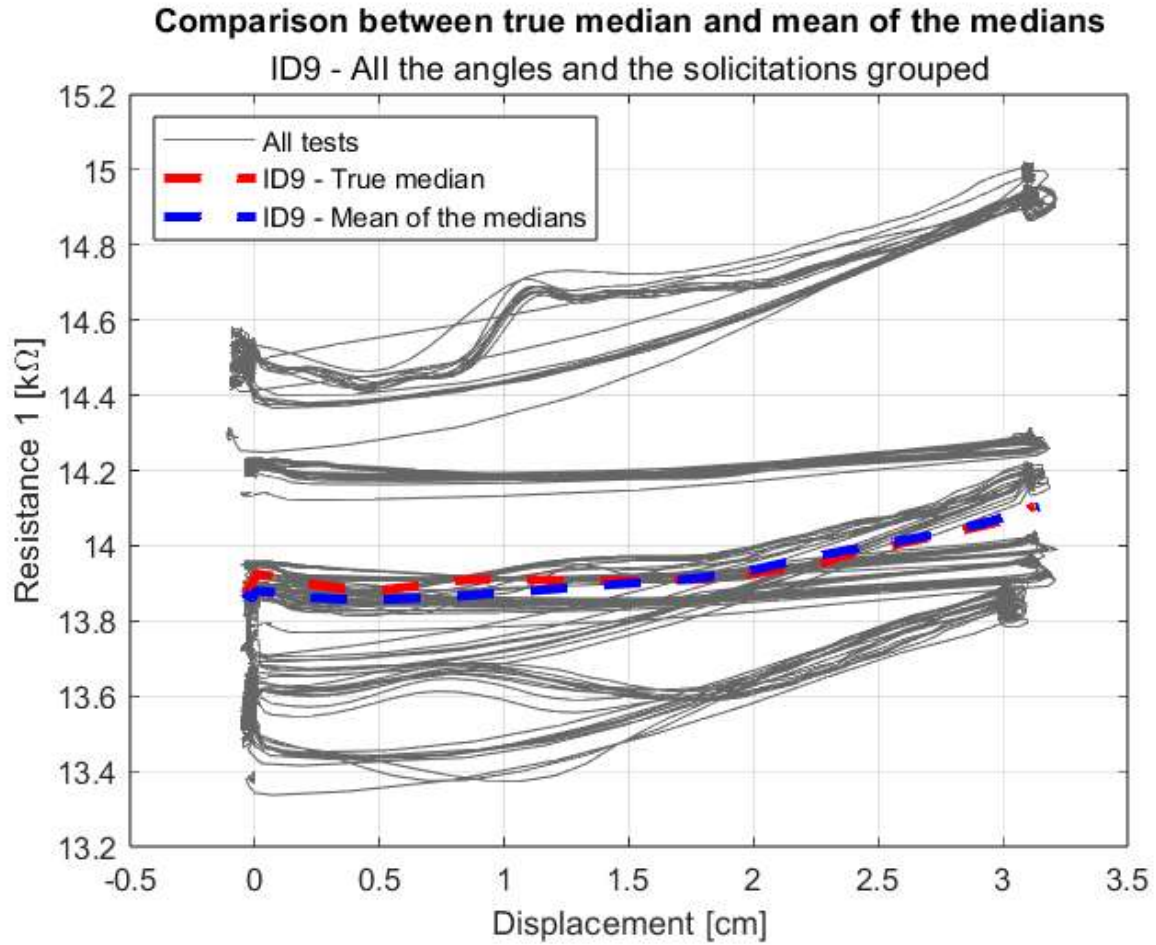


Figure 64: Comparison between true median and mean of the medians.

- Once the median curves have been built, we can use the test profiles to classify the chosen variable. Given the displacement-resistance profile of the test, we compute a score to estimate the probability of belonging to each family. The record is reorganized in 200 points equally spaced along the displacement axis, using interpolation for the values of the resistance. The score of each family is the quadratic norm of the difference between the two arrays of the resistance values: the test and the family median.

Even here, the rigorous way to compare the vectors would be exactly for the same values of displacement. But again, since the minimum and the maximum values of the displacement are fixed in this dataset (unless small measurement errors), thanks to the high number of points, the values of the displacement are roughly the same, and this procedure let us to reduce the computational cost.

- The family with the lowest score is the chosen family.

5.1.3 Example

Here is an example that shows on a graph how the algorithm works. We want to classify the number of broken branches of a test assuming that we do not know neither the stress angle nor the load phase. We arbitrarily choose to use resistance 2 for the classification. We collect the displacement-resistance profiles for each family; we group the families with the same number of broken branches and different angle and load; we compute the mean of the medians for all the families, and we obtain the curves in the graph (see Figure 65). Here we can see a particular aspect of this approach. In order not to lose information about the ID referred to the number of broken branches, we maintain as separated families the couples of IDs referred to the same number of broken branches (i.e. those whose sum is 21). This procedure has a good effect on the performance. The algorithm searches the most similar ID median and then it derives the number of broken branches. Thus, we do not compute the mean of two medians, avoiding the risk of confusing two close families (this could happen for low values of broken branches because there is a smaller resistance difference). The dot line shows the experimental points of the test. The IDs between 7 and 14 (6 to 9 broken branches) are not shown because they are too far from the test points. IDs with the same colour correspond to the same number of broken branches.

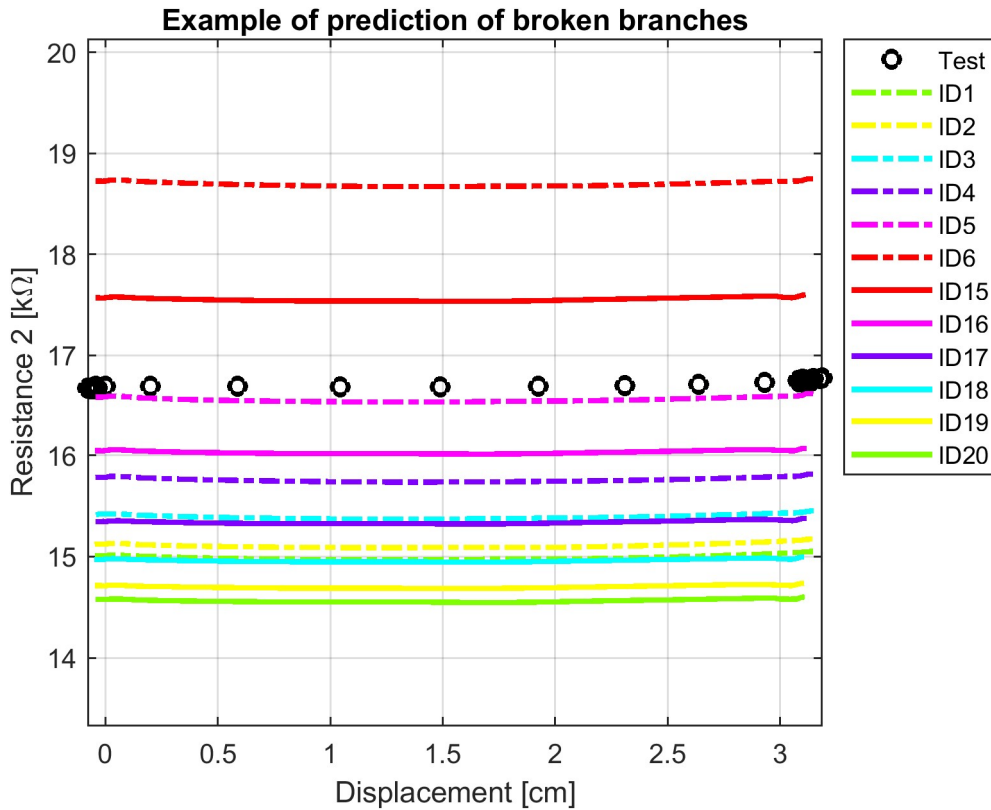


Figure 65: Broken branches classification without knowing the other variables.

The test has a stress angle of 15° , with 4 broken branches (ID5), during the loading phase. The family scores computed by the algorithm are the following:

ID1: 585.6	ID8: 19144.8	ID15: 145.6
ID2: 507.6	ID9: 57097.7	ID16: 89.8
ID3: 342.4	ID10: 317229.4	ID17: 370.9
ID4: 177.8	ID11: 367892.2	ID18: 605.1
ID5: 4	ID12: 49068.1	ID19: 800
ID6: 793.7	ID13: 16676	ID20: 914.1
ID7: 5614.3	ID14: 3919.9	

As we can see, the family with the lowest score is the one related to 4 broken branches (ID5). Looking at the median curves of the same colour, it can be easier to see the risk of losing information computing the mean of two IDs related to the same number of broken branches.

5.2 Results

As for the first approach, all the 24 possible confusion matrices have been built. See the Matlab script *Build_confusion_matrix_broken_branches_medians.m* in the appendix for the details.

Stress angle classification

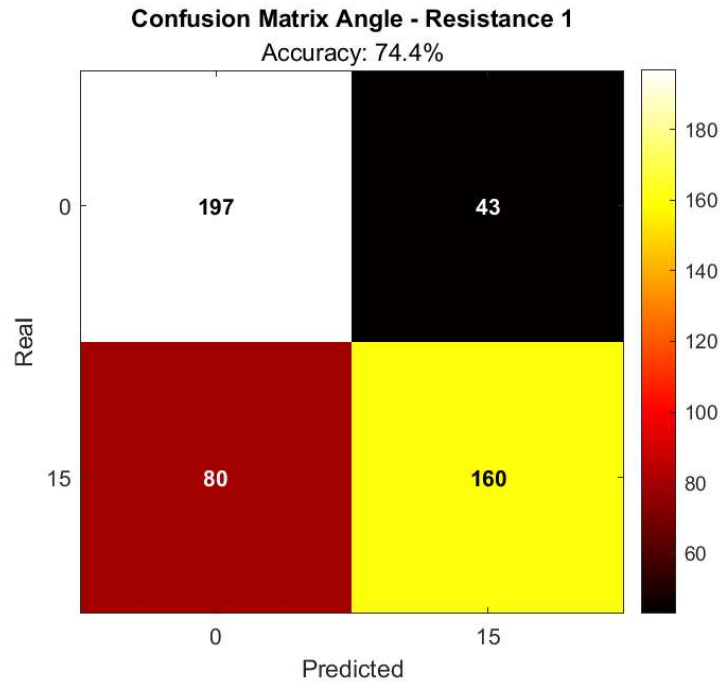


Figure 66: Confusion matrix for angle, resistance 1 – Broken branches known, load known.

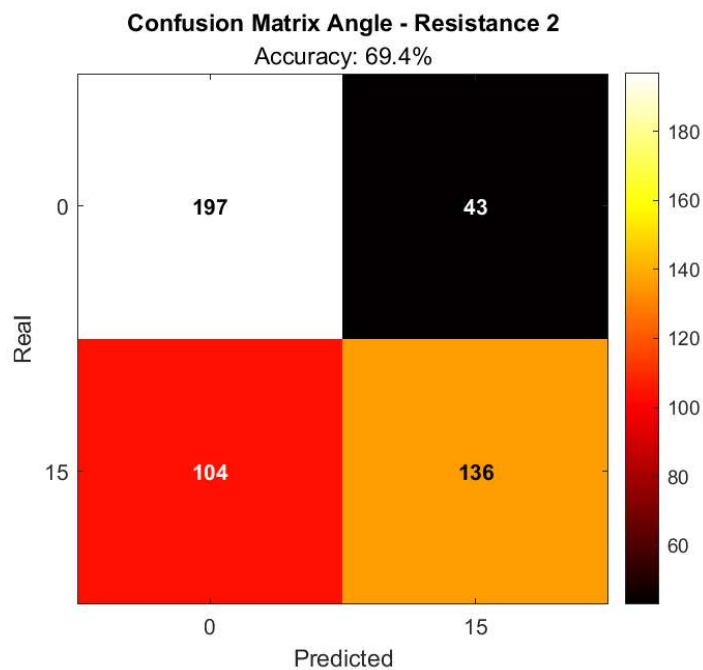


Figure 67: Confusion matrix for angle, resistance 2 – Broken branches known, load known.

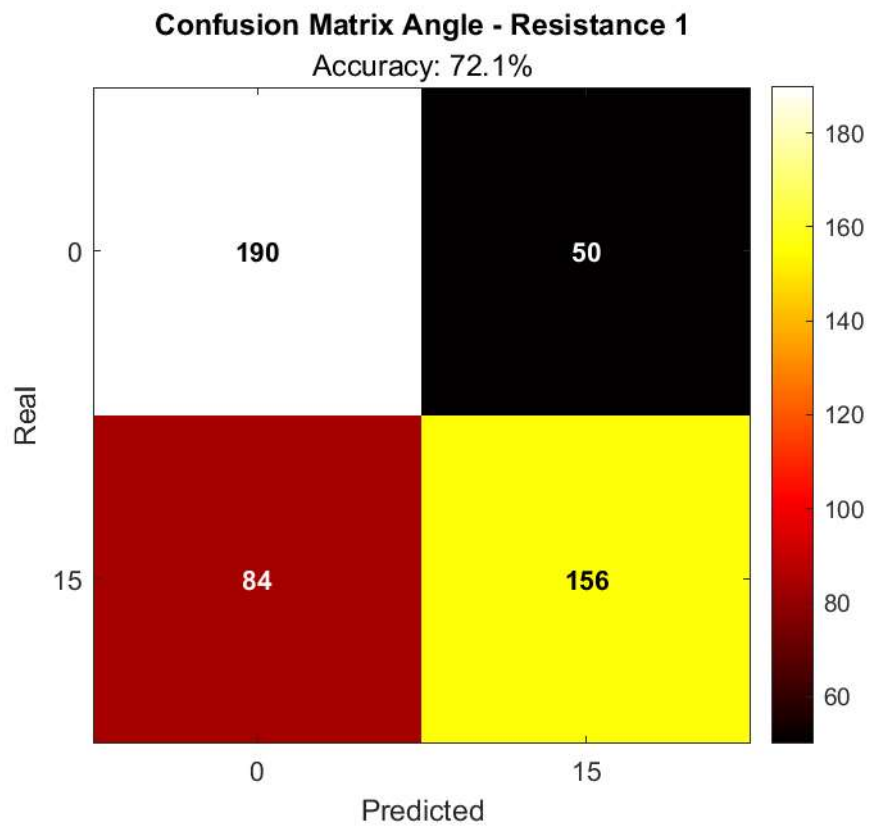


Figure 68: Confusion matrix for angle, resistance 1 – Broken branches known, load unknown.

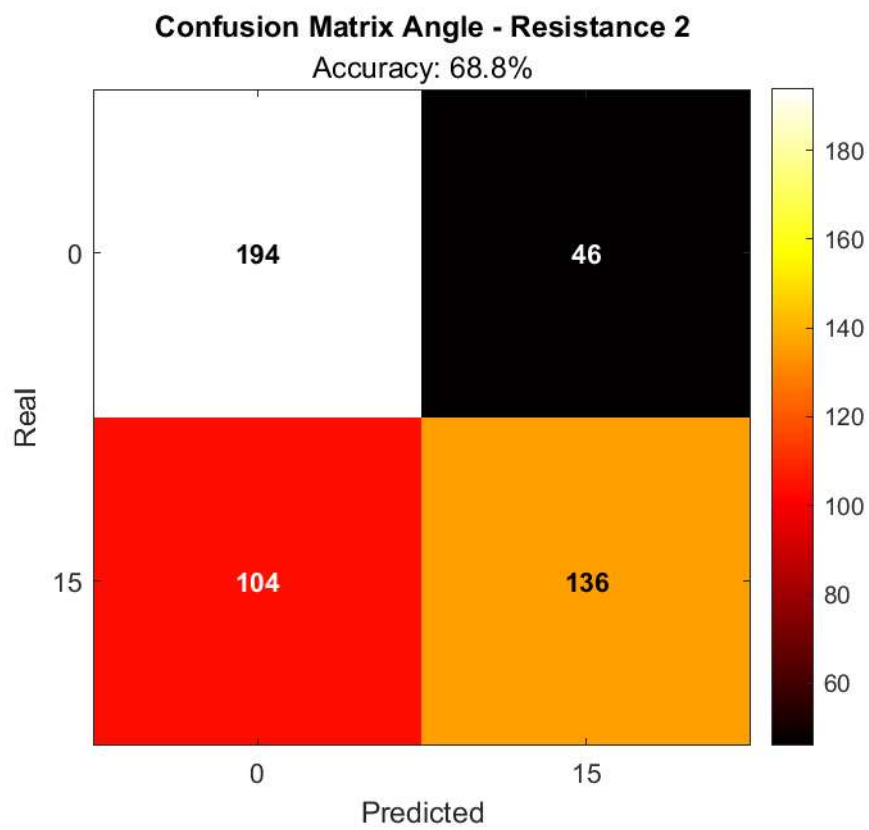


Figure 69: Confusion matrix for angle, resistance 2 – Broken branches known, load unknown.

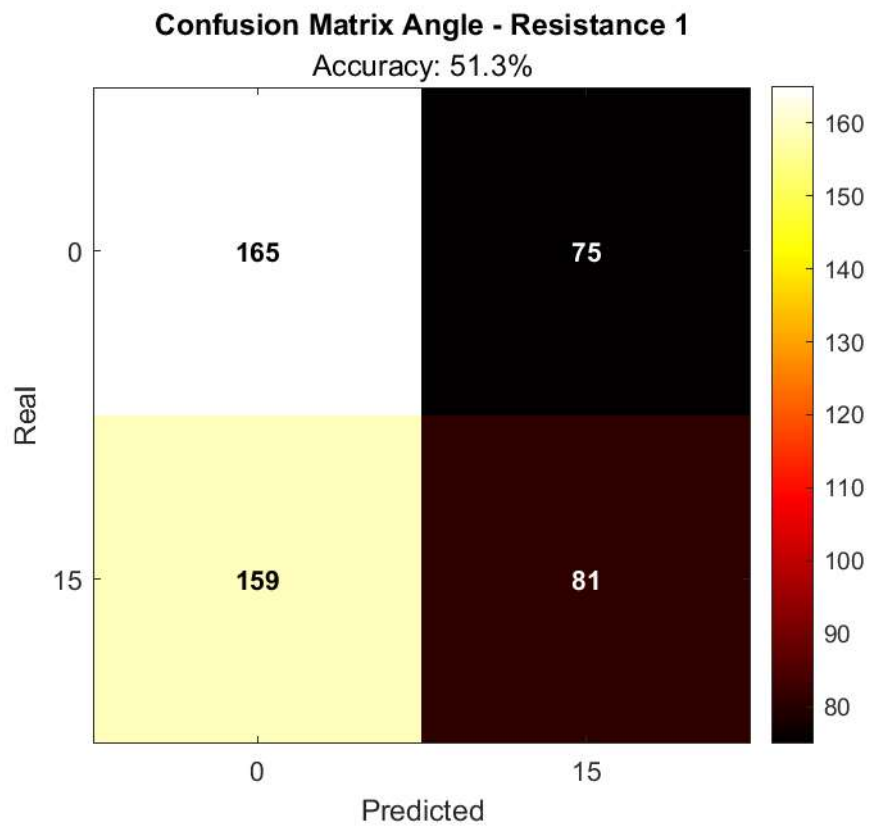


Figure 70: Confusion matrix for angle, resistance 1 – Broken branches unknown, load known.

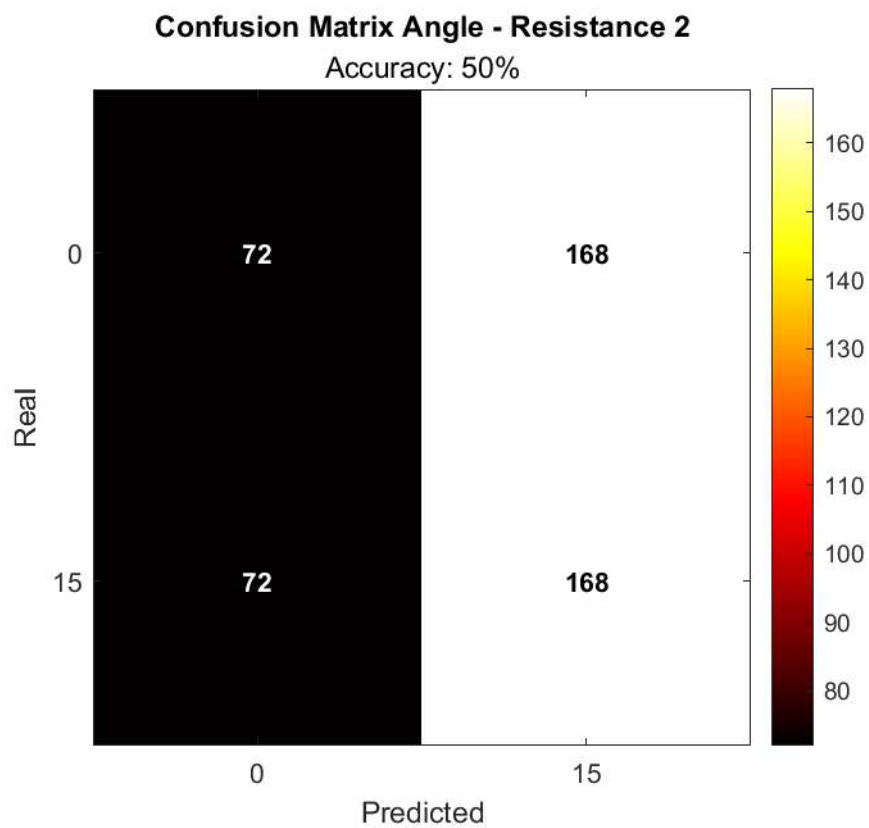


Figure 71: Confusion matrix for angle, resistance 2 – Broken branches unknown, load known.

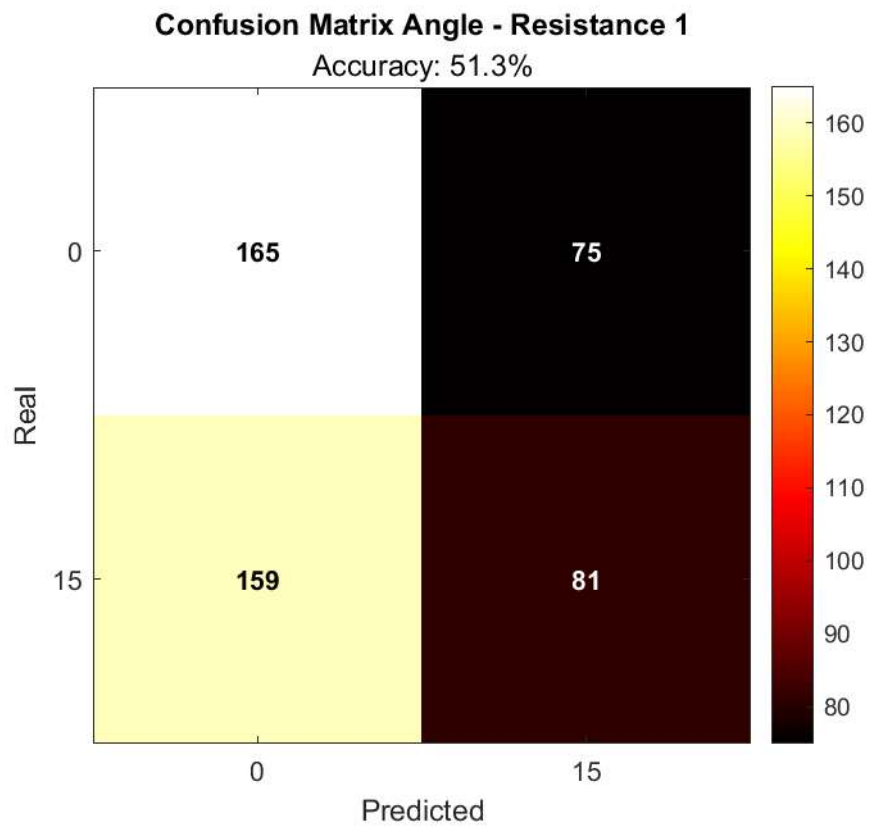


Figure 72: Confusion matrix for angle, resistance 1 – Broken branches unknown, load unknown.

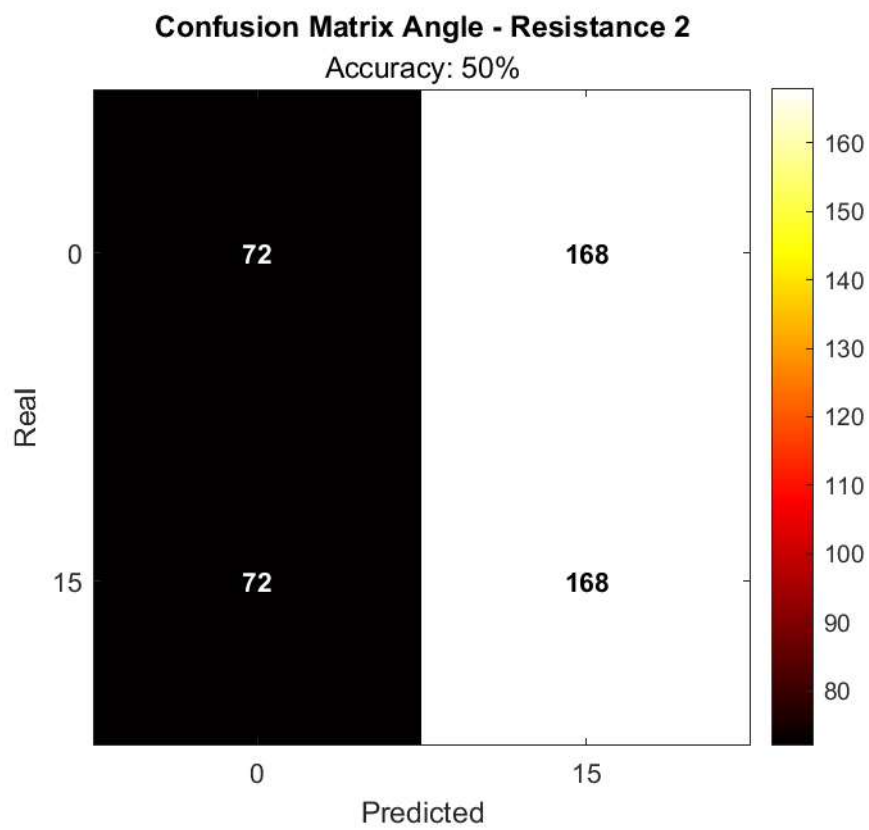


Figure 73: Confusion matrix for angle, resistance 2 – Broken branches unknown, load unknown.

Broken branches classification

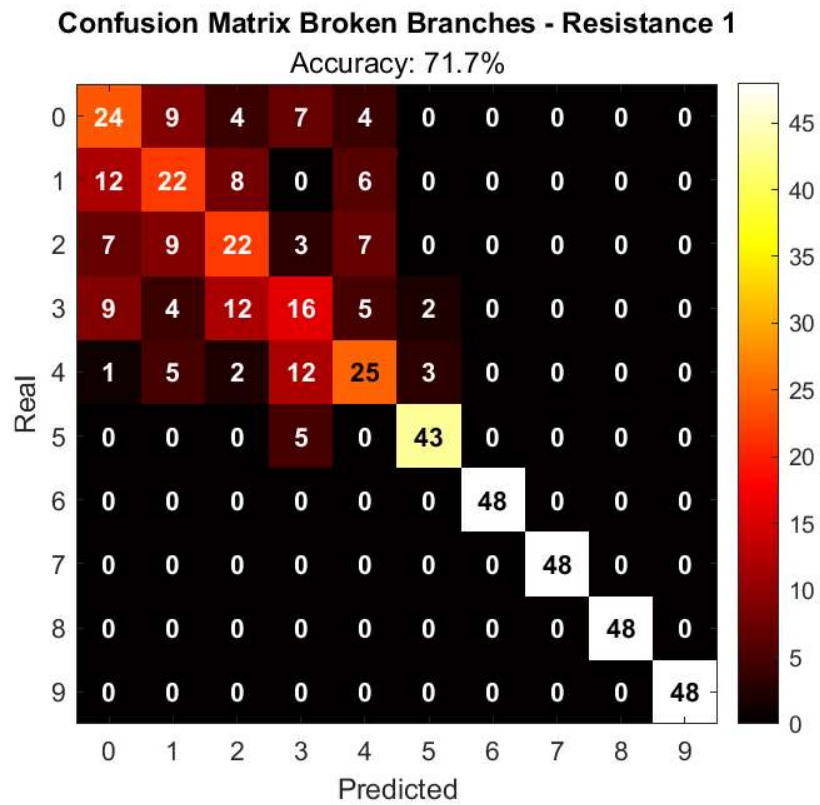


Figure 74: Confusion matrix for broken branches, resistance 1 – Angle known, load known.

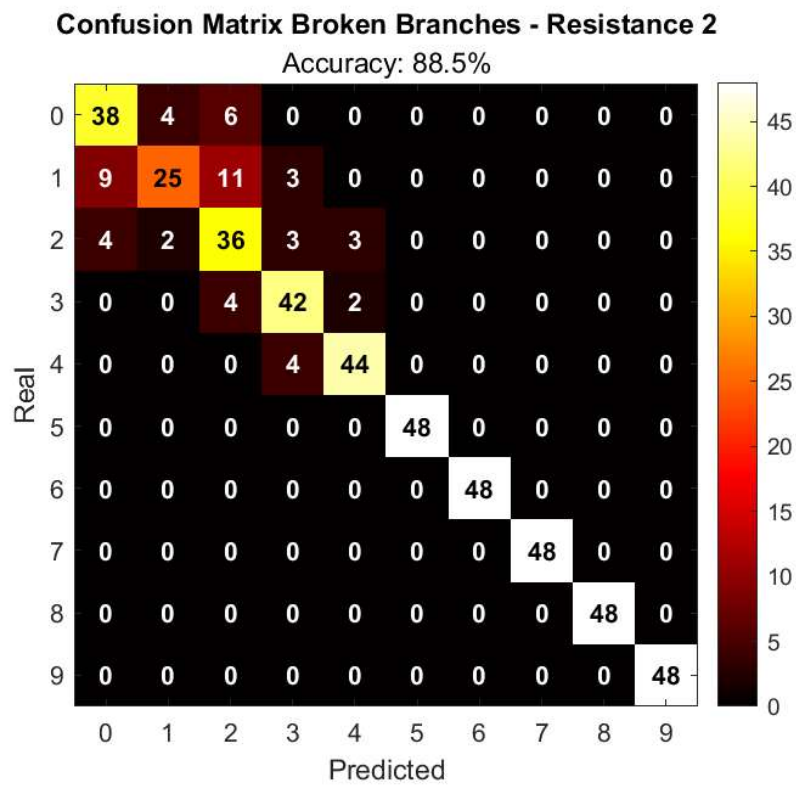


Figure 75: Confusion matrix for broken branches, resistance 2 – Angle known, load known.

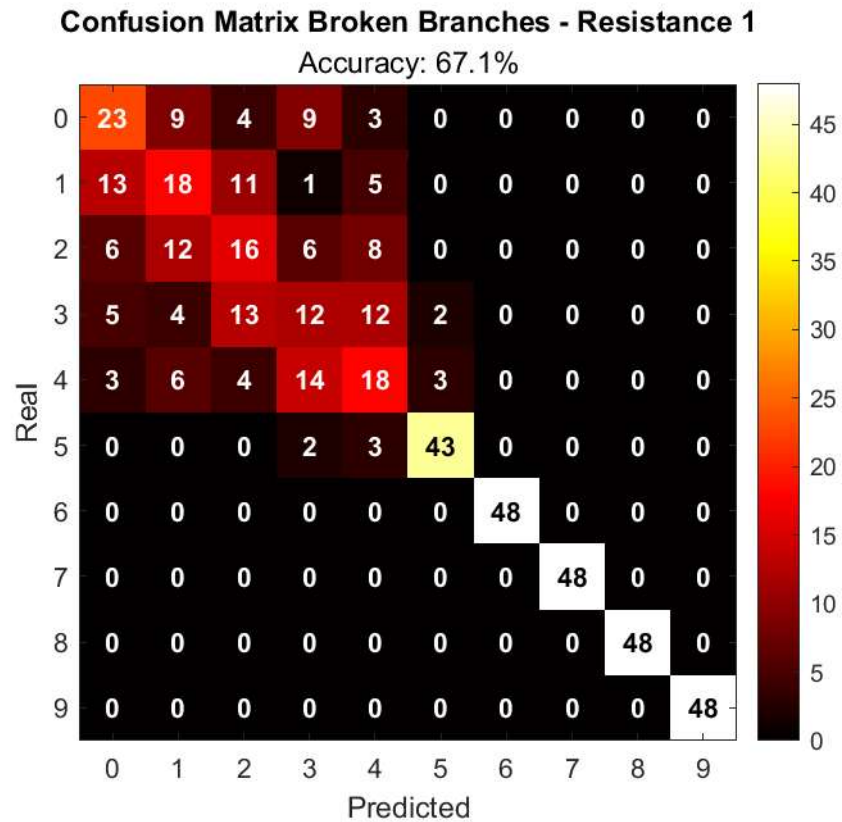


Figure 76: Confusion matrix for broken branches, resistance 1 – Angle known, load unknown.

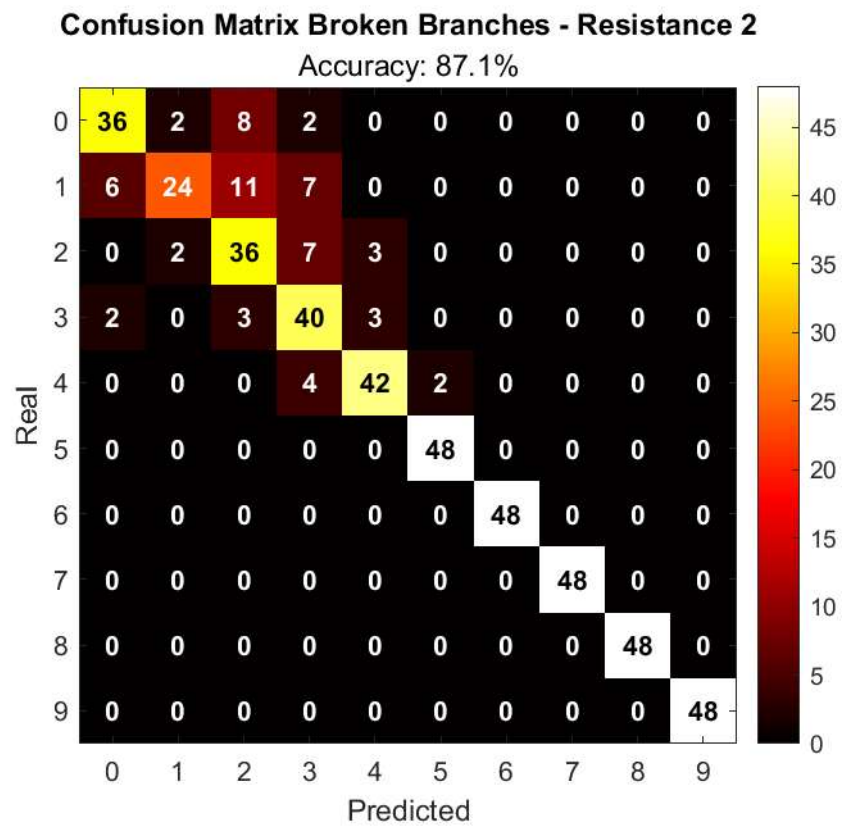


Figure 77: Confusion matrix for broken branches, resistance 2 – Angle known, load unknown.

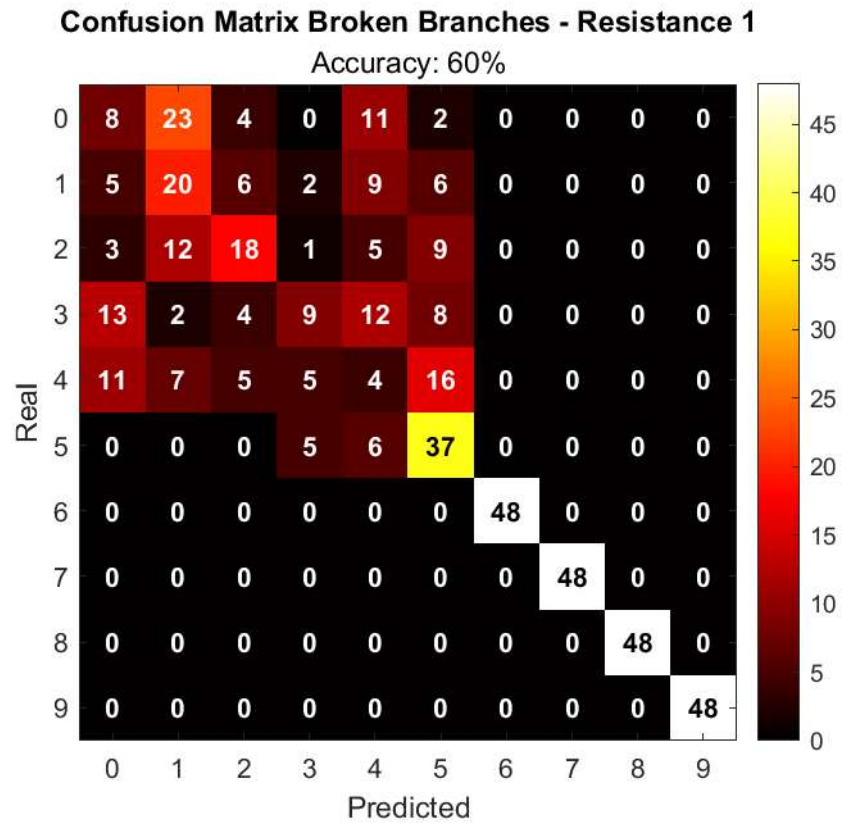


Figure 78: Confusion matrix for broken branches, resistance 1 – Angle unknown, load known.

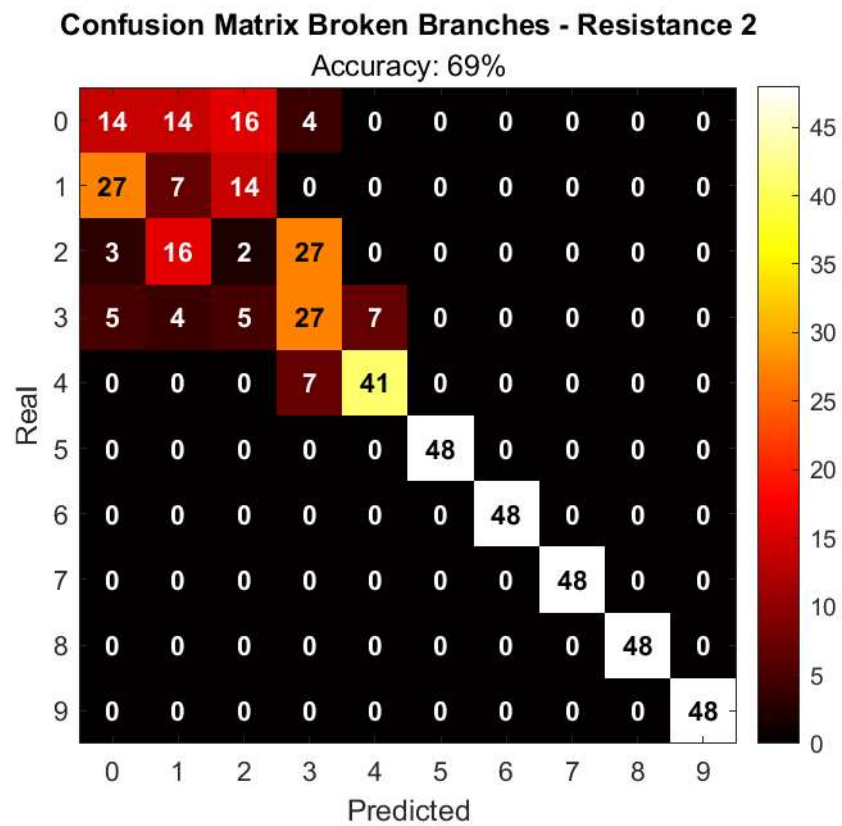


Figure 79: Confusion matrix for broken branches, resistance 2 – Angle unknown, load known.

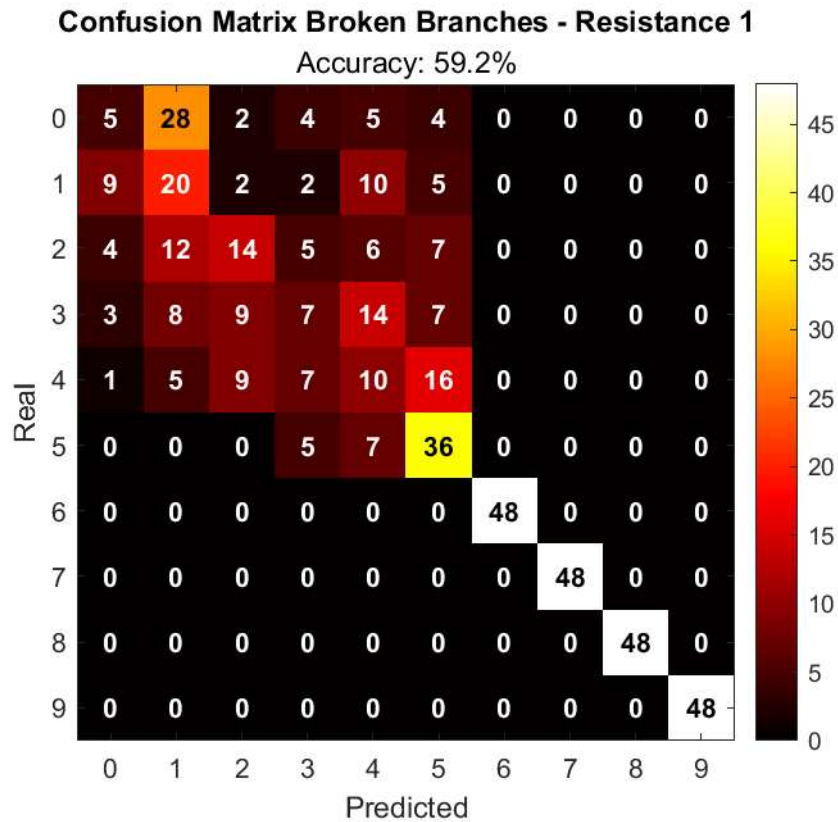


Figure 80: Confusion matrix for broken branches, resistance 1 – Angle unknown, load unknown.

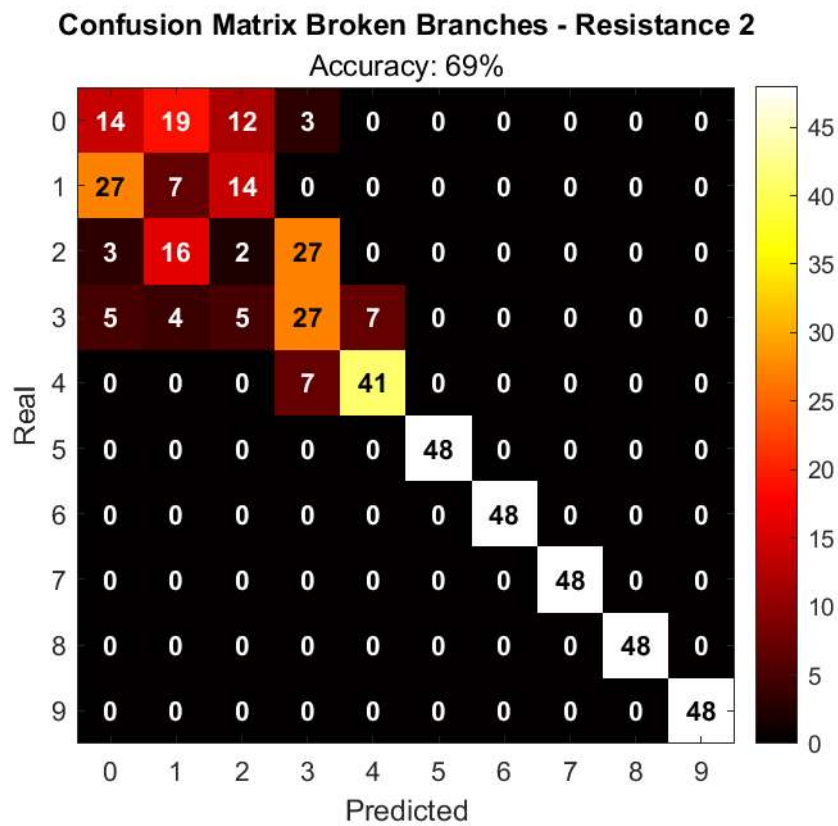


Figure 81: Confusion matrix for broken branches, resistance 2 – Angle unknown, load unknown.

Load classification

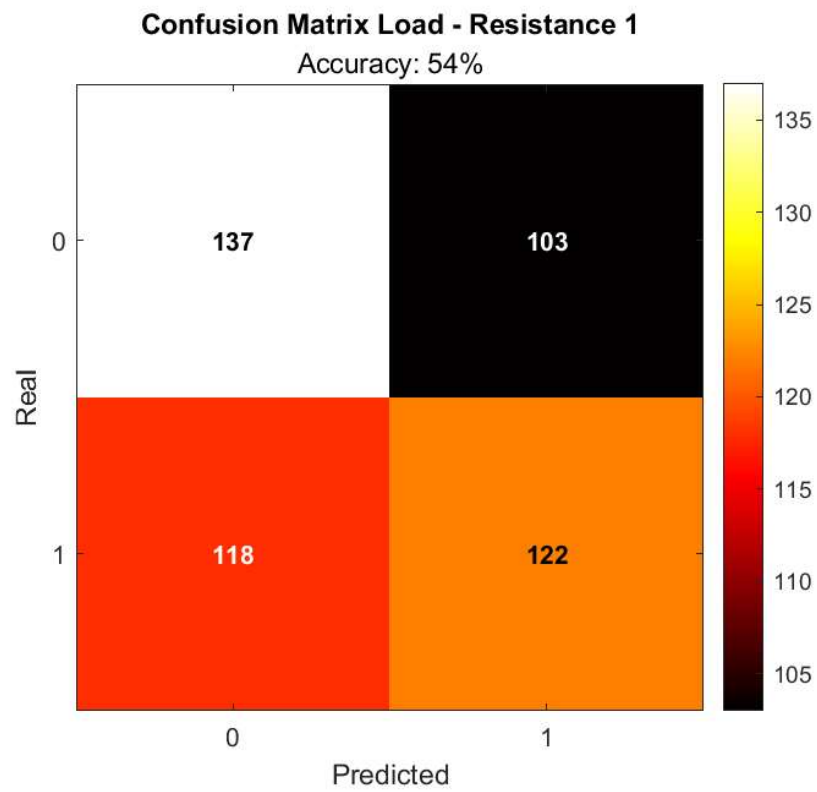


Figure 82: Confusion matrix for load, resistance 1 – Angle known, broken branches known.

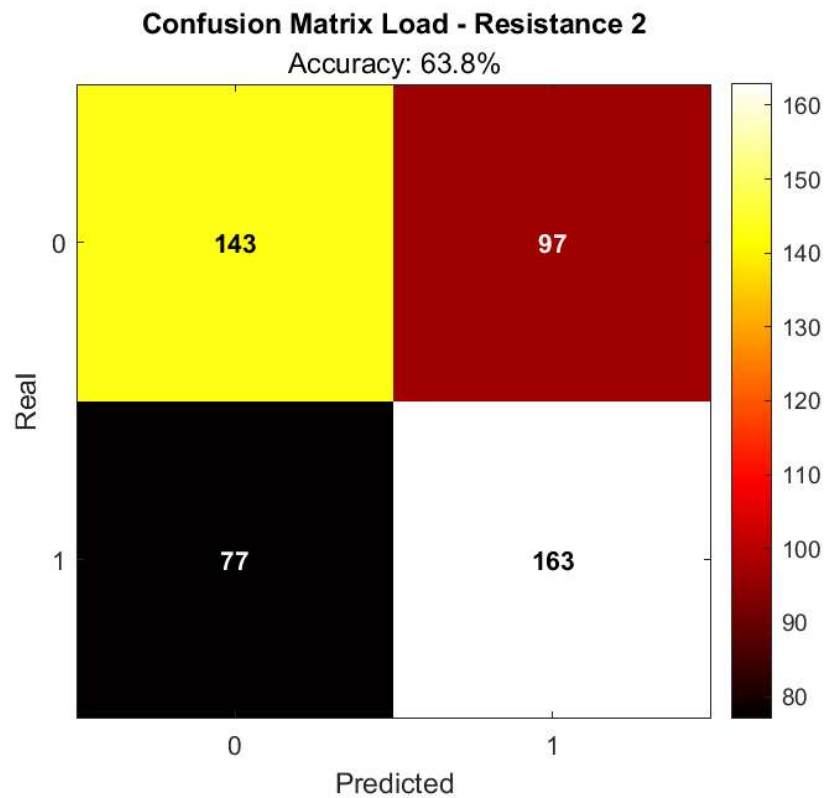


Figure 83: Confusion matrix for load, resistance 2 – Angle known, broken branches known.

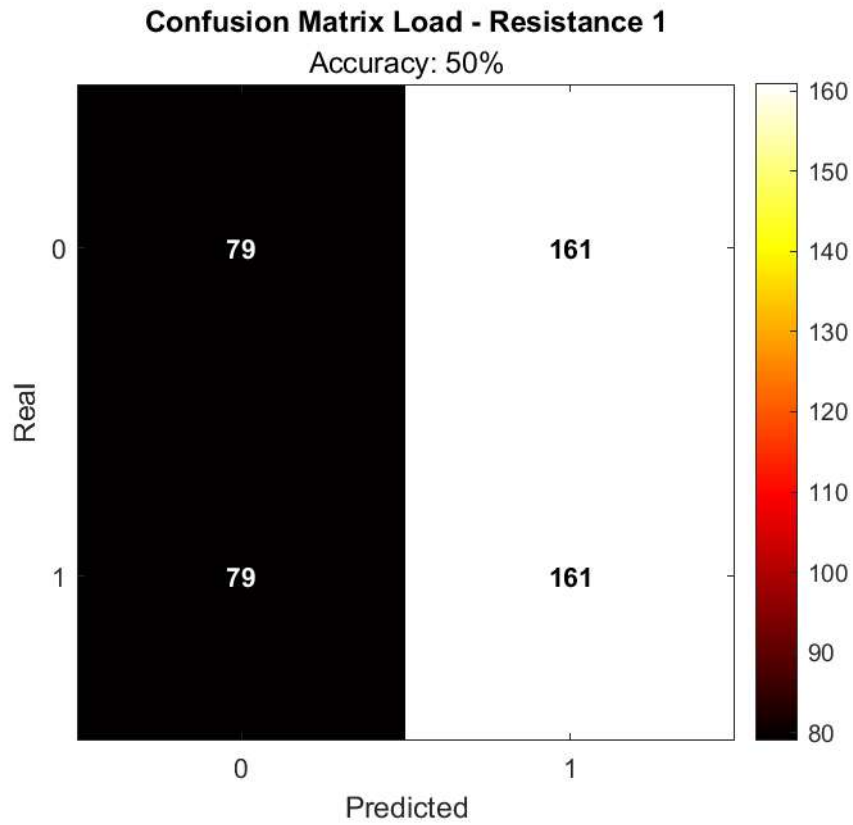


Figure 84: Confusion matrix for load, resistance 1 – Angle known, broken branches unknown.

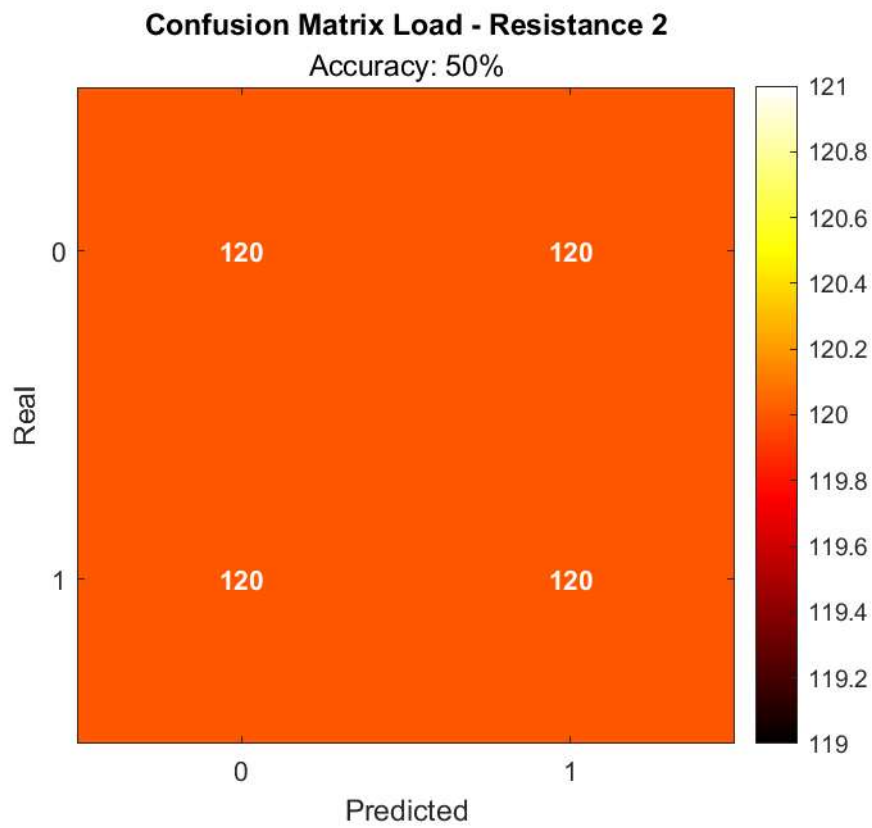


Figure 85: Confusion matrix for load, resistance 2 – Angle known, broken branches unknown.

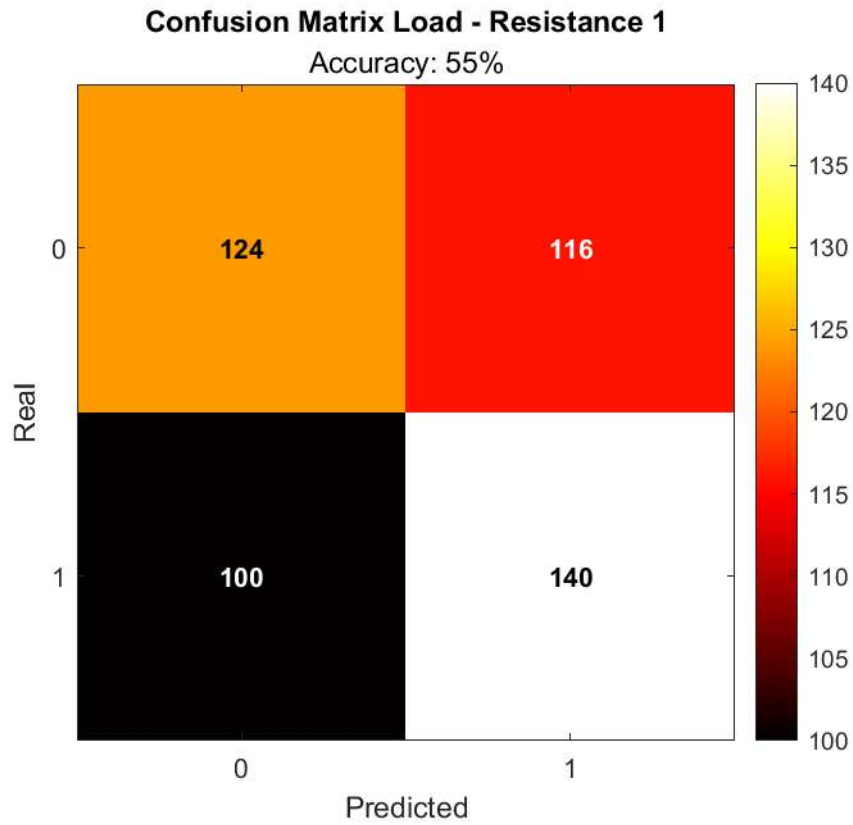


Figure 86: Confusion matrix for load, resistance 1 – Angle unknown, broken branches known.

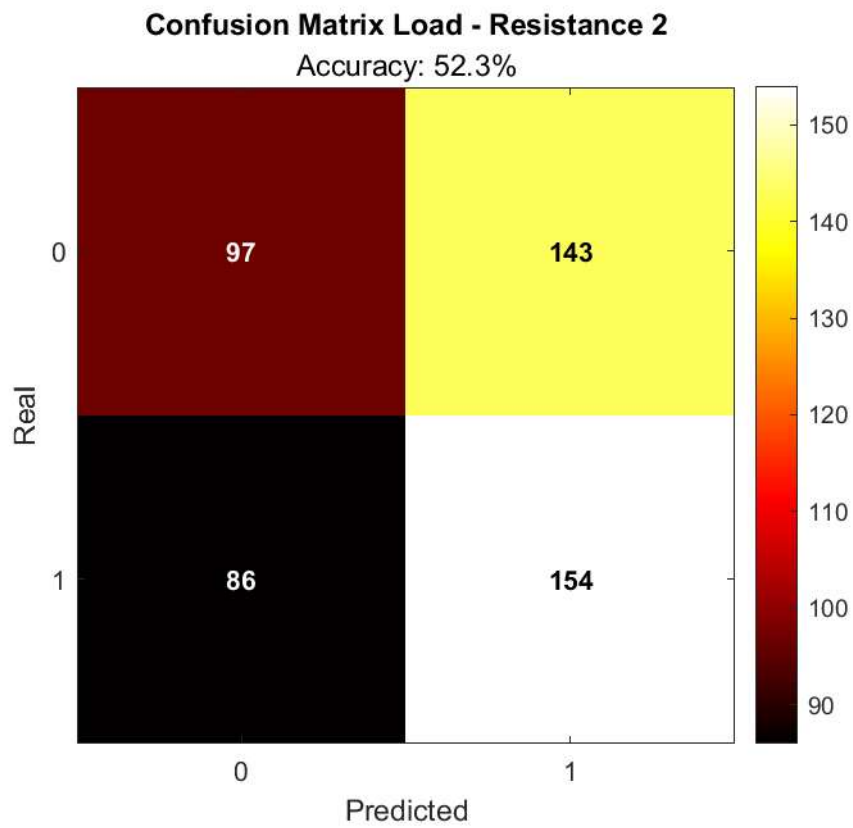


Figure 87: Confusion matrix for load, resistance 2 – Angle unknown, broken branches known.

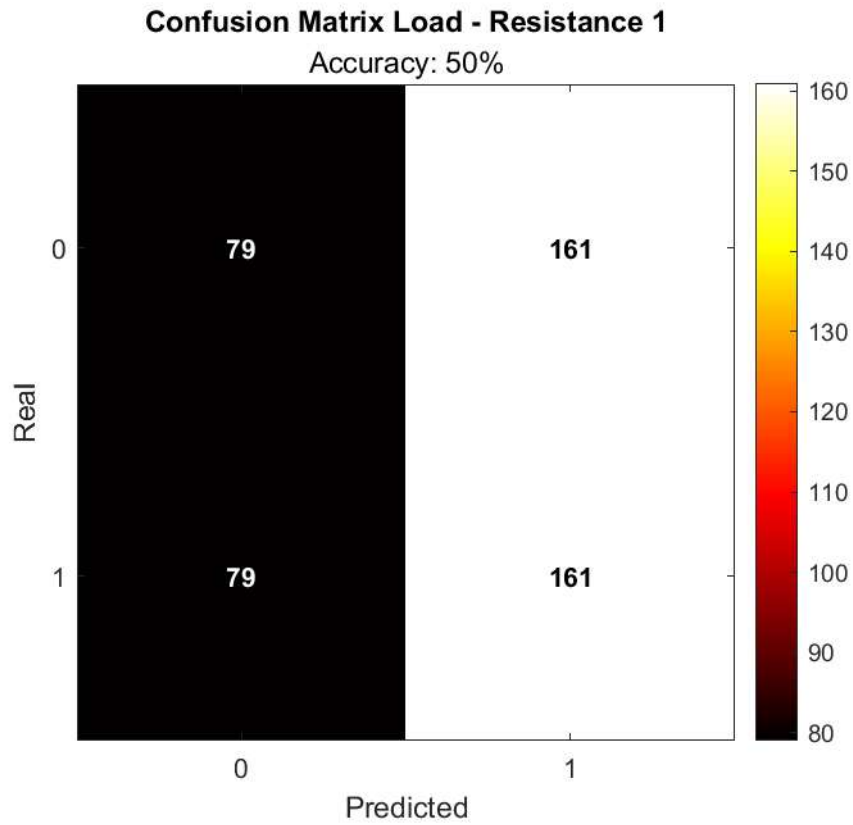


Figure 88: Confusion matrix for load, resistance 1 – Angle unknown, broken branches unknown.

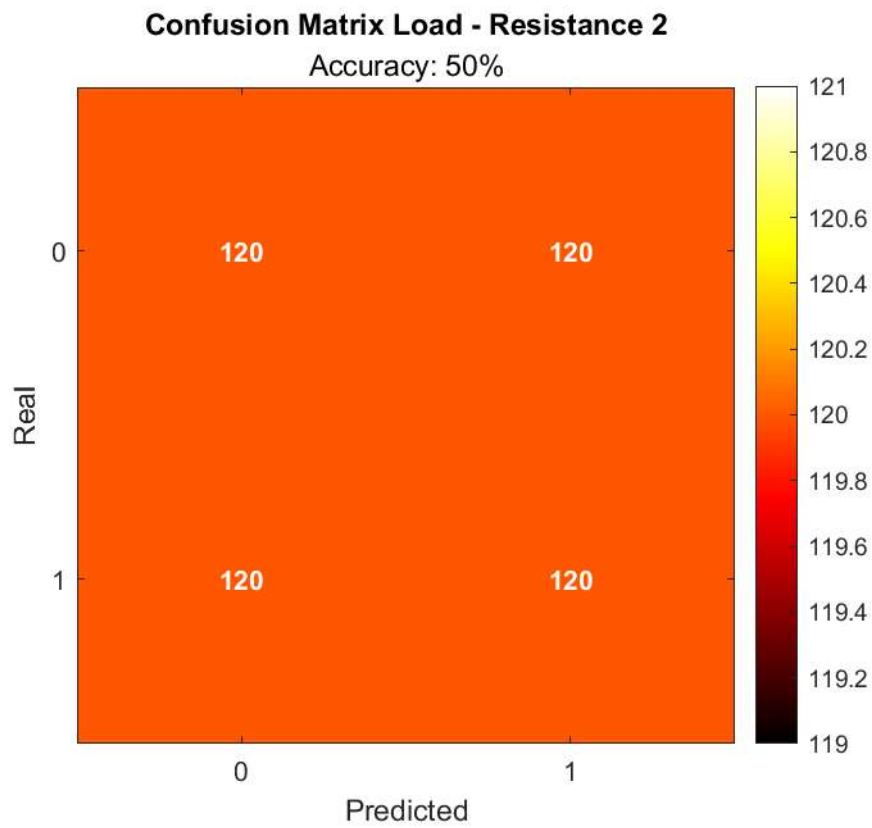


Figure 89: Confusion matrix for load, resistance 2 – Angle unknown, broken branches unknown.

The matrices closely resemble those of the previous approach because the global idea remains similar. Comparing the equivalent resistance implies a strong influence of the number of broken branches, followed by the stress angle and finally by the load. Hence, the classifications of angle or load without knowing the broken branches are not reliable, since they are based on two curves that are too close compared to the range of values of the resistances. In other words, in these cases the classification is guided only by the number of broken branches: if they involve an equivalent resistance whose value is below the two close medians, the value classified is the one related to the lowest curve and vice versa. Figure 90 shows this effect: the change of angle (A) and load (L) does not affect the closeness to the two medians used for the classification when the number of broken branches is unknown. Only the ID related to them has a real effect.

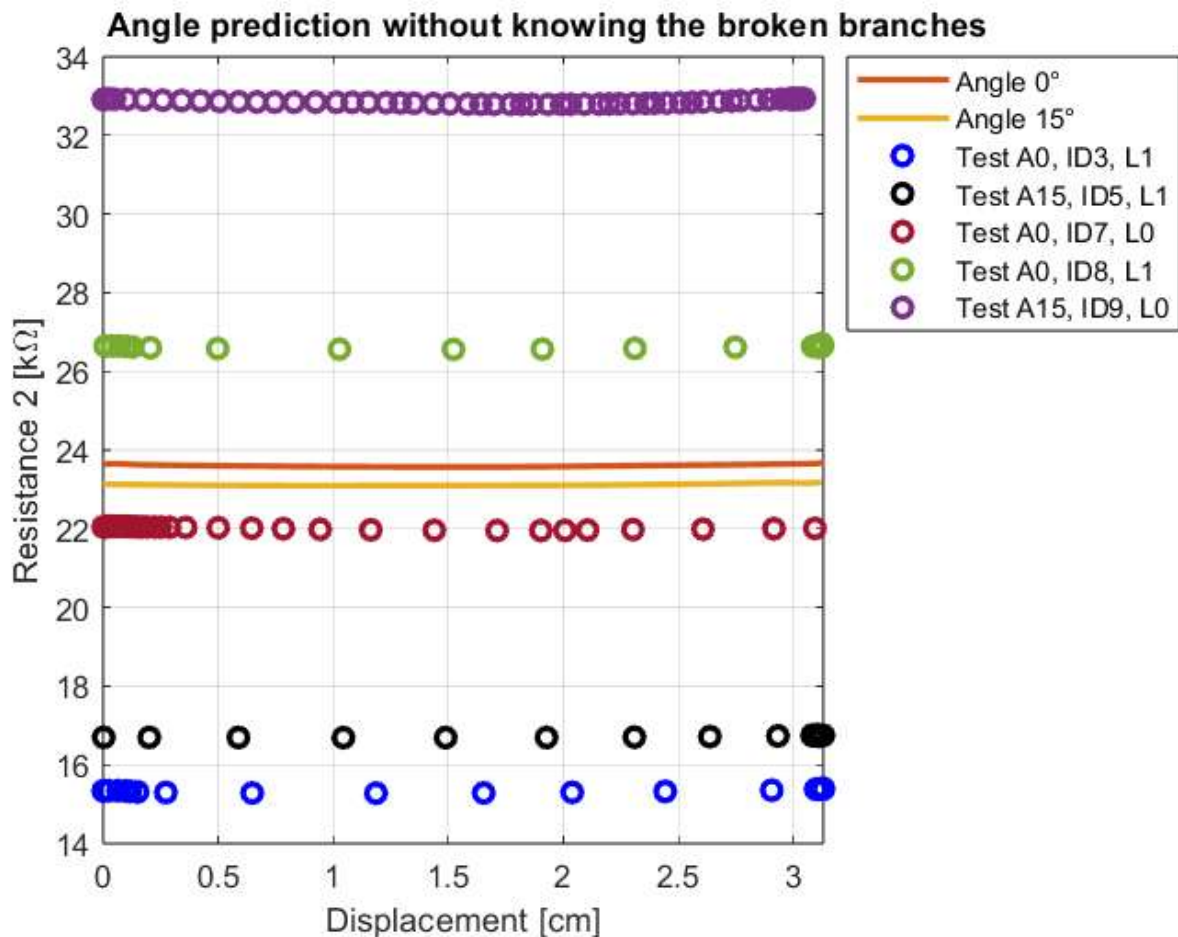


Figure 90: Angle classification without knowing the broken branches.

Here are resumed the accuracy values for each case, compared with the results of the first approach.

Classified variable	Angle known?	Broken branches known?	Load known?	Resistance used	Accuracy first approach [%]	Accuracy second approach [%]
Angle	-	Yes	Yes	1	79.2	74.4
				2	59.4	69.4
			No	1	77.5	72.1
				2	57.9	68.8
		No	Yes	1	53.5	51.3
				2	52.1	50.0
			No	1	51.7	51.3
				2	50.2	50.0
Broken branches	Yes	-	Yes	1	65.0	71.7
				2	84.2	88.5
			No	1	64.4	67.1
				2	83.5	87.1
	No		Yes	1	56.7	60.0
				2	83.1	69.0
			No	1	54.8	59.2
				2	83.3	69.0
Load	Yes	-	Yes	1	57.9	54.0
				2	61.7	63.8
			No	1	51.0	50.0
				2	53.5	50.0
	No		Yes	1	52.7	55.0
				2	59.8	52.3
			No	1	50.6	50.0
				2	53.1	50.0

Table 7: Accuracy of the second approach compared with the first one.

Like the confusion matrices, also the values of the accuracy are similar to the first approach, but generally slightly lower. Once again, the best choice is to classify first the broken branches, then the stress angle and finally the load. This method allows to reach the following accuracies:

- accuracy for the broken branches (resistance 2): 69.0 %;
- accuracy for the stress angle (resistance 1): 72.1 %;
- accuracy for the load (resistance 2): 63.8 %.

The same accuracies in the first approach are, respectively: 83.3%, 77.5% and 61.7%. Consequently, we can notice a higher accuracy in the classification of the load and a lower one for the other variables.

5.3 Remarks

Since the main idea of this approach is similar to the first one, also the qualitative metrics give similar results.

Accuracy

The accuracy assumes values between 60 % and 75 %. It is medium for the classification of the broken branches and of the angle, and it is low for the load. In general, it is lower than the accuracy of the first approach.

Interpretability

Exactly like in the first approach, the model is 100 % explainable and compact.

Robustness

The model is highly robust. The use of the median strongly decreases or even cancels the influence of the outliers. However, missing data about a certain variable is still a problem and the classification for a value of one variable is impossible without information about it.

Incrementality

The incrementality of the model is medium-high, because every newly labelled record has an influence on the medians regarding it, but the more the amount of data, the less the influence of every

new information. As in the first approach, the model can be easily updated with new data, but it is not automatic and a human intervention is needed to update the table containing the family medians.

Efficiency

The efficiency of the model is excellent: both building the medians (model building) and the classification require about a second using a common computer.

Scalability

The model has a good scalability, thanks to the small amount of experimental data required and the low number of attributes.

All in all, the second approach gives some good results, but to better understand it there is a need to compare it with the first approach.

Strengths

- The algorithm **preserves the strengths** of the first approach: the classification is reliable (especially for the number of broken branches), the training requires a small number of records, and both the training and the classification require a small amount of time.
- There is **no more arbitrariness** in the score, avoiding the loss of generality.
- The **robustness** is **high**: the influence of the outliers is strongly decreased.

Weaknesses

- **Information** on the dispersion of individual families is **lost**.
- The **accuracy** of the model is **lower** than the one in the first approach.

Since the strengths of this approach should have a good effect on the accuracy, the lower performance can be attributed to the information loss. This likely occurs because of the significant dispersion of the experimental data, given that the component is a prototype subject to various irreversible phenomena. Therefore, in a Machine Learning approach, preserving the information about the range of resistance values is more important than fitting an ideal curve that the resistance should follow.

Once again, the third approach is designed to fix the weaknesses maintaining the strengths of the second approach.

6 Third Approach: Gaussians

6.1 Algorithm Explanation

As the second approach, the third one is based on the previous one, but it is designed to fix its weak points. Hence, the main idea is to substitute the median of each family with a Gaussian distribution containing both information about the ideal curve to fit with and the data dispersion. The score of each family reflects the probability of belonging to that family. Thus, this approach allows to combine the strengths of both the previous ones.

6.1.1 Training

The dataset used is again the one related to the crack propagation. It is managed the same way as the first two approaches (see paragraph 4.1.1), obtaining the same training and test tables.

Also the training has the same structure, with a few differences. It follows the steps below.

- Choose which one of the two measured resistances will be used for the classification. Once again, only one resistance at the time can be used.
- Build a family set for each combination of stress angle, ID and load, using only the records contained in the training table.
- For each family, compute the parameters (mean and standard deviation) of the ideal Gaussian distribution of the resistance for each displacement value. Start with interpolating each test using 200 equally spaced values of displacement, as in the second approach (see paragraph 5.1.1). Then, for each displacement value, compute the parameters of the ideal Gaussian distribution based on the 24 training records of that family. Store all the information in a table that constitutes the model of this algorithm. For each displacement value, the mean and the standard deviation are computed using the following formulas:

$$\mu(dis) = \frac{1}{N} \sum_{i=1}^N res_i(dis)$$
$$\sigma(dis) = \sqrt{\frac{1}{N-1} \sum_{i=1}^N [res_i(dis) - \mu(dis)]^2}$$

Where:

- $disp$ is the displacement of the interpolated point;
- res_i is the resistance of the interpolated point, for the i -th training record;
- N is the number of training records of the family (hence, N is always 24);
- $\mu(disp)$ is the ideal mean of the family computed for that displacement;
- $\sigma(disp)$ is the ideal standard deviation of the family computed for that displacement.

See the Matlab script *Build_gaussians_table.m* and *gaussian.m* in the appendix for the details.

6.1.2 Test

- One main difference with the other approaches is the management of the unknown variables. Following a reasoning that will be explained at the end of this paragraph, we conclude that the variables must be classified all together. Consequently, all the variables are considered unknown and there is no family reorganization.
- The parameters of the distributions stored in the table during the training are used here. Given the displacement-resistance profile of the test we want to classify, we compute a score to estimate the probability of belonging to each family. The record is reorganized in 200 points equally spaced along the displacement axis, using interpolation for the values of the resistance. For each new point of the test curve, it is added to each family score the value of the Probability Density Function (PDF) evaluated for the resistance value of that point, using the mean and the standard deviation computed for that displacement value. This is resumed in the following equation:

$$score(disp) = \frac{1}{\sigma(disp)\sqrt{2\pi}} e^{-\frac{(res-\mu(disp))^2}{2\sigma(disp)^2}}$$

Where:

- $disp$ is the displacement of the interpolated test point;
- res is the resistance of the interpolated test point;
- $score(disp)$ is the score computed for that displacement, to be added to the total score of that family;
- $\sigma(disp)$ is the ideal standard deviation of the family, computed for that displacement;
- $\mu(disp)$ is the ideal mean of the family, computed for that displacement.

The score of each family is the sum of the scores given by all the test point.

Actually, *disp* is the index that identifies which Gaussian parameters must be picked in the array containing their values with respect to the displacement. As in the other approaches, the rigorous way to compare the arrays would be using the PDF built with the Gaussian parameters referred exactly to the right displacement value for each point. But again, since the minimum and the maximum values of the displacement are fixed in this dataset (unless small measurement errors), thanks to the high number of points, basing the compare on the index of the displacement array or on the displacement value itself gives roughly the same results. This procedure let us to reduce the computational cost.

- The family with the highest score is the chosen family. It is already related to one unique combination of stress angle, broken branches and load.

The choice of classifying all the variables at the same time is given by the following reasoning.

When two or more families should be grouped together, there are two possible ways:

- maintain all the information about the training records also during the test. When a groupment is required, combine all the records and compute the parameters of the ideal Gaussian distribution;
- use the table built during the training and combine the parameters using the following formulas.

$$\mu_{new} = \frac{1}{N} \sum_{i=1}^N \mu_i$$

$$\sigma_{new}^2 = \frac{1}{N} \sum_{i=1}^N \sigma_i^2 + \frac{1}{N} \sum_{i=1}^N (\mu_i - \mu_{new})^2$$

Where:

- μ_{new} is the ideal mean of the family groupment;
- μ_i is the ideal mean of the i-th family;
- σ_{new} is the ideal standard deviation of the family groupment;
- σ_i is the ideal standard deviation of the i-th family;
- N is the number of families grouped together.

Since the number of training records is the same for every family set, there is no difference between the two ways. Therefore, the combination of the parameters is preferable because it has a smaller

computational cost. Moreover, it allows not to maintain all the information about the experimental records, using only the table that constitutes the model of the algorithm. Finally, it highlights if the groupment of the families can be done or if it leads to a PDF that is no representative of the records.

Figure 91 shows an example of combination of all the families with the same ID (ID1), when the displacement is a half of its maximum. As we can see, combining families with different angles is not a good statistical approach for this dataset: we have already seen in the first approach that the variance is significantly higher when the stress angle is 15° (see Figure 61). Hence, the groupment cannot be done between different angles. Instead, it is possible between different loads, thanks to the fact that a different load leads to a different slope in the displacement-resistance curve, but it does not involve a change in the mean value or in the variance of the family set.

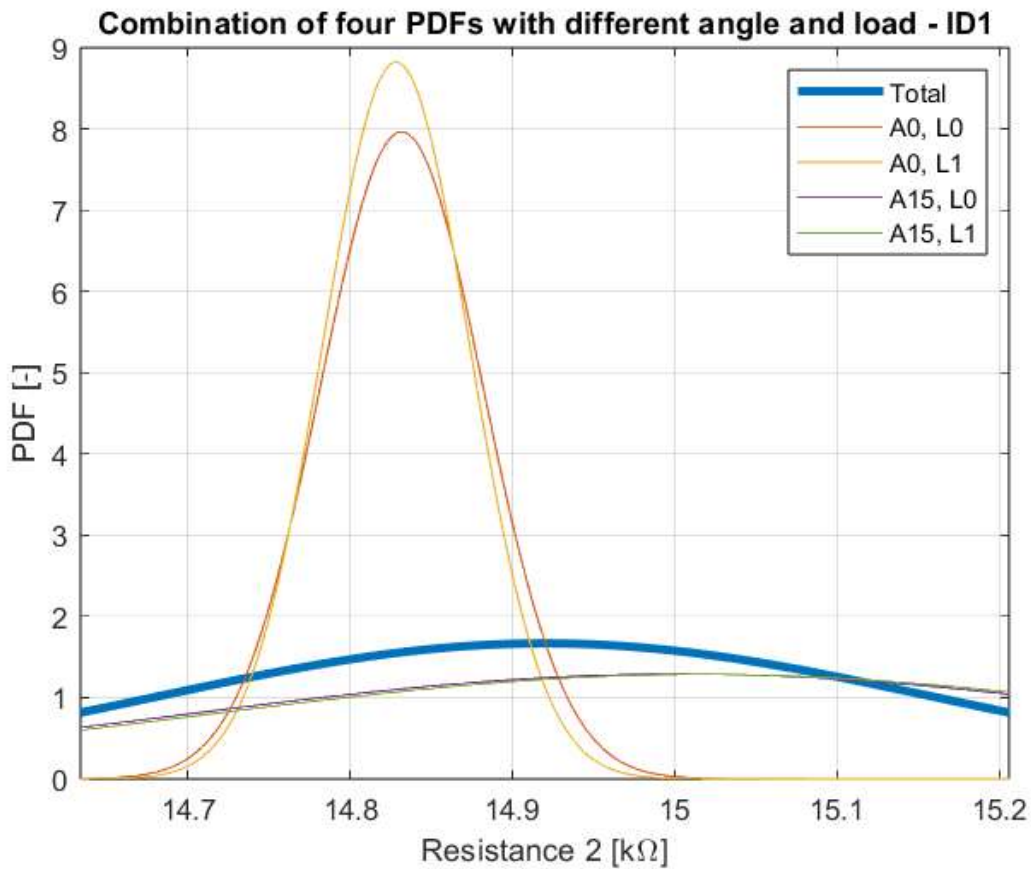


Figure 91: Combination of four PDFs with the same ID.

Notation: “Ax” stays for “Angle = x”; “Ln” stays for “Load = n”; “Total” is referred the combination of all the families.

The combination of PDFs with IDs referred to the same number of broken branches (those whose sum is 21) leads to results that are acceptable, as we can see in Figure 92. It shows the combination of ID1 and ID20 to obtain the PDF for zero broken branches, and the combination of ID2 and ID19 for one broken branch. The displacement is a half of its maximum and both the angle and the load are null. The two total PDFs shown hide the peak given by the higher ID, but they are also well separated (unusual for a small number of broken branches, as we have seen in the previous approaches), so they can be used for the classification.

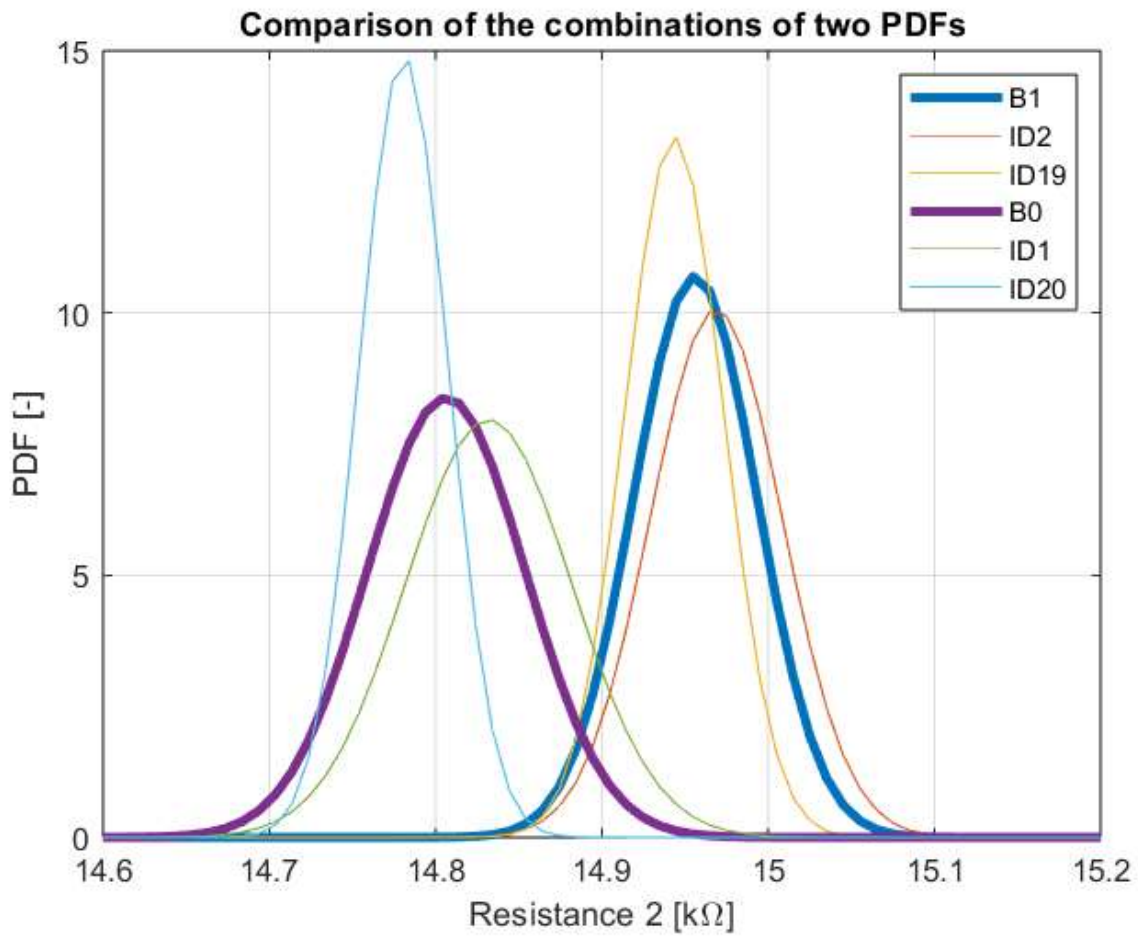


Figure 92: Combination of PDFs with the same number of broken branches.

Notation: “Bn” stays for “n broken branches”.

Recalling all the reasoning about the influence of each variable on the equivalent resistance, it is intuitive that the combination of PDFs referred to a different number of broken branches does not lead to a representative total PDF, as we can see in Figure 93. It shows the PDFs for ID1 (B0) and ID2 (B1), with null angle and load, when the displacement is a half of its maximum.

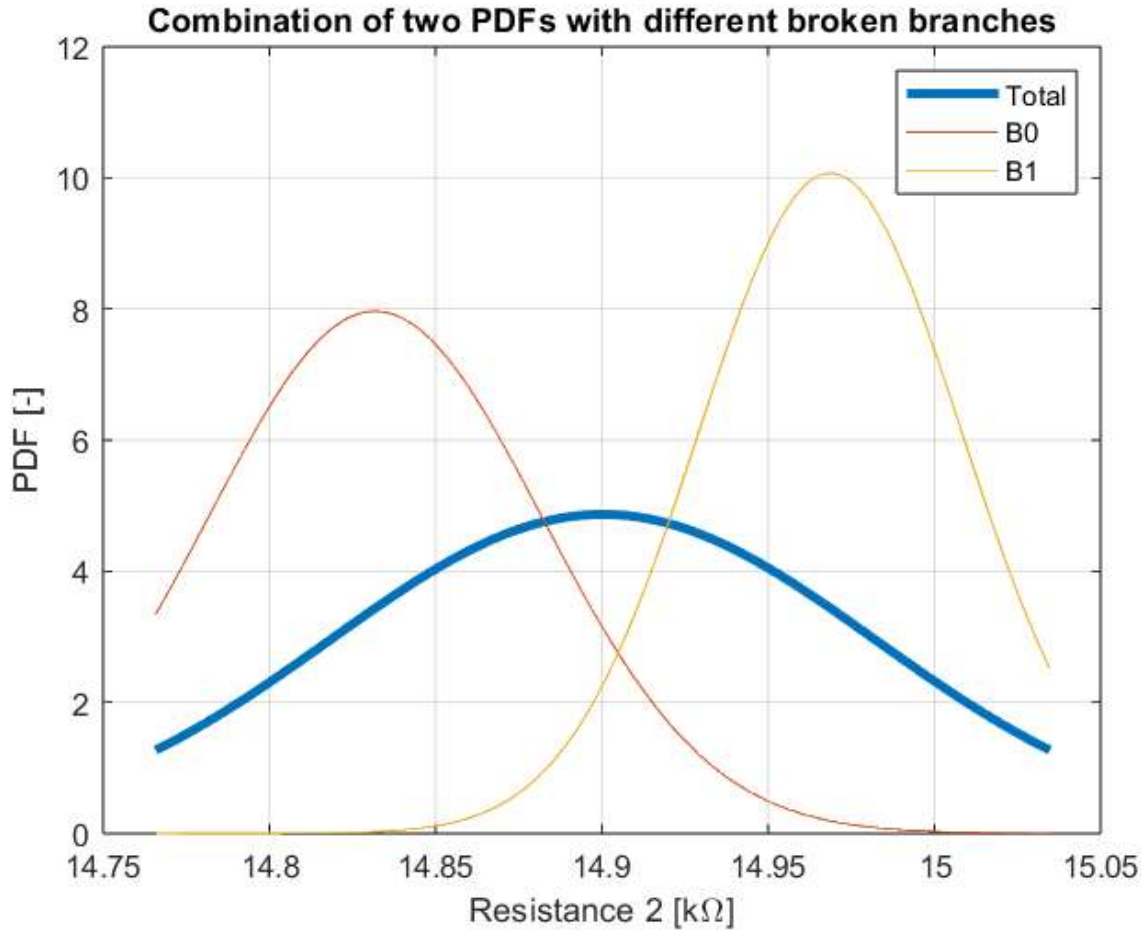


Figure 93: Combination of two PDFs with different number of broken branches.

All in all, it seems that the combination of some families can be done in some cases. For example, we could imagine classifying first the number of broken branches, grouping the IDs referred to the same number of broken branches and the different loads. Since the angles cannot be grouped, we would obtain 20 families (two for each number of broken branches) and we could finally classify this variable. Then, when we would classify the angle, if we follow what we have done in the other approaches, we should compare only the PDFs with the number of broken branches obtained in the

previous step and different values of angle, grouped through the load. But this is redundant because in the previous step the same curves have been compared. Hence, after the first step, we already classified the number of broken branches and the stress angle. Finally, during the load classification, we should divide the combined PDF used so far into the two PDFs referred to different loads. Looking at this method, we can derive that it is not necessary to group the families for the classification of some variables, splitting them again in the final steps. Thus, the most reasonable method is to maintain all the PDFs separated and directly search the most similar to the test to classify. Hence, the classified family contains all the information about the value of the three variables: stress angle, broken branches and load.

6.1.3 Example

Unlike in the other approaches, here it is difficult to show the PDFs related to the families, since there are three dimensions to look at: displacement, resistance and PDF value, being the last one a function of the first two variables. Thus, Figure 94 shows a test and the PDF related to the family with angle 15° , 5 broken branches and load 1. The intensity of the blue colour reflects the PDF value. Only the PDF related to one family is plotted to make the figure more readable.

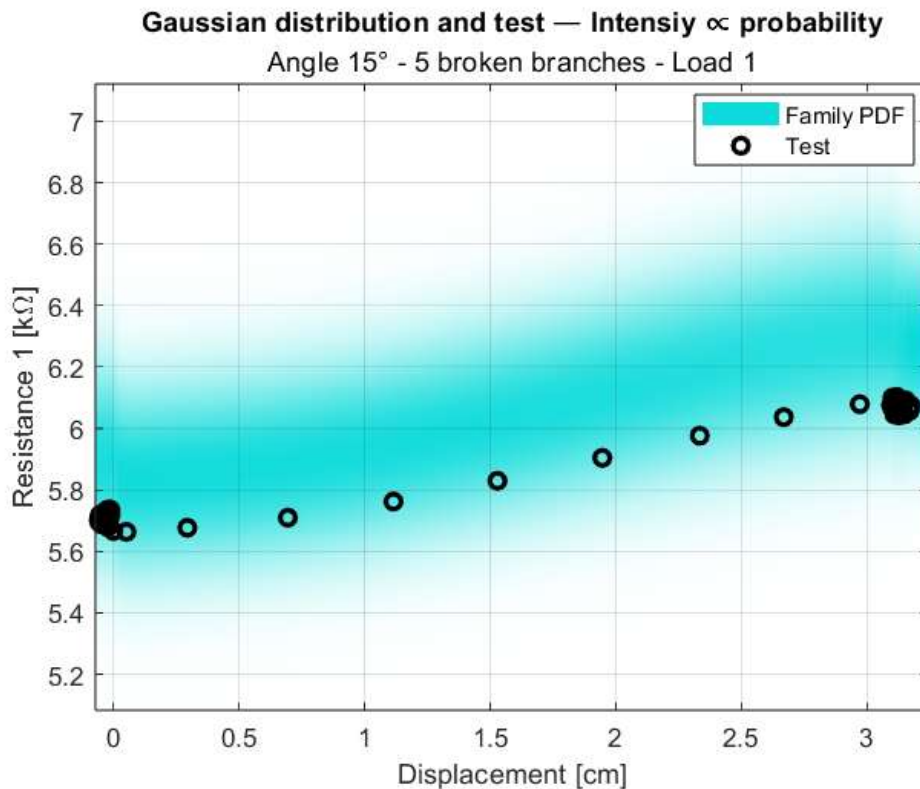


Figure 94: Gaussian distribution and test for a family.

The same information can be shown in the following 3D graph.

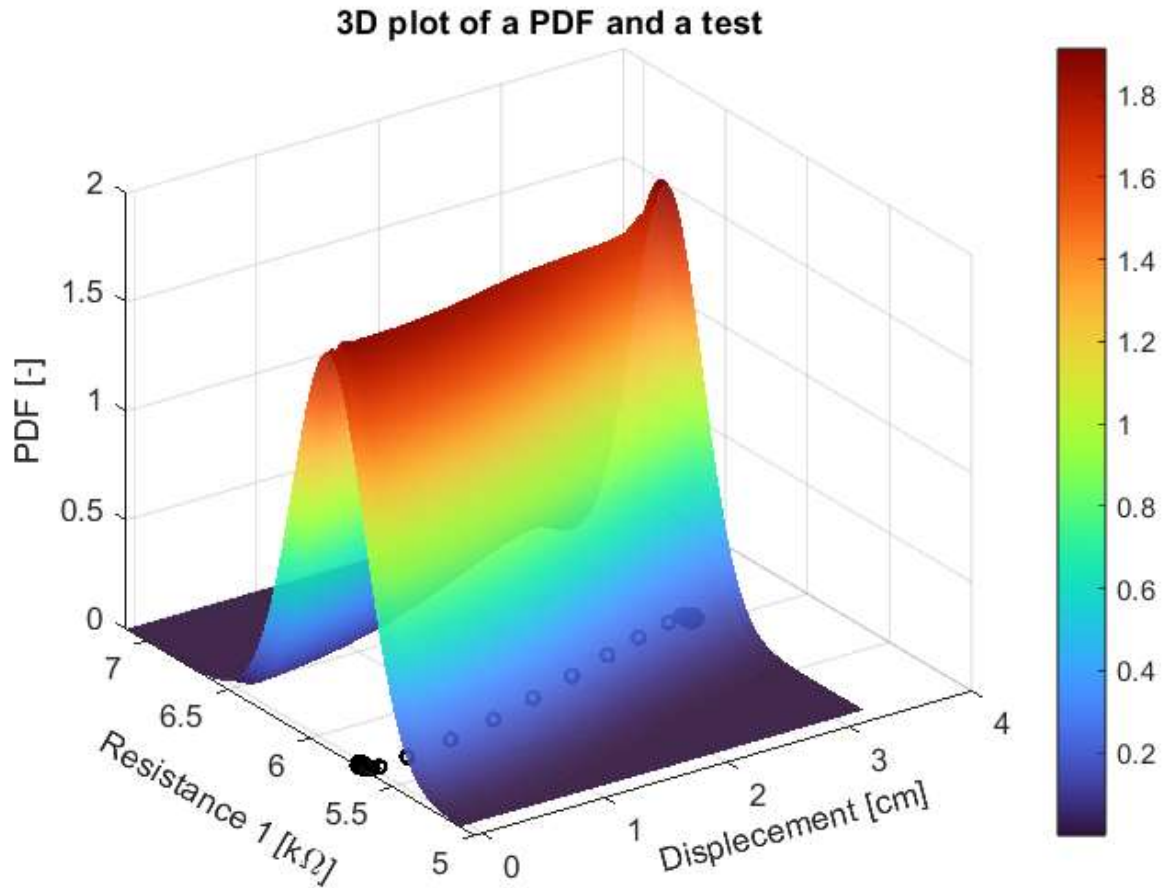


Figure 95: Gaussian distribution and test for a family – 3D plot.

Therefore, for this approach a new way to present a classification is needed. Clearly, both the ways to show the PDF of a family are not applicable when the number of curves to compare becomes 80, given by all the possible combinations of angle, broken branches and load. Hence, this part will not be shown graphically. Figure 96 is a bar chart to show the results to classify the variables for a test with angle 0° , 4 broken branches (ID5) and load 1.

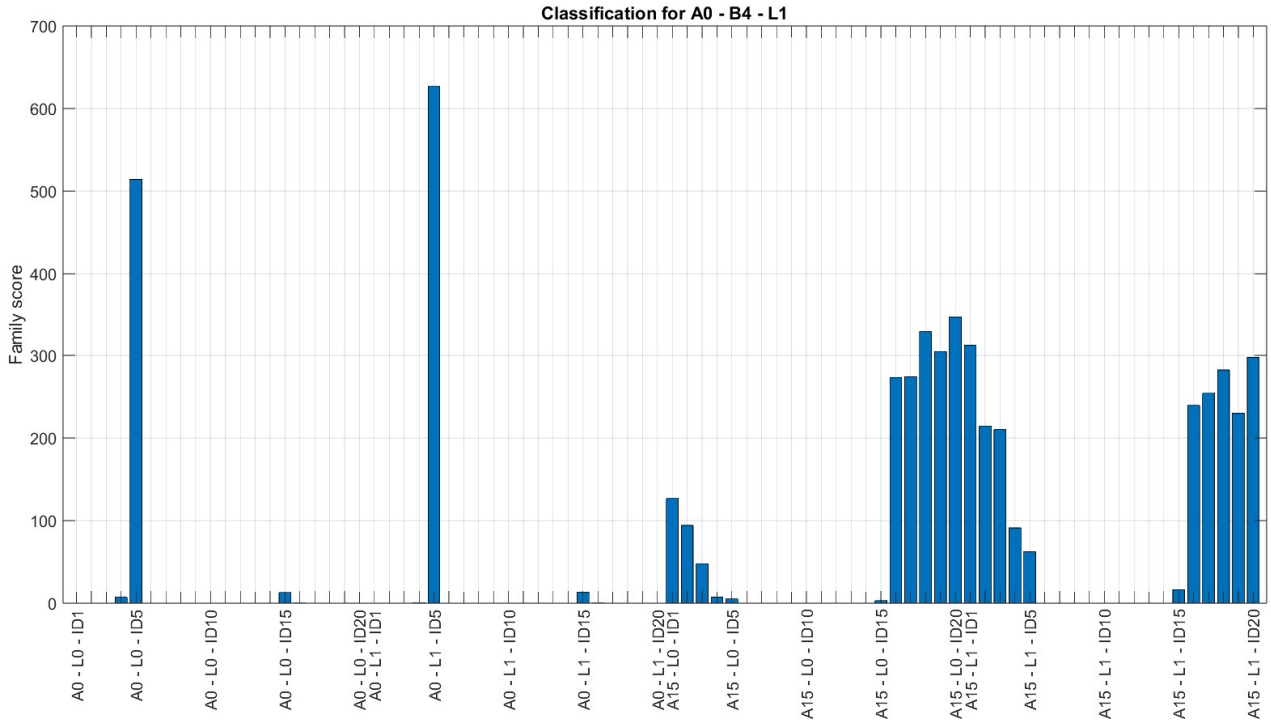


Figure 96: Classification example using the third approach.

As we can see, the chosen family (so the one with the highest score) is the right one. The family with the second highest score is the one with same angle and ID, but different load, highlighting the fact that also for this algorithm the load is the most difficult variable to classify. Other families with a significant score are the ones with a small number of broken branches: as we have seen, they are more difficult to distinguish.

6.2 Results

As in the other approaches, the confusion matrices can be built. But, since here the variables are classified all at once, there are no more issues about which are known and which are not. Although all the variables are classified during the test, the matrices are built looking only at the classification of one chosen variable. Thus, only 6 matrices can be built, one per variable, using one resistance at the time. See the Matlab script *Build_confusion_matrix_broken_branches_gaussians.m* in the appendix for the details.

Stress angle classification

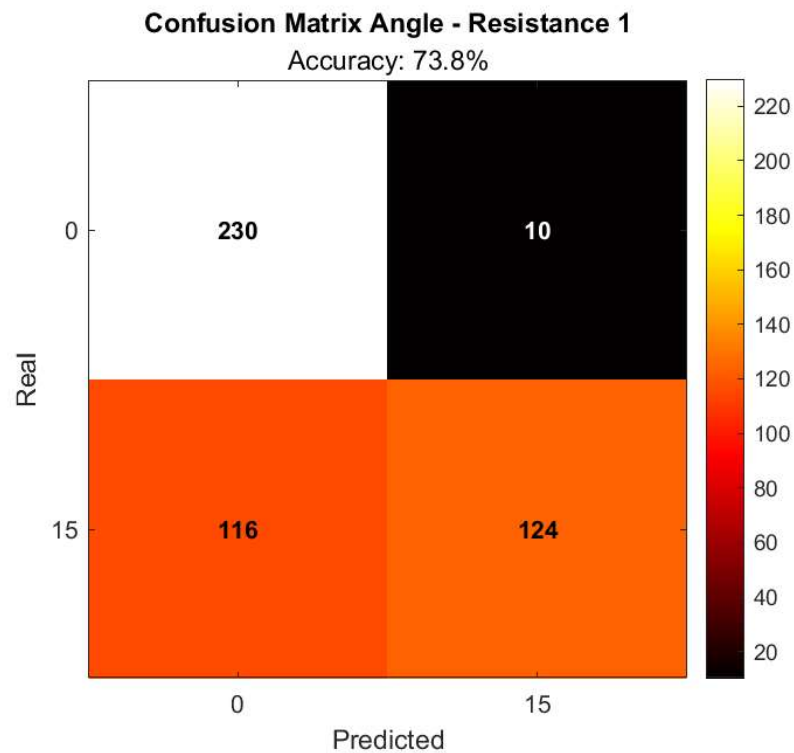


Figure 97: Confusion matrix for angle, resistance 1.

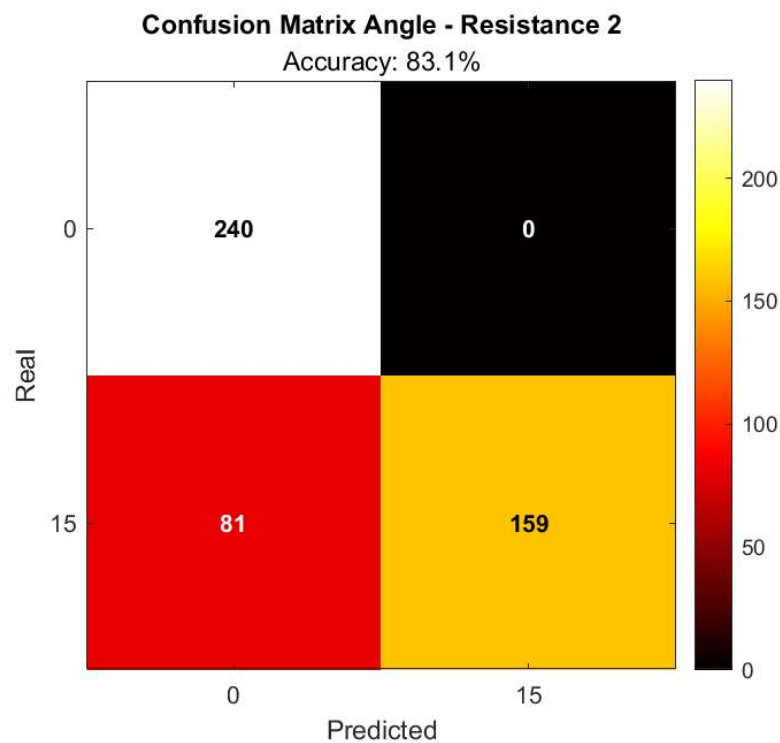


Figure 98: Confusion matrix for angle, resistance 2.

Broken branches classification

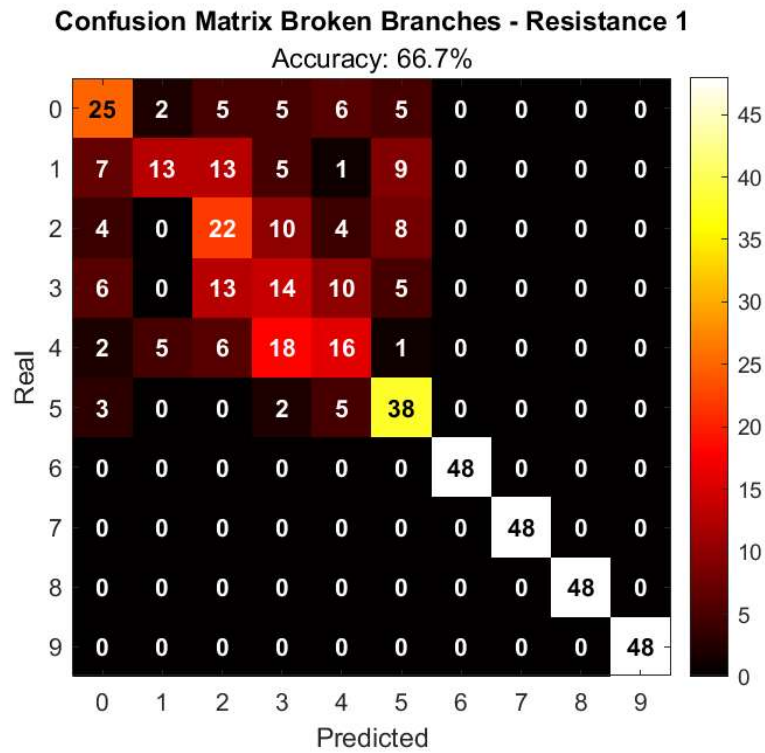


Figure 99: Confusion matrix for broken branches, resistance 1.

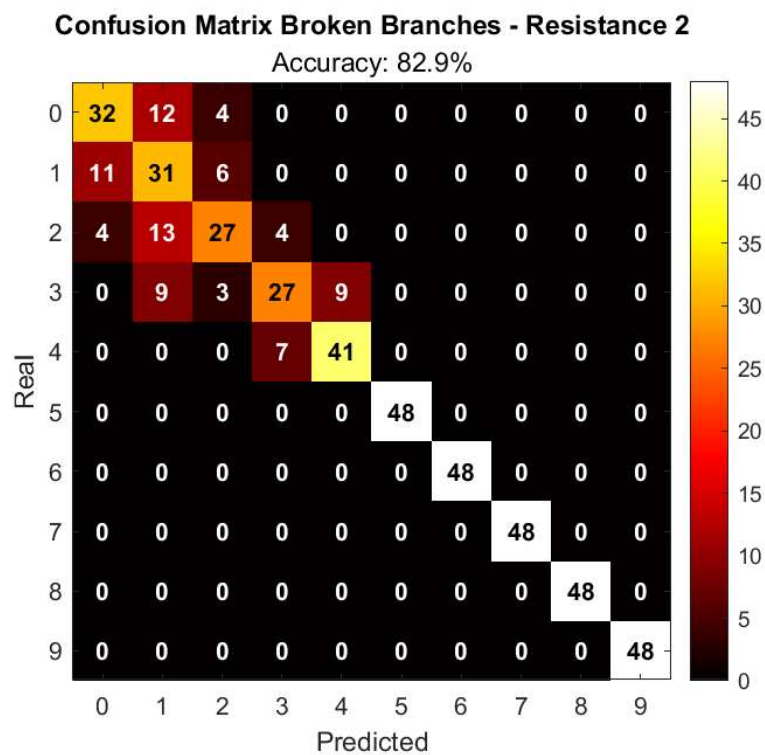


Figure 100: Confusion matrix for broken branches, resistance 2.

Load classification

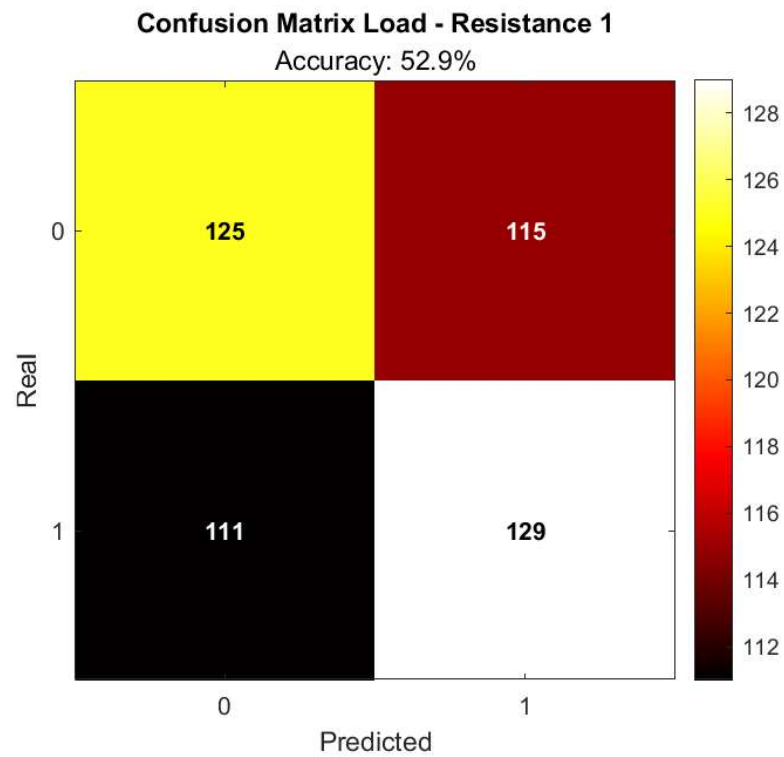


Figure 101: Confusion matrix for load, resistance 1.

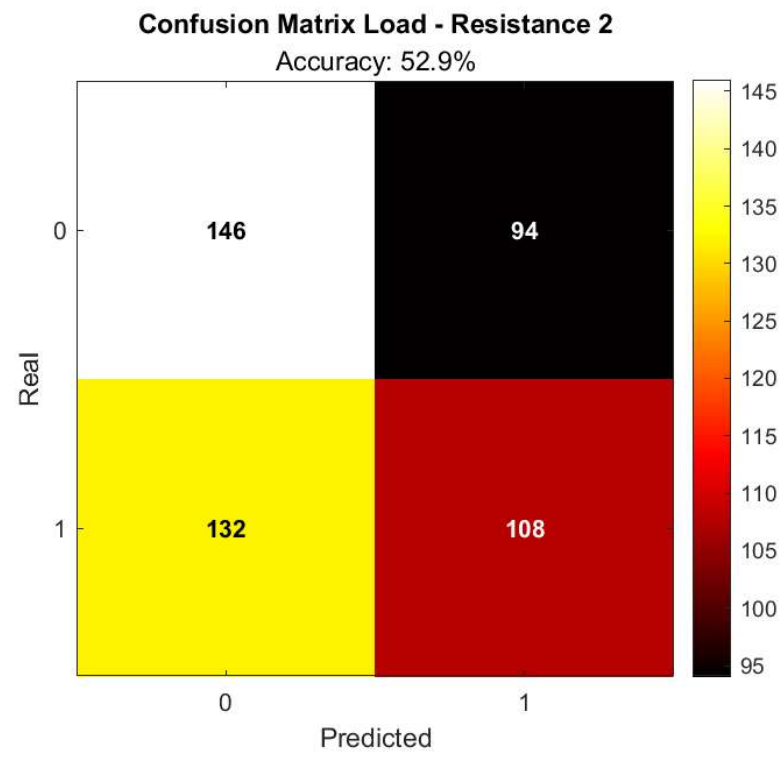


Figure 102: Confusion matrix for load, resistance 2.

In Table 8 are resumed the performances for every approach. For the first two approaches, the accuracy value is the one obtained following the discussed order of variable classification (see paragraphs 4.2 and 5.2); for this approach, we can choose the best accuracy for each variable, since they are classified all at once. We can observe that the resistance 2 gives the best results. This is probably caused by two facts:

- the resistance 2 has generally a higher value than the resistance 1. Hence, its variation can be better appreciated, and this reduces the influence of noise and instrument uncertainty;
- the resistance 2 is set in a position that is more stressed by the deformation imposed to the component. Consequently, it varies more as a result of the stress, and this makes the classification more reliable.

Classified variable	Accuracy first approach [%]	Accuracy second approach [%]	Accuracy third approach [%]
Angle	83.3	69.0	83.1
Broken branches	77.5	72.1	82.9
Load	61.7	63.8	52.9

Table 8: Accuracy of the third approach compared with the others.

We can observe an increase in the accuracy of the classification of the broken branches. This is an excellent result, because they are the variable most linked to the state of health of the element. Also, the accuracy for the angle is comparable to the one of the first approach. The load classification is not reliable, since the PDFs are too similar and the classification is influenced by the noise.

6.3 Remarks

Once again, the qualitative metrics give results similar to the other approaches.

Accuracy

The accuracy assumes values between 50 % and 85 %. It is high for the classification of the broken branches and of the angle, and it is low for the load.

Interpretability

Exactly like in the other approaches, the model is 100 % explainable and compact.

Robustness

The model is highly robust. The use of the Gaussian distribution is designed to reduce the influence of noise even more than already happens with the use of medians. Once more, the classification for a label is impossible without information about it.

Incrementality

The incrementality of the model is high, because every newly labelled record refines the Gaussian distribution making it more representative of the reality. As in the other approaches, the model can be easily updated with new data, but it is not automatic and a human intervention is needed to update the table containing the parameters of the family PDFs.

Efficiency

The efficiency of the model is excellent: both building the Gaussian distributions (model building) and the classification require about a second using a common computer.

Scalability

The model has a good scalability, thanks to the small amount of experimental data required and the low number of attributes.

All in all, the third approach gives good results. Comparing it with the other approaches, we can see that it maintains their strengths improving the performances.

Strengths

- The algorithm **preserves the strengths** of the first two approaches: the classification is reliable (not only for the broken branches but also for the angle), the training requires a small number of records, and both the training and the classification require a small amount of time.
- The **robustness** is the **highest**: the use of the Gaussian distribution strongly decreases the influence of the outliers.

- The **accuracy** is **high** for the classification of both angle and broken branches.
- **Information** on the dispersion of individual families is **preserved**.

Weakness

- The **load classification** is **not reliable**, but this is a common problem for all the approaches, whose methods do not value the slope variation given by a different load.

Reasoning on the second approach, we concluded that “in a Machine Learning approach, preserving the information about the range of resistance values is more important than fitting an ideal curve that the resistance should follow” (see paragraph 5.3). This hypothesis is confirmed by the performance of this approach. Indeed, since the Machine Learning is based on statistics, it is sensible that the finest approach from this point of view gives the best results.

Hence, we can conclude that the third approach is the best one, since it is the most robust and its performances are the best, except for the load whose classification in the other approaches is better but still not reliable.

6.4 Graphical Comparison between the First Three Approaches

Now that the three approaches have been presented, we can proceed with a graphical representation that allows for their comparison (see Figure 103). As with the third approach, the resistance-dependent score functions are shown for a given displacement value. In this case, the score functions are obtained using resistance 2, for a load angle of 15° , ID1 (no broken branches), and load 1 (L1), when the displacement is a half of its maximum (≈ 1.56 cm). To facilitate the comparison, the score function of the Medians approach has been modified. The original score function (i.e., the squared distance from the median point of the family) generates a parabola whose value increases moving away from the vertex. For this reason, this approach involves that the chosen family is the one with the lowest score. However, nothing changes in the comparison between family scores if one decides to make the score negative; consequently, the chosen family will be the one with the highest score. Furthermore, it has no influence on the comparison between family scores if the score for each point is increased by 1. In this way, a parabola is obtained whose vertex has a score of 1 and whose value decreases moving away from the vertex: this shape is closer to the score functions of the other approaches and therefore allows for a better comparison.

Hence, the new score function of a family for a given displacement in the second approach is:

$$score(dis) = 1 - (res - med_res(dis))^2$$

Where:

- $disp$ is the displacement of the interpolated test point;
- res is the resistance of the interpolated test point;
- $score(dis)$ is the score computed for that displacement, to be added to the total score of that family;
- $med_res(dis)$ is the median value of the resistance computed for that displacement, for the family whose score we are computing.

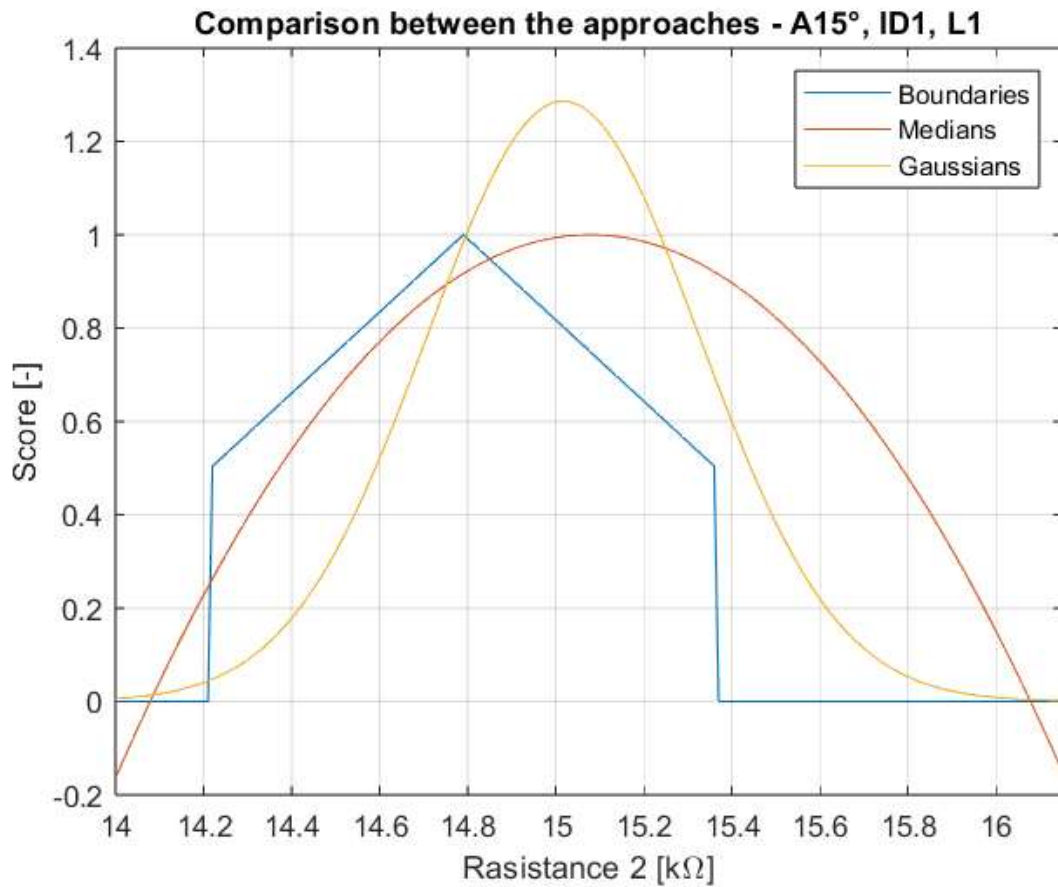


Figure 103: Graphical comparison between the first three approaches.

The resulting figure is very similar for all the combinations of variables. Therefore, some general observations can be made.

The peaks of the last two approaches (red and yellow) highlight that the mean value and the median value of resistance are not centred with respect to the range of all resistance values assumed by that family for that displacement value. This range is identified by the points where the score of the first approach (blue) presents a step of amplitude 0.5. Looking at Figure 28, we can conclude that this is not caused by the presence of outliers that excessively widen the range, but by hysteresis, which increases the mean resistance value over time.

This means that most of the records of this family, for this displacement value, show a resistance value greater than the peak of the red curve (at approximately 15.1 k Ω), far from the mean value of the range identified by the peak of the blue curve (at approximately 14.8 k Ω). This highlights a problem with the first approach: it only takes into account the range of resistance values, but not their distribution.

However, the second approach also presents a problem. Indeed, it can be noted that the score decreases too slowly for resistance values outside the range of all the assumed values (approximately between 14.2 k Ω and 15.35 k Ω). This highlights a weak point that had been identified in the second approach: it does not take into account the range of values assumed by the resistance.

Finally, as the analysis of the results had already shown, it is noted that the third approach (yellow) is the best: the score is high in the area where most of the tests are contained, and it rapidly decreases outside the range of the assumed values. It may seem that the score is too low for values between 14.2 k Ω and 14.8 k Ω , but this is justified by the fact that the number of tests with such resistance values for this displacement is low.

7 Fourth Approach: Ideal Model

7.1 Algorithm Explanation

As anticipated in the introduction, in addition to the three ML based approaches, also an ideal model of the sensor was constructed to compare completely different methods and obtain not only an experimental but also a theoretical validation of the obtained results. The base idea of the model is to characterize each individual segment of the electrical circuits present in the two arms of the sensor, in order to then construct a model of the circuit itself. In this way, it is possible to classify the equivalent resistance corresponding to each broken branches configuration by applying the elementary rules of series and parallel connections between resistances. To do this, a new experimental campaign was conducted: this allowed to derive the resistance associated with each segment of the circuit of each arm (see the Single Segment Characterization dataset in paragraph 3.3.2). From these, the ideal value of equivalent resistance was obtained for each broken branches configuration stored in the Crack Propagation dataset (see paragraph 3.3.1). Finally, this value was compared with the experimental one stored in the Crack Propagation dataset. Before proceeding with the explanation of this approach, it is necessary to establish a unique notation and recall the terms already used previously.

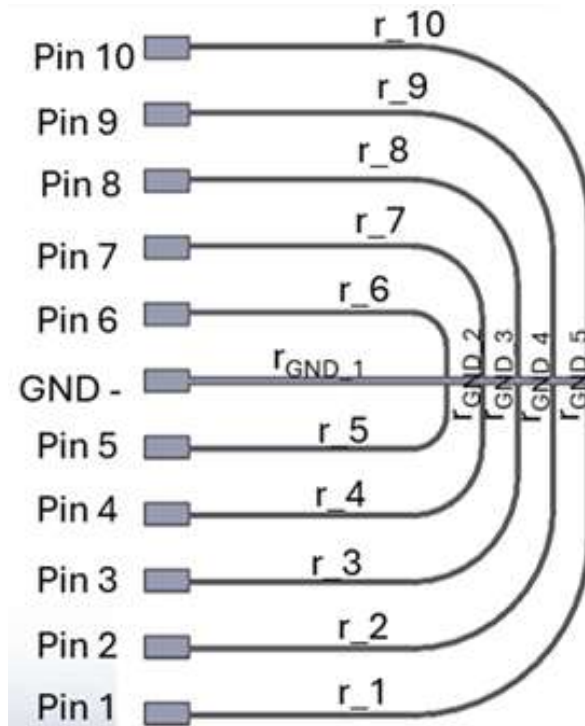


Figure 104: Sensor scheme.

- The two electrical circuits that constitute the sensor are called **“arms”**.
- The equivalent resistances of the two arms are called **“resistance 1”** and **“resistance 2”**.
- Each arm consists of 15 **“segments”** (see Figure 104):
 - 10 pairwise symmetrical segments, called **“branches”** and named **“r₁”, ... , “r₁₀”** (for example, r_6 is the resistance in the segment connecting GND to pin 6);
 - 5 central interconnection segments, named **“r_{GND_1}”, ... , “r_{GND_5}”** or, when all resistances are contained in a single array, **“r₁₁”, ... , “r₁₅”** (for example, r_{GND_1} is the resistance in the first of the five segments in which all the branches of the transducer converge).
- The dataset already used in the previous approaches for the simulation of the crack propagation is called **“Crack Propagation dataset”** or **“CP dataset”**.
- The dataset used only in this approach for the characterization of the individual segments of the arms is called **“Single Segment Characterization dataset”** or **“SSC dataset”**.
- Therefore, the **“IDs”** related to the two different datasets refer to different configurations:
 - in the CP dataset, as the IDs increase, the number of disconnected branches (simulating their failure) increases up to ID10; from ID11 up to ID20, it goes from the maximum number of disconnected branches (9) back to the configuration with all branches connected. For details, see paragraph 3.3.1;
 - in the single segment characterization dataset, the IDs refer to particular circuit configurations that allow to derive the resistance values of the 15 segments of each arm. Since up to this point IDs have been discussed with reference to those of the CP dataset, the IDs of the SSC dataset will be called **“C1”, ... , “C15”**, where “C” stands for “Configuration”. For details, see paragraph 3.3.2.
- Configurations C1 to C10 measure the resistances between a pin and GND, denoted as **“r_{1_GND}”, ... , “r_{10_GND}”** (for example, r_{10_GND} is the resistance measured between pin 10 and GND).
- Configurations C11 to C15 measure the resistances between two pins, denoted as **“r_{1_10}”, ... , “r_{5_6}”** (for example, r_{1_10} is the resistance measured between pin 1 and pin 10).

Let us now return to the construction of the ideal model. In the SSC dataset, the values of displacement and of resistances 1 and 2 over time are collected for the different configurations from C1 to C15. From there, by applying a transformation matrix, the resistance values of the individual segments can be derived. Looking at Figure 104, the ideal value of the equivalent resistance relative to each configuration is elementarily obtained. For example:

- $r_{10_GND} = r_{10} + r_{GND_1} + r_{GND_2} + r_{GND_3} + r_{GND_4} + r_{GND_5}$

- $r_{10_1} = r_{10} + r_1$

Proceeding for all configurations, the transformation matrix is obtained to pass from the resistance values of each individual segment to the equivalent resistance values measured in the various configurations. 15 configurations to derive 15 resistances: the problem has a unique solution.

$$\begin{pmatrix} r_{1_GND} \\ r_{2_GND} \\ r_{3_GND} \\ r_{4_GND} \\ r_{5_GND} \\ r_{6_GND} \\ r_{7_GND} \\ r_{8_GND} \\ r_{9_GND} \\ r_{10_GND} \\ r_{1_10} \\ r_{2_9} \\ r_{3_8} \\ r_{4_7} \\ r_{5_6} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \\ r_{14} \\ r_{15} \end{pmatrix}$$

However, since the acquired data are the equivalent resistances in the various configurations, and the resistance values for each segment ($r_1, \dots, r_{15}$) need to be obtained, the formula must be inverted. Let us denote:

- R_{meas} the vector of measured resistances (composed of $r_{1_GND}, \dots, r_{10_GND}, r_{1_10}, \dots, r_{5_6}$);
- T the transformation matrix;
- R_{seg} the vector of the resistances of the individual segments (composed of $r_1, \dots, r_{15}$).

The formula to derive R_{seg} is therefore:

$$R_{seg} = T^{-1} * R_{meas}$$

Once R_{seg} is obtained, the equivalent resistance values for each ID of the CP dataset can finally be determined. Again, elementary knowledge regarding the parallel and series connections of ideal resistances is applied. Recalling that in the CP dataset all the pins are connected to the positive pole, the circuit scheme to understand the IDs is the following.

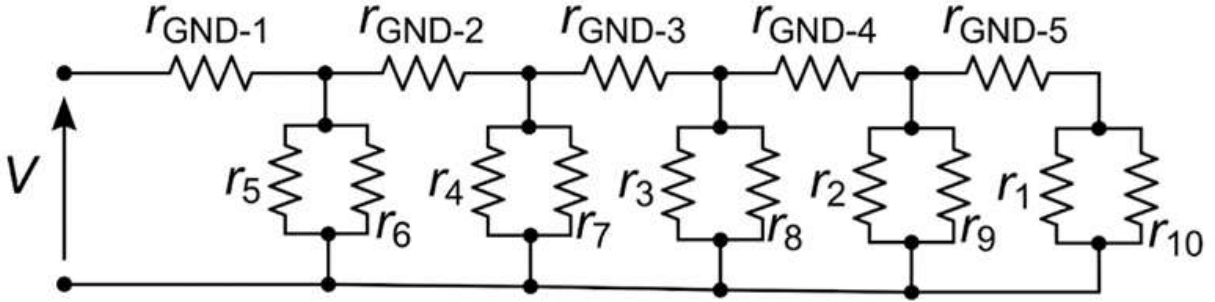


Figure 105: Electrical scheme of the sensor.

For example, ID10 refers to the configuration in which pin 10 and GND are connected to each other, passing through branch 10 and all the central connection segments (all the other pins are disconnected). Therefore, calling this equivalent resistance R_{ID10} , it should ideally be:

$$R_{ID10} = r_{10} + r_{11} + r_{12} + r_{13} + r_{14} + r_{15}$$

Decreasing the ID number, for ID9 the equivalent resistance between GND and the parallel combination of pins 9 and 10 is computed (pins 1 to 8 disconnected). Therefore, calling this equivalent resistance R_{ID9} , it should ideally be:

$$R_{ID} = r_9 // (r_{10} + r_{11}) + r_{12} + r_{13} + r_{14} + r_{15}$$

Analogously, all the equivalent resistance values for each ID can be calculated.

7.2 Results and Issues

Now, the obtained resistance value must be compared with the measured equivalent resistances for each ID and collected in the CP dataset. Differently from usual, it was decided not to present the results on the displacement-resistance plane: rather, the significant phenomena are more highlighted by showing the average resistance values for each ID. More precisely, first of all, the median curves of resistance with respect to displacement were computed (as in the second approach) and then the average value of these curves was obtained. In this way, the uncertainty due to the use of a single test is reduced and all the records are used to get the expected resistance value for each ID. The median approach was chosen because it not only has good robustness but is also the one that allows a more immediate visualization: it associates to each displacement value a single resistance value and not a range of resistances (or their distribution). Showing only an average value and not the entire resistance profile is permissible when comparing the different IDs because, as seen in the previous approaches, as the displacement varies, the resistance value changes insignificantly compared to the difference caused by a different configuration. In other words, almost perfectly constant lines are obtained with respect to the range of values considered for the resistance; hence, the displacement axis is unnecessary in this case. Therefore, applying the ideal model to resistance 2, with a loading angle of 0° in the unloading phase (load = 0), the result shown in Figure 106 is obtained.

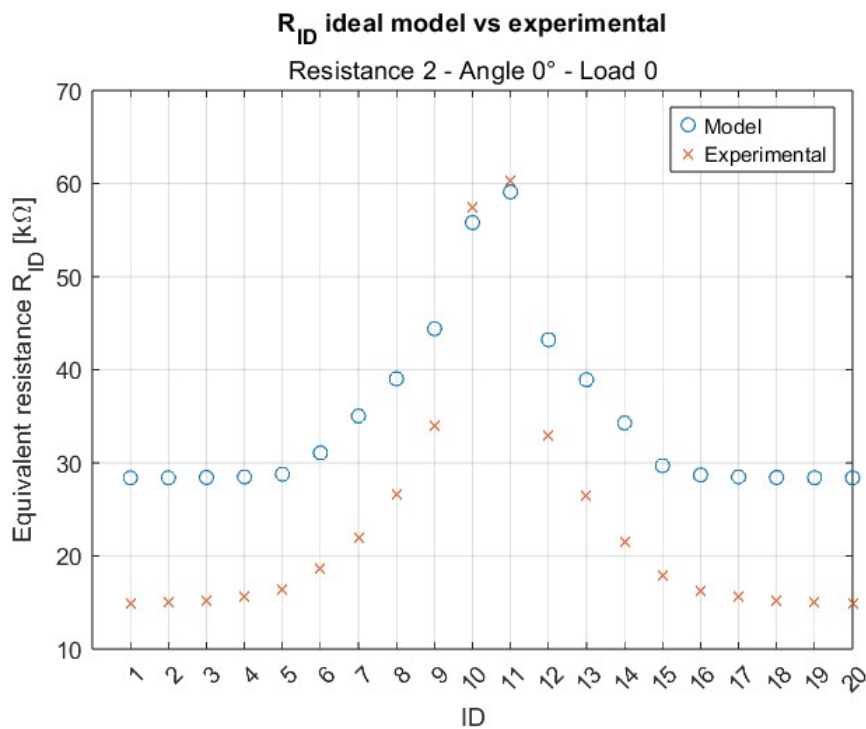


Figure 106: Comparison between ideal model and experimental tests.

As can be noticed, for all IDs except ID10 and ID11, there is a difference of approximately 15 k Ω between the value obtained through the ideal model and the experimentally measured one. The same problem occurs for other configurations of angle and load, and for resistance 1.

To investigate the causes of this phenomenon, two hypotheses were formulated by looking at the mechanics of the problem, concerning the irreversible phenomena of plasticity and hysteresis.

When a force is applied to the gear, the strain induces a deformation on the material which causes the conductive particles of the composite to move closer or further apart. As a result, the electric circuit shows a different equivalent resistance and this information can be processed. Then, when the applied force returns null, the elastic properties of the component cause its return to its regular form, but the distribution of the conductive particles is no more the previous one. This uncertainty is mitigated by conducting multiple experimental tests. However, it is likely that plastic phenomena are present, which do not allow the complete return of the specimen to its unloaded configuration, or allow it on a macroscopic scale but not a microscopic one. Thus, irreversible deformations can be generated in the material, leading, as seen in the graph, to an overall increase in resistance.

Secondly, by looking at the resistance profile over time when the specimen is subjected to multiple loading/unloading cycles, the presence of hysteresis can be noticed. In particular, it is found that the first cycle applied after a certain rest period leads to resistance values far from those obtained when the component has been recently stressed (see Figure 107, first cycle in particular). Furthermore, it can be noticed that even during the permanence in a certain position, whether loaded or unloaded, the resistance value changes over time, despite the displacement remaining fixed. This viscoelastic behaviour is typical of CPC materials.

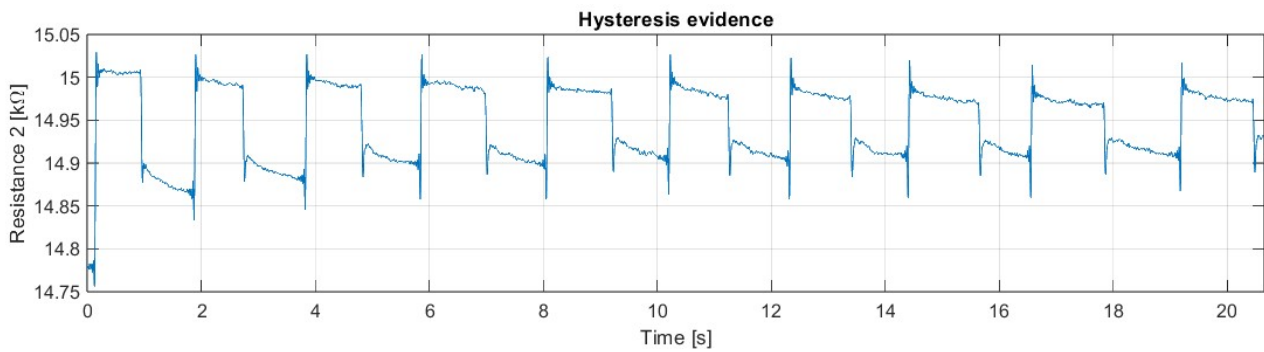


Figure 107: Hysteresis evidence in time domain.

Since these phenomena were already suggested before performing the experimental tests, the tests performed for the two datasets, CP and SSC, were not executed after a long period of time from one another, nor by completing one entire dataset before the second. On the contrary, the tests related to the two datasets were performed alternately. Evidently, this precaution is not sufficient due to the high number of irreversible and perhaps unknown phenomena involving the component. Note: these irreversibilities had also been highlighted, but not discussed, in the paragraph about the CP dataset (see paragraph 3.3.1) and in the first approach (see paragraph 4.1.2): here we noted that sets of experimental tests performed even a short time apart still led to different groupings in the displacement-resistance curves (see Figure 32 about grouping the families with the same stress angle).

In conclusion, the built ideal model cannot be used for the classification. The irreversibility phenomena affecting the component must be compensated by designing a more complex model. Now, the need for Machine Learning approaches can be fully understood: by dealing the problem as a black box, while still paying attention to the physical phenomena involving it, thanks to the first three ML approaches it was possible to extract information and build applicable classification models.

Having examined in detail all the approaches designed to address the problem, it is possible to proceed with the design of a final tool that combines the three Machine Learning models (the ideal model must not be considered because of its unreliability).

To see how the ideal approach is implemented in Matlab, see the script *Build_ideal_medians_table.m*.

8 Final Tool

By combining the first three ML approaches, it is possible to design a tool that leverages the strengths of each. This combining methodology is well established in the field of Machine Learning: Random Forest is a prominent example. Random Forest is an algorithm that combines multiple decision trees, which are models of lower complexity, to achieve a final algorithm whose performances surpass the ones of any single tree in terms of both accuracy and robustness.

Analogously, the objective is to develop an overall algorithm that, given a simultaneous measurement of resistances 1 and 2, compares the classifications provided by each of the three primary algorithms and estimates the most probable classification for angle, broken branches, and load. To achieve this, it is necessary to account for the classification reliability of each approach regarding each specific variable.

8.1 Explanation

The final tool executes the following sequential steps to classify a given test acquisition (from now on, the three approaches are also referred to as “sub-models”, and their names are abbreviated as *Boundaries*, *Medians*, and *Gaussians*, respectively).

- Construct the training and test datasets for all three sub-models (see the training section for each approach), ensuring that the records used for training are the same for all the sub-models and that resistances 1 and 2 refer to the same record.
- Apply the three sub-models to both resistance curves (1 and 2) of the test acquisition to classify the number of broken branches. For the first two approaches (*Boundaries* and *Medians*), the angle and load are assumed to be unknown. Conversely, the third approach (*Gaussians*) inherently classifies also the angle and the load simultaneously. Disregarding the angle and the load at this stage, this step gives six independent estimates for the number of broken branches: each sub-model generates two classifications, one for each resistance.
- Combine the six sub-model estimates to derive the most reliable classification for the number of broken branches. To do this, the accuracy values for the classification of broken branches obtained during the validation of the individual algorithms are taken into account (see the tables in the result section for each algorithm). These values are reported below:

- Boundaries Accuracy, Resistance 1: 54.8 %;
- Boundaries Accuracy, Resistance 2: 83.3 %;
- Medians Accuracy, Resistance 1: 59.2 %;
- Medians Accuracy, Resistance 2: 69.0 %;
- Gaussians Accuracy, Resistance 1: 66.7 %;
- Gaussians Accuracy, Resistance 2: 82.9 %.

It is reasonable to assume that the classifications given by the *Boundaries* and *Gaussians* sub-models using resistance 2 should be weighted more heavily than those with lower accuracy. Although this may appear trivial, it introduces non negligible risks.

To illustrate this, let us consider a simplified scenario where three sub-models (A, B, and C) give a single output each, with respective accuracies of $P_A = 80\%$, $P_B = 60\%$, and $P_C = 60\%$.

- If all sub-models yield different results, the final output of the tool must clearly coincide with the output of sub-model A, as it is the most reliable.
- If A provides the same result as one of the other sub-models, the final tool must evidently give that output.
- If A provides a specific result (a), while B and C yield an identical alternative result ($b \neq a$), the choice of the final output becomes problematic. Specifically: is it more probable that A classified correctly while B and C are incorrect, or vice versa?

The probability ($P_{A, \text{right}}$) that a is correct and b is incorrect is formulated as:

$$P_{A, \text{right}} = P_A * (1 - P_B) * (1 - P_C) = 0.8 * (1 - 0.6) * (1 - 0.6) = 0.128$$

Conversely, the probability ($P_{BC, \text{right}}$) that a is incorrect and b is correct is:

$$P_{BC, \text{right}} = (1 - P_A) * P_B * P_C = (1 - 0.8) * 0.6 * 0.6 = 0.072$$

Evaluating these expressions yields a higher probability that the output of A is correct compared to the collective output of B and C ($P_{A, \text{right}} > P_{BC, \text{right}}$). Therefore, adopting a rigorous probabilistic approach, we conclude that the output of A must be selected even when B and C are in consensus and contradict A.

Given that six distinct outputs must be combined in the actual application, it is worth investigating whether a simpler comparing method exists. An initial hypothesis might involve a weighted average of the outputs based on accuracy. Let us examine how this applies to the previous example.

Since classification relies on discrete outputs, certain adjustments are required. Assume that output a (from sub-model A) equals 0 and output b (from sub-models B and C) equals 1. If the weighted average of the sub-models is a non-integer value, it is rounded to the nearest integer between 0 and 1. The final tool output would be computed as:

$$Output = \frac{a * P_A + b * P_B + b * P_C}{P_A + P_B + P_C} = \frac{0 * 0.8 + 1 * 0.6 + 1 * 0.6}{0.8 + 0.6 + 0.6} = 0.6 \rightarrow 1$$

This leads to a result that directly contradicts the rigorous probabilistic approach: the output of sub-models B and C overrides the one of A. Indeed, while weighted averages are highly effective for continuous variables, they are fundamentally unsuitable for discrete classification tasks. Alternative averaging methods were evaluated, but all exhibited the same intrinsic limit.

Consequently, the rigorous probabilistic approach is selected for the implementation of the final tool.

Thus, each potential classification output (label) for the number of broken branches is assigned a score proportional to the probability that the sub-models yielding that specific output are correct while all dissenting sub-models are incorrect. The scoring formula is defined as follows:

$$Score(label) = \prod_{i=1}^{N_{lab}} Acc_i * \prod_{j=1}^{N_{not-lab}} Acc_j$$

Where:

- N_{lab} is the number of sub-models whose output matches the label for which the score is being computed (treating models applied to different resistances as separate sub-models);
- Acc_i is the accuracy of these sub-models in classifying broken branches (which varies depending on the specific resistance curve applied);

- $N_{not-lab}$ is the number of sub-models whose output differs from the label for which the score is being computed (again, treating models applied to different resistances as separate sub-models);
- Acc_j is the accuracy of these dissenting sub-models in classifying broken branches.

The number of broken branches returned by the final tool corresponds to the label that achieves the highest probability score.

- Now that the classified value for broken branches has been established, the first two sub-models (*Boundaries* and *Medians*) are re-applied to both resistance values, incorporating the classified number of broken branches as a known input parameter. The third sub-model (*Gaussians*) is not re-applied, as its formulation remains invariant regardless of which variables are known (see paragraph 6.1.2).
- Mirroring the procedure used for the broken branches, the six resulting outputs for the angle classification are combined to determine the final value selected by the tool. Of these six outputs, the two associated with the *Gaussians* sub-model are picked from the initial execution (Step 2), whereas the remaining four are obtained during the previous step.
- Following the same logic as the angle classification, the *Boundaries* and *Medians* sub-models are applied to both resistance curves to classify the load, this time incorporating both the angle and the number of broken branches determined in the previous steps as known inputs.
- Finally, the six load outputs obtained are combined using the same probabilistic approach to derive the final value selected by the tool.

To see how the final tool is implemented on Matlab, see the script *Final_tool.m* in the appendix.

8.2 Example

An illustrative example is presented below to clarify the operational workflow of the final tool. By sequentially classifying the broken branches, the angle, and finally the load, the percentage accuracy values used are those reported in Table 9 (see the tables in the result section for each algorithm).

Variable	Boundaries		Medians		Gaussians	
	Res. 1	Res. 2	Res. 1	Res. 2	Res. 1	Res. 2
Angle	77.5	57.9	72.1	68.8	73.8	83.1
Broken branches	54.8	83.3	59.2	69.0	66.7	82.9
Load	57.9	61.7	54.0	63.8	52.9	52.9

Table 9: Accuracy values of all the approaches.

The classification example is applied to a test sample characterized by a stress angle of 15° , 3 broken branches, and a loading phase (load = 1). Each sub-model gives the outputs collected in Table 10, which also includes the final values selected by the final tool.

Variable	Real value	Bound. res. 1	Bound. res. 2	Medians res. 1	Medians res. 2	Gauss. res. 1	Gauss res. 2	Final tool
Angle	0°	0°	15°	15°	0°	0°	0°	0°
Broken branches	3	1	3	0	3	4	3	3
Load	1	0	1	0	1	0	1	1

Table 10: Results of an example using the final tool.

The classified values generated by the final tool are derived via the algorithmic process described in the previous section. For the classification of the number of broken branches, the following probability scores are obtained for each target label:

0:	0.0008	5:	0
1:	0.0007	6:	0
2:	0	7:	0
3:	0.0293	8:	0
4:	0.0011	9:	0

Consequently, the final tool classifies a value of 3 broken branches. This estimated value is subsequently used into the first two approaches for the classification of the stress angle, yielding the following probability scores for the available labels:

0°: 0.0384

15°: 0.0013

Therefore, the tool assigns the 0° label to the angle variable, using it as an input constraint for the classification of the load, which results in the following scores:

0: 0.0108

1: 0.0190

As a result, the final tool assigns the label 1 to the load.

As demonstrated by this case study, although discrepancies naturally arise among the individual sub-models, the probabilistic cross-comparison approach enables the final tool to successfully mitigate local errors and converge on a robust, highly reliable global output.

8.3 Results

As for the previous approaches, confusion matrices can be built for each variable. However, in this case the distinction regarding which resistance curve has been used is no longer applicable, nor is the distinction concerning which variables were assumed to be known or unknown. Consequently, only three confusion matrices are obtained: one for each target variable.

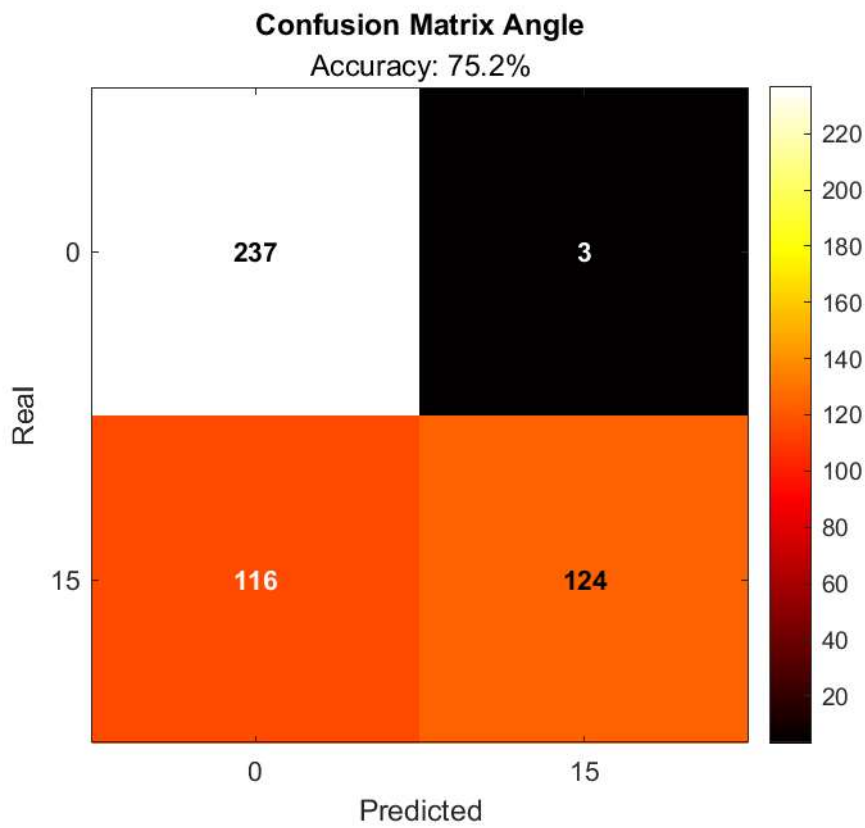


Figure 108: Confusion matrix for angle.

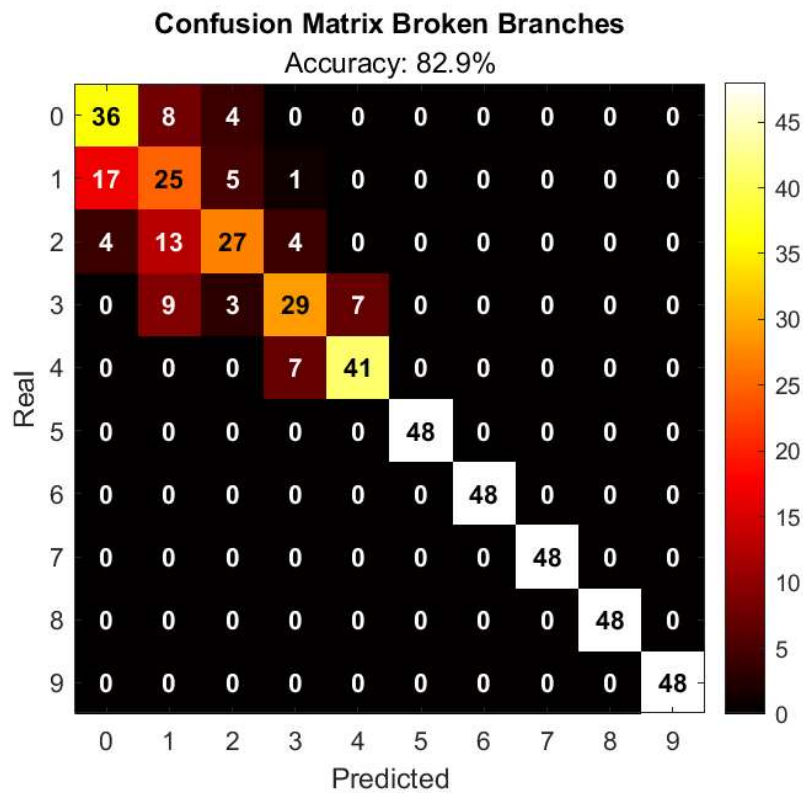


Figure 109: Confusion matrix for broken branches.

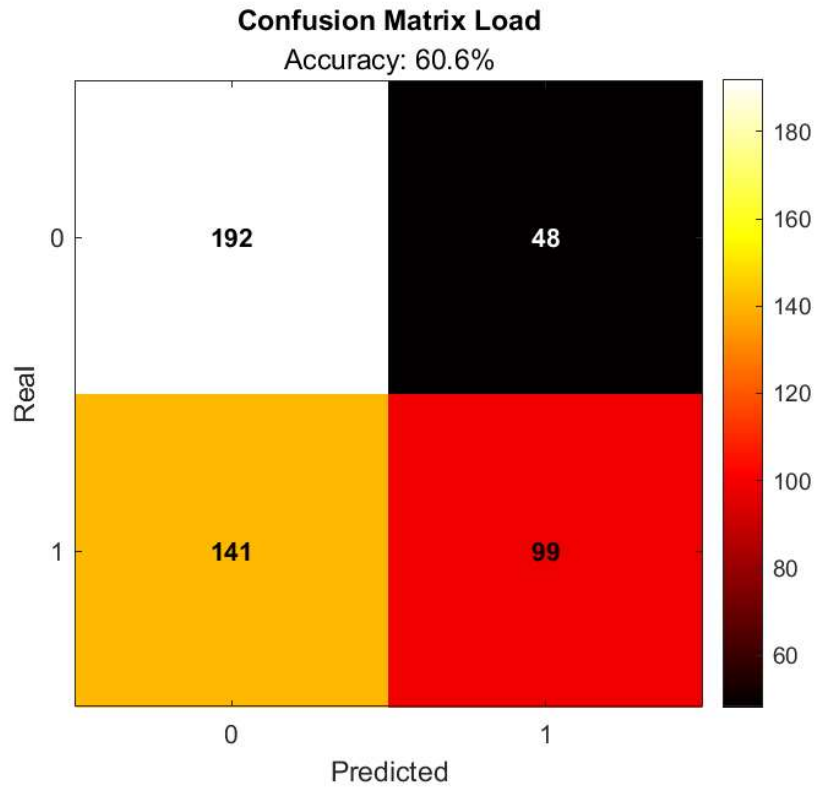


Figure 110: Confusion matrix for load.

As can be observed, the accuracy values reflect the top performances among the various individual sub-models. This confirms the achievement of an integrated global model that not only delivers the highest reliability for each variable but also exhibits significant robustness due to the cross-comparison of the different sub-models. The overall accuracy values achieved by the final tool are reported below:

- Angle: 75.2 %;
- Broken branches: 82.9 %;
- Load: 60.6 %.

8.4 Remarks

Accuracy

The accuracy ranges between 60 % and 85 %, maintaining a high value for each variable compared to the classifications of the individual sub-models.

Interpretability

Exactly like in the other approaches, the model is 100 % explainable and compact.

Robustness

The model is highly robust because it compares three similar yet distinct approaches. However, a lack of data regarding a specific label renders its classification impossible.

Incrementality

The incrementality of the model is medium-high: some sub-models (*Medians* and *Gaussians*) are refined by the addition of new data, whereas the *Boundaries* sub-model risks being influenced by outliers when the data volume is large.

Efficiency

The efficiency of the model is excellent: both building the sub-models and the classification require about a second using a common computer.

Scalability

The model has a good scalability, thanks to the small amount of experimental data required and the low number of attributes.

Strengths

- The final tool **preserves the strengths** of the single approaches: the training requires a small number of records, and both the training and the classification require a small amount of time.
- The **robustness is high**.
- The **accuracy is the highest** possible with these approaches.
- **Information** on the dispersion of individual families **is preserved**.

Weakness

- The **load classification is less reliable** than the other variables: this is a common problem for all the approaches; however, the maximum possible accuracy value has been achieved for this variable as well.

9 Conclusions

9.1 Research Path

9.1.1 Initial Problem

The use of multi-material structures is attracting researchers in industry and academia with the goal to exploit their different properties and to obtain more performant and reliable multifunctional structures. In this frame is included the development of Internet of Things (IoT) supporting natively sensorized mechanical components. The design of a sensorized landing gear for an eVTOL fits into this research line, combining the IoT field with FDM (Fused Deposition Modeling) and innovative materials, as the CPCs (Conductive Polymer Composites). Naturally, to design such a component, prototypes must first be realized and characterized. Therefore, the characterization of the behaviour and the properties of the sensor embedded within the landing gear constituted the initial problem and the main objective of this thesis.

Since the manufacturing of new prototypes took place concurrently with the analysis of data obtained from the first ones, the characterization was not limited to the study of a definitive component. Instead, it became a mutual collaboration with the laboratory staff for the progressive improvement of the several manufactured prototypes.

Thus, the initial problem consists of two aspects:

- the design of the mechanical component, specifically the electrical circuit (made of innovative CPC material) that acts as a strain gauge;
- the characterization of the sensor, studying what information can be derived from it and which experimental tests are necessary to obtain it.

9.1.2 Proposed Technological Solution

The proposed geometry of the specimen changed compared to the initial one (see paragraph 3.1.4 about the alternative specimens), but the main idea remained the same. It consists of a square base to be inserted into the aircraft fuselage and a curved beam connecting to the skid. The two electrical circuits that constitute the sensor are embedded within the base and communicate with the outside

through copper electrodes. To manufacture the specimen of such a component, FDM is used, employing PLA for the main body and a CPC for the strain gauges.

The materials of the specimen differ from those of the final component. For this reason, the sensor characterization method must be sufficiently generalized so that it can be applied to components with different geometries and materials. Consequently, two possible approaches have been proposed:

- through Machine Learning algorithms, which treat the problem as a black box without considering its geometry and physics, but rely only on quantities that can also be obtained for sensors with different geometries and materials (like resistance and displacement);
- through an ideal model of the component, ensuring that this does not turn out to be too specific for the specimen and cannot subsequently be applied to the final component.

9.1.3 Working Hypotheses

The material and the manufacturing process of the specimen leave room for numerous irreversibility phenomena; therefore, a good model should be developed specifically for the specimen and would need to undergo some variations to be applied to the final component as well. Consequently, the intention was to create a simple and general model of the strain gauge to observe its reliability, but attention was focused on ML algorithms from the beginning. This was because there were two hypotheses regarding the outcome of the ideal model:

- if the model proved to be reliable, ML algorithms could be integrated to derive further information from the sensor, and both the model and the experimental tests would serve for training and validation;
- if the model proved to be unreliable, ML algorithms would become the only way to derive information from the sensor, having only the experimental tests available for training and validation.

In any case, it was necessary to perform experimental tests on the specimens. As mentioned, the measured quantities must also be obtainable for sensors with different geometries and materials. Therefore, the following quantities were selected:

- total resistance of the two circuits constituting the strain gauges;

- displacement of the beam end;
- stress angle of the force imposing the displacement;
- integrity of the strain gauges, evaluated through the number of uninterrupted circuit segments;
- time corresponding to each measurement.

It was hypothesized to work with these quantities, leaving room for potential variables to be added or removed.

Once the results from the initial experimental tests were obtained, the most appropriate ML algorithms would be developed. The objective of this step was to obtain an algorithm that, by training on a certain number of experimental tests, receives the resistance profile of the strain gauges over time (or with respect to displacement) as input during operation, and returns in output as much information as possible regarding stress angle, strain gauge integrity and other parameters.

9.1.4 Execution of the Work and Verification of Results

As described in paragraph 3.3.3 about the alternative dataset, an initial experimental campaign was conducted to study the behaviour of the specimen and to make evaluations for future campaigns. A first important aspect that was noticed is the non-biunique relationship between the imposed displacement and the measured resistances; this particular fact shifted the focus toward ML algorithms. A further incentive was the high dispersion of the tests, mainly due to the various irreversibility phenomena affecting the specimen and the unpredictable imperfections caused by its manufacturing process. Consequently, a simple fitting curve would have been unreliable and poorly applicable.

Therefore, the first two ML approaches were designed: one based on regions of the displacement-resistance plane, and the other based on the distance from median curves on the same plane. This highlighted several issues to be improved:

- the number of intact branches is particularly relevant; hence, a method is needed to simplify their characterization while avoiding the destruction of the specimens (this will be achieved by using jumpers to simulate branch failure);

- the difference in resistance recorded by varying only the loading angle and the loading/unloading phase is small compared to its absolute value. Consequently, this reinforces the idea of using a single specimen to perform all the tests, avoiding the introduction of additional uncertainty due to the use of different specimens;
- the manual application of force via a hand crank makes the process slower and less repeatable (it will be replaced by the use of a pneumatic piston);
- better geometries may exist for the electrical circuits of the strain gauges (specifically, the two resistors will be relocated to the areas of maximum stress on the component, identified through FEM analysis).

By implementing the described improvements, the two experimental campaigns on which subsequent studies were based were conducted, leading from the alternative dataset to the Crack Propagation and the Single Segment Characterization datasets (see paragraph 3.3). Thus, the strengths and weaknesses of each designed approach were analyzed, leading to the Gaussian-based approach, the objective of which is to combine the advantages of the two previous approaches while limiting their disadvantages. Meanwhile, the ideal model was completed, and it was found, as expected, that greater complexity is required for it to be reliable.

Analyzing the three ML approaches used, it was decided to combine them to obtain a final tool featuring the following aspects:

- it receives the resistance values of the two strain gauges with respect to the displacement as input, and outputs the values of the stress angle, the number of compromised circuit branches, and whether the system is in the loading or unloading phase;
- it is based only on resistance and displacement values, making it general enough to be applied to components with different geometries and materials (obviously requiring new experimental tests for training and validation);
- it exhibits the best available performances among the three approaches;
- it is more robust than any single approach taken individually;
- it can be refined with the addition of new experimental data.

9.1.5 Final Comment

The research path highlighted the potential of Machine Learning to complement the modelling of mechanical components. Where an overly simple ideal model proved unreliable and an excessively complex one would not be applicable to the final component, a ML approach allows for getting information and studying the specimen behaviour while maintaining the necessary generality to be applied to similar but not identical mechanical elements. Furthermore, the versatility of FDM was highlighted, as it allowed for the manufacturing of the component with a native sensor embedded inside it.

9.2 Future Perspectives

9.2.1 Applications and Provided Advantages

The best performances of the sensor concern its structural health, monitored via the number of intact branches within the strain gauges. Consequently, it fits perfectly into the field of the Internet of Things, providing the major advantage of knowing the real state of health of the component. In this way, it can be replaced only when truly necessary, preventing unexpected failures, reducing costs, and increasing sustainability by avoiding premature replacements.

Furthermore, thanks to the generality of the used approach, the application of the final tool is possible for any other component to be sensorized in which resistance-displacement curves can be obtained.

9.2.2 Issues to be Further Investigated

As mentioned, the main point of future interest would be to observe the application of the final tool to other mechanical elements with native sensors, or directly to the definitive component (once manufactured) rather than to its specimen. Additionally, to improve the health monitoring aspect, it would be equally interesting to combine this tool with predictive algorithms.

A validation of the ML models provided not only by experimental tests but also by a reliable model would be more robust, as it would compensate for uncertainties due to noise, manufacturing defects, and measuring instruments. This purpose is similar to the goal of the Physics-Informed Neural Network (paragraph 2.2.3). In the specific case of the specimen studied in this thesis, a model accounting for the observed hysteresis would be necessary.

Since experimental tests always involve a cost, a study to understand how much data is required to achieve a reliable classification would undoubtedly be useful.

It is probable that more information can be extracted from the resistance-displacement profile than what was obtained through our approaches. For instance, the force applied to the beam end, which could also be easily acquired via a load cell. Therefore, determining how many and which additional quantities can be derived remains an issue to be further investigated.

10 Appendix: Matlab Scripts

This appendix presents the Matlab scripts used to implement the described approaches and to build the confusion matrices. A total of 17 scripts is presented, categorized as follows:

- scripts for model creation (training): scripts 1 to 4. Starting from the experimental dataset containing displacement and resistance acquisitions over time, these scripts build the tables that constitute the models for the various approaches;
- scripts to build the confusion matrices for the first three approaches: scripts 5 to 7. These scripts illustrate the matrix generation process for broken branches classification. Scripts for building matrices with different variables or considering alternative combinations of known and unknown variables are omitted to avoid redundancy, as they require only minor modifications to a few lines of code and specific for-loops;
- final tool: script 8. For this part, rather than the code for confusion matrix construction, the script for the classification of a single test is provided, as it represents the actual final tool resulting from this thesis. Similarly, to build the corresponding confusion matrices, it is sufficient to integrate the appropriate for-loops to iterate the classification across all the tests contained within the test tables;
- auxiliary functions used for both testing and training: scripts 9 to 16;
- script to implement the ideal model: script 17.

10.1 Build_training_and_test_table.m

This script gets the dataset, applies segmentation to split it into single records and distribute them into two tables: one for training and one for validation (test). The function used for the segmentation is *split_test* (script 9).

```
clear
close all
clc

Dataset = importdata("Dataset.mat");

column_names = {'Angle_0-Load_0', 'Angle_0-Load_1', ...
    'Angle_15-Load_0', 'Angle_15-Load_1'};
row_names = {'ID1', 'ID2', 'ID3', 'ID4', 'ID5', 'ID6', 'ID7', 'ID8', 'ID9', 'ID10', ...
    'ID11', 'ID12', 'ID13', 'ID14', 'ID15', 'ID16', 'ID17', 'ID18', 'ID19', 'ID20'};

Table_training_res = cell2table(cell(20, 4), 'VariableNames', ...
    column_names, 'RowNames', row_names);

Table_test_res = cell2table(cell(20, 4), 'VariableNames', ...
    column_names, 'RowNames', row_names);

perc_test = 0.2; % 20 % of the data are kept apart for the validation
rng(1) % the random seed is the same for all the scripts: consequently, all
    % the talbes are built using the same samples

% the Threshold_matrix contains the threshold values to well split the
% record into 10 cycles. This value is often 0.08, but not always

Threshold_matrix = 0.08*ones(20,6); % dimensions: 20 id x (2 ang * 3 n)
Threshold_matrix(2, 1) = 0.5;
Threshold_matrix(3, 1) = 0.5;
Threshold_matrix(3, 2) = 0.5;
Threshold_matrix(4, 1) = 0.5;
Threshold_matrix(4, 2) = 0.5;
Threshold_matrix(4, 3) = 0.5;
Threshold_matrix(5, 2) = 0.5;
Threshold_matrix(6, 1) = 0.5;
Threshold_matrix(6, 2) = 0.5;
Threshold_matrix(7, 2) = 0.92;
Threshold_matrix(7, 3) = 0.5;
Threshold_matrix(8, 1) = 0.5;
Threshold_matrix(8, 2) = 0.5;
Threshold_matrix(8, 3) = 0.5;
Threshold_matrix(9, 1) = 0.5;
Threshold_matrix(9, 2) = 0.5;
Threshold_matrix(9, 3) = 0.5;
Threshold_matrix(10, 1) = 0.5;
Threshold_matrix(10, 2) = 0.5;
Threshold_matrix(10, 3) = 0.85;
Threshold_matrix(11, 1) = 0.5;
Threshold_matrix(11, 2) = 0.5;
Threshold_matrix(11, 3) = 0.5;
Threshold_matrix(12, 1) = 0.5;
Threshold_matrix(12, 2) = 0.5;
Threshold_matrix(12, 3) = 0.5;
Threshold_matrix(13, 1) = 0.5;
Threshold_matrix(13, 2) = 0.5;
```

```

Threshold_matrix(13, 3) = 0.5;
Threshold_matrix(14, 1) = 0.5;
Threshold_matrix(14, 2) = 0.5;
Threshold_matrix(15, 1) = 0.5;
Threshold_matrix(15, 2) = 0.5;
Threshold_matrix(15, 3) = 0.5;
Threshold_matrix(16, 1) = 0.5;
Threshold_matrix(16, 2) = 0.5;
Threshold_matrix(17, 2) = 0.5;
Threshold_matrix(17, 3) = 0.5;
Threshold_matrix(18, 2) = 0.5;
Threshold_matrix(19, 1) = 0.5;
Threshold_matrix(19, 2) = 0.5;
Threshold_matrix(19, 3) = 0.5;
Threshold_matrix(20, 1) = 0.5;
Threshold_matrix(20, 2) = 0.5;
Threshold_matrix(20, 3) = 0.5;
Threshold_matrix(2, 6) = 0.4;
Threshold_matrix(4, 6) = 0.3;
Threshold_matrix(6, 6) = 0.3;
Threshold_matrix(7, 6) = 0.3;
Threshold_matrix(8, 6) = 0.3;
Threshold_matrix(9, 6) = 0.3;
Threshold_matrix(10, 6) = 0.3;
Threshold_matrix(13, 6) = 0.3;
Threshold_matrix(14, 6) = 0.3;
Threshold_matrix(15, 6) = 0.3;
Threshold_matrix(16, 6) = 0.2;
Threshold_matrix(17, 5) = 0.3;
Threshold_matrix(19, 6) = 0.25;

% through the for cycles for each variable, all the tests for each
% combination of variables are stored into the same cell of
% Table_boundaries

for a = 0:1 % angle index
    columns_load = cell(20, 2);
    columns_test = cell(20, 2);
    for id = 1:20

        threshold_1 = Threshold_matrix(id, a*3+1);
        threshold_2 = Threshold_matrix(id, a*3+2);
        threshold_3 = Threshold_matrix(id, a*3+3);

        Dataset_condition_1 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_1;
        Table_condition_1 = split_test(Dataset_condition_1, threshold_1);

        % since not all the combination of variables in the dataset contain
        % the same number of cycles, some of them are discarded to have an
        % equal test distribution (this sometimes happens twice for
        % condition 2 (n_2))

        while height(Table_condition_1) ~= 20
            Table_condition_1 = Table_condition_1(1:end-2,:);
        end
        Dataset_condition_2 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_2;
        Table_condition_2 = split_test(Dataset_condition_2, threshold_2);
        while height(Table_condition_2) ~= 20
            Table_condition_2 = Table_condition_2(1:end-2,:);
        end
        Dataset_condition_3 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_3;
        Table_condition_3 = split_test(Dataset_condition_3, threshold_3);
        while height(Table_condition_3) ~= 20
            Table_condition_3 = Table_condition_3(1:end-2,:);
        end
    end
end

```

```

end
Table_condition = [Table_condition_1; Table_condition_2; Table_condition_3];
Time = Table_condition.Time;
Displacement = Table_condition.Displacement;
Resistance_1 = Table_condition.Resistance_1;
Resistance_2 = Table_condition.Resistance_2;

tot_tests = height(Table_condition);

for l = 0:1 % load (0 = unloading, 1 = loading)
    tests = cell(1, tot_tests);
    counter_tests = 0;
    for k = 1:tot_tests
        if Table_condition.Load(k) == 1
            resistance_1 = Resistance_1{k};
            resistance_2 = Resistance_2{k};
            displacement = Displacement{k};
            counter_tests = counter_tests + 1;
            % choosing which resistance use
            tests{counter_tests} = [displacement, resistance_2];
        end
    end
    tests = tests(1:counter_tests);

    % keeping some tests apart for the validation

    tot_kept_for_testing = round(counter_tests * perc_test);
    idx_random = randperm(counter_tests, tot_kept_for_testing);
    tests_kept_for_testing = tests(idx_random);
    tests(idx_random) = [];

    % in columns_load are contained the tests
    % for the training: they are stored in Table_training_res
    % in columns_test are contained all the tests kept apart: they
    % are stored in Table_test_res

    columns_load{id, l+1} = tests;
    columns_test{id, l+1} = tests_kept_for_testing;
end
end
Table_training_res(:, a*2+1) = columns_load(:, 1);
Table_training_res(:, a*2+2) = columns_load(:, 2);
Table_test_res(:, a*2+1) = columns_test(:, 1);
Table_test_res(:, a*2+2) = columns_test(:, 2);
end

save('Table_training_res_1.mat', 'Table_training_res')
save('Table_test_res_1.mat', 'Table_test_res')

```

10.2 Build_boundaries_table.m

Starting from the training table, this script builds a table containing the boundaries for each family (i.e. each combination of angle, broken branches and load). The function used to build the boundaries is *boundaries* (script 10). The built table represents the model of this first approach.

```
clear
close all
clc

Table_training = importdata("Table_training_res_1.mat");

columns_names = {'Angle_0-Load_0-Up', 'Angle_0-Load_0-Low', ...
    'Angle_0-Load_1-Up', 'Angle_0-Load_1-Low', ...
    'Angle_15-Load_0-Up', 'Angle_15-Load_0-Low', ...
    'Angle_15-Load_1-Up', 'Angle_15-Load_1-Low'};
row_names = {'B0', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9'};
Table_boundaries = cell2table(cell(10, 8), 'VariableNames', ...
    columns_names, 'RowNames', row_names);

% applying the function 'boundaries' to each family. The two conditions
% given as input are related to IDs with the same number of broken branches

for k = 1:width(Table_training)
    [c1, c2] = boundaries(Table_training{1,k}{1}, Table_training{20,k}{1});
    [c3, c4] = boundaries(Table_training{2,k}{1}, Table_training{19,k}{1});
    [c5, c6] = boundaries(Table_training{3,k}{1}, Table_training{18,k}{1});
    [c7, c8] = boundaries(Table_training{4,k}{1}, Table_training{17,k}{1});
    [c9, c10] = boundaries(Table_training{5,k}{1}, Table_training{16,k}{1});
    [c11, c12] = boundaries(Table_training{6,k}{1}, Table_training{15,k}{1});
    [c13, c14] = boundaries(Table_training{7,k}{1}, Table_training{14,k}{1});
    [c15, c16] = boundaries(Table_training{8,k}{1}, Table_training{13,k}{1});
    [c17, c18] = boundaries(Table_training{9,k}{1}, Table_training{12,k}{1});
    [c19, c20] = boundaries(Table_training{10,k}{1}, Table_training{11,k}{1});
    Table_boundaries(:,k*2-1:k*2) = [{c1,c2}; {c3,c4}; {c5,c6}; {c7,c8}; ...
        {c9,c10}; {c11,c12}; {c13,c14}; {c15,c16}; {c17,c18}; {c19,c20}];
end
save('Table_boundaries_res_1.mat', 'Table_boundaries')
```

10.3 Build_medians_table.m

This script gets the dataset, applies segmentation to split it into single records and keeps a part of them for validation (test). The function used for the segmentation is *split_test* (script 9). Using the remaining records, this script builds a table containing the median curves for each family (i.e. each combination of angle, broken branches and load). The function used to build the median curves is *median_func* (script 12). The built table represents the model of this second approach.

```
clear
close all
clc

Dataset = importdata("Dataset.mat");

column_names = {'Angle_0-Load_0', 'Angle_0-Load_1', ...
    'Angle_15-Load_0', 'Angle_15-Load_1'};
row_names = {'ID1', 'ID2', 'ID3', 'ID4', 'ID5', 'ID6', 'ID7', 'ID8', 'ID9', 'ID10', ...
    'ID11', 'ID12', 'ID13', 'ID14', 'ID15', 'ID16', 'ID17', 'ID18', 'ID19', 'ID20',};

Table_medians_res = cell2table(cell(20, 4), 'VariableNames', ...
    column_names, 'RowNames', row_names);

Table_test_res = cell2table(cell(20, 4), 'VariableNames', ...
    column_names, 'RowNames', row_names);

perc_test = 0.2; % 20 % of the data are kept apart for the validation
rng(1) % the random seed is the same for all the scripts: consequently, all
    % the talbes are built using the same samples

% the Threshold_matrix contains the threshold values to well split the
% record into 10 cycles. This value is often 0.08, but not always

Threshold_matrix = 0.08*ones(20,6); % dimensions: 20 id x (2 ang * 3 n)
Threshold_matrix(2, 1) = 0.5;
Threshold_matrix(3, 1) = 0.5;
Threshold_matrix(3, 2) = 0.5;
Threshold_matrix(4, 1) = 0.5;
Threshold_matrix(4, 2) = 0.5;
Threshold_matrix(4, 3) = 0.5;
Threshold_matrix(5, 2) = 0.5;
Threshold_matrix(6, 1) = 0.5;
Threshold_matrix(6, 2) = 0.5;
Threshold_matrix(7, 2) = 0.92;
Threshold_matrix(7, 3) = 0.5;
Threshold_matrix(8, 1) = 0.5;
Threshold_matrix(8, 2) = 0.5;
Threshold_matrix(8, 3) = 0.5;
Threshold_matrix(9, 1) = 0.5;
Threshold_matrix(9, 2) = 0.5;
Threshold_matrix(9, 3) = 0.5;
Threshold_matrix(10, 1) = 0.5;
Threshold_matrix(10, 2) = 0.5;
Threshold_matrix(10, 3) = 0.85;
Threshold_matrix(11, 1) = 0.5;
Threshold_matrix(11, 2) = 0.5;
Threshold_matrix(11, 3) = 0.5;
Threshold_matrix(12, 1) = 0.5;
```

```

Threshold_matrix(12, 2) = 0.5;
Threshold_matrix(12, 3) = 0.5;
Threshold_matrix(13, 1) = 0.5;
Threshold_matrix(13, 2) = 0.5;
Threshold_matrix(13, 3) = 0.5;
Threshold_matrix(14, 1) = 0.5;
Threshold_matrix(14, 2) = 0.5;
Threshold_matrix(15, 1) = 0.5;
Threshold_matrix(15, 2) = 0.5;
Threshold_matrix(15, 3) = 0.5;
Threshold_matrix(16, 1) = 0.5;
Threshold_matrix(16, 2) = 0.5;
Threshold_matrix(17, 2) = 0.5;
Threshold_matrix(17, 3) = 0.5;
Threshold_matrix(18, 2) = 0.5;
Threshold_matrix(19, 1) = 0.5;
Threshold_matrix(19, 2) = 0.5;
Threshold_matrix(19, 3) = 0.5;
Threshold_matrix(20, 1) = 0.5;
Threshold_matrix(20, 2) = 0.5;
Threshold_matrix(20, 3) = 0.5;
Threshold_matrix(2, 6) = 0.4;
Threshold_matrix(4, 6) = 0.3;
Threshold_matrix(6, 6) = 0.3;
Threshold_matrix(7, 6) = 0.3;
Threshold_matrix(8, 6) = 0.3;
Threshold_matrix(9, 6) = 0.3;
Threshold_matrix(10, 6) = 0.3;
Threshold_matrix(13, 6) = 0.3;
Threshold_matrix(14, 6) = 0.3;
Threshold_matrix(15, 6) = 0.3;
Threshold_matrix(16, 6) = 0.2;
Threshold_matrix(17, 5) = 0.3;
Threshold_matrix(19, 6) = 0.25;

% through the for cycles for each variable, all the tests for each
% combination of variables are stored into the same cell of Table_medians

for a = 0:1 % angle index
    columns_load = cell(20, 2);
    columns_test = cell(20, 2);
    for id = 1:20

        threshold_1 = Threshold_matrix(id, a*3+1);
        threshold_2 = Threshold_matrix(id, a*3+2);
        threshold_3 = Threshold_matrix(id, a*3+3);

        Dataset_condition_1 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_1;
        Table_condition_1 = split_test(Dataset_condition_1, threshold_1);

        % since not all the combination of variables in the dataset contain
        % the same number of cycles, some of them are discarded to have an
        % equal test distribution (this sometimes happens twice for
        % condition 2 (n_2))

        while height(Table_condition_1) ~= 20
            Table_condition_1 = Table_condition_1(1:end-2,:);
        end
        Dataset_condition_2 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_2;
        Table_condition_2 = split_test(Dataset_condition_2, threshold_2);
        while height(Table_condition_2) ~= 20
            Table_condition_2 = Table_condition_2(1:end-2,:);
        end
        Dataset_condition_3 = Dataset(['Angle_' num2str(15*a)]).(['ID' num2str(id)]).n_3;

```

```

Table_condition_3 = split_test(Dataset_condition_3, threshold_3);
while height(Table_condition_3) ~= 20
    Table_condition_3 = Table_condition_3(1:end-2,:);
end
Table_condition = [Table_condition_1; Table_condition_2; Table_condition_3];
Time = Table_condition.Time;
Displacement = Table_condition.Displacement;
Resistance_1 = Table_condition.Resistance_1;
Resistance_2 = Table_condition.Resistance_2;

tot_tests = height(Table_condition);

for l = 0:1 % load (0 = unloading, 1 = loading)
    tests = cell(1, tot_tests);
    counter_tests = 0;
    for k = 1:tot_tests
        if Table_condition.Load(k) == 1
            resistance_1 = Resistance_1{k};
            resistance_2 = Resistance_2{k};
            displacement = Displacement{k};
            counter_tests = counter_tests + 1;
            % choosing which resistance use
            tests{counter_tests} = [displacement, resistance_1];
        end
    end
    tests = tests(1:counter_tests);

    % keeping some tests apart for the validation

    tot_kept_for_testing = round(counter_tests * perc_test);
    idx_random = randperm(counter_tests, tot_kept_for_testing);
    tests_kept_for_testing = tests(idx_random);
    tests(idx_random) = [];

    % in columns_load are contained the medians
    % for the model: they are stored in Table_medians_res
    % in columns_test are contained all the tests kept apart: they
    % are stored in Table_test_res

    columns_load{id, l+1} = median_func(tests);
    columns_test{id, l+1} = tests_kept_for_testing;
end
end
Table_medians_res(:, a*2+1) = columns_load(:, 1);
Table_medians_res(:, a*2+2) = columns_load(:, 2);
Table_test_res(:, a*2+1) = columns_test(:, 1);
Table_test_res(:, a*2+2) = columns_test(:, 2);
end

save('Table_medians_res_1.mat', 'Table_medians_res')
save('Table_test_res_1.mat', 'Table_test_res')

```

10.4 Build_gaussians_table.m

This script gets the dataset, applies segmentation to split it into single records and keeps a part of them for validation (test). The function used for the segmentation is *split_test* (script 9). Using the remaining records, this script builds a table containing the Gaussian parameters for each family (i.e. each combination of angle, broken branches and load). The function used to find the Gaussian parameters is *gaussian* (script 15). The built table represents the model of this third approach.

```
clear
close all
clc

res = 1; % choose the resistance used to build the tables

Dataset = importdata("Dataset.mat");

column_names = {'Angle_0-Load_0', 'Angle_0-Load_1', ...
               'Angle_15-Load_0', 'Angle_15-Load_1'};
row_names = {'ID1', 'ID2', 'ID3', 'ID4', 'ID5', 'ID6', 'ID7', 'ID8', 'ID9', 'ID10', ...
            'ID11', 'ID12', 'ID13', 'ID14', 'ID15', 'ID16', 'ID17', 'ID18', 'ID19', 'ID20'};

Table_gaussians = cell2table(cell(20, 4), 'VariableNames', ...
                             column_names, 'RowNames', row_names);

Table_test = cell2table(cell(20, 4), 'VariableNames', ...
                        column_names, 'RowNames', row_names);

perc_test = 0.2; % 20 % of the data are kept apart for the validation
rng(1) % the random seed is the same for all the scripts: consequently, all
        % the tables are built using the same samples

% the Threshold_matrix contains the threshold values to well split the
% record into 10 cycles. This value is often 0.08, but not always

Threshold_matrix = 0.08*ones(20,6); % dimensions: 20 id x (2 ang * 3 n)
Threshold_matrix(2, 1) = 0.5;
Threshold_matrix(3, 1) = 0.5;
Threshold_matrix(3, 2) = 0.5;
Threshold_matrix(4, 1) = 0.5;
Threshold_matrix(4, 2) = 0.5;
Threshold_matrix(4, 3) = 0.5;
Threshold_matrix(5, 2) = 0.5;
Threshold_matrix(6, 1) = 0.5;
Threshold_matrix(6, 2) = 0.5;
Threshold_matrix(7, 2) = 0.92;
Threshold_matrix(7, 3) = 0.5;
Threshold_matrix(8, 1) = 0.5;
Threshold_matrix(8, 2) = 0.5;
Threshold_matrix(8, 3) = 0.5;
Threshold_matrix(9, 1) = 0.5;
Threshold_matrix(9, 2) = 0.5;
Threshold_matrix(9, 3) = 0.5;
Threshold_matrix(10, 1) = 0.5;
Threshold_matrix(10, 2) = 0.5;
Threshold_matrix(10, 3) = 0.85;
Threshold_matrix(11, 1) = 0.5;
Threshold_matrix(11, 2) = 0.5;
```

```

Threshold_matrix(11, 3) = 0.5;
Threshold_matrix(12, 1) = 0.5;
Threshold_matrix(12, 2) = 0.5;
Threshold_matrix(12, 3) = 0.5;
Threshold_matrix(13, 1) = 0.5;
Threshold_matrix(13, 2) = 0.5;
Threshold_matrix(13, 3) = 0.5;
Threshold_matrix(14, 1) = 0.5;
Threshold_matrix(14, 2) = 0.5;
Threshold_matrix(15, 1) = 0.5;
Threshold_matrix(15, 2) = 0.5;
Threshold_matrix(15, 3) = 0.5;
Threshold_matrix(16, 1) = 0.5;
Threshold_matrix(16, 2) = 0.5;
Threshold_matrix(17, 2) = 0.5;
Threshold_matrix(17, 3) = 0.5;
Threshold_matrix(18, 2) = 0.5;
Threshold_matrix(19, 1) = 0.5;
Threshold_matrix(19, 2) = 0.5;
Threshold_matrix(19, 3) = 0.5;
Threshold_matrix(20, 1) = 0.5;
Threshold_matrix(20, 2) = 0.5;
Threshold_matrix(20, 3) = 0.5;
Threshold_matrix(2, 6) = 0.4;
Threshold_matrix(4, 6) = 0.3;
Threshold_matrix(6, 6) = 0.3;
Threshold_matrix(7, 6) = 0.3;
Threshold_matrix(8, 6) = 0.3;
Threshold_matrix(9, 6) = 0.3;
Threshold_matrix(10, 6) = 0.3;
Threshold_matrix(13, 6) = 0.3;
Threshold_matrix(14, 6) = 0.3;
Threshold_matrix(15, 6) = 0.3;
Threshold_matrix(16, 6) = 0.2;
Threshold_matrix(17, 5) = 0.3;
Threshold_matrix(19, 6) = 0.25;

% through the for cycles for each variable, all the tests for each
% combination of variables are stored into the same cell of Table_gaussians

for a = 0:1 % angle index
    columns_load = cell(20, 2);
    columns_test = cell(20, 2);
    for id = 1:20

        threshold_1 = Threshold_matrix(id, a*3+1);
        threshold_2 = Threshold_matrix(id, a*3+2);
        threshold_3 = Threshold_matrix(id, a*3+3);

        Dataset_condition_1 = Dataset(['Angle_' num2str(15*a)](['ID' num2str(id)]).n_1;
        Table_condition_1 = split_test(Dataset_condition_1, threshold_1);

        % since not all the combination of variables in the dataset contain
        % the same number of cycles, some of them are discarded to have an
        % equal test distribution (this sometimes happens twice for
        % condition 2 (n_2))

        while height(Table_condition_1) ~= 20
            Table_condition_1 = Table_condition_1(1:end-2,:);
        end
        Dataset_condition_2 = Dataset(['Angle_' num2str(15*a)](['ID' num2str(id)]).n_2;
        Table_condition_2 = split_test(Dataset_condition_2, threshold_2);
        while height(Table_condition_2) ~= 20
            Table_condition_2 = Table_condition_2(1:end-2,:);

```

```

end
Dataset_condition_3 = Dataset(['Angle_' num2str(15*a)])(['ID' num2str(id)]).n_3;
Table_condition_3 = split_test(Dataset_condition_3, threshold_3);
while height(Table_condition_3) ~= 20
    Table_condition_3 = Table_condition_3(1:end-2,:);
end
Table_condition = [Table_condition_1; Table_condition_2; Table_condition_3];
Time = Table_condition.Time;
Displacement = Table_condition.Displacement;
Resistance_1 = Table_condition.Resistance_1;
Resistance_2 = Table_condition.Resistance_2;

tot_tests = height(Table_condition);

for l = 0:1 % load (0 = unloading, 1 = loading)
    tests = cell(1, tot_tests);
    counter_tests = 0;
    for k = 1:tot_tests
        if Table_condition.Load(k) == 1
            resistance_1 = Resistance_1{k};
            resistance_2 = Resistance_2{k};
            displacement = Displacement{k};
            counter_tests = counter_tests + 1;
            if res == 1
                tests{counter_tests} = [displacement, resistance_1];
            else
                tests{counter_tests} = [displacement, resistance_2];
            end
        end
    end
    tests = tests(1:counter_tests);

    % keeping some tests apart for the validation

    tot_kept_for_testing = round(counter_tests * perc_test);
    idx_random = randperm(counter_tests, tot_kept_for_testing);
    tests_kept_for_testing = tests(idx_random);
    tests(idx_random) = [];

    % in columns_load are contained the gaussian parameters
    % for the model: they are stored in Table_gaussians
    % in columns_test are contained all the tests kept apart: they
    % are stored in Table_test

    columns_load{id, l+1} = gaussian(tests);
    columns_test{id, l+1} = tests_kept_for_testing;
end
end
Table_gaussians(:, a*2+1) = columns_load(:, 1);
Table_gaussians(:, a*2+2) = columns_load(:, 2);
Table_test(:, a*2+1) = columns_test(:, 1);
Table_test(:, a*2+2) = columns_test(:, 2);
end

if res == 1
    save('Table_gaussians_res_1.mat', 'Table_gaussians')
    save('Table_test_res_1.mat', 'Table_test')
else
    save('Table_gaussians_res_2.mat', 'Table_gaussians')
    save('Table_test_res_2.mat', 'Table_test')
end
end

```

10.5 Build_confusion_matrix_broken_branches_boundaries.m

This script gets all the records contained in the test table and applies the first approach to classify the broken branches. Then, it builds the confusion matrix representing the results. Here is shown the classification assuming as unknown both the angle and the load: through the inputs given to the function *find_boundaries* (script 11), this hypothesis can be changed.

```
clear
close all
clc

res = '1'; % choose the resistance used for the test
table_boundaries = importdata(['Table_boundaries_res_' res '.mat']);
table_test = importdata(['Table_test_res_' res '.mat']);

branches_cases = 0:9;
tot_cases = length(branches_cases);
true_labels = [];
predicted_labels = [];

% iterating for all the tests contained in table_test

for angle = 0:15:15
    for idx_branches = 1:tot_cases
        branches = branches_cases(idx_branches);
        for load = 0:1
            % grouping the IDs related to the same broken branches
            test_condition_1 = table_test{branches+1, angle/15*2+load+1}{1};
            test_condition_2 = table_test{20-branches, angle/15*2+load+1}{1};
            test_condition = [test_condition_1, test_condition_2];
            tot_tests = length(test_condition);
            for idx_test = 1:tot_tests
                boundaries_all = cell(tot_cases,2);
                for k = 1:tot_cases
                    % change the input of find_boundaries to change the
                    % known/unknown variables (see comments in the function script)
                    [bound_up, bound_low] = find_boundaries(table_boundaries, branches_cases(k), [], []);
                    boundaries_all(k,:) = {bound_up, bound_low};
                end
                test = test_condition{idx_test};

                % now that the test is given, apply the classification
                % algorithm

                family_scores = zeros(tot_cases, 1);
                for d = 1:length(test)
                    displacement = test(d, 1);
                    for k = 1:tot_cases
                        bound_up = boundaries_all{k, 1};
                        bound_low = boundaries_all{k, 2};
                        disp_bound = bound_up(:, 1);
                        [~, idx_closest] = min(abs(disp_bound - displacement));
                        y_low = bound_low(idx_closest, 2);
                        y_up = bound_up(idx_closest, 2);
                        y_test = test(d, 2); % resistance
                        if (y_low <= y_test) && (y_test <= y_up)
                            midpoint = (y_up + y_low)/2;
                            half_width = (y_up - y_low)/2;
```

```

        if half_width > 0
            point_score = 0.5 + 0.5/half_width*(half_width - abs(midpoint - y_test));
            family_scores(k) = family_scores(k) + point_score;
        end
    end
end
end
end
[~, idx_pred] = max(family_scores);
branches_predicted = branches_cases(idx_pred);
% storing the classification information
true_labels(end+1) = branches; %#ok<SAGROW>
predicted_labels(end+1) = branches_predicted; %#ok<SAGROW>
end
end
end

% building the confusion matrix

conf_mat = zeros(tot_cases);
for k = 1:length(true_labels)
    r = find(branches_cases == true_labels(k));
    c = find(branches_cases == predicted_labels(k));
    conf_mat(r, c) = conf_mat(r, c) + 1;
end

figure
imagesc(conf_mat)
colormap('hot')
colorbar
title(['Confusion Matrix Broken Branches - Resistance ' res])
xlabel('Predicted')
ylabel('Real')
xticks(1:tot_cases); xticklabels(string(branches_cases))
yticks(1:tot_cases); yticklabels(string(branches_cases))
axis equal tight

max_val = max(conf_mat(:));
min_val = min(conf_mat(:));
threshold = (max_val + min_val) / 2;

for r = 1:size(conf_mat, 1)
    for c = 1:size(conf_mat, 2)
        val = conf_mat(r,c);
        if val > threshold
            text_color = 'black';
        else
            text_color = 'white';
        end
        text(c, r, num2str(val), 'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', 'Color', text_color, ...
            'FontWeight', 'bold');
    end
end

accuracy = sum(diag(conf_mat)) / sum(conf_mat(:));
subtitle(['Accuracy: ', num2str(round(accuracy*100, 1)), '%'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_boundaries_res_' res '.m'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_boundaries_res_' res '.png'])

```

10.6 Build_confusion_matrix_broken_branches_medians.m

This script gets all the records contained in the test table and applies the second approach to classify the broken branches. Then, it builds the confusion matrix representing the results. Here is shown the classification assuming that the angle is unknown and the load is known: through the inputs given to the function *find_medians* (script 14), this hypothesis can be changed.

```
clear
close all
clc

res = '1'; % choose the resistance used for the test
Table_medians = importdata(['Table_medians_res_' res '.mat']);
Table_test = importdata(['Table_test_res_' res '.mat']);

% computing tot_tests

tot_tests = 0;
for r = 1:20
    for c = 1:4
        tot_tests = tot_tests + size(Table_test{r, c}{1}, 2);
    end
end
true_labels = zeros(tot_tests, 1);
predicted_labels = zeros(tot_tests, 1);

% iterating for all the tests contained in Table_test

id_cases = (1:1:20);
l = 1; % counter for all the tests
for angle = 0:15:15
    for id = id_cases
        for load = 0:1
            tests_set = Table_test{id, angle/15*2+load+1}{1};
            tot_tests = length(tests_set);
            for idx_test = 1:tot_tests
                test = tests_set{idx_test};
                x = test(:, 1); % displacement
                y = test(:, 2); % resistance
                % interpolate the test using 200 points
                [xu, ~, ic] = unique(x);
                yu = accumarray(ic, y, [], @mean);
                x_interp = linspace(min(xu), max(xu), 200)';
                y_interp = interp1(xu, yu, x_interp, 'linear');

                % now that the test is given, apply the classification
                % algotythm

                family_scores = zeros(20, 1);
                % change the input of find_medians to change the
                % known/unknown variables (see comments in the function script)
                medians = find_medians(Table_medians, [], [], load, 2);
                for k = 1:length(medians)
                    y = medians{k}(:, 2);
                    dist_quadr = (y - y_interp).^2;
                    family_scores(k) = sum(dist_quadr);
                end
                [~, idx_pred] = min(family_scores);
```

```

        id_pred = idx_pred;

        if id < 11
            broken_branches = id - 1;
        else
            broken_branches = 20 - id;
        end
        if id_pred < 11
            broken_branches_pred = id_pred - 1;
        else
            broken_branches_pred = 20 - id_pred;
        end

        true_labels(1) = broken_branches;
        predicted_labels(1) = broken_branches_pred;
        l = l+1;
    end
end
end

% building the confusion matrix

broken_branches_cases = (0:1:9);
conf_mat = zeros(10);
for k = 1:length(true_labels)
    r = find(broken_branches_cases == true_labels(k));
    c = find(broken_branches_cases == predicted_labels(k));
    conf_mat(r, c) = conf_mat(r, c) + 1;
end

figure
imagesc(conf_mat)
colormap('hot')
colorbar
title(['Confusion Matrix Broken Branches - Resistance ' res])
xlabel('Predicted')
ylabel('Real')
xticks(1:length(broken_branches_cases)); xticklabels(string(broken_branches_cases))
yticks(1:length(broken_branches_cases)); yticklabels(string(broken_branches_cases))
axis equal tight

max_val = max(conf_mat(:));
threshold = max_val / 2;

for r = 1:size(conf_mat, 1)
    for c = 1:size(conf_mat, 2)
        val = conf_mat(r,c);
        if val > threshold
            text_color = 'black';
        else
            text_color = 'white';
        end
        text(c, r, num2str(val), 'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', 'Color', text_color, ...
            'FontWeight', 'bold');
    end
end

accuracy = sum(diag(conf_mat)) / sum(conf_mat(:));
subtitle(['Accuracy: ', num2str(round(accuracy*100, 1)), '%'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_medians_res_' res '.m'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_medians_res_' res '.png'])

```

10.7 Build_confusion_matrix_broken_branches_gaussians.m

This script gets all the records contained in the test table and applies the third approach to classify the broken branches. Then, it builds the confusion matrix representing the results. Here is shown the classification assuming as unknown both the angle and the load: through the inputs given to the function *find_gaussians* (script 16), this hypothesis can be changed.

```
clear
close all
clc

res = '1'; % choose the resistance used for the test
Table_gaussians = importdata(['Table_gaussians_res_' res '.mat']);
Table_test = importdata(['Table_test_res_' res '.mat']);

% computing tot_tests

tot_tests = 0;
for r = 1:20
    for c = 1:4
        tot_tests = tot_tests + size(Table_test{r, c}{1}, 1);
    end
end
true_labels = zeros(tot_tests, 1);
predicted_labels = zeros(tot_tests, 1);

% iterating for all the tests contained in Table_test

id_cases = (1:1:20);
l = 1; % counter for all the tests
for angle = 0:15:15
    for id = id_cases
        for load = 0:1
            tests_set = Table_test{id, angle/15*2+load+1}{1};
            tot_tests = length(tests_set);
            for idx_test = 1:tot_tests
                test = tests_set{idx_test};
                x = test(:, 1); % displacement
                y = test(:, 2); % resistance
                % interpolate the test using 200 points
                [xu, ~, ic] = unique(x);
                yu = accumarray(ic, y, [], @mean);
                x_interp = linspace(min(xu), max(xu), 200)';
                y_interp = interp1(xu, yu, x_interp, 'linear');

                % now that the test is given, apply the classification
                % algotythm

                family_scores = zeros(20, 1);
                % change the input of find_gaussians to change the
                % classified variable (see comments in the function script
                % and in the intrduction of this script)
                gaussians = find_gaussians(Table_gaussians, angle, [], load);
                for k = 1:length(gaussians)
                    y_m = gaussians{k}(:, 2);
                    y_s = gaussians{k}(:, 3);
                    prob = 1./(y_s*sqrt(2*pi)) .* exp(-1/2.*((y_interp-y_m)./y_s).^2);
                    family_scores(k) = sum(prob);
```

```

        end
        [~, idx_pred] = max(family_scores);
        id_pred = idx_pred - 20*floor((idx_pred-1)/20);

        if id < 11
            broken_branches = id - 1;
        else
            broken_branches = 20 - id;
        end
        if id_pred < 11
            broken_branches_pred = id_pred - 1;
        else
            broken_branches_pred = 20 - id_pred;
        end

        true_labels(1) = broken_branches;
        predicted_labels(1) = broken_branches_pred;
        l = l+1;
    end
end
end

% building the confusion matrix

broken_branches_cases = (0:1:9);
conf_mat = zeros(10);
for k = 1:length(true_labels)
    r = find(broken_branches_cases == true_labels(k));
    c = find(broken_branches_cases == predicted_labels(k));
    conf_mat(r, c) = conf_mat(r, c) + 1;
end

figure
imagesc(conf_mat)
colormap('hot')
colorbar
title(['Confusion Matrix Broken Branches - Resistance ' res])
xlabel('Predicted')
ylabel('Real')
xticks(1:length(broken_branches_cases)); xticklabels(string(broken_branches_cases))
yticks(1:length(broken_branches_cases)); yticklabels(string(broken_branches_cases))
axis equal tight

max_val = max(conf_mat(:));
threshold = max_val / 2;
for r = 1:size(conf_mat, 1)
    for c = 1:size(conf_mat, 2)
        val = conf_mat(r,c);
        if val > threshold
            text_color = 'black';
        else
            text_color = 'white';
        end
        text(c, r, num2str(val), 'HorizontalAlignment', 'center', ...
            'VerticalAlignment', 'middle', 'Color', text_color, ...
            'FontWeight', 'bold');
    end
end

accuracy = sum(diag(conf_mat)) / sum(conf_mat(:));
subtitle(['Accuracy: ' num2str(round(accuracy*100,1)) '%'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_gaussians_res' res '.m'])
saveas(gcf, ['Figures/Confusion_matrix_broken_branches_gaussians_res_' res '.png'])

```

10.8 Final_tool.m

This script is the final tool obtained in this thesis. As described in paragraph 8, it combines the three designed ML approaches using their accuracy. It is based on the tables that represent the models of all the approaches. This particular script is designed to classify a record contained in the test table, chosen by the user. To build the confusion matrices, the usual for-loops are needed.

```
clear
close all
clc

% accuracy values. Notation: an: n stays for the approach number (1 =
%                               boundaries, 2 = medians, 3 = gaussians)
%                               rm: m stays for the resistance used (1 or 2)
%                               a/b/l: classified variable involving that
%                               accuracy (angle/branches/load)

acc_a1_r1_a = 77.5/100;
acc_a1_r2_a = 57.9/100;
acc_a1_r1_b = 54.8/100;
acc_a1_r2_b = 83.3/100;
acc_a1_r1_l = 57.9/100;
acc_a1_r2_l = 61.7/100;
acc_a2_r1_a = 72.1/100;
acc_a2_r2_a = 68.8/100;
acc_a2_r1_b = 59.2/100;
acc_a2_r2_b = 69.0/100;
acc_a2_r1_l = 54.0/100;
acc_a2_r2_l = 63.8/100;
acc_a3_r1_a = 73.8/100;
acc_a3_r2_a = 83.1/100;
acc_a3_r1_b = 66.7/100;
acc_a3_r2_b = 82.9/100;
acc_a3_r1_l = 52.9/100;
acc_a3_r2_l = 52.9/100;

% building structures containing the tables not to importing them several
% times

Struct_Tab_bound = struct();
Struct_Tab_med = struct();
Struct_Tab_gauss = struct();
Struct_Tab_test = struct();
for idx_res = 1:2
    res = num2str(idx_res);
    Struct_Tab_bound.(['R' res]) = importdata(['Table_boundaries_res_' res '.mat']);
    Struct_Tab_med.(['R' res]) = importdata(['Table_medians_res_' res '.mat']);
    Struct_Tab_gauss.(['R' res]) = importdata(['Table_gaussians_res_' res '.mat']);
    Struct_Tab_test.(['R' res]) = importdata(['Table_test_res_' res '.mat']);
end

% choosing the test

ang = input('Select the angle of the test: ');
id = input('Select the ID of the test: ');
br = id - 1;
if br > 9
    br = 20 - id;
```

```

end
load = input('Select the load of the test: ');
idx_test = input('Select the index of the test (max 6): ');

% BROKEN BRANCHES CLASSIFICATION

for idx_res = 1:2 % a classification for each resistance
res = num2str(idx_res);

% using the right tables

Table_boundaries = Struct_Tab_bound.(['R' res]);
Table_medians = Struct_Tab_med.(['R' res]);
Table_gaussians = Struct_Tab_gauss.(['R' res]);
Table_test = Struct_Tab_test.(['R' res]);
tests_set = Table_test{id, ang/15*2+load+1}{1};
test = tests_set{idx_test};

x_test = test(:, 1); % displacement
y_test = test(:, 2); % resistance
[xu, ~, ic] = unique(x_test); % not to take equal values of x, and related y
yu = accumarray(ic, y_test, [], @mean);
x_interp = linspace(min(xu), max(xu), 200)';
y_interp = interp1(xu, yu, x_interp, 'linear');

% FIRST APPROACH (BOUNDARIES) - BROKEN BRANCHES CLASSIFICATION
% see 'Confusion_matrix_broken_branches_boundaries.m' for the comments

broken_branches_cases = 0:9;
tot_cases = length(broken_branches_cases);
boundaries_all = cell(tot_cases,2);
for k = 1:tot_cases
    [bound_up, bound_low] = find_boundaries(Table_boundaries, broken_branches_cases(k), [], []);
    % input []: both angle and load are unknown
    boundaries_all(k,:) = {bound_up, bound_low};
end

family_scores = zeros(tot_cases, 1);
for d = 1:length(test)
    displacement = x_test(d);
    for k = 1:tot_cases
        bound_up = boundaries_all{k, 1};
        bound_low = boundaries_all{k, 2};
        disp_boundaries = bound_up(:, 1);
        [~, idx_closest] = min(abs(disp_boundaries - displacement));
        y_low = bound_low(idx_closest, 2);
        y_up = bound_up(idx_closest, 2);
        y_point = y_test(d);
        if (y_low <= y_point) && (y_point <= y_up)
            midpoint = (y_up + y_low)/2;
            half_width = (y_up - y_low)/2;
            point_score = 0.5 + 0.5/half_width*(half_width - abs(midpoint - y_point));
            family_scores(k) = family_scores(k) + point_score;
        end
    end
end
[~, idx_pred] = max(family_scores);
br_pred_bound(idx_res) = broken_branches_cases(idx_pred); %#ok<*SAGROW>

% SECOND APPROACH (MEDIAN) - BROKEN BRANCHES CLASSIFICATION
% see 'Confusion_matrix_broken_branches_medians.m' for the comments

medians = find_medians(Table_medians, [], [], [], 2);
% input []: both angle and load are unknown

```

```

family_scores = zeros(height(Table_medians), 1);

for k = 1:length(medians)
    x = medians{k}(:, 1);
    y = medians{k}(:, 2);
    dist_quadr = (y - y_interp).^2;
    family_scores(k) = sum(dist_quadr);
end
[~, idx_pred] = min(family_scores);
br_pred_med(idx_res) = idx_pred - 1;
if br_pred_med(idx_res) > 9
    br_pred_med(idx_res) = 20 - idx_pred;
end

% THIRD APPROACH (GAUSSIANS) - CLASSIFICATION OF ALL THE VARIABLES
% see 'Confusion_matrix_broken_branches_gaussians.m' for the comments

family_scores = zeros(40, 1);
gaussians = [Table_gaussians{:, 1}; Table_gaussians{:, 2};
             Table_gaussians{:, 3}; Table_gaussians{:, 4}];

for k = 1:length(gaussians)
    x = gaussians{k}(:, 1);
    y_m = gaussians{k}(:, 2);
    y_s = gaussians{k}(:, 3);
    prob = 1./(y_s*sqrt(2*pi)) .* exp(-1/2.*((y_interp-y_m)./y_s).^2);
    family_scores(k) = sum(prob);
end

[~, idx_piu_prob] = max(family_scores);
ang_pred_gauss(idx_res) = 15*floor((idx_piu_prob-1)/40);
id_pred_gauss = idx_piu_prob - 20*floor((idx_piu_prob-1)/20);
load_pred_gauss(idx_res) = mod(floor((idx_piu_prob-1)/20), 2);
br_pred_gauss(idx_res) = id_pred_gauss - 1;
if br_pred_gauss(idx_res) > 9
    br_pred_gauss(idx_res) = 20 - id_pred_gauss;
end

end

% FINAL BROKEN BRANCHES CLASSIFICATION

% building the matrix 'br_and_prob_pred' relating all the outputs with
% their accuracy

br_and_prob_pred = [
    br_pred_bound(1), acc_a1_r1_b;
    br_pred_bound(2), acc_a1_r2_b;
    br_pred_med(1), acc_a2_r1_b;
    br_pred_med(2), acc_a2_r2_b;
    br_pred_gauss(1), acc_a3_r1_b;
    br_pred_gauss(2), acc_a3_r2_b];
br_and_prob_tot = [(0:9)', zeros(10,1)];

% computing the probabilities for each output

for k = 1:6
    idx_br = br_and_prob_pred(k,1) + 1;
    prob_br = br_and_prob_pred(k,2);
    if br_and_prob_tot(idx_br,2) == 0
        br_and_prob_tot(idx_br,2) = prod(1-br_and_prob_pred(:,2))*prob_br/(1-prob_br);
    else
        br_and_prob_tot(idx_br,2) = br_and_prob_tot(idx_br,2)*prob_br/(1-prob_br);
    end
end

```

```

end
[~, idx_max_prob] = max(br_and_prob_tot(:,2));

% final classified value for the broken branches

br_pred_tot = br_and_prob_tot(idx_max_prob, 1);

% ANGLE CLASSIFICATION

for idx_res = 1:2 % a classification for each resistance
res = num2str(idx_res);

% using the right tables

Table_boundaries = Struct_Tab_bound.(['R' res]);
Table_medians = Struct_Tab_med.(['R' res]);
Table_gaussians = Struct_Tab_gauss.(['R' res]);
Table_test = Struct_Tab_test.(['R' res]);
tests_set = Table_test{id, ang/15*2+load+1}{1};
test = tests_set{idx_test};

% FIRST APPROACH (BOUNDARIES) - ANGLE CLASSIFICATION

angle_cases = [0, 15];
tot_angles = length(angle_cases);
boundaries_all = cell(tot_angles,2);
for k = 1:tot_angles
    [bound_up, bound_low] = find_boundaries(Table_boundaries, br_pred_tot, angle_cases(k), []);
    % input: broken branches known (value classified), load unknown (= [])
    boundaries_all(k,:) = {bound_up, bound_low};
end

family_scores = zeros(tot_angles, 1);
for d = 1:length(test)
    displacement = x_test(d);
    for k = 1:tot_angles
        bound_up = boundaries_all{k, 1};
        bound_low = boundaries_all{k, 2};
        disp_boundaries = bound_up(:, 1);
        [~, idx_closest] = min(abs(disp_boundaries - displacement));
        y_low = bound_low(idx_closest, 2);
        y_up = bound_up(idx_closest, 2);
        y_point = y_test(d);
        if (y_low <= y_point) && (y_point <= y_up)
            midpoint = (y_up + y_low)/2;
            half_width = (y_up - y_low)/2;
            point_score = 0.5 + 0.5/half_width*(half_width - abs(midpoint - y_point));
            family_scores(k) = family_scores(k) + point_score;
        end
    end
end

[~, idx_pred] = max(family_scores);
ang_pred_bound(idx_res) = angle_cases(idx_pred);

% SECOND APPROACH (MEDIAN)S - ANGLE CLASSIFICATION

medians = find_medians(Table_medians, [], br_pred_tot+1, [], 1);
% input: broken branches known (value classified), load unknown (= []).
% Knowing how 'find_medians' works, to put as known the broken branches it
% is needed to put only one of the related IDs (= br_pred_tot + 1)
family_scores = zeros(width(Table_medians)/2, 1);

for k = 1:length(medians)
    x = medians{k}{:, 1};

```

```

    y = medians{k}(:, 2);
    dist_quadr = (y - y_interp).^2;
    family_scores(k) = sum(dist_quadr);
end
[~, idx_pred] = min(family_scores);
ang_pred_med(idx_res) = (idx_pred-1)*15;

end

% the third approach has already been applied for the angle in the
% classification of broken branches

% FINAL ANGLE CLASSIFICATION

% building the matrix 'ang_and_prob_pred' relating all the outputs with
% their accuracy

ang_and_prob_pred = [
    ang_pred_bound(1), acc_a1_r1_a;
    ang_pred_bound(2), acc_a1_r2_a;
    ang_pred_med(1), acc_a2_r1_a;
    ang_pred_med(2), acc_a2_r2_a;
    ang_pred_gauss(1), acc_a3_r1_a;
    ang_pred_gauss(2), acc_a3_r2_a];
ang_and_prob_tot = [0, 0; 15, 0];

% computing the probabilities for each output

for k = 1:6
    idx_ang = ang_and_prob_pred(k,1)/15 + 1;
    prob_ang = ang_and_prob_pred(k,2);
    if ang_and_prob_tot(idx_ang,2) == 0
        ang_and_prob_tot(idx_ang,2) = prod(1-ang_and_prob_pred(:,2))*prob_ang/(1-prob_ang);
    else
        ang_and_prob_tot(idx_ang,2) = ang_and_prob_tot(idx_ang,2)*prob_ang/(1-prob_ang);
    end
end
[~, idx_max_prob] = max(ang_and_prob_tot(:,2));

% final classified value for the angle

ang_pred_tot = ang_and_prob_tot(idx_max_prob, 1);

% LOAD CLASSIFICATION

for idx_res = 1:2 % a classification for each resistance
    res = num2str(idx_res);

    % using the right tables

    Table_boundaries = Struct_Tab_bound(['R' res]);
    Table_medians = Struct_Tab_med(['R' res]);
    Table_gaussians = Struct_Tab_gauss(['R' res]);
    Table_test = Struct_Tab_test(['R' res]);
    tests_set = Table_test{id, ang/15*2+load+1}{1};
    test = tests_set{idx_test};

    % FIRST APPROACH (BOUNDARIES) - LOAD CLASSIFICATION

    load_cases = 0:1;
    tot_load = length(load_cases);
    boundaries_all = cell(tot_load,2);
    for k = 1:tot_load
        [bound_up, bound_low] = find_boundaries(Table_boundaries, br_pred_tot, ang_pred_tot, load_cases(k));
    end
end

```

```

    % input: broken branches and angle are known (values classified)
    boundaries_all(k,:) = {bound_up, bound_low};
end

family_scores = zeros(tot_load, 1);
for d = 1:length(test)
    displacement = x_test(d);
    for k = 1:tot_load
        bound_up = boundaries_all{k, 1};
        bound_low = boundaries_all{k, 2};
        disp_boundaries = bound_up(:, 1);
        [~, idx_closest] = min(abs(disp_boundaries - displacement));
        y_low = bound_low(idx_closest, 2);
        y_up = bound_up(idx_closest, 2);
        y_point = y_test(d);
        if (y_low <= y_point) && (y_point <= y_up)
            midpoint = (y_up + y_low)/2;
            half_width = (y_up - y_low)/2;
            point_score = 0.5 + 0.5/half_width*(half_width - abs(midpoint - y_point));
            family_scores(k) = family_scores(k) + point_score;
        end
    end
end
[~, idx_pred] = max(family_scores);
load_pred_bound(idx_res) = load_cases(idx_pred);

% SECOND APPROACH (MEDIAN) - LOAD CLASSIFICATION

medians = find_medians(Table_medians, ang_pred_tot, br_pred_tot+1, [], 3);
% input: broken branches and angle are known (values classified)
family_scores = zeros(2, 1);

for k = 1:length(medians)
    x = medians{k}(:, 1);
    y = medians{k}(:, 2);
    dist_quadr = (y - y_interp).^2;
    family_scores(k) = sum(dist_quadr);
end
[~, idx_pred] = min(family_scores);
load_pred_med(idx_res) = idx_pred-1;

end

% the third approach has already been applied for the load in the
% classification of broken branches

% FINAL LOAD CLASSIFICATION

% building the matrix 'load_and_prob_pred' relating all the outputs with
% their accuracy

load_and_prob_pred = [
    load_pred_bound(1), acc_a1_r1_s;
    load_pred_bound(2), acc_a1_r2_s;
    load_pred_med(1), acc_a2_r1_s;
    load_pred_med(2), acc_a2_r2_s;
    load_pred_gauss(1), acc_a3_r1_s;
    load_pred_gauss(2), acc_a3_r2_s;
];
load_and_prob_tot = [0, 0; 1, 0];

% computing the probabilities for each output

for k = 1:6
    idx_load = load_and_prob_pred(k,1) + 1;

```

```

    prob_load = load_and_prob_pred(k,2);
    if load_and_prob_tot(idx_load,2) == 0
        load_and_prob_tot(idx_load,2) = prod(1-load_and_prob_pred(:,2))*prob_load/(1-prob_load);
    else
        load_and_prob_tot(idx_load,2) = load_and_prob_tot(idx_load,2)*prob_load/(1-prob_load);
    end
end
[~, idx_max_prob] = max(load_and_prob_tot(:,2));

% final classified value for the load

load_pred_tot = load_and_prob_tot(idx_max_prob, 1);

% printing results

fprintf('%-12s%-12s%-12s%-12s%-12s%-12s%-12s%-12s\n', 'VARIABLE', 'REAL', ...
        'BOUND R1', 'BOUND R2', 'MED R1', 'MED R2', 'GAUSS R1', 'GAUSS R2', 'FINAL')
fprintf('%-12s%-12d%-12d%-12d%-12d%-12d%-12d%-12d\n', 'Angle', ...
        ang, ang_pred_bound(1), ang_pred_bound(2), ang_pred_med(1), ...
        ang_pred_med(2), ang_pred_gauss(1), ang_pred_gauss(2), ang_pred_tot)
fprintf('%-12s%-12d%-12d%-12d%-12d%-12d%-12d%-12d\n', 'Branches', ...
        br, br_pred_bound(1), br_pred_bound(2), br_pred_med(1), ...
        br_pred_med(2), br_pred_gauss(1), br_pred_gauss(2), br_pred_tot)
fprintf('%-12s%-12d%-12d%-12d%-12d%-12d%-12d%-12d\n', 'Load', ...
        load, load_pred_bound(1), load_pred_bound(2), load_pred_med(1), ...
        load_pred_med(2), load_pred_gauss(1), load_pred_gauss(2), load_pred_tot)

```

10.9 split_test.m

This function applies segmentation to a test in the time domain. Starting from the arrays of time, displacement and resistances, it splits the test, composed of 10 loading/unloading cycles, into 20 records containing the proper part of the arrays and the load label (0 = unloading, 1 = loading). The cut point is the midpoint in time domain of a constant displacement phase. To better understand the segmentation, see Figure 27 and paragraph 3.3.1.

```
function T = split_test(Dataset_condition, threshold_par)

% input: Dataset_condition: the test contained in the dataset, related to a
% certain condition (= values of angle, broken branches and load)
%       threshold_par: value used to split the arrays into the cycle
%       phases. Each phase becomes a single test (segmentation)
% output: a table containing the values of time, displacement, resistances
%         and soicitation for each test

% collecting the values of time, displacement and resistances

time = Dataset_condition.Time;
displacement = Dataset_condition.Displacement;
resistance_1 = Dataset_condition.Resistance1;
resistance_2 = Dataset_condition.Resistance2;

N = length(time);

% computing d_disp, linked to the derivative of the displacement, but with
% an index difference of 20 to reduce the effect of the noise (performances
% better than using movmean)

d_disp = displacement(21:end) - displacement(1:end-20);

threshold = threshold_par*max(displacement);

% recognizing the load phases ('stage')

stage = zeros(N-20,1);
stage(d_disp > threshold) = 1; % loading
stage(d_disp < -threshold) = -1; % unloading
stage(abs(d_disp) <= threshold) = 0; % almost constant
stage_full = zeros(N,1);
stage_full(11:end-10) = stage; % centering on the difference of 20 samples

% identifying plateaus (almost constant sections)

is_plateau = (stage_full == 0);
plateau_start = find(diff([0; is_plateau]) == 1);
plateau_end = find(diff([is_plateau; 0]) == -1);
plateau_center = round((plateau_start + plateau_end)/2); % cut points

% managing starting and final plateaus

plateau_len = plateau_end - plateau_start + 1;
threshold_plateau = 0.2 * median(plateau_len);

if plateau_len(1) < threshold_plateau
    plateau_center(1) = plateau_start(1);
```

```

end
if plateau_len(end) < threshold_plateau
    plateau_center(end) = plateau_end(end);
end

% building cut points (segmentation)

stage_change = [1; plateau_center; N+1];
stage_change = sort(stage_change);
stage_change(stage_change < 1) = [];

diff_c = diff(stage_change);
stage_change(diff_c == 0) = [];
diff_c = diff(stage_change);
for k = 1:length(diff_c)
    if diff_c(k) == 1
        if k == 1
            stage_change(k+1) = [];
        else
            stage_change(k) = [];
        end
    end
end
end

Time = {};
Displacement = {};
Resistance_1 = {};
Resistance_2 = {};
Load = [];

% segmentation

row = 1;
for k = 1:length(stage_change)-1
    idx = stage_change(k):stage_change(k+1)-1;

    Time{row,1} = time(idx); %#ok<*AGROW>
    Displacement{row,1} = displacement(idx);
    Resistance_1{row,1} = resistance_1(idx);
    Resistance_2{row,1} = resistance_2(idx);

    % load = loading/unloading based on the most present phase

    if mean(stage_full(idx)) > 0
        Load(row,1) = 1; % loading
    else
        Load(row,1) = 0; % unloading
    end

    row = row + 1;
end

% building table

T = table(Time, Displacement, Resistance_1, Resistance_2, Load);

```

10.10 boundaries.m

This function receives in input the cells of the training table related to two combinations of the variables, called “conditions”. These two conditions must present the same values for angle and load, and IDs related to the same number of broken branches (i.e. IDs whose sum is 21). The output of the function consists of the arrays constituting the boundaries: each array contains both displacement and resistance information.

```
function [upper_boundary, lower_boundary] = boundaries(condition_1, condition_2)

% input: cells of the training table corresponding to the chosen condition
% of angle, broken branches and load (= the families)
% output: the boundaries limiting both the grouped families. This function
% is built to combine families with IDs referred to the same number of
% broken branches: to group different families, see 'find_boundaries.m'

% searching the lowest and the highest displacement values to build an
% array between them

disp_min = inf;
disp_max = -inf;
tot_tests_1 = length(condition_1);
for k = 1:tot_tests_1
    disp_max_candidate = max(condition_1{k}(:, 1));
    disp_min_candidate = min(condition_1{k}(:, 1));
    if disp_max_candidate > disp_max
        disp_max = disp_max_candidate;
    end
    if disp_min_candidate < disp_min
        disp_min = disp_min_candidate;
    end
end

tot_tests_2 = length(condition_2);
for k = 1:tot_tests_2
    disp_max_candidate = max(condition_2{k}(:, 1));
    disp_min_candidate = min(condition_2{k}(:, 1));
    if disp_max_candidate > disp_max
        disp_max = disp_max_candidate;
    end
    if disp_min_candidate < disp_min
        disp_min = disp_min_candidate;
    end
end

% building the matrix 'tests' containing all the tests interpolated using
% the same displacement array 'x_interp'

tot_tests = tot_tests_1 + tot_tests_2;
tests = zeros(200, tot_tests);
x_interp = linspace(disp_min, disp_max, 200)';

for k = 1:tot_tests_1
    test = condition_1{k};
    y = test(:, 2);
    x = test(:, 1);
    [xu, ~, ic] = unique(x); % not to take equal values of x, and related y
```

```

yu = accumarray(ic, y, [], @mean);

% since the interpolation can not be done for x out of the
% range (min(xu), max(xu)), all the values of y_interp for x out of
% that range become NaN

ind_l = 1;
while x_interp(ind_l) < min(xu)
    ind_l = ind_l + 1;
end
ind_r = 200;
while x_interp(ind_r) > max(xu)
    ind_r = ind_r - 1;
end
x_interp_internal = x_interp(ind_l:ind_r);
y_interp_internal = interp1(xu, yu, x_interp_internal, 'linear');
y_l = NaN*ones(ind_l - 1, 1);
y_r = NaN*ones(200 - ind_r, 1);
y_interp = [y_l; y_interp_internal; y_r];
tests(:, k) = y_interp;
end

for k = 1:tot_tests_2
    test = condition_2{k};
    y = test(:, 2); % resistance
    x = test(:, 1); % displacement
    [xu, ~, ic] = unique(x); % not to take equal values of x, and related y
    yu = accumarray(ic, y, [], @mean);

    ind_l = 1;
    while x_interp(ind_l) < min(xu)
        ind_l = ind_l + 1;
    end
    ind_r = 200;
    while x_interp(ind_r) > max(xu)
        ind_r = ind_r - 1;
    end
    x_interp_internal = x_interp(ind_l:ind_r);
    y_interp_internal = interp1(xu, yu, x_interp_internal, 'linear');
    y_l = NaN*ones(ind_l - 1, 1);
    y_r = NaN*ones(200 - ind_r, 1);
    y_interp = [y_l; y_interp_internal; y_r];
    tests(:, tot_tests_1 + k) = y_interp;
end

% each boundary contains two columns: the displacement and the minimum or
% the maximum of all the interpolated test values

lower_boundary = [x_interp, min(tests, [], 2)];
upper_boundary = [x_interp, max(tests, [], 2)];

```

10.11 find_boundaries.m

This function combines more families computing the boundaries containing all the tests of all the grouped families. Since the boundaries of each family already contain all its records, this function can be based on the boundary table instead of working directly on the records. The user must give the boundary table and the values of the variables identifying the families to group.

```
function [upper_bound_tot, lower_bound_tot] = find_boundaries(table, Nbranches, angles, loads)

% input: table: boundaries table
%        Nbranches, angles, loads: values of the variables of the
%        families to be grouped. To group different values for a variable,
%        write a 1xn array. To group all the values for a variable, write []
% output: the boundaries limiting all the grouped families

if isempty(Nbranches)
    Nbranches = 0:9;
end
if isempty(angles)
    angles = 0:15:15;
end
if isempty.loads)
    loads = 0:1;
end

% collect the boundaries of all the families to be grouped

tot_conditions = length(Nbranches)*length(angles)*length.loads);
set_boundary_up = cell(tot_conditions, 1);
set_boundary_low = cell(tot_conditions, 1);
counter = 1;
for branches = Nbranches
    for angle = angles
        for load = loads
            set_boundary_up{counter} = table{branches + 1, angle/15*4 + 2*load + 1}{1};
            set_boundary_low{counter} = table{branches + 1, angle/15*4 + 2*load + 2}{1};
            counter = counter + 1;
        end
    end
end

% apply 'boundaries' to obtain the upper boundary of all the upper
% boundaries, and the lower boundary of all the lower boundaries

[upper_bound_tot, ~] = boundaries(set_boundary_up, {});
[~, lower_bound_tot] = boundaries(set_boundary_low, {});
```

10.12 median_func.m

This function computes the median curve of a family in the displacement-resistance plane. Since each record has its own displacement values, we interpolate it using 200 points: consequently, the interpolated displacement values are almost the same for all the records, and the computed median curve is representative of the family.

```
function Median_values = median_func(test_set)

% input: a cell containing all the tests for whom we want to compute the
% median curve
% output: the median curve (median value of resistance for each
% displacement)

% building the matrices 'Matrix_x_interp' and 'Matrix_y_interp' containing
% all the tests interpolated using 200 points

Matrix_x_interp = zeros(200, length(test_set)); % displacement
Matrix_y_interp = zeros(200, length(test_set)); % resistance
for k = 1:length(test_set)
    test = test_set{k};
    y = test(:, 2); % resistance
    x = test(:, 1); % displacement
    [xu, ~, ic] = unique(x); % not to take equal values of x, and related y
    yu = accumarray(ic, y, [], @mean);
    x_interp = linspace(min(xu), max(xu), 200)';
    y_interp = interp1(xu, yu, x_interp, 'linear');
    Matrix_x_interp(:, k) = x_interp;
    Matrix_y_interp(:, k) = y_interp;
end

median_values_x = median(Matrix_x_interp, 2); % displacement
median_values_y = median(Matrix_y_interp, 2); % resistance
Median_values = [median_values_x, median_values_y];
```

10.13 mean_medians.m

This function combines the median curves of more families. It computes the mean values of displacement and resistance for each index: the reason of this simplification is explained in paragraph 5.1.2.

```
function median_values = mean_medians(medians)

% input: median curves of the families to be grouped
% output: the median curve for all the grouped families. Since the values
% of displacement and resistance are close, we compute the mean of the
% medians (to better understand why, see the paragraph about this approach)

Matrix_x = zeros(200, length(medians)); % displacement
Matrix_y = zeros(200, length(medians)); % resistance
for k = 1:length(medians)
    test = medians{k};
    Matrix_x(:, k) = test(:, 1);
    Matrix_y(:, k) = test(:, 2);
end
median_values_x = mean(Matrix_x, 2); % displacement
median_values_y = mean(Matrix_y, 2); % resistance
median_values = [median_values_x, median_values_y];
```

10.14 find_medians.m

This function gives the set of median curves to use for the classification using the second approach. It is also able to combine more families computing the median curve of all the grouped families. Like *find_boundaries*, this function is based on the median table instead of working directly on the records. The user must give the median table, the values of the known or unknown variables and the variable to classify. To group the families, the *mean_medians* function is used (script 13).

```
function median_set = find_medians(table, angle, id, load, requested_var)

% input: table: medians table
%       angle, id, load: values of the variables of the
%       families to be grouped. If a variable is unknown or if it has to
%       be classified, write []
%       requested_var: variable to classify
% output: the medians of all the grouped families

% apply 'mean_medians' to group the selected families

if requested_var == 1 % to classify the angle
    tot_angles = int32(width(table) / 2);
    median_set = cell(tot_angles, 1);
    if isempty(id) & isempty(load) % branches and load unknown
        for k = 0:(tot_angles - 1)
            medians_family = [table(:, k*2+1); table(:, k*2+2)];
            median_set{k+1} = mean_medians(medians_family);
        end
    elseif isempty(id) % branches unknown and load known
        for k = 0:(tot_angles - 1)
            medians_family = table(:, k*2+1+load);
            median_set{k+1} = mean_medians(medians_family);
        end
    elseif isempty(load) % branches known and load unknown
        for k = 0:(tot_angles - 1)
            medians_family = [table{id, k*2+1}; table{id, k*2+2};
                             table{21-id, k*2+1}; table{21-id, k*2+2}];
            median_set{k+1} = mean_medians(medians_family);
        end
    else % branches and load known
        for k = 0:(tot_angles - 1)
            medians_family = [table{id, load+1+k*2}; table{21-id, load+1+k*2}];
            median_set{k+1} = mean_medians(medians_family);
        end
    end
end

elseif requested_var == 2 % to classify the broken branches
    num_id = height(table);
    median_set = cell(num_id, 1);
    if isempty(angle) & isempty(load) % angle and load unknown
        for k = 1:num_id
            medians_family = table{k, :};
            median_set{k} = mean_medians(medians_family);
        end
    elseif isempty(angle) % angle unknown and load known
        for k = 1:num_id
            medians_family = table{k, (load + 1):2:end};
            median_set{k} = mean_medians(medians_family);
        end
    end
end
```

```

        end
    elseif isempty(load) % angle known and load unknown
        for k = 1:num_id
            medians_family = table{k, (angle/15*2 + 1):(angle/15*2 + 2)};
            median_set{k} = mean_medians(medians_family);
        end
    else % angle and load known
        median_set = table{:, angle/15*2+load+1};
    end

elseif requested_var == 3 % to classify the load
    median_set = cell(2, 1);
    if isempty(angle) & isempty(id) % angle and branches unknown
        tot_angles = int32(width(table) / 2);
        num_id = height(table);
        for k = 0:1
            medians_family = cell(tot_angles*num_id, 1);
            for n = 0:(tot_angles -1)
                medians_family(n*num_id+1:(n+1)*num_id) = table{:, n*2+k+1};
            end
            median_set{k+1} = mean_medians(medians_family);
        end
    elseif isempty(angle) % angle unknown and branches known
        for k = 0:1
            medians_family = [table{id, k+1:2:end}, table{21-id, k+1:2:end}];
            median_set{k+1} = mean_medians(medians_family);
        end
    elseif isempty(id) % angle known and branches unknown
        for k = 0:1
            medians_family = table{:, angle/15*2+k+1};
            median_set{k+1} = mean_medians(medians_family);
        end
    else % angle and branches known
        for k = 0:1
            medians_family = [table{id, angle/15*2+k+1}; table{21-id, angle/15*2+k+1}];
            median_set{k+1} = mean_medians(medians_family);
        end
    end
end
end

```

10.15 gaussian.m

This function computes the Gaussian parameters of the resistance depending on the displacement. Like in the second approach, we have to interpolate the displacement using 200 points: consequently, the interpolated displacement values are almost the same for all the records, and the computed Gaussian parameters are representative of the family.

```
function gaussian_values = gaussian(test_set)

% input: a cell containing all the tests for whom we want to compute the
% gaussian parameters (mean and standard deviation)
% output: mean and standard deviation of the resistance for each value of
% displacement

% searching the lowest and the highest displacement values to build an
% array between them

x_min = inf;
x_max = -inf;
for k = 1:length(test_set)
    test = test_set{k};
    if min(test(:,1)) < x_min
        x_min = min(test(:,1));
    end
    if max(test(:,1)) > x_max
        x_max = max(test(:,1));
    end
end
x_interp = linspace(x_min, x_max, 200)';

% building the matrix 'Matrix_y_interp' containing all the tests
% interpolated using the same displacement array 'x_interp'

Matrix_y_interp = zeros(200, length(test_set));
for k = 1:length(test_set)
    test = test_set{k};
    x = test(:, 1); % displacement
    y = test(:, 2); % resistance
    [xu, ~, ic] = unique(x); % not to take equal values of x, and related y
    yu = accumarray(ic, y, [], @mean);
    y_interp = interp1(xu, yu, x_interp, 'linear');

    % since the interpolation can not be done for x out of the range
    % (min(xu), max(xu)), all the values of y_interp for x out of that
    % become equal to the nearest not-NaN value of y_interp

    while isnan(y_interp(1))
        n = 1;
        while isnan(y_interp(n))
            y_interp(n) = y_interp(n+1);
            n = n + 1;
        end
    end
    while isnan(y_interp(end))
        n = length(y_interp);
        while isnan(y_interp(n))
            y_interp(n) = y_interp(n-1);
            n = n - 1;
        end
    end
end
end
```

```

        end
    end
    Matrix_y_interp(:, k) = y_interp;
end

gaussian_values_x = x_interp; % displacement
gaussian_values_ym = mean(Matrix_y_interp, 2); % mean (resistance)
gaussian_values_ys = std(Matrix_y_interp, 0, 2); % std. dev. (resistance)
% second input in 'std' is 0 because it is a sample and not a population
gaussian_values = [gaussian_values_x, gaussian_values_ym, gaussian_values_ys];

```

10.16 find_gaussians.m

This function is designed to identify all the Gaussian parameters to be picked for a certain classification, based of which variable we want to classify. It is assumed that the classification happens in the usual order: first the broken branches, then the angle and finally the load. Reasoning on this approach, we understood that the classification based on it uses always the Gaussian parameters of all the families without combining them (see paragraph 6.1.2). Hence, this function results useless, but it is part of the process and then it is here shown.

```
function gaussians_set = find_gaussians(table, angle, id, load)

% input: table: gaussian parameters table
%       angle, id, load: values of the variables of the
%       families to be grouped. Assume that the variables are
%       classified in the following order: id, angle, load. To
%       classify a certain variable, write []
% output: the gaussian parameters of all the grouped families

if isempty(id) % to classify id. Hence, angle and load are unknown
    gaussians_set = [table{:, 1}; table{:, 2};
                    table{:, 3}; table{:, 4}];
end

if isempty(angle) % to classify angle. Hence, the load is unknown and the
    %             broken branches are known (so, we have to take both the
    %             IDs related to that number of broken branches)
    gaussians_set = [table{id, :}, table{20-id, :}];
end

if isempty(load) % to classify load. Hence, angle and
    %             broken branches are known (so, we have to take
    %             both the IDs related to that number of broken
    %             branches)
    gaussians_set = [table{id, angle/15*2+1:angle/15*2+2}, ...
                    table{20-id, angle/15*2+1:angle/15*2+2}];
end
```

10.17 Build_ideal_medians_table.m

This script builds the median curves ideally expected in the displacement-resistance plane. It is based on a table called “Tables_medians_SSC”: it is built exactly as “Table_medians” (see the script *Build_medians_table.m*), containing all the median curves for the CP dataset, but using the SSC dataset. Hence, it contains the median curves for all the combinations of stress angle (0° or 15°), load (0 or 1) and configuration of connected pins (from C1 to C15) (see paragraph 3.3.2 about the SSC dataset). See the paragraph 7.1 for the explanation of the transformation matrix.

Although the ideal model was not used for classification, the script to make it is presented here as part of the research path.

```
clear
close all
clc

% creating the function 'par' to have the parallel of the resistances

function res_par = par(r1, r2)
    res_par = (r1 .* r2) ./ (r1 + r2);
end

% importing Table_medians_SSC: it is a table containing all the median
% curves in the displacement-resistance plane obtained with the Singel
% Segment Characterization (SSC) dataset (with C1, ..., C15 instead of
% ID1, ..., ID20)

Table_medians_SSC = importdata("Table_medians_SSC.mat");

column_names = {'Angle_0-Load_0', 'Angle_0-Load_1', ...
    'Angle_15-Load_0', 'Angle_15-Load_1'};
row_names = {'ID1', 'ID2', 'ID3', 'ID4', 'ID5', 'ID6', 'ID7', 'ID8', 'ID9', 'ID10', ...
    'ID11', 'ID12', 'ID13', 'ID14', 'ID15', 'ID16', 'ID17', 'ID18', 'ID19', 'ID20'};

% preparing Table_medians_ideal, that will contain the expected resistances
% for the configurations of the Crack Propagation (CP) dataset (ID1 to ID20)

Table_medians_ideal = cell2table(cell(20, 4), 'VariableNames', ...
    column_names, 'RowNames', row_names);

% defining the transformation matrix from the measured resistances to the
% single resistances. Calling y the vector of measured resistances
% (combination of the individual resistances) and x the vector of the
% single segment resistances, we have y = A * x

A = [1 0 0 0 0 0 0 0 0 0 1 1 1 1 1;
    0 1 0 0 0 0 0 0 0 0 1 1 1 1 0;
    0 0 1 0 0 0 0 0 0 0 1 1 1 0 0;
    0 0 0 1 0 0 0 0 0 0 1 1 0 0 0;
    0 0 0 0 1 0 0 0 0 0 1 0 0 0 0;
    0 0 0 0 0 1 0 0 0 0 1 0 0 0 0;
    0 0 0 0 0 0 1 0 0 0 1 1 0 0 0;
    0 0 0 0 0 0 0 1 0 0 1 1 1 0 0;
    0 0 0 0 0 0 0 0 1 0 1 1 1 1 0;
    0 0 0 0 0 0 0 0 0 1 1 1 1 1 1;
```

```

1 0 0 0 0 0 0 0 0 1 0 0 0 0 0;
0 1 0 0 0 0 0 0 1 0 0 0 0 0 0;
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0;
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0;
0 0 0 0 1 1 0 0 0 0 0 0 0 0 0];

for column_name = column_names
col = column_name{1};
condition = Table_medians_SSC.(col);

% interpolating the resistances contained in a corresponding column
% with a certain angle and load condition, according to 200 new points
% equally spaced between the minimum and maximum of all records

disp_min = inf;
disp_max = -inf;
for conf = 1:15 % all the configurations of the SSC dataset (C1 to C15)
    if min(condition{conf}{:,1}) < disp_min
        disp_min = min(condition{conf}{:,1});
    end
    if max(condition{conf}{:,1}) > disp_max
        disp_max = max(condition{conf}{:,1});
    end
end

% building the matrix Res_meas, containing the measured resistances for
% all the configurations (C1 to C15) for 200 values of the displacement

Res_meas = zeros(15, 200);
disp_interp = linspace(disp_min, disp_max, 200)';
for conf = 1:15
    test = condition{conf};
    displ = test(:, 1);
    res = test(:, 2);
    res_interp = interp1(displ, res, disp_interp, 'linear');

    % the interpolation range is [disp_min, disp_max]. Hence, for some
    % tests a displacement value out of range is required for the
    % interpolation. To fix them, all the initial and ending NaN values
    % are replaced with the closest not NaN resistance value

    while isnan(res_interp(1))
        k = 1;
        while isnan(res_interp(k))
            res_interp(k) = res_interp(k+1);
            k = k + 1;
        end
    end
    while isnan(res_interp(end))
        k = length(res_interp);
        while isnan(res_interp(k))
            res_interp(k) = res_interp(k-1);
            k = k - 1;
        end
    end
    Res_meas(conf, :) = res_interp';
end

% computing the median curves of the resistances by applying the
% transformation matrix

Res_sing = A \ Res_meas;

% obtaining all the single segment resistances

```

```

r1 = Res_sing(1, :)';
r2 = Res_sing(2, :)';
r3 = Res_sing(3, :)';
r4 = Res_sing(4, :)';
r5 = Res_sing(5, :)';
r6 = Res_sing(6, :)';
r7 = Res_sing(7, :)';
r8 = Res_sing(8, :)';
r9 = Res_sing(9, :)';
r10 = Res_sing(10, :)';
r11 = Res_sing(11, :)';
r12 = Res_sing(12, :)';
r13 = Res_sing(13, :)';
r14 = Res_sing(14, :)';
r15 = Res_sing(15, :)';

% computing the expected resistance value for all the configurations of
% the CP dataset (ID1 to ID20)

Res_CP = zeros(200, 20);
Res_CP(:,1) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(par(r3, r8), (r14 +
par(par(r2, r9), (r15 + par(r1, r10))))))));
Res_CP(:,2) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(par(r3, r8), (r14 +
par(par(r2, r9), (r15 + r10))))));
Res_CP(:,3) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(par(r3, r8), (r14 + par(r9,
(r15 + r10))))));
Res_CP(:,4) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(r8, (r14 + par(r9, (r15 +
r10))))));
Res_CP(:,5) = r11 + par(par(r5, r6), (r12 + par(r7, (r13 + par(r8, (r14 + par(r9, (r15 + r10))))));
Res_CP(:,6) = r11 + par(r6, (r12 + par(r7, (r13 + par(r8, (r14 + par(r9, (r15 + r10))))));
Res_CP(:,7) = r11 + r12 + par(r7, (r13 + par(r8, (r14 + par(r9, (r15 + r10))))));
Res_CP(:,8) = r11 + r12 + r13 + par(r8, (r14 + par(r9, (r15 + r10)))));
Res_CP(:,9) = r11 + r12 + r13 + r14 + par(r9, (r15 + r10));
Res_CP(:,10) = r11 + r12 + r13 + r14 + r15 + r10;
Res_CP(:,11) = r11 + r12 + r13 + r14 + r15 + r1;
Res_CP(:,12) = r11 + r12 + r13 + r14 + par(r2, (r15 + r1));
Res_CP(:,13) = r11 + r12 + r13 + par(r3, (r14 + par(r2, (r15 + r1))));
Res_CP(:,14) = r11 + r12 + par(r4, (r13 + par(r3, (r14 + par(r2, (r15 + r1))))));
Res_CP(:,15) = r11 + par(r5, (r12 + par(r4, (r13 + par(r3, (r14 + par(r2, (r15 + r1))))));
Res_CP(:,16) = r11 + par(par(r5, r6), (r12 + par(r4, (r13 + par(r3, (r14 + par(r2, (r15 + r1))))));
Res_CP(:,17) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(r3, (r14 + par(r2, (r15 +
r1))))));
Res_CP(:,18) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(par(r3, r8), (r14 + par(r2,
(r15 + r1))))));
Res_CP(:,19) = r11 + par(par(r5, r6), (r12 + par(par(r4, r7), (r13 + par(par(r3, r8), (r14 +
par(par(r2, r9), (r15 + r1))))));
Res_CP(:,20) = Res_CP(:,1);

% collecting data in Table_medians_ideal

for k = 1:20
    Table_medians_ideal.(col){k} = [disp_interp, Res_CP(:,k)];
end

end

save('Tabella_mediane_tot_res_1.mat', 'Table_medians_ideal')

```