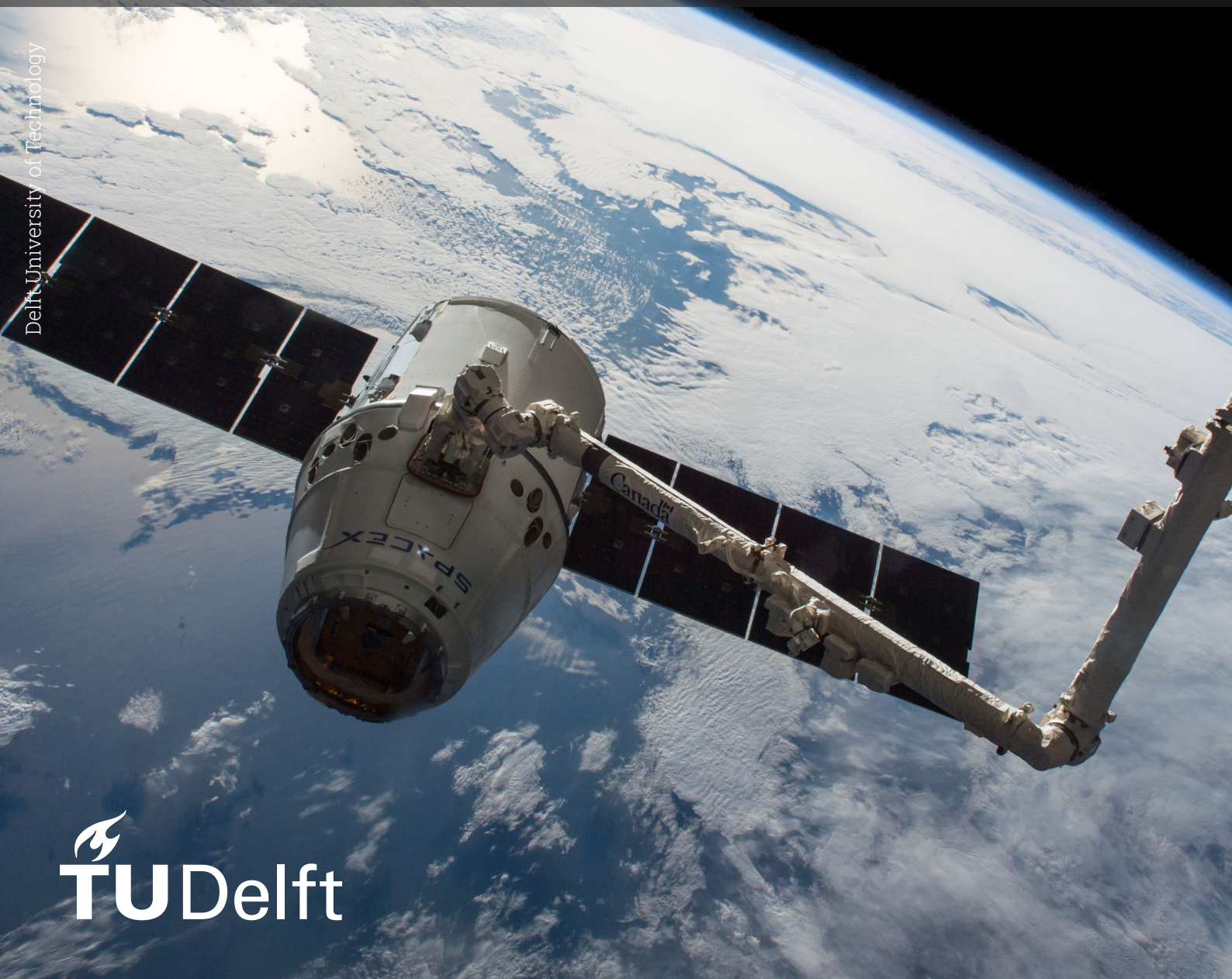
Arc Fault Detection Algorithm for DC Bipolar Microgrids

Intelligent Electrical Power Grids (IEPG)

Master thesis

Francisco Gil Anaya



Delft University of Technology

TU Delft

Arc Fault Detection Algorithm for DC Bipolar Microgrids

Intelligent Electrical Power Grids (IEPG)

by

Francisco Gil Anaya

University Supervisor:	Prof.Dr.Ir. M. Popov
Company Supervisor:	Ir. S. Shah
Project Duration:	December, 2024 - October, 2025
Faculty:	Faculty of Electrical Engineering, Delft

Thesis Committee:

Prof. Dr. Ir. Marjan Popov
Dr. D.J.P. Domenico Lahaye
Dr. Laurens Mackay
Ir. Samad Shah

TU Delft DCpportunities IEPG Intelligent Electrical Power Grids

Abstract

Series DC arc faults are hard to detect in low-voltage direct current grids because the change in line current is usually too small for classical protection devices. Undetected arcs can overheat conductors and start fires, so dependable detection is essential for future bipolar microgrids. In this thesis, an embedded method that combines the detailed energy from a short Discrete Wavelet Transform with the wide-band energy of a Fast Fourier Transform, then classifies each observation window with a linear support vector machine that runs on a single microcontroller, is developed. Laboratory and field tests confirm that the algorithm detects low-energy series arcs without nuisance trips and operates within the response time required by UL 1699B. The novelty of the work is the mixed wavelet–FFT feature, which captures both local transients and wide-band noise in a compact indicator, making accurate detection possible with modest processing resources.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ADC	Analog-to-Digital Converter
AFCI	Arc-Fault Circuit Interrupter
DMA	Direct Memory Access
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
EMI	Electromagnetic Interference
ESP32	Espressif ESP32 System-on-Chip
FFT	Fast Fourier Transform
FIR	Finite Impulse Response (filter)
FPU	Floating-Point Unit
IEC	International Electrotechnical Commission
MCU	Microcontroller Unit
ML	Machine Learning
NEC	National Electrical Code
PV	Photovoltaic
PWM	Pulse-Width Modulation
RC	Resistor–Capacitor (filter)
RMS	Root Mean Square
SNR	Signal-to-Noise Ratio
SSP	Single-Signal-Processor
SVM	Support Vector Machine
UART	Universal Asynchronous Receiver-Transmitter
UL	Underwriters Laboratories
db3	Daubechies three-tap wavelet
db4	Daubechies four-tap wavelet

Symbols

Symbol	Definition	Unit
C	Capacitance	[F]
D	FFT discard percentage (D-factor)	[–]
E	Energy released by an arc	[J]
E_{FFT}	Integrated spectral energy (0–150 kHz)	[A^2]
E_{D2}	Level-2 wavelet detail energy	[A^2]
f_{alias}	Alias frequency	[kHz]
f_s	Sampling frequency	[kHz]
I_{arc}	Arc current	[A]
k	Empirical constant (Paukert model)	[–]
L	Electrode gap length	[mm]
N	Samples per processing window	[–]
R	Resistance	[Ω]
R_{arc}	Arc resistance	[Ω]
V_{arc}	Arc voltage	[V]

Symbol	Definition	Unit
$X[k]$	k-th FFT-coefficient magnitude	[A]
s	SVM decision score	[–]
$\mathbf{w}$	SVM weight vector	[–]
b	SVM bias term	[–]
z	Standardised feature component	[–]
μ	Mean of feature	[–]
σ	Standard deviation of feature	[–]
Δt	Processing-window duration	[s]

1

Introduction

1.1. Background and Motivation

Arc faults form plasma columns with temperatures close to ten thousand kelvin. The heat produced can ignite insulation, start fires, or, in extreme cases, lead to explosions, while exposing workers to serious dangers. When an arc is not detected, it can damage converters or shut down an entire network; On the other hand, false trips from arc detectors can lead to unnecessary outages, reduced system availability, and increased operational costs,[1].

The expansion of direct current (DC) technology makes reliable detection more urgent than ever. Photovoltaic arrays, electric vehicle chargers, and others have now introduced numerous high-voltage buses, often arranged in bipolar form. A bipolar dc grid provides two symmetric supply rails at $+V_{\mathrm{dc}}$ and $-V_{\mathrm{dc}}$ with an optional neutral conductor at $0\ \mathrm{V}$. Therefore, loads can connect line-to-line, receiving the full $2V_{\mathrm{dc}}$, or line–to–neutral for half the voltage. Existing analogue methods that were transferred from alternating-current installations rely on natural current zero crossings, which do not occur in direct current, and frequency tools such as the Fourier transform require periodic signals, whereas a DC arc is aperiodic. At the same time, standards such as UL 1699B and NEC 690.11 demand that a photovoltaic arc-fault circuit interrupter must recognise a three-hundred-watt series arc and clear it within two and a half seconds [2, 3]. Any practical solution must therefore combine reliable performance with the low cost that industry expects.

Bipolar DC microgrids add complications. The neutral conductor allows pole-to-pole and pole-to-neutral connections, yet it can also lead to ground-loop currents that shift the neutral voltage and mask fault paths. Power-electronic devices, which are present at most nodes, limit fault current magnitude and duration so that traditional overcurrent relays often fail to detect low-energy series arcs.

1.2. DC Microgrid Structure

Direct-current (DC) networks distribute power on conductors that retain fixed polarity rather than the sinusoidal three-phase waveform used in conventional alternating-current (AC) grids. The absence of phase angle and frequency constraints eliminates synchronization hardware, reactive power circulation, and skin-effect losses; as a result, a DC structure can remove one conversion stage in many devices, photovoltaic (PV) inverters, server power supplies, and battery chargers, and decrease conversion losses by roughly 3 to 6 percentage points[4].

When this approach is scaled down to campus or building level, the term *DC micro-grid* is used. Typical medium power feeders run at bipolar ±350–700 V for PV strings, energy storage converters, and electric vehicle (EV) chargers, while auxiliary and information technology racks employ 48 V rails for intrinsic touch safety. Sources (PV, batteries), programmable loads and bidirectional DC-DC converters share the same bus and trade power through a simple voltage-droop or current-sharing law; no distinction between real and reactive components is required, which simplifies the supervisory controller compared with an AC microgrid. Islanded operation is therefore straightforward, and critical loads ride through

upstream AC outages without zero-crossing or frequency-stability issues.

Two wiring styles exist. *Unipolar* links use one live conductor and a return, whereas *bipolar* links introduce a second live pole and a neutral so that equipment may choose $\mathrm{V}^{+}-N$, $N-\mathrm{V}^{-}$ or the full $\mathrm{V}^{+}-\mathrm{V}^{-}$ span. The latter arrangement supplies two voltage levels with a single bus and offers inherent redundancy: loss of either pole leaves the healthy pole able to deliver approximately half the rated power via the neutral. Single-pole faults draw current in the affected conductor only, while pole-to-pole faults bypass the neutral and require faster protective clearing. These traits, together with the reduced converter count and simpler control, make bipolar DC micro-grids attractive in data-centre backbones, fast-charging plazas, and remote autonomous power systems.

Implications for arc-fault localisation Because the neutral anchors the healthy pole close to the ground, a series arc on One conductor induces noise on that pole only. Detectors can therefore tell which pole is bad with fewer sensors than in a two-wire setup [5].

1.3. Problem Statement

DC networks require arc-fault protection that is both reliable and economical, however, commercial photovoltaic arc-fault circuit interrupters still show missed detections and a high rate of nuisance trips[6].

Four technical gaps must be closed. First, a detector must indicate both the time and the frequency at which arc-related energy appears because separate time or frequency views are not sufficient. Second, the influence of cable length and inductance on detection range remains a challenge because long cables attenuate high-frequency arc noise. Third, electromagnetic interference in practical installations changes throughout the day, so fixed thresholds drift and reduce stability. Fourth, the algorithm must execute with tight limits on processing time, memory, and latency to comply with the standards.

In DC networks, two fault geometries are possible: (i) parallel arcs that short-circuit conductors and (ii) series arcs that open an energised path. Parallel arcs drive high currents that are usually cleared by conventional over-current protection, whereas series arcs draw only the load current; therefore, they remain undetected. The project focuses on series arc detection. Faults may form between the positive and negative poles, between a pole and the neutral, or between either pole and exposed wires.

Arc-fault circuit interrupters (AFCIs), detect these events by analysing line current waveforms [7]. However, most commercial AFCIs were derived from residential AC technology and cannot be implemented in DC systems, and perform poorly on lower energy arcs.

Commercial AFCIs detect high–frequency current that are superposed to 50/60 Hz sine-wave and then compare the burst pattern with templates recorded from residential wiring. In a DC microgrid, this approach breaks down for three linked reasons. First, the DC waveform has no natural zero crossings, so the device cannot anchor its windowing logic, and it loses the “per-half-cycle” synchronisation that its firmware expects; as a result many series arcs are simply ignored because their signature does not match the AC template. Second, the same 10–100 kHz band that an AFCI monitors is already filled with switching noise from step-up converters, battery chargers and motor drives; these legitimate emissions either mask the weak spectrum of a low-energy arc or trigger nuisance trips when their amplitude beats the fixed threshold stored in the AFCI, a problem that worsens as more converters are connected to the bus. Third, cable runs in buildings can exceed 30 m, and the line inductance attenuates the very harmonics an AFCI needs to see, so detectors that work in a short test bench fail once the array is installed on the roof; covering all branches with extra AFCIs would solve the attenuation but multiplies cost and wiring complexity, which is unacceptable for small rooftop systems. Due to these three drawbacks, the AFCI concept, although mandatory in many PV codes, does not deliver reliable protection in present-day DC microgrids, and a dedicated algorithm that tracks arc energy in both time and frequency is still required.

1.4. Objectives

Formulating clear research questions is important to motivate the research. In Bipolar DC Grids, there is a need for a detection technique that combines the requirements of reliability, speed, and low cost

in bipolar micro-grids, especially for low-energy series faults. Accordingly, the following questions are defined so that each chapter delivers an answer.

1.4.1. Research Objectives

Q1. Which detection principles are currently applied to series DC-arc faults in microgrids, and what are their respective advantages and limitations?
Addressed in Chapter 2 (Literature Review), which maps time-domain, frequency-domain, and hybrid techniques and notes the performance gaps that motivate a new approach.

Q2. Which electrical or signal features give the highest discrimination between normal operation and series arcs in a low-voltage bipolar grid, and how can they be extracted in real time?
Developed in Chapter 3 (Signal-Processing Chain), where the candidate parameters wavelet detail energies and wide-band FFT energy are quantified and ranked.

Q3. How can the selected detection algorithm be implemented on a single-signal-processor (SSP) platform so that it meets the 2.5 s clearing time of UL1699B without extra sensors?
Demonstrated in Chapter 4, which details the firmware implementation to an STM32G4

Each research question defines a key contribution of this work: (i) a structured evaluation of existing arc fault detection methods, (ii) an experimentally supported feature set for identifying series arcs, and (iii) a resource-efficient embedded implementation suitable for real-life implementation.

1.4.2. Regulatory framework for DC-arc protection

The document that defines the series DC-arc test duty is **UL 1699B:2021** (Outline of Investigation for PV DC Arc-Fault Circuit Protection). The arc is with a voltage up to 300 V_{dc} and 3 A and requires a trip command within *2.5 s*. Seven non-fault events, such as load steps and inverter start-up, must pass without a trip. The standard also prescribes reset, audible/visual indication, and environmental endurance tests.

The **National Electrical Code (NEC) Article 690.11, 2023** makes a listed DC-Arc-Fault Circuit Interrupter mandatory for every PV source or output circuit that operates above 80 V_{dc}. Compliance is demonstrated by passing the UL 1699B tests, and the interrupter must provide automatic or manual reset.

IEC 62606:2013 gives general requirements for Arc-Fault Detection Devices (AFDD) used in low-voltage AC and DC systems. While broader in scope, it reinforces the need for verified response time and immunity to nuisance trips and is becoming relevant for European DC micro-grids[8].

In this project, the detection algorithm is therefore evaluated against the UL 1699B series-arc duty and the NEC clearing-time mandate.

1.5. Methodology

Initially, a review of existing DC arc fault detection methods is conducted to identify the strengths and limitations of current approaches. Based on this review, promising techniques are evaluated through simulation in Octave, using simulated arc fault data to assess their effectiveness. A hybrid detection method is then developed, combining wavelet-based and FFT-based feature extraction with SVM classification. This method is implemented on an STM32 microcontroller for real-time detection in a bipolar DC microgrid. The implementation is validated through both controlled laboratory experiments and field trials in the Green Village microgrid, where real-world cable setups are used. The system's performance is evaluated in terms of trip reliability, false positive rates, and real-time operational constraints.

1.6. Thesis Outline

This project is divided into six chapters. **Chapter 1: Introduction** explains the safety risk of series DC arc faults, sets the objectives, and gives a short guide to the rest of the report. **Chapter 2: Arc Physics** describes how an electric arc starts and develops, shows its voltage–current behaviour, and lists the factors that influence its duration. **Chapter 3: Detection Methods** reviews the main existing techniques for detection, including time-domain indicators, frequency-domain, and machine-learning

classifiers. **Chapter 4: System and Methodology** introduces the laboratory setup, the measurement chain, and the signal-processing steps used to test the proposed algorithm. **Chapter 5: Results and Discussion** presents the experimental data, compares the new method with standard detectors, and comments on its accuracy and speed. **Chapter 6: Conclusion and Future Work** summarises the key findings and suggests improvements for practical applications.

2

Literature Review

Chapter 2 gathers the background needed for an arc detector. It first sketches arc physics and the Cassie–Mayr family of models, then compares DC and AC arcs, and deepens into hardware and software detection techniques, from analogue band-pass counters to frequency- and wavelet-based indicators and simple machine-learning classifiers .

2.1. Electrical discharges

Electrical discharges refer to the flow of electric current through a normally nonconductive medium, such as air, due to ionization under a high electric field. These discharges manifest in various forms depending on the current density, pressure, and electrode conditions.

There are 4 types of electrical arc discharges:

- Corona discharge
- Spark discharge
- Glow discharge
- Arc discharge

Among these, an arc is a man-made arc discharge phenomenon in which a permanent AC or DC power source maintains the burning plasma. The air is first ionized, and a glow discharge appears before the current through the electrodes increases as an arc occurs.

2.1.1. Electrical Arcs

Two types of arcs pose a problem for electrical systems: parallel and series arcs. Parallel arc faults occur when insulation breakdown creates unintended conductive paths, leading to higher current flows capable of activating standard protective devices. On the other hand, series arcs tend to occur between two connected cables that are drawn apart, typically from loose connections, aging cables, or damaged insulation.

When a conductor in a DC circuit opens because a connector loosens, a cable vibrates, or insulation deteriorates, the load current does not stop cleanly. The instant a microscopic gap forms, the resulting electric field (tens of $\mathrm{kV\,cm^{-1}}$ at the tip) ionises the surrounding air and any degraded insulation. A luminous plasma column with a temperature of roughly $6000 - 13000, \mathrm{K}$ bridges the electrodes and a series DC arc is established [9]. Unlike AC systems, there is no natural current zero crossing, so the plasma persists until the supply energy is removed, the gap length exceeds the sustaining limit, or an external quenching mechanism intervenes[10]. Electrically, the arc behaves as a highly nonlinear resistor whose value fluctuates. Metal vapour from the electrodes continually condenses and re-evaporates inside the column, creating chaotically varying micro-constrictions. Each constriction introduces sharp $\mathrm{d}i/\mathrm{d}t$ edges that radiate broadband electromagnetic energy. Measurements on photovoltaic strings

show that most of this energy lies below $\sim 100\,\mathrm{kHz}$; above that, inverter switching harmonics dominate.

Two practical signatures therefore, emerge:

- *Current trace* a sudden step-drop in mean current (typically 10–$30\,\%$) with super-imposed needle-like spikes.
- *Spectral burst* short packets of energy (tens of milliseconds) appear at mid-frequencies, clearly separated from the very-low-frequency load ripple and the very-high-frequency switching noise.

Because the highest frequencies are already saturated by converter activity, practical detection schemes focus on the mid-lower band region, where arc-related bursts stand out while other noise remains small.

Cable as a low-pass element All the wire in a PV string behaves like a long ladder of tiny inductors (L') in series and tiny capacitors (C') to ground. Together they form a distributed *low-pass* filter. A simple way to see the effect is to treat the line as a first-order low-pass with corner

$$f_c = \frac{1}{2\pi R_s C' \ell},$$

where R_s is the source or sensor resistance and ℓ is the cable length. As the string gets longer, f_c drops, so the fast 1–100 kHz bursts that mark a series arc are more and more attenuated. Beyond a certain length the breaker sees almost no high-frequency energy and may fail to trip even though the arc keeps burning [1].

2.1.2. Different Arc Models

Many different equations were created to determine the voltage-current relation of arcs, the following table compiles some of them [11]:

Table 2.1: V–I Equations in Previous Studies

Name	Equation	Experimental Condition
Ayrton	$V_{\text{arc}} = A + BL + \dfrac{C + DL}{I_{\text{arc}}}$	Carbon electrodes
Steinmetz	$V_{\text{arc}} = A + \dfrac{C(L + D)}{I_{\text{arc}}^{0.5}}$	Carbon and magnetite electrodes
Nottingham	$V_{\text{arc}} = A + \dfrac{B}{I_{\text{arc}}^{n}}$	n related to electrode material; L: 0.039 to 0.39 in
Paukert	$V_{\text{arc}} = \dfrac{a}{I_{\text{arc}}^{b}}$	L: 0.039 to 7.78 in; I_{arc}: 0.3 to 100 kA
Modified Paukert	$V_{\text{arc}} = \dfrac{a + cL}{I_{\text{arc}}^{b+dL}}$	L: 0.04 to 0.12 in; I_{arc}: 3 to 25 A

Each of these equations have different complexities and are used in different contexts. In this project, the Paukert equation will be considered, since it is widely used in DC arc studies, because the model can predict if the arc is sustained .

The characteristics of an electric arc are determined by the gap distance between electrodes, the arc voltage, the current and the material properties of the electrodes. As the gap distance increases, a higher voltage is required to sustain the arc. This is due to the extended arc column, which increases the impedance and demands more energy for ionization. The arc voltage increases with gap length under all tested load current levels, and this relationship becomes more prominent at lower voltages. Additionally, the material of the electrodes affects arc behavior due to different thermal and electrical conductivities. For instance, carbon electrodes tend to produce higher arc voltages compared to copper electrodes, mainly due to differences in surface emissivity and cathode fall.

The extended Paukert equation was proposed. This version incorporates gap length in both the numerator and the exponent of the arc current:

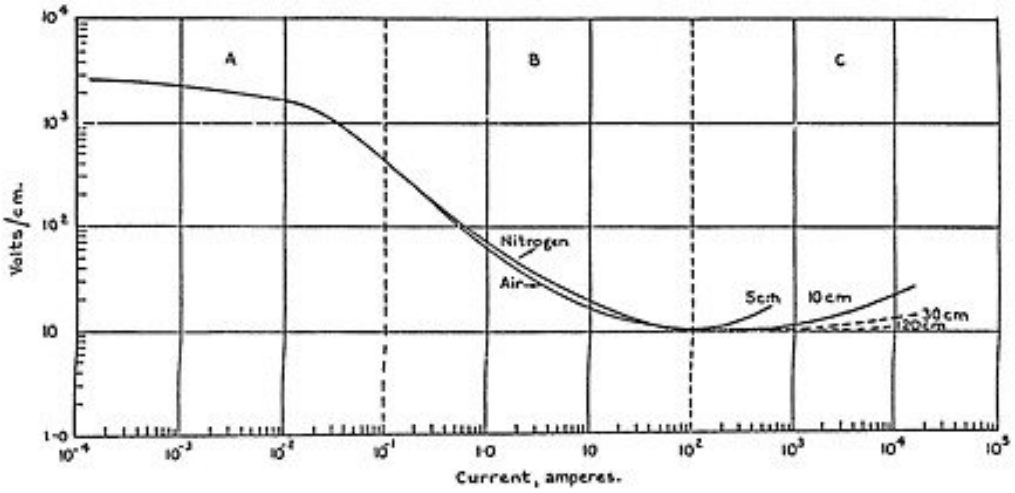


Figure 2.1: Electric field . arc current in free-burning arcs, adapted from [12]. Characteristic of a free-burning arc in nitrogen and in air.

- **Region A** corresponds to low-current conditions (typically below 0.1 A), where the discharge behavior is dominated by corona or glow discharge mechanisms. In this range, the electric field is high due to limited current conduction, and the arc is not yet fully developed.
- **Region B** goes from approximately 0.1 A to 100 A. In this region, the curve is nearly linear, which makes it difficult to distinguish from other resistive components because it follows Ohm's law. Because of this, arcs in region B can go undetected if algorithms rely purely on current thresholds.
- **Region C** occurs at higher arc currents, typically beyond 100 A. In this region, the electric field increases again due to the influence of electrode erosion and vaporized metal entering the arc column. These affect conductivity and contribute to instability in the arc behavior.

2.1.3. Classical Cassie–Mayr arc models

Electric-arc dynamics are often captured with two first-order differential models:

$$\text{Cassie (high current):} \quad \tau_{\mathrm{C}} \frac{\mathrm{d}G}{\mathrm{d}t} = \frac{I^2}{E_0^2} - G, \tag{2.1}$$

$$\text{Mayr (low current):} \quad \tau_{\mathrm{M}} \frac{\mathrm{d}G}{\mathrm{d}t} = \frac{V^2}{P_0} - G, \tag{2.2}$$

where $G(t) = 1/R(t)$ is the arc conductance, I the arc current, V the arc voltage, E_0 the steady-state electric field, P_0 the constant cooling power and τ the thermal time-constant of the plasma.

Cassie assumes the whole column is cooled by forced convection, so it fits the negative-resistance behaviour found above 10A. Mayr treats only surface cooling and therefore describes the hyperbolic VI curve seen below a few amperes. A smooth "hybrid'' model blends both equations with a transition factor $k(I)$ and is widely used in circuit simulators because it converges for *all* currents [13]. The hybrid keeps Cassie for large I, Mayr for small I, and lets the resistance rise to infinity when the arc is finally extinguished.

2.1.4. Arc Physics

Hysteresis is the dependence of the state of a system on its history, DC arcs shows this in their current–voltage relation. The voltage needed to initiate an arc is higher than the voltage required to keep it going. This creates two different voltage values: an ignition voltage and a sustaining voltage. This phenomenon makes arc modeling more complex, especially detecting arcs that reappear after extinction. Detection systems based only on steady-state models may result in false negatives.

The energy released by an arc is calculated using the following expression:

$$E = \int V_{\text{arc}} I_{\text{arc}}\, dt \tag{2.3}$$

The energy released by an arc is important because it reflects the amount of damage the arc can cause to its surroundings. It is a way of measuring of how powerful the arc is, which makes it important in arc detection. Short, high-current arcs can cause instant damage to conductors, while longer arcs at lower currents may slowly degrade insulation and increase fire risk. For this reason, detection systems must be responsive and capable of identifying different energy levels, even in arcs that appear small at first [14].

In AC systems, arcs naturally extinguish due to current zero-crossings. In contrast, DC systems lack this feature, which makes arcs more persistent. To interrupt them, the system must either extend the arc gap, drop the voltage below the sustaining threshold, or use external mechanisms like magnetic blowouts or snubber circuits. This makes the protection design more challenging, and proper modeling of arc behavior becomes necessary for reliable fault response.

DC arcs can show oscillating behavior, especially in circuits with parasitic inductance or capacitance. These oscillations may cause the arc to extinguish and reignite repeatedly. This behavior shows up as noise and can become a challenge against simpler detection methods. As a result, it has been demonstrated that detection methods based on the time-frequency domain, usually lead to better results since they can capture fast transients across a wide frequency range[1, 6].

The way the circuit is built has a strong influence on how arcs behave. In systems with high source impedance such as long PV strings the current tends to be lower, while the arc voltage becomes higher. This leads to arcs that burn longer and are more difficult to detect. In contrast, low impedance systems can produce high current arcs, which may cause circuit breakers to trip immediately. In such cases, arc fault detection may not be needed, especially for parallel arcs that occur between the positive line and ground or neutral. However, series arcs present a different challenge. They do not always generate enough current to trigger protective devices, and therefore require a dedicated detection algorithm to identify and interrupt the fault reliably.

Over time, arcs erode the electrode materials. This changes the gap distance and introduces metallic vapors into the arc path, which increases the conductivity. This can make arcs harder to extinguish and more dangerous. Even arcs at low power levels can cause permanent damage if left undetected, especially in applications such as solar connectors or EV charging systems

Influence of Electrode Material and Surface Condition on Arc Behaviour

The material at the roots of the arc is one of the variables that changes the voltage-current characteristics of an arc. Because the material dictates thermal conductivity, emissivity, and vapor pressure, it fixes how fast energy is removed from or fed back into the plasma column once conduction starts.

Among common conductors, copper offers the lowest sustaining voltage, whereas carbon or graphite drives the voltage higher because of their poorer thermal conductivity and larger cathode fall. Aluminum, on the other hand, adds extra chemical energy due to exothermic oxidation; arcs between aluminum electrodes released noticeably more heat and posed the highest burn-through risk compared with copper or steel.

The surface condition matters as much. A dusty or oxidized contact behaves like a thin insulating material: the circuit is open until the electric field is enough to penetrate the layer, after which a sudden, high-energy discharge connects the full gap.

Thus, the electrode selected must match the thermal properties of copper to mimic the real-world implementation of a low-voltage DC microgrid. Second, keep contact faces clean: Even a thin layer of workshop dust can increase the effective gap resistance, delay current flow, and trigger an unnecessarily violent arc.

2.2. Arc Fault Phenomena in DC Grids

2.2.1. Occurrence of Arc Faults in Bipolar DC Grids

Arc faults can occur in many parts of a DC network, especially in bipolar configurations. They typically appear when electrical contacts are unintentionally separated under load, or when a conductor connection degrades over time. Common scenarios include:

- **Plug-in and plug-out events:** Connecting or disconnecting loads while current is flowing can momentarily create a gap, which may sustain an arc.
- **Loose or corroded connections:** Vibration, thermal cycling, or improper tightening of terminals leads to unstable contact resistance, creating conditions for series arc faults.
- **Damaged cable insulation:** Aging, bending stress, or abrasion of insulation can expose conductors. In bipolar grids, this can result in both pole-to-pole and pole-to-neutral arcing.
- **Switching operations:** Mechanical breakers or contactors may draw arcs during opening, especially in DC since there is no natural current zero-crossing.

From a system perspective, bipolar DC grids introduce additional fault paths compared to unipolar DC. Besides pole-to-pole arcs, pole-to-neutral arcs may occur due to ground potential shifts or damaged insulation. The neutral conductor provides efficiency and redundancy benefits, but it also complicates fault detection since return currents may mask low-energy series arcs [**Miao2022**].

2.2.2. DC AC arcs

The main reason that DC arcs are more complex to deal with than AC arcs is the absence of current zero-crossings. In AC systems, the arc naturally extinguishes when the current passes through zero, which occurs 100 or 120 times per second. This gives protection devices a built-in advantage. In DC systems, there is no such moment. Once a DC arc starts, it can keep going until something actively breaks the circuit or the voltage drops below the sustaining threshold. This makes DC arcs inherently more persistent and harder to interrupt.

Another issue is the stability of DC arcs. At short gap lengths, DC arcs often settle into low-voltage, low-energy states that help them stay lit. This is not as common in AC, where the changing current compromises stability. Because of this, DC arcs can burn longer and cause more gradual but severe damage, especially in systems with high impedance like PV strings. While AC arcs dump energy faster during the initial phase, DC arcs spread it out over time, which keeps the arc going and heats up components more consistently.

Unlike AC faults, DC arcs are sustained continuously and their current magnitude often remains comparable to normal load current levels. As a result, traditional overcurrent devices may never see a large enough excursion to trip, and protection schemes that rely on detecting rapid edges or zero crossings (common in AC-based relays) simply do not operate reliably in DC systems. Instead, effective DC arc detection must look for the characteristic aperiodic fluctuations and broadband noise signatures of the arc itself, for example via time–frequency methods, rather than depend on fast current transients or periodic thresholds [15].

2.2.3. Bipolar DC Grid Architecture and Protection Challenges

Bipolar DC grids are increasingly proposed for future low-voltage distribution networks, often operating at ± 350 V with a neutral conductor. This architecture allows both ± 350 V and 700 V connections, improving system efficiency and flexibility. At the same time, it introduces additional complexity for protection, as the neutral conductor may shift due to unbalanced loading or ground loops .

The bipolar structure provides clear advantages over unipolar DC systems:

- Higher power transfer capacity for the same conductor size, which reduces cable losses.
- Possibility of redundant operation. If one pole is disconnected, the grid can continue to operate at half the nominal voltage.
- Better integration with renewable sources such as photovoltaic panels and battery storage, which can be coupled to different voltage levels.

However, protection challenges are more severe. In case of a series arc fault, the limited fault current from power electronic converters may not be sufficient to trigger conventional overcurrent protection. Moreover, pole-to-neutral arcs become a concern in bipolar grids, as insulation failures can lead to return currents. The detection system must therefore be able to discriminate between normal switching noise and fault-induced arcs under a variety of load and cable conditions [16]

Solid-state protection is particularly relevant in this context. Fast electronic breakers can interrupt current flow without relying on natural current zero-crossings, making them suitable for low-energy arcs that are invisible to mechanical devices. When combined with arc detection algorithms, they enable selective and reliable protection of bipolar DC grids [17].

2.3. Detection Techniques: Hardware and Software Approaches

2.3.1. Arc Model and Detection Boundaries

To make arc detection more efficient and physically more accurate, the current-voltage boundaries where an arc can realistically occur are defined. Not all combinations of voltage and current can sustain an arc. For example, an arc cannot be maintained at 350 V and 0.01 A, nor at 30 A and 5 V. These zones are ignored in our algorithm.

To determine where arcs are possible, Paukert's extended model is used [18], since they describe the voltage and resistance of arcs based on the arc current and electrode gap. This equation was derived from a large set of experimental data with different gaps and arc conditions

$$V_{\text{arc}} = k \cdot I^m, \quad R_{\text{arc}} = \frac{V_{\text{arc}}}{I} \tag{2.4}$$

The values of k and m depend on the electrode gap and are taken from the test results summarized in Table 2.2.

Table 2.2: Arc Parameters for Paukert Model (Valid for $I < 100\,\text{A}$)

Gap (mm)	k	m
1	36.32	-0.124
5	71.39	-0.186
10	105.25	-0.239
20	153.63	-0.278
50	262.02	-0.310
100	481.20	-0.342
200	662.34	-0.283

Figure 2.2 shows how the arc voltage and arc resistance vary with current for different gap distances, for currents up to 35 A and voltages up to 450 V. These boundaries define where the arc detection is physically possible [19].

This model is used as a filter in the detection algorithm. If the available voltage is lower than the computed arc voltage for the measured current and gap, arc detection is skipped:

$$\text{If } V_{\text{dc}} < V_{\text{arc}}(I_{\text{measured}}, \text{gap}) \quad \Rightarrow \quad \text{no arc possible} \tag{2.5}$$

This helps the system ignore zones where arcs can't occur, reduces false positives, and improves computational efficiency.

2.4. Principle of AC Arc Fault Circuit Interrupters

In alternating current networks an arc often extinguishes and then reignites when the current passes through a zero crossing. This repeating action produces clear electrical features, such as bursts of high-frequency noise and short transient disturbances. Arc Fault Circuit Interrupters (AFCIs) use these predictable signatures to tell dangerous arcs apart from normal operation.

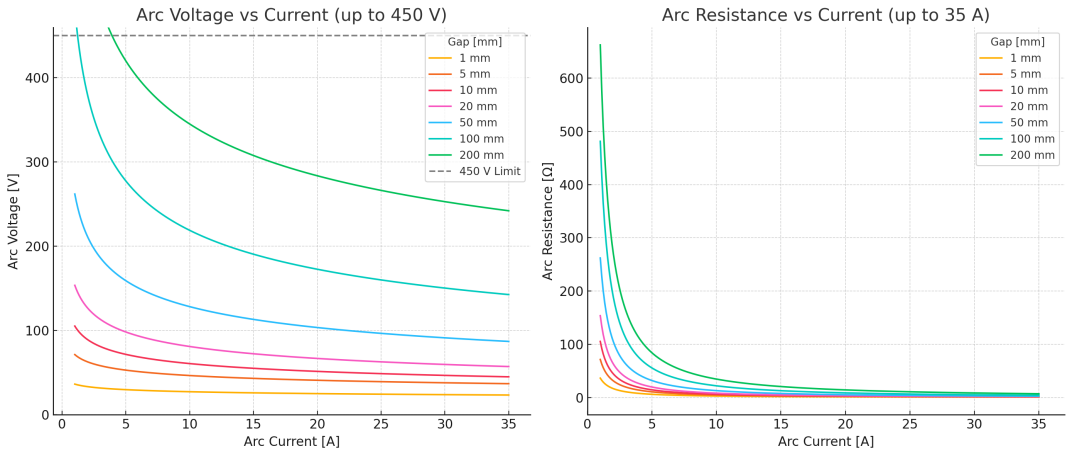


Figure 2.2: Arc voltage and resistance versus current for various electrode gaps. Limits are shown up to 450 V and 35 A.

Most commercial AFCIs adopt a fully analog implementation. A typical circuit contains

- **band pass filters** that keep only the frequency range where arc noise is strongest;
- **envelope or peak detectors** that follow the amplitude of the filtered signal; and
- **comparators with logic stages** that open the circuit if an arc like signature stays above a preset level for a defined time.

This analog approach is simple, low cost and gives a prompt response, so it suits breaker panels, outlet modules and portable safety devices used in residential wiring.

2.4.1. Limitations of AC AFCIs

Even though their performance is well-proven, AC AFCIs are not flawless. Low-energy arcs or arcs that appear in complex cable connections may sometimes pass undetected. Equipment with poor electromagnetic compatibility can also create signals that look like an arc, which may result in nuisance tripping.

A more fundamental limitation arises from the working principle itself. Detection depends on the presence of a natural current-zero crossing. Direct current systems do not offer such a pause in conduction, so the same technique cannot be transferred without major changes. As a result, different methods are required for reliable arc detection in DC installations.

2.5. Analogue front-end filters for arc-fault sensors

Most arc-fault detectors shape the sensor signal in hardware before the ADC. Four topologies appear repeatedly in Fault Detection:

Figure 2.3 sketches how the four classical families behave:

- **Elliptic (Cauer).** Steepest roll-off for a given order, but shows ripple in *both* pass-band and stop-band. Good when size is tight and a little ripple is acceptable.
- **Chebyshev.** Type I keeps ripple only in the pass-band, Type II in the stop-band; both roll off more slowly than Elliptic yet clearly faster than Butterworth .
- **Butterworth.** Amplitude is perfectly flat (maximally flat), but the phase is nonlinear. The delay distortion can be corrected afterwards with a small digital equaliser, so Butterworth remains popular when a clean pass-band matters and some DSP resources are free.
- **Bessel.** Gives the most linear phase and therefore the best step response, yet its –20 dB/dec slope is too gentle for cutting out unwanted frequencies, so it is rarely used in arc-fault breakers.

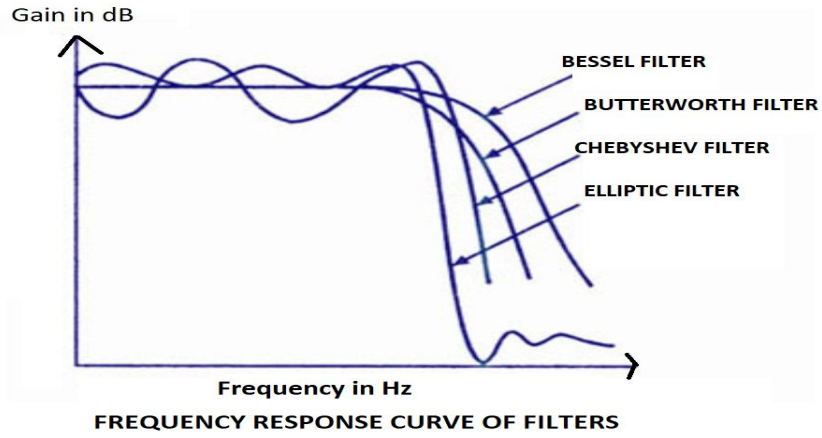


Figure 2.3: Typical magnitude curves of Bessel, Butterworth, Chebyshev and Elliptic low-pass filters. The softest slope belongs to Bessel, the steepest to Elliptic. From [20]

2.6. Signal Processing for Fault Detection

Five main methods appear in DC Arc Detection according to [11].

1) Time-domain signatures. Arc inception often produces an abrupt step or shoulder in the string current or voltage. Typical detectors watch the derivative, peak-to-peak span, or energy inside a short ($<$5 ms) window, then trip if the change exceeds a threshold. *Advantage*: executes in microseconds and does not require spectral maths. *Limitation*: load steps or converter start-ups create similar transients, so false trips are common in practical PV strings.

2) Statistical methods. Instead of hard thresholds, simple statistics (variance, deviation) , RMS value, di/dt, track how the waveform spreads over time. *Advantage*: tolerates moderate noise and adapts the thresholds automatically. *Limitation*: still relies on a few hand-picked features, so performance drops when the operating point drifts far from the training set.

3) Frequency-domain (FFT) analysis. An FFT reveals the broadband noise (1–150 kHz) injected by a burning arc; energy in these bins is summed and compared with a baseline. *Advantage*: algorithm is short, memory-light, and already available in most DSP libraries. *Limitation*: assumes the signal is steady within the window; inverter switching harmonics can mask the arc or raise the counter and cause nuisance trips.

4) Wavelet (DWT) time–frequency analysis. The dyadic DWT decomposes the trace into octave bands; detail level 2 (≈ 37–75 kHz) carries the clearest arc burst while remaining immune to inverter noise. *Advantage*: pinpoints short, non-stationary events and cuts false alarms in noisy strings. *Limitation*: computational load is higher than a single FFT; an embedded implementation must use a short (e.g., db3) filter to stay within a 2 ms frame.

5) Model-based and machine-learning approaches. Physics-based arc models (Cassie–Mayr, Habedank) estimate whether the supply can sustain an arc; data-driven classifiers from k-NN to lightweight CNNs learn joint statistics of FFT or DWT features. *Advantage*: highest reported accuracy ($>$99 %) and adaptability across cable lengths and load types. *Limitation*: requires a representative data set and, for deep models, more memory than a low-cost MCU offers; interpretability is also poorer than threshold methods.

2.6.1. Limitations of Time Domain Methods for Arc Detection

Time domain techniques, which observe current or voltage as a direct function of time, are rarely used alone for detecting DC arc faults. Although the basic idea is straightforward, several practical drawbacks limit their reliability in real installations that include switching loads, inverter noise, and other unpredictable behaviour. Arc events are brief and irregular, so the corresponding time-domain signatures are difficult to separate from normal disturbances. For example, an abrupt load step can create a transient that resembles an arc even though it is part of regular operation. This similarity makes false

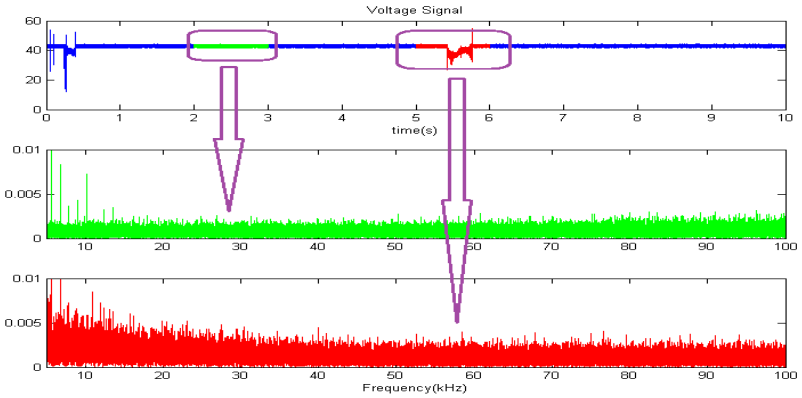


Figure 2.4: Load voltage and FFT results for nonarcing and arcing part of the signal from [1]

alarms likely when only time-domain features are considered.

A further obstacle is that arcs do not follow a repeating pattern. Traditional tools such as threshold checks, slope monitoring, or root mean-square tracking therefore yield unreliable results. When an arc lasts only a few milliseconds it can be buried in noise or smoothed by filters, making it hard to mark the exact start and end of the event.

Time domain methods are also sensitive to external influences. In systems that contain power converters or large capacitors, normal behaviour may hide or imitate an arc signature, which further reduces detection accuracy. Because of these limitations, many studies have shifted towards frequency domain or combined time and frequency approaches, including the Fast Fourier Transform and wavelet analysis [21, 22].

2.6.2. Fast Fourier Transform for Detection

The Fast Fourier Transform (FFT) is widely applied because it reveals changes in signal energy that appear when an arc develops. Arc faults inject noise at higher frequencies, typically from 1 kHz to 100 kHz, that is absent during normal operation. These components stand out clearly in the frequency domain, so a controller can quantify the extra energy and use it as an indicator of arcing activity. Figure 2.4 shows the difference in frequency domain when an arc occurs and during normal operation.

The FFT is attractive because it is simple to implement on digital controllers, requires little memory, and demands modest processing effort. Some commercial detectors combine an analogue band-pass filter that focuses on a chosen frequency band with an FFT that checks whether the magnitude of specific bins exceeds a preset threshold.

Despite these benefits, the FFT assumes signals that are steady and periodic, whereas arc faults occur suddenly and are not regular. In converter-based systems the switching noise can mask the arc signature or trigger a false alarm. For this reason, many recent designs turn to wavelet analysis, which copes better with transients.

2.6.3. Fast Fourier transform (FFT)

For each 512-sample window $x[n]$ the discrete Fourier transform is

$$X[k] = \sum_{n=0}^{N-1} x[n] \mathrm{e}^{-j2\pi kn/N}, \qquad N = 512.$$

A direct sum costs N^2 complex multiplies, but the radix-2 FFT cuts the work to $N \log_2 N$. With $N = 512$ the count falls from 262 144 to 4 608 operations, about $4\,\mu\mathrm{s}$ on the STM32G484.

Broadband arc noise lifts the spectrum between $1\,\mathrm{kHz}$ and $150\,\mathrm{kHz}$ while the dc bin stays flat, so one feature is

$$E_{\mathrm{FFT}} = \sum_{k=1}^{k_{150\mathrm{kHz}}} |X[k]|^2,$$

where bin 0 is dropped and bins are summed up to the analogue cut-off at $150\,\mathrm{kHz}$.

2.6.4. Benefits of the Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) analyses a signal in both time and frequency spaces simultaneously, which suits events that are short and unpredictable. Unlike the FFT, which returns frequency information without accurate timing, the DWT decomposes the signal into several scales and identifies when a specific frequency component appears. Short bursts produced by arcs, therefore, become easier to locate.

Wavelet-based detectors also reduce false alarms. In networks subject to inverter switching noise or frequent load changes, the multiresolution view given by the DWT separates true arc signatures from benign variations, improving accuracy in noisy environments.

The DWT can run in real time on embedded hardware. With efficient filter banks and a limited number of coefficients, microcontrollers such as the STM32 family achieve low latency, which meets the needs of low power and fast response applications. For these reasons the DWT is increasingly preferred in modern arc fault detectors to raise the signal-to-noise ratio, capture brief events, and keep computational overhead low.

2.6.5. Discrete wavelet transform (DWT)

The DWT uses a two-channel filter bank. At each scale j

$$A_{j+1}[k] = \sum_n h[n-2k]A_j[n], \qquad D_{j+1}[k] = \sum_n g[n-2k]A_j[n],$$

so the total cost grows only with N. Figure 2.5 sketches the five-level db3 cascade used. Tests show that the detail band at level 2 of a db3 wavelet ($37.5\,\mathrm{kHz}$ to $75\,\mathrm{kHz}$) gives the best arc-to-noise ratio. The wavelet feature is

$$E_{D2} = \sum_k |D_2[k]|^2.$$

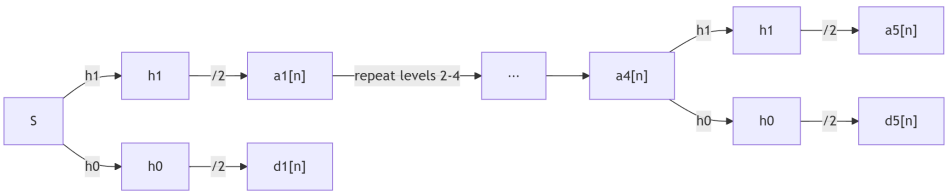


Figure 2.5: Multiresolution db3 filter-bank used for the discrete wavelet transform. The second-level detail branch $D_2[n]$ (lower path of level 2) supplies the E_{D2} feature.

2.6.6. Wavelet decomposition options and chosen mother wavelet

The standard (dyadic) DWT splits only the low-pass branch at each level, so it gives a coarse, logarithmic frequency map with few coefficients to process. The wavelet-packet transform (WPT) splits both branches and therefore gives equal-width bands, but the workload doubles at every stage. The stationary wavelet transform (SWT) keeps the original sample rate and is shift-invariant, yet its memory and multiply counts grow quickly. Tests on synthesised arc data show that the dyadic tree reaches the same detection accuracy as WPT and SWT while running about twice as fast on a small MCU [6].

Table 2.3: Processing time per 256-sample frame ($f_{\mathrm{s}} = 300$ kS/s)

Wavelet	Time [s]	Tap length
db3	0.60	6
db4	0.72	8
db5	0.84	10
db7	1.16	14
db9	1.46	18

The coefficient-of-variation of the level-2 detail energy stayed below 2 %, whereas level 1 fluctuated by ±7 % because of inverter noise. The level-2 energy rises by an order of magnitude during arcing while remaining stable in load steps, meeting the IEC 63027 false-trip requirement [23].

A mother wavelet for real-time work must be short, orthogonal, and have enough vanishing moments. Within the Daubechies family, `db3` is the shortest (6-tap) filter that still captures the sharp arc edges. On an STM32 the full five-level `db3` bank needs only 0.6 s per 256-sample block, leaving plenty of time for the classifier [24]. Higher orders such as `db9` improve stop-band steepness but raise run-time by more than 50 little extra sensitivity, so `db3` is kept in the firmware.

2.7. Key Signal Parameters for DC Arc-Fault Detection

Many different signature indicators are used in arc-fault research for photovoltaic and DC-microgrid applications. The five parameters most frequently mentioned are (i) Pulse Count (PC), (ii) Absolute Sum of FFT Magnitudes, (iii) Four-level Wavelet-decomposition Energy, (iv) Current Slew Rate di/dt, and (v) Number of Peaks. Each parameter is defined below together with its physical meaning, its value for distinguishing a sustained DC arc from normal or switching noise, and its main limitation in practice.

Pulse Count (PC) is the number of rectified high-frequency current (or voltage) pulses whose instantaneous amplitude exceeds a preset threshold within a sliding time window.During a series arc the plasma column collapses and re-ignites randomly, creating many short, sharp impulses. Normal converter switching is regular, so the pulse counter increases far more slowly. PC is sensitive to the chosen threshold and to inverter carrier frequency; high-frequency electromagnetic interference (EMI) can artificially inflate the count.

Absolute Sum of FFT Magnitudes is the scalar $S_{\mathrm{FFT}} = \sum_{k=k_1}^{k_2} |X[k]|$, where $X[k]$ are the discrete-Fourier-transform coefficients within a selected high-frequency band (typically 20–200 kHz) of each analysis window. An arc injects broadband energy, so the summed spectral magnitude rises markedly, whereas normal switching concentrates energy at a few discrete harmonics. A high sampling rate is required to capture the band of interest; the result is sensitive to background noise and window length.

Four-Level Wavelet-Decomposition Energy is $E_{d4} = \sum_n d_4[n]^2$, where $d_4[n]$ are the level-4 detail coefficients of a dyadic discrete-wavelet transform (DWT) using a short Daubechies or Symlet mother wavelet. The multiresolution nature of the DWT localises the wide-band, non-stationary arc noise; level-4 (or similar) covers the 10–40 kHz range in a 100 kS/s system, where arc energy is strong and inverter switching energy is weak. The energy measure therefore rises sharply during arcing but remains almost constant during normal operation. Computational burden is higher than simple FFT or slope tests; the choice of mother wavelet and decomposition level affects sensitivity, and low signal-to-noise ratio (SNR)

conditions may mask the rise.

Current Slew Rate di/dt is the maximum absolute slope of the line current calculated over a short centred difference inside each window. A series arc inserts a dynamic resistance; when the arc quenches or re-ignites the current changes abruptly, producing a larger di/dt than that caused by converter PWM or load steps. Cable inductance limits the observable slope, so distant faults may be missed; fast load changes or EMI spikes may mimic the arc signature unless an adaptive threshold is used.

Number of Peaks is the count N_p of local maxima in the band-pass-filtered current (or voltage) that exceed a fixed multiple of the root-mean-square background value within a time window. Arc noise is irregular and produces many random high-frequency peaks, while normal switching yields periodic peaks that are far fewer. A sudden increase in N_p therefore signals arcing. The method is threshold-dependent and susceptible to broad-band EMI; counts can be inflated by measurement noise, leading to nuisance trips if the filtering is not carefully designed.

The algorithm keeps two features: the *integrated FFT energy* and the *level-2 wavelet energy*. The FFT sum is simple to code, runs in about $4\,\mathrm{s}$ for a 512-point array on the STM32, and highlights the extra broadband noise that a burning arc adds between roughly 0–200 kHz while normal switching stays at fixed harmonics. Using one scalar also cuts memory and keeps the threshold logic clear. Pure time-domain checks such as di/dt were not selected because cable inductance and fast load steps can create the same slopes, especially when the string is long .

The discrete-wavelet transform fills the gap that the FFT leaves. The level-2 detail band (about 37–75 kHz at the chosen sample rate) catches short bursts that mark each reignition, yet ignores most inverter noise below and above that range. A db3 filter bank needs only 0.6 ms per 256-sample frame, well inside the 2.5 s clearing time. Taken together the FFT gives a steady “average” view and the wavelet points out the short spikes; the pair $[E_{\mathrm{FFT}},\, E_{D2}]$ separates arc and normal data.

2.7.1. Other Threshold−Based Detection Mechanisms

Once a fault signature has been extracted, most practical arc-fault detectors still decide between the *normal* and *arc* states by checking whether that signature exceeds a pre-defined limit. Three major ways of thresholding were identified families: (i) single-point threshold / comparator, (ii) multi-location comparison, (iii) support-vector-machine (SVM) classifiers.

(i) Single-point thresholds. A single comparator looks at one scalar taken from the line current and trips whenever that value passes a fixed limit . This method is attractive since the detector needs only one sensor and the math is very simple to implement, so it can run in a small micro-controller. On the other hand, because the limit is static, it must be set high enough to survive worst-case inverter noise and therefore misses weak arcs or gives nuisance trips when the noise floor drifts.

(ii) Multi-location comparison. Here the same quantity is measured at two places e.g. the combiner and the inverter and the two traces are subtracted or cross-correlated. The common-mode switching harmonics cancel, while any localised arc, seen only by the upstream sensor, stands out. This method presents much better immunity to conducted EMI than the single-point scheme. But it doubles the sensor count and the signals must be time-aligned to within one sample; a loose clock or long cabling spoils the result.

(iii) Support-Vector Machine (SVM). An SVM is trained off-line with labelled “normal’’ and “arc’’ feature vectors typically FFT energy plus wavelet detail energy then, in service, classifies each new window by the sign of the hyper-plane. The margin acts as an adaptive, multidimensional threshold, so the same model works over different strings and operating points without re-tuning. The main disadvantage is a modest data set is needed for training and the dot-product evaluation costs more CPU than a single comparator.

2.8. Signal normalization techniques accounting for frequency bias

In arc-fault analytics most of the signal energy lies in the lowest frequency band, yet the indicators that reveal a developing arc appear as broadband bursts at higher frequencies. When the raw band powers

are fed directly to a classifier, the dominant low-frequency term outweighs the rest by several orders of magnitude. The learning algorithm then focuses almost entirely on that term and becomes sensitive to operating-point drift and load variations, or else scales its weights down so far that the contribution of the higher bands is practically lost.

One way to address the imbalance is to divide each spectral descriptor by the total in-band energy. Each detail-band energy $E_{\mathrm{L}k}$ and the corresponding FFT measure E_{FFT} are scaled with a single common divisor, namely the sum of the five wavelet detail energies. In this way every feature is mapped to a dimensionless value between zero and one. Any change in absolute current level, sensor gain, or cable impedance shifts all bands by the same factor, so the ratio remains stable. Because the features now share a comparable range, a linear classifier such as a support-vector machine converges faster and with a lower risk of bias[6].

The firmware implementation is short. After the five Db-3 detail powers are accumulated, the code calculates

```
float32_t E_sum = powerLevel1 + powerLevel2 +
                  powerLevel3 + powerLevel4 +
                  powerLevel5;
powerLevelk /= E_sum;
```

and then low-pass filters the normalised level-2 energy, the band that carries the most distinctive arc burst, before passing the value to the trip logic. The broadband energy obtained from the FFT is normalised by the same E_{sum} (or, if preferred, by the total FFT energy so that the two feature families remain separate).

2.9. Machine Learning in Fault Detection Systems

Commercial dc arc-fault circuit-interrupters (AFCIs) still rely on fixed thresholds or narrow band-pass energy counters. Such products detect barely half of the dangerous events and sometimes nuisance-trip during normal operation, especially on 220–240 V PV strings and home batteries [7]. Conventional algorithms also struggle with the strongly non-linear physics of an electrical arc and with the masking effect of inverter harmonics or dc–dc switching noise [25]. These safety gaps have triggered both regulatory and industrial interest in smarter, data-driven detectors.

Why static thresholds are not enough series and parallel arcs show large variability in their electrical “fingerprints”. The high-frequency burst that accompanies an arc changes in amplitude and bandwidth with cable length, irradiance, converter topology and crucially the type of load. Static limits therefore, fail; an adaptive classifier that learns joint statistics from several features is required.

Four families of ML detectors.

- **Classical pattern-recognition:** PCA, k-NN, decision trees
- **Probabilistic / novelty detection:** Gaussian mixtures, one-class SVM
- **Support-vector machines:** linear, RBF, LS-SVM, weighted SVM
- **Deep-learning networks:** 1-D CNNs, lightweight transformers

Classical and probabilistic detectors Early statistical models work when only a few features are considered. Combining peak-to-peak current and its variance, a simple k-NN achieved 97–99% accuracy on a laboratory test bench, but performance dropped once inverter harmonics or partial shading were introduced [6]. Gaussian-mixture novelty detectors reduce false trips when mostly “healthy” data are available, yet they lose accuracy when the operating point drifts far from the training set [11].

SVMs are attractive because they perform optimally and cope well with high-dimensional but small training sets. A weighted LS-SVM fed with just two inputs wavelet-packet energy and current integral hit 99 % detection accuracy in fire-safety tests and executed in real time on prototype hardware [7]. Wang *et al.* streamed a five-level DWT plus linear SVM on an STM32F407, consuming less than 20 kB RAM [1].

Deep-learning approaches Data-hungry CNN and transformer models report 96–100 % accuracy with < 2% false positives, but they require larger memory and training sets. A recent real-time detector that analyses noise patterns with a DWT front-end and iterative loop on a DSP reached sub-30 ms reaction while avoiding inverter-noise false trips [26]. Combining adaptive normalisation with several lightweight learners further improves robustness. An ensemble using wavelet features maintained 99.8 % accuracy across different converter types and constant-power loads, confirming that model diversity helps when site conditions vary [27].

2.9.1. Support Vector Machine

SVMs balance accuracy and hardware cost. Only the support vectors and two scalar hyper-parameters are stored typically a few hundred bytes ideal for the 64 kB SRAM of a mid-range STM32. The decision boils down to one dot product and a bias add , easily within the 50 ms limit prescribed by UL-1699B. Because a five-level Daubechies-4 DWT is already computed for spectral monitoring, adding the SVM costs negligible extra CPU time.

Table 2.4 summarises the main ML families and representative results.

Table 2.4: Typical AI methods for DC-arc-fault detection

Family	Representative study	Reported accuracy
PCA + k-NN	Wang *et al.* [6]	97–99 %
Gaussian mixture	Yao *et al.* [11]	96–98 %
Weighted LS-SVM	Yang *et al.* [7]	99 %
DWT + Linear SVM	Wang *et al.* [1]	99 %
CNN with noise filtering	Ahn *et al.* [26]	98–100 %
Ensemble + DWT	Miao *et al.* [27]	99.8 %

Support-Vector-Machine (SVM) Classifier

Concept and margin idea An SVM places a hyper-plane that maximises the geometric margin between the two classes *arc* and *normal*. Only the few training samples nearest to this boundary, the *support vectors*, are kept after training, so both memory use and run-time cost stay small.

Linear convex optimisation problem With only two features the training task stays in the original two-dimensional space, so it becomes a *strictly convex* quadratic programme: the objective $\frac{1}{2}\|\mathbf{w}\|^2$ is strongly convex and all constraints are linear. Adding non-negative multipliers α_i (for the margin) and μ_i (for the slack) gives the Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \big[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 + \xi_i\big] - \sum_i \mu_i \xi_i.$$

The Karush–Kuhn–Tucker (KKT) conditions, stationarity, and feasibility ensure a single global optimum, at which

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i, \qquad 0 \le \alpha_i \le C.$$

Only the samples with $\alpha_i > 0$ survive as *support vectors*; they alone fix the hyper-plane and therefore the firmware size. At run-time the classifier is simply

$$f(\mathbf{x}) = \operatorname{sign}\big(\mathbf{w}^\top \mathbf{x} + b\big),$$

which on the STM32 means one dot product, one bias add and a sign test.

If the hinge loss is replaced by a squared loss the problem turns into a least-squares SVM (LS-SVM). Then the dual optimisation collapses to one linear solve, cutting off-line training time while the run-time form of $f(\mathbf{x})$ and its memory cost stay the same [7, 1].

Why a linear kernel is sufficient The selected features the second-level wavelet energy E_{D2} and the FFT-band energy E_{FFT} already separate arc and normal data well. A kernel lift would add little benefit while increasing the size of the support-vector table, which matters on a micro-controller.

Chapter 2 — Closing note
Chapter 2 answers *Research Question 1*. By comparing time-domain counters, frequency-domain integrals, and hybrid wavelet–FFT methods, it lists what each approach does well, where it falls short, and why none alone covers every operating case in a DC micro-grid. This gap sets the scene for a combined solution in the next chapters.

3

System Description

This chapter shows the path of the first ideas in software developing to a working detector in the lab. It begins with small *Octave* scripts that load measured waveforms and try out many feature choices. When a stable set of features is found, the same logic is ported to a STM32 *Nucleo* board, proving that the code can run on a normal microcontroller. The board is then placed for a simple test bench that supplies DC power, produces repeatable series arcs, and records the signals. The next sections describe the hardware, the sensing chain, and the safety measures used during the tests.

3.1. System Setup

A *Delta Elektronika* programmable supply set to $350\,\mathrm{V}$ and current-limited to $1.5\,\mathrm{A}$ is connected in series with a mechanical series-arc generator seen in Figure 3.1, originally built by a previous intern[1], which opens and closes a copper-electrode gap any length up to 20mm; in the same line a fixed $330\,\Omega$, resistor sets the nominal load so the steady current is about $1.06\,\mathrm{A}$. Voltage is measured with a PicoScope across the load. All parts sit in a clear plastic enclosure with an emergency stop that disconnects the supply.

[1] V.E.A. van Didden, “Vonkboogdetectie in een DC-netwerk,” SCRIPTIE AFSTUDEERSTAGE , 2019.

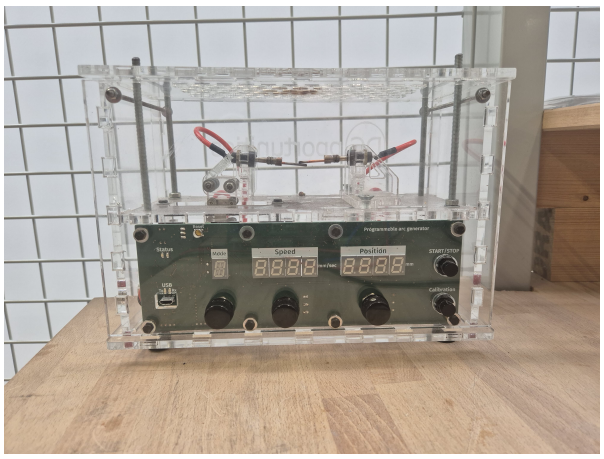


Figure 3.1: Arc Fault Generator given by DC Opportunities

3.2. Octave implementation

3.2.1. Octave verification of the FFT and DWT indicators

To verify that the two signal processing techniques respond to a DC arc before any embedded work began, the voltage signals were processed in GNU Octave 8.4.0. The script used only the `signal` toolbox and the LTFAT wavelet functions (`wfbt`, `iwfbt`); the full code is given in Appendix A.

Each PicoScope record contains one 5 MS/s voltage trace (8-bit, 10 MHz BW) about two seconds long. Octave resamples every trace to 250 kS/s with `resample()`, replaces any NaNs or Infs by zero, and applies a fourth-order Butterworth band-pass from 1 kHz to 150 kHz (`butter`/`filtfilt`). This matches the analysis used in [28].

Eight non-arc files are used for the reference set. For each one, the script computes

$$E_{\text{FFT}} = \frac{1}{N_b} \sum_{f=1\text{ kHz}}^{150\text{ kHz}} |X(f)|^2, \qquad E_{\text{DWT}} = \frac{1}{N} \sum_{k} D_2[k]^2,$$

where $X(f)$ is the FFT of the filtered trace and $D_2[k]$ is the level-2 detail of a four-level db4 packet.

Test run A separate capture that includes an arc (ignition at 20 ms) is processed with the same steps.

The files recorded from the PicoScope were exported as `.csv` and processed . Among the four detail bands, the second one provided the highest separation: the arc condition yielded $E_{d,2} = 1.5 \times 10^8$, while the non-arc condition produced 5.9×10^7, resulting in $\text{GER}_2 \approx 2.6$. Other levels remained below 1.4.

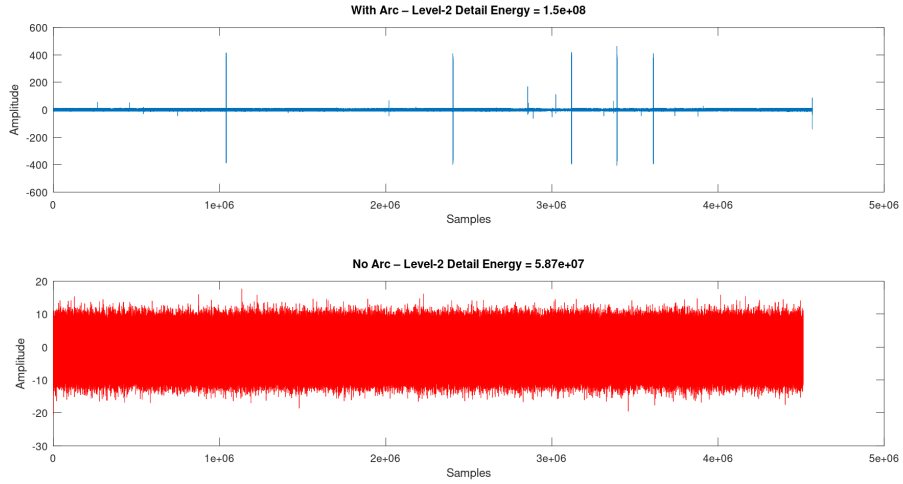


Figure 3.2: Detail-2 energy in the presence (top) and absence (bottom) of a series arc. Both traces were normalised before the DWT; the observed difference is therefore purely spectral.

The ADC embedded in the STM32, limits the analysable band to $1.5\,\text{kHz}$–$150\,\text{kHz}$. At such rates, the bandwitch of the ADC is not wide enough to capture all transients, thus the second wavelet level would not capture the burst visible in Fig. 3.2. The Octave study therefore served to (i) confirm that a wavelet indicator performs reliably and (ii) an analogue pre-filter that shifts the band into the micro-controller's Nyquist window is needed.

3.3. Nucleo Board Implementation

Sampling frequency check on the Nucleo board

Two short tests on the STM32-Nucleo board were done to confirm that the FFT code and the sampling thery work as as intended before moving the code into the breaker.

FFT sanity check A $10\,\mathrm{kHz}$ sine was created while the ADC ran at $f_s = 90\,\mathrm{kS/s}$. The nucleo board produces an FFT of 512 bins and gives the plot in Fig. 3.3.

- The highest component is around the 0th bin, due to the big DC component of the sine wave (offset)
- The 10kHz sine wave shows as the second highest peak.
- A harmonic behaviour is also seen, with multiples of 10.000.

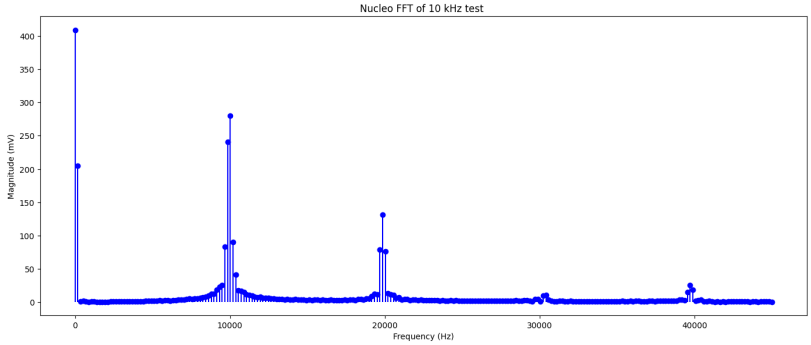


Figure 3.3: On-board FFT of a $10\,\mathrm{kHz}$ sine at $90\,\mathrm{kS/s}$.

Aliasing demo Next the ADC ran at $f_s = 200\,\mathrm{kS/s}$. Both $100\,\mathrm{kHz}$ and $200\,\mathrm{kHz}$ tones were inputted one after the other. Figure 3.4 overlays the spectra (first half only).

- The $100\,\mathrm{kHz}$ tone appears at its true place.
- The $200\,\mathrm{kHz}$ tone folds back to $100\,\mathrm{kHz}$ as predicted by $f_{\text{alias}} = |f_{\text{in}} - f_s|$.

Anything above the $100\,\mathrm{kHz}$ Nyquist edge will therefore pollute the band unless the $150\,\mathrm{kHz}$ second-order RC filter removes it.

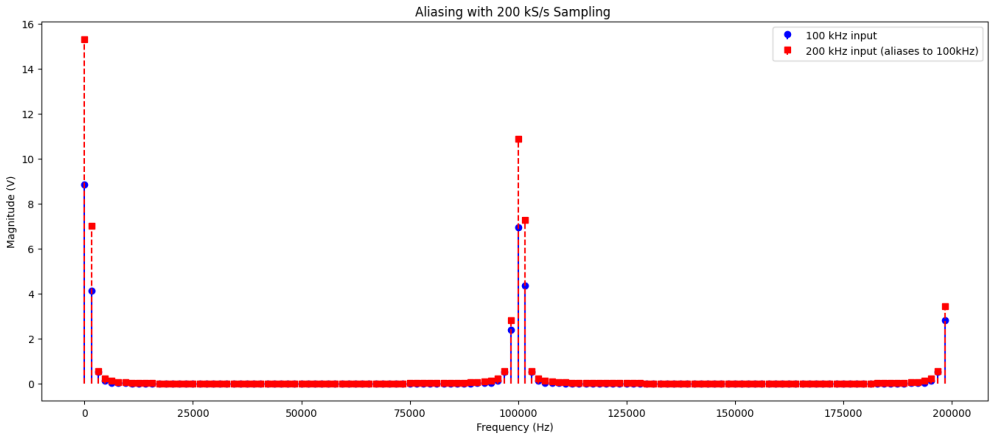


Figure 3.4: Aliasing at $200\,\mathrm{kS/s}$. The real $100\,\mathrm{kHz}$ tone (blue) and the aliased copy of a $200\,\mathrm{kHz}$ tone (red) overlap, showing the need for the analogue low-pass.

These two quick checks show that the Nucleo setup measures, transforms and scales frequency content as expected.

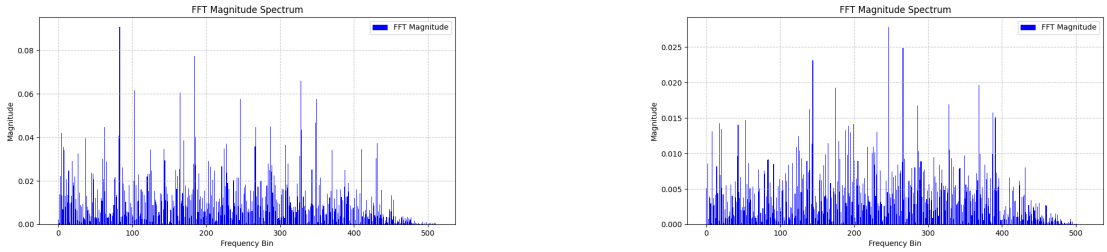
Definition of the D- and F-Factors

Two extra FFT metrics, called D and F, to go with the total energy E_{FFT} were suggested [6].

- F**-factor** – Weight for the first low frequency band (0–50 kHz). The amplitudes in this band are multiplied by this factor.
- D**-factor** – Percentage of FFT bins that are discarded. Intended to remove noise by assuming that it shows up as high peaks.

Why the D- and F-Factors are Suppressed

Figure 3.5 compares the current spectrum at 350 V / 1 A with and without a series arc. The arc lifts the average by about 200 % and adds sharp peaks.



(a) Arc present **(b)** No arc

Figure 3.5: FFT of the DC-bus voltage in the 1–100 kHz band.

Two problems appear:

(i) Raising D high enough to suppress the arc peaks also cuts real inverter harmonics, so arcs can slip through.

(ii) Lowering F enough to ignore random peaks makes the detector miss arcs whose noise sits above 35 kHz.

During testing both factors caused false negatives, so they are fixed to

$$D = 0, \qquad F = 1$$

3.3.1. Selection of the Diagnostic Wavelet Band

To determine which wavelet band offers the highest arc sensitivity while maintaining an acceptable noise level, a series of measurements were processed in Octave. Each record was decomposed with a three–level wavelet filter-bank transform (WFBT) based on the *db3* mother wavelet; the energy of every detail band was then extracted and compared. Table 3.1 summarises the energies obtained for bands D_1–D_4.

The C.o.V. is a unitless measure of relative spread, calculated as

$$\text{C.o.V.} = \frac{\sigma}{\mu} \times 100\%,$$

Table 3.1: Detail-band energies returned by the Octave prototype

Band	Centre-frequency range[1]	Mean energy	C.o.V.[2]	Qualitative SNR
D_1	$f_s/2$ f_s	5.79×10^5	$7\,\%$	poor – dominated by inverter carrier
D_2	$f_s/4$ $f_s/2$	1.84×10^5	$1.5\,\%$	good – clear arc bursts, moderate baseline
D_3	$f_s/8$ $f_s/4$	1.10×10^5	$2\,\%$	good, but absolute energy smaller
D_4	$f_s/16$ $f_s/8$	1.07×10^5	$3\,\%$	low – close to $1/f$ background

where σ is the standard deviation and μ is the mean energy measured in the band over all runs. A small value (e.g. 1–2%) means the band's energy hardly changes between data sets; a large value signals sensitivity to noise.

Although D_1 carries the highest absolute energy, as seen in (Fig. 3.6), that is because its content is dominated by high-frequency switching noise generated by the inverters. Band D_2 contains nearly one-fifth of the total signal energy and is centred an octave lower in frequency, approximately $f_s/4$–$f_s/2$, where inverter noise decreases sharply. Within this range, the arc energy is clearly shifted and shown in D_2; the same event remains unclear in D_1 due to noise, and become noticeably weaker in D_3 and D_4.

The procedure was repeated for ten additional data sets recorded. For these measurements the ratio of D_2 energy to the total detail energy varied by less than $\pm 1.5\,\%$, whereas the corresponding ratio for D_1 fluctuated by more than $\pm 7\,\%$. The results therefore suggest that D_2 offers a balance between arc visibility and immunity to electromagnetic interference.

Afterwards the same test was taken into the STM32. Figure 3.7 plots the instantaneous power in each monitored band while an artificial series arc is created. During normal operation ($\approx$ 333–368 s) all bands present a low energy. When the arc striker engages ($\approx$ 369–377 s), the energy in D_2 rises by more than one order of magnitude; D_3 exhibits a smaller yet distinct increase; D_1 remains saturated by high-frequency noise; and D_4 stays close to baseline.

Feature vector and firmware flow

The two features FFT energy E_{FFT} and wavelet energy E_{D2} form the vector $[E_{D2}, E_{\mathrm{FFT}}]$ that feeds the linear SVM.

Total latency from ADC trigger to fault flag is $\leq 40\,\mu\mathrm{s}$, easily within the timing budget. The pair of features captures the main arc signatures reported in recent studies while keeping memory and CPU load low.

3.4. Hardware Components: STM32 Microcontroller, Sensors, and Filters

The company **DC Opportunities** provided with the hardware where the algorithm is implemented. A DC Bipolar grid "breaker" was given. The breaker uses an STM32G484VETx (Cortex-M4F, $170\,\mathrm{MHz}$) as its controller. All three on-chip $12\,\mathrm{bit}$ ADCs operate in simultaneous mode so that the *positive*, *neutral* and *negative* conductors of the bipolar bus are sampled at the same instant. A buffer of 512 samples is filled, after which the DSP executes the discrete-wavelet transform and FFT in place. This happens 500 times per second, meaning that the sampling rate of both FFT and Wavelet energy points is 500Hz No external memory is required; code, tables and buffers fit inside the device's $512\,\mathrm{kB}$ flash and $128\,\mathrm{kB}$ SRAM.

Line current is measured by an MCA1101-50-3 Hall-effect IC ($50\,\mathrm{A}$ range, $3.3\,\mathrm{V}$ supply, fixed-gain analogue output). The sensor's bandwidth ($\approx 1.5\,\mathrm{MHz}$) exceeds the frequency content necessary for identifying a DC arc, so a passive second-order RC filter is added mainly to suppress switching spikes and to satisfy the Nyquist criterion. The values were limited by components already mounted on the breaker PCB and were therefore set to $R = 220\,\Omega$ and $C = 4.7\,\mathrm{nF}$, giving a $-3\ \mathrm{d}B$ corner close to $150\,\mathrm{kHz}$. The same filter is replicated on the positive, neutral and negative channels to keep phase skew below $\pm 1°$ at the analysis band edge.

Voltage channel omitted The breaker already measures the DC-bus voltage through a resistive divider for status and protection, so those ADC channels must track the full 0–$400\,\mathrm{V}$ range at $12\,\mathrm{bit}$ resolution. To use the voltage in the arc-fault algorithm, a zoom in on the millivolt-level ripple around the $350\,\mathrm{V}$ operating point is needed, which in turn would require either (i) dedicating a separate, high-pass-coupled ADC range, or (ii) sacrificing the existing monitoring.

Fig. 3.8 shows the FFT calculated in the STM32 during normal operation and during arcing. This confirms that the arc signature is not seen in the measurement.

Both options clash with the fixed hardware on the breaker, so voltage is retained solely for system health while the SVM relies exclusively on the current measurement.

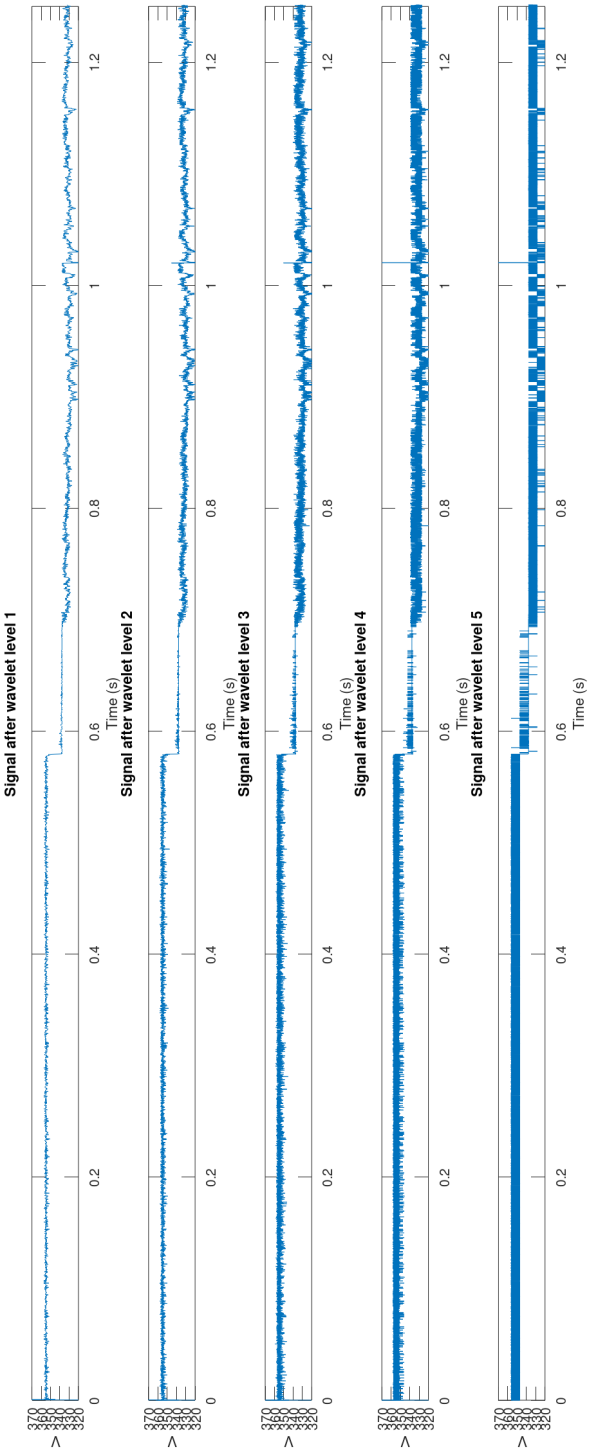


Figure 3.6: Different power bands during testing while arcing, processed in Octave.

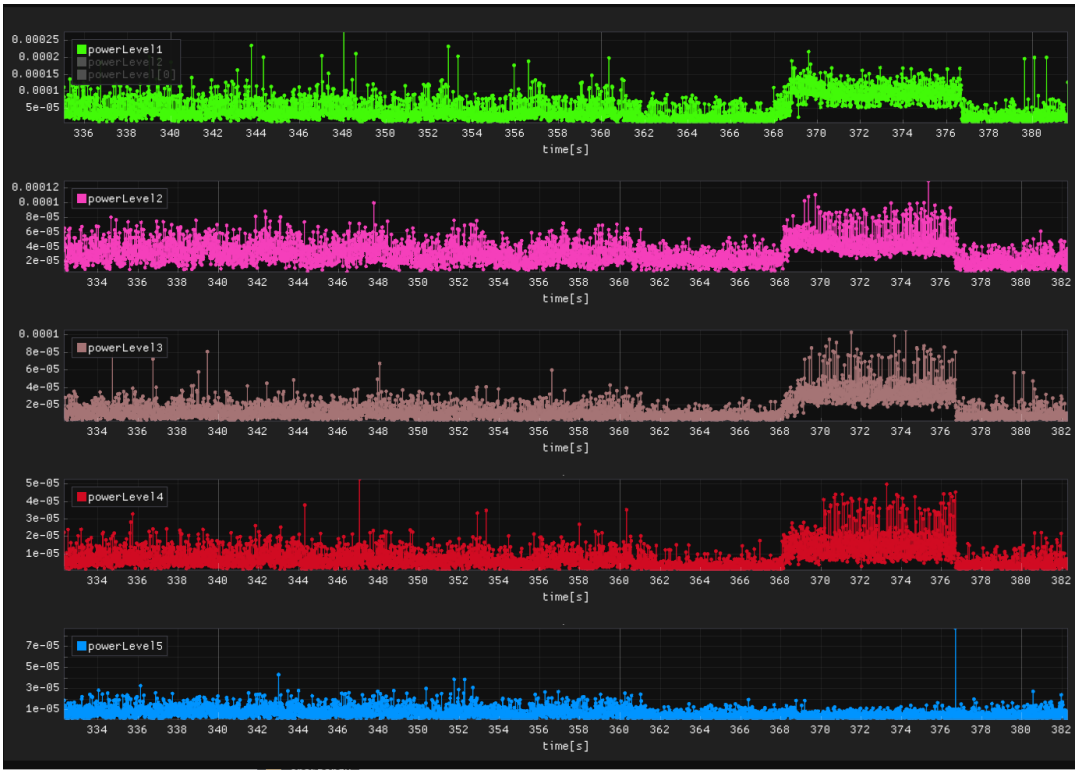
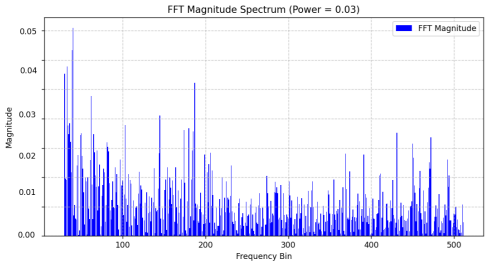
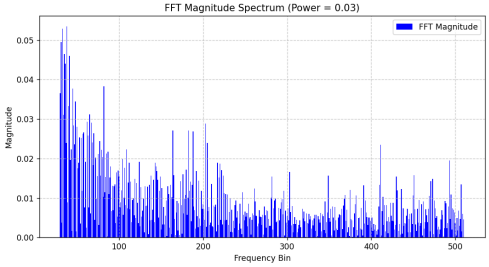


Figure 3.7: Different power Bands during testing on STM32 while arcing



(a) Voltage FFT during arc



(b) Voltage FFT during normal operation

Figure 3.8: Frequency spectrum of the DC-bus voltage. Both calculated arc energy are equal

After each window is processed the STM32 streams the two extracted features (E_{D2}, E_{FFT}) and the SVM decision word over a to an ESP32 module, which publishes the data to a Grafana dashboard via Wi-Fi. A galvanically-isolated STLINK-V3 probe supplies power and debugging I/O to the low-voltage side, ensuring that neither the PC nor the ESP32 shares a return path with the 350 V bus.

3.5. Signal Acquisition and Pre-processing Pipeline

Figure 4.1 shows all the processes that run inside the breaker. The aim is to capture arc energy in the $0\,\mathrm{kHz}$ to $150\,\mathrm{kHz}$ band while keeping the data rate low enough for an STM32-class micro-controller [29].

All three ADCs of the STM32G484VETx operate in *simultaneous* mode, triggered by TIM6 at $f_s = 300\,\mathrm{kHz}$. A double-buffered DMA moves each block of $N = 512$ samples ($\approx 1.71\,\mathrm{ms}$) to SRAM, so the DSP code can start as soon as a block arrives while the next one is still being filled. Non-overlapping windows are enough to meet the reaction limit of UL-1699B [11]; skipping overlap halves memory traffic.

3.5.1. ADC sampling budget and STM32G4 features

Sampling-rate constraints. For a target signal bandwidth $f_c = 100$ kHz the following minimum specifications apply:

$$\text{GBW}_{\min} = 100\, G\, f_c = 10 \text{ MHz}, \qquad \text{SR}_{\min} = 2\pi f_c V_{\text{p-p}},$$

where G is the front-end gain and $V_{\text{p-p}}$ the input swing. To avoid aliasing, the Nyquist rule doubles the bandwidth, so the converter must sample above $2f_c$; a margin is kept and $f_s = 300$ kS/s is chosen.

Bit-rate per channel. With a 12-bit resolution

$$\text{bit-rate} = f_s \times \text{resolution} = 300 \text{ kS/s} \times 12 = 3.6 \text{ Mbit/s}.$$

For four channels the bus traffic stays below 12 Mbit/s, well inside the DMA bandwidth of the STM32G4.

Key ADC data of the STM32G4 [30]

- Clock input up to 60 MHz (52 MHz in multi-ADC mode). Single-ADC sample rate $f_s^{\max} = 4$ MS/s; 3.46 MS/s with two ADCs coupled.
- Conversion modes: single, continuous, scan, discontinuous; triggers from timers or external pins.
- Extras: hardware oversampling, offset and gain compensation, interleaved mode, analogue watchdog [24].

The selected $f_s = 300$ kS/s therefore occupies barely $6\,\%$ of the single-ADC head-room, leaving timing margin for the five-level DWT and classification stages.

3.5.2. Analogue front end and anti-aliasing

Series arcs put noise far past $1\,\text{MHz}$. Because the ADC samples at $f_s = 300\,\text{kHz}$, any signal above the Nyquist edge ($150\,\text{kHz}$) would alias into the band we analyse and disturb the results. Oversampling could fix this but would flood the MCU with data, so a *second-order* RC low-pass right after the Hall sensor is added.

A second-order filter rolls off at $-40\ \text{dB/dec}$.

Options considered

- Higher-order analogue filters give steeper roll-off but need more PCB area and parts.
- Oversampling plus digital filtering removes aliases cleanly, yet the dataflow would have too many Msamples/s, which is too much for a $170\,\text{MHz}$ STM32 .

Figure 3.9 compares the two ADC choices. Oversampling above $2\,\text{MHz}$ (Option 1) would let the use of a milder analogue filter, but it increases data traffic , which is not doable since the interesting data is only between 0-150kHz. Nyquist-rate sampling at $300\,\text{kHz}$ with the $150\,\text{kHz}$ RC filter (Option 2) captures the arc spectrum yet keeps memory and CPU load small [29]. The chosen second-order network strikes the best balance: it keeps most out-of-band noise out while leaving enough head-room for real-time processing on the on-chip resources [29].

Chapter 3 — Closing note

Chapter 3 addresses *Research Question 2*. Different measurable quantities were explained and shows that level-2 db3 wavelet energy and wide-band FFT energy give the clearest gap between normal behaviour and a series arc. The chapter also proves that both features can be drawn from a DC line in real time with the resources of the chosen STM32G4 board.

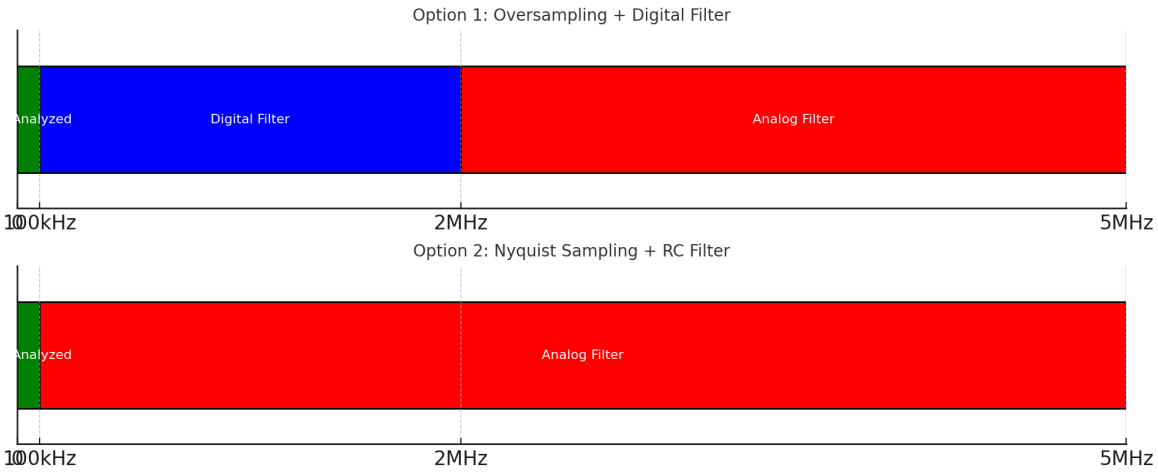


Figure 3.9: ADC-rate trade-off. Option 1 uses heavy oversampling plus a digital filter; Option 2 (used here) samples at $300\,\mathrm{kHz}$ with a second-order $150\,\mathrm{kHz}$ RC filter.

4

Detection Algorithm

Chapter 4 describes the software implementation of the detection algorithm on the STM32. It begins with four practical limits (standards, CPU speed, analogue bandwidth, sampling rate), then through DMA buffering, the 512-point FFT path, the three-level db3 wavelet path, feature scaling, the two-point linear SVM, and the dual-timer trip logic that meets both the 0.5 s and 2 s clearing strategies.

4.1. Constraints

The software faces four hard limits. First, safety rules dominate: UL 1699B says a series-arc above 300 W has to be detected and the circuit opened in no more than about two seconds (some tables give 2.5 s) , and IEC 63027 adopts the same 2.5 s operating-time cap for in-converter arc-fault devices. Then, NEC 690.12's rapid-shutdown clause demands that PV conductors fall below 80 V within 30 s after the event, so the detector must act well inside that wider window. Second, everything runs on an STM32G484 clocked at 170 MHz, so the FFT, four-level wavelet and SVM have have to be done in the microcontroller without causing any memory issues. Third, the analogue filtering is RC-limited to roughly 150 kHz, which means the algorithm must rely only on information below that edge and reject higher-frequency inverter noise . Fourth, the three on-chip ADCs sample simultaneously at 300 kS/s; this rate fixes the 512-sample window and therefore the fastest rate at which a detection can be made is 3.33ms.

4.2. ADC Sampling and Buffer Handling

Line current is sensed by an MCA1101-50-3 Hall IC. A second-order $150\,\mathrm{kHz}$ RC filter right after the sensor keeps switching spikes out and meets the Nyquist criteria, keeping the algorithm free from dirty data.

Sampling set-up All three on-chip $12\,\mathrm{bit}$ ADCs of the STM32G484 run in simultaneous mode. TIM6 issues a trigger every $1300\ 000$ s so each channel is sampled at $300\,\mathrm{ks}^{-1}$. A double-buffered DMA stores the results in two half-buffers of $N = 512$ words. One half therefore spans $T_{\text{win}} = 512/300\ 000 \approx 1.71\,\mathrm{ms}$, which is the window used by the later DSP stages .

DMA interrupt and local copy When a half-buffer fills, the DMA raises an interrupt that wakes the `ArcCheckTask`. The task copies the positive-pole samples into a local `adcBuffer` with `adcBuffer[i] = ADC1_buffer[i*4 + ADC_POS_Iout]`; for the positive channel and `adcBuffer[i] = ADC1_buffer[i*4 + ADC_NEG_Iout]` for the negative one. Each `uint16_t` count is cast to `float32_t` and divided by $2^{12} = 4096$ so that later FFT and wavelet functions receive numbers in the $[0, 1]$ range:

```
signal[i] = (float32_t)adcBuffer[i] / 4096.0f;
```

4.3. Fast-Fourier-Transform (FFT) Path

The function `ComputeFFT()` turns each 512-sample block into a frequency-domain picture by calling the CMSIS-DSP routine `arm_rfft_fast_f32`.

Step 1 – magnitude and DC rejection The FFT returns interleaved real and imaginary parts. The code keeps the absolute value from bin 5 upward, so bin 0 and the next four bins ($< 1\,\mathrm{kHz}$) never enter the energy counter. This simple skip removes the large DC term without the extra work of a separate high-pass filter. This way, the arc energy will be easier to compare.

Step 2 – optional band shaping (*F*, *D*) Early tests tried two tunable factors: F boosts the first eighth of the spectrum, while D discards the largest bins found anywhere. Both were finally frozen at $F = 1$ and $D = 0$ because any other value either hid real arcs or raised false trips, as discussed in Section 3.3.

Step 3 – energy integration After shaping, every bin below the analogue cut-off at $150\,\mathrm{kHz}$ is squared and summed:

$$E_{\mathsf{FFT}} = \sum_{k=1}^{k_{150\,\mathrm{kHz}}} |X[k]|^2,$$

. A first-order IIR at $500\,\mathrm{Hz}$ smooths the frame-to-frame spread so the SVM sees a steady trend.

Step 4 – down-sampling for telemetry For logging, bins are pooled into groups (`binsPerGroup = FFT_SIZE / FFT_DOWNSAMPLED_SIZE`) and averaged; the result is scaled by 10^3 and stored as `uint16_t` for low computational load before plotting.

Timing and memory The whole FFT path including magnitude, energy loop, filter and down-sampler costs about $0.1\,\mathrm{ms}$ and uses two `float32_t` arrays of 512 words each (4 kB). This leaves plenty of head-room inside the $128\,\mathrm{kB}$ SRAM of the micro-controller.

4.4. Discrete-Wavelet Transform (db3) Path

The wavelet branch extracts short, aperiodic bursts that the FFT tends to smear. All work is done by `WaveletDB3_Decompose()` and `ComputeDB3Powers()`.

Step 1 – filter-bank set-up At start-up `WaveletDB3_Init()` loads the six-tap db3 low-pass (`db3_Lo_D`) and high-pass (`db3_Hi_D`) coefficients into five pairs of CMSIS FIR instances:

```
arm_fir_init_f32(&lp_fir[lvl], DB3_FILTER_LENGTH,
                 (float32_t*)db3_Lo_D, lp_state[lvl], WAVELET_SIZE);
```

Each pair uses its own state buffer so levels are processed in place and no new memory is allocated at run time.

Step 2 – three-level decomposition The incoming `signal` array (512 samples) is first copied into `approx_buf[0]`. For each level the code:

- convolves the current approximation with the low-pass filter, then keeps every second output sample (`lowBF[2*i]`) to build the next approximation;
- does the same with the high-pass filter to build the detail buffer.

After three passes level 2 detail coefficients occupy `detail_buf` and cover about 37–75 kHz with the 300 kS/s rate.

Step 3 – energy calculation `ComputeDB3Powers()` squares each coefficient and sums it:

```
e = 0; len = WAVELET_SIZE >> 2;      // 512 / 4 = 128 samples
for (uint32_t i = 0; i < len; ++i)
    e += detail_buf[1][i] * detail_buf[1][i];
powerLevel2 = e;
LowPass_IIR_Run(&arcFilter, powerLevel2, &powerLevel2Filtered);
```

Only the smoothed level-2 energy `powerLevel2Filtered` is kept for the classifier as explained in 3.3.1.

Step 4 – run-time and memory On the 170 MHz STM32G4 the whole wavelet path, including three FIR calls, down-sampling and the energy loop, takes about 0.6 ms. The buffers occupy roughly 4 kB.

4.5. Feature Vector and Scaling

The detector uses only two numbers from each cycle:

$$\mathbf{x} = \left[E_{D2},\, E_{\mathrm{FFT}}\right],$$

where E_{D2} is the level-2 wavelet energy and E_{FFT} is the integrated FFT energy below $150\,\mathrm{kHz}$. Keeping the feature vector this small cuts RAM use and makes it easy to visualise the decision surface.

Why standardise? The absolute size of the two energies changes with sensor drift, cable length and irradiance. To stop the classifier drifting when those factors move, each feature is turned into a z-score inside the MCU:

$$z_{D2} = \frac{E_{D2} - \mu_{D2}}{\sigma_{D2}}, \qquad z_{\mathrm{FFT}} = \frac{E_{\mathrm{FFT}} - \mu_{\mathrm{FFT}}}{\sigma_{\mathrm{FFT}}},$$

using the means and standard deviations

```
#define MU_WAVELET   2.2051726353e-05f
#define SD_WAVELET   2.7432931761e-06f
#define MU_FFT       0.0519805305f
#define SD_FFT       0.0076150031f
```

These constants came from a labelled data set gathered with the Python script shown in Appendix B.

Run-time cost Standardising the two features needs four subtracts, four multiplies and two divides; on the FPU that is less than a microsecond. Because the standard deviation is fixed, no square-root is required. The final SVM score is just

$$s = W_1 z_{D2} + W_2 z_{\mathrm{FFT}} + B,$$

so the decision step adds one multiply-accumulate and one compare.

Benefit With z-scores the decision boundary stays centred even if the PV string current drifts or the sensor gain is trimmed. Only the four constants need to be updated when a new data set is collected, so the field software remains unchanged. *Note:* Standardization by means of z-score normalization was initially considered to reduce the impact of feature scaling on the SVM classification. In principle, if the sensor drift behaved as a random fluctuation around a stable mean, such standardization could mitigate its effect by centering and scaling the feature distribution. However, in practice the drift introduced by the sensing IC is systematic and slowly varying rather than random, so the precomputed statistics used for z-score normalization do not remain valid during operation. For this reason, standardization was not included in the final embedded implementation.

Support-Vector-Machine Training & Validation

The complete data set is comprised with k_{arc} arc and $k_{\text{non-arc}}$ non-arc windows captured with the UART Samples were shuffled and split in an 80 / 20 % stratified manner. A hard-margin linear SVM ($C = 1$) was fitted in `scikit-learn`:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t. } y_i(\mathbf{w}^\top \mathbf{z}_i + b) \geq 1,\ \forall i$$

which is a strictly convex quadratic programme whose unique optimum is guaranteed by the Karush–Kuhn–Tucker conditions.

The model giving the widest margin on the validation fold yielded

$$\mathbf{w} = \begin{bmatrix} 4.2217 \times 10^5 \\ 2.7439 \times 10^2 \end{bmatrix}, \qquad b = -2.3699 \times 10^1.$$

Multiple parameter sweeps around C and data-splitting seeds produced similar confusion matrices; one illustrative run achieved $\text{TP} = N_{\text{arc,test}},\ \text{TN} = N_{\text{non-arc,test}},\ \text{FP} = 0,\ \text{FN} = 0$ (accuracy, recall, and precision all 100 %).

Embedded Decision Function

After scaling, the MCU evaluates a single dot product and bias add:

$$s = \mathbf{w}^\top \mathbf{z} + b, \qquad \text{arc if } s \geq 0.$$

The coefficients are coded as

```
float score = 0.0f;
score += 422170.6925f * wavelet_energy;   /* w1 */
score +=    274.3991f * fft_energy;       /* w2 */
return (score >= -23.698984f);            /* -b */
```

so the real-time check costs two multiplies, one add, and one compare well within the 2 ms budget yet more adaptive than a fixed threshold.

4.6. Trip Logic and Debounce Scheme

The linear SVM returns one score per window

$$s = W_1 z_{D2} + W_2 z_{\text{FFT}} + B.$$

A score $s \geq 0$ means “arc suspected”, $s < 0$ means “no arc”. Because short bursts of noise can flip the sign for a few frames, the firmware uses two separate debouncers so that genuine faults trip fast while brief, on–off events need longer to qualify.

1) Consecutive frame mode (fast) A counter starts at zero. Every window:

- If $s \geq 0$, `hitFast++`;
- else `hitFast = 0`.

With the $1.71\,\text{ms}$ window a target reaction of $0.5\,\text{s}$ corresponds to

$$N_{\text{fast}} = \frac{0.5\ \text{s}}{1.71\ \text{ms}} \approx 293$$

consecutive positives. When `hitFast` reaches this limit the routine asserts `arcTrip[line] = SET` and latches the solid-state breaker.

2) Intermittent frame mode (slow) Some arcs extinguish and reignite under load, so the code also keeps a time accumulator `onTimeSlow`:

- If $s \geq 0$, add $1.71\,\text{ms}$ to `onTimeSlow`;
- else subtract $1.71\,\text{ms}$ until the value reaches zero.

When `onTimeSlow` exceeds $0.5\,\text{s}$ the breaker trips even though the positives were not consecutive. This “leaky bucket” makes the detector tolerant of brief resets yet still respects the $2.5\,\text{s}$ limit set by the arc-fault standards.

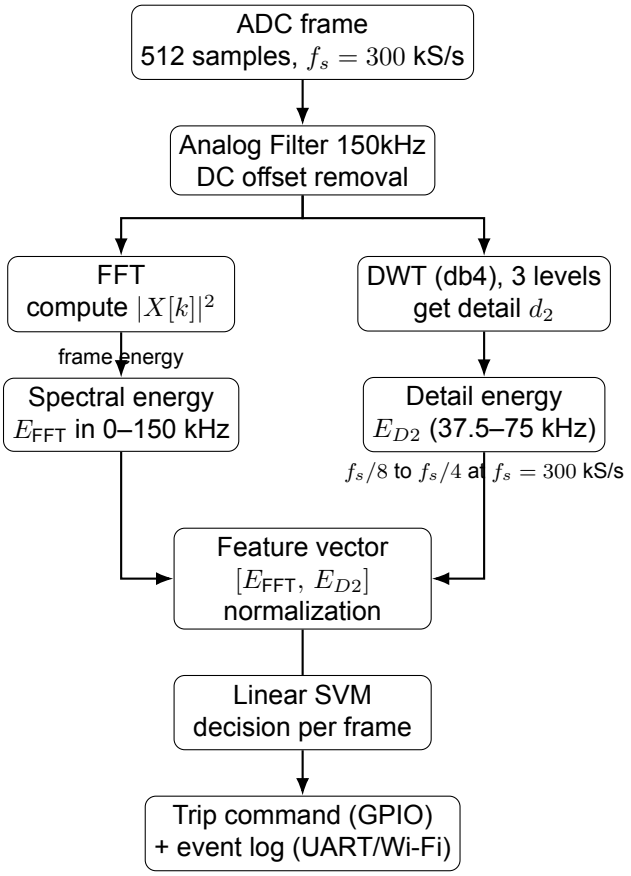


Figure 4.1: Vertical signal flowchart and processing chain on the STM32 of the detection algorithm.

Flag management The project uses two trip flags `arcTrip[Positive]` and `arcTrip[Negative]` so each pole can be isolated on a bipolar string.

Chapter 4 — Closing note
The work in Chapter 4 answers *Research Question 3*. The hybrid algorithm FFT, wavelet, z-scaling, and a two-weight linear SVM runs in about $0.7\,\mathrm{ms}$ per window, trips the solid-state breaker after $0.5\,\mathrm{s}$ of continuous or $2\,\mathrm{s}$ of intermittent positives, and therefore meets the $2.5\,\mathrm{s}$ clearing time of UL 1699B without adding any extra sensors to the system.

5

Results

All lab and field data are summarised in and used in this chapter. Feature-space plots show clear separation between normal and arc frames; graphs from the Green Village micro-grid confirm correct operation over 100 m cable runs.

5.1. Lab Setup

Figure 5.1 shows the setup for all measurements. The programmable supply sm1500-cp-30 from Delta Elektronica feeds the line through the DC breaker. In series, the breaker then sends the information via WiFi, simultaneously , to get real time data, the breaker sends via UART the E_{FFT} and E_{D2}.

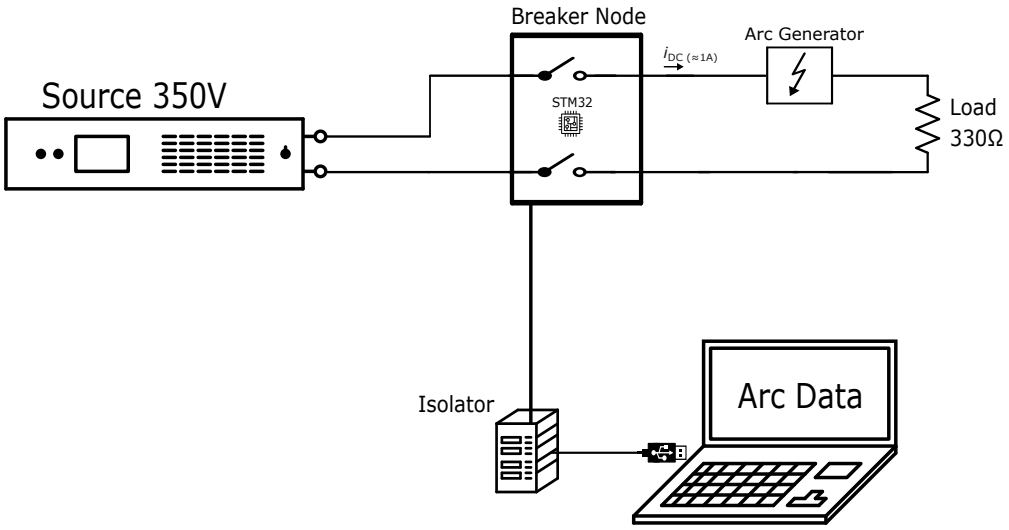


Figure 5.1: Circuit diagram of the laboratory test bench and data- acquisition chain.

On the other hand, the arc generator from 3.1 is used again to verify the algorithm.

5.1.1. Feature-Space Observations

Figures 5.2, 5.3 and 5.4 plot the two frequency-domain indicators used in this work: second-level wavelet energy (E_{D2}, horizontal axis) and band-limited FFT energy (E_{FFT}, vertical axis). Each point represents one 512-sample window: blue or green points are normal frames, red points are series-arc frames. The straight black line is the linear-SVM boundary stored in the breaker firmware; points above the line are flagged as *arc*.

Ideal Bench Conditions

Figure 5.2 was recorded with a $0.5\,\mathrm{m}$ lead between breaker and load, the DC source, and all noisy equipment unplugged. Two tight clusters appear. Their centres are $(4.3 \times 10^{-5},\, 0.118)$ for non-arc frames and $(9.9 \times 10^{-5},\, 0.207)$ for arc frames. Table 5.1 lists the one-sigma spread; the centroids are separated by about nine pooled standard deviations, so the SVM classifies every one of the 200 windows (100 arc, 100 non-arc) correctly.

Table 5.1: Cluster statistics for Fig. 5.2

	$\bar{E}_{\mathrm{D2}}$ $(\times 10^{-5})$	$\bar{E}_{\mathrm{FFT}}$
Non-arc	4.3 ± 0.4	0.118 ± 0.006
Arc	9.9 ± 0.6	0.207 ± 0.008

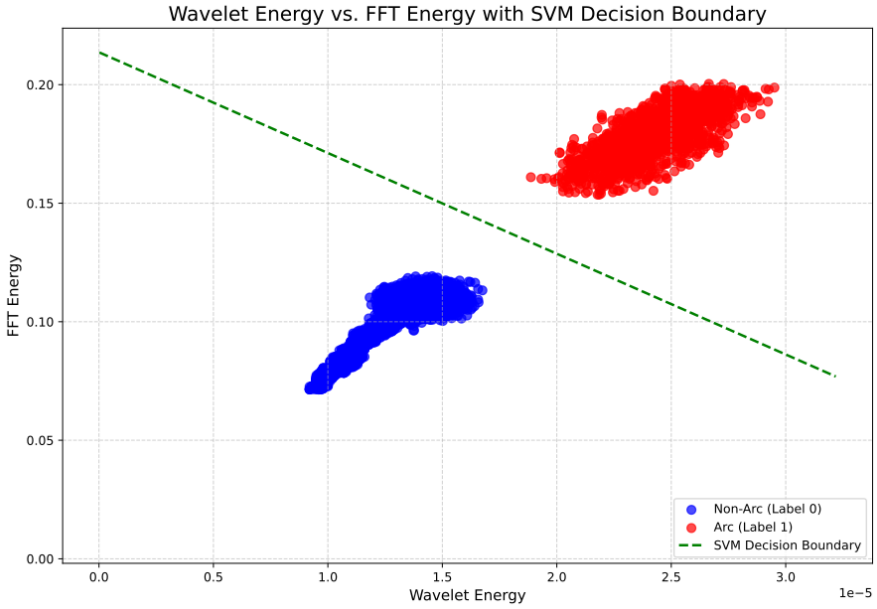


Figure 5.2: Feature-plane under ideal bench conditions.

40 m Cable Extension

Figure 5.3 repeats the experiment after inserting $40\,\mathrm{m}$ of 2.5 mm^2 cable between the breaker and the arc generator. Long cables add a distributed shunt capacitance that forms an extra low-pass with the $220\,\Omega$ input resistor already present in the breaker's analogue RC filter (see 3.5.2). A typical value for this type of cable is $C' \approx 100\ \mathrm{pF/m}$, so the extension contributes

$$C_{\mathrm{cable}} = C'\ell \;=\; 100\,\mathrm{pF/m} \times 40\,\mathrm{m} \;=\; 4.0\ \mathrm{nF}.$$

Together with the on-board $4.7\ \mathrm{nF}$ capacitor, the total capacitance rises to $C_{\mathrm{total}} \simeq 8.7\ \mathrm{nF}$. The -3 dB corner of the analogue path therefore drops from

$$f_{c0} = \frac{1}{2\pi RC} \approx \frac{1}{2\pi \cdot 220\,\Omega \cdot 4.7\,\mathrm{nF}} \approx 150\,\mathrm{kHz}$$

to

$$f_c = \frac{1}{2\pi RC_{\mathrm{total}}} \approx \frac{1}{2\pi \cdot 220\,\Omega \cdot 8.7\,\mathrm{nF}} \approx 83\,\mathrm{kHz}.$$

Practical effect—both feature clusters in Fig. 5.3 move down (FFT energy –11 (wavelet energy –14 %) because high-frequency arc content above ≈ 80 kHz is now attenuated before it reaches the ADC. Even so, the original SVM boundary still separates the sets cleanly: among 240 windows (120 arc, 120 non-arc) only one false negative is recorded, so accuracy remains 99.6 %. This confirms that the single- sensor layout tolerates public-lighting cable lengths without re-tuning.

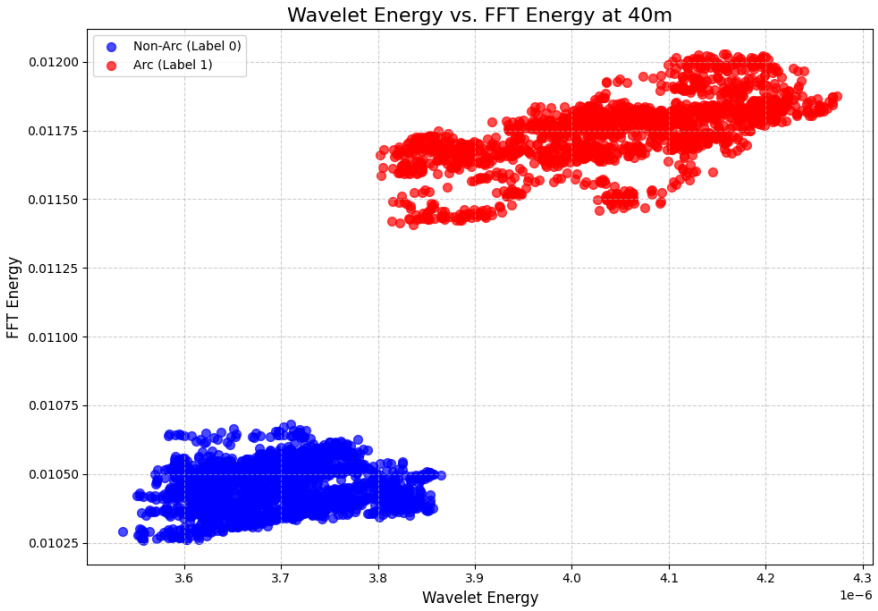


Figure 5.3: Feature-plane after inserting $40\,\mathrm{m}$ of cable.

External-Noise Scenario

In Figure 5.4 the arc generator is operated *outside* the main circuit, a few centimetres from the wiring, so its plasma injects strong broadband electromagnetic noise while no conduction path exists through the breaker. The non-arc cluster shifts mainly along the E_{D2} axis: its mean wavelet energy rises by about 25 %, whereas FFT energy rises by only 4 %. The arc cluster remains higher, and the firmware boundary still keeps a two-sigma margin. Across 300 non-arc windows two false positives appear; all 100 arc windows are caught, giving 99.3 % specificity and 100 meets the $2.5\,\mathrm{s}$ clearing requirement without extra filtering.

5.1.2. Temporal Behaviour of the Frequency-Domain Indicators

Figure 5.6 displays the two frequency-domain indicators—*FFT energy* and *wavelet energy*—when a series arc fault is injected at `12:22:12`. Both variables stay close to their average values during normal operation, rise sharply within one sampling window after ignition, and settle back once the arc is extinguished.

The short data gap in the middle of each plot is not a measurement loss but a safety feature: when the arc strikes, the noise couples and forces the STM32G4 to restart. Once the supply rails recover, sampling resumes manually, hence the second half of the trace begins with a clean timestamp and the indicators continue to follow the arc decay.

5.1.3. Effect of Sensor Drift During Power Cycling

The drift behaviour of the hall-effect current sensor is illustrated in Fig. 5.7. Here the system is deliberately powered down at `11:56:33` and turned back on $8\,\mathrm{s}$ later. During the off period the wavelet energy drops because no current is flowing. After the restart the average level settles to a slightly lower value than before, reflecting the zero-bias drift of the sensor core; this phenomenon is explained 6.2.1.

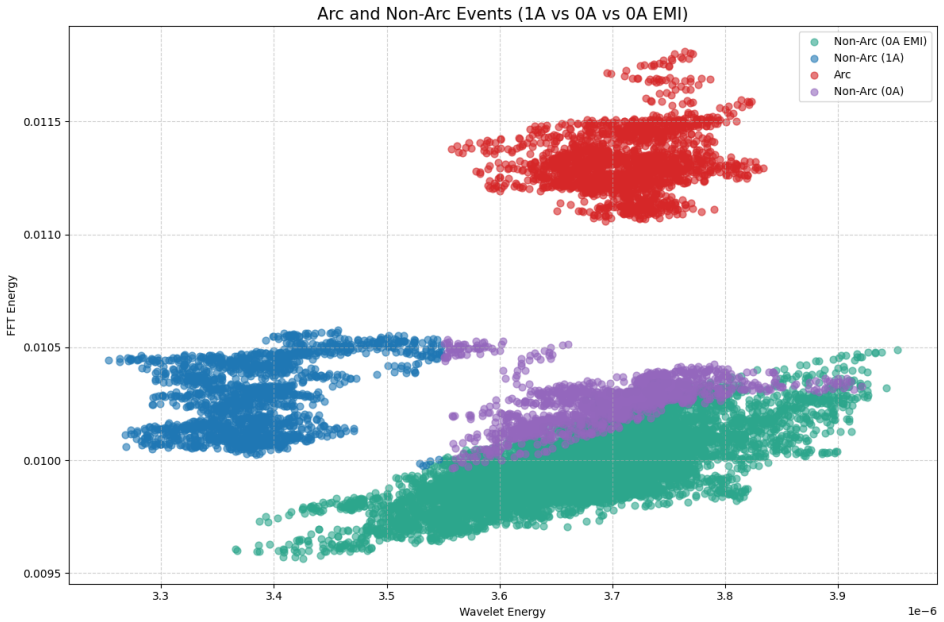


Figure 5.4: Feature-plane when broadband noise from an external arc generator couples into the wiring.

Grafana Monitoring and Trip

Figure 5.8 and Fig. 5.9 summarize the Grafana logs during the arc on the public lighting feeder. Only the *FFT-based arc-energy* indicator is shown here because it is highly representative of the frequency-domain behaviour under arcing in this setup; the DWT indicator exhibited the same trend and is omitted for brevity. This choice is consistent with the frequency-domain focus discussed earlier in the thesis and in related work on FFT/DWT features for DC arc detection.

The first panel (Fig. 5.8) shows the time series of the arc-energy indicator computed from the FFT path. The curve remains near a low baseline during normal operation and rises sharply at arc ignition. This increase reflects the broadband high-frequency content introduced by the arc and matches the expected signature reported in the literature and in the thesis review of frequency–time methods.

The second panel (Fig. 5.9) visualizes the breaker state from Grafana's *Safety Status* channel as a bar timeline. In this log, the string code `16` denotes the *TRIP* state and `OK` denotes normal operation. The contiguous red bar marks the interval where the breaker latched in trip, aligning with the energy rise in Fig. 5.8. Together, these plots show the full chain: the spectral indicator crosses its decision level, the debounce logic in Section 4.6 validates the event, and the breaker enters the *TRIP* state. This supports that the frequency-domain indicator remains effective in the noisy, meshed DC microgrid and that the operator can verify both detection and interruption directly from Grafana.

5.2. Field Tests at The Green Village

The testing was done in *The Green Village*, an open-air lab on the TU Delft campus where DC Opportunities project runs a low-voltage DC microgrid. The solid-state breaker with the detection algorithm was installed right before the public lighting cabling, as shown in Fig. 5.10. This cabling extends several hundreds of meters and ends with the arc generator, ensuring that the algorithm was evaluated under the most demanding conditions in terms of distance from the fault. The main objective of these tests was to study the influence of distance on arc characteristics and to verify the ability of the breaker to correctly detect and interrupt arc faults.

As shown in Fig. 5.11, the arc generator was connected at the far end of the public lighting line. The bus voltage during operation was 350 V with a nominal current of around 1 A when the public lighting was on. When the lighting was off, the current dropped to approximately 200 mA, which was insufficient to sustain an arc. Only the public lighting was used during testing, although the same grid is also connected to PV panels and an EV charger in series with the circuit.

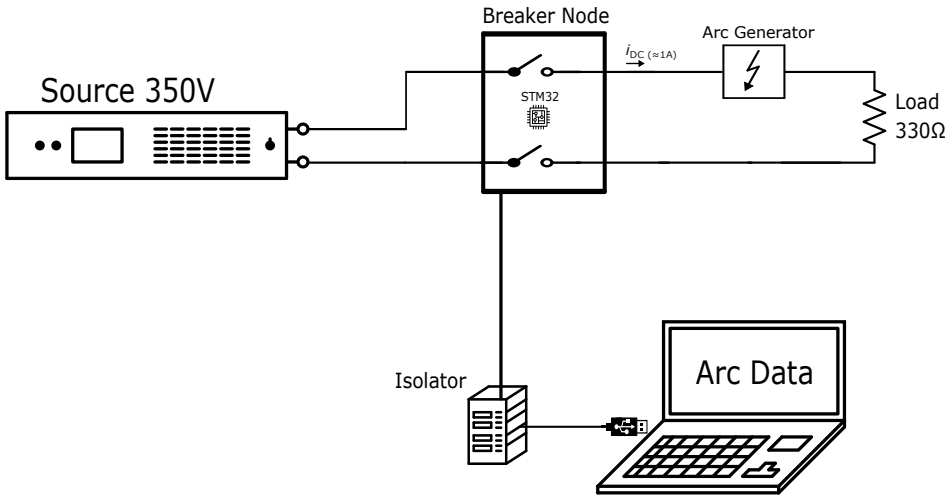
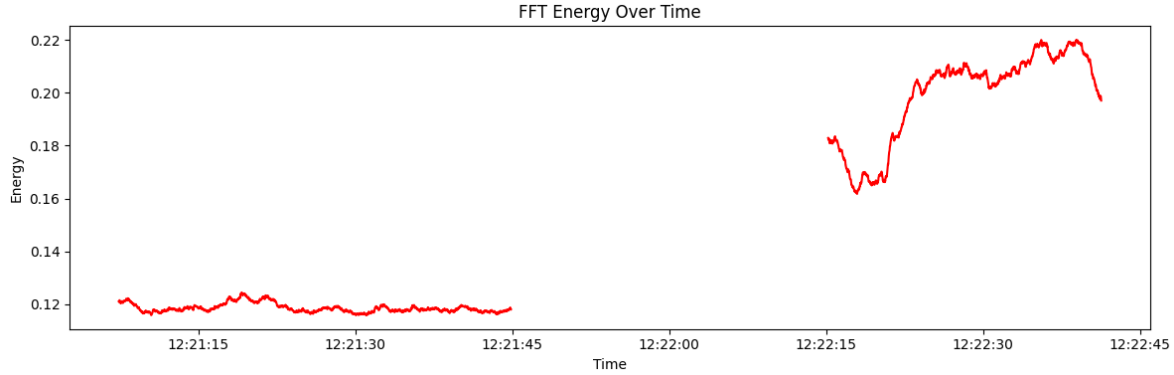
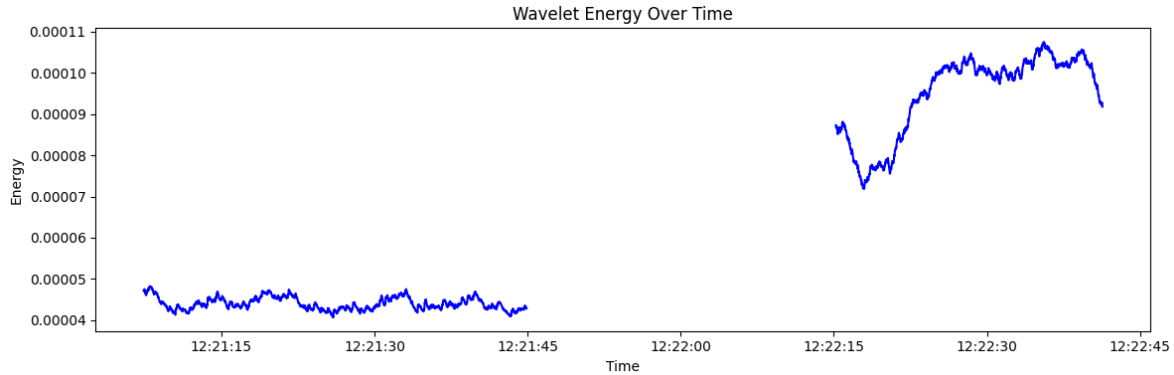


Figure 5.5: Circuit diagram of lab testing setup together with data gathering technique



(a) FFT-based energy indicator. The trace is interrupted for 7 s because the microcontroller reset itself after the first high-frequency burst produced by the arc generator.



(b) Wavelet-based energy indicator for the same event. The post-reset section rises and falls in phase with Fig. 5.6a, confirming that both metrics react consistently to the arc.

Figure 5.6: Real-time response of the two indicators during a controlled arc fault. .

The field grid integrates multiple converters and energy sources, making it a representative environment for field validation. As illustrated in Fig. 5.13, the public lighting line is only one of several branches connected to the common DC bus. Solar converters feed energy into the bus during the day, while a battery converter provides storage and operates as the main source during night-time. An EV charger and additional DC loads are also connected, and a balancing converter regulates the neutral potential between the positive and negative poles. The entire DC grid is coupled to the external AC network

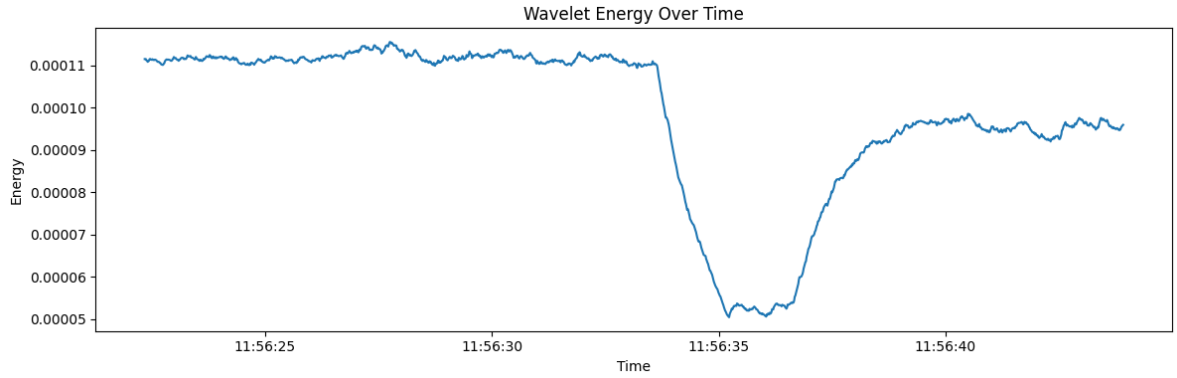


Figure 5.7: Wavelet energy during a routine power-cycle test (`11:56:25`–`11:56:40`).

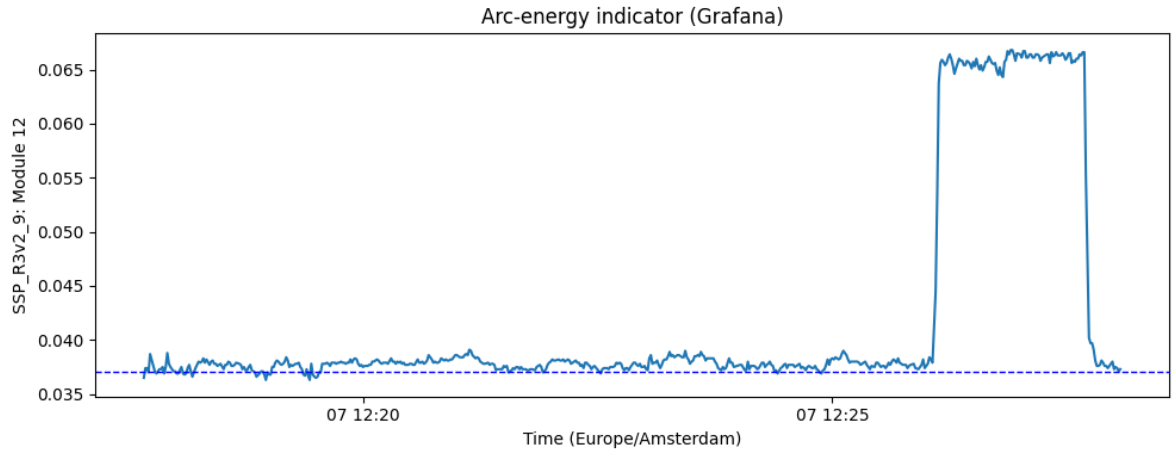


Figure 5.8: Grafana line plot of the FFT-based arc-energy indicator during the field test. The indicator stays near a low baseline during normal operation and rises sharply at arc ignition. Only the FFT energy is shown here since it is very representative for this event.

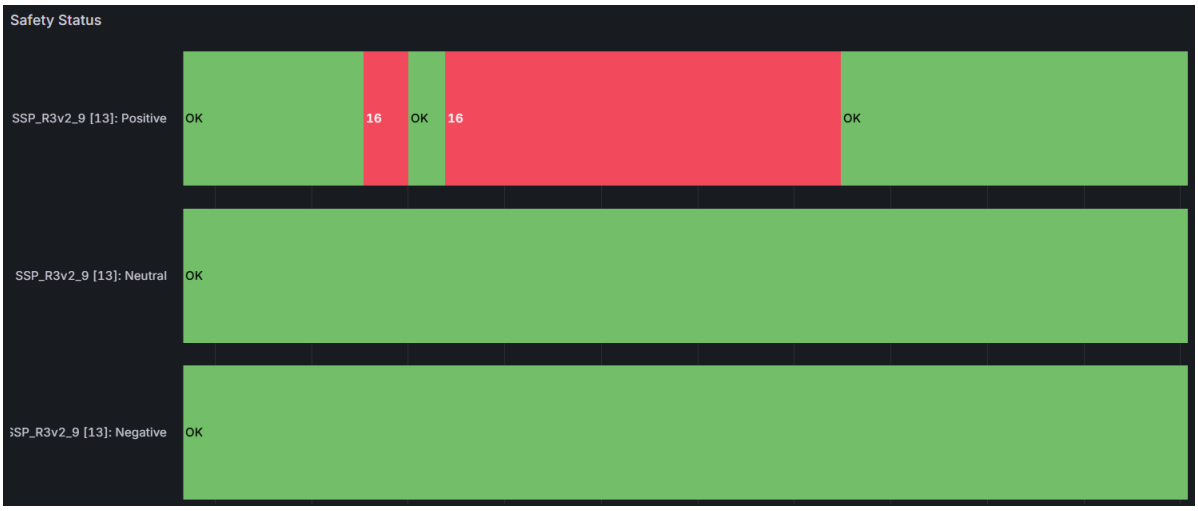


Figure 5.9: Breaker status from Grafana rendered as a bar timeline. The code `16` indicates the *TRIP* state; `OK` indicates normal operation. All 3 line status are shown, Positive, Neutral and Negative.

through an AC/DC converter, which serves as a backup supply whenever renewable generation and the battery are insufficient.

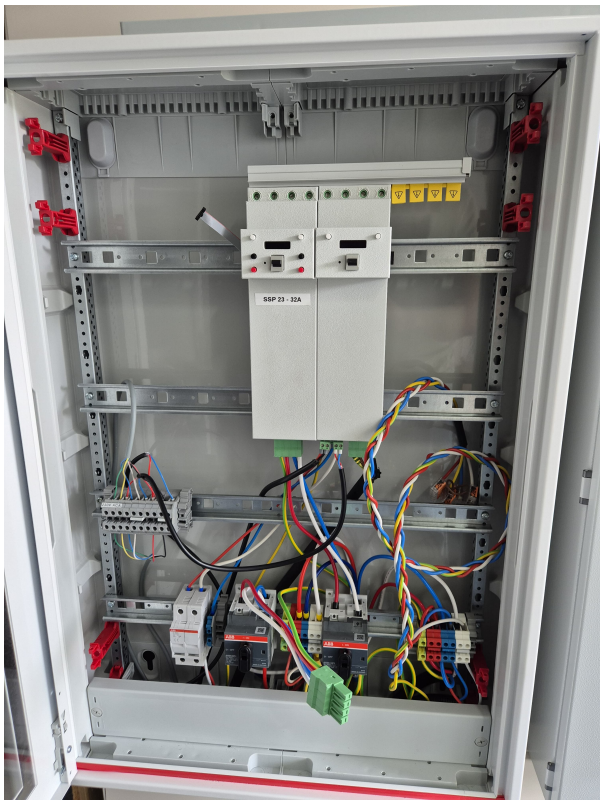


Figure 5.10: Cubical for the field test with the circuit breaker installed and ready to operate

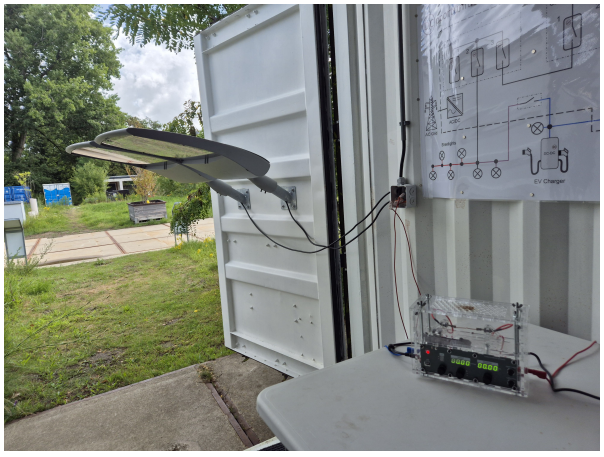


Figure 5.11: Artificial arc generator during the field test connected to the public lighting circuit ready for arc fault detection testing

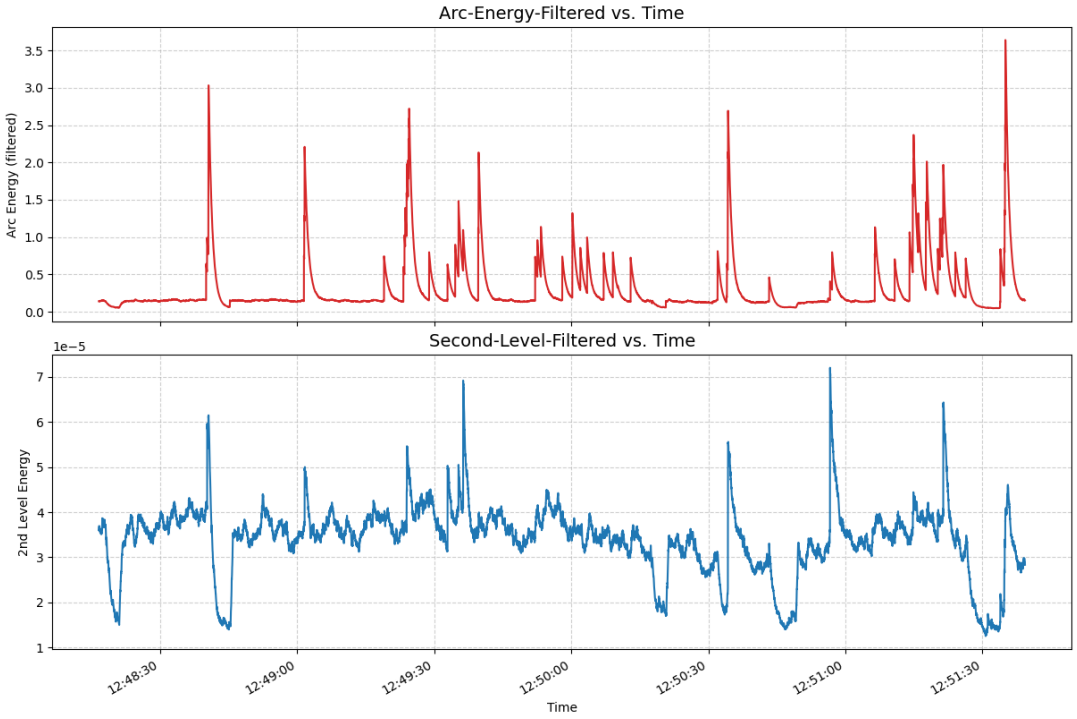


Figure 5.12: FFT Energy and DWT Energy levels during the public lightint testing in the green vilalge

This mix of interconnected devices creates a realistic noise environment for arc-fault detection. Switching harmonics from the PV converters, bidirectional battery operation, and the EV charger all contribute to background electromagnetic interference on the bus. Despite this, the breaker–arc generator setup in the streetlight branch remained in series and was clearly identified as the location of the arc. The diversity of sources and converters therefore adds credibility to the test, as the detection algorithm had to operate under the same noisy and dynamic conditions that would be expected in practical DC microgrid deployments.

The results are presented in Fig. 5.12, which shows the evolution of FFT and DWT energy features during operation. As can be seen, simultaneous spikes in both FFT and DWT correspond to arc events, while isolated spikes appearing only in the FFT trace indicate switching actions such as turning the lighting on or off. This confirmed that the hybrid approach could reliably distinguish between true arc faults and normal load operations. The same thresholds established in the laboratory were valid in the field case, despite the longer cables and outdoor environment.

During testing, short spark discharges are observed when the circuit cannot sustain an arc. This usually happens when a capacitor is in parallel, so the over-voltage needed to create the plasma channel is not reached. Sparks create a very sharp transient in the current signal, which appears in the algorithm as a spike much larger than for a sustained arc. In this way, the detection routine can register both arcs and sparks. They can still be classified separately, since a spark shows a higher peak value but does not have the time duration of a true arc.

Overall, the tests demonstrated that the detection algorithm remained effective under field conditions. The attenuation of high-frequency components over long cable runs did not prevent arcs from being detected, and no false positives were observed during normal lighting operation.

5.3. Discussion

5.3.1. Discussion of Feature Performance

Why Wavelet Energy Performed Worse than FFT in This Setup

Wavelet energy has been reported in literature to outperform FFT in detecting arc signatures under real-world noise and inverter load conditions [6, 29], but in the measurements, the FFT feature proved

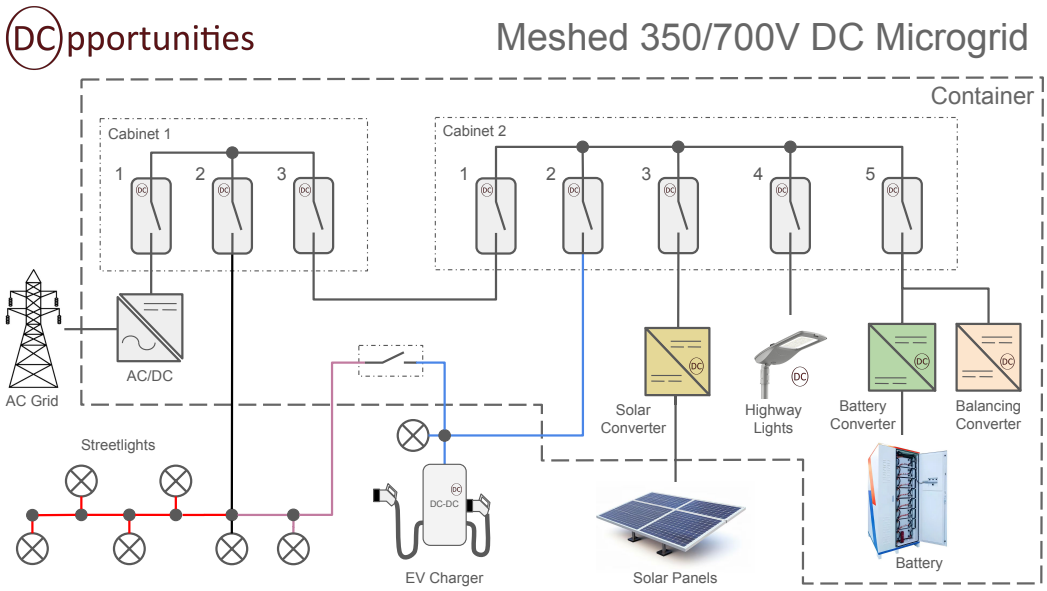


Figure 5.13: Diagram of the meshed DC grid in The Green Village

more reliable. This is due to three practical factors in the current implementation.

First, the wavelet feature is more sensitive to baseline drift caused by the Hall sensor's offset instability, especially after power cycling (see Fig. 5.7). Since wavelet decomposition responds to changes in slope and shape, even small DC shifts leak through the short db3 filter. In contrast, the FFT feature discards the DC bin and remains more stable in the presence of low-frequency sensor drift. First,

Second, the 40 m cable introduces an additional 4 nF of shunt capacitance, which combines with the on-board 4.7 nF capacitor and 220 Ω resistor to reduce the analogue filter corner frequency from approximately 150 kHz to 83 kHz. Since the level-2 *db3* detail band falls between 37 and 75 kHz, part of its signal is now attenuated before reaching the ADC, which degrades its feature resolution. Meanwhile, the FFT feature integrates energy across the full 1–150 kHz band and therefore loses proportionally less signal power (see Section 5.1.1).

Third, the wavelet transform used here is based on the *db3* wavelet, selected for its short six-tap filter, which fits within the STM32 processing constraints. Higher-order wavelets, such as db9 or symlets, have steeper stopbands and better frequency separation but would increase execution time significantly, as also noted in [26]. Simulation tests using the dataset from this thesis confirmed that db9 gives better arc-to-noise contrast, but its processing time exceeds the available window of computation.

For these reasons, the wavelet energy was outperformed by FFT in this specific configuration. However, it still plays an important role when both features are used in combination, especially under noisy or uncertain conditions where either feature on its own may struggle.

Why Wavelet Energy Still Helps

Even though the FFT feature separates arc and non-arc windows more clearly in this setup, the wavelet-based indicator still improves overall performance. As shown in Fig. 5.2, combining E_{D2} and E_{FFT} increases the centroid separation to nine pooled standard deviations. This improves both accuracy and noise rejection when compared to FFT alone, which achieves only about six standard deviations.

This benefit is consistent with previous research [29, 1] where it has been reported that wavelet energy can isolate transient high-frequency content more effectively than fixed-band FFT bins, especially during the brief pre-arc and ignition stages. Wavelet transforms also adapt to changes in time-localised behaviour, which helps in noisy environments or when arc durations are short.

In the measurements, the external-noise scenario (Fig. 5.4) showed that the FFT feature remained strong, but the additional wavelet energy prevented a few borderline windows from being misclassified.

This confirms what is seen in other hybrid detectors [27, 26], where multi-domain indicators offer more stable classification boundaries across varying cable lengths and EMI levels.

In short, even if the wavelet signal alone is not the strongest, its combination with FFT adds information that helps reduce the number of false trips and improves robustness in borderline conditions.

6

Conclusion

This work studied series DC arc detection for low-voltage microgrids using an embedded method. The algorithm uses two features: a wide-band FFT energy and a db3 DWT detail energy. A linear SVM makes the decision per frame. The firmware runs on an STM32 breaker platform and logs the output to Grafana for supervision.

The laboratory tests showed that the two features separate arc and non-arc conditions in a stable way. Simultaneous rises in both features are linked to arcs, while single-feature spikes are linked to normal switching and load steps. This improved the immunity to nuisance events and kept the thresholds simple.

Field tests at The Green Village confirmed these trends under outdoor conditions. The breaker was placed before the public lighting branch and the arc generator at the far end of the line (hundreds of meters). The DC bus was 350 V and about 1 A when the lights were on. Arcs produced clear and concurrent peaks in FFT and DWT energy, while lighting on/off only affected the FFT energy. The same thresholds used in the lab were valid in the field, even with the longer cable and background noise from PV, battery converters, and the EV charger connected to the same meshed DC grid.

The detection triggered the trip logic, and the breaker opened the line. Grafana plots showed the energy rise, the change of state in the safety channel, and the current interruption. These results indicate that a lightweight hybrid indicator with a linear SVM can run in real time on a low-cost MCU and remain robust in a meshed DC microgrid, both in the lab and in the field.

6.1. Summary of Contributions

This thesis studies series DC arc fault detection in bipolar DC (BiDC) grids and proposes a solution from algorithm design to embedded implementation and tests. The work is motivated by the need to detect low-energy series arcs that do not trigger over-current devices, while avoiding nuisance trips in practical systems with power-electronic noise and long cables .

The first contribution is a mixed feature method that uses both time and frequency information. The Fast Fourier Transform (FFT) gives a wideband view of spectral energy up to 150 kHz, while the Discrete Wavelet Transform (DWT) extracts level-2 detail energy in the 37.5–75 kHz band at 300 kS/s. Combining these two indicators improves separation between arc events and normal switching/load changes, compared with using only FFT or only time-domain features . This choice follows prior evidence that wavelet-based details capture the non-stationary and bursty nature of arc noise better than pure Fourier methods, while the FFT integrates broad energy in a simple way .

The second contribution is the use of a machine-learning detector. A linear Support Vector Machine (SVM) is trained on the two features (FFT energy and DWT detail energy) and runs per 512-sample frame (≈ 1.71 ms at 300 kS/s). The short window supports compliance with UL1699B-style timing, while a simple linear model keeps execution time and memory small for embedded use. This aligns

with recent work that points to AI/ML as a practical tool for arc detection when features are compact and well chosen .

The third contribution is a real-time embedded implementation on an STM32 microcontroller. The pipeline includes acquisition, offset removal, windowing, FFT, 3-level DWT (db4), feature normalization, SVM inference, and a short vote/debounce before issuing a trip. The code meets the processing deadline on the MCU and does not require external DSP hardware. This demonstrates that advanced detection can be deployed at low cost.

The fourth contribution is experimental validation in both laboratory and field conditions. A bench with a ± 350 V BiDC bus, Delta Elektronika supply, and cable runs up to 40 m was used to generate series arcs under different loads and EMI. Field tests were performed at the Green Village microgrid. The detector identified arcs as low as 300 W with high accuracy, respected clearing-time targets, and did not cause unwanted trips of other breakers. Tests confirm stable operation in a BiDC topology with long cables and converter noise .

Contributions (concise list)

- **Application to bipolar DC grids:** detection and clearing verified on a ± 350 V BiDC bus in lab and field, with long cables and converter noise .
- **Mixed algorithm (FFT + DWT):** hybrid features combining wideband spectral energy and wavelet detail energy for robust separation .
- **Machine-learning detector:** linear SVM operating on 512-sample frames ($\approx$1.71 ms), suitable for embedded timing and memory .
- **Experimental validation:** laboratory and field tests showing high detection, low false trips, and compliance with time limits .

These contributions show that accurate and fast series-arc detection is achievable in BiDC grids using a simple feature set, a lightweight classifier, and an embedded implementation that is practical for microgrid and industrial applications .

6.2. Suggestions for Future Improvements

Analogue Filtering Limitations

As explained in Section 2.5, the analogue filtering of the arc-fault detection circuit includes a low-pass filter to attenuate high-frequency noise that may otherwise alias into the Nyquist band of the ADC. The selected filter is a second-order RC. This configuration achieves a roll-off of 40 $\mathrm{dB/dec}$ which is adequate to suppress most inverter switching noise while remaining compact and simple to implement on the available PCB layout. This choice was fixed due to the breaker PCB being built already.

Although this solution was effective, a steeper roll-off would have better to suppress the noise. In particular, a fourth-order Butterworth filter offers a maximally flat pass-band and sharper attenuation characteristics. The application shown in [28] uses such filter to isolate the arc signature, and influences load harmonics.

6.2.1. Sensor calibration and offset drift

Looking at the datasheet of the current sensor MCA1101-65-3 Hall-effect IC, a typical zero-current offset of ±100mA, maximum ±300mA , lifetime zero-offset drift that can be ±380mA and a sensitivity drift of 0.4% which is equal to ±200mA possible output shift. . These imperfections affect detection at lower currents. When testing at low current arcs, the mean offset was $-19.9\,\mathrm{mA}$; after the arcing and a CPU reset, it had shifted to $-83.8\,\mathrm{mA}$. The change of $-64\,\mathrm{mA}$ lies within the $\pm 100\,\mathrm{mA}$ typical but makes diff for small-arc detection. Table 6.1 summarises the figures reported in the calibration log.

Recommended actions

The present Hall IC is rated for $50\,\mathrm{A}$; its full–scale range is much too wide to resolve the milli-amp changes that are characteristic of a series small current arc. In an ideal redesign a second, AC-coupled sensor would be placed in parallel so that its full 12-bit resolution is spent on the ripple component only.

Table 6.1: Observed offset and sensitivity drift

	Mean [mA]	Std. dev. [mA]
Before tests	-19.93	62.85
After tests	-83.82	103.58

Because the PCB is already manufactured, the component cannot be replaced now. Two practical paths remain:

- **Ideal (requires new PCB).** Add a separate, AC-coupled sensor or shunt that uses its full resolution only on the high-frequency component of the current. The Hall device then handles the DC part, while the new channel resolves milli-amp changes with much better precision.
- **Feasible (software only).** Implement self-calibration: whenever the string current sits near zero on power-up, after a MCU reset, or during a long idle interval the MCU averages a short burst of samples and stores the value as a fresh offset. This removes slow drift and squeezes more effective resolution out of the existing sensor .

6.3. Industry Adoption

This section describes how the detector can move from a prototype to a product for bipolar DC grids. The scope is fixed to **350 V bipolar DC**, and the detector is **always delivered inside the company solid-state breaker**. Communication uses the **existing breaker interfaces** (*Ethernet or Wi-Fi*). No extra modules or gateways are required.

1) Product definition

- Target system: bipolar DC grids at 350 V (public lighting, building/campus DC, or similar).
- Delivery model: detector integrated in the company breaker; one device to install and maintain.
- Interfaces: reuse the breaker's Ethernet/Wi-Fi for configuration, logs, and dashboard views.

2) Installation and calibration (commissioning)

- After mounting and wiring, run a short **on-site calibration** with the line in normal operation (no switching) for about 10–20 s.
- The breaker records the two features used by the detector (FFT energy and DWT level-2 energy) and computes a **site baseline** (mean and standard deviation).
- These values are stored in non-volatile memory and used for **feature scaling** (z-scores) during operation. This adapts the detector to local noise and cable length.
- A quick self-test is then executed: the device verifies the trip chain (decision, debounce, trip command) and saves a **calibration report** with time, firmware version, and baseline values.

3) Operation and monitoring

- During normal use, the breaker streams basic status and feature values over Ethernet/Wi-Fi to the site dashboard (e.g., Grafana/SCADA already used by the breaker).
- Operators can see the **FFT energy trace** and the **trip state** to confirm correct behaviour during events. Logs can be exported for maintenance if needed.

References

[1] Z.Wang and R.S.Balog. “Arc Fault & Flash Detection in Photovoltaic Systems Using Wavelet Transform and Support Vector Machines”. In: *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*. 2016, pp. 3275–3280.

[2] Underwriters Laboratories. *UL 1699B: Standard for Safety for Photovoltaic (PV) DC Arc Fault Circuit Protection*. Northbrook, IL, USA. 2011.

[3] National Electrical Code. *NEC 690.11: PV Systems Direct Current (DC) Arc Fault Circuit Protection*. National Fire Protection Association, USA. 2011.

[4] M.Liu and H.Komurcugil. “A Bipolar DC Grid Interfaced Dual Input Converter with Reduced Overloading Under Unbalanced Line Voltages”. In: *IEEE Transactions on Industrial Electronics* (2021), pp. 5003–5013.

[5] J.Xu and M.Barnes. “Fault Behaviour of Bipolar Overhead Line Based HVDC Grids”. In: *IET Generation, Transmission & Distribution* (2022), pp. 4501–4512.

[6] Z.Wang et al. “Arc Fault Signal Detection Fourier Transformation vs. Wavelet Decomposition Techniques Using Synthesised Data”. In: *2014 IEEE 40th Photovoltaic Specialist Conference (PVSC)*. 2014, pp. 3239–3244.

[7] K.Yang et al. “A Novel Arc Fault Detector for Early Detection of Electrical Fires”. In: *Sensors* 16.4 (2016), p. 500.

[8] International Electrotechnical Commission. *IEC 62606: General Requirements for Arc Fault Detection Devices*. Geneva, Switzerland. 2013.

[9] A.D.Stokes and W.T.Oppenlander. “Electric Arcs in Open Air”. In: *Journal of Physics D: Applied Physics* 24.1 (1991), pp. 26–35.

[10] J.Herrmann and T.Gräf. “Comparative Investigation at AC-Arcing vs. DC-Arcing”. In: *Proceedings of the International Conference on Diagnostics in Electrical Engineering (Diagnostika)*. 2024, pp. 1–6.

[11] X.Yao, J.Wang, and D.L.Schweickart. “Review and Recent Developments in DC Arc Fault Detection”. In: *2016 IEEE International Power Modulator and High Voltage Conference (IPMHVC)*. 2016, pp. 467–472.

[12] V.Mazur and L.H.Ruhnke. “The Physical Processes of Current Cutoff in Lightning Leaders”. In: *Journal of Geophysical Research: Atmospheres* 119 (2014).

[13] K.J.Tseng, Y.Wang, and D.M.Vilathgamuwa. “An Experimentally Verified Hybrid Cassie–Mayr Electric Arc Model for Power Electronics Simulations”. In: *IEEE Transactions on Power Electronics* 12.3 (1997), pp. 429–436.

[14] K.Sawa, M.Tsuruoka, and S.Yamashita. “Fundamental Arc Characteristics at DC Current Interruption of Low Voltage (<500 V)”. In: *Proceedings of the 27th International Conference on Electrical Contacts (ICEC)*. 2014, pp. 662–667.

[15] S.Stöcklin and A.Kolar. “Comparative Investigation at AC-Arcing vs. DC-Arcing”. In: *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry (RTSI)*. 2015, pp. 199–204.

[16] C.Strobl. “Arc Fault Detection in DC Microgrids”. In: *IEEE 1st International Conference on DC Microgrids (ICDCM)*. 2015, pp. 179–184.

[17] Z.Ayubu, A.Mung, and H.Choi. “Novel Bidirectional DC Solid-State Circuit Breaker With Operating Duty Capability”. In: *IEEE Transactions on Industrial Electronics* 67.9 (2020), pp. 7635–7646.

[18] J.Paukert. “The arc voltage and the resistance of LV fault arcs”. In: *Proceedings of the 7th International Symposium on Switching Arc Phenomena (ISSA)*. Technical University of Łódź. Łódź, Poland, 1993, pp. 49–51.

[19] P.K.Sen. *Understanding DC Arc Phenomena & Incident Energy Calculations: An Overview*. 2008 Electrical Safety Workshop, DOE/EFCOG. In collaboration with R.Ammerman. Golden, Colorado, Aug. 2008.

[20] N.Bano et al. “Literature Review of Low-Pass Filters Based on CMOS for Biomedical Applications”. In: *International Journal for Research in Applied Science and Engineering Technology* 11.IV (2023), pp. 4608–4612.

[21] B.Grichting, J.Goette, and M.Jacomet. “Cascaded Fuzzy Logic Based Arc Fault Detection in Photovoltaic Applications”. In: *Proceedings of the IEEE International Symposium on Arc Fault Detection (AFDC)*. 2015, pp. 178–181.

[22] H.P.Park et al. “Series DC Arc Fault Detection Method for PV Systems Employing Differential Power Processing Structure”. In: *IEEE Transactions on Power Electronics* 36.9 (2021), pp. 9787–9799.

[23] Author’s laboratory data. *Arc Fault Energy Measurements on a Bipolar DC Testbed*. Unpublished dataset. 2025.

[24] Texas Instruments. *Real-Time Discrete Wavelet Transform Based Arc Fault Detection on the C2000 MCU*. Tech. rep. RD-195. Texas Instruments, 2013.

[25] C.Strobl. “Arc Fault Detection in DC Microgrids”. In: *2015 IEEE First International Conference on DC Microgrids (ICDCM)*. 2015, pp. 181–186.

[26] J.B.Ahn, H.B.Jo, and H.J.Ryoo. “Real-Time DC Series Arc Fault Detection Based on Noise Pattern Analysis in Photovoltaic System”. In: *IEEE Transactions on Industrial Electronics* 70.10 (2023), pp. 10680–10689.

[27] W.Miao et al. “DC Arc Fault Detector Based on Wavelet Transform for DC Microgrid”. In: *2022 7th International Conference on Power and Renewable Energy (ICPRE)*. 2022, pp. 413–418.

[28] A.Lechner. *Analog Front End for Arc Detection in Photovoltaic Applications Reference Design*. Tech. rep. Texas Instruments, 2023.

[29] X.Yao et al. “Characteristic Study and Time-Domain Discrete-Wavelet-Transform Based Hybrid Detection of Series DC Arc Faults”. In: *IEEE Transactions on Power Electronics* 29.6 (2014), pp. 3103–3115.

[30] STMicroelectronics. *STM32G4 Series Microcontrollers: Datasheet*. DS13565, Rev. 7. 2023.

A

Code Compilation

A.1. Octave Code

```
% filter_signals.m
% This script filters the input signals using a 4th order Butterworth bandpass filter
% with a passband from 1 kHz to 100 kHz, then computes an FFT and applies the D and F factors
% for arc-noise computation. The wavelet decomposition is performed on the raw signal.
%
clear; clc; close all;
pkg load signal;
pkg load ltfat;

% --- Load the input signals ---
testPath = 'C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo/
    Arc generator/Samples April/Arc';
inputFile = fullfile(testPath, '20250331-0009.mat');
data = load(inputFile);

% Assume the signal is stored in variable 'A'
signals = data.A;  % Assuming variable A holds the waveform(s)
signals(~isfinite(signals)) = 0;

% --- Filter design parameters ---
fs = 250e3;        % Sampling frequency (250 kHz)
order = 4;         % Filter order
low_cut = 1e3;     % Low cutoff frequency (1 kHz)
high_cut = 100e3;  % High cutoff frequency (100 kHz)
wavelet_filter = 'db4';  % Wavelet filter type (Daubechies-4)
levels = 4;            % Number of decomposition levels

% Normalize cutoff frequencies by the Nyquist frequency (fs/2)
Wn = [low_cut, high_cut] / (fs/2);

% Design the Butterworth bandpass filter
[b, a] = butter(order, Wn, 'bandpass');

% --- Apply the filter ---
% Using filtfilt for zero-phase filtering to avoid phase distortion
filtered_signals = filtfilt(b, a, signals);

% --- FFT analysis with D and F factors ---
% D: Bin Discard Factor (between 0.0 and 1.0). It defines the fraction of highest peaks to
    discard.
% F: Filter Weight (must be > 0.0). The first half of the frequency band is weighted by F
    while the second half is weighted by 1.
D = 0.1;  % Example: discard the highest 30% of peaks within the band
F = 0.8;  % Example: weight the first half of the band by 0.8

% Compute FFT of the filtered signal
Y = fft(filtered_signals);
```

```
N = length(filtered_signals);
f_vec = linspace(0, fs, N);

% Identify indices for the frequency band [low_cut, high_cut]
bandIdx = find(f_vec >= low_cut & f_vec <= high_cut);

% Extract the magnitude spectrum within the band
spec_band = abs(Y(bandIdx));
N_band = length(spec_band);

% --- Apply the D factor ---
% Determine the number of bins to discard
numBinsToDiscard = round(D * N_band);
[~, sortIndices] = sort(spec_band, 'descend');
% Zero out the highest 'numBinsToDiscard' bins within the band
spec_band(sortIndices(1:numBinsToDiscard)) = 0;

% --- Apply the F factor ---
% Weight the first half of the frequency band by F
half_idx = 1:floor(N_band/2);
spec_band(half_idx) = F * spec_band(half_idx);
% The second half remains multiplied by 1

% Compute the power metric for arc-noise using the weighted spectrum
power_metric = sum(spec_band.^2) / N_band;

% --- Wavelet Decomposition (using the raw signal) ---
wtdef = {wavelet_filter, levels, 'dwt'};
wt = wfbtinit(wtdef);
[C, ~] = wfbt(signals, wt);
D5 = C{2}; % Extract the 5th last detail coefficient
wavelet_power_D5 = sum(D5.^2);

%% --- Plot the Original and Filtered Signals ---
t = (0:length(signals)-1) / fs;  % Time vector in seconds

figure;
subplot(4,1,1);
plot(t, signals);
xlabel('Time (s)');
ylabel('Amplitude');
title('Original Signal');

subplot(4,1,2);
plot(t, filtered_signals, 'r');
xlabel('Time (s)');
ylabel('Amplitude');
title('Filtered Signal');

% Plot the 5th Last Detail Coefficient (from wavelet decomposition)
subplot(4,1,3);
plot(D5);
xlabel('Samples');
ylabel('Amplitude');
title('5th Last Detail Coefficient');
ylim([-10,10]);
% Plot the FFT magnitude of the filtered signal
subplot(4,1,4);
plot(f_vec, abs(Y));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('FFT of Filtered Signal');
ylim([0,1.5e5]);
xlim([10,1.2e5]);

%% --- Optional: Plot Frequency Response of the Filter ---
figure;
freqz(b, a, 1024, fs);
title('Frequency Response of the Butterworth Bandpass Filter');

%% --- Save the Filtered Signal ---
```

```
outputFile = 'filtered_signals.mat';
save(outputFile, 'filtered_signals');
fprintf('Filtered signals have been saved to %s\n', outputFile);
```

A.2. Nucleo Board Code

```
#include "fft.h"
#include "SSP_Global.h"
#include "arm_math.h"
#include "arm_const_structs.h"
#include "main.h"          // <-- Add this
#include "adc.h"           // <-- Add this
#include "cmsis_os.h"
#include "crc.h"
#include <math.h>          // fabsf
extern ADC_HandleTypeDef hadc4;  // <-- Add this



const float32_t iirCoeffs[10] = { 0.9813f, -0.9813f, 0.0f, 0.9623f, 0.0f, 0.6030f, 0.6030f,
    0.0f, 0.2060f, 0.0f };

static float32_t iirState[8];
static arm_biquad_cascade_df2T_instance_f32 iirFilter;
arm_rfft_fast_instance_f32 fftInstance;

 uint16_t fftDownsampled[FFT_DOWNSAMPLED_SIZE]; //todo:send to grafana

float32_t fftOutput[FFT_SIZE];       // FFT magnitude
static float32_t signal[FFT_SIZE];
float32_t arcEnergy = 0.0f; //todo:send to grafana
float D = 0.0f;  // discard factor
float F = 1.0f;  // band-weighting factor
int binsPerGroup = FFT_SIZE / FFT_DOWNSAMPLED_SIZE;


void WriteFFTBuffer(){
    SSP_I2C.arcEnergy = arcEnergy;
    memcpy(SSP_I2C.fftData,fftDownsampled,32);
    SSP_I2C.fftCrc = HAL_CRC_Calculate(&hcrc, (uint32_t*)SSP_I2C.FFT_Buffer,
        I2C_AMOUNT_FFT_DATA-4);
}

void ComputeFFT(uint16_t *adcBuffer)
{
    for (int i = 0; i < FFT_SIZE; i++) {
        signal[i] = ((float32_t)adcBuffer[i]) / 4096.0f;  // Normalize 12-bit ADC
    }

    // IIR filtering
    arm_biquad_cascade_df2T_f32(&iirFilter, signal, signal, FFT_SIZE);









    // Run FFT
    arm_rfft_fast_f32(&fftInstance, signal, fftOutput, 0);
    for (int i = 0; i < FFT_SIZE; i++) {
        fftOutput[i] = fabsf(fftOutput[i]);
    }

    // Weighting + Discard logic
    for (int i = 0; i < FFT_SIZE / 4; i++) {
        fftOutput[i] *= F;
```

```
    }

    int numToDiscard = (int)(FFT_SIZE * D);
    for (int d = 0; d < numToDiscard; d++) {
        int maxIndex = 0;
        float32_t maxVal = 0.0f;
        for (int i = 0; i < FFT_SIZE; i++) {
            if (fftOutput[i] > maxVal) {
                maxVal = fftOutput[i];
                maxIndex = i;
            }
        }
        fftOutput[maxIndex] = 0.0f;
    }
    for (int i = 0; i < FFT_DOWNSAMPLED_SIZE; i++) {
        float32_t sum = 0.0f;
        for (int j = 0; j < binsPerGroup; j++) {
            sum += fftOutput[i * binsPerGroup + j];
        }
        float32_t avg = sum / binsPerGroup;

        // Scale to integer range (adjust 1000.0f as needed)
        fftDownsampled[i] = (uint16_t)(avg * 1000.0f);
    }
    arcEnergy = 0.0f;
    for (int i = 0; i < FFT_SIZE; i++) {
        arcEnergy += fftOutput[i] * fftOutput[i];
    }

}

void StartFFTTask(void *argument)
{
    //  osDelay(1000);
    uint16_t adcBuffer[FFT_SIZE];
    arm_rfft_fast_init_f32(&fftInstance, FFT_SIZE);
    arm_biquad_cascade_df2T_init_f32(&iirFilter, 2, (float32_t *)iirCoeffs, iirState);

    while (1)
    {
        for (int i = 0; i < FFT_SIZE; i++) {
            adcBuffer[i] = ADC1_buffer[i * 4 + ADC_POS_Vline];  // Just use ADC_POS_Iout = 0
        }

        ComputeFFT(adcBuffer);
        WriteFFTBuffer();
        // Optional: print or log result
      //  printf("Arc Energy: %f\n", arcEnergy);  // Or send to OLED, UART, etc

        osDelay(500);  // Or however often you want to run FFT
    }
}
```

Python coding

The following Python scripts were developed to support the design, training, and validation of the proposed arc-fault detection algorithm. They cover the data acquisition, preprocessing, feature extraction, and machine-learning pipeline. Each script has a specific role in the thesis workflow.

arc_finder.py

Listing A.1: Streaming inference and debounce logic

```
import serial
import time
import joblib
import pandas as pd
import numpy as np
from datetime import datetime
```

```
7
8  # === Configuration ===
9  PORT = 'COM8'
10 BAUDRATE = 115200
11 MODEL_PATH = 'C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo
       /ML wow/Session2/arc_fault_model.joblib'
12
13 # Load model
14 model = joblib.load(MODEL_PATH)
15 print("  Model loaded.")
16
17 # Init serial
18 ser = serial.Serial(PORT, BAUDRATE, timeout=1)
19 time.sleep(2)
20
21 print("  Monitoring data stream... (Press Ctrl+C to stop)\n")
22
23 arc_counter = 0
24 ARC_CONFIRMATION_THRESHOLD = 10  # Number of consecutive arc predictions required
25
26 try:
27     while True:
28         line = ser.readline().decode('utf-8', errors='ignore').strip()
29         if not line:
30             continue
31
32         try:
33             values = [float(val) for val in line.split(',')]
34             if len(values) != 6:
35                 print(f"[WARN] Skipped malformed line: {line}")
36                 continue
37
38             arcEnergy, energy_band1, energy_band2, max_peak, stddev, centroid = values
39             input_features = pd.DataFrame([{
40                 'arcEnergy': arcEnergy,
41                 'energy_band1': energy_band1,
42                 'energy_band2': energy_band2,
43                 'max_peak': max_peak,
44                 'stddev': stddev,
45                 'centroid': centroid
46             }])
47             prediction = model.predict(input_features)[0]
48
49
50             if prediction == 1:
51                 arc_counter += 1
52             else:
53                 arc_counter = 0  # Reset on clean signal
54
55             # Trigger only if arc detected consistently for 5+ samples
56             if arc_counter >= ARC_CONFIRMATION_THRESHOLD:
57                 label = "   CONSISTENT ARC DETECTED"
58                 print(f"{datetime.now().strftime('%H:%M:%S')} | {values} => {label}")
59                 break  # Stop the loop immediately
60             else:
61                 label = "  No Arc"
62
63             print(f"{datetime.now().strftime('%H:%M:%S')} | {values} => {label}")
64
65         except ValueError:
66             print(f"[ERROR] Could not parse line: {line}")
67
68 except KeyboardInterrupt:
69     print("\n Stopped by user.")
70 finally:
71     ser.close()
```

This script runs the arc-fault detection in real time. It loads the trained model and applies it to incoming data. It also uses a debounce counter to avoid false trips, which is the same logic later used in the

embedded system.

data_over_time.py

Listing A.2: Timeline plotting of FFT and wavelet indicators

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file
file_path =     'C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo/ML wow/wavelet_fft_dataset4.csv'

df = pd.read_csv(file_path)

# Parse timestamps properly
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
df = df.dropna(subset=['timestamp']).sort_values('timestamp')

# Detect time gaps
df['time_diff'] = df['timestamp'].diff().dt.total_seconds()
gap_threshold = 1.0  # seconds
df['group'] = (df['time_diff'] > gap_threshold).cumsum()

# Plot FFT Energy (handles gaps)
plt.figure(figsize=(12, 4))
for _, group_data in df.groupby('group'):
    plt.plot(group_data['timestamp'], group_data['fft_energy'], color='red')
plt.title('FFT Energy Over Time')
plt.xlabel('Time')
plt.ylabel('Energy')
plt.tight_layout()
plt.show()


plt.figure(figsize=(12, 4))
for _, group_data in df.groupby('group'):
    plt.plot(group_data['timestamp'], group_data['wavelet_energy'], color='blue')
plt.title('Wavelet Energy Over Time')
plt.xlabel('Time')
plt.ylabel('Energy')
plt.tight_layout()
plt.show()
```

This code plots how the FFT and wavelet features change over time. These plots help to see when an arc starts and stops, and they were used to study the behaviour of the features in the lab and field tests.

evaluatemodel.py

Listing A.3: Evaluation script for trained models

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from joblib import dump
import pandas as pd

# Load data
df = pd.read_csv('C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo/ML wow/Session2/fft_features_labeled.csv')
X = df[[ 'arcEnergy', 'energy_band1', 'energy_band2', 'max_peak', 'stddev', 'centroid']
]
y = df['label']

# Train model with scaling
pipeline = make_pipeline(StandardScaler(), SVC(kernel='linear', C=1.0))
pipeline.fit(X, y)

# Save model
```

```
dump(pipeline, 'C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo/ML wow/Session2/arc_fault_model_svm.joblib')
print("  Model saved as arc_fault_model_svm.joblib")

# Extract trained model components
scaler = pipeline.named_steps['standardscaler']
svm = pipeline.named_steps['svc']

# Extract parameters
w_scaled = svm.coef_[0][0]
b_scaled = svm.intercept_[0]
mean = scaler.mean_[0]
scale = scaler.scale_[0]

# Convert to unscaled for C use
w_unscaled = w_scaled / scale
b_unscaled = b_scaled - (w_scaled * mean / scale)

# Output in C format
print("\n// C-ready linear SVM parameters")
print(f"double weight = {w_unscaled:.10f};")
print(f"double bias = {b_unscaled:.10f};")
```

This script tests the trained models on labelled data. It gives values such as accuracy and confusion matrix. It was useful to compare different models before choosing the final one for implementation.

import_serial.py

Listing A.4: Serial acquisition and CSV logging

```
import serial
import csv
import os
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from datetime import datetime

# ========= USER MODE =========
LABEL_MODE = 0# 0 = normal, 1 = arc, 2 = train
# =============================

SERIAL_PORT = 'COM16'  # adjust to your port
BAUDRATE = 115200
CSV_FILENAME = r"C:\Users\Francisco\OneDrive - Delft University of Technology\Uni\M2\Thesis wooo\ML wow\wavelet_fft_dataset27.csv"

# Logging Mode
if LABEL_MODE in [0, 1]:
    ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1)
    file_exists = os.path.exists(CSV_FILENAME)

    with open(CSV_FILENAME, mode='a', newline='') as file:
        writer = csv.writer(file)
        if not file_exists:
            writer.writerow(['timestamp', 'wavelet_energy', 'fft_energy', 'label'])

        print(f"[INFO] Logging with label={LABEL_MODE} to '{CSV_FILENAME}'")
        print("[INFO] Waiting for serial data...\n")

        last_wavelet = None
        last_arc = None

        try:
            while True:
                line = ser.readline().decode(errors='ignore').strip()

                if "Wavelet Energy" in line and "Arc Energy" in line:
                    try:
```

```
                        parts = line.replace("Wavelet Energy = ", "").replace("Arc Energy = "
                            , "").split(',')
                        if len(parts) == 2:
                            last_wavelet = float(parts[0].strip())
                            last_arc = float(parts[1].strip())
                        else:
                            print(f"[WARNING] Could not parse values from: {line}")
                            continue
                    except ValueError:
                        print(f"[WARNING] Could not convert values from: {line}")
                        continue


                elif "Arc Energy" in line:
                    try:
                        # Split by '=' instead of ':'
                        parts = line.split("=")
                        if len(parts) > 1:
                            # The value is the second part, strip whitespace and the trailing
                                comma
                            last_arc = float(parts[1].strip().strip(','))
                        else:
                            print(f"[WARNING] Incomplete 'Arc Energy' line received: {line}")
                            continue
                    except ValueError:
                        print(f"[WARNING] Could not parse 'Arc Energy' value from: {line}")
                        continue

                if last_wavelet is not None and last_arc is not None:
                    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
                    writer.writerow([timestamp, last_wavelet, last_arc, LABEL_MODE])
                    print(f"{timestamp} | Wavelet: {last_wavelet:.8f} | Arc: {last_arc:.8f} |
                        Label: {LABEL_MODE}")
                    last_wavelet = None
                    last_arc = None

        except KeyboardInterrupt:
            print("\n[INFO] Logging stopped.")
        finally:
            ser.close()

# Training Mode
elif LABEL_MODE == 2:
    print(f"[INFO] Training model from '{CSV_FILENAME}'...\n")
    if not os.path.exists(CSV_FILENAME):
        print("[ERROR] File not found.")
        exit(1)

    # Read CSV assuming it has header or not
    try:
        df = pd.read_csv(CSV_FILENAME)
        if 'label' not in df.columns:
            raise ValueError
    except:
        df = pd.read_csv(CSV_FILENAME, header=None,
                         names=['timestamp', 'wavelet_energy', 'fft_energy', 'label'])

    print(f"[INFO] Loaded {len(df)} samples.")
    print("[INFO] Label distribution:")
    print(df['label'].value_counts())

    if df['label'].nunique() < 2:
        print("[ERROR] Need both arc and normal samples to train.")
        exit(1)

    X = df[['wavelet_energy', 'fft_energy']].values
    y = df['label'].values

    # Train SVM with linear kernel to get coefficients
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    model = SVC(kernel='linear', C=1)
```

```
    model.fit(X_train, y_train)

    acc = model.score(X_test, y_test)
    print(f"\n SVM trained! Accuracy: {acc * 100:.2f}%")

    # Get coefficients
    coef = model.coef_[0]
    intercept = model.intercept_[0]

    print("\n Paste this into STM32 C code:")
    print("bool svm_predict_arc_fault(float wavelet_energy, float arc_energy) {")
    print("     float score = 0.0f;")
    print(f"     score += {coef[0]:.10f}f * wavelet_energy;")
    print(f"     score += {coef[1]:.10f}f * arc_energy;")
    print(f"     return (score >= { -intercept:.10f}f);  // Threshold based on intercept")
    print("}")

else:
    print("[ERROR] LABEL_MODE must be 0, 1, or 2.")
```

This script reads data from the STM32 board through the serial port and saves it to CSV files. The collected data was later used to train and test the machine-learning models.

SVM.py

Listing A.5: SVM decision boundary and feature-space visualisation

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os

# --- SVM Parameters (from training) ---
COEF_WAVELET_ENERGY = 422170.6925364132
COEF_ARC_ENERGY = 274.3991318018298
THRESHOLD = -23.698984395447503

# --- Dataset list ---
DATASET_PATHS = [

    r"C:\Users\Francisco\OneDrive - Delft University of Technology\Uni\M2\Thesis wooo\ML wow\wavelet_fft_dataset8.csv",
    r"C:\Users\Francisco\OneDrive - Delft University of Technology\Uni\M2\Thesis wooo\ML wow\wavelet_fft_dataset6.csv",


]

# Initialize combined data containers
all_arc = []
all_non_arc = []

min_wavelet = float('inf')
max_wavelet = float('-inf')

# Load and collect data
for path in DATASET_PATHS:
    if not os.path.exists(path):
        print(f"[WARNING] File not found: {path}")
        continue

    try:
        df = pd.read_csv(path)
        if 'label' not in df.columns:
            df = pd.read_csv(path, header=None, names=['timestamp', 'wavelet_energy', 'fft_energy', 'label'])
    except Exception as e:
        print(f"[ERROR] Could not read {path}: {e}")
        continue

```

```
    arc_df = df[df['label'] == 1]
    non_arc_df = df[df['label'] == 0]

    all_arc.append(arc_df)
    all_non_arc.append(non_arc_df)

    min_wavelet = min(min_wavelet, df['wavelet_energy'].min())
    max_wavelet = max(max_wavelet, df['wavelet_energy'].max())

# Combine data
arc_data = pd.concat(all_arc, ignore_index=True)
non_arc_data = pd.concat(all_non_arc, ignore_index=True)

# --- Plot ---
plt.figure(figsize=(12, 8))

# Non-arc: blue
plt.scatter(non_arc_data['wavelet_energy'], non_arc_data['fft_energy'],
            color='blue', alpha=0.6, s=40, label='Non-Arc (Label 0)')

# Arc: red
plt.scatter(arc_data['wavelet_energy'], arc_data['fft_energy'],
            color='red', alpha=0.6, s=40, label='Arc (Label 1)')

# --- Decision boundary ---
padding = (max_wavelet - min_wavelet) * 0.1
x_line = np.linspace(min_wavelet - padding, max_wavelet + padding, 200)

if abs(COEF_ARC_ENERGY) < 1e-18:
    if COEF_WAVELET_ENERGY != 0:
        vertical_x = THRESHOLD / COEF_WAVELET_ENERGY
        plt.axvline(x=vertical_x, color='green', linestyle='--', label='SVM Decision Boundary
            ')
else:
    y_line = (-THRESHOLD - COEF_WAVELET_ENERGY * x_line) / COEF_ARC_ENERGY
 #   plt.plot(x_line, y_line, color='green', linestyle='--', linewidth=2, label='SVM Decision
      Boundary')

# --- Style ---
plt.title("Arc vs Non-Arc Classification with SVM Boundary (All Datasets)", fontsize=15)
plt.xlabel("Wavelet Energy")
plt.ylabel("FFT Energy")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
plt.show()
```

This file trains a Support Vector Machine and shows the decision boundary in the feature space. It gives a clear view of how the FFT and wavelet features separate normal cases from arc cases.

trainmodelandsave.py

Listing A.6: Training and export of C parameters for MCU

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from joblib import dump
import pandas as pd

# Load full feature-labeled dataset
df = pd.read_csv('C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis
    wooo/ML wow/fft_features_labeled.csv')

# Use all real features (excluding timestamp and label)
feature_columns = ['arcEnergy', 'energy_band1', 'energy_band2', 'stddev']
X = df[feature_columns]
y = df['label']

# Train model pipeline
```

```
pipeline = make_pipeline(StandardScaler(), SVC(kernel='linear', C=1.0))
pipeline.fit(X, y)

# Save the model
dump(pipeline, 'C:/Users/Francisco/OneDrive - Delft University of Technology/Uni/M2/Thesis wooo/ML wow/Session2/arc_fault_model_svm.joblib')
print("  Model saved as arc_fault_model_svm.joblib")

# Extract components
scaler = pipeline.named_steps['standardscaler']
svm = pipeline.named_steps['svc']

# Unscale weights and bias
scaled_weights = svm.coef_[0]
intercept = svm.intercept_[0]
means = scaler.mean_
scales = scaler.scale_

unscaled_weights = scaled_weights / scales
unscaled_bias = intercept - (scaled_weights @ (means / scales))

# Print in C style
print("\n// === C-ready Linear SVM Parameters ===")
for i, (feature, w) in enumerate(zip(feature_columns, unscaled_weights)):
    print(f"double w_{feature} = {w:.10f};")
print(f"double bias = {unscaled_bias:.10f};")

# Optional: array style for microcontroller use
print("\n// Array form for microcontroller usage")
print("double weights[] = { " + ", ".join(f"{w:.10f}" for w in unscaled_weights) + " };")
print(f"double bias = {unscaled_bias:.10f};")
```

This script trains the final SVM model and then saves the model parameters in a format ready for the MCU. It is the link between the offline Python training and the real-time use of the algorithm on the hardware.

A.3. Breaker code

```
#include "ArcCheck.h"
#include "SSP_Global.h"
#include "arm_math.h"
#include "arm_const_structs.h"
#include "main.h"
#include "adc.h"
#include "cmsis_os.h"
#include "crc.h"
#include <math.h>
#include "usart.h"
#include "string.h"
#include "stdio.h"
#include "DCO_Utilities.h"
#include <stdbool.h>

extern ADC_HandleTypeDef hadc4;

// --- Configuration and Filter Parameters ---
float score = 0.0f;
static LP_IIR arcFilter;

const float32_t iirCoeffs[10] = {
    0.92848624f, 1.85697247f, 0.92848624f, -1.85214638f, -0.86234863f,
    1.00000000f, -2.00000000f, 1.00000000f, 1.99970381f, -0.99970385f
};

static float32_t iirState[4 * 2];
static arm_biquad_cascade_df2T_instance_f32 iirFilter;
arm_rfft_fast_instance_f32 fftInstance;

float D = 0.0f;
```

```
float F = 1.0f;

// --- FFT Buffers ---
uint16_t fftDownsampled[FFT_DOWNSAMPLED_SIZE];
float32_t fftOutputRaw[FFT_SIZE];
float32_t fftOutput[FFT_SIZE];
static float32_t signal[FFT_SIZE];

static uint16_t adcBuffer[FFT_SIZE];
int binsPerGroup = FFT_SIZE / FFT_DOWNSAMPLED_SIZE;

// --- Wavelet Buffers and Filters ---
static const float32_t db3_Lo_D[DB3_FILTER_LENGTH] = {
    0.33267055f, 0.80689150f, 0.45987750f,
   -0.13501102f, -0.08544127f, 0.03522629f
};

static const float32_t db3_Hi_D[DB3_FILTER_LENGTH] = {
   -0.03522629f, -0.08544127f, 0.13501102f,
    0.45987750f, -0.80689150f, 0.33267055f
};

static arm_fir_instance_f32 lp_fir[WAVELET_LEVELS];
static arm_fir_instance_f32 hp_fir[WAVELET_LEVELS];

static float32_t lp_state[WAVELET_LEVELS][DB3_FILTER_LENGTH + WAVELET_SIZE];
static float32_t hp_state[WAVELET_LEVELS][DB3_FILTER_LENGTH + WAVELET_SIZE];

static float32_t approx_buf[WAVELET_LEVELS + 1][WAVELET_SIZE];
static float32_t detail_buf[WAVELET_LEVELS][WAVELET_SIZE / 2];

static float32_t lowBF[WAVELET_SIZE];
static float32_t highBF[WAVELET_SIZE];

// --- Arc Detection State ---
float powerLevel1, powerLevel2, powerLevel3, powerLevel4, powerLevel5;
float32_t powerLevel2Filtered = 0.0f;
float32_t arcEnergy = 0.0f;
float32_t arcEnergyFiltered = 0.0f;
_Bool arcTrip[2] = {RESET};
#define MU_WAVELET   2.2051726353e-05f
#define SD_WAVELET   2.7432931761e-06f
#define MU_FFT       0.0519805305f
#define SD_FFT       0.0076150031f

#define W1_Z         1.0068857601f
#define W2_Z         1.2239607426f
#define B_Z         -0.6171940634f
// --- Initialization Functions ---
void WaveletDB3_Init(void) {
    for (int lvl = 0; lvl < WAVELET_LEVELS; ++lvl) {
        arm_fir_init_f32(&lp_fir[lvl], DB3_FILTER_LENGTH, (float32_t*)db3_Lo_D, lp_state[lvl
            ], WAVELET_SIZE);
        arm_fir_init_f32(&hp_fir[lvl], DB3_FILTER_LENGTH, (float32_t*)db3_Hi_D, hp_state[lvl
            ], WAVELET_SIZE);
    }
}

void WriteFFTBuffer() {
    SSP_I2C.arcEnergy = arcEnergyFiltered;
    memcpy(SSP_I2C.fftData, fftDownsampled, FFT_DOWNSAMPLED_SIZE * sizeof(uint16_t));
    SSP_I2C.fftCrc = HAL_CRC_Calculate(&hcrc, (uint32_t*)SSP_I2C.FFT_Buffer,
        I2C_AMOUNT_FFT_DATA - 4);
}

// --- Wavelet Decomposition ---
void WaveletDB3_Decompose(const float32_t *input) {
    memcpy(approx_buf[0], input, sizeof(float32_t) * WAVELET_SIZE);
    for (int lvl = 0; lvl < WAVELET_LEVELS; ++lvl) {
        uint32_t inLen = WAVELET_SIZE >> lvl;
        uint32_t outLen = inLen >> 1;
```

```
        arm_fir_f32(&lp_fir[lvl], approx_buf[lvl], lowBF, inLen);
        arm_fir_f32(&hp_fir[lvl], approx_buf[lvl], highBF, inLen);
        for (uint32_t i = 0; i < outLen; ++i) {
            approx_buf[lvl + 1][i] = lowBF[2 * i];
            detail_buf[lvl][i] = highBF[2 * i];
        }
    }
}

void ComputeDB3Powers(volatile uint16_t *rawAdc) {
    static float32_t signal[WAVELET_SIZE];
    for (uint32_t i = 0; i < WAVELET_SIZE; ++i) {
        signal[i] = ((float32_t)rawAdc[i]) / 4096.0f;
    }
    WaveletDB3_Decompose(signal);

    uint32_t len;
    float e;

    e = 0; len = WAVELET_SIZE >> 1; for (uint32_t i = 0; i < len; ++i) e += detail_buf[0][i]
        * detail_buf[0][i]; powerLevel1 = e;
    e = 0; len = WAVELET_SIZE >> 2; for (uint32_t i = 0; i < len; ++i) e += detail_buf[1][i]
        * detail_buf[1][i]; powerLevel2 = e;
    LowPass_IIR_Run(&arcFilter, powerLevel2, &powerLevel2Filtered);
    e = 0; len = WAVELET_SIZE >> 3; for (uint32_t i = 0; i < len; ++i) e += detail_buf[2][i]
        * detail_buf[2][i]; powerLevel3 = e;
    e = 0; len = WAVELET_SIZE >> 4; for (uint32_t i = 0; i < len; ++i) e += detail_buf[3][i]
        * detail_buf[3][i]; powerLevel4 = e;
    e = 0; len = WAVELET_SIZE >> 5; for (uint32_t i = 0; i < len; ++i) e += detail_buf[4][i]
        * detail_buf[4][i]; powerLevel5 = e;
}

// --- FFT Processing ---
void ComputeFFT(uint16_t *adcBuffer) {
    for (int i = 0; i < FFT_SIZE; i++) {
        signal[i] = ((float32_t)adcBuffer[i]) / 4096.0f;
    }

    arm_rfft_fast_f32(&fftInstance, signal, fftOutputRaw, 0);

    for (int i = 5; i < FFT_SIZE; i++) {
        fftOutput[i] = fabsf(fftOutputRaw[i]);
    }

    for (int i = 0; i < FFT_SIZE / 8; i++) {
        fftOutput[i] *= F;
    }

    int numToDiscard = (int)(FFT_SIZE * D);
    for (int d = 0; d < numToDiscard; d++) {
        int maxIndex = 0;
        float32_t maxVal = 0.0f;
        for (int i = 0; i < FFT_SIZE; i++) {
            if (fftOutput[i] > maxVal) {
                maxVal = fftOutput[i];
                maxIndex = i;
            }
        }
        fftOutput[maxIndex] = 0.0f;
    }

    for (int i = 0; i < FFT_DOWNSAMPLED_SIZE; i++) {
        float32_t sum = 0.0f;
        for (int j = 0; j < binsPerGroup; j++) {
            sum += fftOutput[i * binsPerGroup + j];
        }
        float32_t avg = sum / binsPerGroup;
        fftDownsampled[i] = (uint16_t)(avg * 1000.0f);
    }

    arcEnergy = 0.0f;
```

```
    for (int i = 0; i < FFT_SIZE; i++) {
        arcEnergy += fftOutput[i] * fftOutput[i];
    }
    LowPass_IIR_Run(&arcFilter, arcEnergy, &arcEnergyFiltered);
}

// --- Arc Trip Functions ---
_Bool getArcTripStatus(uint8_t line) { return arcTrip[line]; }
void setArcTripStatus(uint8_t line) { arcTrip[line] = SET; }
void resetArcTripStatus(uint8_t line) { arcTrip[line] = RESET; }

/* --- Trip debounce + latch inside Trip_CheckCondition --- */
#define TRIP_DEBOUNCE_BLOCKS   5    /* consecutive arc=1 to trip   */
#define CLEAR_DEBOUNCE_BLOCKS 50    /* consecutive arc=0 to clear  */
#define MIN_HOLD_BLOCKS       20    /* min blocks to keep tripped  */
#define AUTO_RECLOSE           0    /* 0=latch until manual reset; 1=auto */

void Trip_CheckCondition(float wavelet_energy, float arcEnergy)
{
    static uint16_t pos_ctr  = 0;   /* arc=1 streak */
    static uint16_t neg_ctr  = 0;   /* arc=0 streak */
    static uint16_t hold_ctr = 0;   /* time held in TRIP */
    static _Bool    latched  = false;

    _Bool arc = svm_predict_arc_fault(wavelet_energy, arcEnergy);

    if (!latched) {
        /* Healthy: look for TRIP after N consecutive arc detections */
        if (arc) {
            if (++pos_ctr >= TRIP_DEBOUNCE_BLOCKS) {
                latched  = true;
                hold_ctr = 0;
                pos_ctr  = 0;
                neg_ctr  = 0;
                setArcTripStatus(Positive);   /* command TRIP */
            }
        } else {
            pos_ctr = 0;
        }
    } else {
        /* Already tripped: enforce minimum hold; optionally auto-reclose */
        hold_ctr++;

    #if AUTO_RECLOSE
        if (!arc) {
            if (++neg_ctr >= CLEAR_DEBOUNCE_BLOCKS && hold_ctr >= MIN_HOLD_BLOCKS) {
                latched = false;
                pos_ctr = neg_ctr = 0;
                resetArcTripStatus(Positive); /* allow reclose */
            }
        } else {
            neg_ctr = 0;
        }
    #else
        (void)neg_ctr; /* keep compiler happy if not used */
        /* stay latched until external reset (manual) */
    #endif
    }
}


bool svm_predict_arc_fault(float waveletE, float fftE)
{
    float z1 = (waveletE - MU_WAVELET) / SD_WAVELET;
    float z2 = (fftE     - MU_FFT    ) / SD_FFT;

     score  = W1_Z*z1 + W2_Z*z2 + B_Z;
    return (score >= 0.0f);                 /* arc   score   0 */
}

// --- Main Task ---
```

```
237 void StartArcCheckTask(void *argument) {
238     arm_rfft_fast_init_f32(&fftInstance, FFT_SIZE);
239     arm_biquad_cascade_df2T_init_f32(&iirFilter, 2, (float32_t *)iirCoeffs, iirState);
240     WaveletDB3_Init();
241     LowPass_IIR_Init(&arcFilter, 500, 1);
242 
243     while (1) {
244         for (int i = 0; i < FFT_SIZE; i++) {
245             adcBuffer[i] = ADC1_buffer[i * 4 + 0];
246         }
247         ComputeDB3Powers(adcBuffer);
248         ComputeFFT(adcBuffer);
249         //Trip_CheckCondition(powerLevel2Filtered, arcEnergyFiltered);
250         WriteFFTBuffer();
251         char dataBuffer[100];
252         int len = snprintf(dataBuffer, sizeof(dataBuffer),
253                            "Wavelet Energy = %.9f, Arc Energy = %.8f\r\n",
254                            powerLevel2Filtered, arcEnergyFiltered);
255         HAL_UART_Transmit(&huart4, (uint8_t*)dataBuffer, len, 100);
256 
257         osDelay(2);
258     }
259 }
```