Report

AMESP - Phase 2 WP3

Title : AMESP - Phase 2 WP3

Authors : K. S. Moreira, Luewton L F Agostinho, R.H. Braga Leivas

Date : December 30, 2024

Code :

Version : 1

Status : draft

Mailing list :

Copy to :

Classification : confidential

Contents

1 WP3 Overview

1.1 Objectives

As stated on the proposal, we have two main goals in the WP3:

1. Investigate methods to identify the electrospray mode during operation by reading current values
2. Investigate methods to perform corrective actions to restore the electrospray operation by sending commands to the pump and/or power supply

We have explored two approaches that could achieve these goals, and they are detailed on Sections 2 and **??**. For each approach, we discuss how they can be implemented in the system developed by Gilbert, their advantadges and disadvantages, and how well they perform in achieving the above goals.

Section **??** shows some results obtained during the control investigations that were not directly used to achieve the goals of this Work Package, but we believe they could still prove useful to Gilbert and provide inputs to how the multinozzle prototype operates.

1.2 Timeline of Executed Tasks

Figure 1 shows a timeline of executed tasks for Work Package 3.

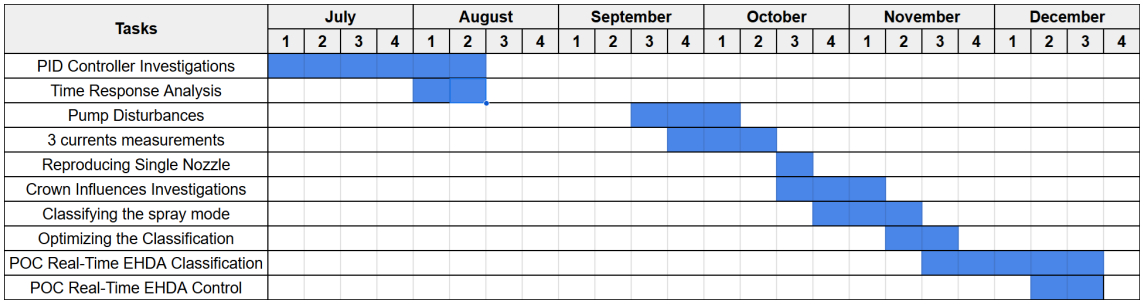

Figure 1. Timeline of executed tasks.

The tasks on Figure 1 relate directly to the sections of this report in the table of contents. The gap between August and September corresponds to the holiday period of several members of the project. Tasks executed of other Work Packages are not included in the timeline.

1.3 Conventions

Throughout this report, we'll refer multiple times to physical variables of the system. Figure 2 shows the variable conventions we'll use in this text.

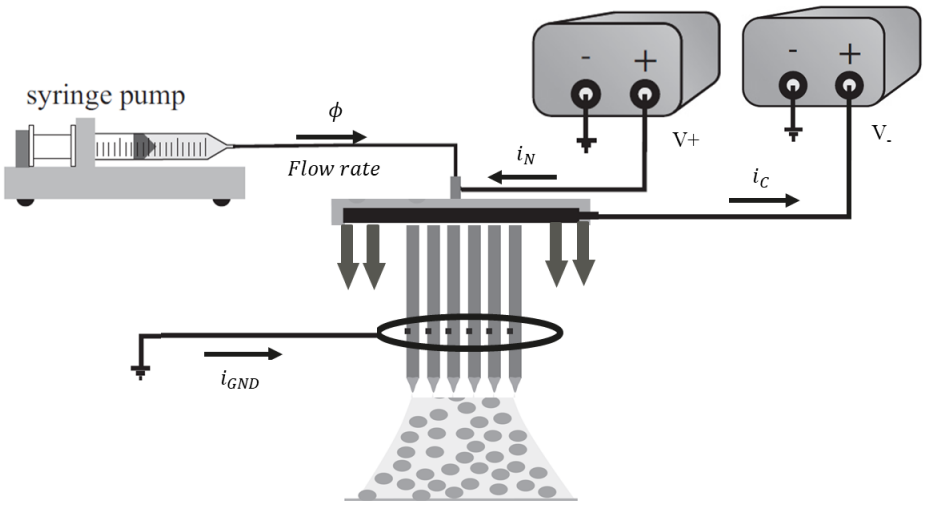

Figure 2. Variable conventions and nomenclature. Source: adapted from Verdoold et al. (2013).

As shown in Figure 2, we have

- i_N: current flowing from the positive high voltage source to the nozzles
- i_C: current flowing from the crown to the negative high voltage source
- i_{GND}: current flowing from the ground to the ring
- ϕ: flow rate of the syringe pump
- V_+: voltage of the positive high voltage source
- V_-: voltage of the negative high voltage source

The direction of the currents was chosen as shown in Figure 2 to ensure they are always positve in the measurements, facilitating the analysis.

2 Current-based Classification and Control

2.1 Proof-of-Concept Real-time EHDA Classification

Now that we know how to acquire the signal, we can develop a Proof-of-Concept (POC) algorithm that classifies the spraying mode in real-time by looking at the current signal. Algorithm 1 shows a pseudocode for the algorithm developed.

Algorithm 1 consists of two helper functions - `GetOscilloscopeMeasurement` and `ClassifyEHDA`. The former is responsible for acquiring and filtering the signal of

i_{GND} read, while the latter classifies the filtered data based on the standard deviation. For simplicity, we used only the standard deviation to classify the spray mode, but the RSD can also be used as discussed previously. The main loop continuouly acquires the data and classifies it.

Do notice that the function that acquires the raw data is highly dependant on the language and hardware used, therefore it was not detailed in the pseudocode. We used a TiePie oscilloscope, which provides an API in Python.

For the complexity of the algorithm, the slowest operations are the Fast Fourier Transform algorithms, with complexity $\mathcal{O}(n \log n)$. Therefore, the complexity of the entire algorithm is $\mathcal{O}(n \log n)$, where $n = T_S \cdot f_s$ i.e. n is the sample size of the measurement.

Figure 3 shows the result of Algorithm 1 implemented in Python.

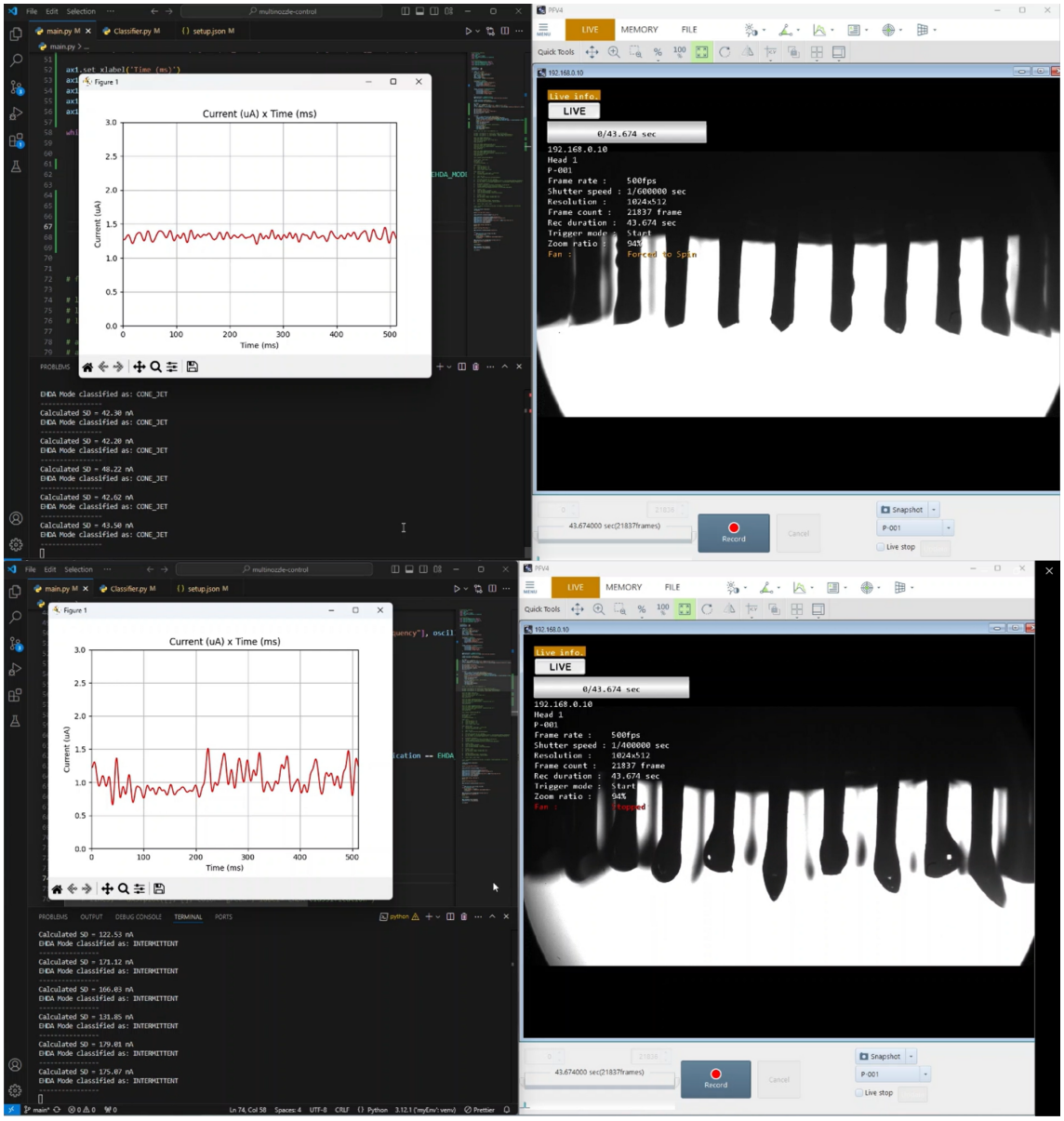

Figure 3. Result of Algorithm 1 implemented in Python.

As we can see on Figure 3, when all the nozzles operate in the cone-jet mode, the acquired signal is stable and displays a small standard deviation. On the other hand, when the nozzles are intermittent, the acquired signal displays a large standard deviation, which is used for the classification.

Figure 4 shows some issues identified, that we need to keep in mind during the classification.

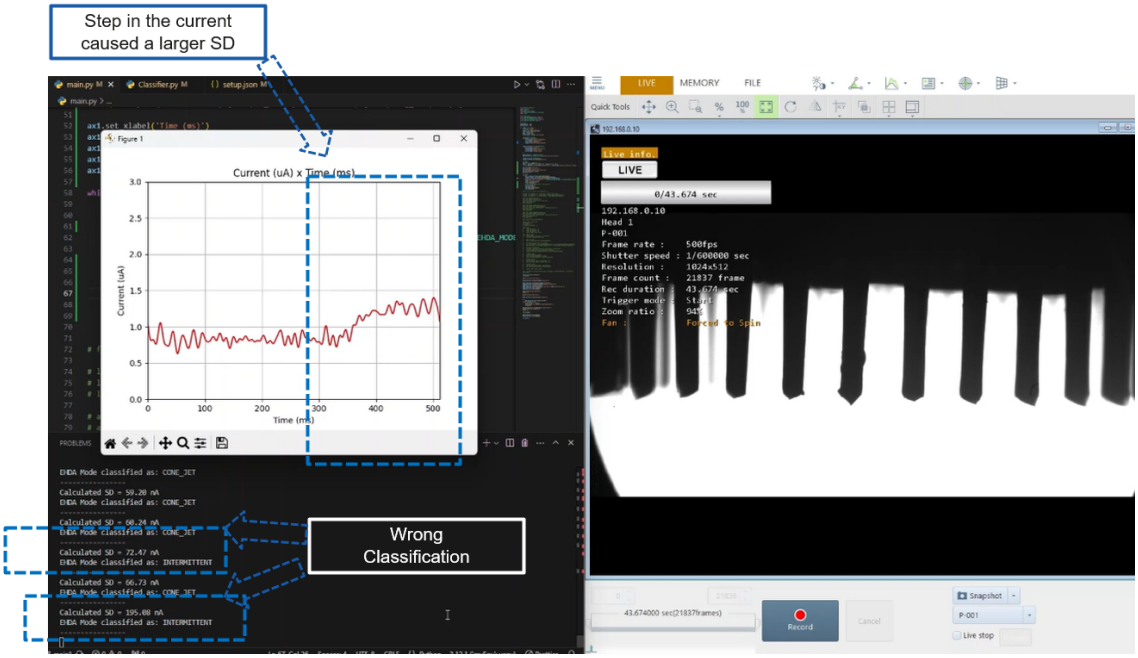

Figure 4. Issues identified on Algorithm 1 implemented in Python.

Firstly, we see that the signal can display "steps", going from one average value of current to another as we see on Figure 4. This leads to a large calculated standard deviation, causing the classification to be intermittent despite all the nozzles operating in a cone-jet mode. This can be solved by discarding measurements where the mean value of the current is different on the start and end of the window.

Secondly, it was observed that signal occasionally did not bahave as expected. There were several instances where the acquired signal would display a much larger standard deviation than expected, and then immediately return to the expected pattern. To reduce the impact of these unexpected instances, we can use a classification based on subsequent windows: instead of looking at only the signal acquired on the current time window, we look at the previous 5 classifications as well. If the majority of the previous classifications is also cone-jet, then we continue to classify the current time window as cone-jet.

This stategy has the main drawback of being significantly slower than looking at just the current time window. When the spray mode indeed changes from intermittent to cone-jet, we need to wait for the previous time windows to update as well. We attempted to reduce the time window to $T_S = 250\,\text{ms}$ to make this option viable, but with such small time windows the calculated standard deviation did not allow for a reliable classification, like we had seen in **??**. $T_S = 500\,\text{ms}$ was the minimum T_S that the algorithm still worked reliably.

During this time, we faced significant challenges to reproduce the results on Figure 3 with the sprayhead. We believe this could have been caused by degradation of the sprayhead and its needles. From now on we used a new sprayhead, with which we spray upside down, and used $f_s = 5\,\text{kHz}$ with a time window of $T_S = 1\,\text{s}$, since we saw we could reproduce the results more easily with this configuration.

2.2 Proof-of-Concept Real-Time EHDA Control

We can make a small change on Algorithm 1 to implement small corrections on V_+ if the signal is classified as intermittent, as shown on Algorithm 2 below. Note that we have not shown all procedures ass we did on Algorithm 1, since they've remained unchanged.

The voltage step of 400 V was determined experimentally. Note that we only did simple tests for this POC, more tests are necessary to find the ideal value the voltage step. Figure 5 shows the result obtained of Algorithm 2 implemented in Python.

On Figure 5, for the first 10 seconds we keep $V_+ = 5500\,\text{V}$ to stabilize the system. On $t = 10\,\text{s}$ we drop the voltage by 1 kV to cause an intermittent spraying mode and then we activate the control algorithm. As we see on the first image on the top, once the nozzles are intermittent the calculated standard deviation is above the threshold of 70 nA. This causes the control algorithm to increase V_+ by steps of 400 V until the standard deviation is below the threshold - which corresponds to a cone-jet spraying mode seen on the bottom image. Once this is achieved, the controller stops to increase the voltage, keeping it fixed.

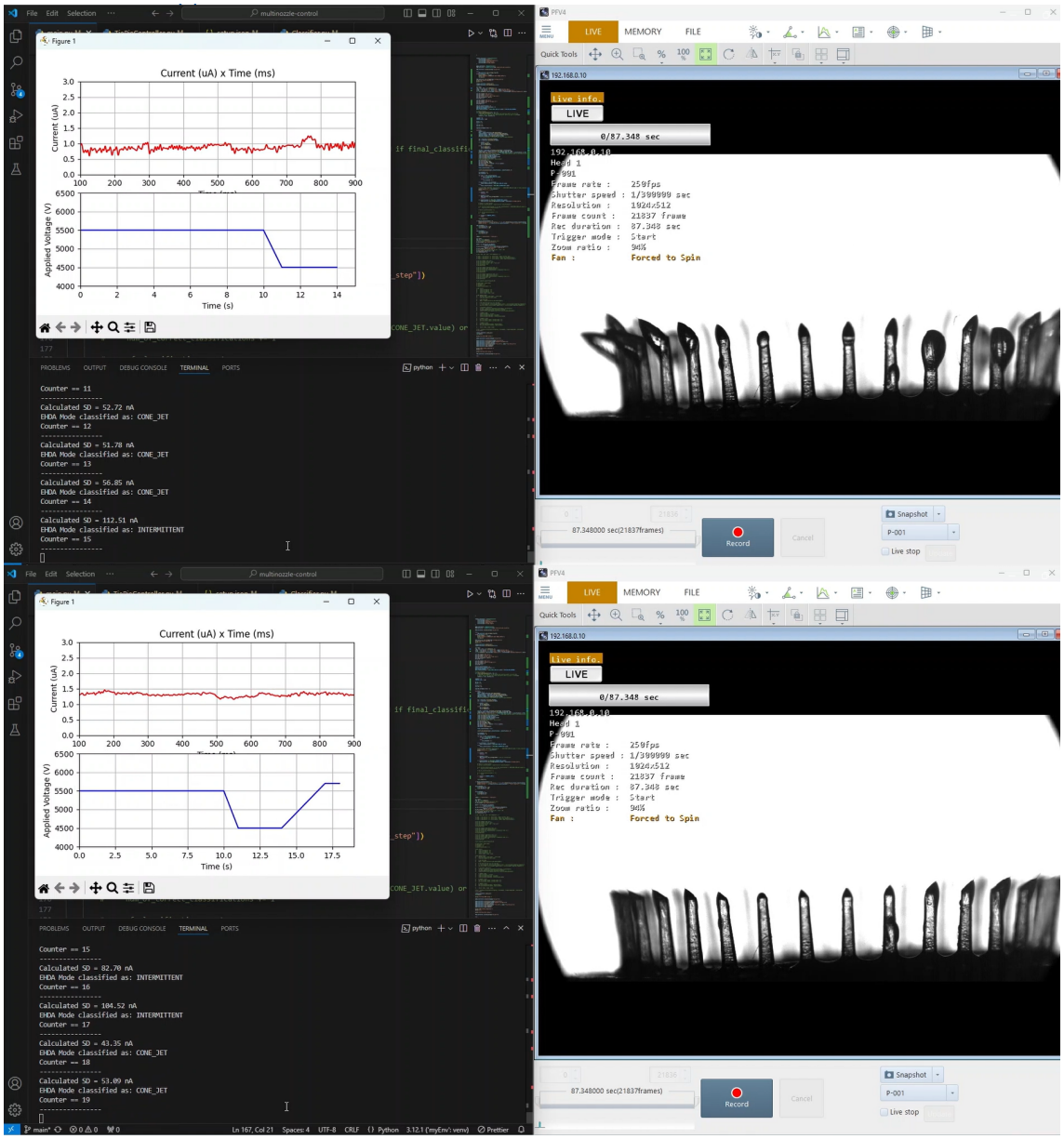

Figure 5. Result of Algorithm 2 implemented in Python.

References

Verdoold, S., Agostinho, L., Yutteri, C., and Marijnissen, J. (2013). A generic electrospray classification. *Journal of Aerosol Science*, 67:87–103.

Algorithm 1 Real-time EHDA Classification.

```
1: ▷ This procedure obtains the raw data fom the oscilloscope and filters it, returning it to the caller
2: procedure GetOscilloscopeMeasurement()
3:     raw_data ← Array[n]                ▷ n is the sample size, defined as n = T_S · f_s.
4:     raw_data ← GetOscilloscopeData()                                   ▷ O(n)

5:     ▷ FFT - Fast Fourier Transform. O(n log n)
6:     fft_result ← GetFFT(raw_data)
7:     ▷ Remove unwanted frequencies from the frequency spectrum.
8:     low_pass_cutoff ← 100
9:     bandstop_center ← 50
10:    bandstop_width ← 2
11:    bandstop_start ← bandstop_center - bandstop_width
12:    bandstop_end ← bandstop_center + bandstop_width
13:    for all f in fft_result do
14:        if f ≥ low_pass_cutoff then
15:            f ← 0
16:        end if
17:        if f > bandstop_start or f < bandstop_endt then
18:            f ← 0
19:        end if
20:    end for

21:    ▷ Rebuild signal using IFFT - Inverse FFT. O(n log n)
22:    recovered_signal ← GetIFFT(fft_result)
23:    return recovered_signal
24: end procedure

25: procedure ClassifyEHDA(filtered_data)
26:    sd_threshold ← 70 · 10^−9          ▷ Threshold for the standard deviation
27:    calculated_sd ← CalcStdDeviation(filtered_data)                    ▷ O(n)
28:    if calculated_sd > sd_threshold then
29:        return INTERMITTENT
30:    else
31:        return CONE_JET
32:    end if
33: end procedure

34: while 1 do
35:    filtered_data ← GetOscilloscopeMeasurement()                       ▷ O(n log n)
36:    classification ← ClassifyEHDA(filtered_data)                       ▷ O(n)
37:    if classification == CONE_JET then
38:        print(CONE_JET)
39:    else
40:        print(INTERMITTENT)
41:    end if
42: end while
```

Algorithm 2 Real-time EHDA Control.

```
voltage_step ← 400
while 1 do
    filtered_data ← GetOscilloscopeMeasurement()
    classification ← ClassifyEHDA(filtered_data)
    applied_voltage ← GetAppliedVoltage()
    if classification == INTERMITTENT then
        SetAppliedVoltage(applied_voltage + voltage_step)
    end if
end while
```