Documentation in regard to the *FFF-Line-Segmentation* repository - *fff_segmenter* algorithm

Abstract

The ***fff_segmenter*** algorithm was developed to address the need for an automatic and precise method to determine the start and end indexes of specific geometrical elements in a monolayer Fused Filament Fabrication (FFF) printed part, referred to in this documentation as the "Workpiece." This is done by analyzing acoustic signals collected during the part's fabrication, aimed at researching the monitoring of the FFF process in-situ. The algorithm was developed for Cartesian RepRap-based FFF printers, which are commonly used in research settings.

Traditionally, segmentation of geometrical elements within the signal was performed manually. Users would identify specific amplitude variations in the signal that indicated the start and end of geometrical elements. This was done using scripts to automatically detect these variations, or by synchronizing the fabrication signal with video footage of the printing process and manually annotating start and end moments, converting these to indexes using the Frequency sampling (Fs) value used during signal acquisition. However, these methods were prone to errors and high variability due to user-dependent factors and video frame rate discrepancies.

The ***fff_segmenter*** algorithm leverages three synchronously acquired signals during the fabrication of the monolayer FFF part, all using the same Fs:

1. The monitoring signal (obtained from sensors attached to the printer, such as Acoustic Emission sensors in the provided example datasets);
2. The direction control signal for the X-axis stepper motors;
3. The direction control signal for the Y-axis stepper motors.

This documentation details the segmentation logic, which is designed to depend on the mechanical fabrication movements determined in the G-code of the workpiece, the geometrical elements of the workpiece, and the specific characteristics of Cartesian RepRap-based FFF printers. Additionally, it covers the development process of the ***fff_segmenter*** algorithm.

In summary, ***fff_segmenter*** addresses the need in FFF process monitoring research for an automatic, accurate, and scalable signal segmentation script, providing a framework for the community to extend the ***fff_segmenter*** algorithm to other FFF printers and parts.

Summary

Acquirement of the Necessary Signals

The ***fff_segmenter*** algorithm leverages three synchronously acquired signals during the fabrication of the monolayer Fused Filament Fabrication (FFF) part, all using the same Frequency sampling (Fs):

1. The monitoring signal (obtained from sensors attached to the printer, such as Acoustic Emission sensors in the provided example datasets);
2. The direction control signal for the X-axis stepper motors;
3. The direction control signal for the Y-axis stepper motors.

The algorithm was developed for Cartesian RepRap-based FFF printers, which are commonly used in research settings. In these type of printers, the direction control signal for both X-axis and Y-axis stepper motors are provided by the RepRap Arduino Mega Pololu Shield (RAMPS) Stepper Drivers, as displayed in Figure 1.

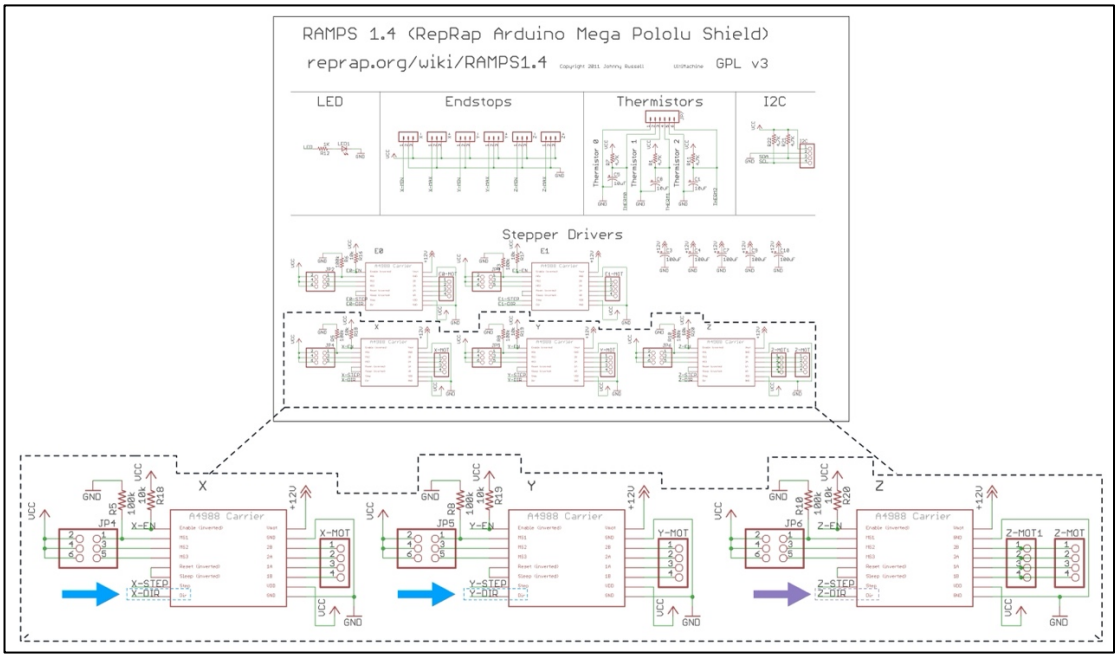

Figure 1 – RAMPS Schematic with indications of the stepper motors direction control signals (altered from https://reprap.org/wiki/Arduino_Mega_Pololu_Shield)

During the signal acquisition process, the "X-DIR" and "Y-DIR" labelled pins, depicted in Figure 1, were connected to a Data Acquisition Device (DAQ) by inserting splits to the pins, as shown in Figure 2. The signals were collected *in-situ* during the fabrication of the workpiece. The workpiece's .gcode file is available on the FFF-Line-Segmentation GitHub repository (.../gcodes). The signals were recorded at a specified sampling frequency (Fs) and temporarily stored on the DAQ hard drive. Subsequently, they were transferred to a Personal Computer (PC) for further signal processing.

In the three exemplary datasets available on the FFF-Line-Segmentation GitHub repository (.../Data), the Fs used during the signal acquisition process was 200 kS/s. The DAQ used was the DL 850 oscilloscope from Yokogawa®, Tokyo, Japan, and the 3D printer was the Cartesian RepRap-based FFF Core A1V2 printer from GTMax3D, Americana, Brazil.

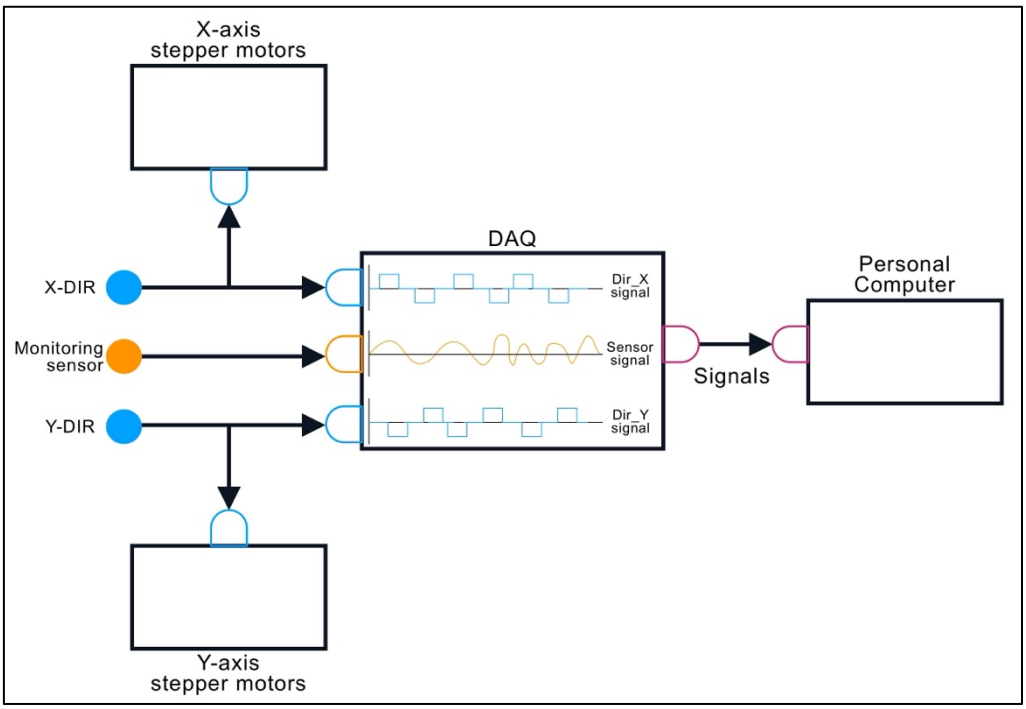

Figure 2 – Signal acquisition process representation

Regarding the monitoring sensor signals, the three exemplary datasets feature different sensor positions on the 3D printer:

1. Test1.mat: This dataset includes the 'Acoustic_signal' monitoring sensor signal, acquired by an acoustic sensor attached to the printer table.
2. Test2.mat: This dataset includes two monitoring sensor signals:
    - 'Acoustic_signal_table': Acquired by an acoustic sensor placed on the printer bed.
    - 'Acoustic_signal_extruder': Acquired by an acoustic sensor placed on the printer extruder block.
3. Test3.mat: This dataset includes two monitoring sensor signals:
    - 'Signal_extruder': Acquired by an acoustic sensor placed on the printer extruder block.
    - 'Signal_extruder2': Acquired by an acoustic sensor placed in a different position on the printer extruder block.

These three datasets were made available to demonstrate that the fff_segmenter algorithm can handle monitoring sensor signals from different sources and accommodate

variations in the start and end indexes of specific geometrical elements of the printed part, which are common in the acquisition process.

Considerations Regarding the Workpiece

The fabrication method used as a reference for the development of the segmentation algorithm consists of the elements mentioned in Figure 3. Among them, it is noticeable that the skirt and the external pattern exhibit similarities in terms of their fill behavior. These similarities lie in the fact that both the skirt and the external pattern feature multiple contour lines, some of which are fabricated at orthogonal angles (0°, 90°, 180°, 270°). This complicates the ability of the segmentation algorithm to utilize this fill behavior (orthogonal fabrications on one of the axes), as the segmentation could confuse the contour lines of the skirt (which are not part of the workpiece itself) with the contour lines of the external pattern.

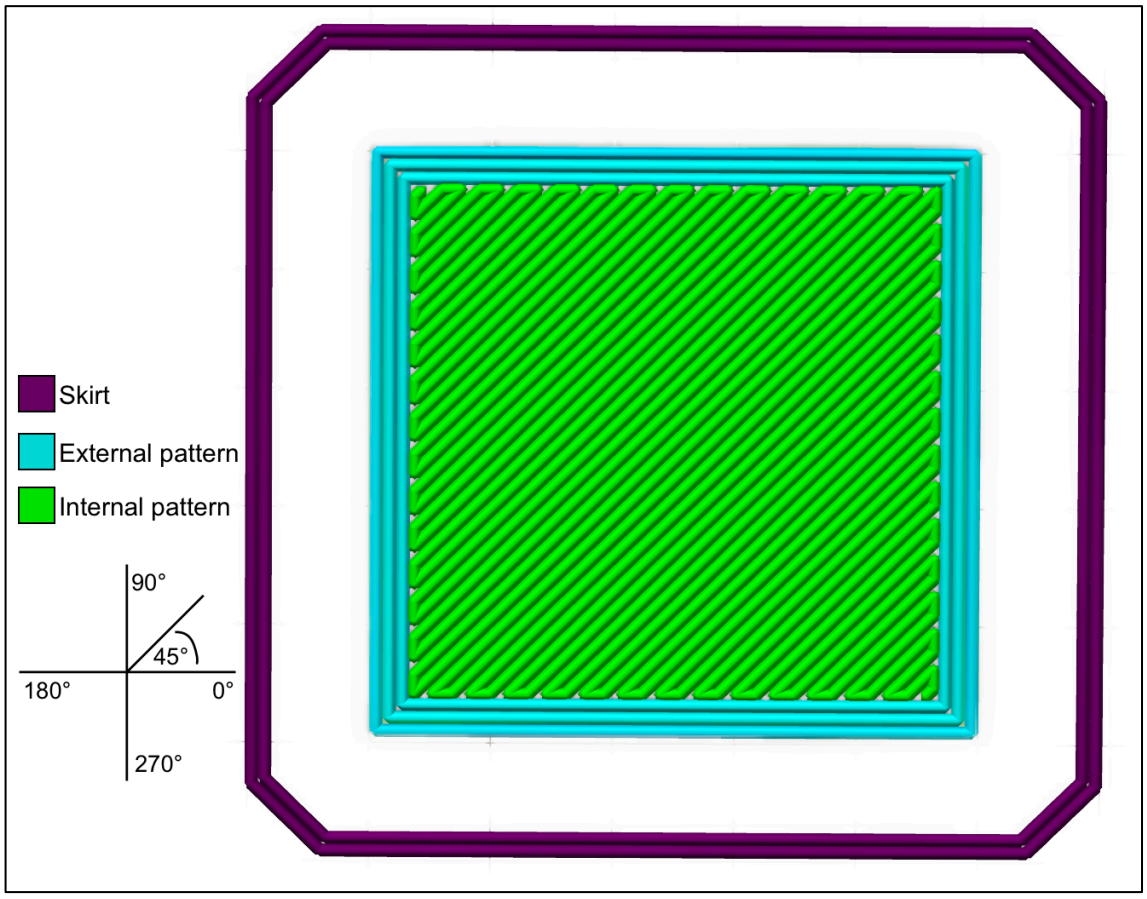

Figure 3 – Workpiece Elements of Fabrication

Given this, another fill behavior was chosen. The primary criterion for segmentation was defined as the identification of the separation point between the external and internal patterns. This point was found through a fill behavior that considers the lengths of the contour and raster lines. Since the fabrication of the contour lines of the external pattern

takes significantly longer than the initial raster lines of the internal pattern, as verified in Figure 4, the algorithm seeks to find, by the duration of the fabrication (in the number of samples), the signal index where there is a significant drop in perceived duration.

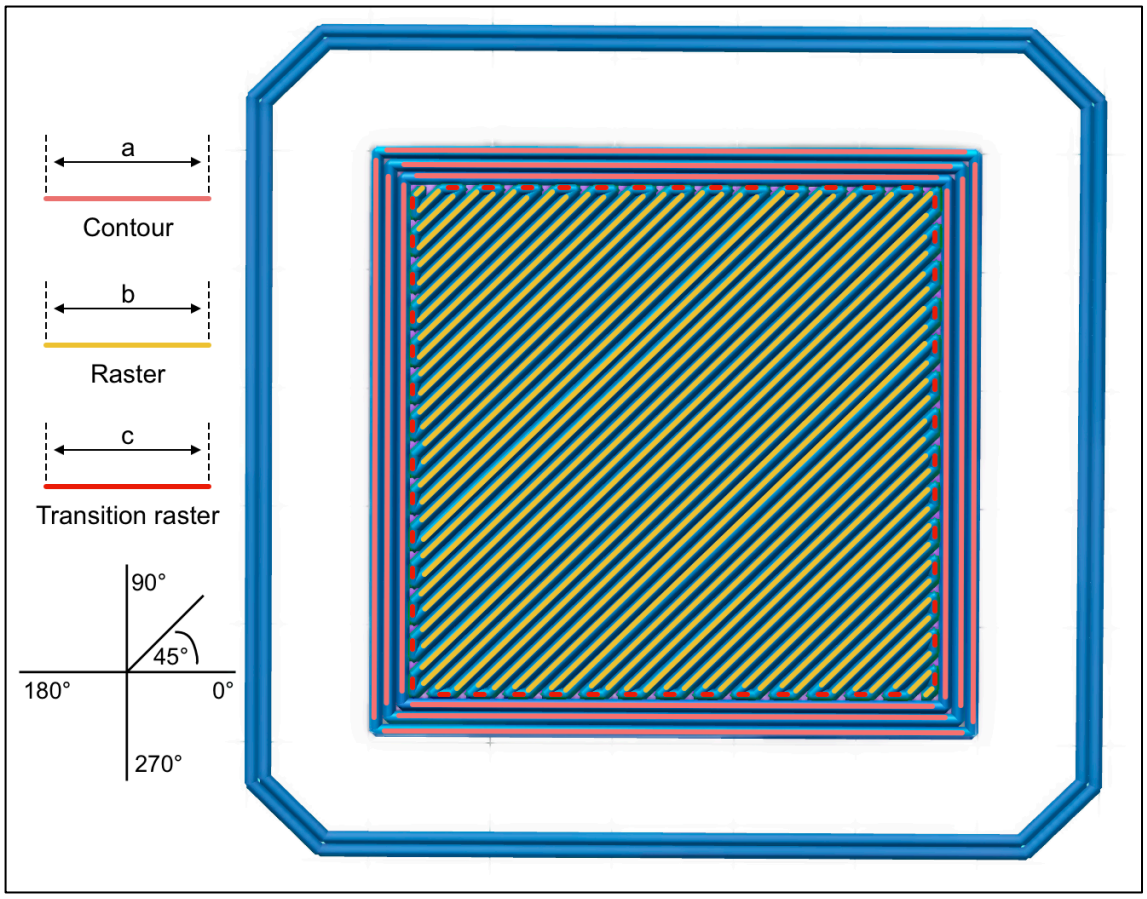

Figure 4 – Workpiece with identifications

Segmentation of the External Pattern

After identifying the separation point between the external and internal patterns, the algorithm aims to segment the external contour lines. For this, the fill behavior of the external pattern was used, as verified in Figure 4:

- The 4 contour lines furthest from the center are the first to be fabricated, have approximately the same length among themselves, and have the longest length among the lines of the external pattern;
- The 4 middle contour lines, fabricated after the 4 contour lines furthest from the center, have approximately the same length among themselves and are shorter than the 4 contour lines furthest from the center;
- The 4 contour lines closest to the center, fabricated after the 4 middle contour lines, have approximately the same length among themselves, have the shortest length among the contour lines, and are the last contour lines before the separation point between fill patterns.

Segmentation of the Internal Pattern

After identifying the separation point between the external and internal patterns and completing the segmentation of the external pattern, the algorithm aims to segment the raster lines and transition raster lines. For this, the fill behavior of the internal pattern was used, as partially verified in Figure 5:

- The raster lines are fabricated at angles of 45° and 225° (180° + 45°);
- The raster lines increase in length from the first fabrication to the midpoint of the workpiece, reversing to a behavior of decreasing length from the midpoint to the last fabrication;
- The raster lines increase/decrease in length by an almost constant value;
- The transition raster lines are fabricated at angles of 90° and 180°;
- The transition raster lines have an almost constant length among themselves;
- A raster line fabrication is always preceded and succeeded by transition raster lines, except for the first and last raster lines;
- The first and last raster lines have a shorter length than the length of the transition raster lines.

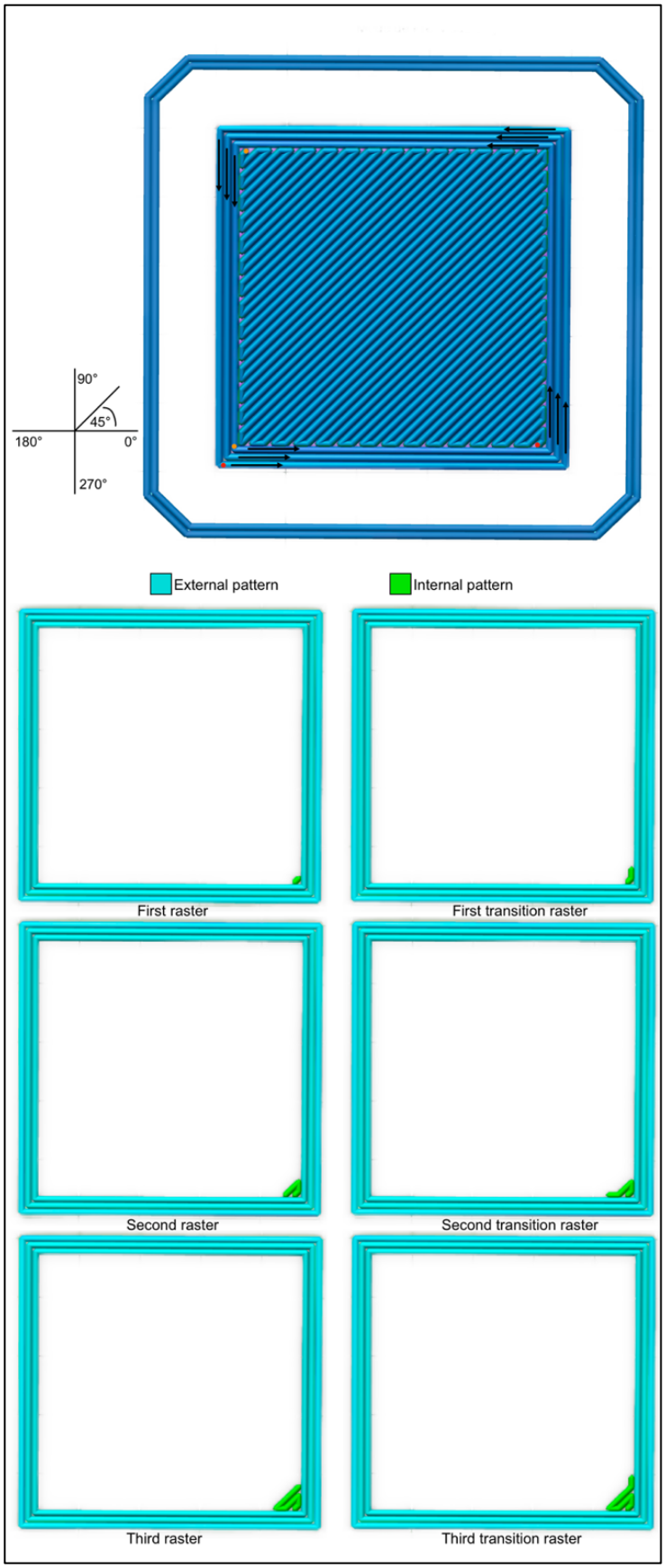

Figure 5 – Internal pattern behaviour

Considerations Based on Fabrication Movements

To carry out the fabrication of the external fill pattern, considering the geometry of the workpiece represented in Figure 4, some repositioning movements without filament deposition are required (Movements Without Deposition – MWD). These MWDs, represented in Figure 6, are highlighted in Table 1, which presents all the movements used for the fabrication of the external fill pattern:

- The first group of MWDs, composed of code lines G 247 and 248, represented in the first row of Figure 6, occurs during the repositioning from the final fabrication point of the skirt to the starting fabrication point of the first contour line (code line G 251).
- The second group of MWDs, composed of code lines G 255, 256, and 258, represented in the second row of Figure 6, occurs during the repositioning from the final fabrication point of the contour lines furthest from the center to the starting fabrication point of the middle contour lines (code line G 260).
- The third group of MWDs, composed of code lines G 264 and 265, represented in the third row of Figure 6, occurs during the repositioning from the final fabrication point of the middle contour lines to the starting fabrication point of the contour lines closest to the center (code line G 267).
- The fourth group of MWDs, composed of code lines G 271 and 273, represented in the fourth row of Figure 6, occurs during the repositioning from the final fabrication point of the contour lines closest to the center to the starting fabrication point of the lines of the internal fill pattern (raster and transition raster).

Table 1 – External printing pattern movements for Test1

Contour printing		Movement Without Deposition (MWD)		Contour Printing sequential to a MWD	
X Coordinate	Y Coordinate	G-code line	Contour line	X direction	Y Direction
215.747	148.508	247	0	0	0
220.297	150.554	248	0	Left	Up
244.747	150.554	251	1	Left	Up
244.747	175.004	252	2	Right	Up
220.297	175.004	253	3	Left	Up
220.297	150.654	254	4	Left	Down
220.297	150.554	255	0	Left	Down
222.297	150.554	256	0	Right	Down
220.847	151.104	258	0	Left	Up
244.197	151.104	260	5	Right	Up
244.197	174.454	261	6	Right	Up
220.847	174.454	262	7	Left	Up
220.847	151.204	263	8	Left	Down
220.847	151.104	264	0	Left	Down
221.397	151.654	265	0	Right	Up
243.647	151.654	267	9	Right	Up
243.647	173.904	268	10	Right	Up
221.397	173.904	269	11	Left	Up
221.397	151.754	270	12	Left	Down
221.397	151.654	271	0	Left	Down
242.865	152.121	273	0	Right	Up

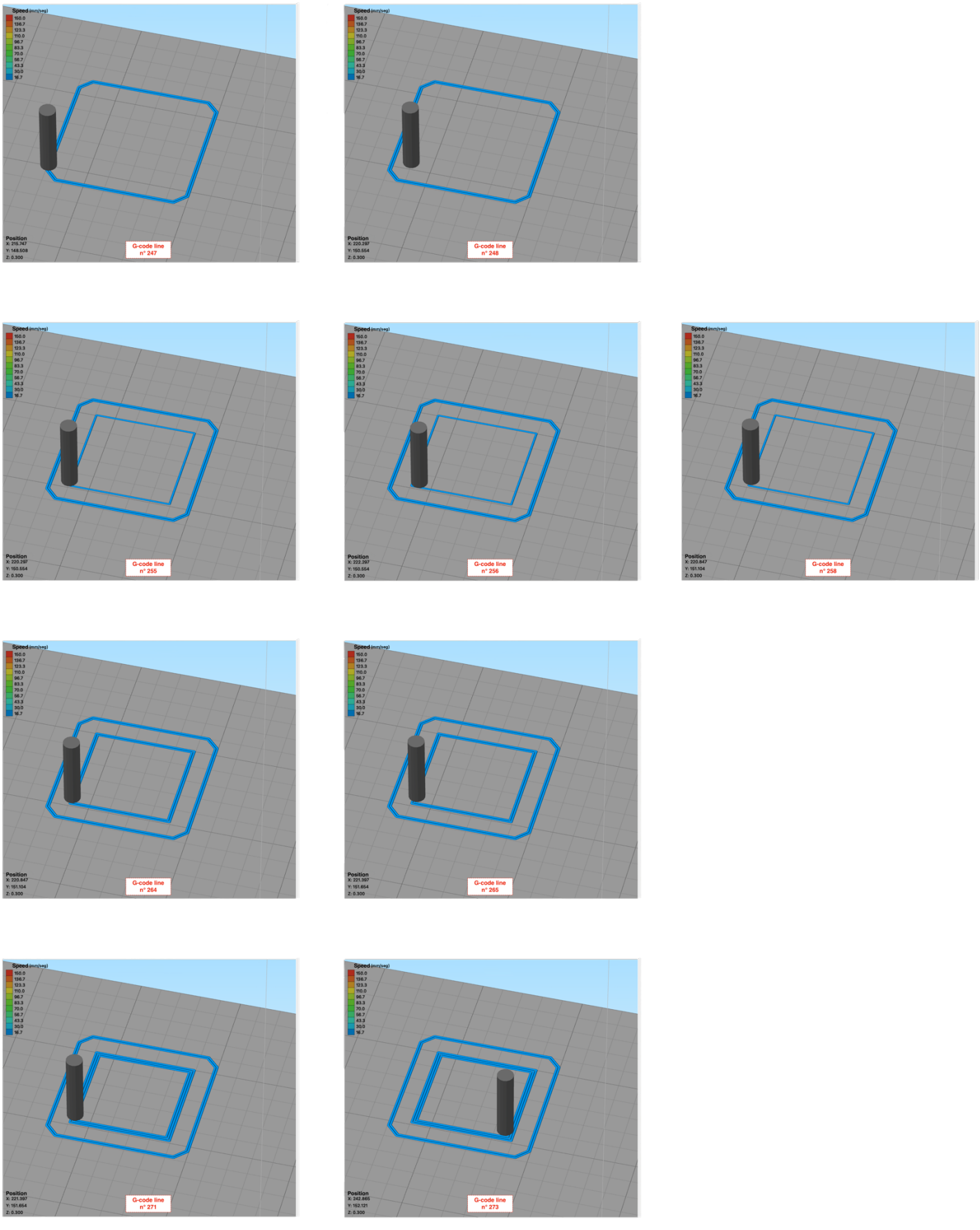

Figure 6 – Movement Without Deposition (MWD) Test1 detailing

Algorithm Segmentation Operation

The fff_segmenter algorithm works with both Matlab and Octave. The segmentation operations work in a similar manner for both software's, with the major distinction being the User Input.

Before using the fff_segmenter algorithm, verify if the following dependencies are installed and available in your Matlab or Octave environment.

Matlab dependencies

-> Signal Processing Toolbox

Octave dependencies

-> Signal Package

-> Control Package

User input

(Matlab) User input

In order to allow for a facilitated users method for input and selection of options, a simple Matlab app Graphical User Interface (GUI) was designed. As illustrated in Figure 7, the app interface comprises several sections and fields to streamline the users input and selection of options for the fff_segmenter algorithm.

At the top, the "Select data" button allows users to navigate between the folders and choose the dataset, in .mat format, for analysis. Next to this button is a field for specifying the sampling frequency (Fs) used in the data acquisition process of the selected dataset, ensuring that the data is accurately processed according to the correct acquisition rate.

Below this, there are fields for "Raw signal identification," "DirX identification", and "DirY identification". These fields are intended for users to input the identifiers for the raw monitoring sensor signal and the directional signals (X and Y), respectively, ensuring proper signal identification.

In the middle section, users can select the segmentation mode for the fff_segmenter algorithm. There are options for choosing either "Points" or "Segments" modes. When "Points" mode is selected, users must also specify the unit for segmentation, choosing between "Number of samples" and "Seconds", as indicated by the radio buttons on the right side.

The lower part of the interface includes toggles for additional functionalities in the fff_segmenter algorithm. Users can enable or disable "Obtain graphical visualization" to obtain a graphical visualization of the segmentation results. If visualization is enabled, there is an option to "Save graphical visualization if obtained," ensuring that the

generated graphs can be stored for further analysis. Another toggle allows users to "Obtain automatic .mat files," facilitating the automatic generation of Matlab files from the segmented data.

Finally, the large switch labeled "Run the segmentation" at the bottom right is used to initiate the segmentation process. When toggled to "On," the main fff_segmenter algorithm resumes execution and also close the app GUI.

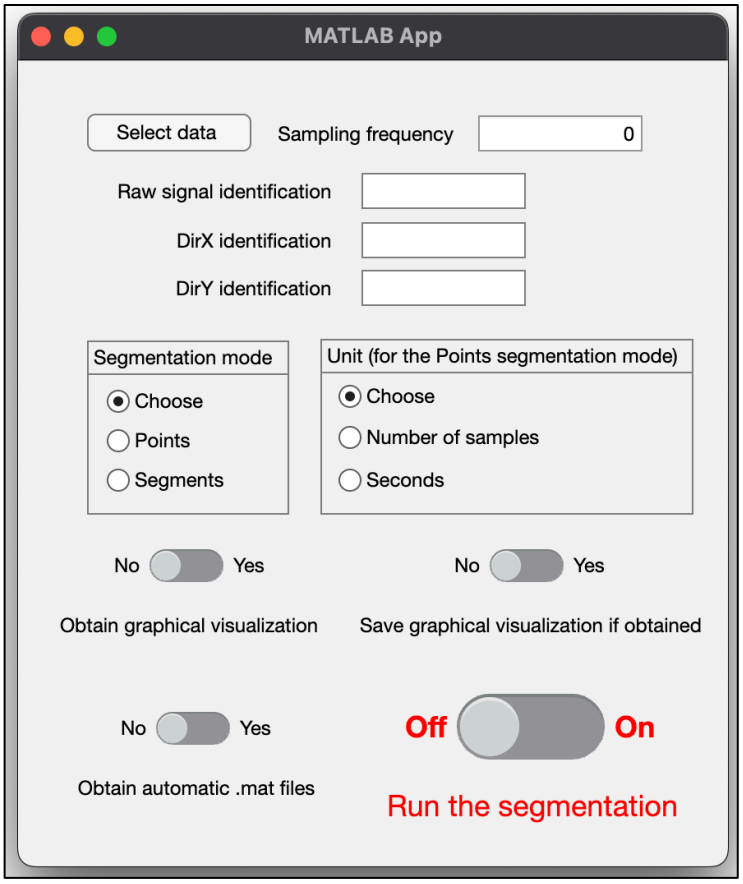

Figure 7 – Matlab GUI for the user inputs

(Octave) User input

In order to allow for a facilitated users method for input and selection of options, a simple Octave User Interface (UI) was designed. As illustrated in Figure 8, the UI interface comprises several sections and fields to streamline the users input and selection of options for the fff_segmenter algorithm.

The fields for "Raw signal identification," "DirX identification", and "DirY identification" are intended for users to input the identifiers for the raw monitoring sensor signal and the directional signals (X and Y), respectively, ensuring proper signal identification.

The field for “Sampling frequency” is intended for users to input the numerical value of the sampling frequency (Fs) used in the data acquisition process of the selected dataset, ensuring that the data is accurately processed according to the correct acquisition rate.

In the same UI, users can select the segmentation mode for the fff_segmenter algorithm. There are options for choosing either "Points" or "Segments" modes. The default segmentation mode is “Points”.

When "Points" mode is selected, users must also specify the unit for segmentation, choosing between "Number of samples" and "Seconds". The default segmentation unit is “Number of samples”.

The lower part of the interface includes fields for additional functionalities in the fff_segmenter algorithm. Another field allows users to opt for "Auto save files (Y/N)", facilitating the automatic generation of Matlab files from the segmented data. The default save results option is “N” for no.

Users can enable or disable "Generate figure (Y/N)" to obtain a graphical visualization of the segmentation results. The default generate figure option is “N” for no.

If visualization is enabled, there is an option to "Save figure" ensuring that the generated graphs can be stored for further analysis. The default save figure option is “N” for no.

With the press of the Enter button on the keyboard the main fff_segmenter algorithm resumes execution and also closes the UI.

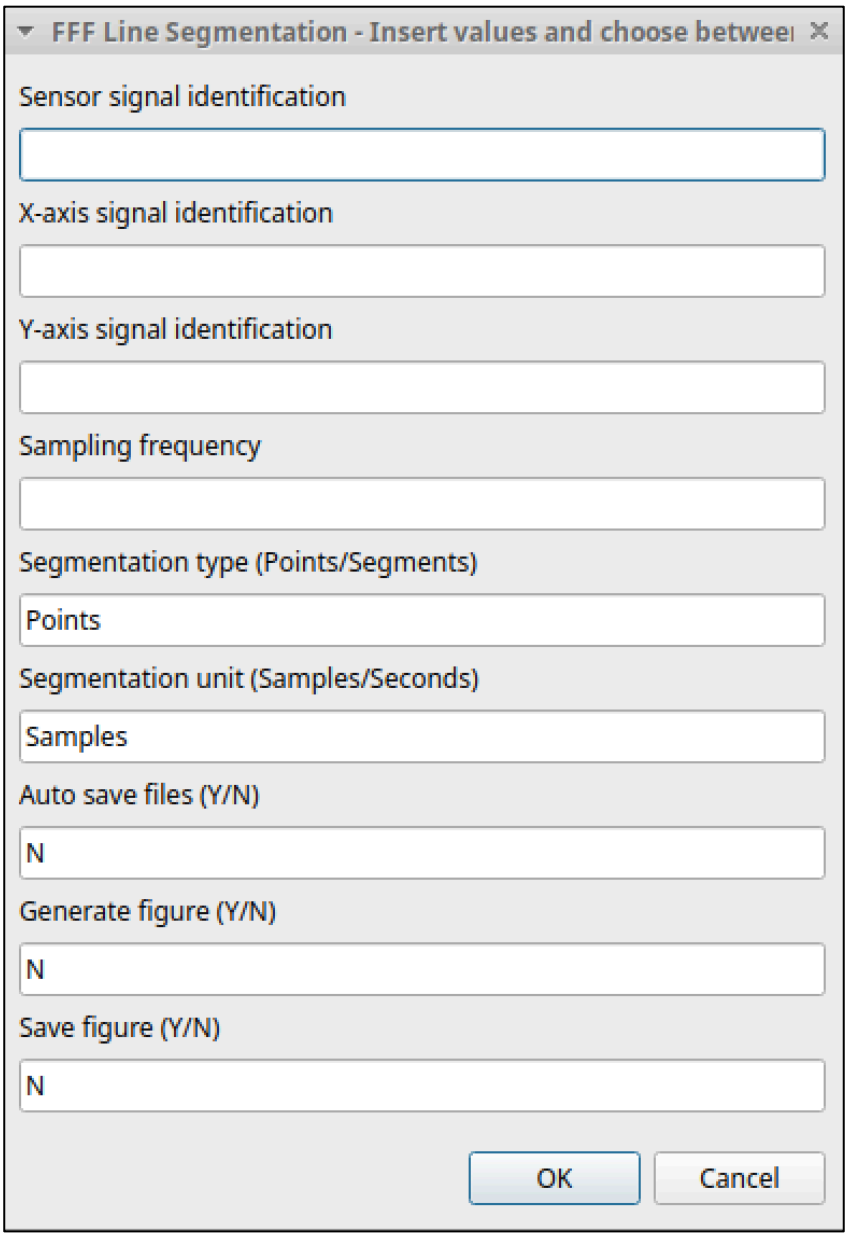

Figure 8 – Octave UI for the user inputs

Attributing Specific Variables

The code displayed on Figure 9 shows how specific variables are attributed based on the geometry of the workpiece and are necessary for the present segmentation logic in the fff_segmenter algorithm. It is important to point out that these variables are specific to the geometry of the workpiece, as well as to the printing parameters/process settings utilized in the preparation of the workpiece g-code (slicing process). In particular, the speed of the first layer fabrication is the most impactful to the segmentation logic variables displayed below.

The slicing process was carried out in the Simplify3D software, by Simplify, with standard printing parameters values, which were provided by the printer manufacturer in

the form of a printing profile. The workpiece's .gcode file is available on the FFF-Line-Segmentation GitHub repository (.../gcodes). The printing parameters values utilized are displayed in g-code comments at the beginning of the workpiece's g-code on 'Settings Summary'.

```
%% Attributing values that are due to the specific part geometry to necessary variables.

% Workpice specific variables
numofRasterLines = 55; % Number of raster lines
numofContourSides = 4; % Number of contour sides
numofContourLoops = 3; % Number of contour loops
numofContourLines = numofContourSides * numofContourLoops; % Number of contour lines

% Variables for the segmentation logic
lowRefTransRaster = Fs/25; % Lower Number of samples reference value for the transition raster
uppRefTransRaster = ceil(Fs/21.505476344); % Upper Number of samples reference value for the transition raster
varDurRaster = floor(Fs/18.1818); % Variation of the duration between raster lines
adpDurTranRaster = floor(Fs/23.8095); % Adopted duration for the transition raster
durationReference = floor(Fs/1.33333); % known duration for the last contour printing
refminimum_sampleValue = floor(Fs/4); % Minimum Number of samples reference for the test_Minvariations subfunction
middleRasterDuration = floor(Fs/0.645161290322581); % Middle raster duration value
```

Figure 9 – Code for attributing specific variables

Workpiece specific variables

The first part of the Figure 9 code assigns values to variables that are specific to the geometry of the workpiece. These variables, such as the number of raster lines (numofRasterLines), contour sides (numofContourSides), and contour loops (numofContourLoops), could be obtained by parsing the workpiece's G-code.

In the given code, numofRasterLines is set to 55, numofContourSides is set to 4, and numofContourLoops is set to 3. The total number of contour lines is then calculated as the product of numofContourSides and numofContourLoops, resulting in 12 contour lines. The value of 3 for the numofContourLoops can be found directly in the workpiece G-code, on line 31 in the comment regarding the 'perimeterOutlines' process parameter, as represented in Figure 10. A script could be designed to parse the G-code and automatically set the value for the numofContourLoops variable.

```
31  ;  perimeterOutlines,3
```

Figure 10 – numofContourLoops value found in the workpiece G-code

To find the numofContourSides value of 4, a script could be designed to parse the G-code and identify the commands for the outer perimeter involving filament deposition (parameter E) with sufficient displacement. This can be evaluated by calculating the variance in the X and Y parameters along the code lines. In the workpiece, an example of these commands is displayed from line 251 to line 254 in regard to the contour lines furthest from the center, as shown in Figure 11.

```
247  ; feature outer perimeter
248  G1 X220.297 Y150.554 F9000
249  G1 E0.0000 F3600
250  G92 E0.0000
251  G1 X244.747 Y150.554 E1.6966 F1200
252  G1 X244.747 Y175.004 E3.3932
253  G1 X220.297 Y175.004 E5.0897
254  G1 X220.297 Y150.654 E6.7794
255  G1 X220.297 Y150.554 F1200
256  G1 X222.297 Y150.554 F1200
```

Figure 11 – Outer perimeter g-code in regard to contour lines furthest from the center

Similarly, to determine the numofRasterLines, a script could be designed to parse the G-code and identify the commands for the 'solid layer' that exhibit higher variability in filament deposition (parameter E) with sufficient displacement. This can also be evaluated by calculating the variance in the X and Y parameters along the code lines. An example of these commands is visible Figure 12, from lines 328 to 329 and lines 330 to 331, representing the raster lines and transitions between raster lines on the middle region of the internal printing pattern.

```
327  G1 X243.179 Y172.658 E29.0264
328  G1 X243.179 Y173.436 E29.0804
329  G1 X221.864 Y152.121 E31.1721
330  G1 X221.864 Y152.899 E31.2261
331  G1 X242.402 Y173.436 E33.2414
332  G1 X241.624 Y173.436 E33.2954
```

Figure 12 – Internal printing pattern g-code in regard to the middle raster lines and transition between raster lines

However, it is important to note that this procedure was not implemented in the current version of the fff_segmenter algorithm due to the lack of necessity. As previously mentioned, the fff_segmenter algorithm was developed with a specific part geometry and consistent process parameters across the available example datasets, making the workpiece-specific variables constant. In its present version, the fff_segmenter algorithm fully meets the signal segmentation needs of the research group that initially developed it.

Further development of the fff_segmenter algorithm by the Free and Open Source (FOSS) community could focus on developing and incorporating these automatic scripts that parse the G-code of various workpieces to obtain the appropriate values for the

workpiece-specific variables. This would enhance the algorithm's versatility and applicability to different geometries and printing conditions.

Segmentation logic variables

The next section of the Figure 9 code defines variables essential for the segmentation logic. These variables rely on the sampling frequency (Fs). In the available example datasets, the sampling frequency used for acquiring the signals was always 200 kS/s. However, the code uses the Fs variable instead of the direct numerical value of 200e3 in the mathematical calculations for these variables. This approach allows for flexibility in altering the sampling frequency if the same workpiece and process parameters are used in future signal acquisition processes.

Another major factor for the segmentation logic variables is the speed value for the first layer fabrication. The numerical values utilized in the mathematical calculations below were obtained considering a fixed value of the first layer printing speed for the fabrications from which the three available datasets originated. As represented in Figure 75, the used default printing speed is 20 mm/s, and the first layer speed percentage regarding the default printing speed is 100%, as verified in Figure 68. Therefore, the printing speed was 20 mm/s throughout the fabrication of the workpiece.

The variables for the segmentation logic include:

- lowRefTransRaster: Lower number of samples reference value for the transition raster, calculated as Fs/25.
    - This variable is used to define the low range reference value for the transition between raster lines duration. Considering the Fs of 200e3, the value calculated by 200e3/25 with the round command is 8,000 samples, which was found to be adequate as a lower range reference for the transition between raster lines with the used printing speed of 20 mm/s.
- uppRefTransRaster: Upper number of samples reference value for the transition raster, calculated as round(Fs/22.2222,0).
    - This variable is used to define the upper range reference value for the transition between raster lines duration. Considering the Fs of 200e3, the value calculated by 200e3/22.2222 with the round command is 9,000 samples, which was found to be adequate as an upper range reference for the transition between raster lines with the used printing speed of 20 mm/s.
- varDurRaster: Variation of the duration between raster lines, calculated as round(Fs/18.1818,0).

    - This variable is used to define the default value for the variation between raster lines durations. Considering the Fs of 200e3, the value calculated by 200e3/18.1818 with the round command is 11,000 samples, which was found to be adequate as a default value for the variation between raster lines durations with the used printing speed of 20 mm/s.
- adpDurTranRaster: Adopted duration for the transition raster, calculated as round(Fs/23.8095,0).
    - This variable is used to define the default reference value for the transition between raster lines duration. Considering the Fs of 200e3, the value calculated by 200e3/23.8095 with the round command is 8,300 samples, which was found to be adequate as a default reference value for the transition between raster lines with the used printing speed of 20 mm/s.
- durationReference: Known duration for the last contour printing, calculated as round(Fs/1.33333,0).
    - This variable provides a reference duration for the final contour line, assisting in determining the end of the external printing pattern. Considering the Fs of 200e3, the value calculated by 200e3/1.33333 with the round command is 150,000 samples, which was found to be adequate as a duration reference for the last contour printing with the used printing speed of 20 mm/s.
- refminimum_sampleValue: Minimum number of samples reference for the test_Minvariations subfunction, calculated as round(Fs/4,0).
    - This variable sets the minimum number of samples required for the test_Minvariations subfunction to validate specific signal variations. Considering the Fs of 200e3, the value calculated by 200e3/4 with the round command is 50,000 samples, which was found to be adequate as a minimum number of samples required for the test_Minvariations subfunction with the used printing speed of 20 mm/s.
- middleRasterDuration: Middle raster duration value, calculated as round(Fs/0.645161290322581,0).
    - This variable defines a middle duration for the raster lines, used as a reference in the segmentation process. Considering the Fs of 200e3, the value calculated by 200e3/0.645161290322581 with the round command is 310,000 samples, which was found to be adequate as a middle duration for the raster lines with the used printing speed of 20 mm/s.

Further development of the fff_segmenter algorithm could focus on creating and incorporating automatic scripts to parse the G-code of various workpieces. As an example, one of these scripts could dynamically extract the printing speed, identify key

durations for the segmentation logic, calculate the duration for all raster lines printing, and determine the maximum duration to set the middleRasterDuration, which corresponds to the longest raster line due to the part geometry. This is only one of the calculations that could be made with information extracted from the G-code. The FOSS community could enhance the algorithm's versatility and applicability to different printing speeds, geometries, and printing conditions by developing and incorporating these automatic scripts to obtain the appropriate values for the segmentation logic variables.

Pre-processing

One of the necessary pre-processing steps for the segmentation algorithm is the correction of amplitude fluctuations in the control signals for the stepper motors' direction. Such fluctuations were observed in all control signals for direction, in both the Test1 dataset (Figure 31) and the Test2 dataset (Figure 43). However, it is important to note that more significant amplitude fluctuations were observed in the directional signal for the X stepper motor in the Test1 dataset. This may have occurred due to issues with the data acquisition system for this dataset. To correct these incorrect fluctuations between 0V and 5V, the code shown in Figure 13 was applied, which uses a 2V threshold to define values of 0V or 5V.

```
163  dirXAdjusted = zeros(size(dirX));
164  dirYAdjusted = zeros(size(dirY));
165
166  % 1° Normalize the X and Y step motor control signals between 0V and 5V
167  dirXAdjusted(dirX > 2) = 5;
168  dirXAdjusted(dirX < 2) = 0;
169  dirYAdjusted(dirY < 2) = 0;
170  dirYAdjusted(dirY > 2) = 5;
```

Figure 13 – Code for normalize the X and Y step motor control signals between 0V and 5V

As a result, directional signals were obtained that displayed values of either 0V or 5V, as shown in Figure 32 (Test1 dataset) and Figure 44 (Test2 dataset).

To facilitate the representation of acoustic sensor signals alongside the control signals for direction, the code shown in Figure 14 was applied. This code employs a sub-function called "Normaliz3r" that normalizes data vectors between binary values (0 and 1) or between -1 and 1.

```
% 2° Normalize the acoustic signal, and the X and Y step motor control
% signals between -1 and 1 (acoustic signal) and 0 & 1 (step motor signals)

sensorSignalNormalized = Normaliz3r(sensorSignal);
dirYAdjustedNormalized = Normaliz3r(dirYAdjusted);
dirXAdjustedNormalized = Normaliz3r(dirXAdjusted);
```

Figure 14 – Code for normalize the X and Y step motor control signals between 0 and 1

As a result, directional signals were obtained that displayed values of either 0 or 1, as shown in Figure 33 (Test1 dataset) and Figure 45 (Test2 dataset).

Another necessary pre-processing step is the verification of the occurrence of an issue concerning the number of variations (directional changes) in the stepper motor signals. As can be seen in Figure 34, specific periods of the directional change signals display a number of variations incompatible with the geometry of the workpiece. This is reflected in the calculation of the duration, which uses the start and end points of the workpiece's manufacturing to calculate the duration of each line, as shown in Figure 35. It is believed that this issue occurs only in the directional signal of the Test1 dataset, given that the directional signal of the Test2 dataset, shown in Figure 46, does not exhibit this phenomenon.

To verify this occurrence, the code shown in Figure 15 was inserted. This code calls the sub-function "test_Minvariations" and returns a 'resultProblem' variable as either true or false.

```
% 3° Test for the occurence of the variation problem
resultProblem = test_Minvariations(sensorSignalNormalized, dirXAdjustedNormalized,...
    dirYAdjustedNormalized, lowRefTransRaster);
% \ 3°
```

Figure 15 – Code to test for the occurrence of the variation problem

External Pattern Segmentation

The first step for external pattern segmentation is to obtain the duration vector. This is done because, as the printing speed was constant throughout the fabrication of the part, various analyses can be conducted. These analyses relate the length of the fabrication lines (contour, raster, and raster transition) shown in Figure 4, with the duration of these fabrication lines in terms of sample numbers.

The duration is obtained from the code represented in Figure 16. This code calls the sub-function obtainDuration and returns the matrix Duration. The first column pertains to the value of the duration itself, the second column is the index related to the endpoint of the fabrication line, and the third column is the index related to the starting point of the fabrication line.

```
191  % 4° Obtain the duration vector
192  Duration = obtainDuration(sensorSignalNormalized, dirXAdjustedNormalized,...
193      dirYAdjustedNormalized, resultProblem, refminimum_sampleValue);
```

Figure 16 – Code to obtain the duration matrix

The first column of the Duration is represented in Figure 31 (Test1 dataset) and in Figure 43 (Test2 dataset). Based on these duration values, the separation point between the external and internal fabrication patterns will be identified. To find this separation point, the entire first column of Duration is traversed until a known duration value is found (‘durationReference’, which is the duration of line 12 fabrication). Then, the positions related to the start of the fabrication for lines 10, 11, and 12 are stored. It is known that these three should be very similar, so a test is run on lines 10 and 11. If they are within the same interval, the end of the external pattern is identified. Otherwise, the search for this pattern in the duration values continues.

```
196  % 5° Find the separation point
197
198  for i = 1:length(Duration(:,1))
199
200      resultSep = determin_separation_point(Duration, i, durationReference);
201
202      if resultSep == true
203          patternVariationPoint = Duration(i,2);
204          break;
205      else
206          clear resultSep
207      end
208  end
209  % \ 5°
```

Figure 17 – Code to find the separation point

Having identified the separation point, the indices related to the start of the contour lines are found. These indices are stored in the variable ‘externalLines’. With these starting indices, a temporary matrix related to the contour lines, named ‘indexContourTemp’, is obtained. The first column pertains to the duration value of the contour line, the second column is the index related to the endpoint of the contour line, and the third column is the index related to the starting point of the contour line.

Obtaining this temporary matrix is necessary because, at a later stage, it will be crucial to identify and separate the periods related to movements without deposition (MWD) from the external pattern. The code displayed in Figure 18 represents the creation of this temporary matrix.

```
% 6° Contour lines obtaining
separationIndex = find(Duration(:,3) == patternVariationPoint);
externalLines = zeros(1,13);
duration3Length = 1;

for i = 1:13
    externalLines(1,i) = Duration(separationIndex-13+i,3);
end
for i = 1:length(externalLines)-1
    indexCountourTemp(duration3Length,1) = externalLines(i+1) - externalLines(i);
    indexCountourTemp(duration3Length,2) = externalLines(i);
    indexCountourTemp(duration3Length,3) = externalLines(i+1);
    duration3Length = duration3Length+1;
end
% \ 6 °
```

Figure 18 – Code to obtain the temporary contour lines matrix

Having obtained the temporary matrix for the contour lines, the next step is to obtain the periods related to the MWDs and, with these, get the correct contour lines matrix. To find the periods related to the MWDs, the information presented in Figure 6 and Table 1, as discussed earlier, should be referred to.

After analyses correlating the external pattern fabrication phenomena with the obtained duration graph, it was observed that the periods related to the MWDs are added to the duration of the subsequent contour lines following an MWD occurrence. This can be confirmed by observing Figure 19 in more detail, which represents the debug point POI-6.

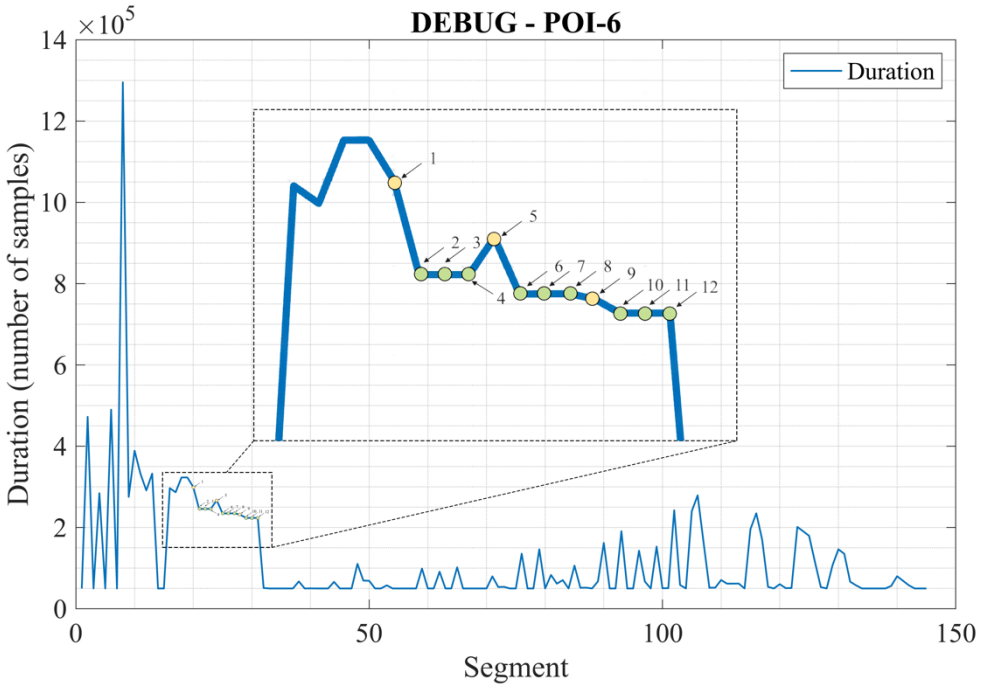

Figure 19 – Test1 Debug POI-6 MWD's detailing

This phenomenon was observed for the Test1 and Test2 datasets. However, for the Test3 dataset, an inverse behavior occurred, where the MWD decreased the duration value of the directly subsequent contour line. This can be observed in Figure 60, between segments 18 and 19. The exact reason for this is undetermined, but it is suspected that there was some type of interference in the data acquisition system during the test for the Test3 dataset, which led to the absence of some change in either the X or Y direction. The code displayed in Figure 20 represents the obtaining of the MWD periods and subsequently, the correct contour lines matrix.

```
% 7° Repositioning evaluation and contour lines correction

% reposition 1
meanCountourGroup1 = round(mean([indexCountourTemp(numofContourSides-2,1) indexCountourTemp(numofContourSides-1,1)...
    indexCountourTemp(numofContourSides,1)]),0);

adjustedContourLine1(1,1) = meanCountourGroup1;
adjustedContourLine1(1,3) = indexCountourTemp(1,3);
adjustedContourLine1(1,2) = indexCountourTemp(1,3) - meanCountourGroup1;

if indexCountourTemp(1,1) - meanCountourGroup1 < 1
    repoToContour1(1,1) = abs(indexCountourTemp(1,1) - meanCountourGroup1);
    repoToContour1(1,3) = adjustedContourLine1(1,2);
    repoToContour1(1,2) = repoToContour1(1,3)...
        - repoToContour1(1,1);
else
    repoToContour1(1,1) = indexCountourTemp(1,1) - meanCountourGroup1;
    repoToContour1(1,3) = adjustedContourLine1(1,2);
    repoToContour1(1,2) = indexCountourTemp(1,2);
end

% reposition 2
meanContourGroup2 = round(mean([indexCountourTemp(6,1) indexCountourTemp(7,1)...
    indexCountourTemp(8,1)]),0);

adjustedContourLine2(1,1) = meanContourGroup2;
adjustedContourLine2(1,3) = indexCountourTemp(5,3);
adjustedContourLine2(1,2) = indexCountourTemp(5,3) - meanContourGroup2;

repoToContour2(1,1) = indexCountourTemp(5,1) - meanContourGroup2;
repoToContour2(1,3) = adjustedContourLine2(1,2);
repoToContour2(1,2) = indexCountourTemp(5,2);

% reposition 3
meanContourGroup3 = round(mean([indexCountourTemp(10,1) indexCountourTemp(11,1)...
    indexCountourTemp(numofContourLines,1)]),0);

adjustedContourLine3(1,1) = meanContourGroup3;
adjustedContourLine3(1,3) = indexCountourTemp(9,3);
adjustedContourLine3(1,2) = indexCountourTemp(9,3) - meanContourGroup3;

repoToContour3(1,1) = indexCountourTemp(9,1) - meanContourGroup3;
repoToContour3(1,3) = adjustedContourLine3(1,2);
repoToContour3(1,2) = indexCountourTemp(9,2);

indexContour = indexCountourTemp;
indexContour(1,:) = adjustedContourLine1;
indexContour(5,:) = adjustedContourLine2;
indexContour(9,:) = adjustedContourLine3;

contourRepositions = [repoToContour1; repoToContour2; repoToContour3];
% \ 7°
```

Figure 20 – Code to obtain the correct contour lines matrix and MWD matrix

To obtain the period related to the first group of MWDs, the average between the duration values of contour lines 2, 3, and 4 is calculated. This average is stored in the variable ‘meanContourGroup1’. Afterward, for the cases observed for the Test1 and Test2 datasets, the value of contour line 1 is subtracted by the obtained average. The result of this operation represents the correct duration of contour line 1, stored in column 3 of the matrix ‘adjustedContourLine1’. This duration is then related to the starting index of contour line 2, to find the appropriate index for the start of contour line 1. The remainder represents the duration of the first group of MWDs, and this duration is related to the appropriate index for the start of contour line 1, to obtain the index indicating the start of this MWD group. For the case observed in the Test3 dataset, the only difference in the code (the “else” situation on line 16 of Figure 20) is in how the average of contour lines 2, 3, and 4 is related to the duration of contour line 1.

Concerning the second and third groups of MWDs, for all three datasets observed, the duration was obtained using the same procedure described as functional for the first MWD group in the Test1 and Test2 datasets, with differences regarding which lines were employed for obtaining the average.

As a result, the matrices indexContour and contourRepositions were obtained. In these matrices, the first column pertains to the duration value of the contour line or MWD period, the second column is the index related to the endpoint of the contour line or MWD period, and the third column is the index related to the starting point of the contour line or MWD period. It's important to note that the contourRepositions matrix is not complete. As verified in Figure 4 and Table 1, there would be a fourth group of MWDs, related to the transition period between the fabrication of the last contour line and the first raster line. To determine this fourth MWD group, and consequently complete the contourRepositions matrix, it is necessary to proceed to the segmentation of the internal pattern, which will allow determining the index related to the start of the first raster line's fabrication.

Internal Pattern Segmentation

The first step in segmenting the internal pattern is to obtain the duration vector. This is done because, as the printing speed was constant throughout the entire fabrication of the part, various analyses can be performed relating the length of the fabrication lines (contour, raster, and raster transition), as shown in Figure 2, to the duration of these fabrication lines in the number of samples.

It's worth noting that, unlike what was done for the external pattern, the issue depicted in Figure 34 will not be checked. This is because the internal pattern has very small raster fill lines, as seen in Figure 3 for the first raster line.

Thus, when calling the sub-function obtainDuration, the code shown in Figure 21 uses the variable resultProblem with a predefined value as false and returns the Duration matrix. The first column is related to the duration value itself, the second column is the index corresponding to the end point of the fabrication line, and the third column is the index corresponding to the starting point of the fabrication line.

```
282  % 8° Obtaining first duration matrix for the internal pattern
283  % There is no need to check for the problem of too many consecutive transition lines here
284  % since there are very small line segments in the internal pattern. On the contrary,
285  % this could cause some issues. So the result_problem value is set to false.
286  resultProblem = false;
287
288  % Obtaining duration matrix
289  Duration = obtainDuration(sensorSignalNormalized, ...
290      dirXAdjustedNormalized,...
291      dirYAdjustedNormalized, ...
292      resultProblem, refminimum_sampleValue);
293      % \ 8°
```

Figure 21 – Code to obtain the first duration matrix for the internal pattern

Upon analyzing the duration values obtained from the Duration matrix, as shown in Figure 37, it was observed that there are significantly more fabrication segments than expected raster lines and raster transitions, as per the geometry of the part represented in Figure 2. Analysis in Figure 37 reveals that there are many segments with very short duration. Therefore, it is necessary to apply a threshold filter so that only internal pattern fabrication lines exceeding the shortest expected duration for the internal geometry of the part are represented.

The chosen threshold was 8,000 samples, given that it is known that the raster transition lines have approximately the same length, as depicted in Figure 2, and this length, in a signal acquired at a sampling frequency of 200kS/s, represents a duration between 8,100 and 9,000 samples.

Nevertheless, it's important to highlight that there are two lines in the internal pattern with a duration less than 8,000 samples—the first and the last raster lines. However, at

this point, the threshold filter will be applied even though segments related to the first and last raster lines will be eliminated. These lines will be obtained later in this analysis, following a manufacturing logic between raster lines and raster transitions.

The code displayed in Figure 22 shows how to obtain the Duration2 matrix with the application of the predetermined threshold. The first column relates to the duration value itself, the second column is the index corresponding to the end point of the fabrication line, and the third column is the index corresponding to the starting point of the fabrication line.

```
295  % 9° Obtaining second duration matrix for the internal pattern
296
297  % Filtering the duration matrix to remove the small line segments
298  indexValuesAbvRef = 1;
299  Duration2 = zeros(3);
300  alterationDuration = lowRefTransRaster;
301  for i = 1:length(Duration(:,1))
302      if Duration(i,1) > alterationDuration
303          Duration2(indexValuesAbvRef,:) = Duration(i,:);
304          indexValuesAbvRef = indexValuesAbvRef + 1;
305      end
306  end
307
308  clear Duration
309      % \ 9°
```

Figure 22 – Code to obtain the second duration matrix for the internal pattern

After obtaining the Duration2 matrix, the segmentation of raster lines and raster transition lines can begin. The segmentation logic will be based on the previously mentioned internal pattern fabrication behavior (Internal Pattern Segmentation). The discussed fabrication behavior is partially verified in Figure 23, with the exception of regions with abnormal behavior (abnormal).

In the abnormal regions identified in Figure 23, the following fabrication behaviors are observed:

- Abnormal 1 - There are eight fabrication segments (identified by green circles) between raster transition lines (identified by red circles), where transitionRaster -> rasterLineCount sets should be observed, with the rasterLineCount adding a value of approximately 11,000 samples in duration compared to the directly preceding rasterLineCount. This abnormal region is preceded and followed by normal regions;

- Abnormal 2 - There are five fabrication segments (identified by green circles) between raster transition lines (identified by red circles), where transitionRaster -> rasterLineCount sets should be observed, with the rasterLineCount adding a value of approximately 11,000 samples in duration compared to the directly preceding rasterLineCount. This abnormal region is preceded and followed by normal regions;
- Abnormal 3 - There are three fabrication segments (identified by green circles) between raster transition lines (identified by red circles), where transitionRaster -> rasterLineCount sets should be observed, with the rasterLineCount adding a value of approximately 11,000 samples in duration compared to the directly preceding rasterLineCount. This abnormal region is preceded by a normal region and followed by abnormal region 4;
- Abnormal 4 - There are three fabrication segments (identified by green circles) between raster transition lines (identified by red circles), where transitionRaster -> rasterLineCount sets should be observed, with the rasterLineCount adding a value of approximately 11,000 samples in duration compared to the directly preceding rasterLineCount. This abnormal region is preceded by abnormal region 3 and followed by a normal region.

It should be noted that such abnormal behavior regions also appear in the Test2 and Test3 datasets. However, they relate to different segments than those observed for the Test1 dataset. Thus, it becomes evident that such abnormal regions occur in the internal pattern fabrication in all evaluated datasets. This characteristic could originate from the data acquisition system or the extruder's control system itself. To determine the source more accurately, more specific tests are required.

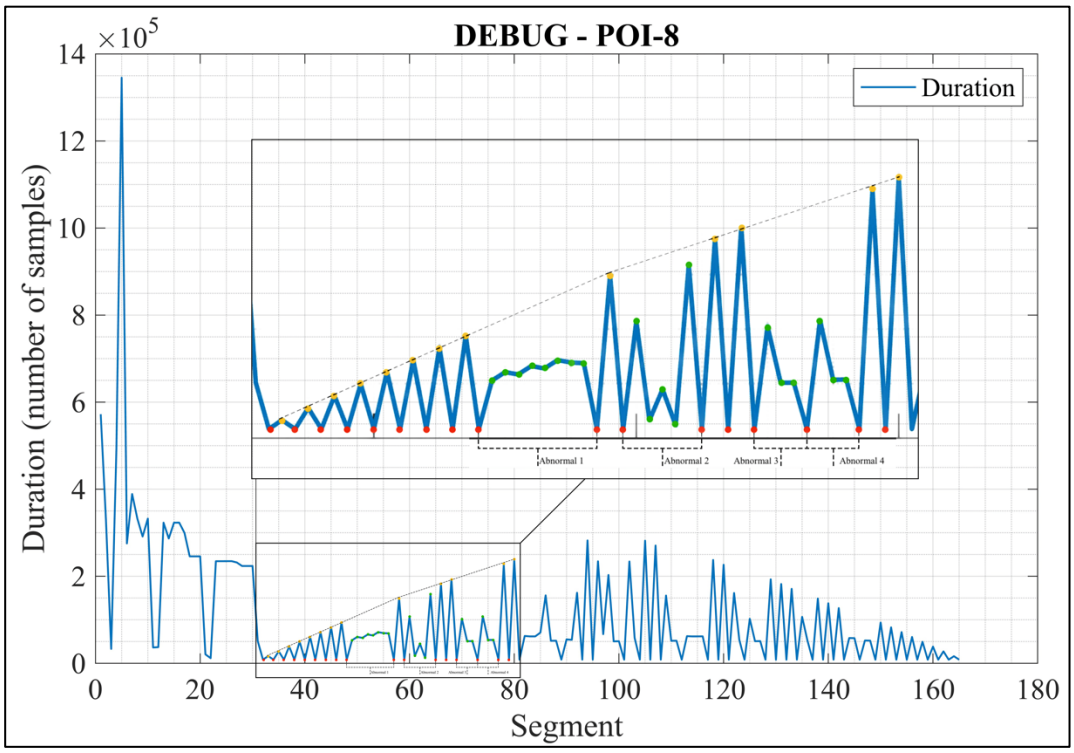

Figure 23 – Test1 Debug POI-8 MWD's detailing

With this understanding, the code shown in Figure 24 represents the acquisition of the Duration3 matrix. Initially, the code locates in the Duration2 matrix the indices corresponding to rows that have fewer than 9,000 samples and saves them in the vector linesUdrUppVal. This vector will serve as a guide for the iteration of the main "for" loop in the algorithm and as a basis for identifying regions of normal and abnormal behavior. Moreover, the code uses the variable middleRasterDuration with the value of 310,000 samples, equivalent to the duration of manufacturing the central raster line with a sampling frequency of 200kS/s.

In each iteration of the "for" loop, sequential indices corresponding to the transition raster lines are initially stored. This is done to allow the diagnosis of a normal situation (transitionRaster -> rasterLineCount -> transitionRaster), or an abnormal situation (such as those observed in Figure 23). This diagnosis is carried out through the subfunction detectAnom.

Subsequently, the raster duration values directly preceding and sequentially preceding the current point in the Duration3 matrix are stored. This allows identification of whether the manufacturing of the internal pattern has reached the raster line immediately subsequent to the middle of the filled area (at which point the "for" loop will end) and provides a reference relative to the duration of the last suitable raster line, necessary for correcting behavior observed in abnormal situations.

```
% 10° Obtaining third duration matrix for the internal pattern

linesUdrUppVal = find(Duration2(:,1) < uppRefTransRaster);
Duration3 = zeros(3);
previousPeak = 0;
previousPeak2 = 0;

for i = 1:length (linesUdrUppVal)
    if i == length (linesUdrUppVal)
        break;
    end
    initialIndex = linesUdrUppVal(i);
    finalIndex = linesUdrUppVal(i+1);
    result = detectAnom(initialIndex, finalIndex);
    previousPeak2 = previousPeak;
    previousPeak = Duration3(end-1,1);
    if  previousPeak < previousPeak2 % Change of behaviour from increasing to decreasing
        break;
    end
    if  previousPeak > middleRasterDuration % Is larger than the middle raster duration,
        % found the end of the increasing behaviour
        break;
    end
    if result == true % normal situation
        tempDuration= isNormal( ...
            Duration2, ...
            initialIndex, ...
            finalIndex, ...
            1);
        if Duration3(1,1) == 0
            Duration3 = tempDuration;
        else
            Duration3 = [Duration3(1:end-1,:); tempDuration];
        end
        clear tempDuration
    else % abnormal situation
        tempDuration = isAbnormal( ...
            Duration2, ...
            initialIndex, ...
            finalIndex, ...
            previousPeak, varDurRaster, adpDurTranRaster);
        Duration3 = [Duration3(1:end-1,:); tempDuration];
        clear tempDuration
    end
end

clear Duration2
% \ 10°
```

Figure 24 – Code to obtain the third duration matrix for the internal pattern

Next, the algorithm will proceed to acquire a temporary matrix, named tempDuration, following the result of the normal or abnormal situation test. If the test result is true, it indicates a normal situation in which the algorithm will proceed to obtain the tempDuration matrix by calling the subfunction isNormal. On the other hand, if the test result is false, it indicates an abnormal situation, in which the algorithm will proceed to obtain the tempDuration matrix by calling the subfunction isAbnormal.

The result obtained from the tempDuration matrix is then added to the Duration3 matrix. The "for" loop continues until one of the two termination situations occurs:

- If the raster duration value directly preceding the current point in the Duration3 matrix is less than the sequentially preceding raster duration value. This indicates that internal manufacturing has transitioned from increasing to decreasing behavior;
- If the raster duration value directly preceding the current point in the Duration3 matrix is greater than the duration value contained in the variable middleRasterDuration.

For both termination situations, the "for" loop is interrupted upon reaching the midpoint of the internal fill pattern manufacturing. The termination conditions may seem redundant, but they allow the segmentation algorithm to handle and correct any deviations should an abnormal situation occur in the region related to the manufacturing of the middle of the internal pattern.

In the obtained Duration3 matrix, the first column relates to the duration value itself, the second column is the index relative to the end point of the manufacturing line, and the third column is the index relative to the starting point of the manufacturing line. Figure 40 represents the duration values corresponding to raster lines and transition raster lines. It is possible to observe that segment 55 represents the manufacturing of the middle raster line, and segment 57 represents the manufacturing of the first raster line where the behavior changes from increasing to decreasing.

After this, the code shown in Figure 25 represents the adjustment of the Duration3 matrix, leading to the creation of the Duration3_v2 matrix. The new matrix was obtained to eliminate the segments with indices higher than the segment representing the fabrication of the middle raster line. This is done to have an internal fill matrix up to the fabrication of the middle line, so as to enable the mirroring of durations in index retrieval. Figure 40 displays the duration values related to raster lines and transition raster lines of the part, providing an opportune moment for applying the duration mirroring procedure in index retrieval.

```
360  % 11° Obtaining fourth duration matrix for the internal pattern
361  indexPreviousPeak2 = find (Duration3(:,1) == previousPeak2);
362  Duration3_v2 = Duration3(1:end-(end-indexPreviousPeak2)+1,:);
363  % \ 11°
```

Figure 25 – Code to obtain the fourth duration matrix for the internal pattern

The code shown in Figure 26 describes the procedure for duration mirroring in index retrieval. This procedure is based on the internal pattern filling characteristics represented in Figure 3 and discussed earlier. In summary, the procedure allows for the automatic retrieval of indices related to raster lines and transition raster lines that make

up the fabrication from the midpoint to the end of the internal fill (characterized by a reduction in duration between the raster lines). These duration values are stored in the Duration4 matrix.

In the obtained Duration4 matrix, the first column relates to the duration value itself, the second column is the index related to the endpoint of the fabrication line, and the third column is the index related to the starting point of the fabrication line. Figure 41 displays the duration values for the external fill pattern fabrications, excluding the first and last raster lines. This can be observed by analyzing Figure 41, noticing that the first and last durations represent transition raster segments. Therefore, it is necessary to obtain a sixth matrix with the missing lines.

```
% 12° Obtaining fifth duration matrix for the internal pattern
% Raster area mirroring

duration3Length = length(Duration3_v2(:,1));
Duration4 = Duration3_v2;

for i = 1:length(Duration3_v2(:,1))-2
    if i == 1
        Duration4(duration3Length,1) = Duration3_v2(length(Duration3_v2(:,1))-i-1,1);
        Duration4(duration3Length,3) = Duration3_v2(length(Duration3_v2(:,1))-i,2);
        Duration4(duration3Length,2) = Duration4(duration3Length,3) + Duration4(duration3Length,1);
        duration3Length = duration3Length + 1;
    end
    if i ~= 1
        Duration4(duration3Length,1) = Duration3_v2(length(Duration3_v2(:,1))-i-1,1);
        Duration4(duration3Length,3) = Duration4(duration3Length-1,2);
        Duration4(duration3Length,2) = Duration4(duration3Length,3) + Duration4(duration3Length,1);
        duration3Length = duration3Length + 1;
    end
end

clear Duration3_v2

% \ 12°
```

Figure 26 – Code to obtain the fifth duration matrix for the internal pattern

The code shown in Figure 27 outlines the procedure for defining the first and last raster lines of the internal fill pattern. To obtain the first raster line, the duration of the second raster line is taken as a starting point, and 11,000 samples are subtracted, which is the increment factor between raster lines. Almost the same procedure is applied to obtain the last raster line, where the duration of the penultimate raster line is taken and 11,000 samples are added, which is the decrement factor between raster lines. This procedure is based on the internal pattern filling characteristics represented in Figure 3 and discussed earlier.

As a result, the Duration4_v2 matrix is obtained. In the Duration4_v2 matrix, the first column relates to the duration value itself, the second column is the index related to the endpoint of the fabrication line, and the third column is the index related to the starting point of the fabrication line. From Duration4_v2, it is only necessary to separate the

indices related to the raster lines and between the transition raster lines, and to obtain the fourth and final MWD group. Figure 42 presents the duration of the lines of the internal fill pattern.

```
% 13° Obtaining sixth duration matrix for the internal pattern
% Obtain the first raster line and add to the beggining of the duraction matrix
Duration4_v2(1,2) = Duration4(1,3);
Duration4_v2(1,1) = Duration4(2,1)-varDurRaster;
Duration4_v2(1,3) = Duration4_v2(1,2) - Duration4_v2(1,1);
Duration4_v2(2:length(Duration4(:,1))+1,:) = Duration4;

% Obtain the last raster line and add to the end of the duraction matrix
Duration4_v2(length(Duration4(:,1))+2,3) = Duration4_v2(length(Duration4(:,1))+1,2);
Duration4_v2(length(Duration4(:,1))+2,1) = Duration4_v2(length(Duration4(:,1)),1)-varDurRaster;
Duration4_v2(length(Duration4(:,1))+2,2) = Duration4_v2(length(Duration4(:,1))+2,1) + Duration4_v2(length(Duration4(:,1))+2,3);

clear Duration4
% \ 13°
```

Figure 27 – Code to obtain the sixth duration matrix for the internal pattern

The code shown in Figure 28 outlines the procedure for separating the indices related to raster lines and between transition raster lines, and for obtaining the fourth MWD group.

The matrix index_raster contains only the values related to raster lines, while the matrix index_trans_raster contains only the values related to transition raster lines. In both cases, the first column relates to the duration value itself, the second column is the index related to the endpoint of the fabrication line, and the third column is the index related to the starting point of the fabrication line.

Next, the fourth group of MWDs is obtained by calculating the difference between the index for the start of the fabrication of the first raster line and the index for the end of the fabrication of the last contour line. The result is stored in the variable contourToRasterReposition. After this, an adjustment is made to the contour indices, storing the altered values in the matrix indexContourAlt, to allow for a decimation procedure when generating the Figures.

```
% 14° Obtaining the index values from the last duration matrix
rasterLineCount = 1;
transitionLineCount = 1;

for j = 1:length(Duration4_v2(:,1))
    if Duration4_v2(j,1) < lowRefTransRaster || Duration4_v2(j,1) > uppRefTransRaster
        index_raster(rasterLineCount,:) = Duration4_v2(j,:);
        rasterLineCount = rasterLineCount + 1;
    else
        index_trans_raster(transitionLineCount,:) = Duration4_v2(j,:);
        transitionLineCount = transitionLineCount + 1;
    end
end

contourToRasterReposition(1,3) = indexContour(numofContourLines,3);
contourToRasterReposition(1,2) = Duration4_v2(1,3);
contourToRasterReposition(1,1) = contourToRasterReposition(1,2) -...
    contourToRasterReposition(1,3);

contourToRasterRepositionAux = contourToRasterReposition;

contourToRasterReposition(1,2) = contourToRasterRepositionAux(1,3);
contourToRasterReposition(1,3) = contourToRasterRepositionAux(1,2);

indexContourAlt = indexContour;

for i = 1:1:numofContourLines
    indexContourAlt(i,2) = indexContour(i,2);
    indexContourAlt(i,3) = indexContour(i,3)-5;
end

if length(index_raster) < numofRasterLines
    [index_raster,index_trans_raster] =...
        adjust_internal(index_raster,index_trans_raster);

end

% Verify external pattern integrity

square1ExternalPattern = mean (indexContourAlt(1:numofContourSides,1));
square2ExternalPattern = mean (indexContourAlt((numofContourSides+1):(numofContourSides*2),1));

if abs(square2ExternalPattern - square1ExternalPattern) > (varDurRaster+4e3)
    [contourRepositions, indexContourAlt] = adjustExternalPattern(contourRepositions, indexContourAlt);
end

%

% Verify internal pattern integrity
rasterAuxMatrix = index_raster;
rasterAuxMatrix(1,numofContourSides) = 0;

for i = 2:length(index_raster(:,1))
    rasterAuxMatrix(i,numofContourSides) = abs(index_raster(i,(numofContourSides-1)) - index_raster(i-1,2));
end

maxRasterAux = max(rasterAuxMatrix(:,numofContourSides));

indexAdjust = find (rasterAuxMatrix(:,numofContourSides) > (varDurRaster-1e3));

if maxRasterAux > uppRefTransRaster
    [index_raster, index_trans_raster] = adjustInternalPattern(index_raster, index_trans_raster, indexAdjust);
end

%

signalReposition = Compos3r(sensorSignalNormalized,contourToRasterReposition(:,2:3))+...
    Compos3r(sensorSignalNormalized,contourRepositions(:,2:3));
signalRaster = Compos3r(sensorSignalNormalized, index_raster(:,2:3));
signalTransRaster = Compos3r(sensorSignalNormalized, index_trans_raster(:,2:3));
signalContour = Compos3r(sensorSignalNormalized, indexContourAlt(:,2:3));

% \ 14°
```

Figure 28 – Code to obtain the index values from the last duration matrix

The code shown in Figure 29 outlines the procedure for obtaining the segmentation algorithm results based on the user's choice ('points' or 'segments').

```
% 15° Obtain the segmentation results

testSegmentChoice = strcmp(segmentationChoice,'Points');

testxticksChoice = strcmp(xticksChoice,'Seconds');

if testSegmentChoice == true
    resultReposition = [contourRepositions; contourToRasterReposition];
    resultContour = indexContourAlt;
    resultRaster = index_raster;
    resultTransitionRaster = index_trans_raster;
    resultWholeWorkpiece = [resultContour(1,2) resultRaster(numofRasterLines,2)];
    resultExternalPattern = [resultContour(1,2) resultContour(numofContourLines,3)];
    resultInternalPattern = [resultRaster(1,3) resultRaster(numofRasterLines,2)];

    if testxticksChoice == true
        resultReposition = convUni(sensorSignalNormalized, resultReposition, Fs);
        resultContour = convUni(sensorSignalNormalized, resultContour, Fs);
        resultRaster = convUni(sensorSignalNormalized, resultRaster, Fs);
        resultTransitionRaster = convUni(sensorSignalNormalized, resultTransitionRaster, Fs);
        resultWholeWorkpiece = convUni(sensorSignalNormalized, resultWholeWorkpiece, Fs);
        resultExternalPattern = convUni(sensorSignalNormalized, resultExternalPattern, Fs);
        resultInternalPattern = convUni(sensorSignalNormalized, resultInternalPattern, Fs);
    end

    resultReposition = obtainTable(resultReposition(:,1),...
        resultReposition(:,2), resultReposition(:,3));
    resultContour = obtainTable(resultContour(:,1),...
        resultContour(:,2), resultContour(:,3));
    resultRaster = obtainTable(resultRaster(:,1),...
        resultRaster(:,3), resultRaster(:,2));
    resultTransitionRaster = obtainTable(resultTransitionRaster(:,1),...
        resultTransitionRaster(:,3), resultTransitionRaster(:,2));

    StartPoint = resultWholeWorkpiece(1);
    EndPoint = resultWholeWorkpiece(2);
    resultWholeWorkpiece = table(StartPoint,EndPoint);

    StartPoint = resultExternalPattern(1);
    EndPoint = resultExternalPattern(2);
    resultExternalPattern = table(StartPoint,EndPoint);

    StartPoint = resultInternalPattern(1);
    EndPoint = resultInternalPattern(2);
    resultInternalPattern = table(StartPoint,EndPoint);

end

testSegmentChoice = strcmp(segmentationChoice,'segments');

if testSegmentChoice == true
    resultReposition = gen_signal_segments(sensorSignalNormalized,...
        contourRepositions);
    resultReposition(1,numofContourSides) = gen_signal_segments(sensorSignalNormalized,...
        contourToRasterReposition);
    resultContour = gen_signal_segments(sensorSignalNormalized, indexContour);
    resultRaster = gen_signal_segments(sensorSignalNormalized, index_raster);
    resultTransitionRaster = gen_signal_segments(sensorSignalNormalized, index_trans_raster);
    resultWholeWorkpiece = gen_signal_segments(sensorSignalNormalized, [resultContour(1,2) resultRaster(numofRasterLines,2)]);
    resultExternalPattern = gen_signal_segments(sensorSignalNormalized, [resultContour(1,2) resultContour(numofContourLines,3)]);
    resultInternalPattern = gen_signal_segments(sensorSignalNormalized, [resultRaster(1,3) resultRaster(numofRasterLines,2)]);
end

% \ 15°
```

Figure 29 – Code to obtain the segmentation results

The code shown in Figure 30 outlines the procedure for storing variables and generating Figures according to the user's choices.

```
546  % 16° Save the results and generate the graphs
547
548  if saveChoice == 'Y'
549      save_files(resultReposition, resultContour, resultRaster,...
550          resultTransitionRaster, resultWholeWorkpiece, resultExternalPattern,...
551          resultInternalPattern, segmentationChoice, signalIdentifier);
552  end
553
554  if graphical == 'Y'
555      gen_graph(signalReposition, sensorSignalNormalized, Fs,...
556          signalIdentifier, signalContour, signalRaster,...
557          signalTransRaster, index_raster, figureChoice);
558  end
559
560  % \ 16°
```

Figure 30 – Code to save the results and generate the graphs

DEBUG – Test1 dataset

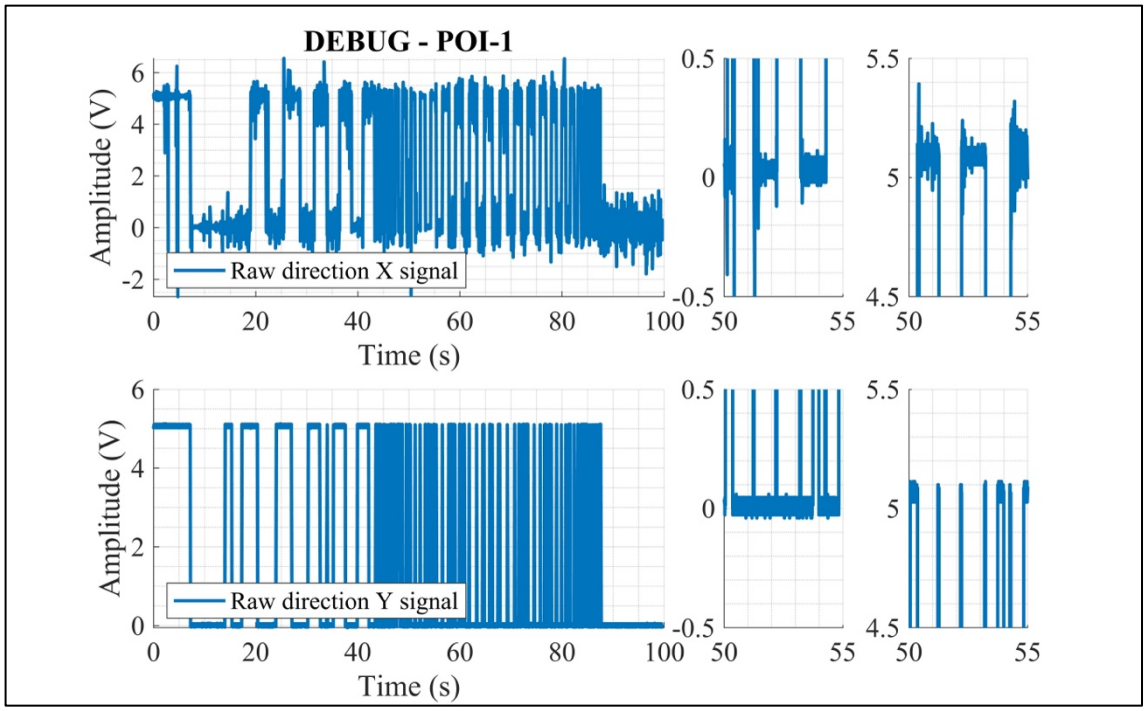

Figure 31 – DEBUG POI-1 for Test1

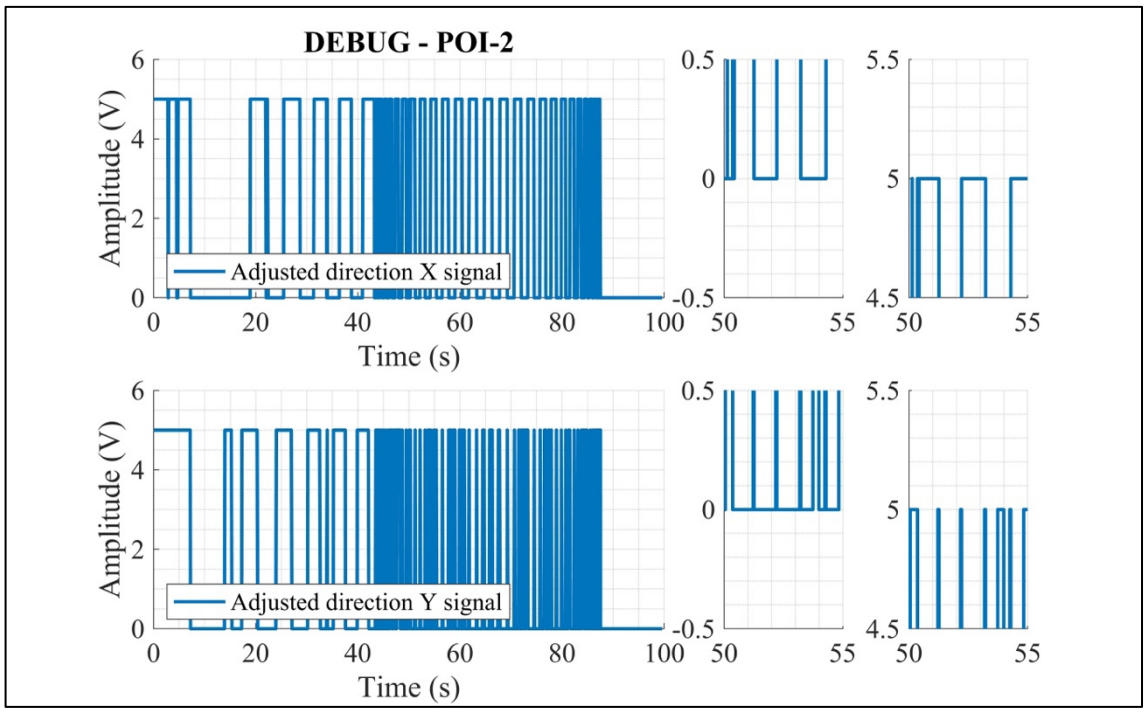

Figure 32 – DEBUG POI-2 for Test1

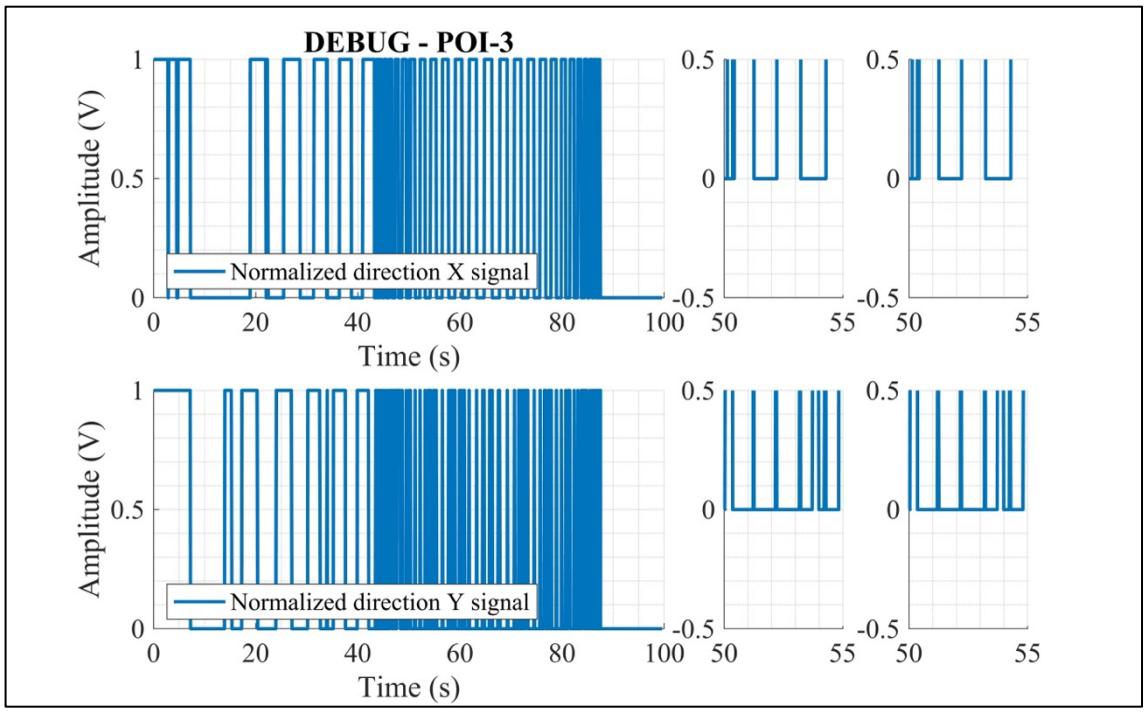

Figure 33 – DEBUG POI-3 for Test1

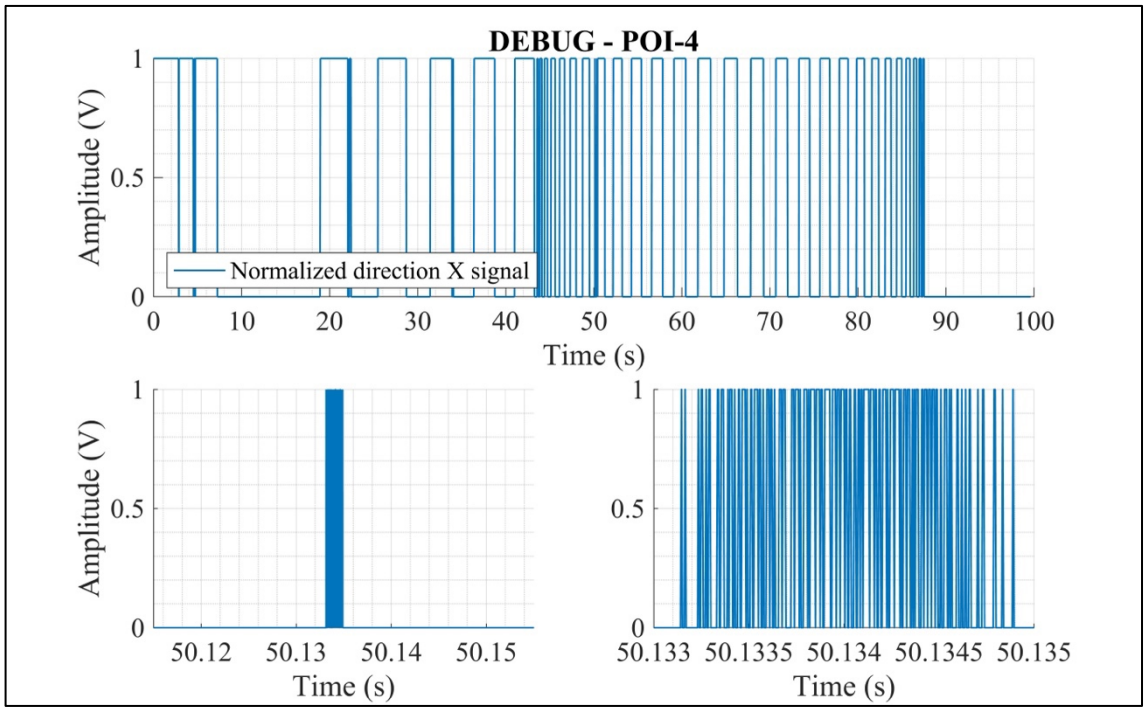

Figure 34 – DEBUG POI-4 for Test1

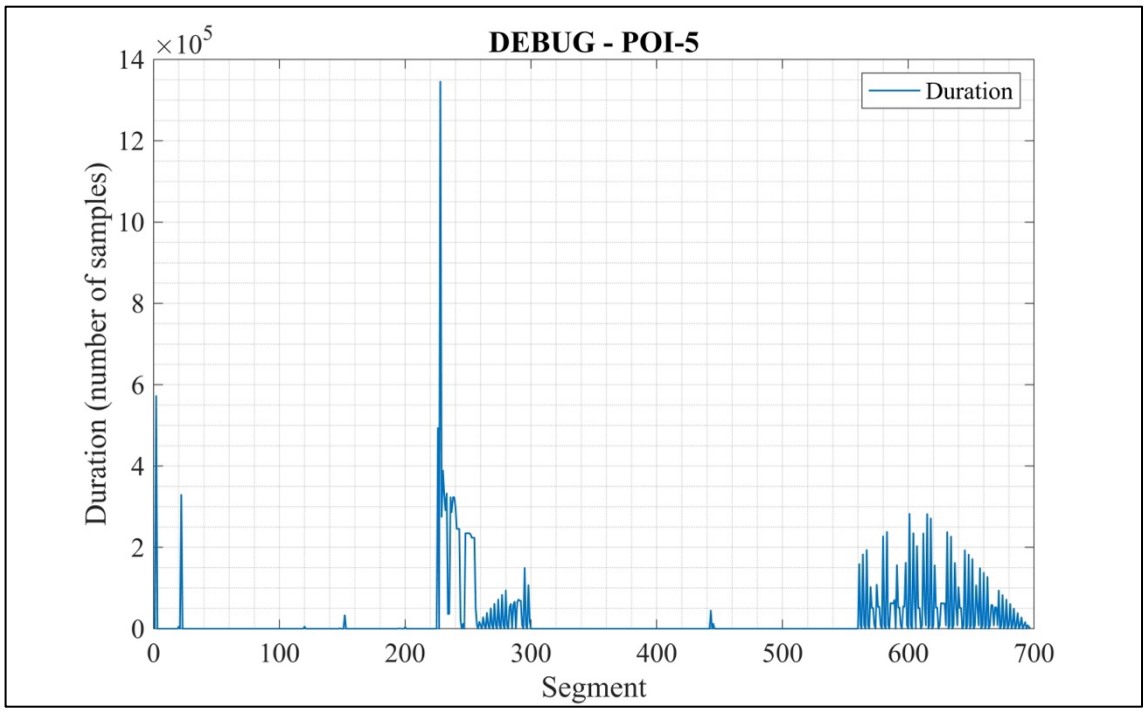

Figure 35 – DEBUG POI-5 for Test1

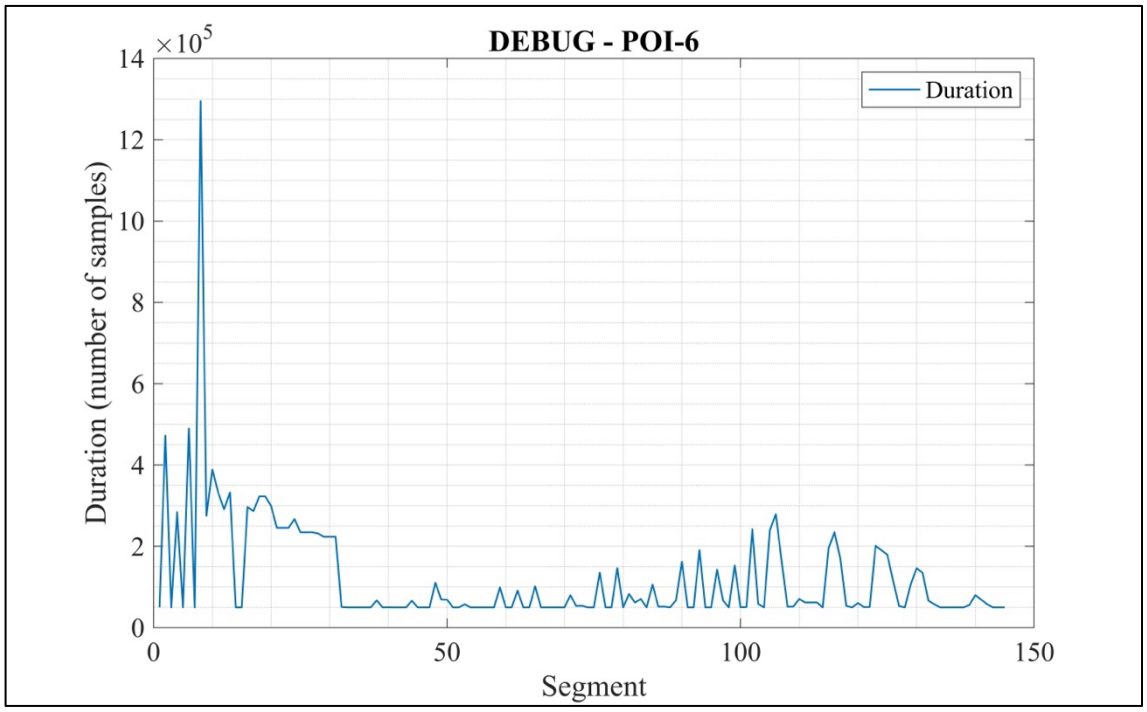

Figure 36 – DEBUG POI-6 for Test1

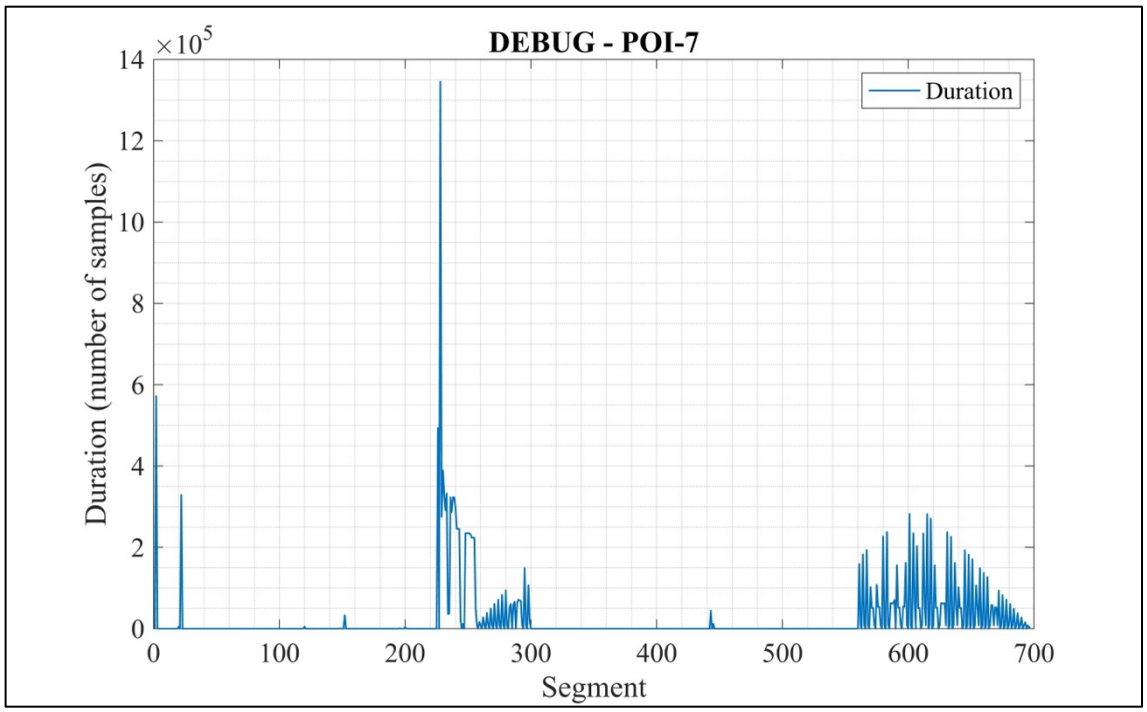

Figure 37 – DEBUG POI-7 for Test1

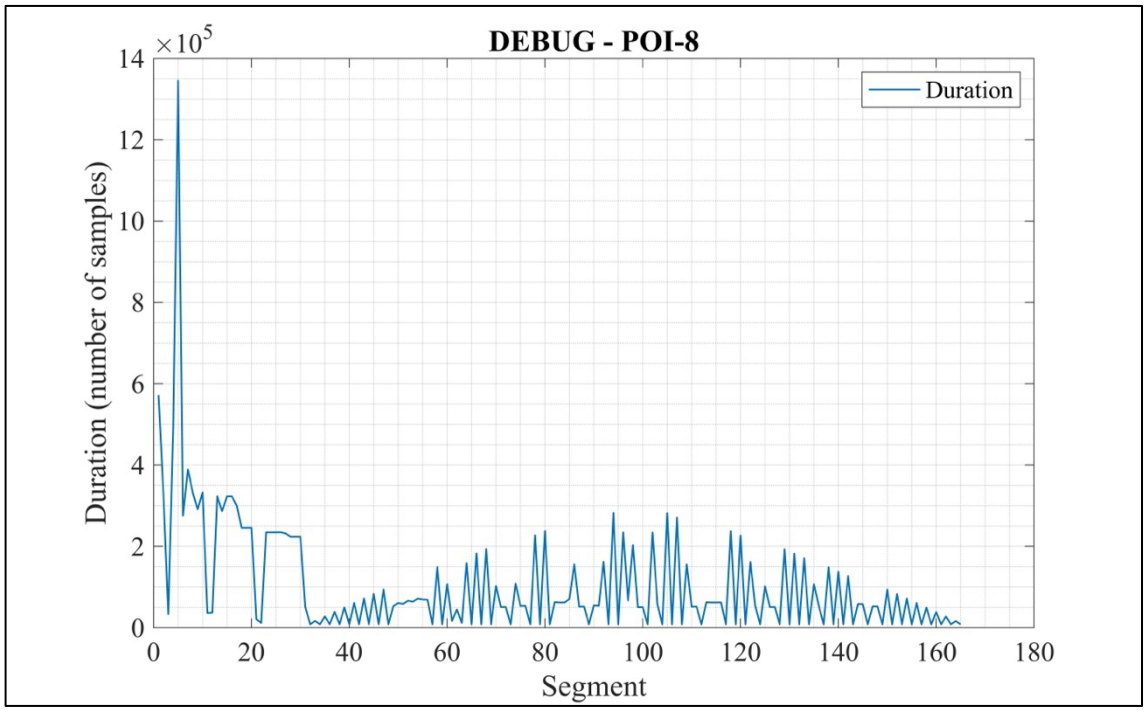

Figure 38 – DEBUG POI-8 for Test1

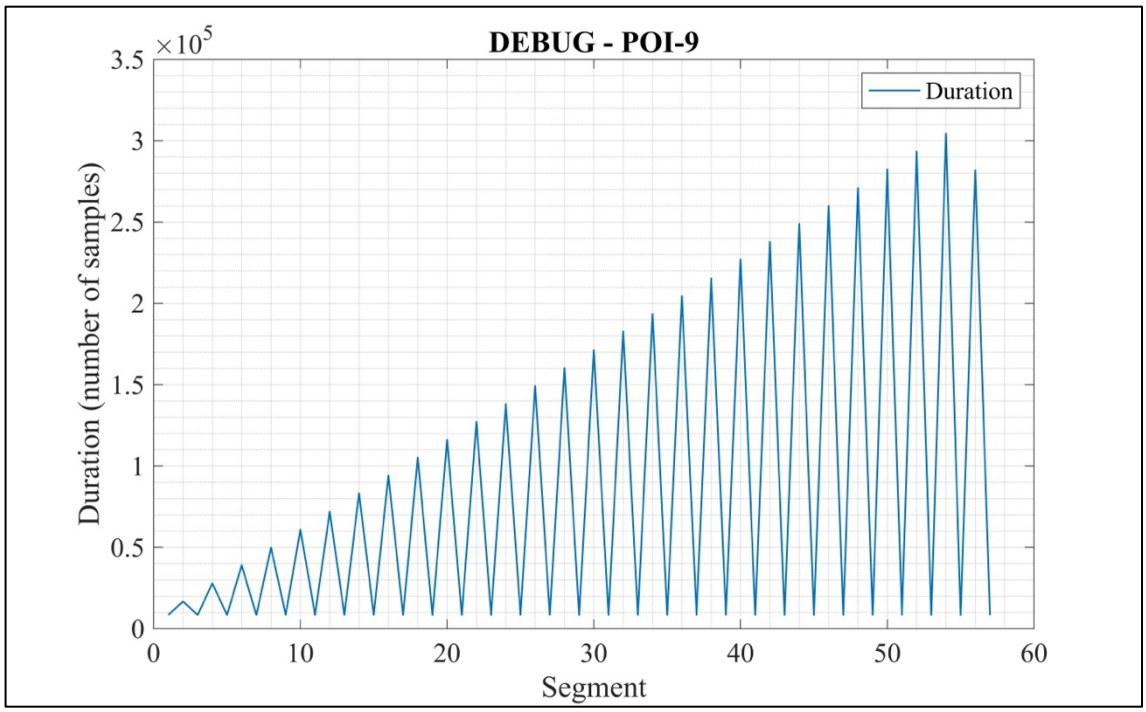

Figure 39 – DEBUG POI-9 for Test1

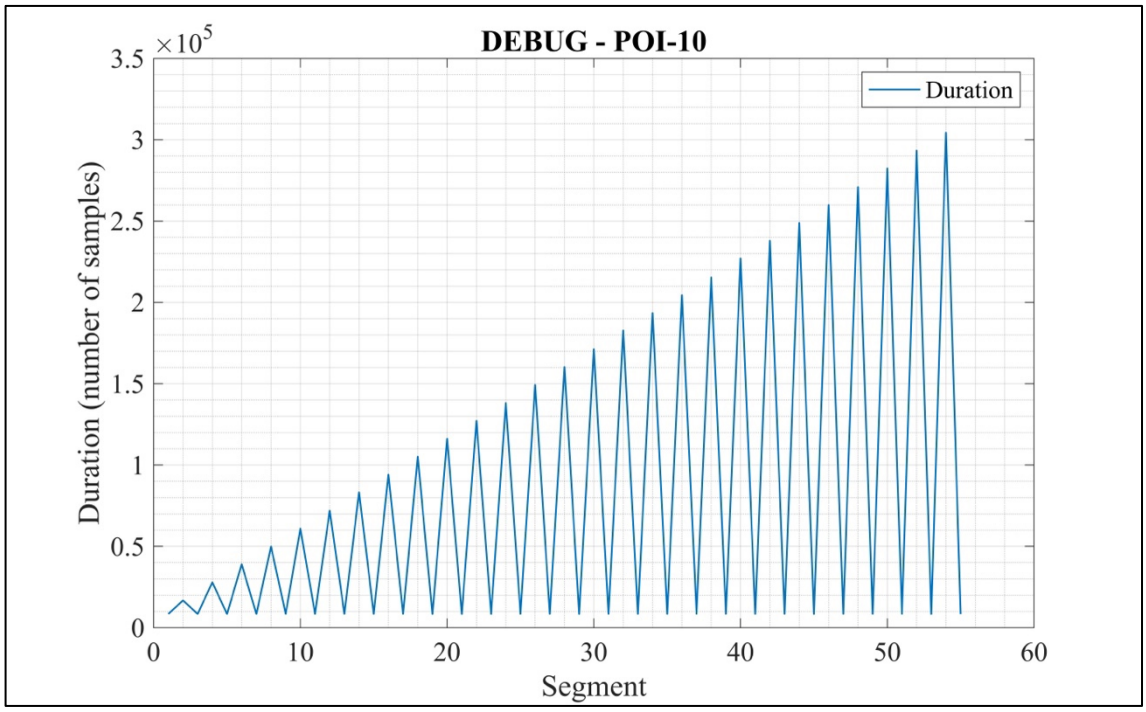

Figure 40 – DEBUG POI-10 for Test1

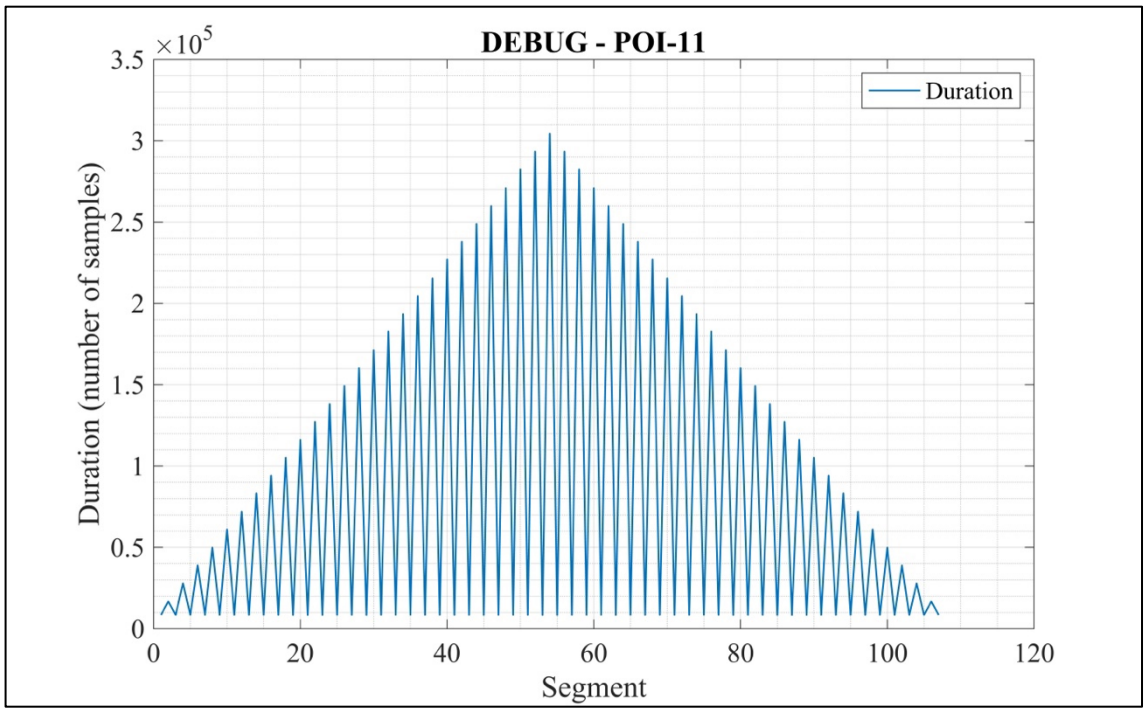

Figure 41 – DEBUG POI-11 for Test1

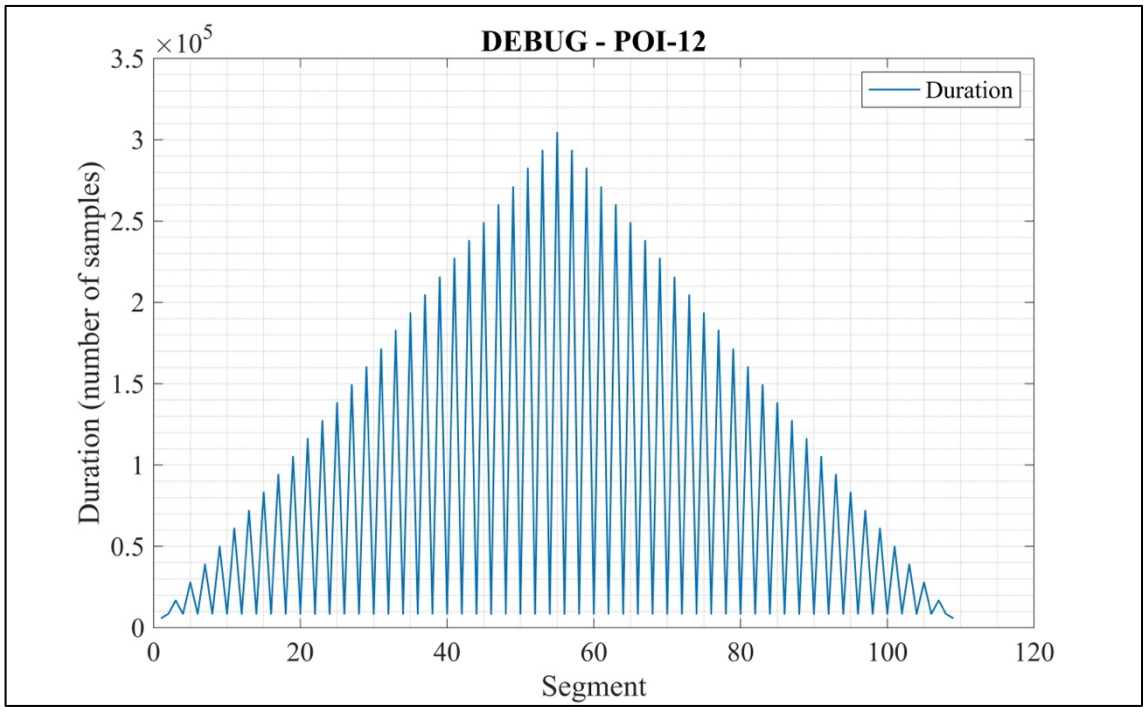

Figure 42 – DEBUG POI-12 for Test1

DEBUG – Test2 dataset

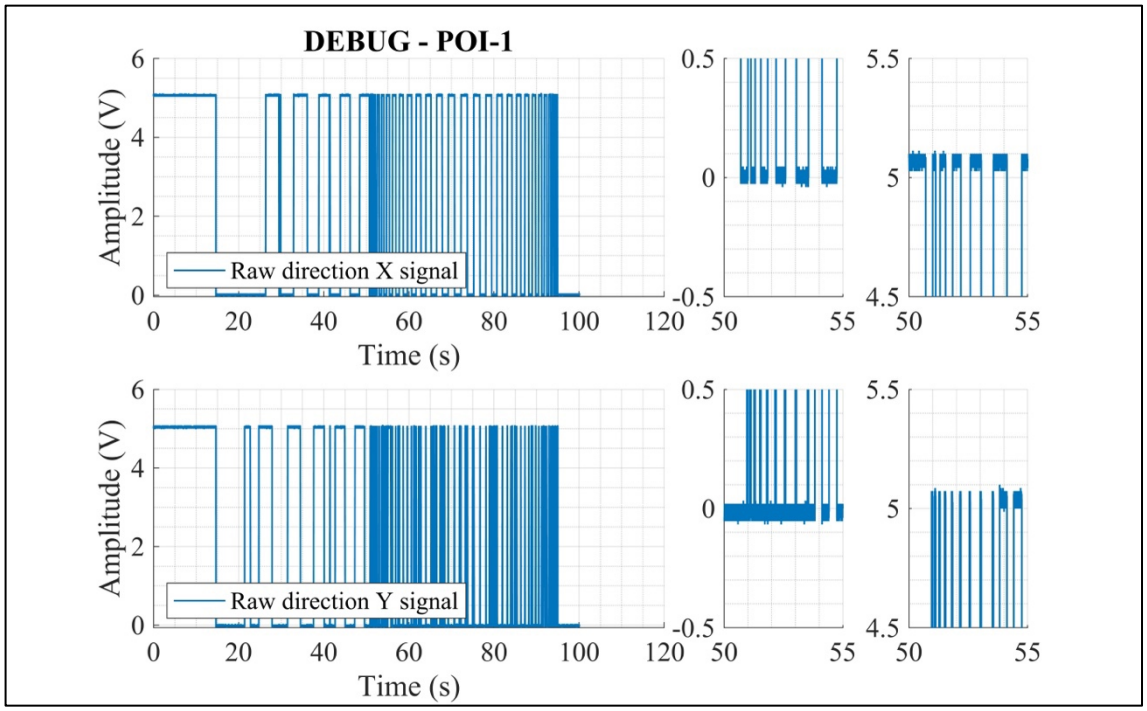

Figure 43 – DEBUG POI-1 for Test2

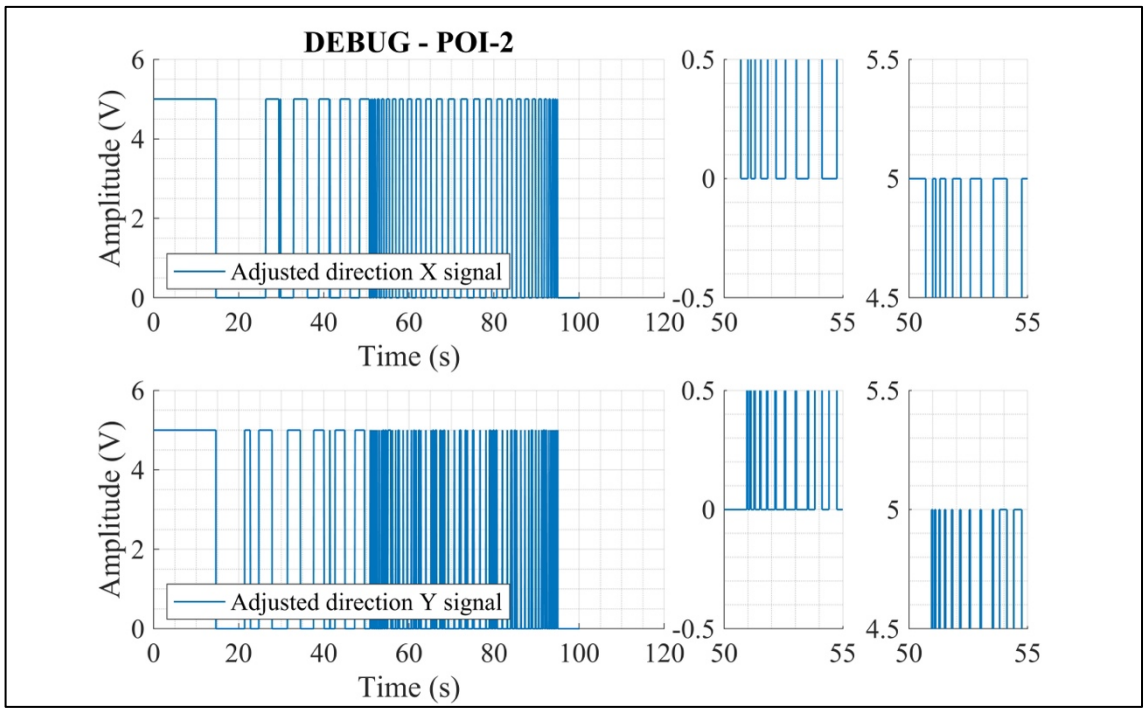

Figure 44 – DEBUG POI-2 for Test2

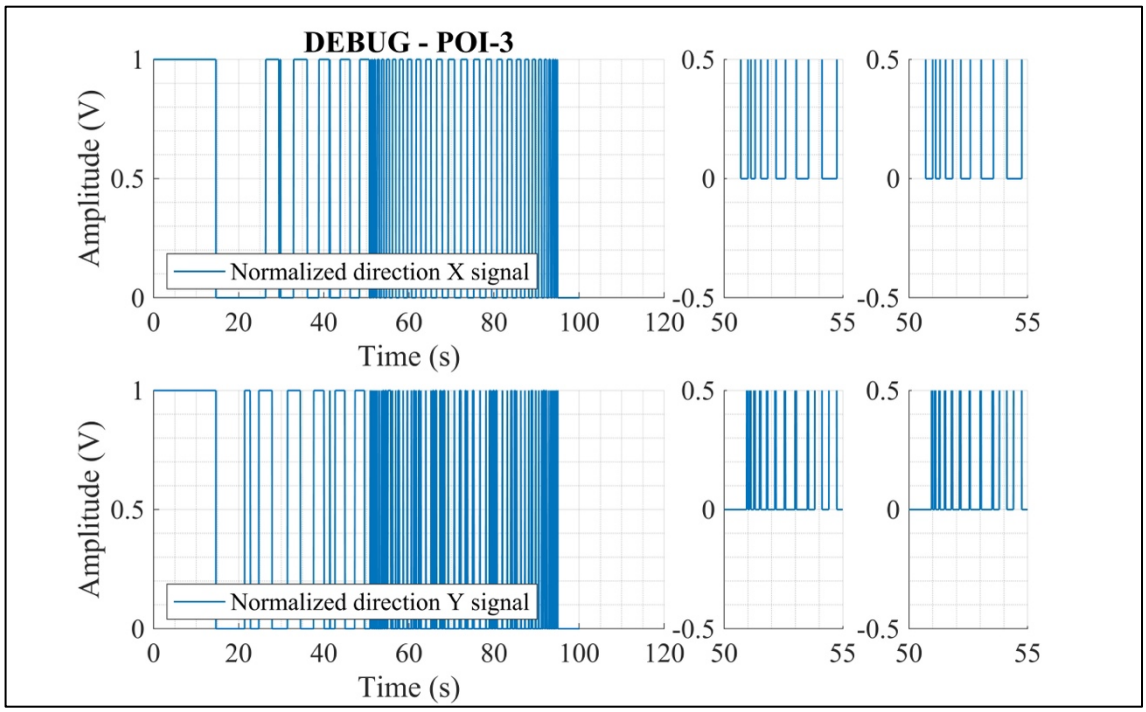

Figure 45 – DEBUG POI-3 for Test2

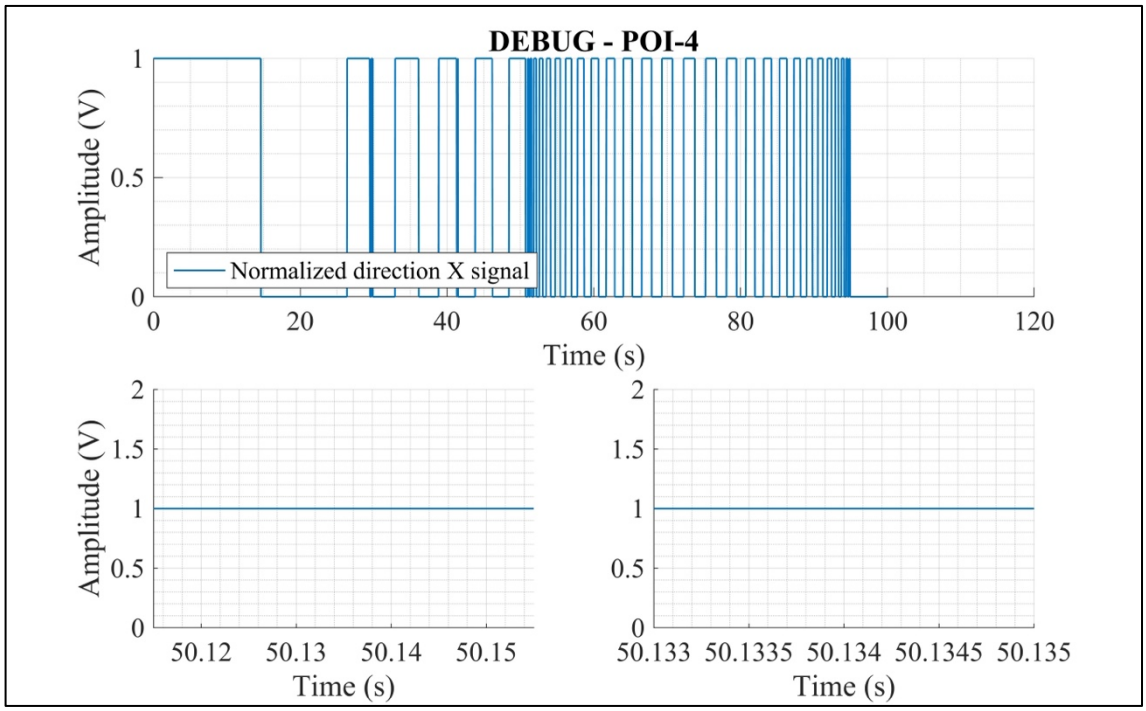

Figure 46 – DEBUG POI-4 for Test2

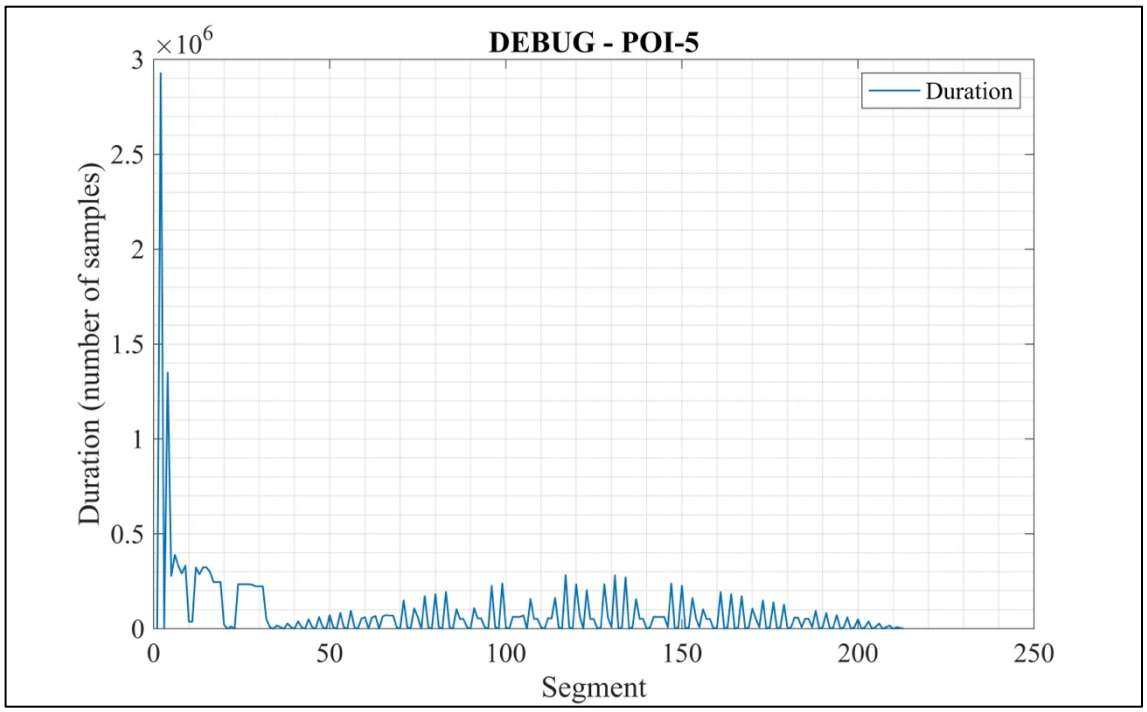

Figure 47 – DEBUG POI-5 for Test2

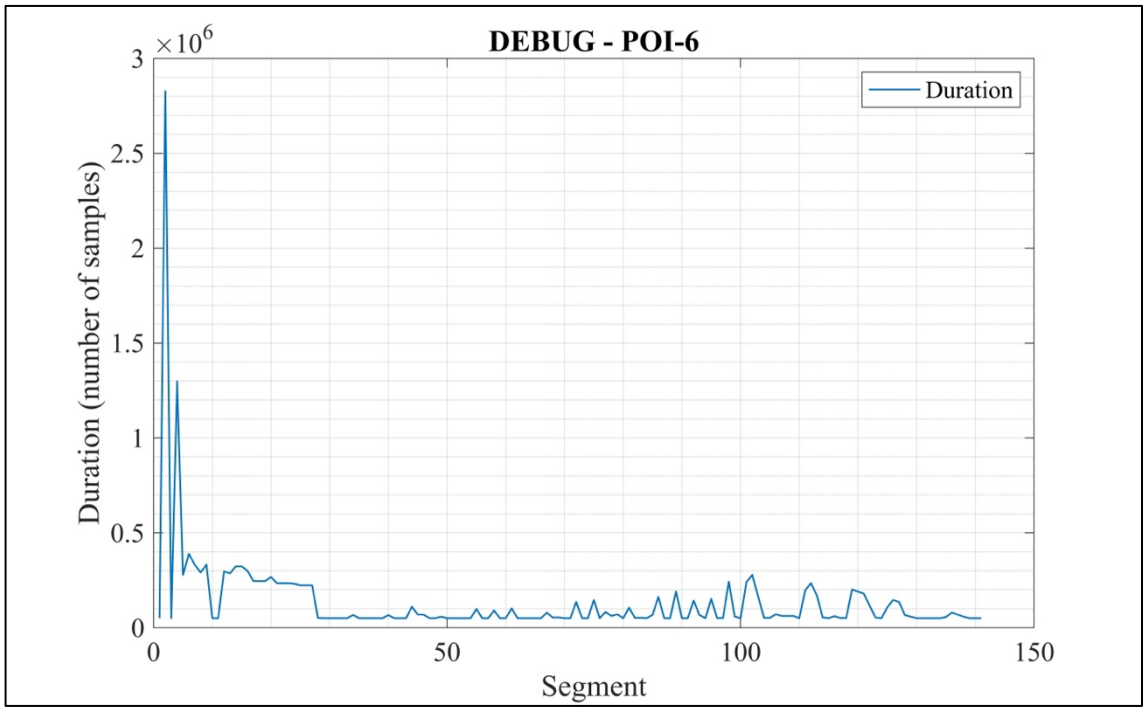

Figure 48 – DEBUG POI-6 for Test2

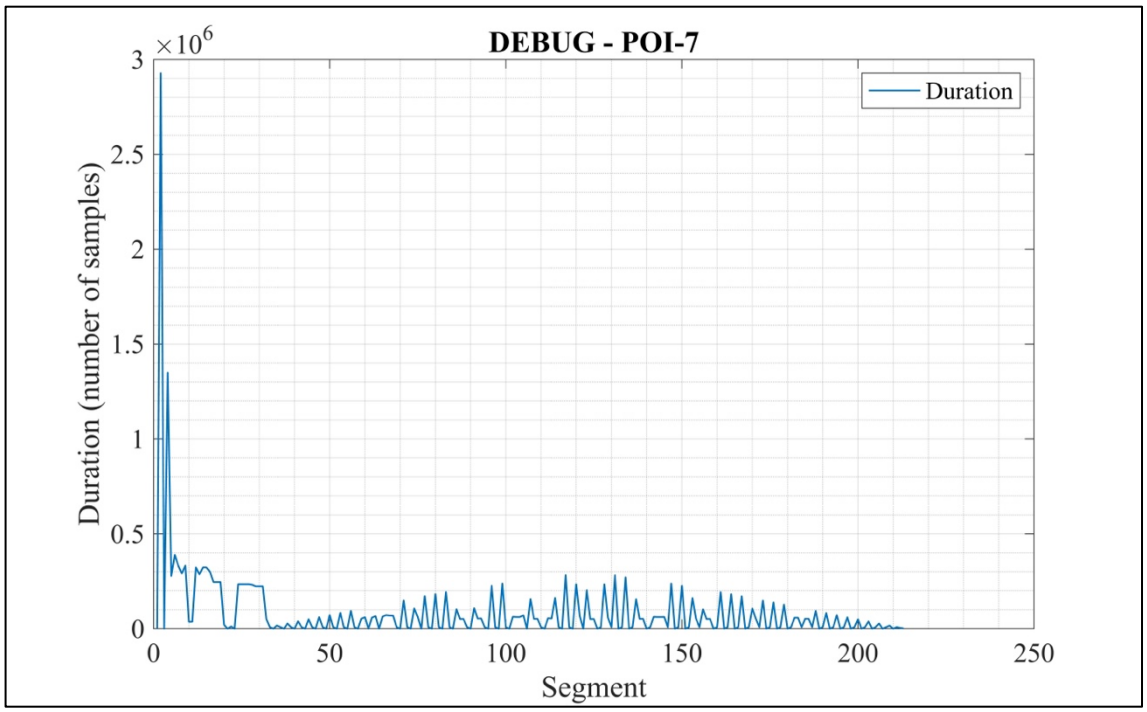

Figure 49 – DEBUG POI-7 for Test2

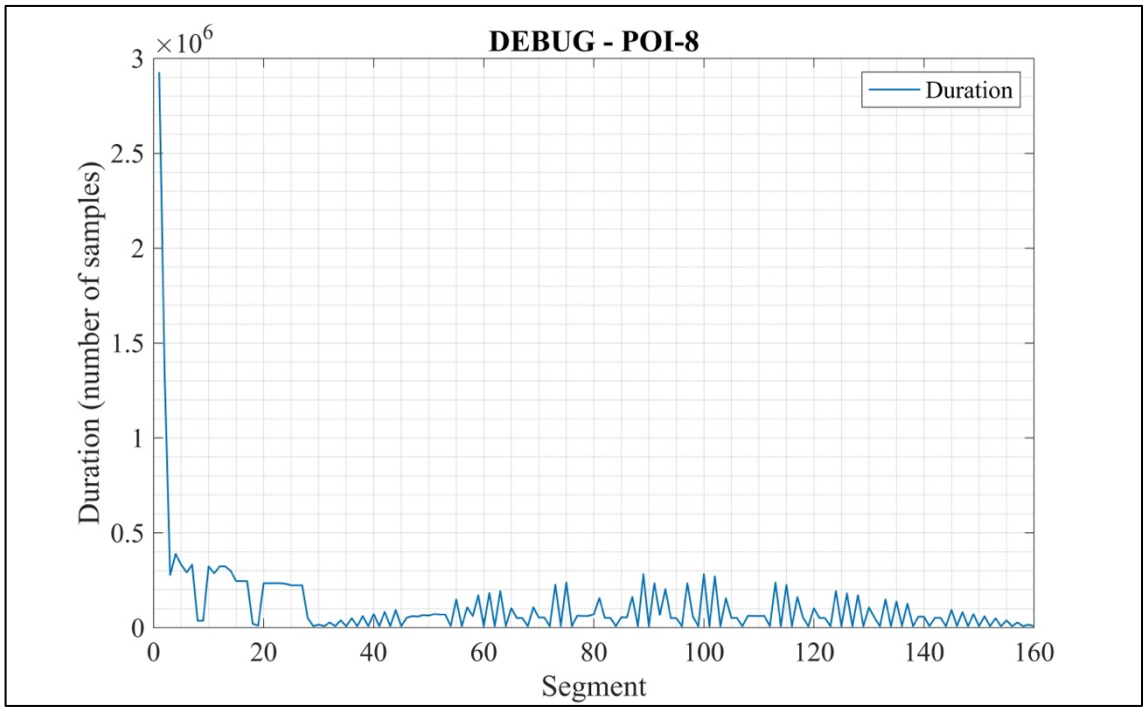

Figure 50 – DEBUG POI-8 for Test2

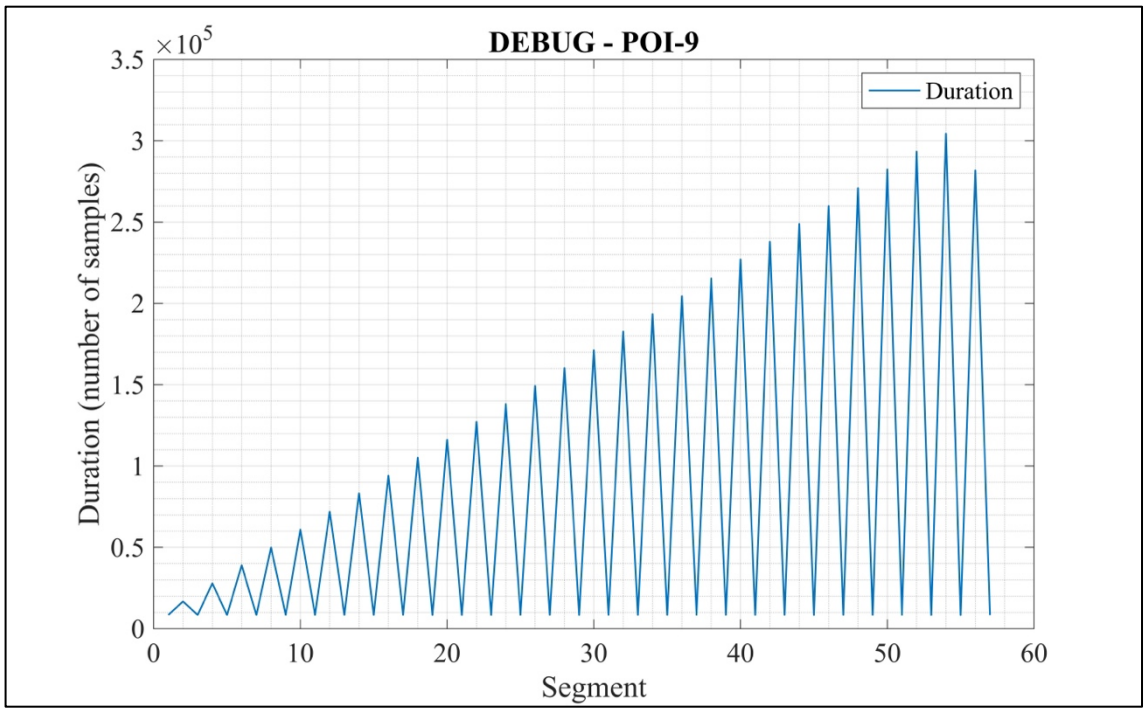

Figure 51 – DEBUG POI-9 for Test2

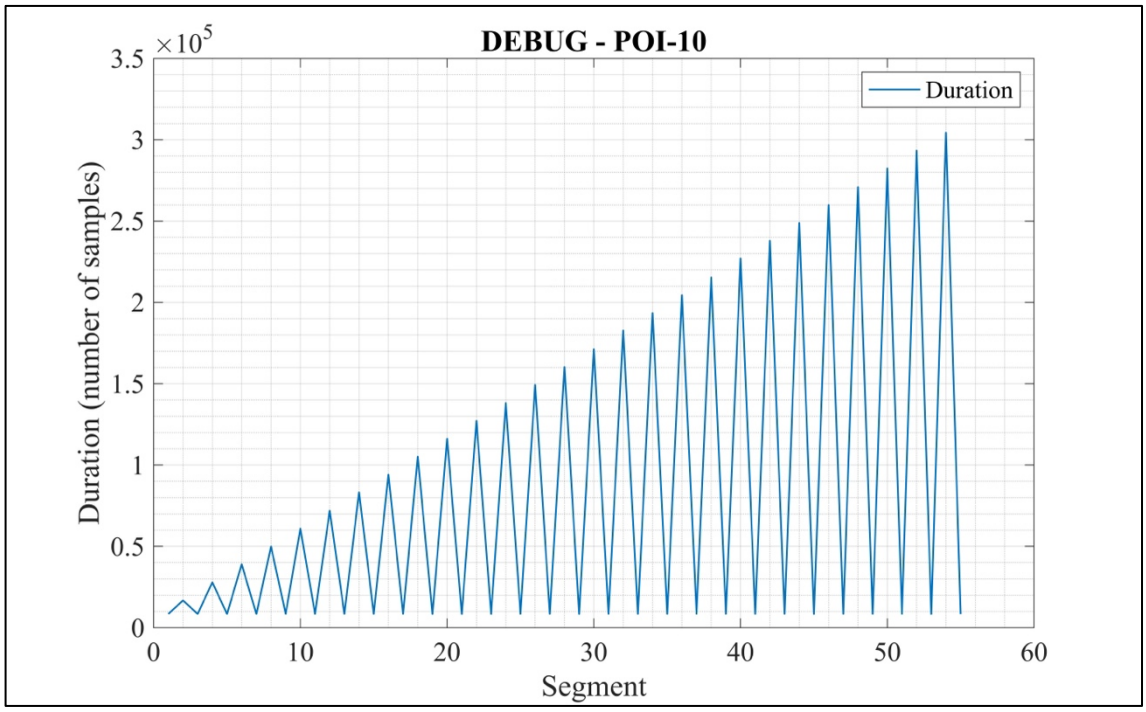

Figure 52 – DEBUG POI-10 for Test2

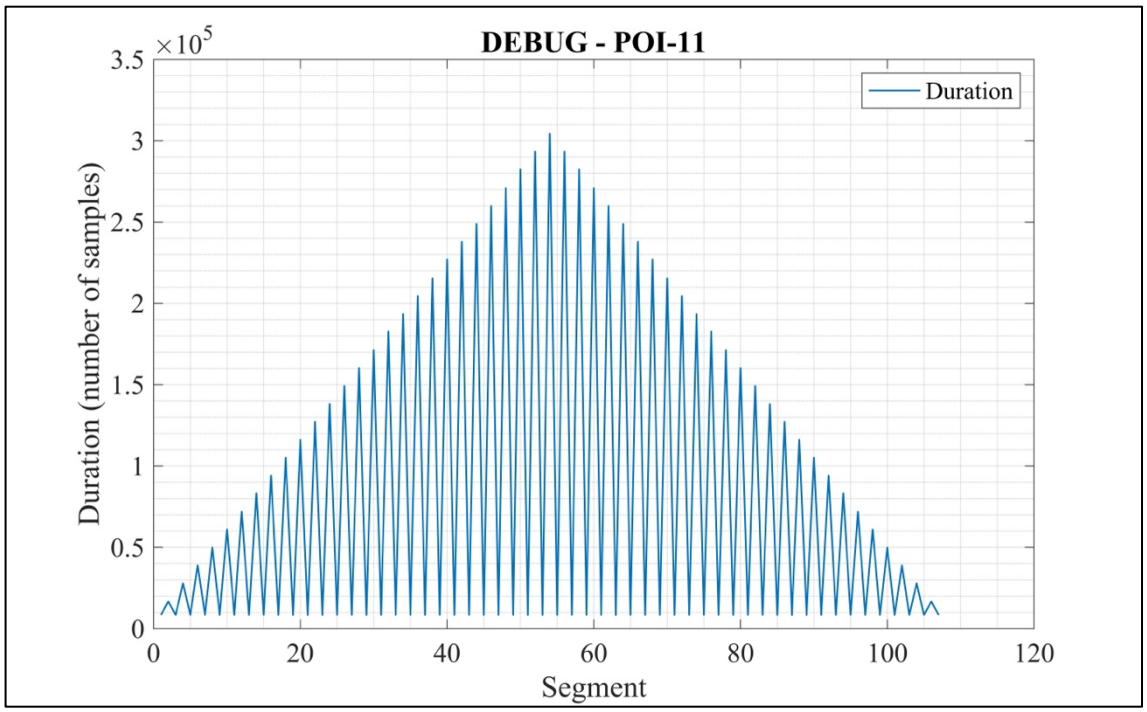

Figure 53 – DEBUG POI-11 for Test2

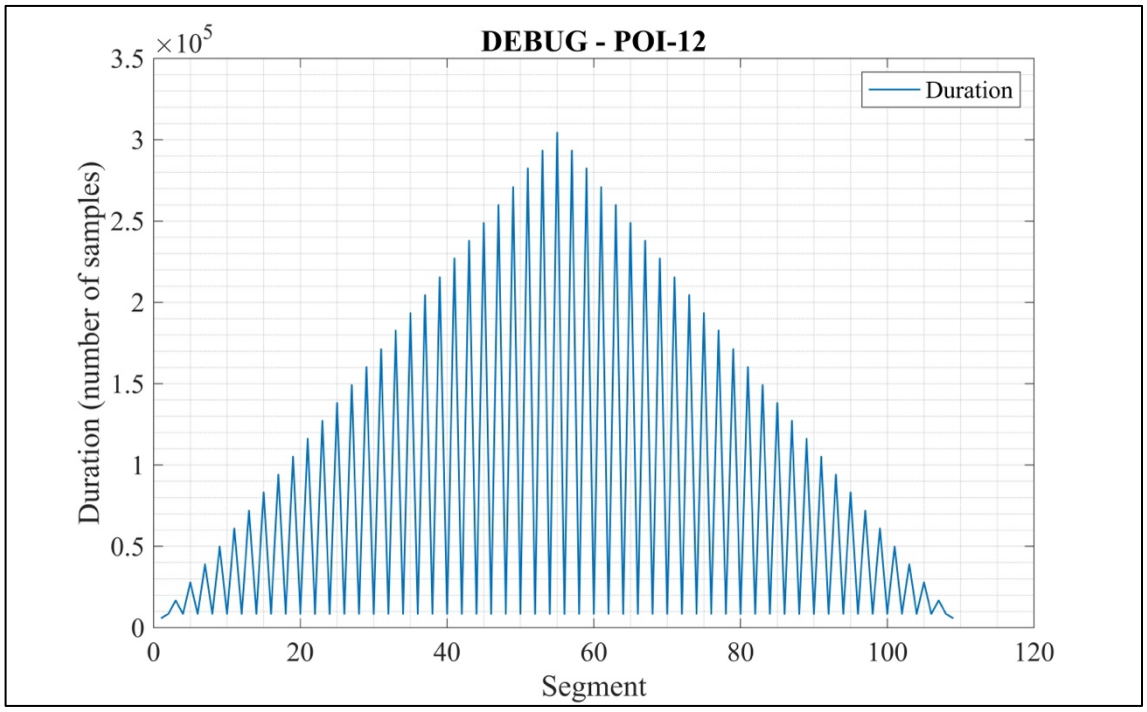

Figure 54 – DEBUG POI-12 for Test2

DEBUG – Test3 dataset

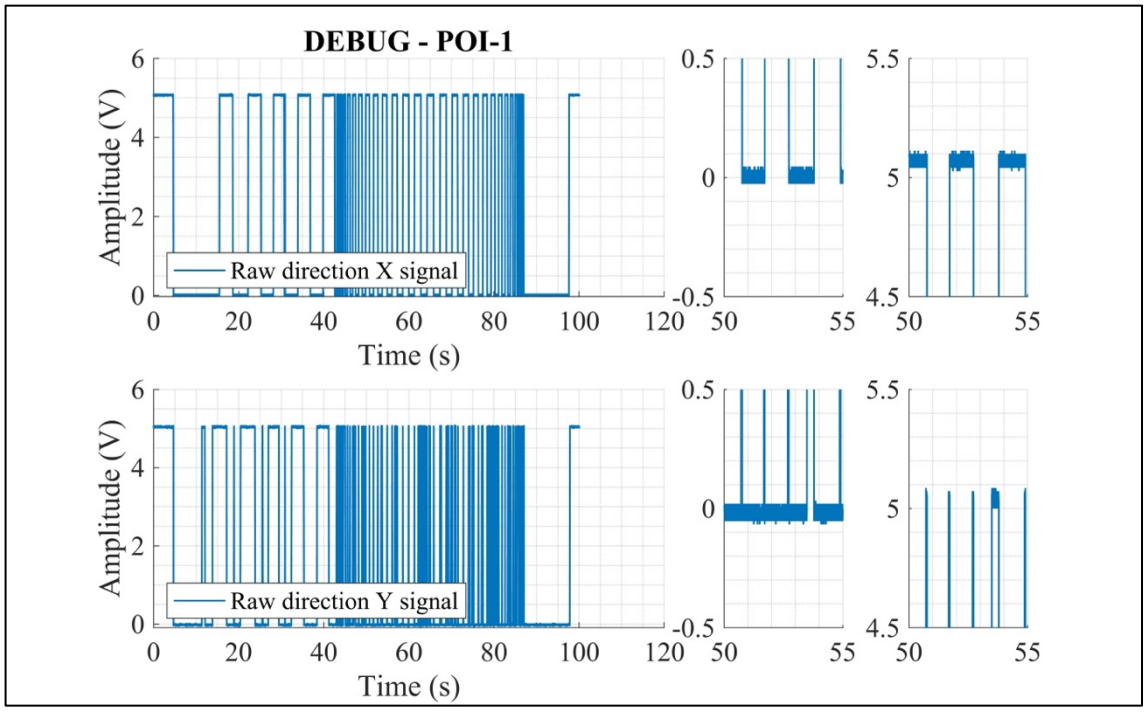

Figure 55 – DEBUG POI-1 for Test3

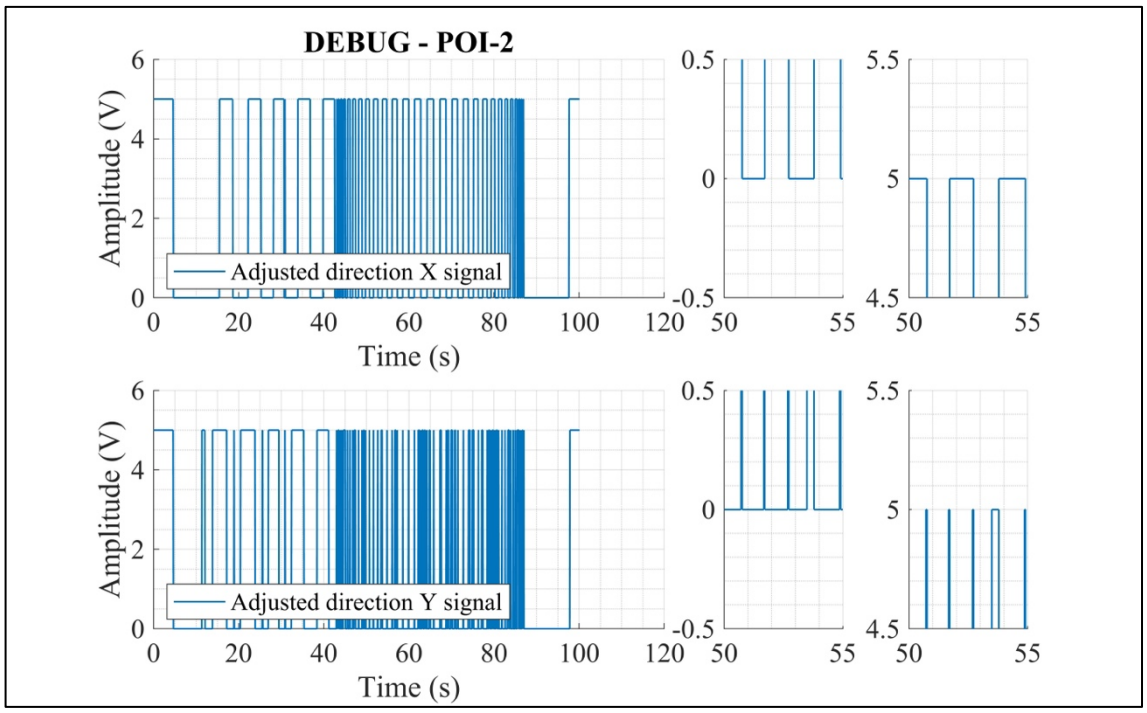

Figure 56 – DEBUG POI-2 for Test3

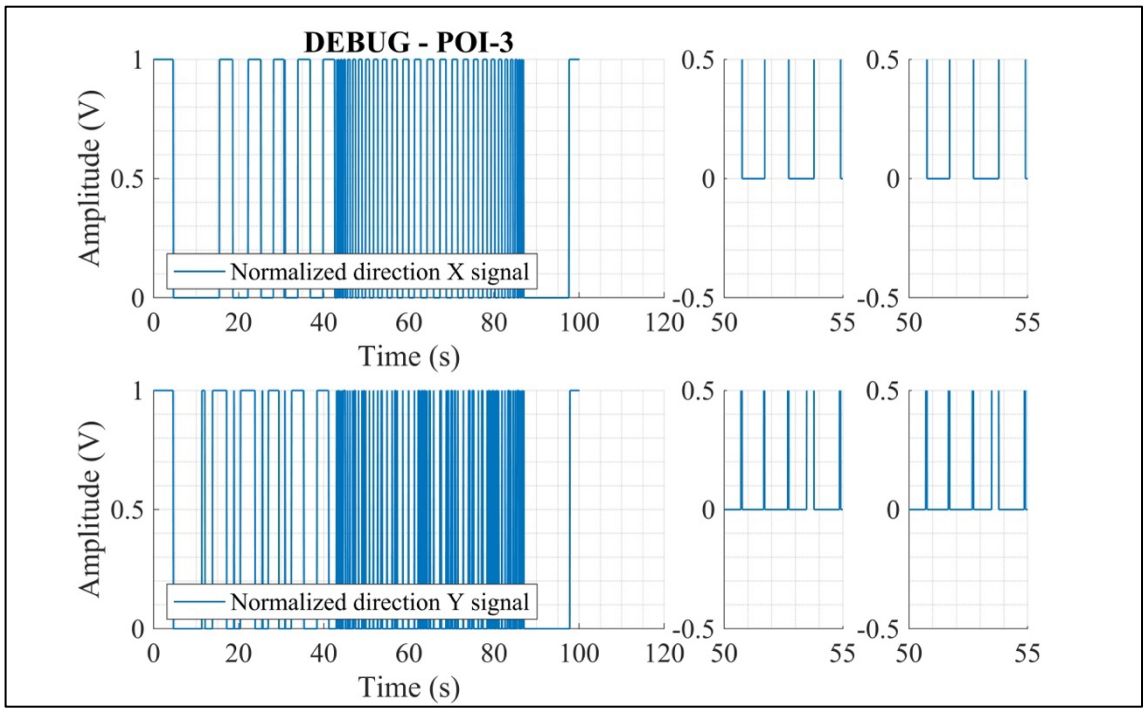

Figure 57 – DEBUG POI-3 for Test3

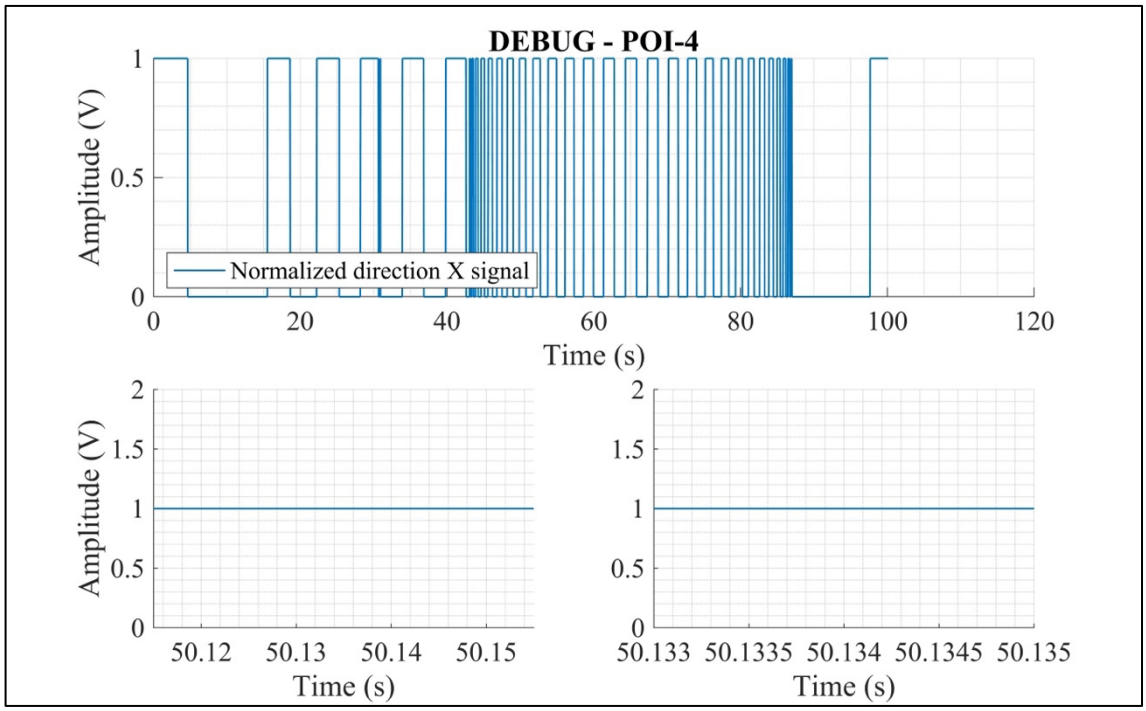

Figure 58 – DEBUG POI-4 for Test3

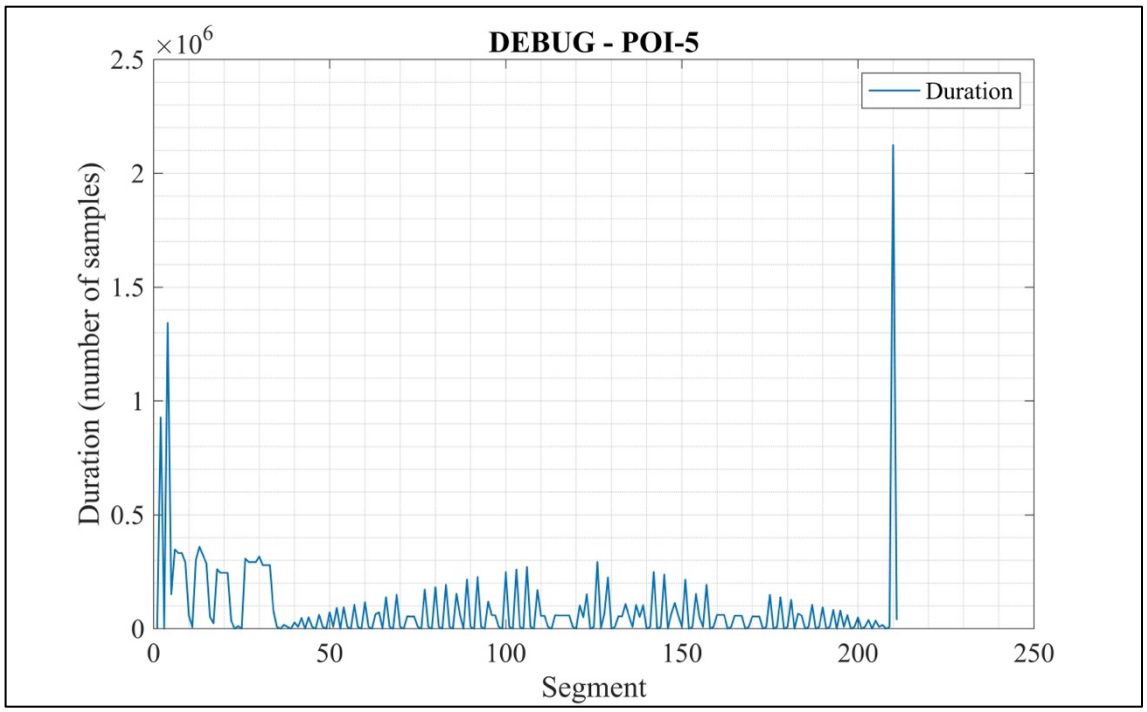

Figure 59 – DEBUG POI-5 for Test3

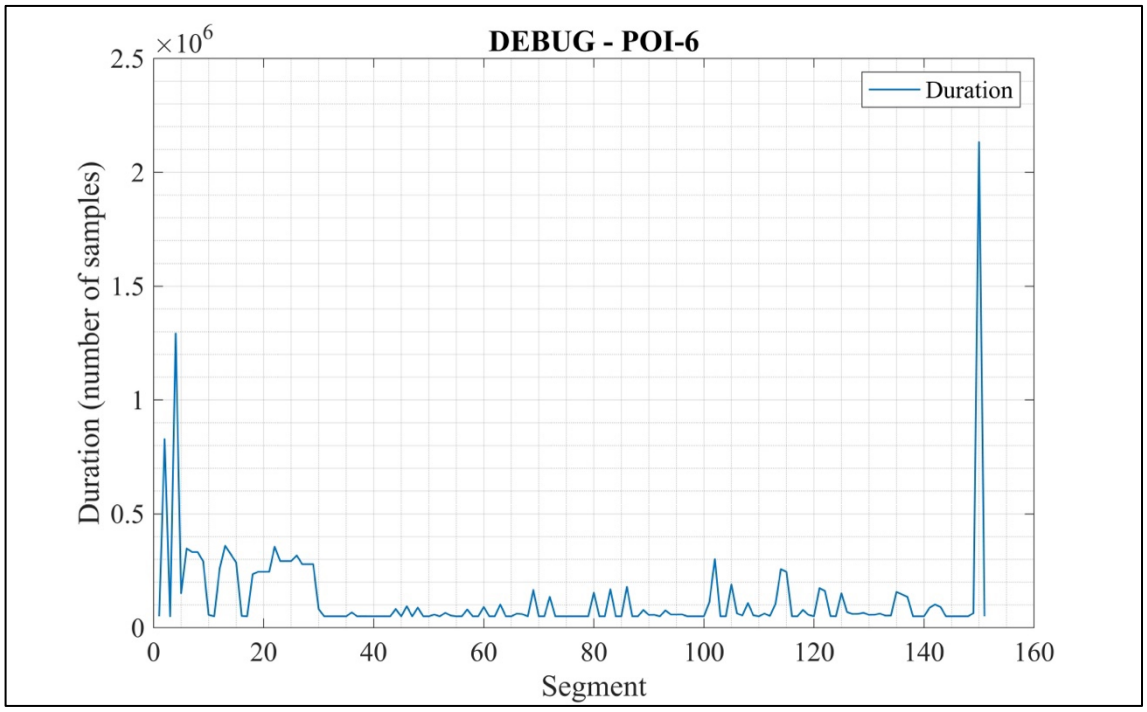

Figure 60 – DEBUG POI-6 for Test3

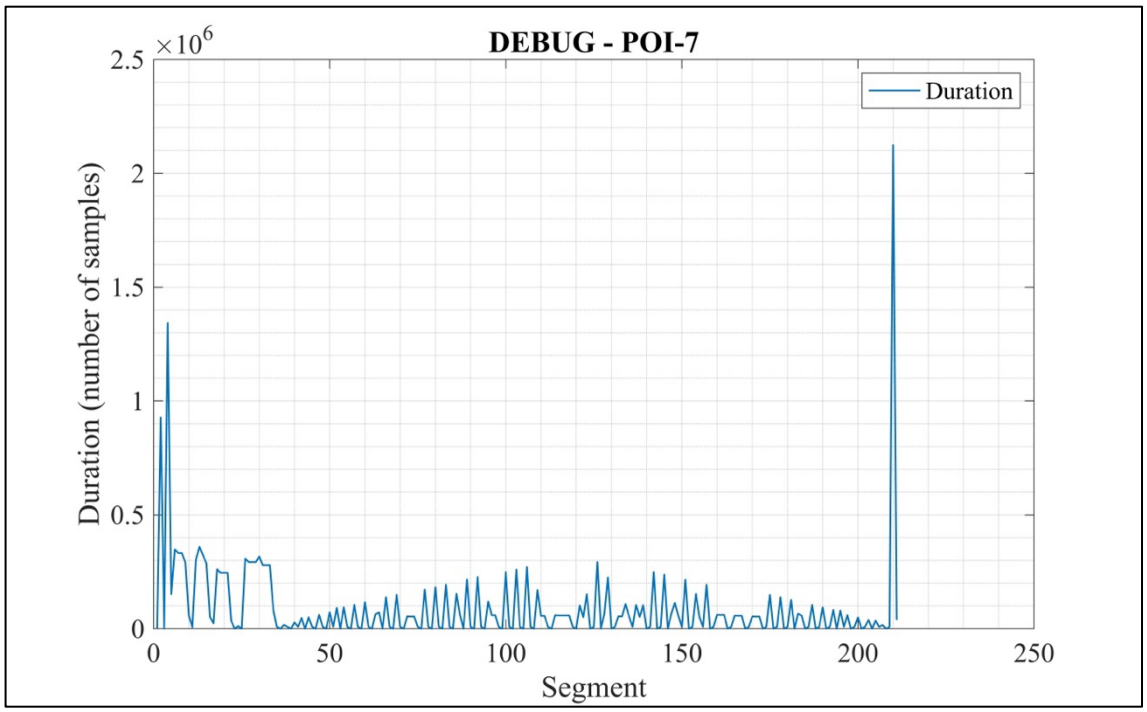

Figure 61 – DEBUG POI-7 for Test3

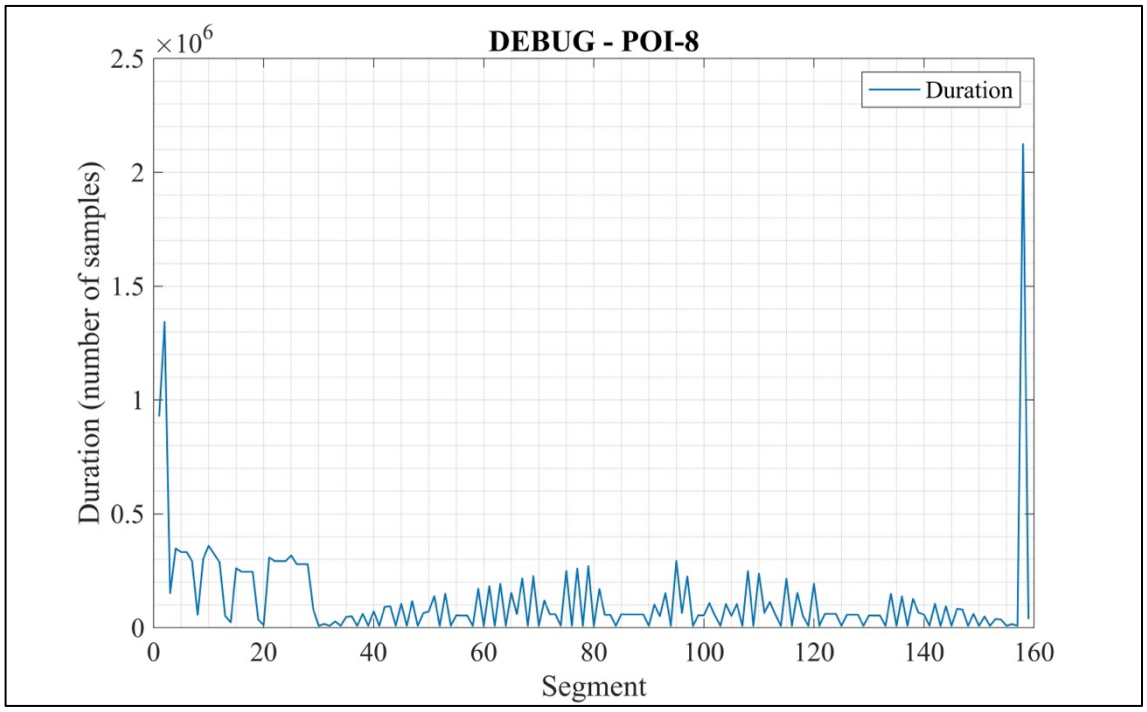

Figure 62 – DEBUG POI-8 for Test3

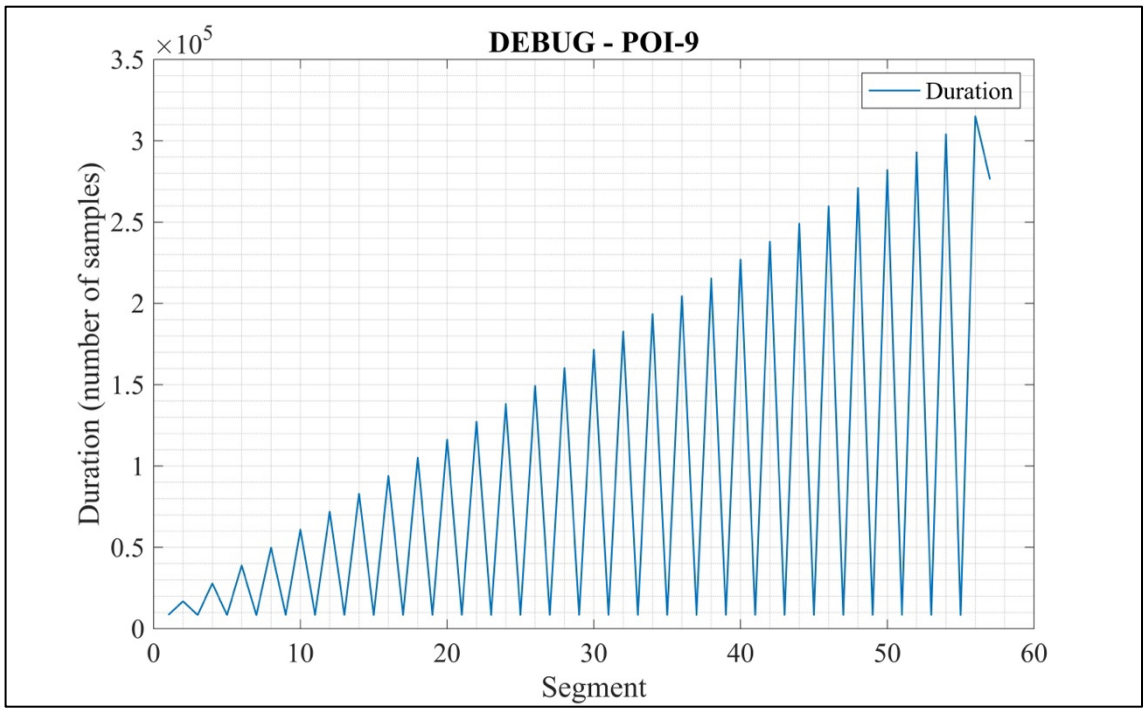

Figure 63 – DEBUG POI-9 for Test3

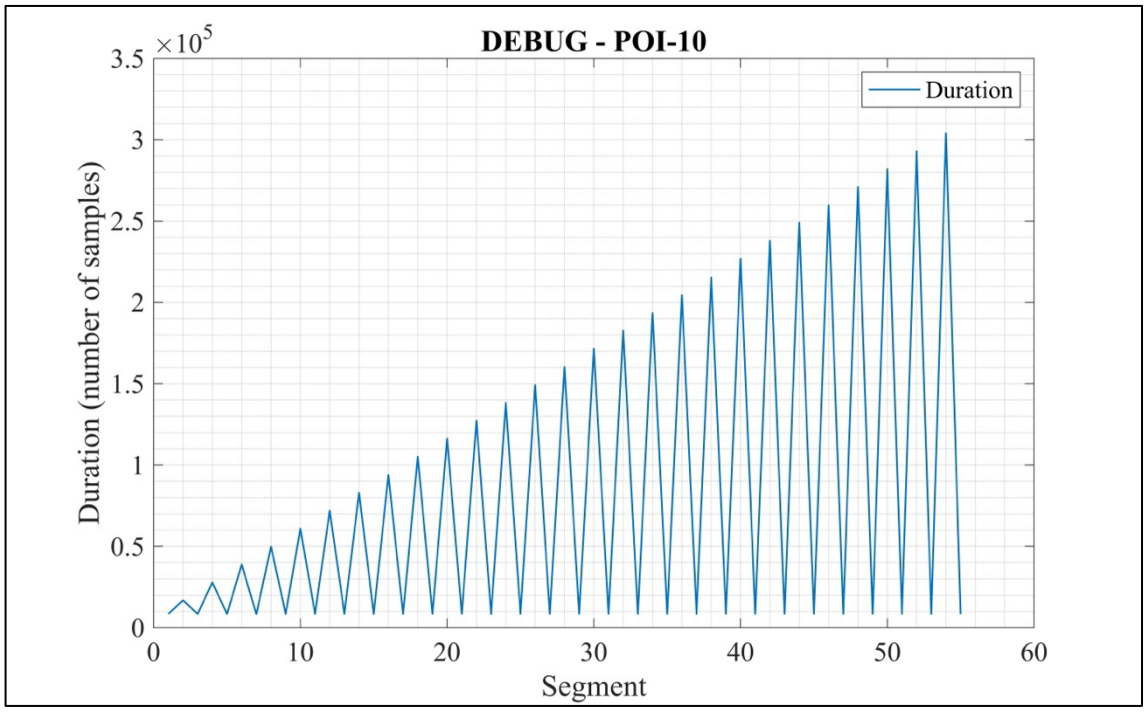

Figure 64 – DEBUG POI-10 for Test3

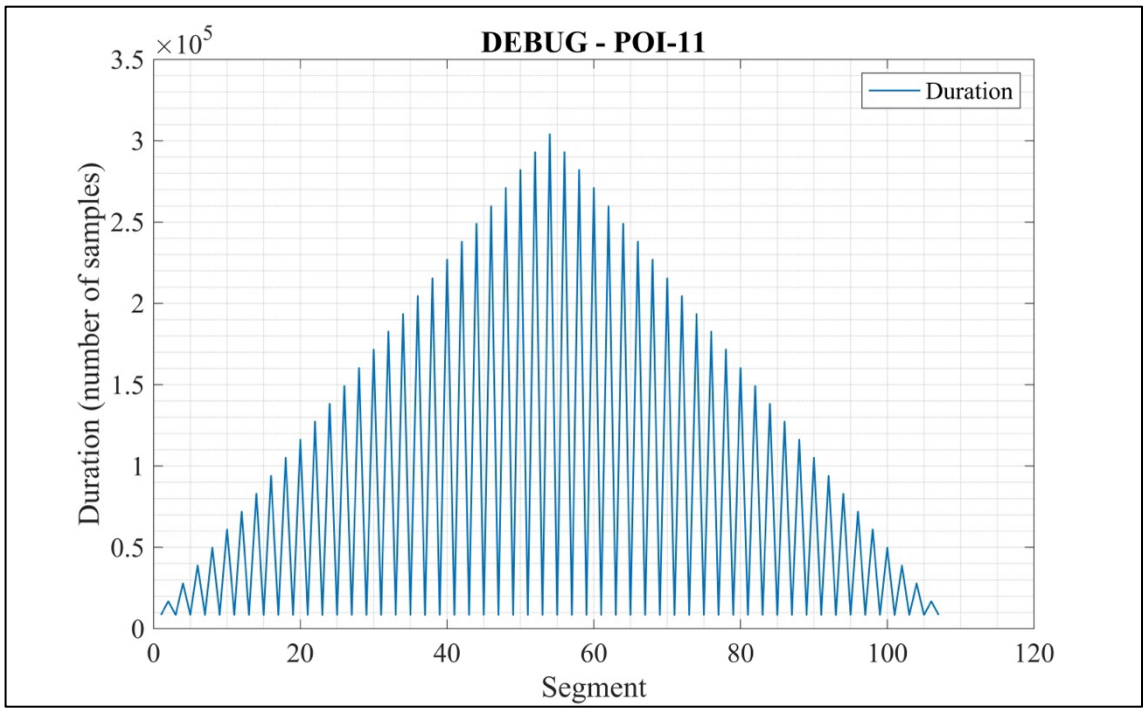

Figure 65 – DEBUG POI-11 for Test3

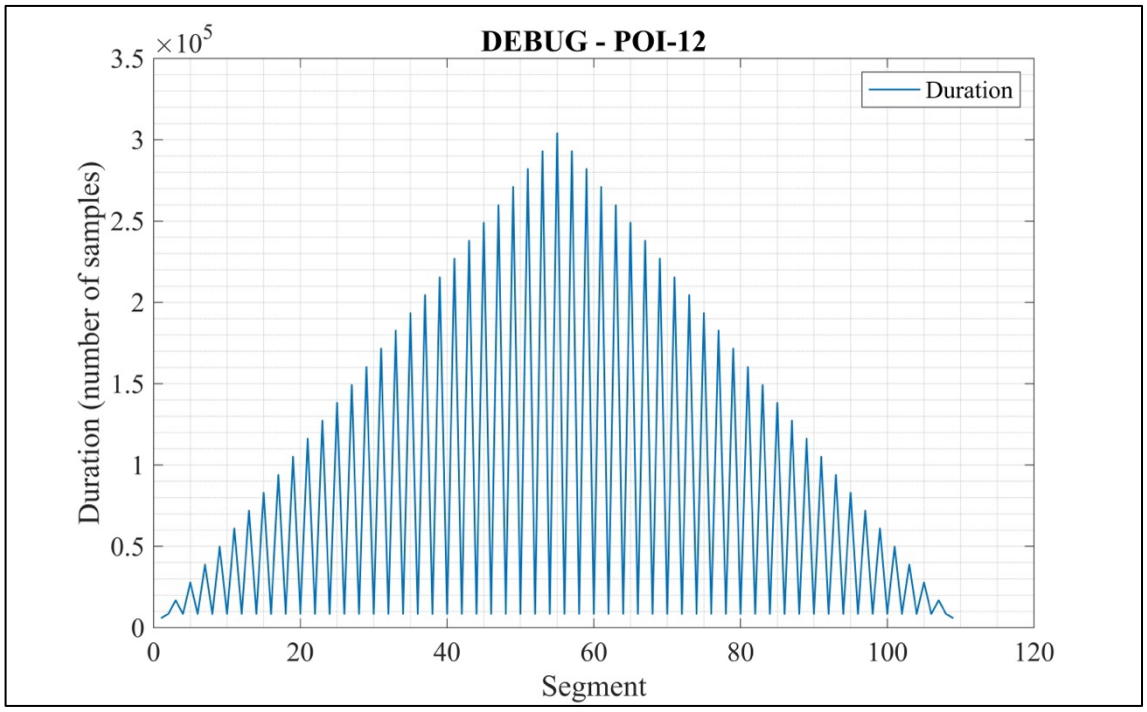

Figure 66 – DEBUG POI-12 for Test3

Printing parameters/process settings used in the slicing process of the workpiece g-code

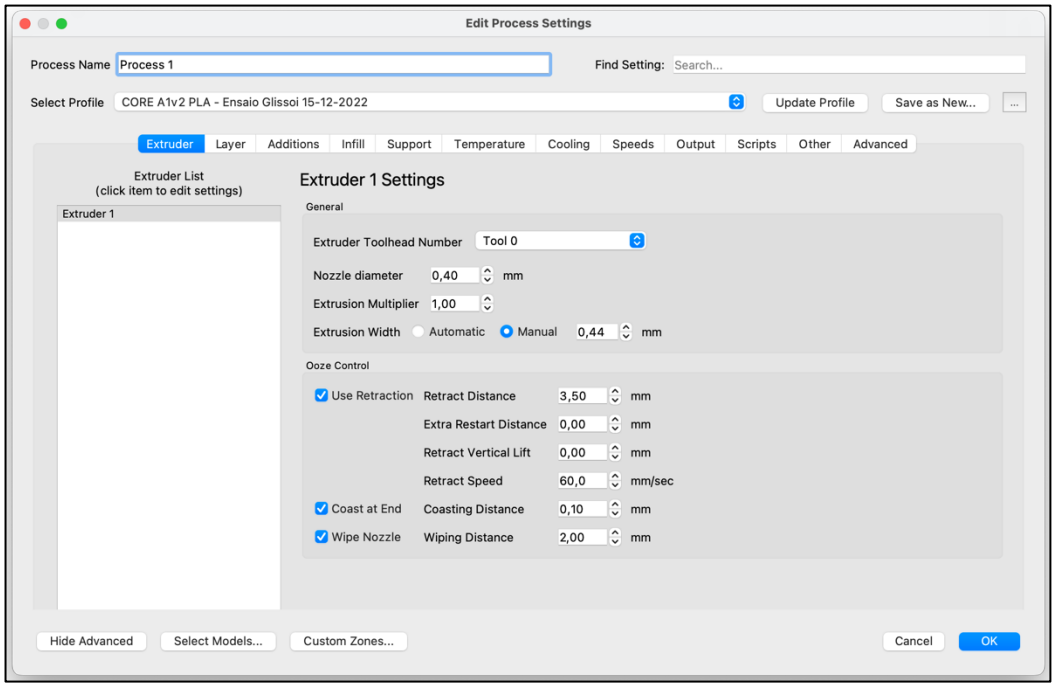

Figure 67 – 'Extruder' process parameters

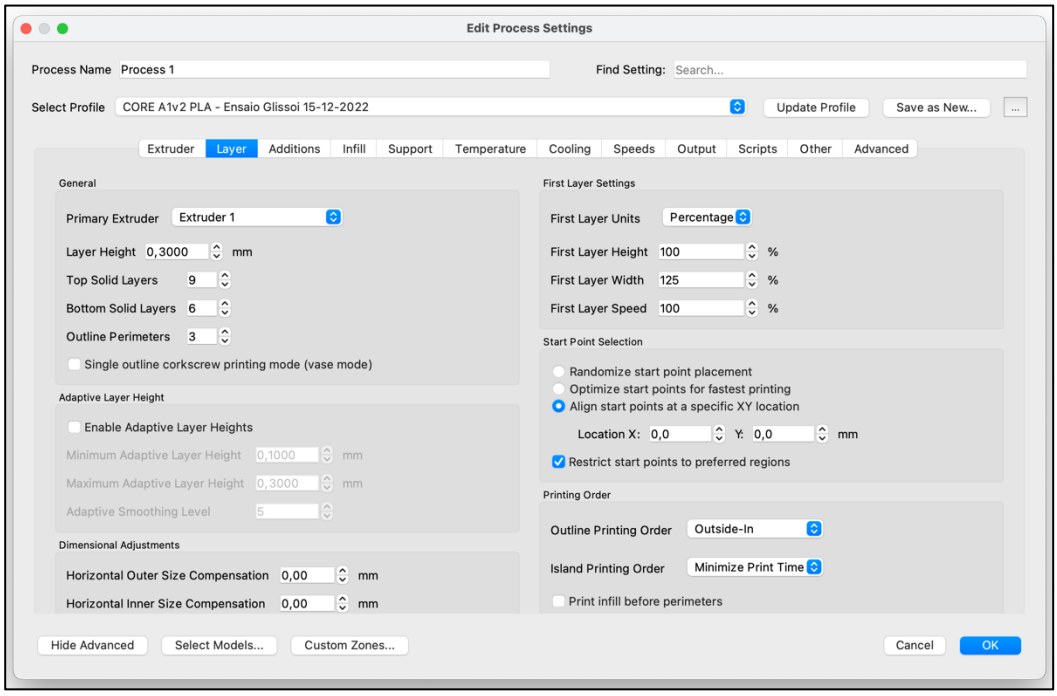

Figure 68 – 'Layer' process parameters

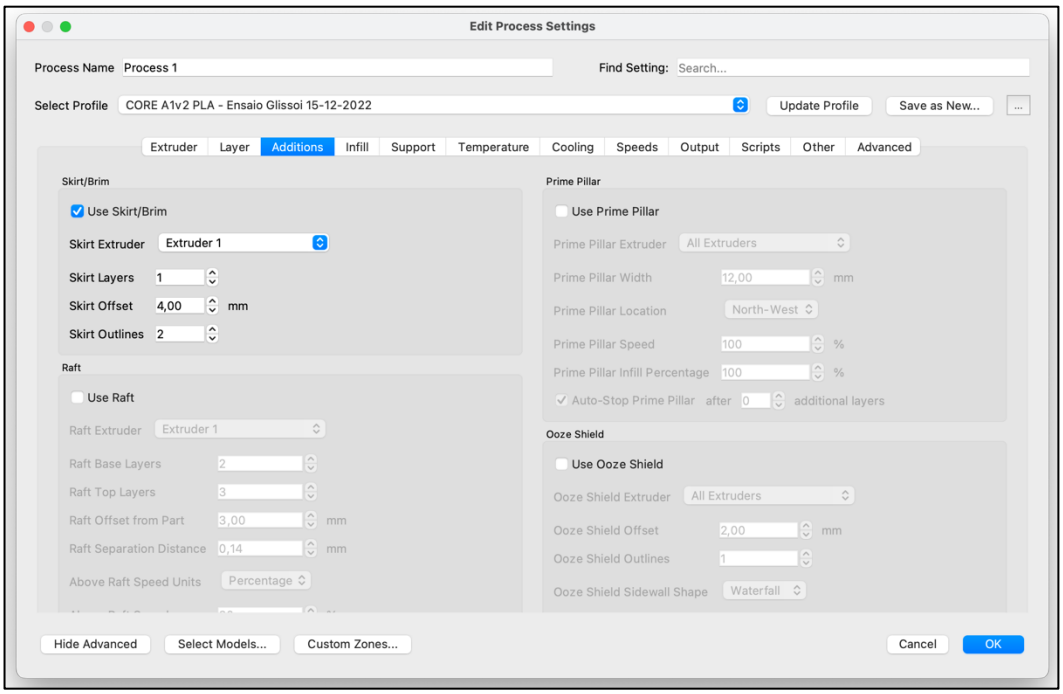

Figure 69 – ‘Additions’ process parameters

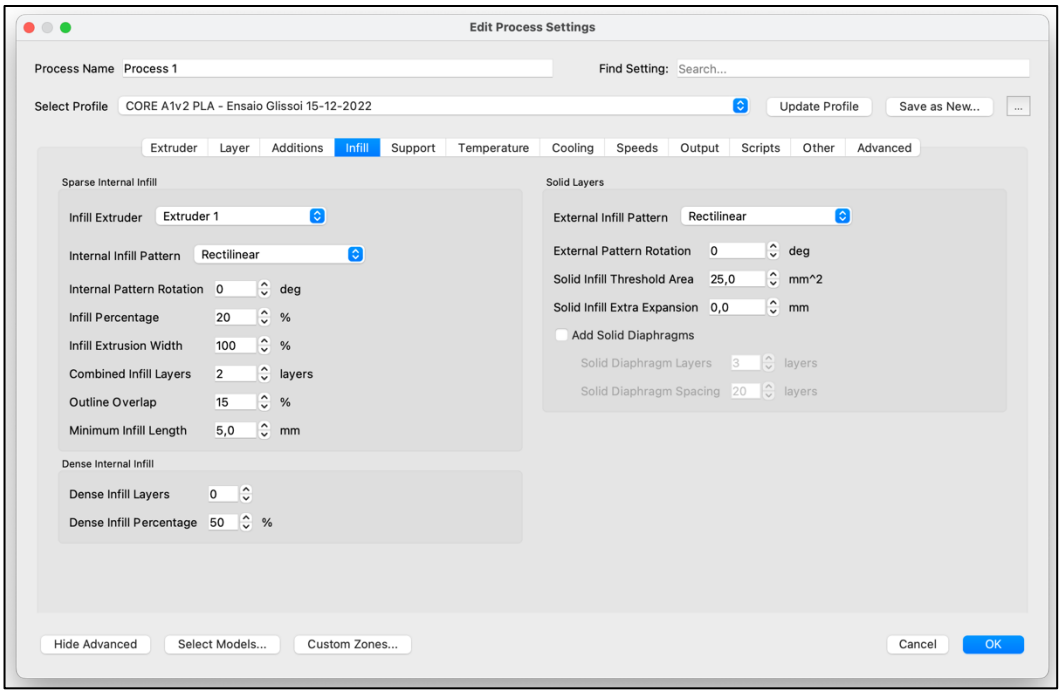

Figure 70 – ‘Infill’ process parameters

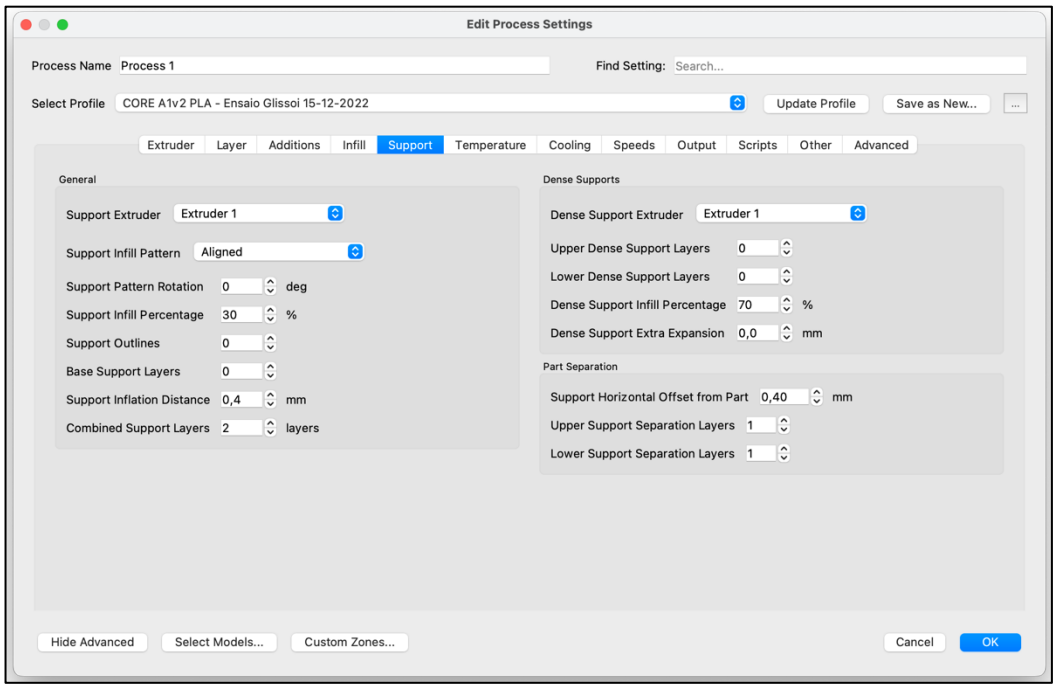

Figure 71 – ‘Support’ process parameters

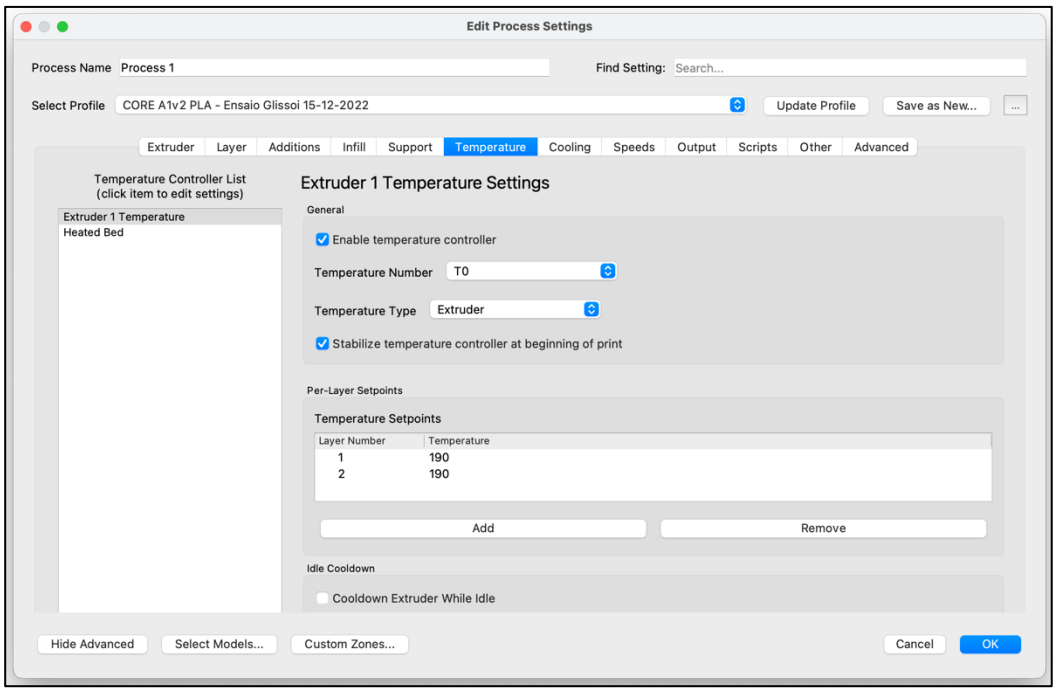

Figure 72 – ‘Temperature – Extruder’ process parameters

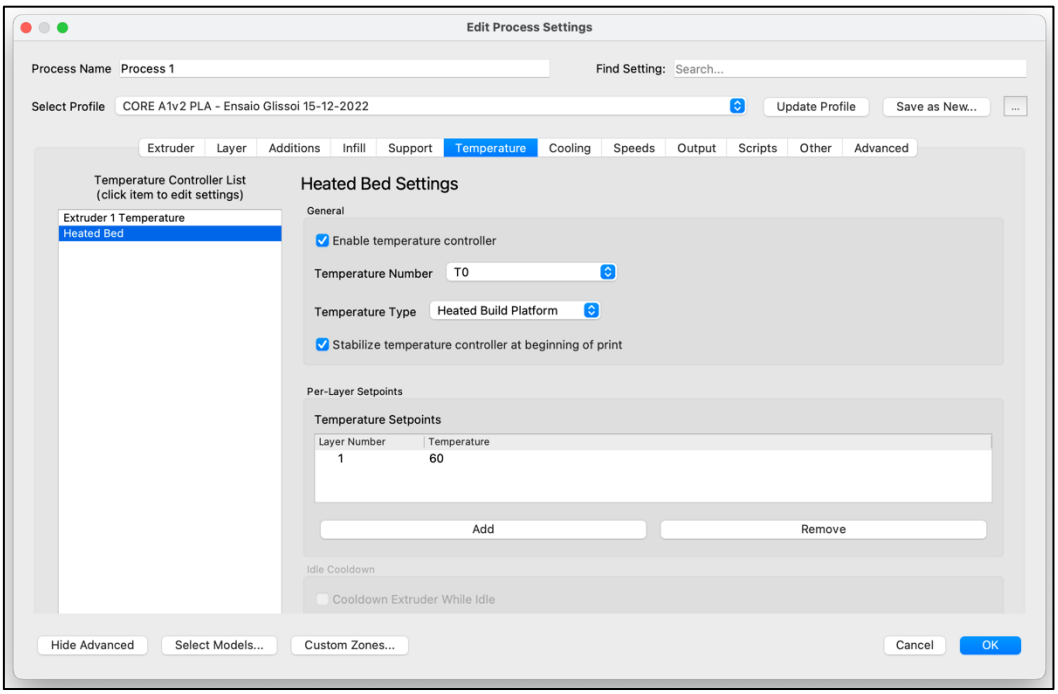

Figure 73 – ‘Temperature – Heated Bed’ process parameters

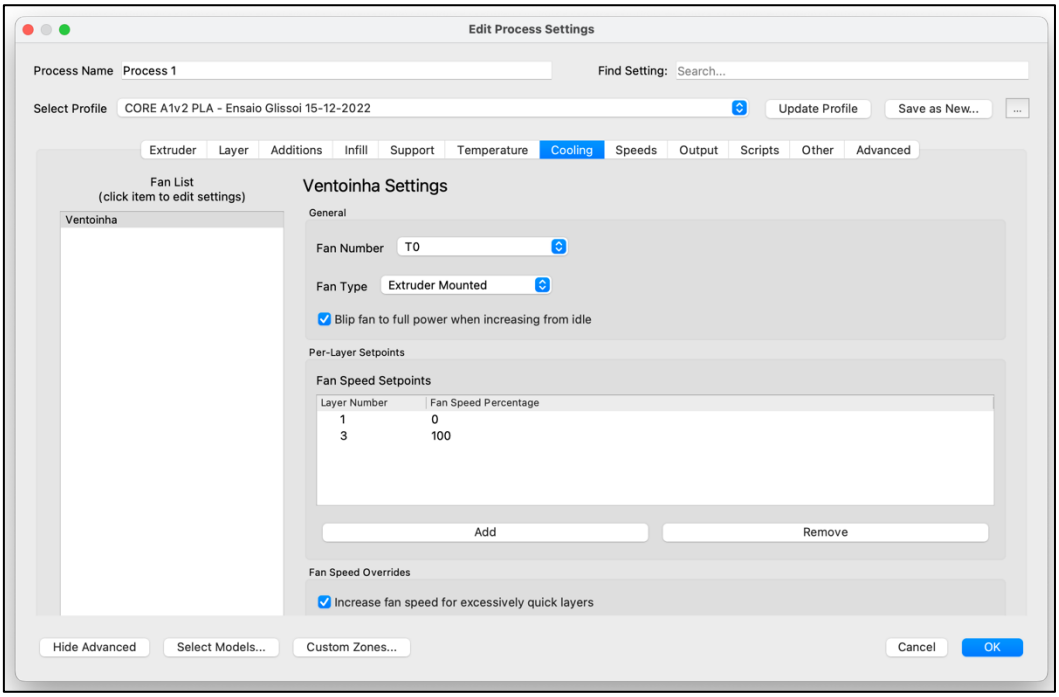

Figure 74 – ‘Cooling’ process parameters

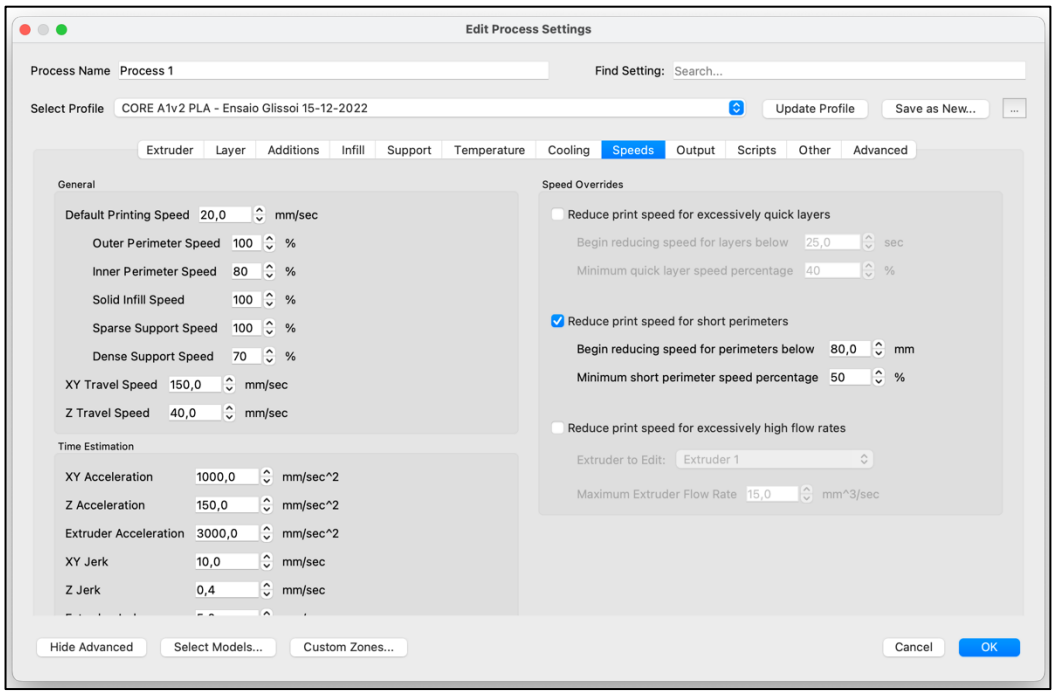

Figure 75 – ‘Speeds’ process parameters

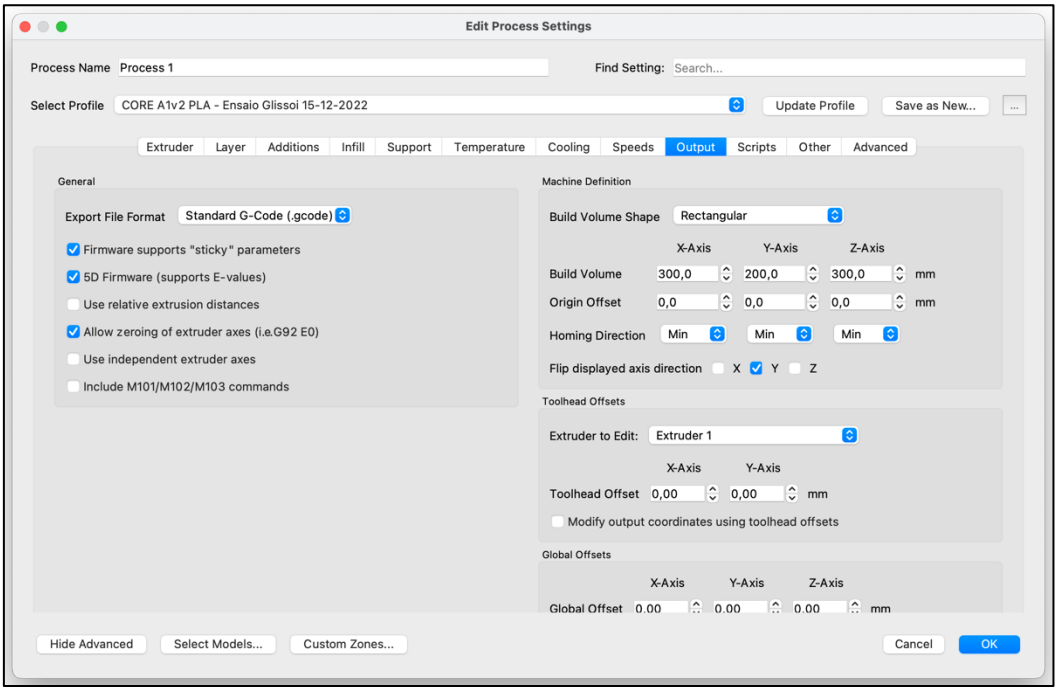

Figure 76 – ‘Output’ process parameters

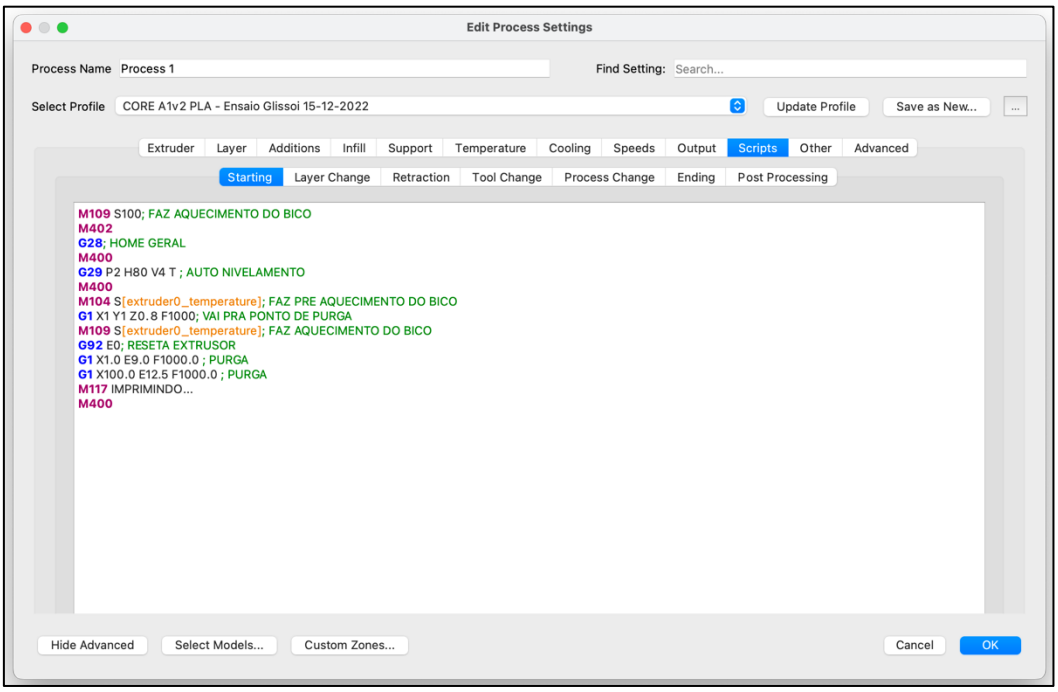

Figure 77 – ‘Scripts - Starting’ process parameters

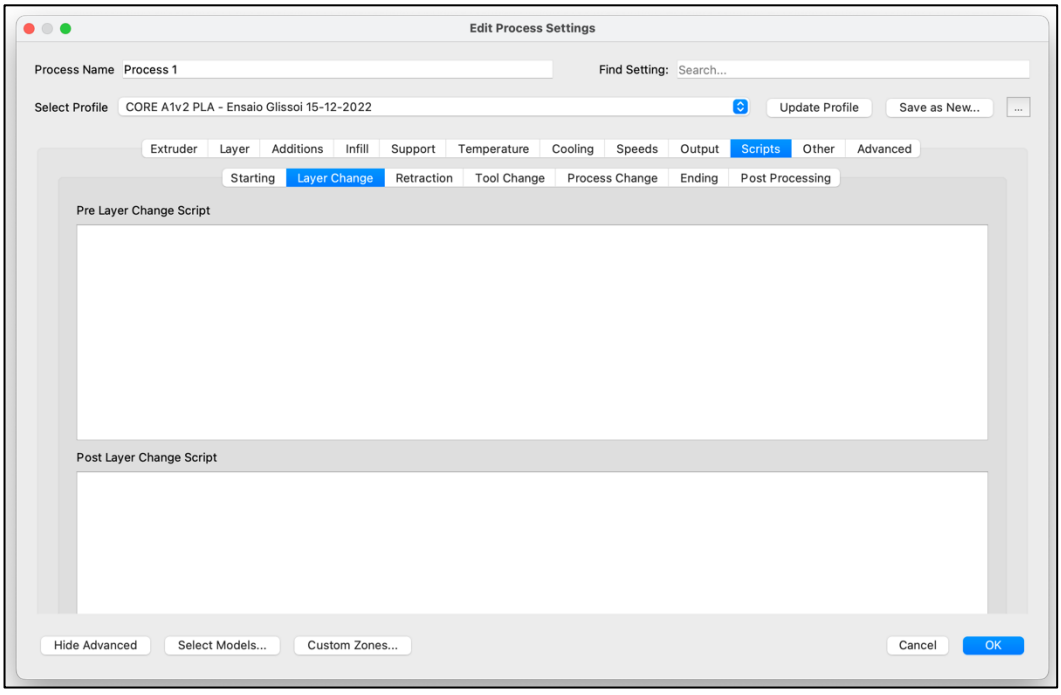

Figure 78 – ‘Scripts – Layer Change’ process parameters

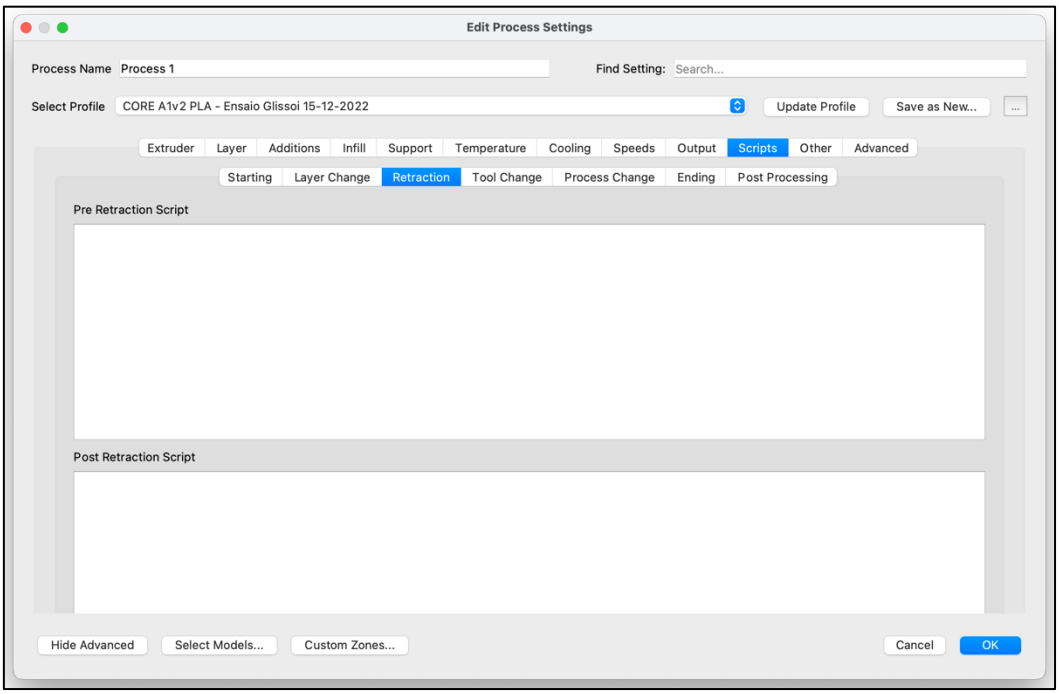

Figure 79 – ‘Scripts – Retraction’ process parameters

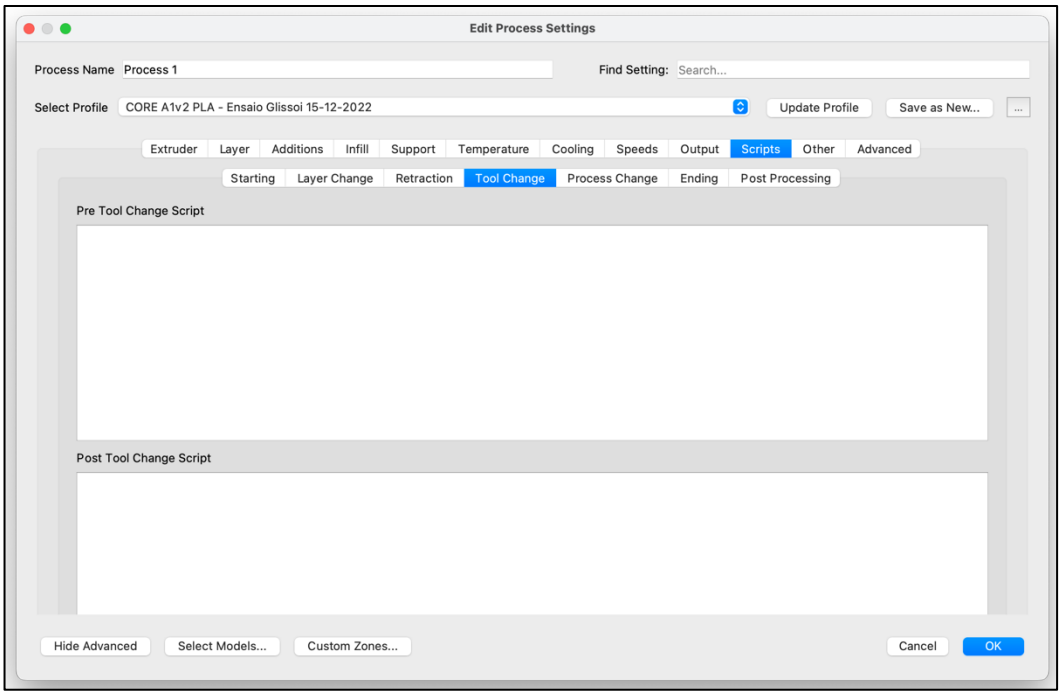

Figure 80 – ‘Scripts – Tool Change’ process parameters

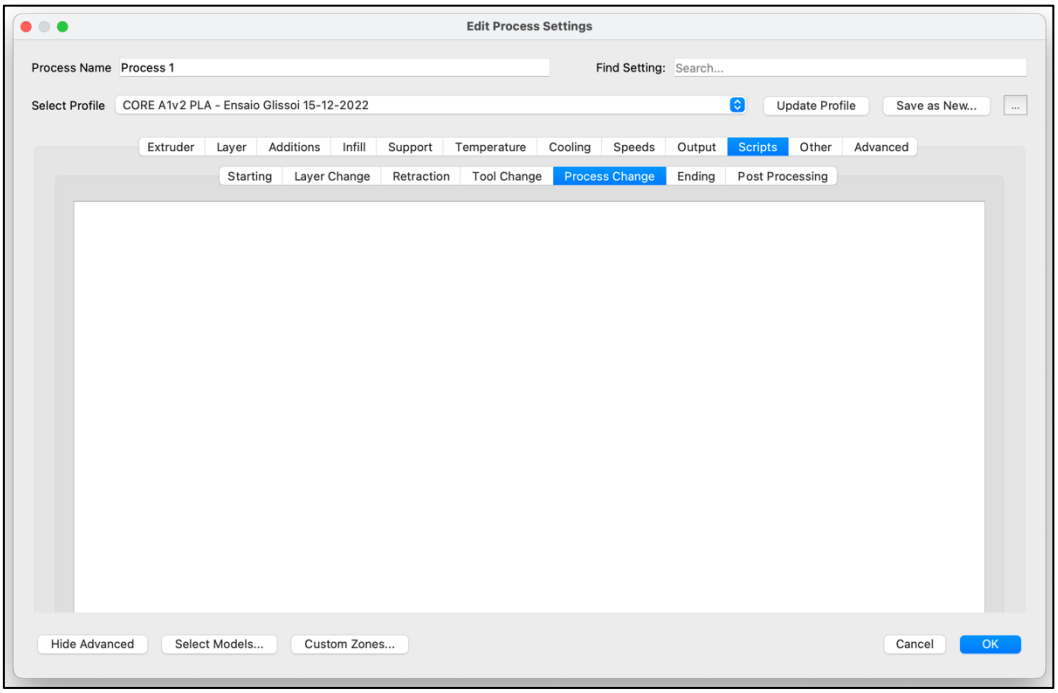

Figure 81 – ‘Scripts – Process Change’ process parameters

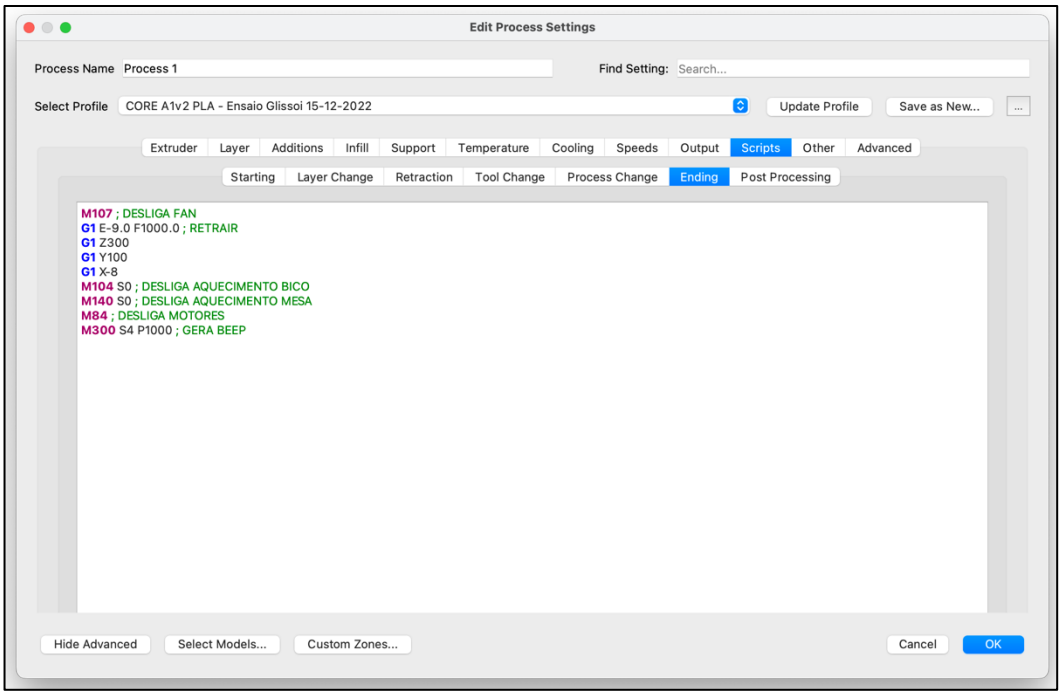

Figure 82 – ‘Scripts – Ending’ process parameters

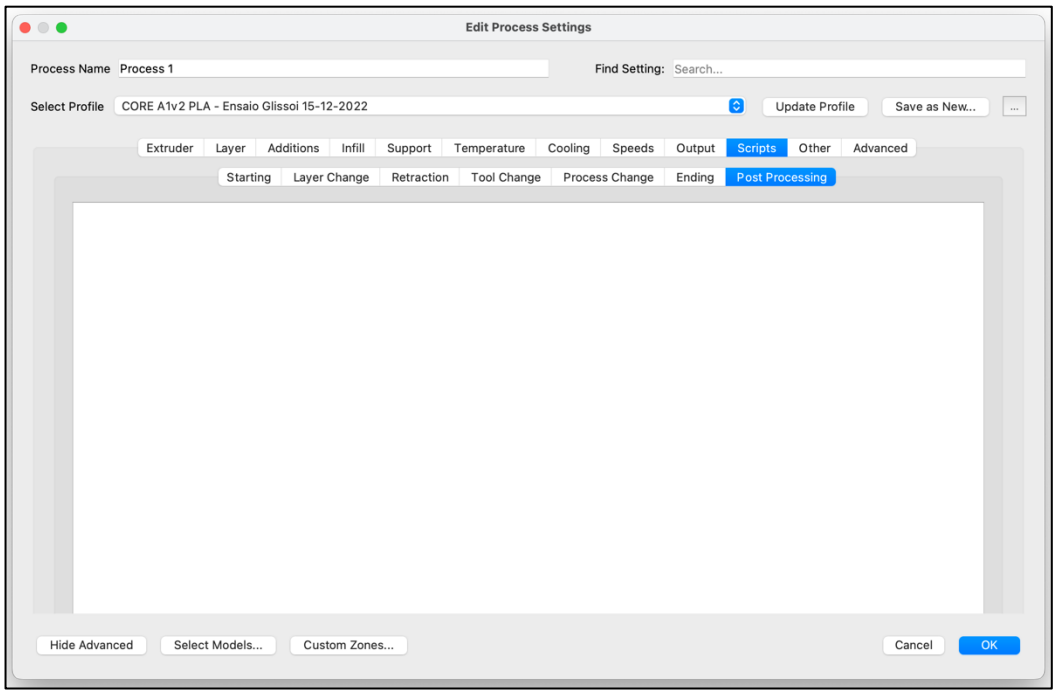

Figure 83 – ‘Scripts – Post Processing’ process parameters

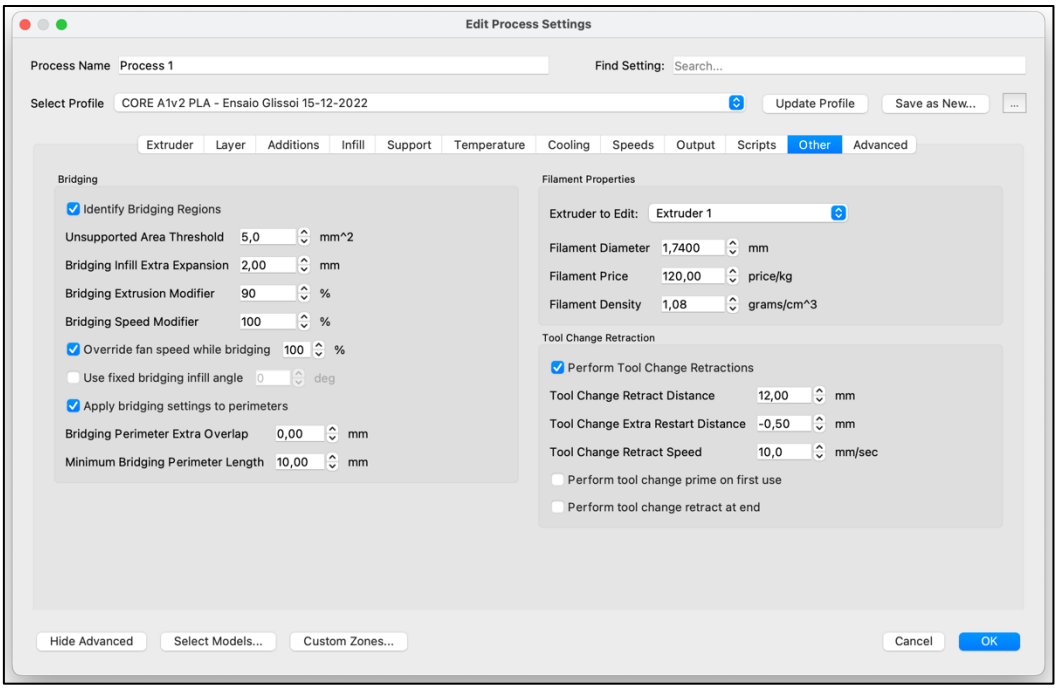

Figure 84 – ‘Other’ process parameters

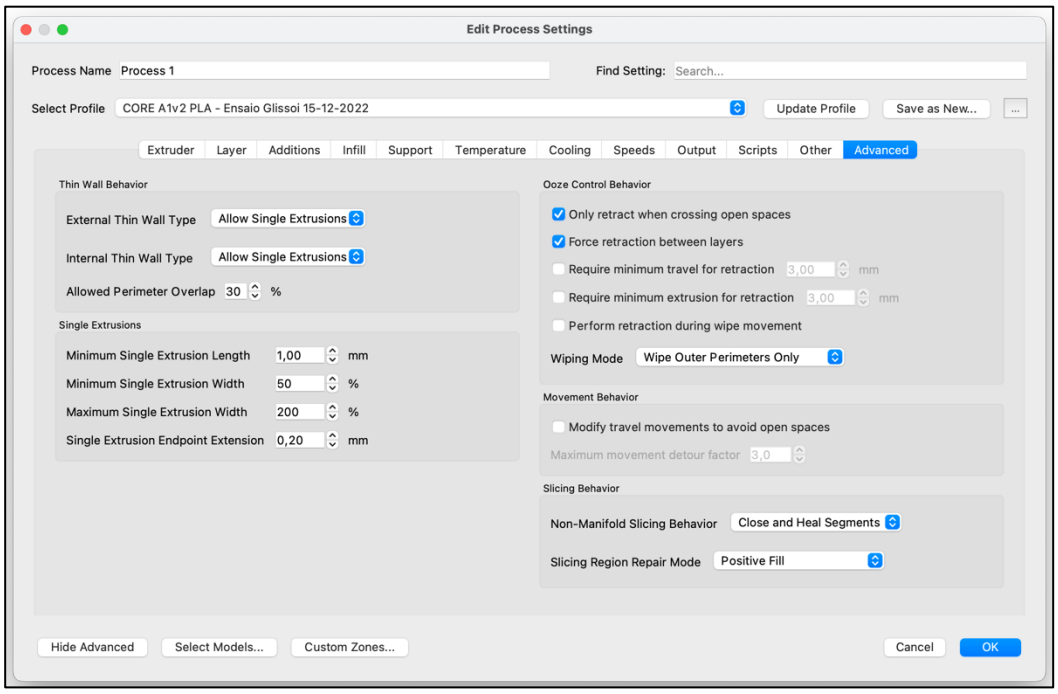

Figure 85 – ‘Advanced’ process parameters