Summary

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 1. 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.

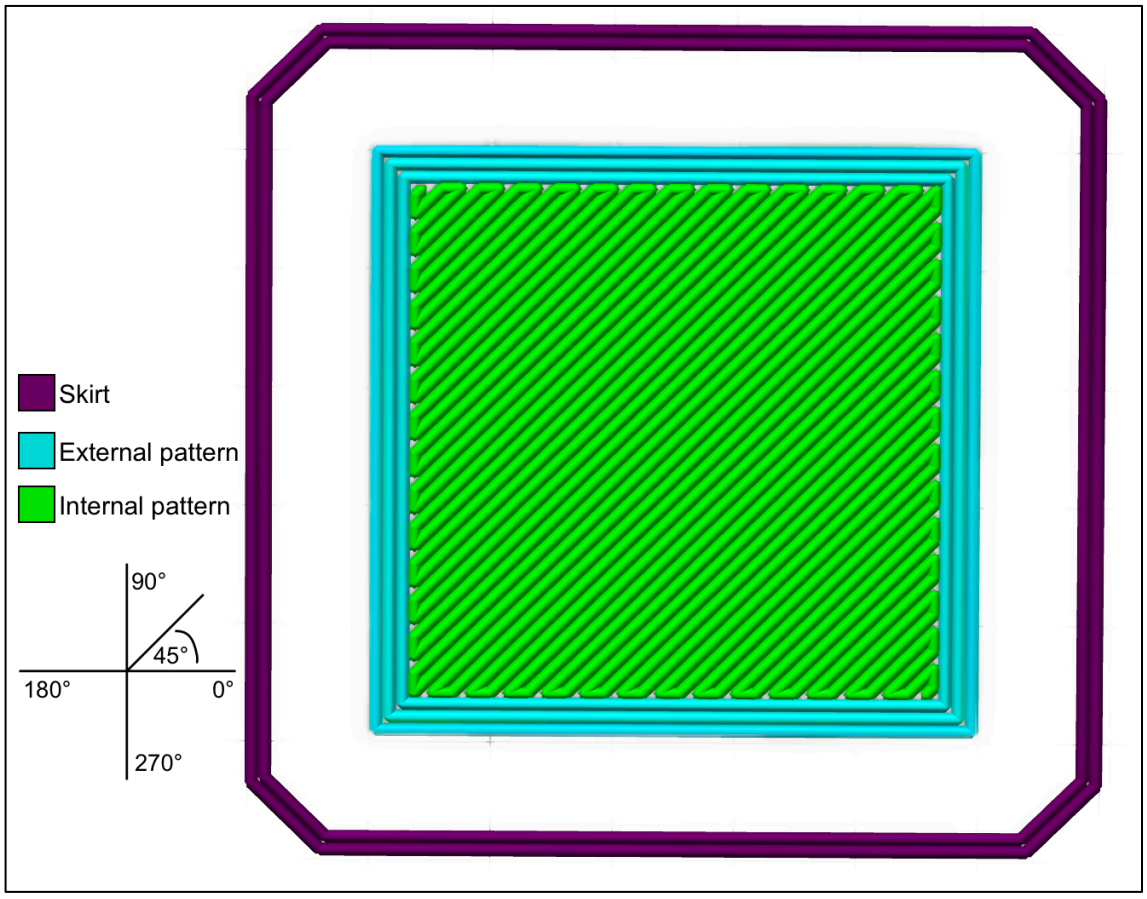

Figure 1 – Workpiece Elements of Fabrication

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 2, 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.

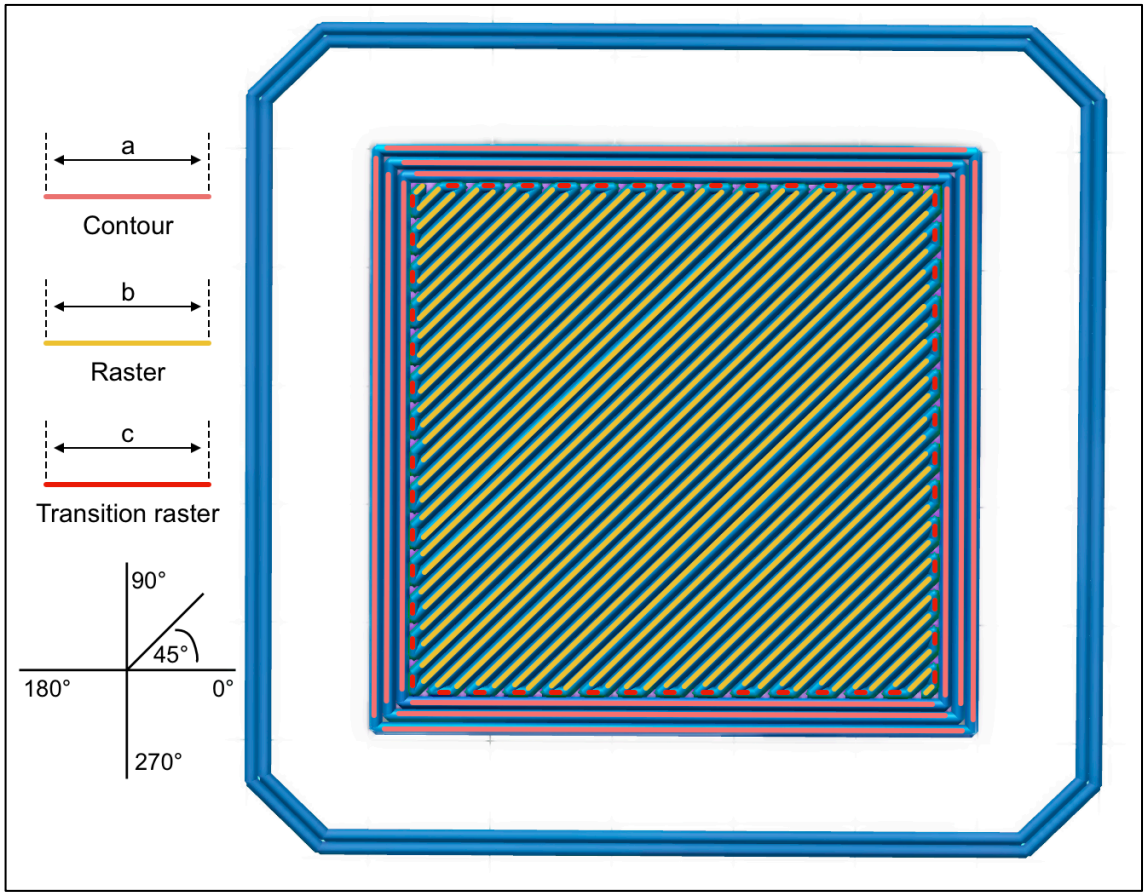

Figure 2 – Workpiece with identifications

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 2:

- 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 3:

- 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.

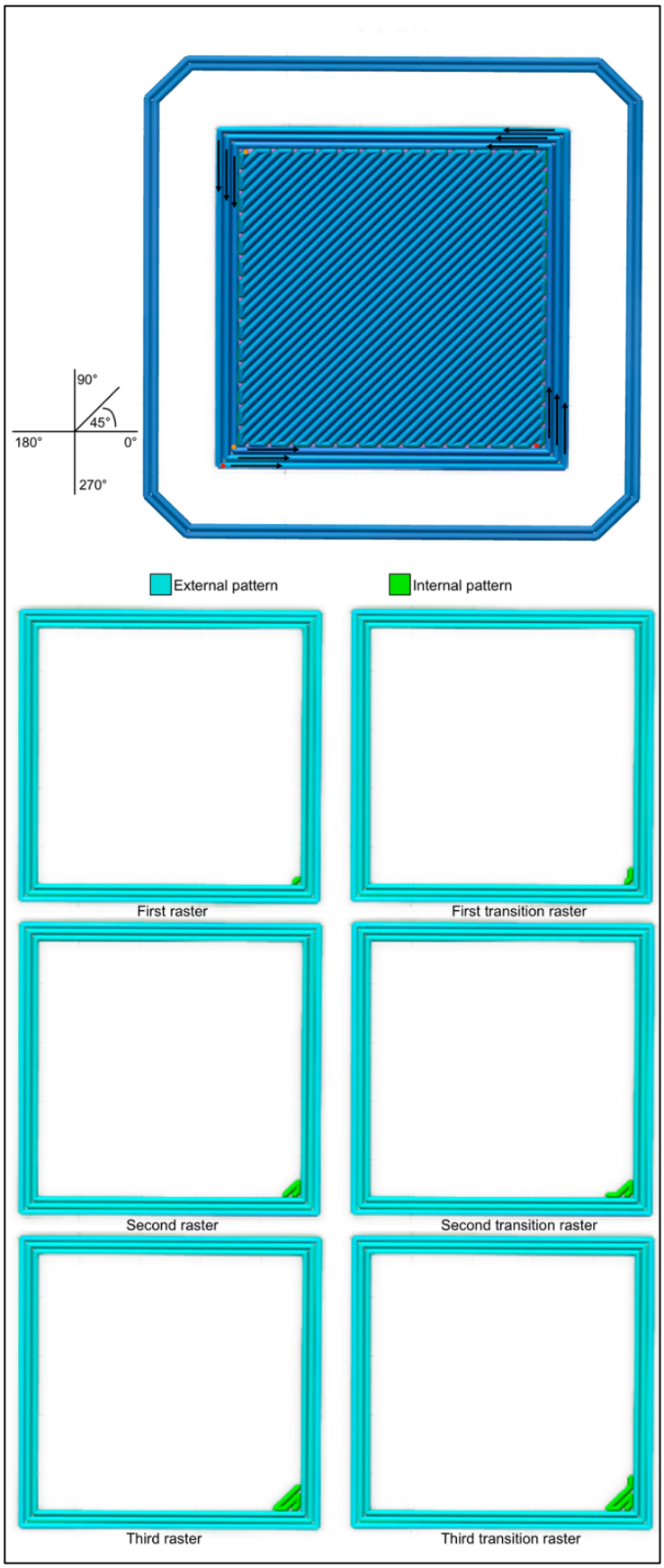

Figure 3 – Internal pattern behaviour

Considerations Based on Fabrication Movements

To carry out the fabrication of the external fill pattern, considering the geometry of the workpiece represented in Figure 2, some repositioning movements without filament deposition are required (Movements Without Deposition – MWD). These MWDs, represented in Figure 4, 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 4, 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 4, 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 4, 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 4, 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

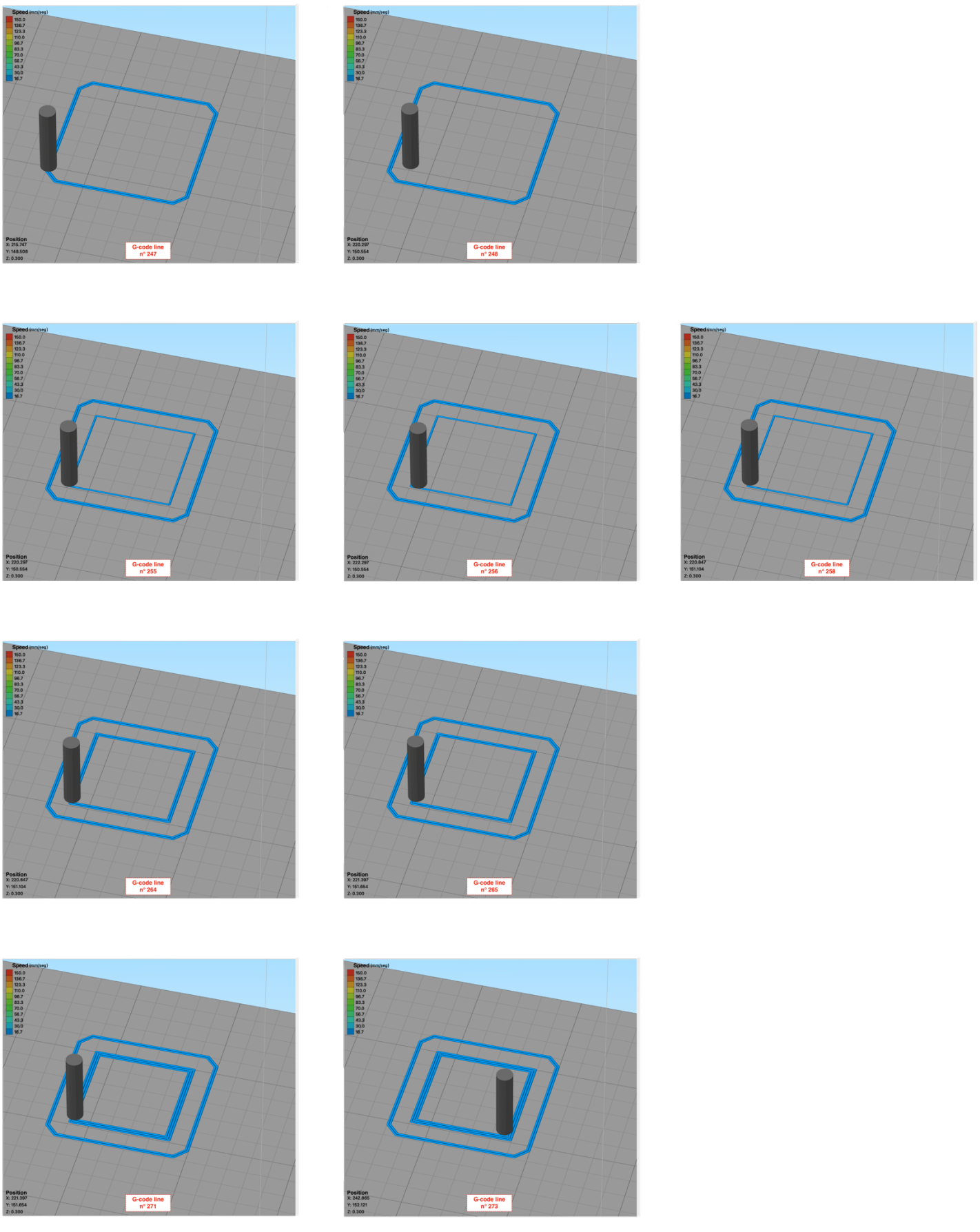

Figure 4 – Movement Without Deposition (MWD) Test1 detailing

Algorithm Segmentation Operation

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 23) and the Test2 dataset (Figure 35). However, it's 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 5 was applied, which uses a 2V threshold to define values of 0V or 5V.

```
111  dirXAdjusted = zeros(size(dirX));
112  dirYAdjusted = zeros(size(dirY));
113
114  % 1° Normalize the X and Y step motor control signals between 0V and 5V
115  for i = 1:length(dirX)
116      if dirX(i) > 2
117          dirXAdjusted(i) = 5;
118      else
119          dirXAdjusted(i) = 0;
120      end
121  end
122
123  for i = 1:length(dirY)
124      if dirY(i) > 2
125          dirYAdjusted(i) = 5;
126      else
127          dirYAdjusted(i) = 0;
128      end
129  end
```

Figure 5 – 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 24 (Test1 dataset) and Figure 36 (Test2 dataset).

To facilitate the representation of acoustic sensor signals alongside the control signals for direction, the code shown in Figure 6 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 6 – 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 25 (Test1 dataset) and Figure 37 (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 26, 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 27. 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 38, does not exhibit this phenomenon.

To verify this occurrence, the code shown in Figure 7 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);
```

Figure 7 – 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 relate the length of the fabrication lines (contour, raster, and raster transition) shown in Figure 2, with the duration of these fabrication lines in terms of sample numbers.

The duration is obtained from the code represented in Figure 8. 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.

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

Figure 8 – Code to obtain the duration matrix

The first column of the Duration is represented in Figure 28 (Test1 dataset) and in Figure 40 (Test2 dataset). Based on these duration values, I will identify the separation point between the external and internal fabrication patterns. To find this separation point, I will traverse the entire first column of Duration until I find a known duration value (‘durationReference’, which is the duration of line 12 fabrication). Then, I store the positions related to the start of the fabrication for lines 10, 11, and 12. I know these three should be very similar, so I run a test on lines 10 and 11. If they are within the same interval, the end of the external pattern is identified. Otherwise, I return to look for this pattern in the duration values.

```
% 5° Find the separation point
durationReference = 150e3; % known duration for the last contour printing

for i = 1:length(Duration(:,1))

    resultSep = determin_separation_point(Duration, i, durationReference);

    if resultSep == true
        patternVariationPoint = Duration(i,2);
        break;
    else
        clear resultSep
    end
end
```

Figure 9 – Code to find the separation point

Having identified the separation point, I wish to find the indices related to the start of the contour lines. These indices are stored in the variable ‘externalLines’. With these starting indices, I aim to obtain a temporary matrix related to the contour lines, named ‘indexContourTemp’. 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 10 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
```

Figure 10 – Code to obtain the temporary contour lines matrix

Having obtained the temporary matrix for the contour lines, I now wish 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, we should refer to the information presented in Figure 4 and Table 1, as discussed earlier.

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 11 in more detail, which represents the debug point POI-6.

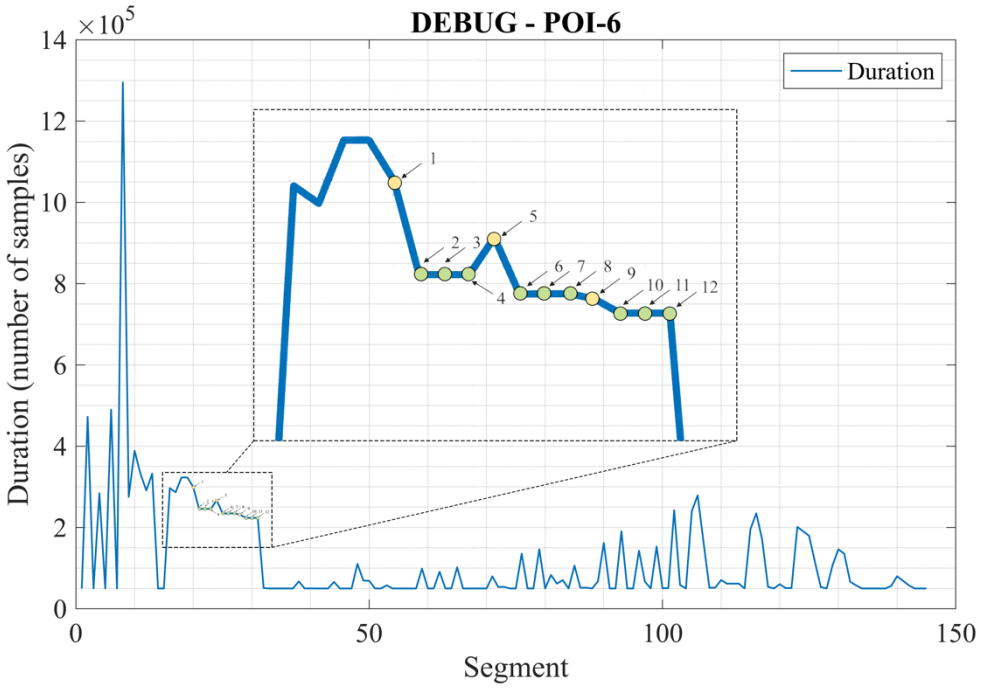

Figure 11 – Test1 Debug POI-6 MWD's detailing

This phenomenon was observed for the Test1 and Test2 datasets, as shown in Figure 40. 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 52, 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 12 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(2,1) indexCountourTemp(3,1)...
    indexCountourTemp(4,1)]),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(12,1)]),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];
```

Figure 12 – 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 12) 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 26 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 13 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.

```
241  % 8° Obtaining first duration matrix for the internal pattern
242  % There is no need to check for the problem of too many consecutive transition lines here
243  % since there are very small line segments in the internal pattern. On the cobtrary,
244  % this could cause some issues. So the result_problem value is set to false.
245  resultProblem = false;
246
247  % Obtaining duration matrix
248  Duration = obtainDuration(sensorSignalNormalized, ...
249      dirXAdjustedNormalized,...
250      dirYAdjustedNormalized, ...
251      resultProblem);
```

Figure 13 – 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 29, 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 29 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 8,500 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 14 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.

```
% 9° Obtaining second duration matrix for the internal pattern

% Filtering the duration matrix to remove the small line segments
indexValuesAbv8000 = 1;
Duration2 = zeros(3);
alterationDuration = 8000;
for i = 1:length(Duration(:,1))
    if Duration(i,1) > alterationDuration
        Duration2(indexValuesAbv8000,:) = Duration(i,:);
        indexValuesAbv8000 = indexValuesAbv8000 + 1;
    end
end

clear Duration
```

Figure 14 – 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 15, with the exception of regions with abnormal behavior (abnormal).

In the abnormal regions identified in Figure 15, 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, as verified in Figure 41 and Figure 53. 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.

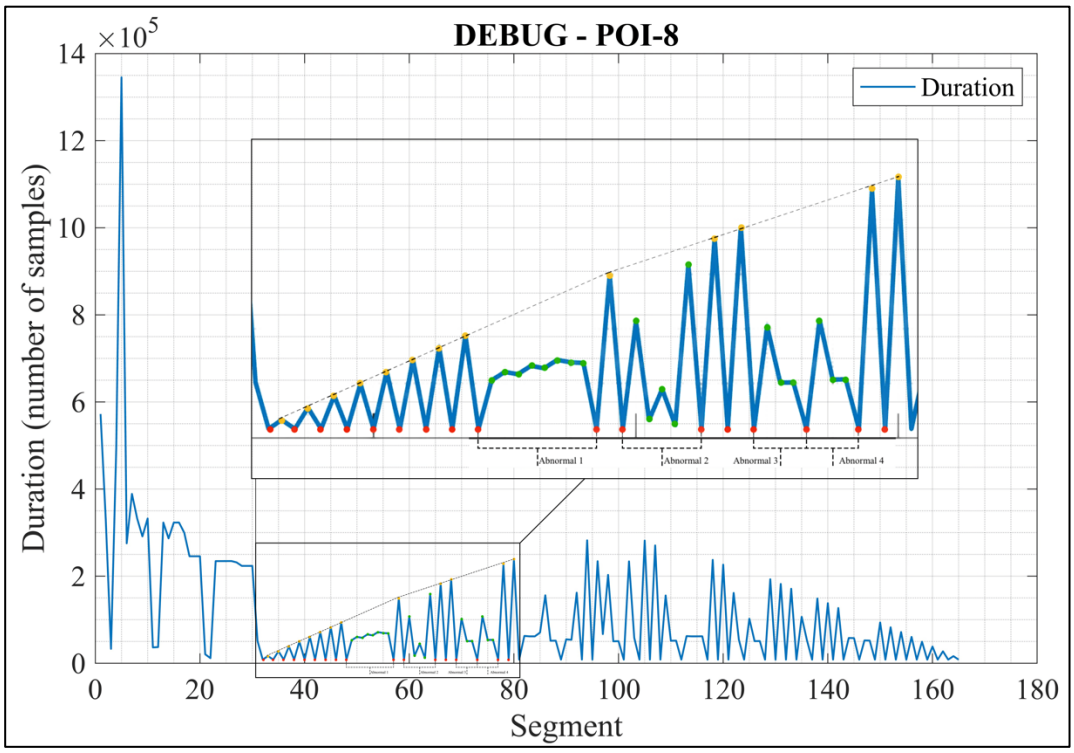

Figure 15 – Test1 Debug POI-8 MWD's detailing

With this understanding, the code shown in Figure 16 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 linesUdr9000. 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 15). 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 one to identify 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

linesUdr9000 = find(Duration2(:,1) < 9000);
Duration3 = zeros(3);
previousPeak = 0;
previousPeak2 = 0;
middleRasterDuration = 310e3;

for i = 1:length (linesUdr9000)
    if i == length (linesUdr9000)
        break;
    end
    initialIndex = linesUdr9000(i);
    finalIndex = linesUdr9000(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 = isAnormal( ...
            Duration2, ...
            initialIndex, ...
            finalIndex, ...
            previousPeak);
        Duration3 = [Duration3(1:end-1,:); tempDuration];
        clear tempDuration
    end
end

clear Duration2
```

Figure 16 – 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 isAnormal.

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 31 represents the duration values corresponding to raster lines and transition raster lines. It is possible to observe that segment 55 presents the manufacturing of the middle raster line, and segment 57 presents the manufacturing of the first raster line where the behavior changes from increasing to decreasing.

After this, the code shown in Figure 17 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 32 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.

```
323  % 11° Obtaining fourth duration matrix for the internal pattern
324  indexPreviousPeak2 = find (Duration3(:,1) == previousPeak2);
325  Duration3_v2 = Duration3(1:end-(end-indexPreviousPeak2)+1,:);
```

Figure 17 – Code to obtain the fourth duration matrix for the internal pattern

The code shown in Figure 18 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 33 displays the duration values for the external fill pattern fabrications, excluding the first and last raster lines. This can be observed by analyzing Figure 33, 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
```

Figure 18 – Code to obtain the fifth duration matrix for the internal pattern

The code shown in Figure 19 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's 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 34 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)-11e3;
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)-11e3;
Duration4_v2(length(Duration4(:,1))+2,2) = Duration4_v2(length(Duration4(:,1))+2,1) + Duration4_v2(length(Duration4(:,1))+2,3);
% \ 13°

clear Duration4
```

Figure 19 – Code to obtain the sixth duration matrix for the internal pattern

The code shown in Figure 20 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) < 8000 || Duration4_v2(j,1) > 9000
        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(12,3);
contourToRasterReposition(1,2) = Duration4_v2(1,3);
contourToRasterReposition(1,1) = contourToRasterReposition(1,2) -...
    contourToRasterReposition(1,3);

indexContourAlt = indexContour;

for i = 1:1:12
    indexContourAlt(i,2) = indexContour(i,2);
    indexContourAlt(i,3) = indexContour(i,3)-5;
end

if length(index_raster) < 55
    [index_raster,index_trans_raster] =...
        adjust_internal(index_raster,index_trans_raster);

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));
```

Figure 20 – Code to obtain the index values from the last duration matrix

The code shown in Figure 21 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');

if testSegmentChoice == true
    resultReposition = [contourRepositions; contourToRasterReposition];
    resultContour = indexContour;
    resultRaster = index_raster;
    resultTransitionRaster = index_trans_raster;
end

testSegmentChoice = strcmp(segmentationChoice,'segments');

if testSegmentChoice == true
    resultReposition = gen_signal_segments(sensorSignalNormalized,...
        contourRepositions);
    resultReposition(1,4) = 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);
end
```

Figure 21 – Code to obtain the segmentation results

The code shown in Figure 22 outlines the procedure for storing variables and generating Figures according to the user's choices.

```
% 16° Save the results and generate the graphs

if saveChoice == 'Y'
    save_files(resultReposition, resultContour, resultRaster,...
        resultTransitionRaster, segmentationChoice, signalIdentifier);
end

if graphical == 'Y'
    gen_graph(signalReposition, sensorSignalNormalized, Fs,...
        signalIdentifier, signalContour, signalRaster,...
        signalTransRaster, index_raster, figureChoice);
end
```

Figure 22 – Code to save the results and generate the graphs

DEBUG – Test1 dataset

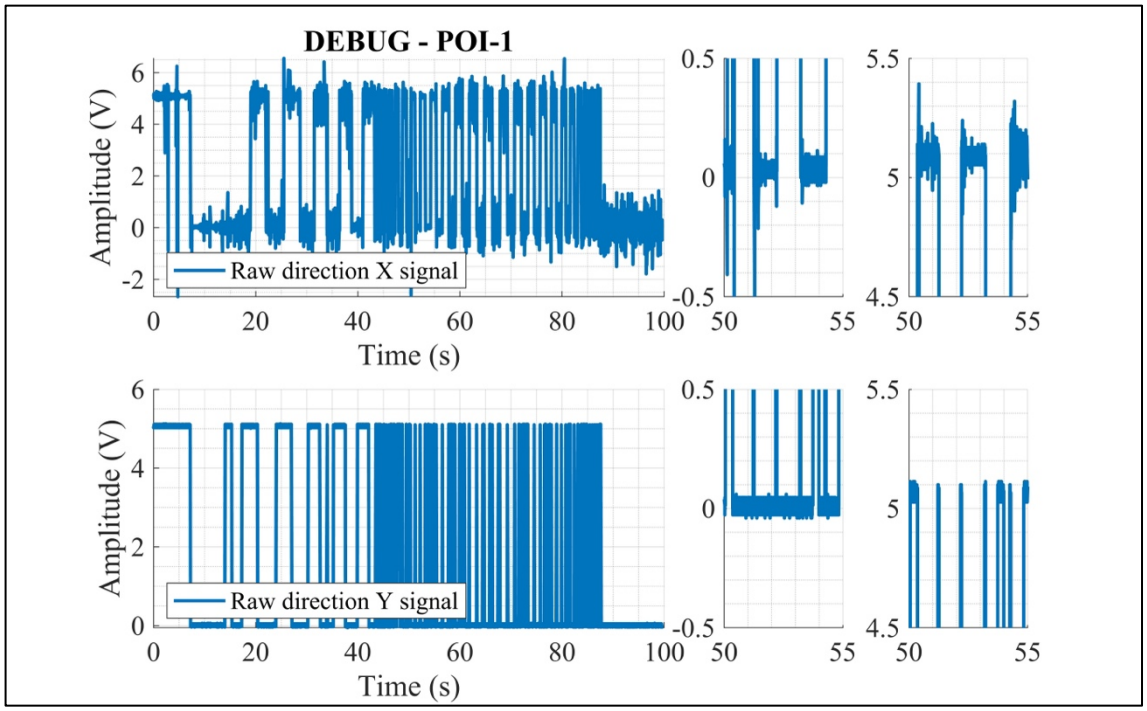

Figure 23 – DEBUG POI-1 for Test1

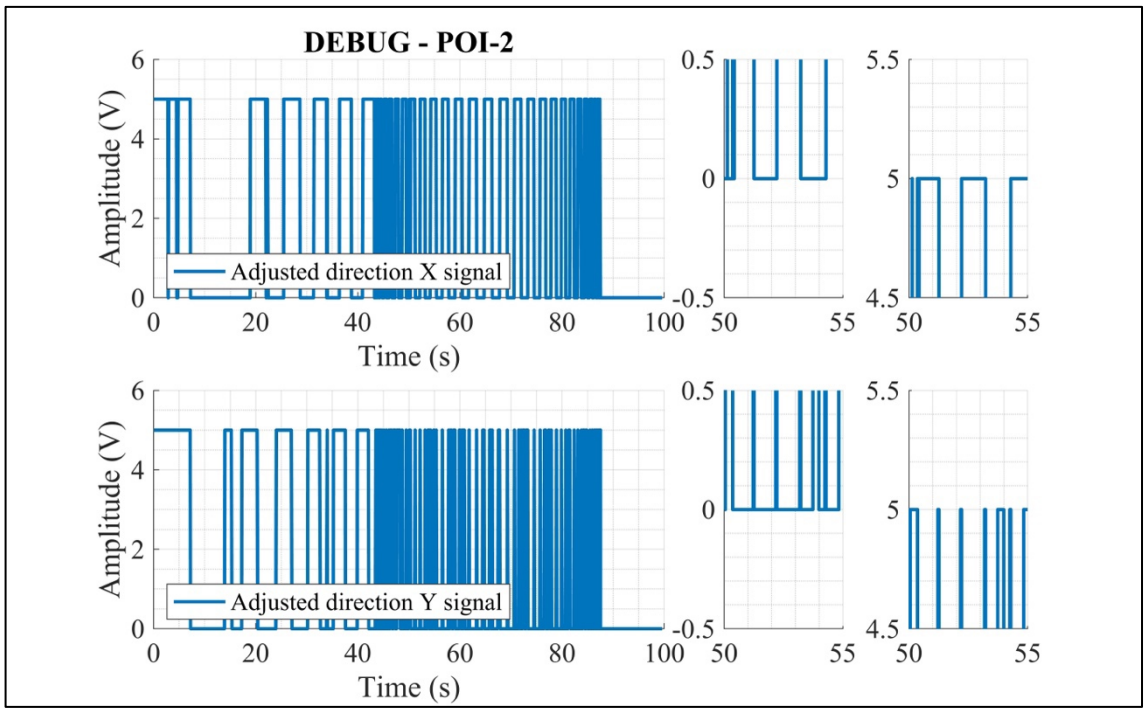

Figure 24 – DEBUG POI-2 for Test1

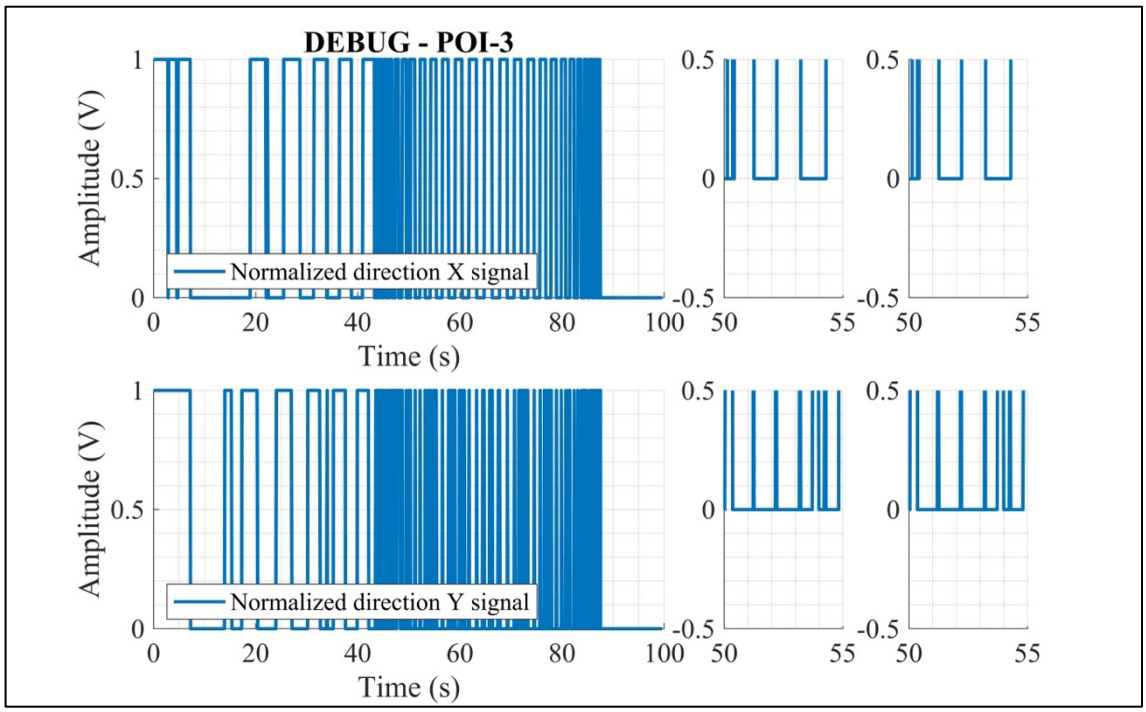

Figure 25 – DEBUG POI-3 for Test1

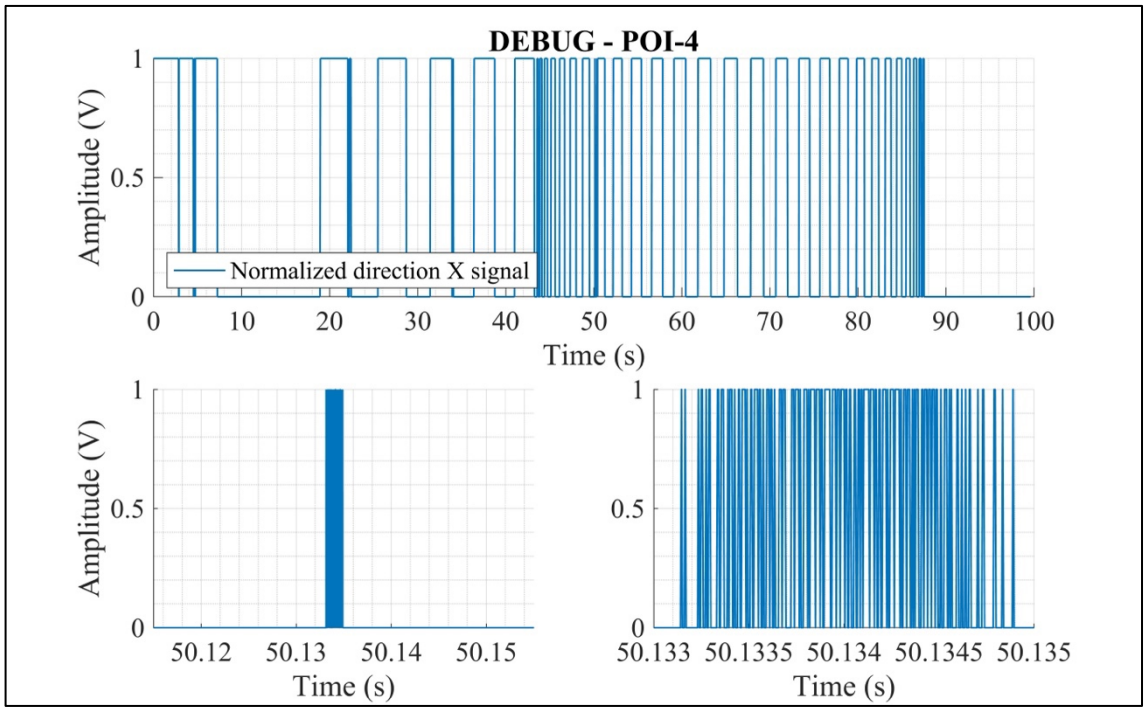

Figure 26 – DEBUG POI-4 for Test1

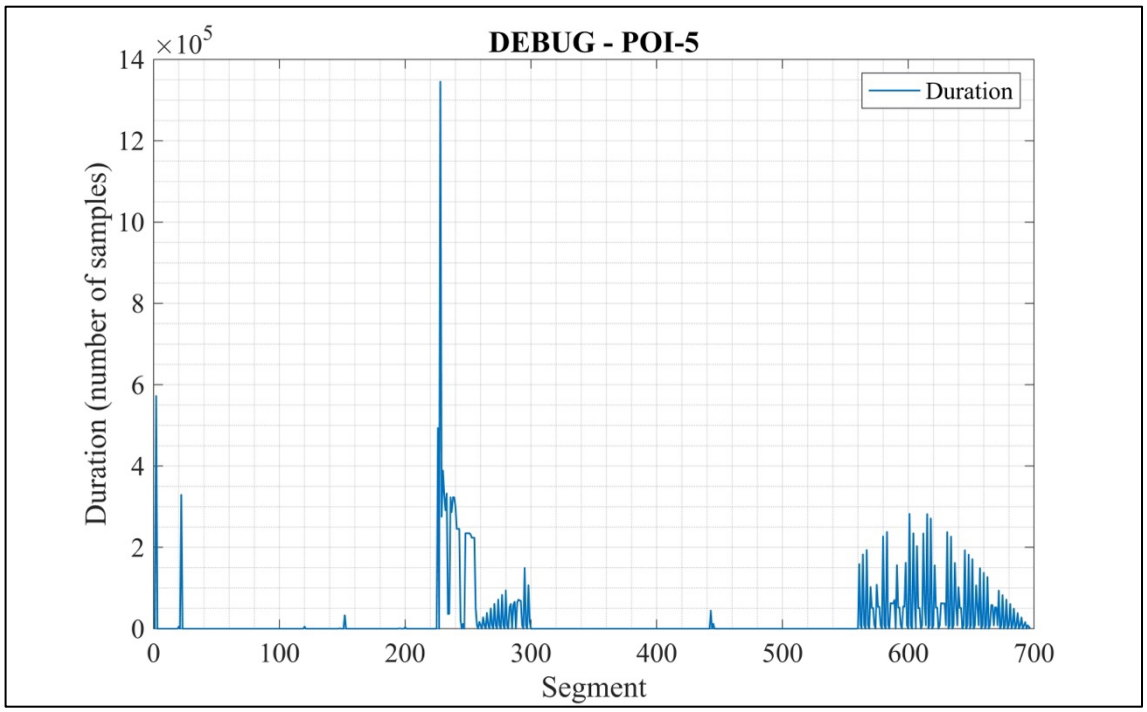

Figure 27 – DEBUG POI-5 for Test1

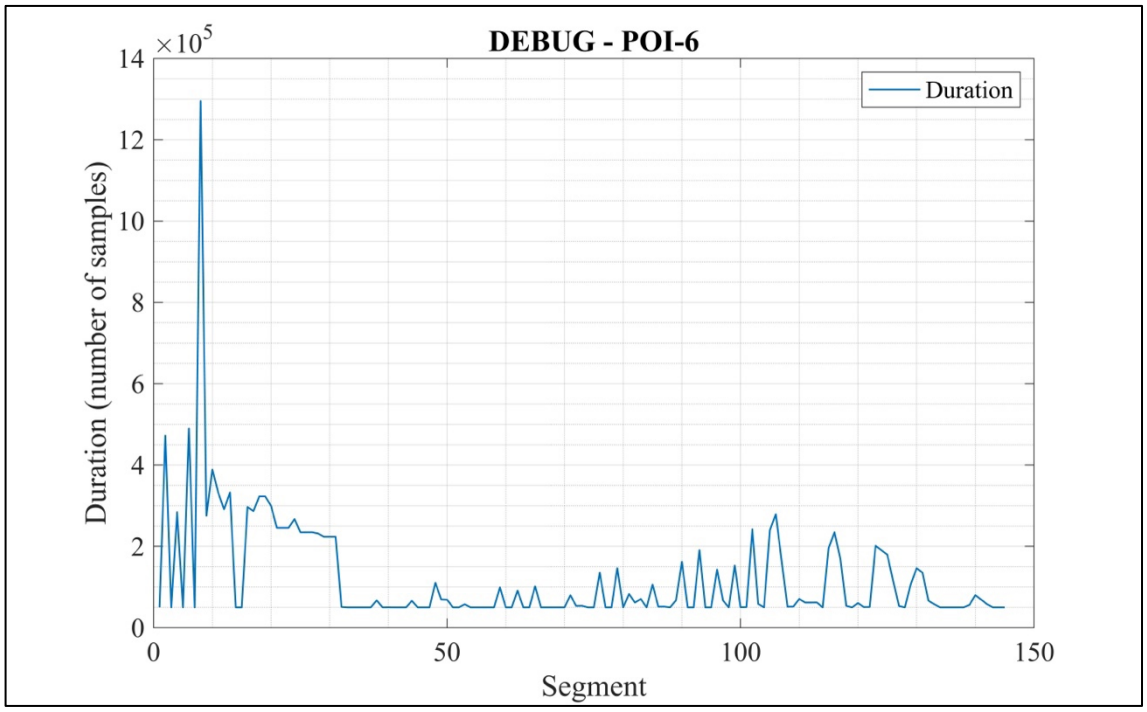

Figure 28 – DEBUG POI-6 for Test1

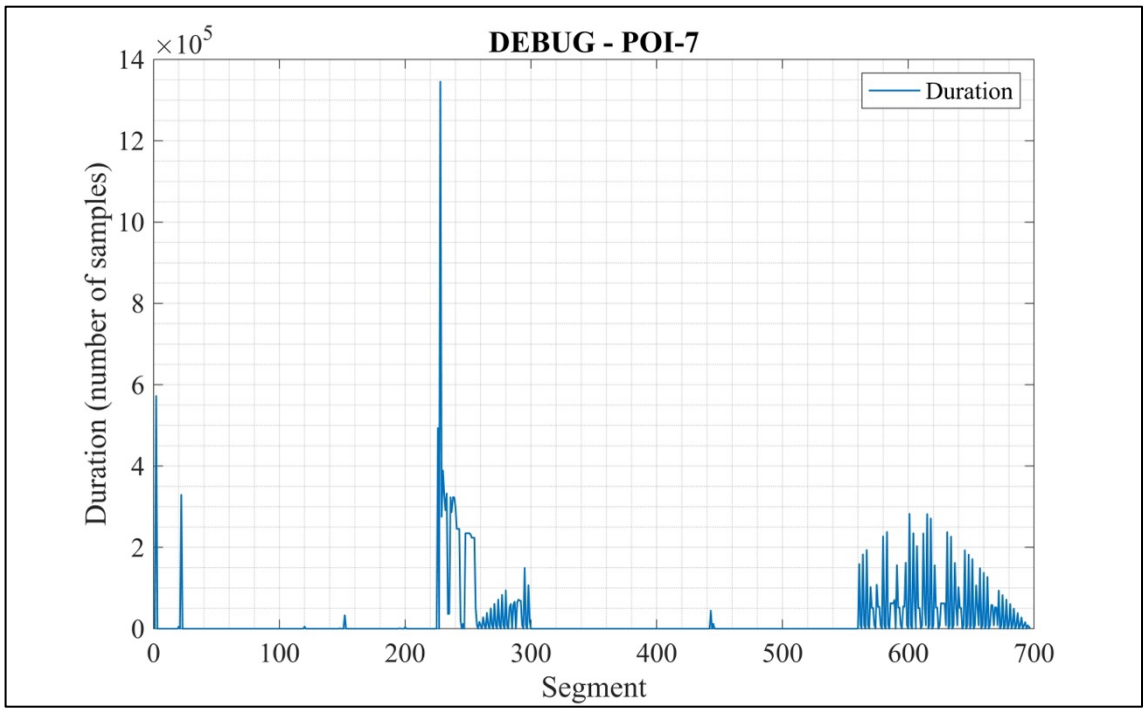

Figure 29 – DEBUG POI-7 for Test1

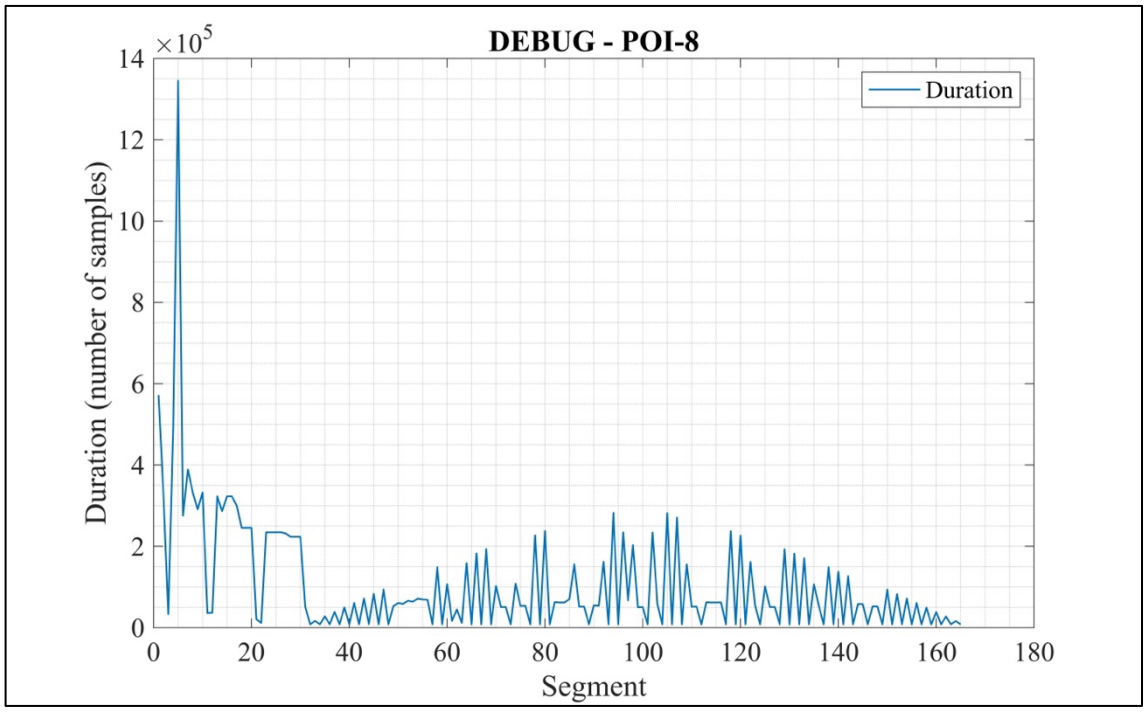

Figure 30 – DEBUG POI-8 for Test1

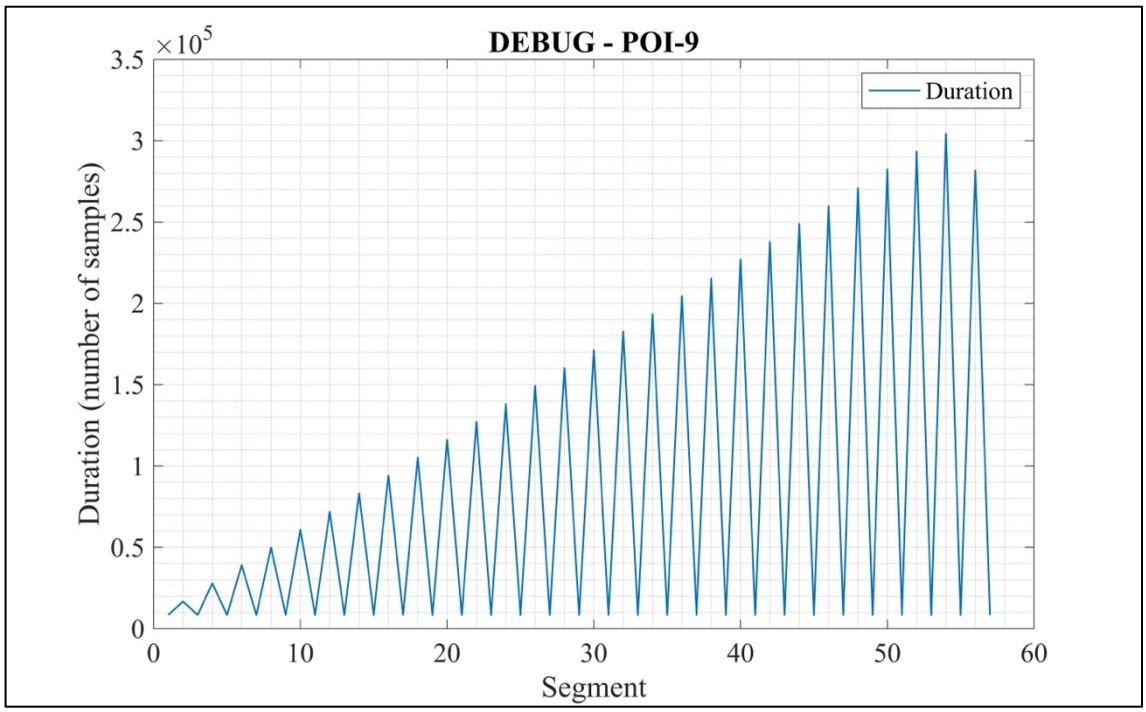

Figure 31 – DEBUG POI-9 for Test1

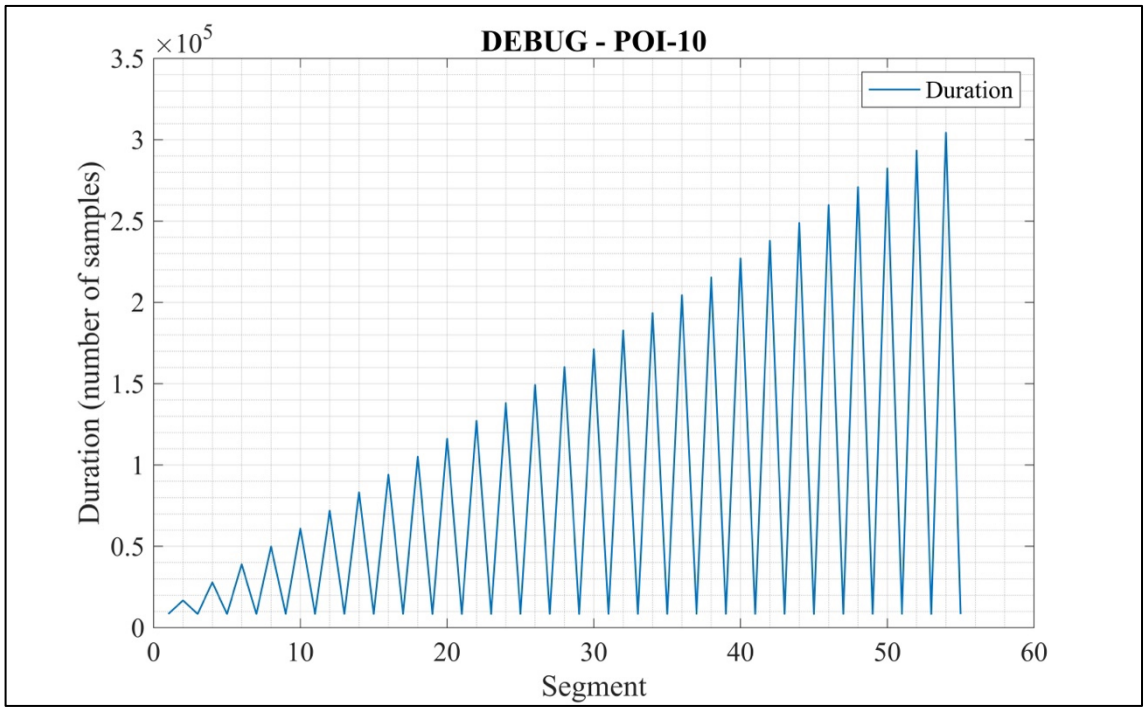

Figure 32 – DEBUG POI-10 for Test1

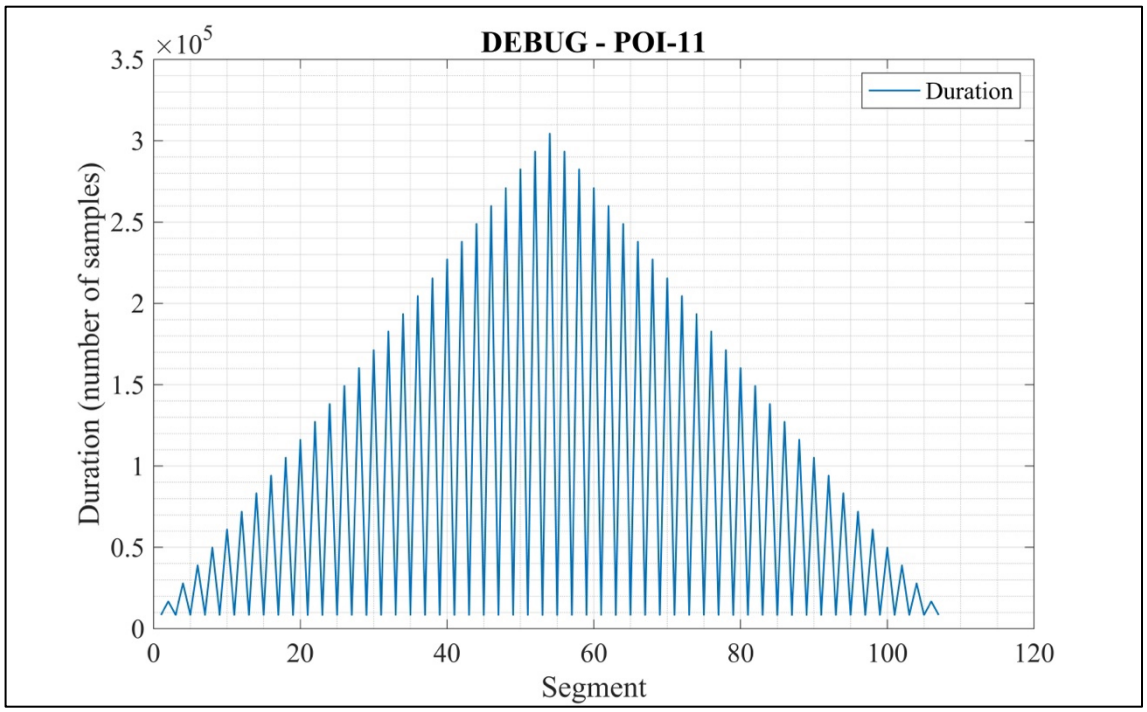

Figure 33 – DEBUG POI-11 for Test1

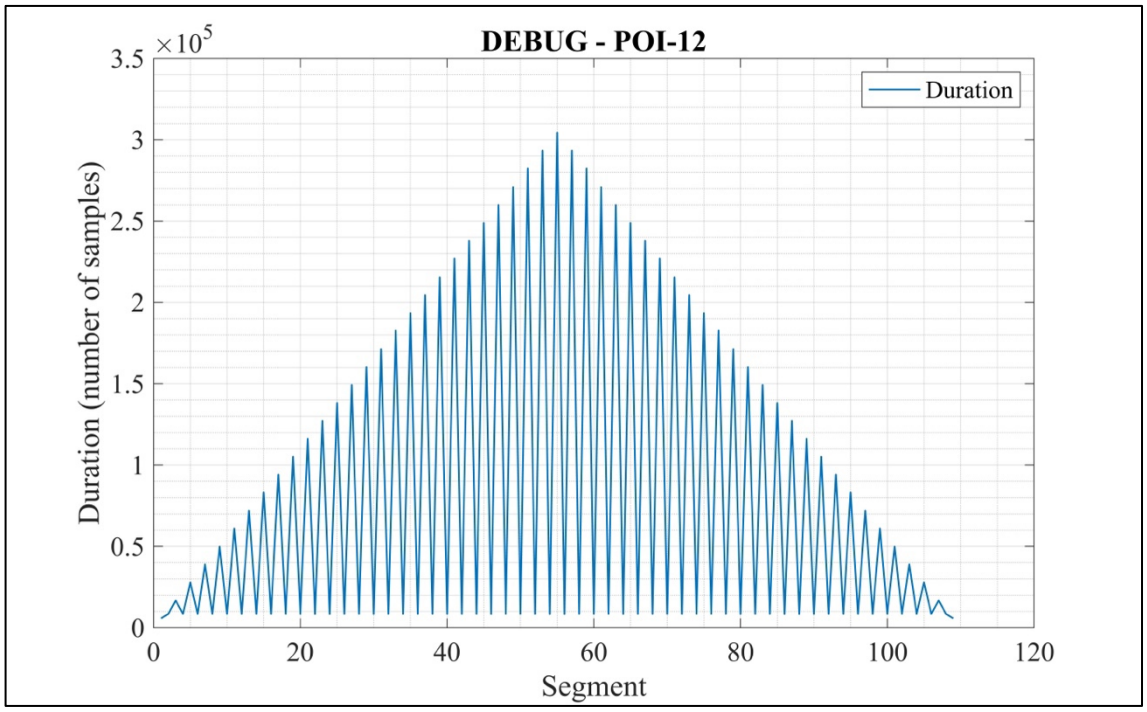

Figure 34 – DEBUG POI-12 for Test1

DEBUG – Test2 dataset

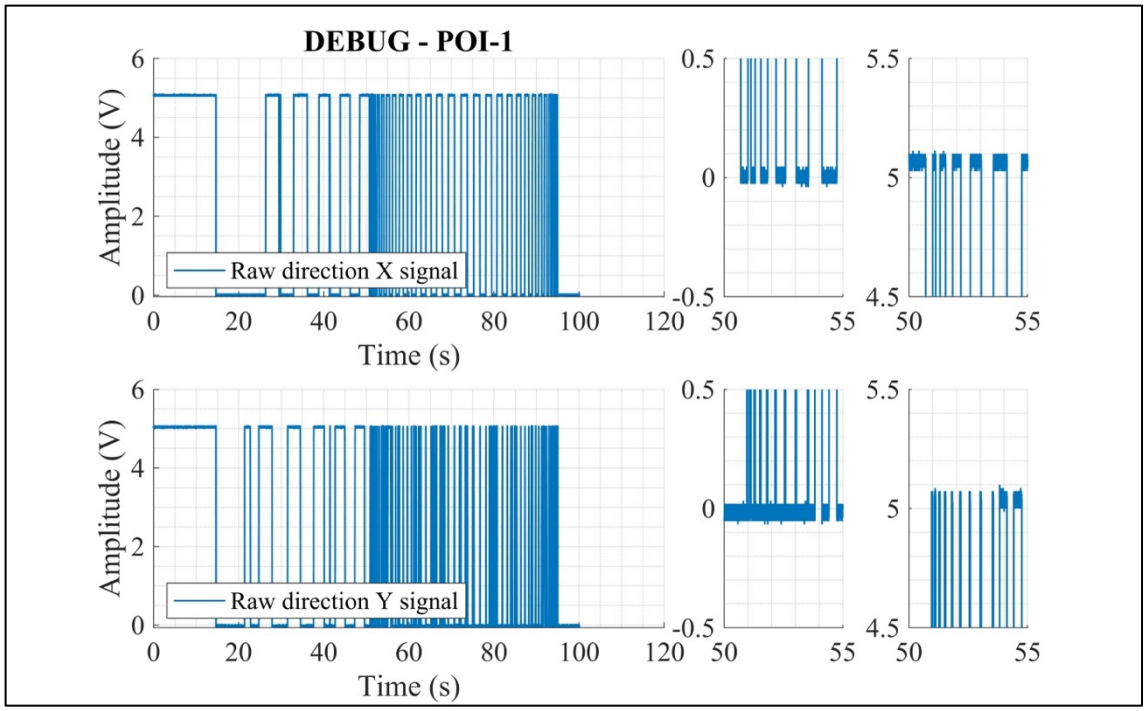

Figure 35 – DEBUG POI-1 for Test2

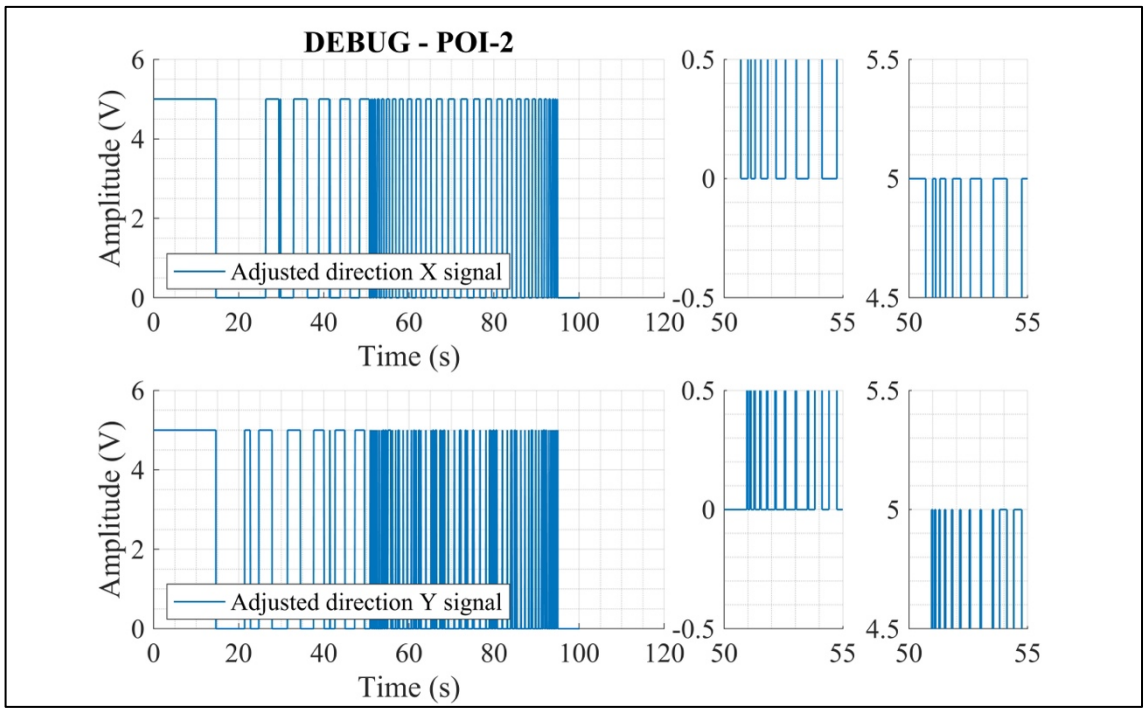

Figure 36 – DEBUG POI-2 for Test2

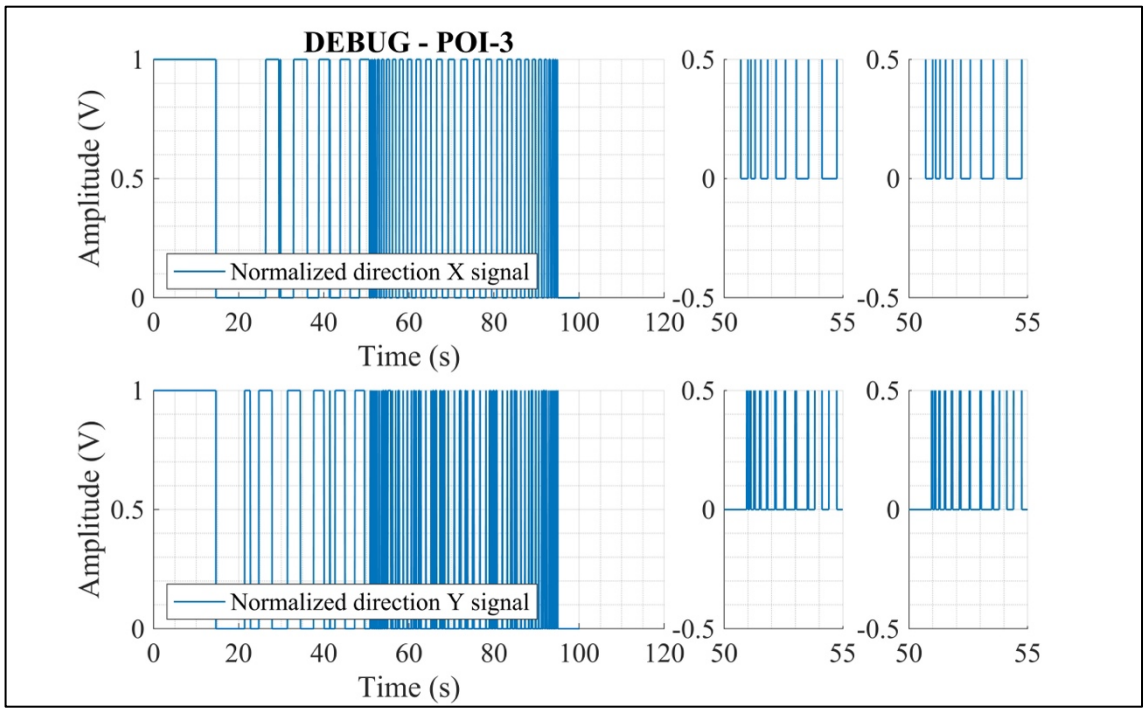

Figure 37 – DEBUG POI-3 for Test2

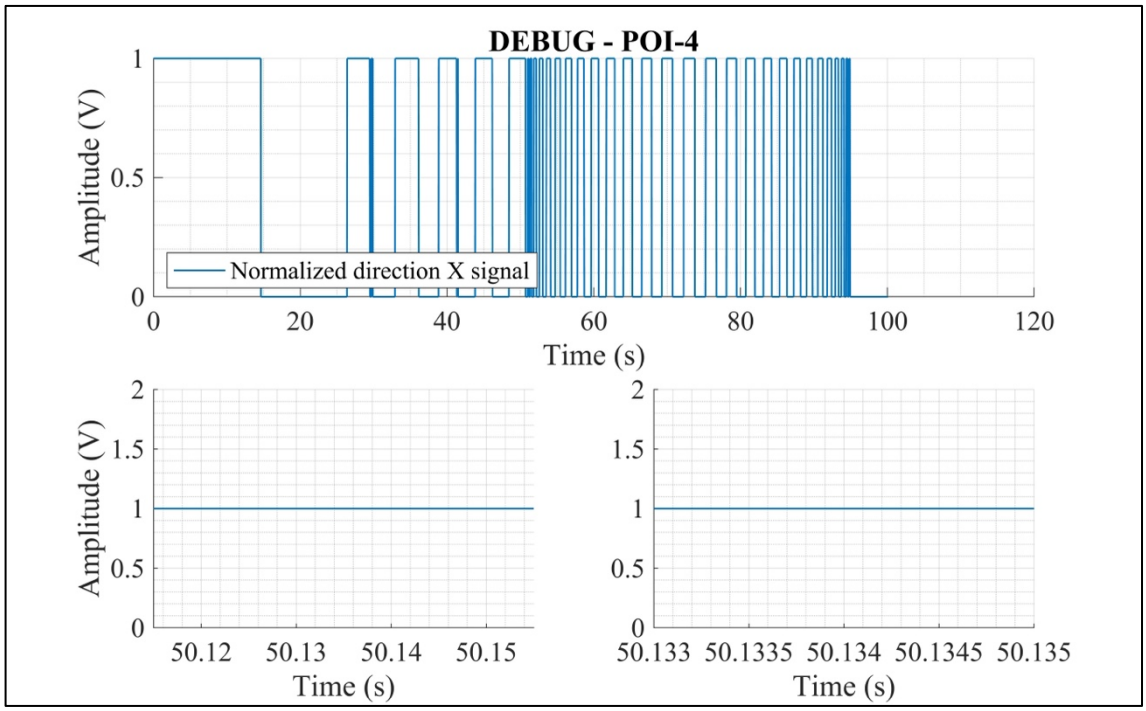

Figure 38 – DEBUG POI-4 for Test2

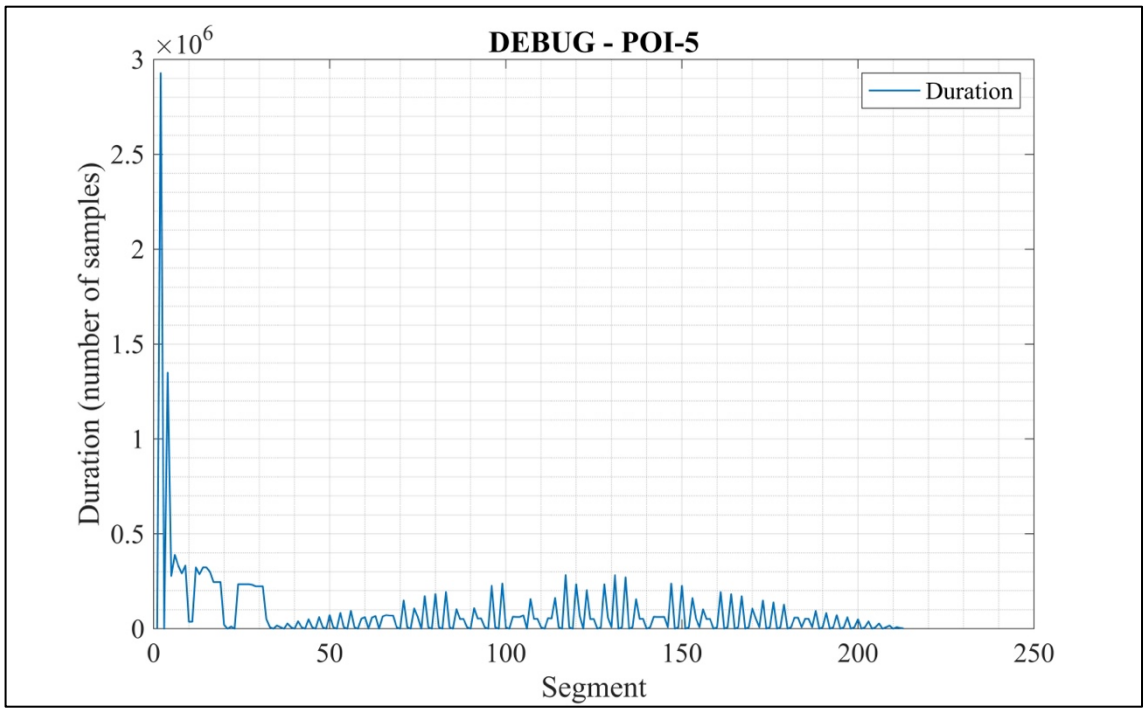

Figure 39 – DEBUG POI-5 for Test2

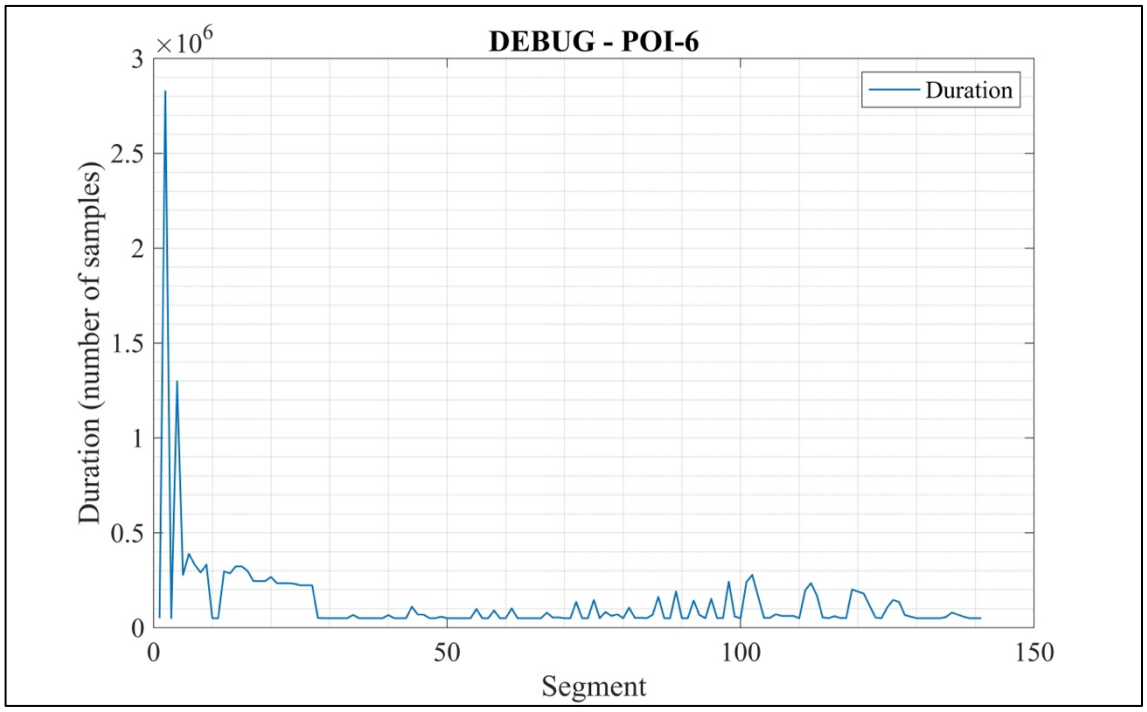

Figure 40 – DEBUG POI-6 for Test2

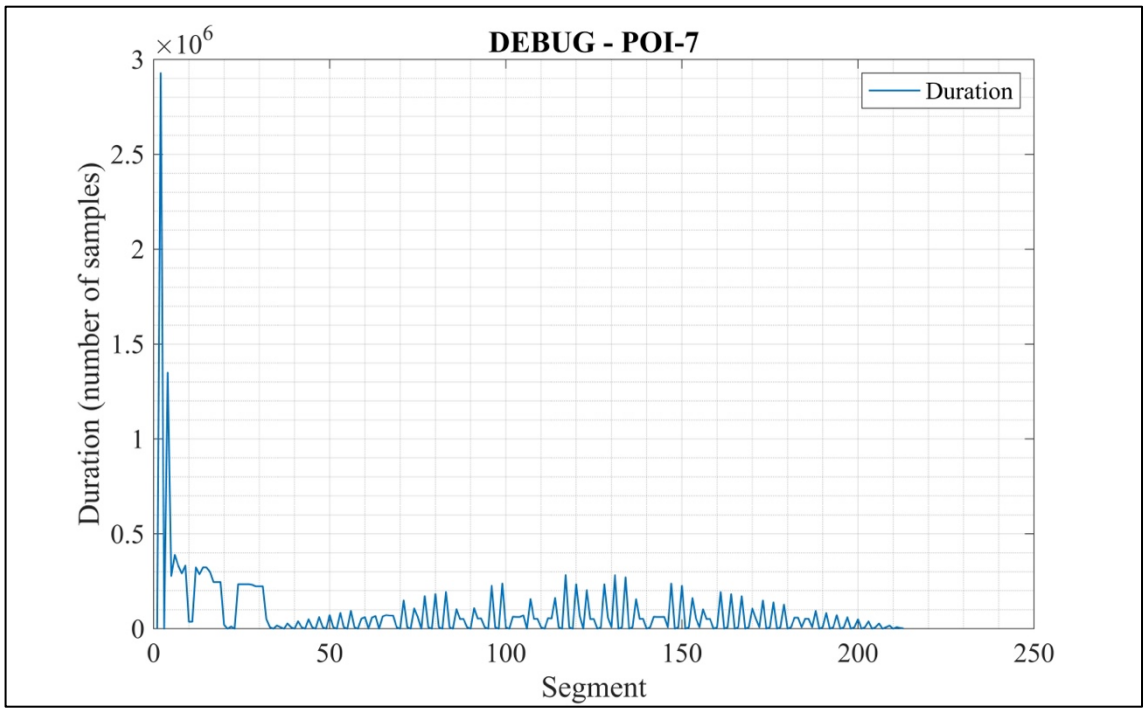

Figure 41 – DEBUG POI-7 for Test2

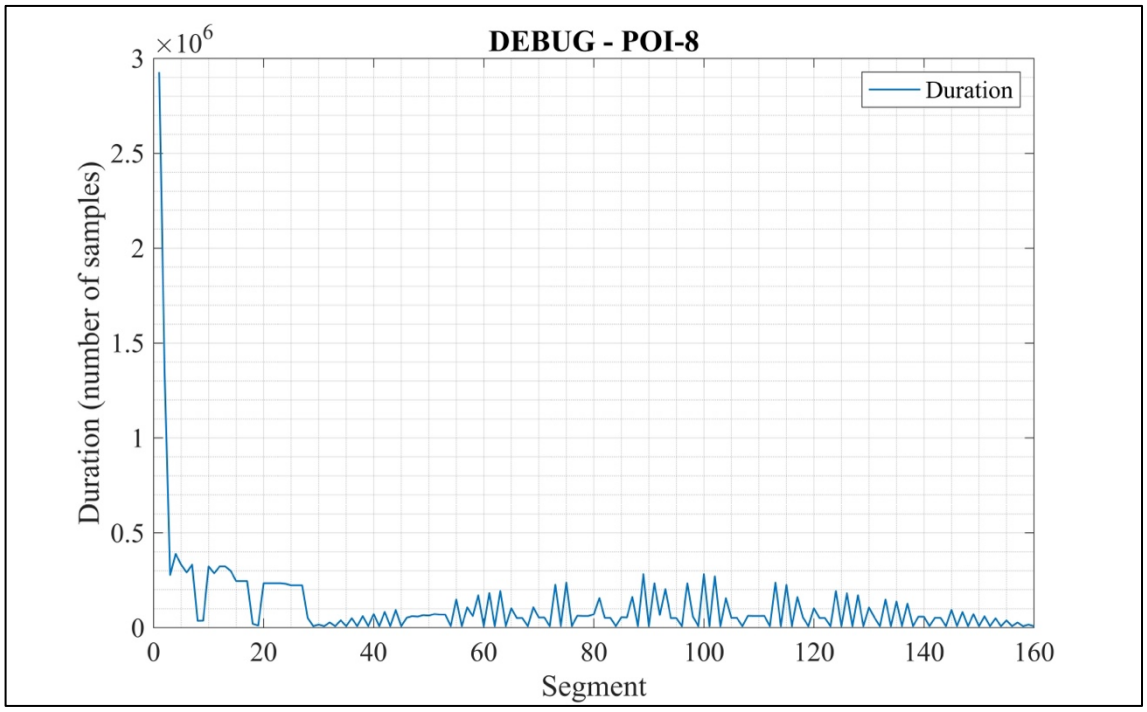

Figure 42 – DEBUG POI-8 for Test2

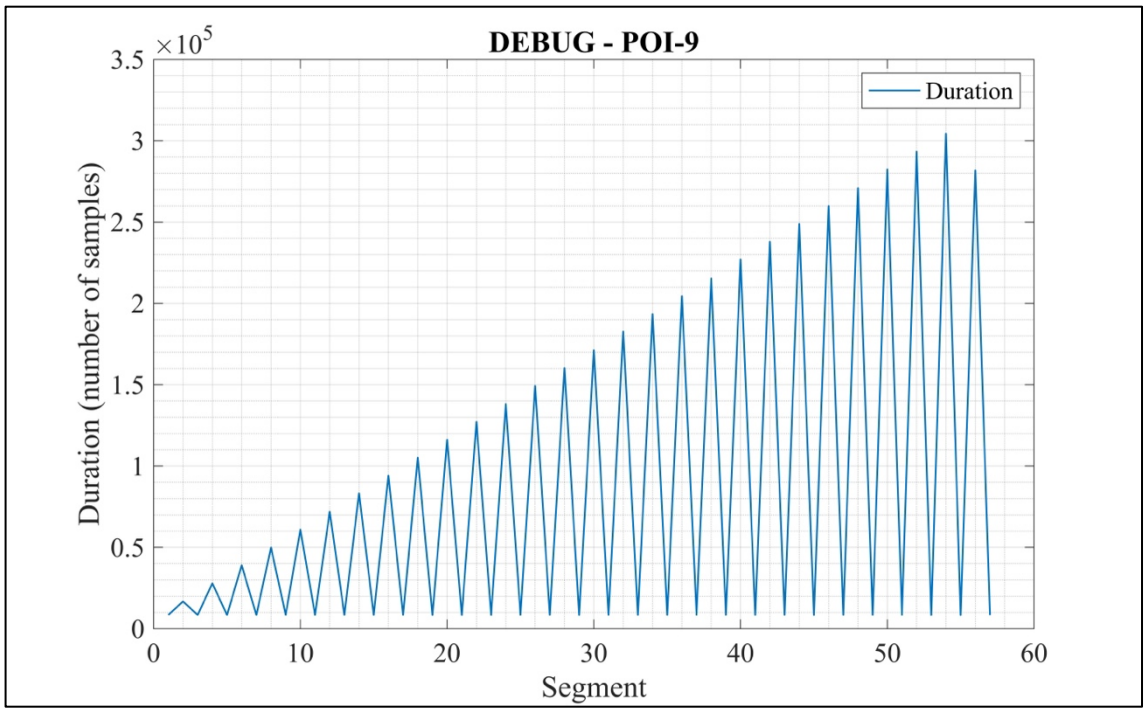

Figure 43 – DEBUG POI-9 for Test2

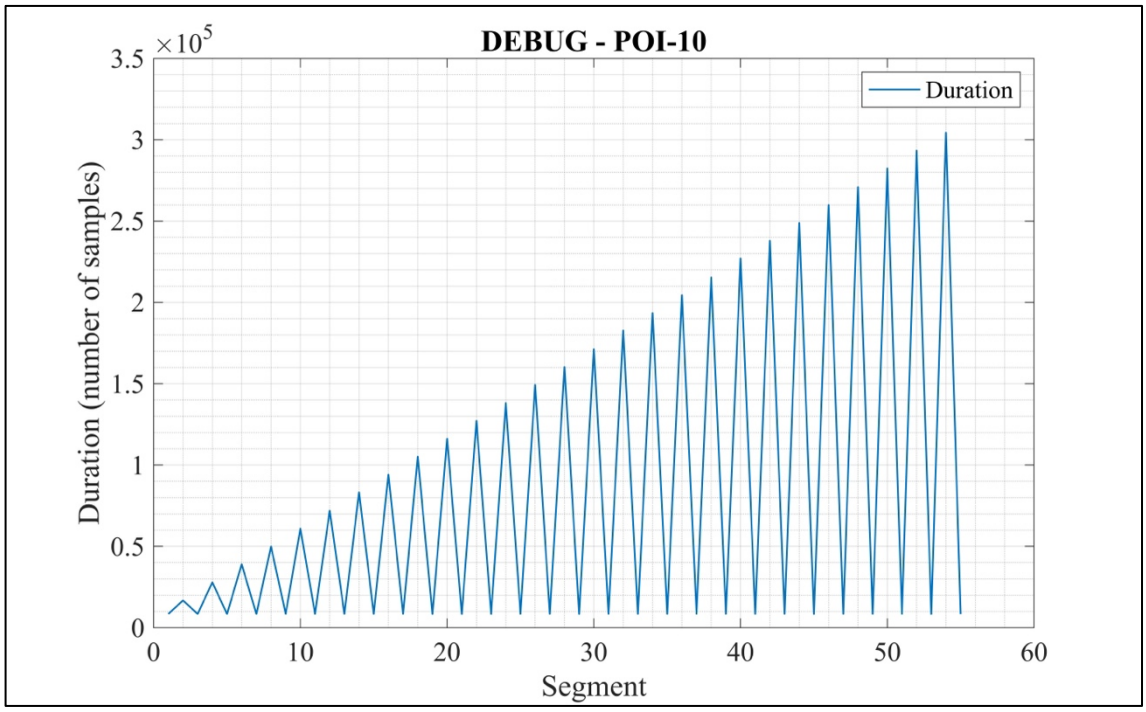

Figure 44 – DEBUG POI-10 for Test2

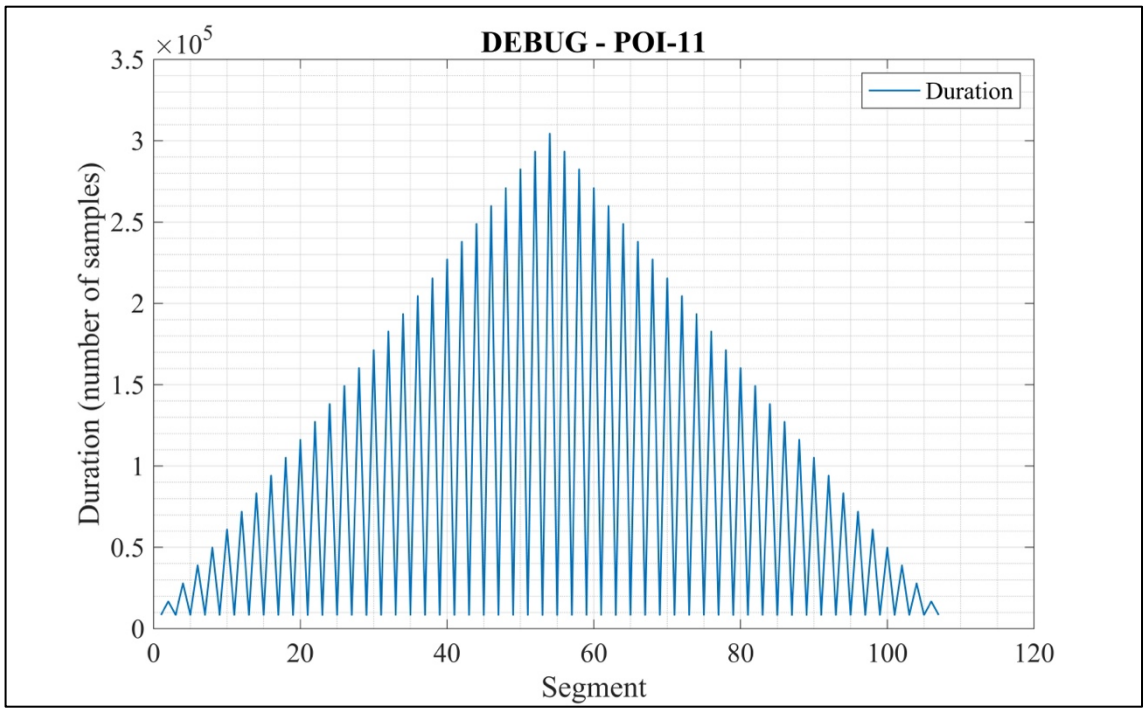

Figure 45 – DEBUG POI-11 for Test2

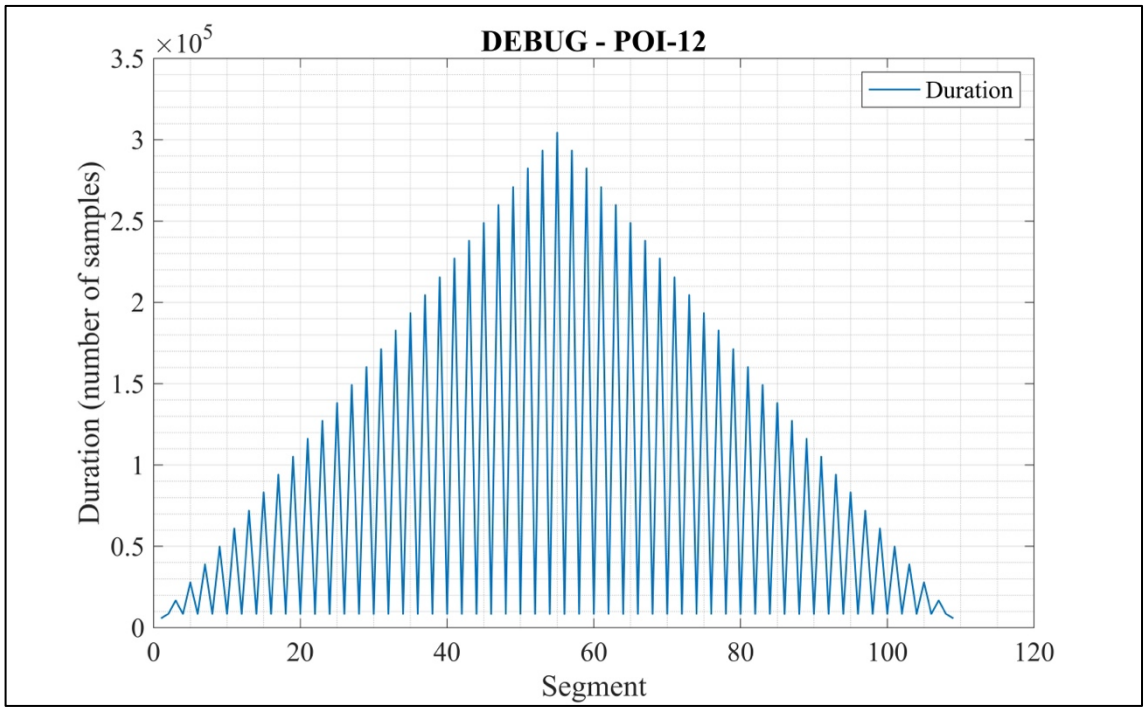

Figure 46 – DEBUG POI-12 for Test2

DEBUG – Test3 dataset

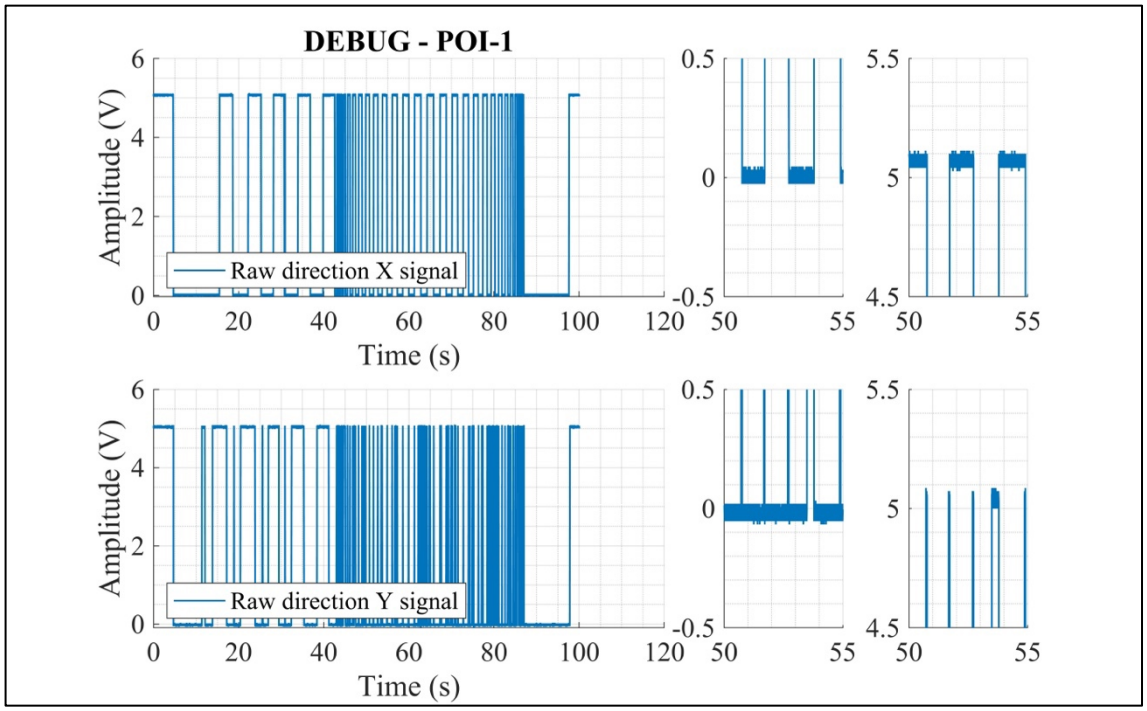

Figure 47 – DEBUG POI-1 for Test3

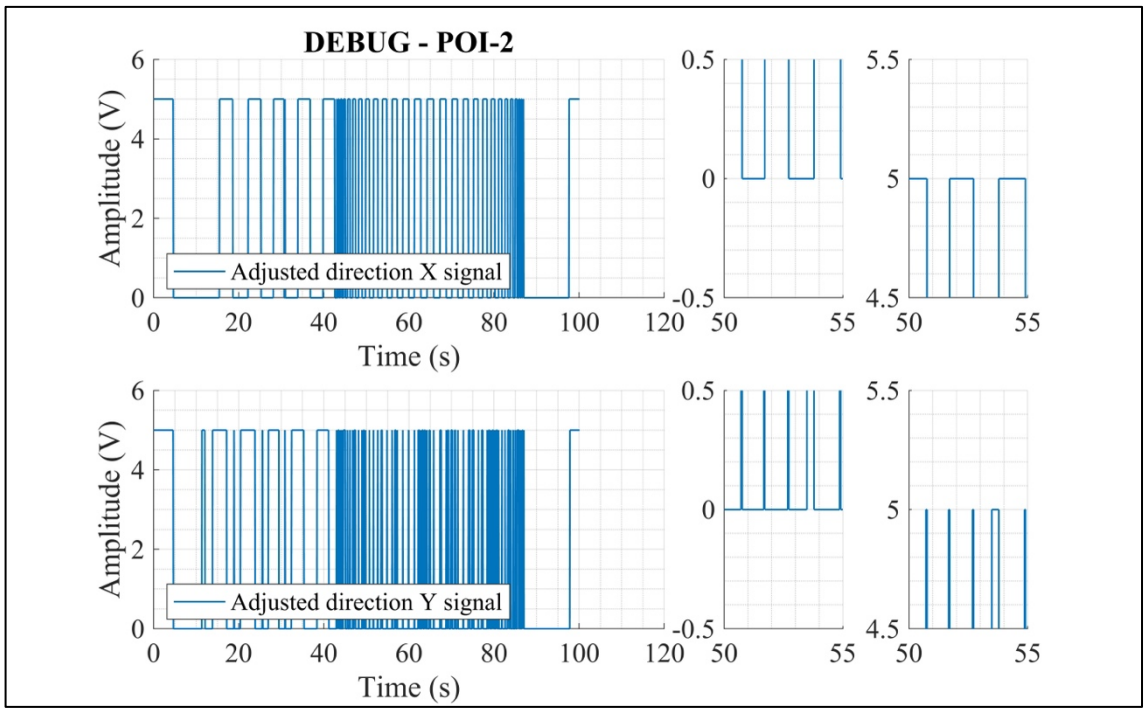

Figure 48 – DEBUG POI-2 for Test3

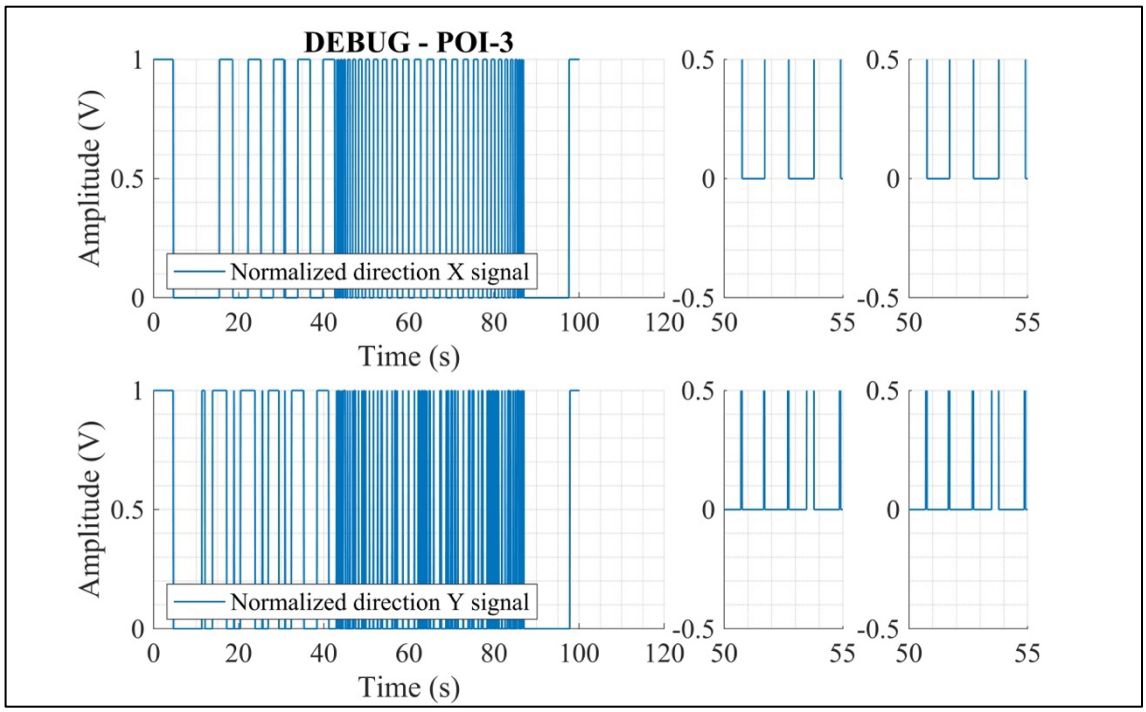

Figure 49 – DEBUG POI-3 for Test3

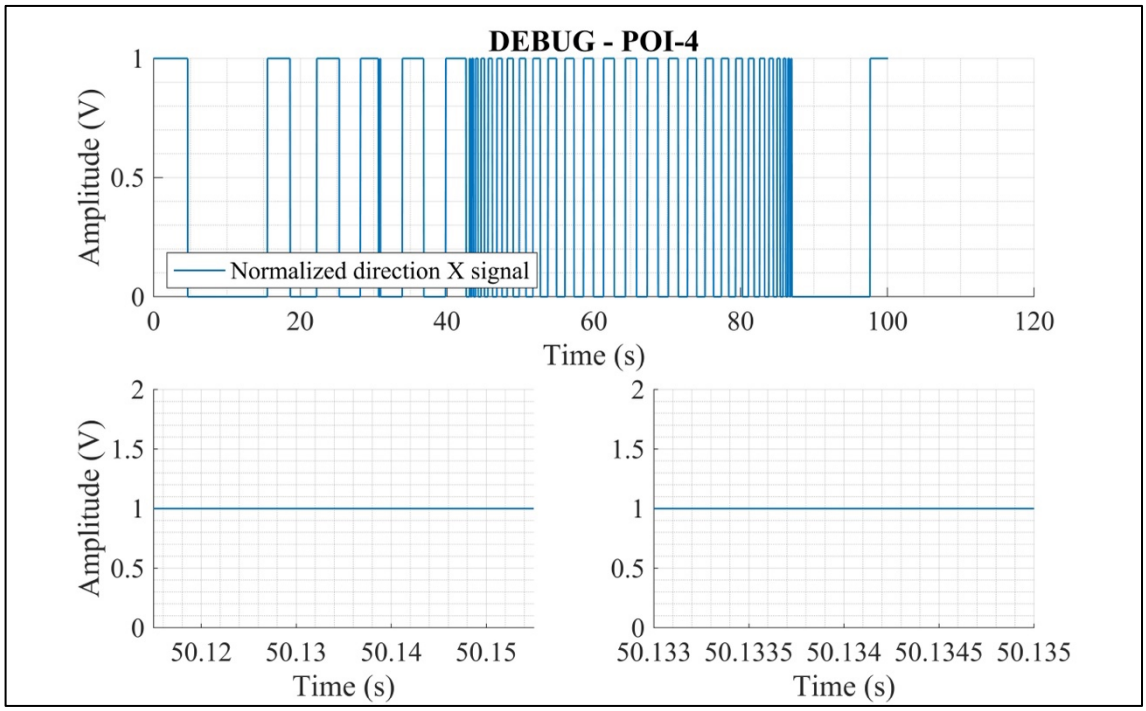

Figure 50 – DEBUG POI-4 for Test3

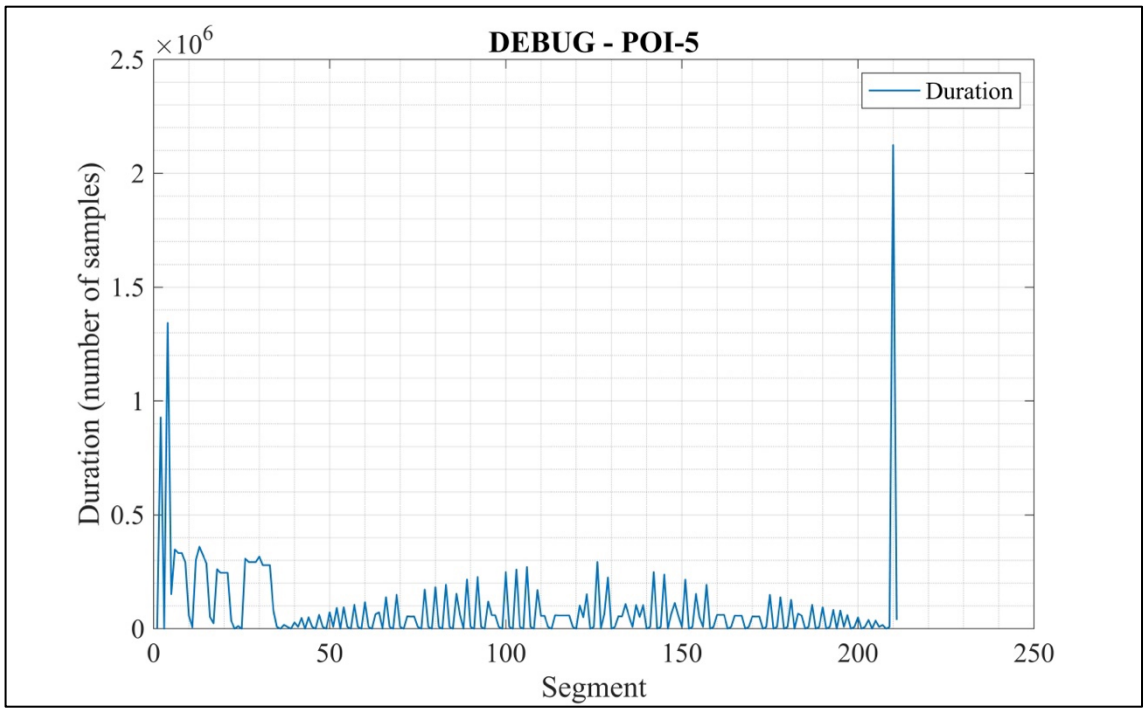

Figure 51 – DEBUG POI-5 for Test3

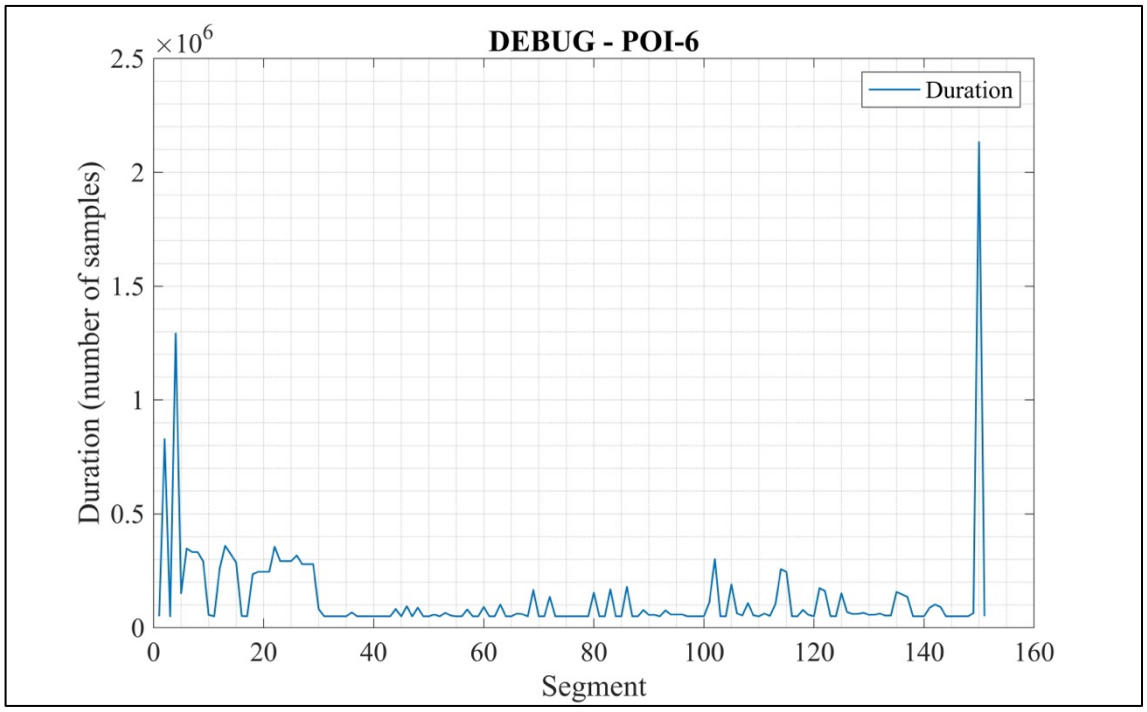

Figure 52 – DEBUG POI-6 for Test3

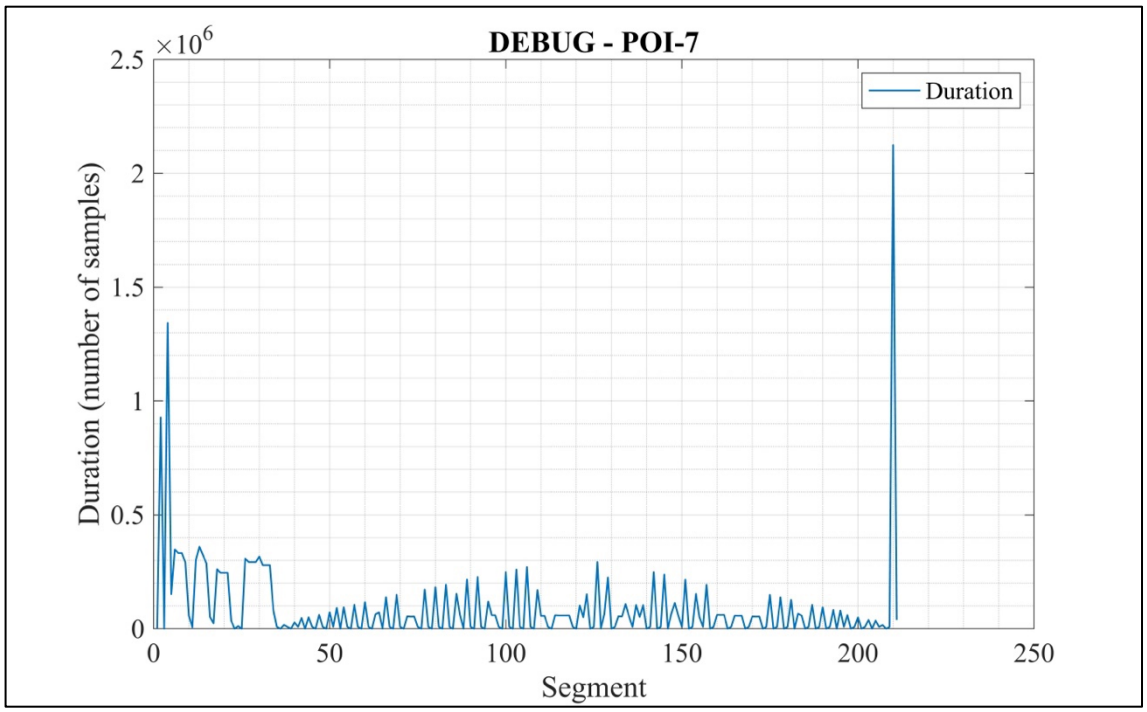

Figure 53 – DEBUG POI-7 for Test3

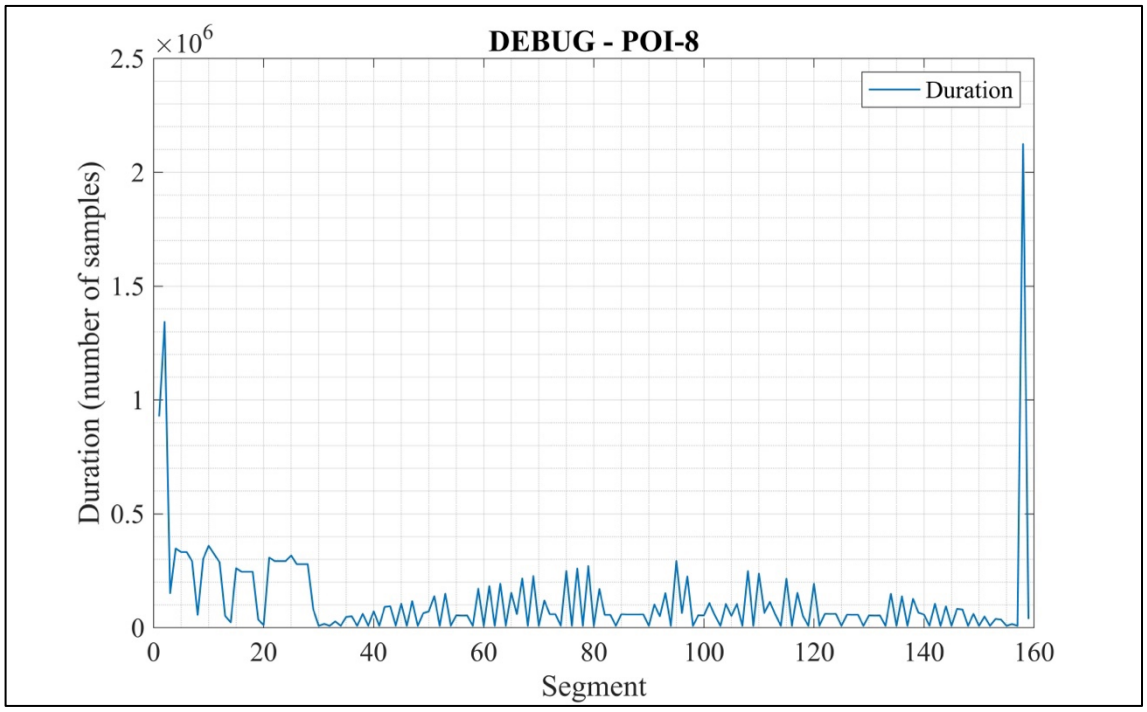

Figure 54 – DEBUG POI-8 for Test3

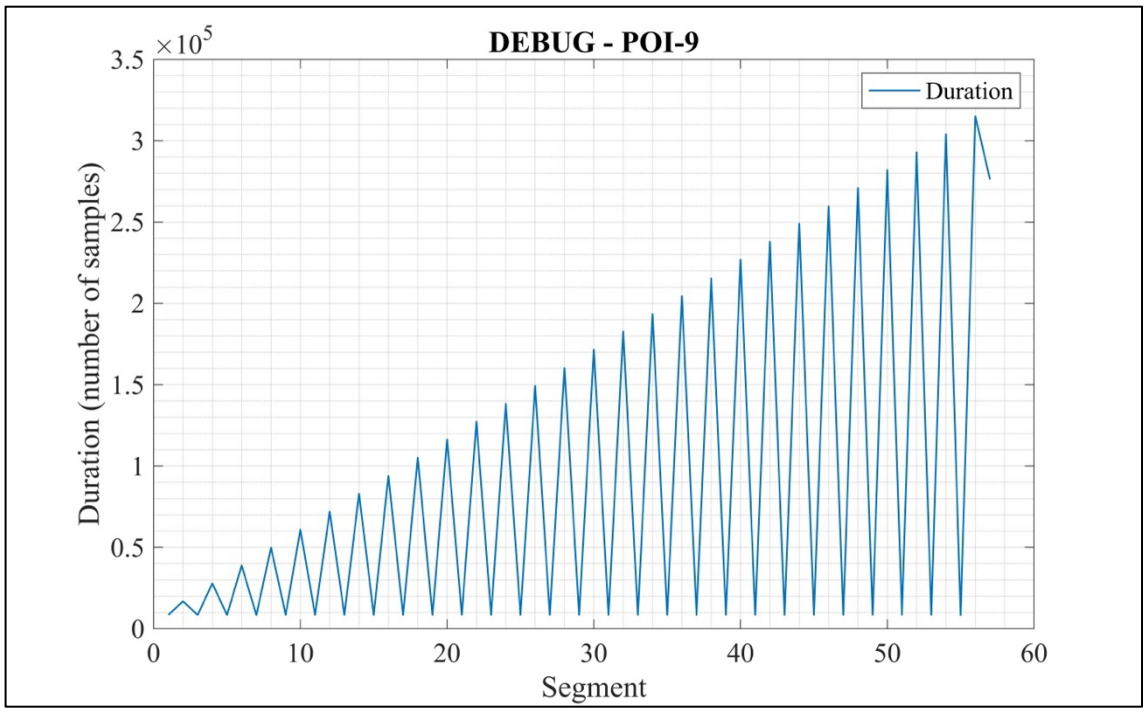

Figure 55 – DEBUG POI-9 for Test3

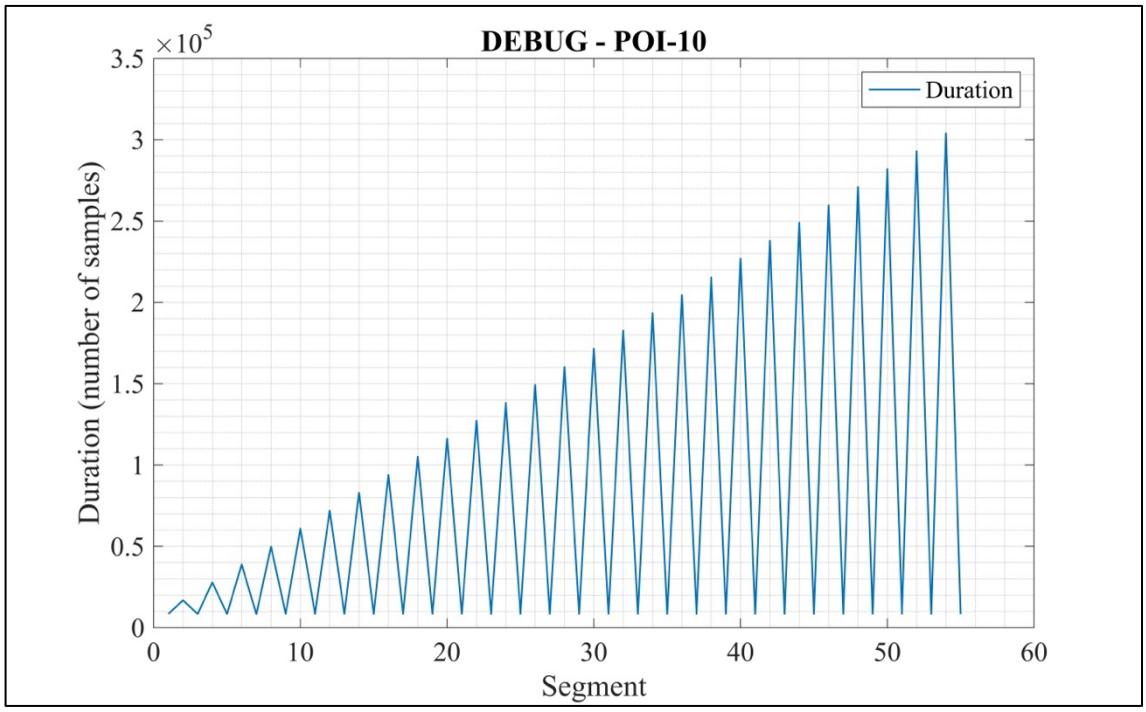

Figure 56 – DEBUG POI-10 for Test3

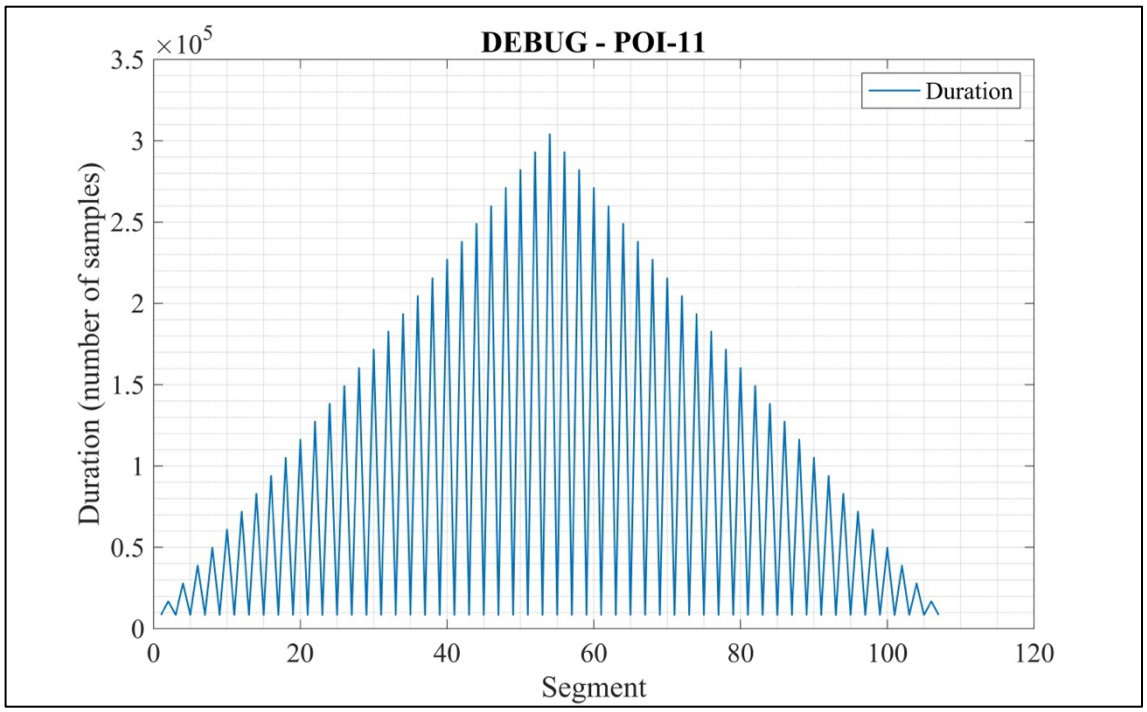

Figure 57 – DEBUG POI-11 for Test3

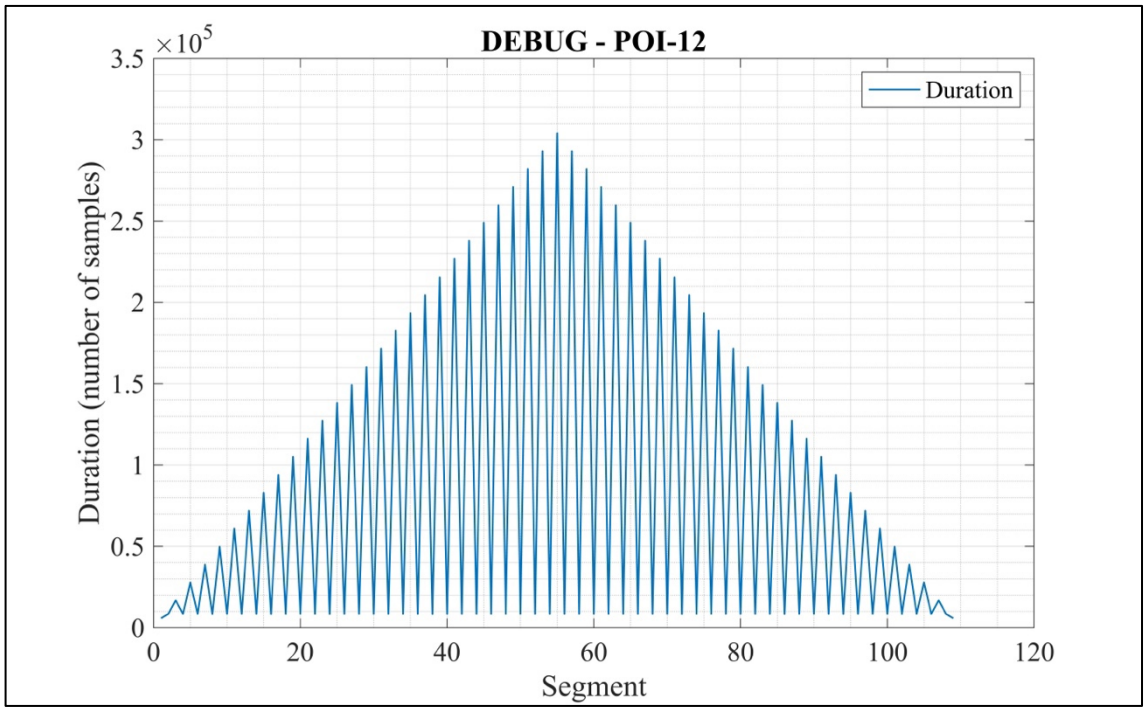

Figure 58 – DEBUG POI-12 for Test3