A Self-Organising Map Based Algorithm for Analysis of ICmetrics Features

Xiaojun Zhai, Kofi Appiah, Shoaib Ehsan, Wah M Cheung, Huosheng Hu, Dongbing Gu, and Klaus McDonald-Maier
School of Computer Science & Electronic Engineering
University of Essex, Colchester, UK
{xzhai, kappiah, sehsan, wmcheu, hhu, dgu, kdm }@essex.ac.uk

Gareth Howells
School of Engineering and Digital Arts
University of Kent
Canterbury, UK
W.G.J.Howells@kent.ac.uk

***Abstract*—ICmetrics is a new approach that exploits the characteristic and behaviour of an embedded system to obtain a collection of properties and features, which aims to uniquely identify and secure an embedded system based on its own behavioural identity. In this paper, an algorithm based on a self-organising map (SOM) neural network is proposed to extract and analyse the features derived from a processor's performance profile (i.e. average cycles per instruction (CPI)), where the extracted features are used to help finding the main behaviours of the system. The proposed algorithm has been tested with different programs selected from the MiBench benchmark suite, and the results achieved show that it can successfully segment each program into different main phases based on the unique extracted features, which confirms its utility for the ICmetrics technology.**

Keywords- ICmetrics; signal processing; feature extraction; embedded systems; self-organizing map (SOM);

I. INTRODUCTION

The rapid growth of digital systems has transformed the way we create, destroy, share, process and manage information, which bring benefits in a number of embedded consumer applications and communication systems [1]. However, this has also paved the way for fraud and other related crimes, which make imperative the need for highly secure embedded systems [2]. Ensuring the identity and authenticity of electronic digital systems is therefore critical for enabling secure communications in embedded applications.

Compromised access to confidential data is growing due to much personal data is increasingly held by medical, legal and law enforcement agencies and normally access remotely through a broad spectrum of mobile and networked devices [3]. Such uses make it vital to increase overall dependability, integrity and robust security of an embedded system, which motivates researchers to searching the solutions for these problems. Generally, an embedded system consists of two main parts: hardware and software. There are some techniques called Physical Unclonable Functions (PUF) [4] or hardware intrinsic security [5] focus on the hardware security, where the manufacturing process variation is used to identify the integrated circuits. However, they are limited by environmental instability, and the need to repeatedly power cycle to gain several samples. On the other hand, there are also many approaches for detecting software failure, tampering and malicious exploitation of embedded systems [1, 6], the common disadvantage of these approach are require store the sensitive data in the system as the template, which may weaken system's safety.

In order to search a better solution for the above problems, a new concept termed ICmetrics (Integrated Circuit metrics) is introduced. The concept is similar to biometrics where human properties and features are characterised to create a uniquely identifying metric value [7]. ICmetrics exploits the structure and behaviour of an embedded system, and obtain a collection of properties and features to identify the system. In principle, an ICmetric generated from a networked embedded system's unique behaviour will enable similar applications to generate unique encryption key to secure communications and stored data [8]. In addition, it provides the additional advantage that no user data is required, which is essential for applications that is no direct interaction with human operators. The ICmetrics only rely on the properties and features of the system, which means the system identifier (i.e. basic number or encryption key) can be regenerated on demand and no need to locally store it. A significant change in the system's operation or the readings from its sensors that will cause the system's properties and features to change, the system identifier will be changed as well. Therefore, the ICmetrics could improve both security and dependability based on exploitation of the system's unique behaviour.

In order to realise the ICmetrics concept for a practical application, many challenges must be overcome. In the our previous works [9] and [10], we aimed to explore the possibility of applying the ICmetrics technology to generate stable encryption keys based on characteristics derived from embedded systems' operation. The program counter (PC) of a processor core was used as a source for ICmetrics features, where full PC profiles were statistically analysed to find the distinct values from different programs. The achieved results showed that independent values of the PC cannot always give unique values that can be used to identify hardware devices. Therefore, the PC should be combined with other measurements extracted from electronic devices' operation as the ICmetrics feature. Investigation of the other suitable features for ICmetrics that is becomes the key task of current research.

In this paper, we present an algorithm based on analyses of a processor's performance profile (i.e. average cycles per instruction (CPI)) to segment the running program into different main behaviours or phases. The algorithm utilises a self-organising map (SOM) to cluster the samples of average CPI into different categories, and then the post-processing module groups them to into different phases. Based on the segmentation results, the detailed features of the running program can be further extracted, which can be served as one of ICmetrics feature to distinguish the embedded systems. The proposed algorithm has been successfully implemented in MATLAB as a proof of concept prior to hardware implementation. Results achieved show that the proposed algorithm can successfully extract and segment each program's main behaviours based on their unique characteristics. The used programs are chosen from the automotive package of the MiBench suite of benchmark algorithms [11].

The rest of paper is organised as follows. The proposed algorithm is introduced in Section 2. The experimental setup and the implementation results are discussed in Section 3. Finally, the conclusions are presented in Section 4.

II. Proposed Algorithm

The PC value was explored as an ICmetrics feature in our previous studies [9] and [10], but only separated PC values were statistically analysed, which are unable to provide enough information about devices' behaviour. Contrary to that, we observe that PC is a good source derived from high level program. Since the way a program's execution changes over time is not totally random, it often can be divided into repeating behaviours or phases [12]. Loops, conditional branches and data transfer all can be observed in the PC profile. More importantly, the overview of PC flow can show the unique shape or skeleton of a program. Therefore, it is possible to find distinctive characteristics of devices' behaviour by analysing these unique shapes. Fig. 1 shows an example of a program's PC profile.

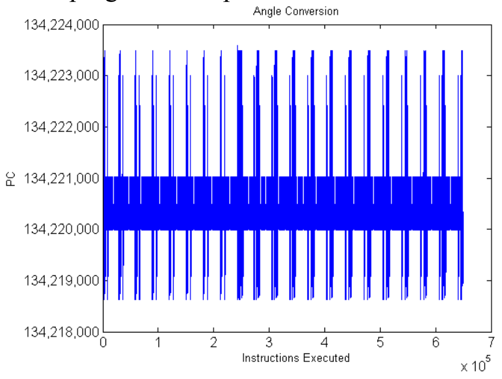


Figure 1. An example of a program's PC profile.

The program used in Fig. 1 is angle conversion, which is selected from the automotive package of the MiBench suite of benchmark algorithms [11]. The angle conversion program is mainly used to perform two tasks: degree to radian and radian to degree angle conversions. As can be seen from Fig. 1, the characteristics of degree to radian and radian to degree functions have a clear boundary. However, in order to automatically detect this boundary, approximately 6.5×10^5 instructions in the angle conversion's PC profile need to be analysed, which has the potential to degrade embedded system performance. In order to solve this problem, an alternative solution is proposed to localise the boundary of each main function.

A. *System Performance Profile*

A typical system performance can be CPI [13], which indicates the complexity of instructions executed within a particular period time. For example, a Branch or Jump instructions are normally need multiple clock cycles to be executed by processor, which also indicates function calls are initiated. The average CPI of a processor can be calculated as given in [13]:

$$CPI = \frac{T \times f_{\max}}{I} \tag{1}$$

where I is the total number of instructions, T is time consumption while executing the total number of instructions, and $f_{\max}$ is the maximum clock frequency of a processor.

The average CPI diagram for every 10,000 instructions of angle conversion program is shown in Fig. 2.

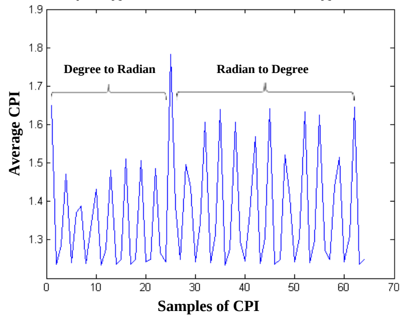


Figure 2. The CPI diagram of angle conversion.

As can be seen from Fig. 2, the CPI diagram of angle conversion program comprises the same two main phases as the PC profile in Fig. 1. However, the number of samples needed to be analysed is reduced to approximately 63, which has significantly less data size compared to directly analysing the PC profile.

The main idea behind the proposed algorithm is trying to automatically segment the major phases from the CPI diagram, and then transform the localised positions from the CPI domain to the PC domain using Equation (1), which would make it possible to extract information about any change in program's behaviour. After finding the local identification information from the major program phases, this information can be used to generate a unique identifier for the embedded system.

B. *CPI Clustering*

In order to segment the major characteristics from the CPI diagram, a SOM based neural network clustering tool is introduced to group the CPI data by similarity. In addition, a pre-processing and a post-processing module are also proposed to smooth and further merge the CPI data respectively.

1) Pre-processing Module

The original CPI diagrams normally consist of a set of high variation peak values at some certain points which indicate the boundaries of the major characteristics of the program. On the contrary, the rest of data values in the CPI diagram generally comprise a set of relatively low variation values. Therefore, once the low variation values are grouped together, the rest of data values define the boundaries of the major characteristics of the program. In order to effectively use the SOM based neural network clustering tool to group the data values, a pre-processing module is applied to smooth the data values, which reduces variances locally in the CPI diagram.

The pre-processing module consists of a mean filter is used to calculate the average value inside a $1\times w$ rectangular window. Suppose that $f(x)$ denotes the CPI value at position x which is always the centre point of a rectangular window B with size $1\times w$. The window mean value $f_{mean}(x)$ is calculated by (2):

$$f_{mean}(x) = \frac{\sum_{x\in B} f(x)}{w} \quad (2)$$

In the proposed algorithm, the higher the value of w the smoother is the CPI result, but high w value means more computations and may remove the peak values at the boundaries. Thus, the value of w is set to minima '3' in order to minimise the effects on the peak values at the boundaries. Fig. 3 shows the resulting diagram after applying the pre-processing module on the original CPI diagram.

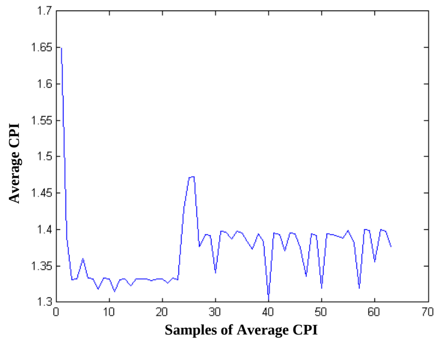


Figure 3. Resulting CPI diagram after applying the per-processing module.

As can be seen from Fig. 3, the curve of the resulting CPI diagram has been significantly smoothed by the pre-processing module, but the main peak values at the boundaries are still intact. The next step is to apply the SOM based neural network to cluster these data.

2) SOM Module

SOM is a type of Artificial Neural Network (ANN) that is trained using unsupervised learning algorithm, which learns to classify the input vectors into different categories according to how they can be grouped in the input space [14].

For proposed algorithm, the input vector of SOM is the smoothed averaged CPI value. The neurons inside SOM competitive layer are organised in hexagonal topology, and the total number of used neurons is '9'. The weights of each neuron are initialised randomly, and each weight is changed along with the training process, and moves to the average position of all of the input vectors for which it is a winner or in the neighbourhood of a winner.

During the training process, the SOM neural network tries to cluster the input values into 9 categories and maps them into the corresponding neurons. Fig. 4 shows clustered averaged CPI diagram when applying the trained SOM neural network on the smoothed average CPI diagram.

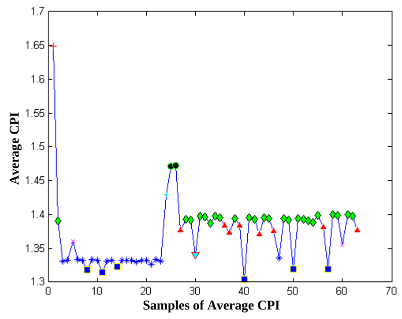


Figure 4. Clustered CPI diagram.

In Fig. 4, similar CPI values are clustered in the same categories, and marked with the same symbols. As evident from Fig. 4, most of the major phases from the averaged CPI diagram have been marked with the same symbols, however, there are still some samples have not been grouped in the corresponding category sets. Therefore, a post-processing module is introduced to further merge the samples into the major group.

3) Post-processing Module

The post-processing module is a stage to merge the discrete minor clusters into few major clusters that can correct the positions of major clusters obtained from the SOM.

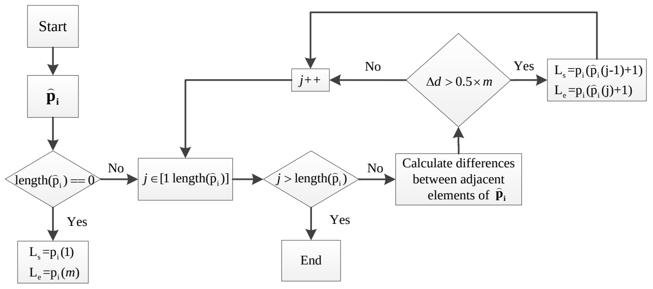
Let l and c denote the total number of averaged CPI samples and clustered categories respectively, which will result in the outputs of the SOM neural network generating a matrix $\mathbf{R}$ with size $c\times l$:

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,l} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,l} \\ & \cdots\cdots & & \\ r_{c,1} & r_{c,2} & \cdots & r_{c,l} \end{bmatrix} \quad (3)$$

The elements in $\mathbf{R}$ are either '0' or '1'. A value of '1' for all elements within the same row vector indicates that they belong to the same category, and their column index is the location where the sample is in the smoothed averaged CPI diagram.

Let vector $\mathbf{p_i}$ consists of the column indices of the elements '1' at row i from $\mathbf{R}$ with size of m. The differences between adjacent elements of the vector $\mathbf{p}_i$ are calculated to form vector $\mathbf{d}_i$.

A scan process is firstly carried out to mark the elements in $\mathbf{d}_i$ one by one, where every element is masked either peak or off-peak element based on threshold *t*, and then, the positions of the peak element form vector $\hat{\mathbf{p}}_i$. All the elements in $\hat{\mathbf{p}}_i$ are analysed in order to find the suitable candidate positions L_s and L_e for the major cluster in category *i*. The flow chart of this analysis is shown in Fig. 5.



Start
$\hat{\mathbf{p}}_i$
$\text{length}(\hat{p}_i) == 0$
No
Yes
$L_s = p_i(1)$
$L_e = p_i(m)$
$j \in [1\ \text{length}(\hat{p}_i)]$
$j > \text{length}(\hat{p}_i)$
Yes
End
No
Calculate differences between adjacent elements of $\hat{\mathbf{p}}_i$
$\Delta d > 0.5 \times m$
No
$j{+}{+}$
Yes
$L_s = p_i(\hat{p}_i(j-1)+1)$
$L_e = p_i(\hat{p}_i(j)+1)$

Figure 5. Flow chart of major group localisation.

In Fig. 5, the obtained positions of major cluster may consist of more than one set of candidates. An extra process is proposed to further check whether the candidates belong to the same major cluster. The distances between the adjacent candidate sets are calculated and if the adjacent candidate sets have close distance, then they will be merged together. Fig. 6 shows the calculated positions of the major clusters.

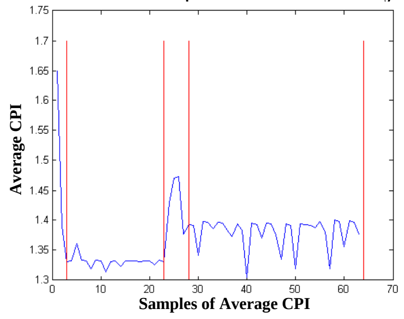


Figure 6. Calculated positions of the major groups.

After obtaining the positions of major clusters in CPI diagram, according to (1), the corresponding positions in PC profile can be also calculated.

III. Experimental Setup and Implementation Results

A. Experimental Setup

For the proposed work, we have employed an embedded system based on a STMicroelectronics STM32F207IG microcontroller equipped with an ARM 32-bit Cortex-M3 processor [15]. A combination of KEIL µVision IDE [16] and ULINKpro Debug and Trace Unit is used to download the program and trace the instructions executed in the microcontroller, where high-Speed data and instruction trace are streamed directly to the host computer allowing off-line analysis of the program behaviour [16].

Five programs from the automotive package of the MIBench suite of benchmark algorithms [11] are selected: angle conversion (AC); bit count (BC); cubic function (CF); random numbers (RN); and square roots (SR). The five programs are mixed together with 6 different combinations to simulate six behaviours that resulted in an embedded system and are listed in Table I.

TABLE I. Combinations of the Programs

Devices	**Original benchmark algorithms**				
	AC	***BC***	***CF***	***RN***	***SR***
Behaviour 1	1[a]	1	1	1	1
Behaviour 2	2	1	1	1	1
Behaviour 3	1	2	1	1	1
Behaviour 4	1	1	2	1	1
Behaviour 5	1	1	1	2	1
Behaviour 6	1	1	1	1	2

a. Means the number of times a program is executed.

In Table I, the device is firstly running AC, BC, CR, RN, and SR only once, namely: ‘Behaviour 1’. The device is secondly the same programs, but running AC twice, namely: ‘Behaviour 2’. Following the same principle, we get six different behaviours in total. In general, the six behaviours are quite similar: the only difference is each behaviour has only one program that is executed twice, which significantly increases the difficulty to find the differences from the large number of executed instructions.

B. Implementation Results

The six mixed programs are downloaded and executed separately in STM32F207IG microcontroller. While running each program, the full executed instruction profile is traced and delivered in real-time to the host computer via ULINKpro Debug and Trace Unit. The µVision IDE exports the trace data to the hard disk for off-line analysis. For the proposed work, only PC and time information of each instruction are recorded in the profile, which are used to generate PC and CPI diagrams.

According to the equation (1), average CPI value is decided by *T*, *f* and *I*. Since the running frequency of the used microcontroller is 120 MHz, and the number of instructions per interval is set to 10,000, it essentially means that each point on the CPI diagram represents the average CPI value taken over 10,000 instructions of execution. The time consumption while executing 10,000 instructions is recorded in the tracing profile.

The proposed algorithm was successfully implemented in MATLAB, and tested using the six traced profiles mentioned in the previous section. The Behaviour 1 in Table I is used to simulate the normal status of the device. Smoothed Averaged CPI samples are firstly calculated in the pre-processing module, and then the averaged CPI samples of the ‘Behaviour 1’ are used to train the SOM neural network. After training, the neural network is applied to cluster them into 9 categories. At the end of process, the positions of major clusters in CPI diagram are obtained by the post-processing module. The rest of behaviours in Table I are used for testing, where the same procedures and trained SOM neural network are applied.

In Fig. 7, the major phases in average CPI profile have been marked with different colours using the positions that are obtained from the testing. By comparing the segmented results of the ‘Behaviour 1’ and the rest of behaviours, it is evident that the parameters of the major phases in the ‘Behaviour 1’ have significant differences from the others. For example, the size and number of each phase are varied in CPI profile, sequentially, if applying the phase locations to PC profile (e.g. Fig. 1), detailed features of the phase can be further analysed. By using these rough-to-fine features, a change in the system’s operation could be detected.

IV. Conclusion

In this paper, an algorithm based on the analysis of a processor’s performance profile is proposed to segment a program into different phases based on their major behaviours from embedded systems’ operation. A pre-processing and a post-processing module are combined with a SOM to cluster the samples of CPI into different groups, and then these positions of each segmented group can be further used to find the main behaviours of each program. The proposed algorithm has been successfully implemented in MATLAB as a proof of concept prior to hardware implementation. Results achieved show that the proposed algorithm can successfully extract features from the processor’s performance profile, based on the segmented results, a unique ICmetric identification information can be generated, which can be used to detect any unusual changes in an embedded system.

Acknowledgment

The authors gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council under grant EP/K004638/1 and the EU Interreg IV A 2 Mers Seas Zeeën Cross-border Cooperation Programme – SYSIASS project: Autonomous and Intelligent Healthcare System (project’s website http://www.sysiass.eu/).

References

[1] M.Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware Assisted Detection of Malicious Software in Embedded Systems," *IEEE Embedded Systems Letters*, vol. 4, pp. 94-97, 2012.

[2] F.-S. Corporation, F-Secure reports amount of malware grew by 100% during 2007. Helsinki, Finland, 2007.

[3] K. W. Miller, J. Voas, and G. F. Hurlburt, "BYOD: Security and Privacy Considerations," *IT Professional*, vol. 14, pp. 53-55, 2012.

[4] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *44th ACM/IEEE Design Automation Conference*, 2007, pp. 9-14.

[5] H. Handschuh, G.-J. Schrijen, and P. Tuyls, "Hardware Intrinsic Security from Physically Unclonable Functions," in *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 39-53.

[6] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *Proceedings of Design, Automation and Test in Europe*, 2005, pp. 178-183 Vol. 1.

[7] C. Rathgeb and A. Uhl, "An iris-based Interval-Mapping scheme for Biometric Key generation," in *Proceedings of 6th International Symposium on Image and Signal Processing and Analysis*, 2009, pp. 511-516.

[8] G. Howells, E. Papoutsis, A. Hopkins, and K. McDonald-Maier, "Normalizing Discrete Circuit Features with Statistically Independent values for incorporation within a highly Secure Encryption System," in *Second NASA/ESA Conference on Adaptive Hardware and Systems*, 2007, pp. 97-102.

[9] Y. Kovalchuk, W. G. J. Howells, H. Hu, D. Gu, and K. D. McDonald-Maier, "A practical proposal for ensuring the provenance of hardware devices and their safe operation," in *7th IET International Conference on System Safety, incorporating the Cyber Security Conference*, 2012, pp. 1-6.

[10] Y. Kovalchuk, W. G. J. Howells, H. Hu, D. Gu, and K. D. McDonald-Maier, "ICmetrics for low resource embedded systems," in *the 3rd International Conference on Emerging Security Technologies*, 2012, pp. 121 - 126.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization*, 2001, pp. 3-14.

[12] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *IEEE Micro*, vol. 23, pp. 84-93, 2003.

[13] M. Annavaram, R. Rakvic, M. Polito, J. Bouguet, R. Hankins, and B. Davies, "The fuzzy correlation between code and performance predictability," in *the 37th International Symposium on Microarchitecture (MICRO)*, 2004, pp. 93-104.

[14] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, pp. 1464-1480, 1990.

[15] STMicroelectronics. *STM32F207IG Data Sheet*. Available: http://www.st.com/internet/mcu/product/245085.jsp

[16] KEIL. *Keil µVision IDE Data Sheet*. Available: http://www.keil.com/uvision/

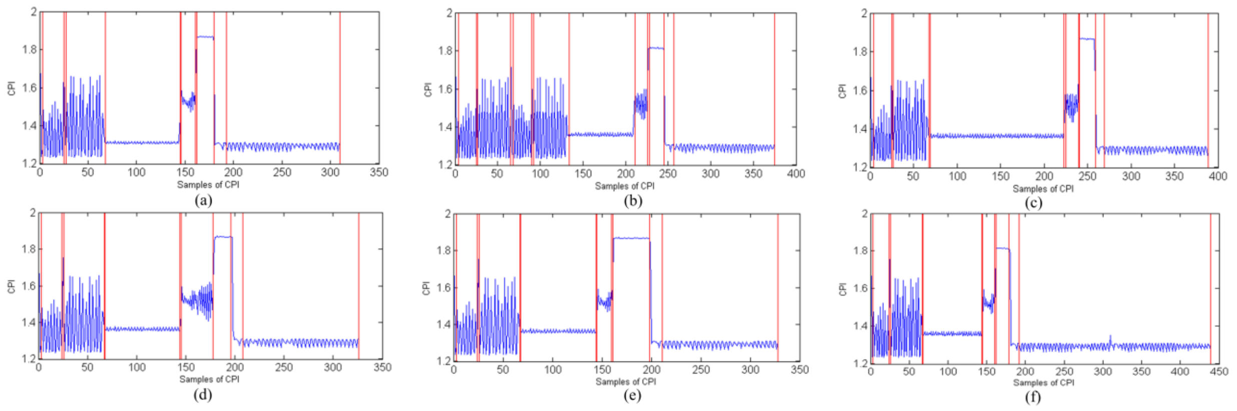


Figure 7. The results of average CPI segmentation. (a) Behaviour 1. (b) Behaviour 2. (c) Behaviour 3. (d) Behaviour 4. (e) Behaviour 5. (f) Behaviour 6.