Embedded One-Class Classification on RF Generator using Mixture of Gaussians

Ryan M. Bowen*, Ferat Sahin†, Aaron Radomski‡, Dan Sarosky‡
*Microsystems Engineering Department
Rochester Institute of Technology, Rochester, NY USA
Email: rmb3518@rit.edu
†Electrical and Microelectronic Engineering Department
Email: feseee@rit.edu
‡Advanced Development Group
MKS ENI Products
Rochester, NY USA
Email: {dan_sarosky,aaron_radomski}@mksinst.com

***Abstract*—In this paper we apply a specific machine learning technique for classification of *normal* and *not-normal* operation of RF (Radio Frequency) power generators. Pre-processing techniques using FFT and bandpower convert time-series system signatures into single feature vectors. These feature vectors are modeled using k-component Mixture of Gaussians (MoG) where components and corresponding parameters are learned using the Expectation Maximization (EM) algorithm. Data is obtained from three different generator models operating under *normal* and multiple different *not-normal* conditions. Exploration into algortihmic parameter effects is conducted and empirical evidence used to select sub-optimum parameters. Robust testing is reported to achieve a 3σ classification accuracy of 95.91% for the targeted RF generator. Additionally, a custom C++ library is implemented to utilize the learned model for accurate classification of time-series data within an embedded environment such as a RF generator. The embedded implementation is reported to have a small storage footprint, reasonable memory consumption and overall fast execution time.**

I. Introduction

The semiconductor industry's trend toward larger wafers (300mm to 450mm) for Integrated Circuit (IC) manufacturing demands reliable/available process equipment. To minimize cost of ownership (COO) IC Process tools must not fail during IC fabrication processes. For example, RF plasma power sources are critical components used during etching and film deposition. Thus, the reliability/availability of these power sources is critical to maximize fab up-time and minimize COO. Furthermore, the ability to accurately determine a process tool's operational condition in vivo has huge potential for cost saving.

Traditional fault detection systems are based on the ability to detect deviations from the normal operation of a system. Based on complexity a realization of an analytical model of a system may be difficult to define. Thus many techniques are used that attempt to describe a system through modeling. Previous work has been done with respect to modeling and classification of normal and fault conditions for RF Power generators [1], [2]. Support Vector Machines (SVM), Radial Basis Function Networks (RBFN) and Novelty detection are some of the approaches explored by Chandrashekar et al. Their work in novelty detection through a modified version of the Novelty Detection Framework (NDF) [3] has been the basis for the work presented in this paper. The preliminary results from the modified NDF have demonstrated feasibility and need for an embedded implementation of fault detection/classification.

This paper focuses on the use of a Mixture of Gaussians (MoG) and Expectation Maximization (EM) as a machine learning method for one-class classification of time-series data from RF power generators. The remainder of the paper will follow this outline: Section II defines MoG and EM, Section III explains in detail the proposed method, Section IV described the embedded implementation/application of the proposed methods, and Section V summarizes the experimental results collected.

II. One-Class Classifier

One-class classification is a unary clasification technique where information from *one* class is used to establish a boundary between the *one* class and *other* classes [4]. With one-class classification, knowledge of outlying information may or may not be available. Different methods have been used to model data for one-class classification. Most cases use probability densities such as those in Mixture of Gaussians (MoG). MoG has been used in one-class classification with fault detection and diagnosis as presented in the Novelty Detection Framework (NDF) [3]. The remainder of this section defines MoG and the use of Expectation Maximization (EM) for learning.

A. Mixture of Gaussians Model

Mixture of Gaussians (MoG) is a density based method where data is assumed to approximate a Gaussian distribution. A probabilistic measure may be used to identity target and outliers, such a measure is:

$$f(z) = I(p(x) > \theta_p) \tag{1}$$

where $p(z)$ is the probability of z, θ_p is the threshold, and I is a decision function that z is accepted as a target.

A Gaussian density in d-dimensional space, is characterized by its mean $\mu \in \mathbb{R}^d$, and $d \times d$ covariance Σ, is defined as:

$$g(x|\mu,\Sigma)=\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \quad (2)$$

A Gaussian mixture model is formalized as a weighted sum of k component Gaussian densities as in the equation below:

$$p(x) = \sum_{i=1}^{k} \alpha_i g(x|\mu_i, \Sigma_i), \text{ with}$$
$$\sum_{i=1}^{k} \alpha_i = 1 \text{ and for } i \in \{1,\ldots,k\} : g(x|\mu_i, \Sigma_i) \geq 1 \quad (3)$$

B. Expectation-Maximization

Expectation Maximization (EM) is a well known algorithm [5] that may be used to update the parameters of k-component mixture. Parameters are updated such that the likelihood of $X_n = \{x_1, \ldots, x_n\}$, with $x_i \in \mathbb{R}^d$, is not less than the previous. Basic EM is an iterative process of two steps, an expectation step (e-step) and a maximization step (m-step) as defined below. These steps are iterated until convergence.

e-step: Compute posterior probabilities.

$$w_{ik} = \frac{p_k(x_i|\lambda_k)\cdot\alpha_k}{\sum_{m=1}^{K} p_m(x_i|\lambda_m)\cdot\alpha_m}; 1 < k \leq K, 1 < i \leq N \quad (4)$$

m-step: Compute new parameter values.

$$N_K = \sum_{i=1}^{N} w_{ik} \quad (5)$$

$$\alpha_k = \frac{N_k}{N} \quad (6)$$

$$\mu_k = \left(\frac{1}{N_k}\right)\sum_{i=1}^{N} w_{ik}\cdot x_i \quad (7)$$

$$\Sigma_k = \left(\frac{1}{N_k}\right)\sum_{i=1}^{N} w_{ik}\cdot(x_i-\mu_k)(x_i-\mu_k)^T \quad (8)$$

III. Proposed Method

The method proposed for using MoG as a one-class classifier consists of four basic steps listed below and is further discussed in detail in the following subsections.

1) Data Collection
2) Pre-processing
3) Feature Selection
4) Model Learning

A. Data Collection

The database of pre-classified collected data is separated into three different sub-sets for *training*, *validation*, and *robust* testing. The importance of this separation is to use one-set of data to learn/train an estimated model, then to use the estimated model on another set of data to validate the model. The robust data is another set of data not used in training nor validation to further test accuracy of the estimated model. Given a collection of data, typical division used is 1/3 training, 1/3 validation, and 1/3 robust. If the data within the entire data collection has some sub-categorical grouping, robust data is selected such that entire sub-groups are exempt from training/validation datasets.

B. Pre-processing and Feature Selection

The data used in this work are collections of sampled time-series data sets of size $n \times m$, where m is the number variables and n is the number of samples. Each dataset in this work will be referred to as as a *fingerprint*. Looking forward to the overall application of the proposed method, the *fingerprint* is a sampled time-series data set representing a predetermined cycle of system inputs and their corresponding outputs (system signature).

The initial step of pre-processing the data is to normalize the data. For this we use a priori knowledge of maximum data ranges. Each data point in every row of the fingerprint is elementally divided by its known maximum value. The normalization compresses the data to a maximum value of 1.

For this proposed work there is an assumption that the classification of a fingerprint is related to variations in frequency components. Thus, the multi-dimensional Fast Fourier Transform (FFT) is applied on each fingerprint to obtain Fourier coefficients. In foresight of cross-platform implementation an n-point FFT is used. The n-point FFT specifies that the size of each 1D FFT computation are padded with zeros to make the input of size $1 \times N$ where:

$$N = \lceil log_2(m) \rceil \quad (9)$$

After performing the n-point FFT of the fingerprint data, the data is truncated to match the size of the original fingerprint data ($n \times m$). Considering the end target platform is an embedded environment, additional pre-processing is performed to further reduce the size of the data. For this work, we are using the bandpower of the absolute values of the FFT coefficients.

Choosing the number of elements in each band is dictated by the desired number of bands and the number of FFT coefficients (n). Our approach to the bandpower evenly distributes the number of elements in each band. Uneven division is handled by using slightly more elements within the lower frequency ranges. The number of elements in each band is calculated as:

$$\Delta = \left\lfloor \frac{n}{\beta} \right\rfloor \quad (10)$$

where n is the number of FFT coefficients and β is the desired number of bands. The number of remaining elements due to uneven band distribution is:

$$r = n - \Delta\beta \quad (11)$$

After determining Δ and r, the actual bandpower per band is accumulated as:

$$x_b[i]=\begin{cases}\sum_{j=1}^{\Delta+1}|x_f[(i-1)(\Delta+1)+j]|, & \text{if } i<=r\\ \sum_{j=1}^{\Delta}|x_f[r(\Delta+1)+(i-r-1)\Delta+j]|, & \text{otherwise}\end{cases} \quad (12)$$

where x_f is the vector of FFT coefficients, $i = 1, \ldots, \beta$, Δ is the general number of elements in each band, r is the number of low frequency band containing an extra element, and x_b is the resulting vector with bandpower values.

The final step of the proposed pre-processing is to convert the 2D dataset into a single feature vector. This is done since the overall goal is to classify the fingerprint as a whole. Thus, the 2D dataset of FFT coefficients is reshaped into a single row-wise vector of size $1 \times \beta \cdot m$, where m is the number of variables in the fingerprint and β is the number of bands used from the FFT.

After the pre-processing of fingerprint data the result is a fingerprint expressed as single feature vector. Our approach to feature selection uses Principle Component Analysis (PCA)[6]. PCA finds the linear projection of the high dimensional dataset into a lower dimensional space. The number of features selected (q) is chosen such that 99% of the variance is retained.

C. Model Learning

After pre-processing and feature selection, the process of learning and validating a model is performed. This method uses a k-component MoG model and Verkbeek et. al's efficient greedy learning to learn model parameters [7]. Verkbeek et. al's greedy method builds a mixture component-wise by (i) inserting a new component and (ii) applying EM until convergence. The stopping criterion chosen in this work is a pre-specified number of components (k). Using a training dataset, the resulting learned model parameters from Verkbeek et. al's greedy method are:

- $M(k \times q)$ - mean values of the mixture.
- $C(q \times q)$ - co-variance matrix of mixture.
- $W(k \times 1)$ - vector of mixing weights.

where q is the number of features in each feature vector of the training dataset. Here it is important to note that the training dataset contains only data that has been expertly classified as *normal*.

The fitness of the learned model parameters with respect to a single feature vector (x) is quantified through calculation of the likelihood that feature vectors belong to the mixture. First each feature vector's likelihood of belonging to component k is determined as:

$$L_k(x) = \frac{2\pi^{-q/2}}{|C_k|} exp\left(-\frac{1}{2}\sqrt{(x-M_k)\,C_k^{-1}\,(x-M_k)^T}\right) \quad (13)$$

Then the log-likelihood that a feature vector belonging to the entire mixture is:

$$L(x) = \sum_{i=1}^{k} log\,(L_i(x) \cdot W) \quad (14)$$

After calculation of each feature vector's log-likelihood, a crisp classification decision of *normal* or *not-normal* is achieved via thresholding. For this a threshold value is chosen based on the variance in log-likelihood values such as:

$$th = \mu_L - \delta\sigma_L \quad (15)$$

where μ_L, σ_L are the mean/standard deviation of log-likelihood values and δ may be chosen. Since all training data is *normal*, a large $\delta(> 6)$ will yield high training accuracy but may not be suitable for separation of *not-normal* from *normal*. The overall classification accuracy or performance for a dataset is determined based on number of False Positives (FP) and False Negatives (FN), as below:

$$performace = \frac{\#FP + \#FN}{N + NN} * 100 \quad (16)$$

where N and NN are the number of known *normal* and *not-normal* feature vectors in the dataset.

After training of the model parameters, cross-validation is performed. The validation dataset is separate from training and contains expertly classified data of both *normal* and *not-normal* data. The learned model parameters (M, C, W) in addition to the threshold th are used with the validation dataset. As with validation, robust testing is done on an additional dataset not used in validation and training. This is done to mimic the event that *new* data, not available during the learning process, is to be classified using the learned model and parameters.

IV. Embedded Solution

The proposed method was originally implemented using MATLAB®computing software because of its ability for rapid algorithm development and extensive toolboxes and libraries. However, the end goal is to implement the one-class classifier within the target embedded system. C++ was chosen as the language for the embedded implementation for its speed, multi-platform support, and object-oriented capabilities. Additionally, no 3rd-parties libraries are used in order to avoid platform dependencies with additional libraries. Moreover the C++98 standard was selected for backward compatibility with legacy systems with compilers that do not support C++0x standards.

Without 3rd party libraries custom math classes and functions were implemented. Among the custom math classes were matrices and vectors for real and complex numbers. Mathematical operators were implemented/overloaded for matrix multiplication, scalar multiplication/division, element-wise multiplication/division/addition/subtraction, and transpose. Some additional operations implemented are n-dimensional sum, mean, max, min, variance, and co-variance. All operations and operators were implemented to conform to MATLAB syntax and validated for output.

A. FFT

The most difficult and computationally exhaustive aspect of the proposed work is the use of the FFT. There multiple

implementations of the FFT in existence. For initial testing the generic 2D Discrete Fourier Transform (DFT) were used. The computation of the 2D DFT was done as follows:

$$F[u,v] = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m,n]\, e^{-j2\pi\left(\frac{k}{M}m + \frac{1}{N}n\right)} \quad (17)$$

where f is a sampled periodized signal with size $(M \times N)$ and F is the transformed signal with $u = 0, \ldots, M-1$ and $v = 0, \ldots, N-1$. The DFT was implemented and tested against MATLAB output.

However, the DFT is very slow considering the 1D DFT has a time complexity of $O(N^2)$. Therefore, the Fast Fourier Transform FFT was considered since it is able to reduce the time complexity in 1D to $O(NlogN)$. The Cooley-Tukey implementation of the FFT was selected and implemented using C++. The particular implementation uses a recursive function in combination with a reference variable as listed in Algorithm 1. The 2D form of the FFT is done by using the 1D FFT row-wise then column-wise.

Algorithm 1 Cooley-Tukey Recursive FFT Algorithm

function FFT($\&x$) ▷ reference to complex array
  $N \leftarrow \text{length}(x)$
  if $N \leq 1$ **then**
    return 1
  end if
  ▷ Divide into even/odd
  $even \leftarrow x[0, 2, \ldots, N-1]$
  $odd \leftarrow x[1, 3, \ldots, N-1]$
  ▷ Conquer
  FFT(even)
  FFT(odd)
  ▷ Combine Results
  for $k = 0$ to $N/2$ **do**
    $t \leftarrow exp(-2\pi k/N) * odd[k]$
    $x[k] \leftarrow even[k] + t$
    $x[k + N/2] \leftarrow even[k] - t$
  end for
end function

B. Architecture

The purpose of the embedded implementation of the MoG one-class classifier is to provide functionality for a system to collect fingerprint data and determine if the system is operating under normal or not-normal conditions. According to the proposed method as described in Section III, this requires data collection, pre-processing, feature selection, and model learning.

For the embedded implementation many of the system variables are obtained offline as seen in Fig. 1. Most importantly the k-component MoG is obtained from offline learning conducted in MATLAB. Where the model is defined by a $(k \times q)$ mean matrix M, $(q \times q)$ co-variance matrix C, and $(1 \times k)$ weight matrix W; with q being the number of selected features resulting from PCA. In addition to the model information, algorithmic and pre-processing parameters are determined offline and hard-coded. These other parameters include: maximum values for the data, number of FFT bands (β), PCA matrix, and a learned threshold value th. The overall algorithm of the embedded implementation is described in 2.

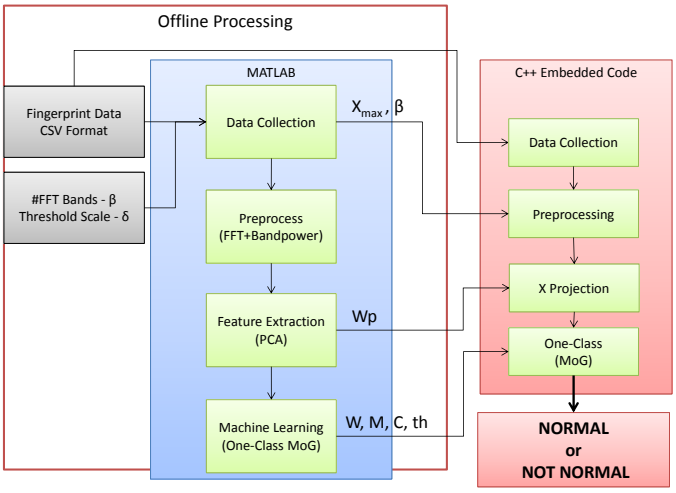

Fig. 1. Block diagram of the embedded system implementation in combination with offline (pre-processed) data.

Algorithm 2 Embedded MoG One-Class Classifier

▷ fingerprint data, processing and model parameters
procedure ONE-CLASS($x, \beta, W_{pca}, M, C, th$)
  $x_n \leftarrow norm_{max}(x)$ ▷ Normalize by max values
  $x_{fft} \leftarrow abs(FFT(x_n, n))$ ▷ abs of n-point 2D FFT
  $x_b \leftarrow bandpower(x_{fft}, \beta)$ ▷ Compute bandpower
  $x_r \leftarrow reshape(x_b)$ ▷ Reshape to array
  $x_{proj} \leftarrow x_b * W_p$ ▷ Feature selection
  $x_l \leftarrow likelihood(x_{proj}, M, C)$ ▷ Calculate likelihood
  if $x_l \geq th$ **then**
    return 0 ▷ Normal
  else
    return 1 ▷ Not-Normal
  end if
end procedure

V. Experimental Results

Primary focus of the experimental results was with the MKS®LVG3527 27MHz RF generator. Fingerprint data was collected from other generators including the MKS LVG3560 60MHz and MKS Keinos 2MHz RF generators. Fingerprint data was collected while generators were operating under known normal conditions as well as some non-normal or *faulty* conditions. For the MKS LVG3527 27MHz and MKS LVG3560 60MHz models, faulty data was seeded from known faulty condition as listed below:

1) Lacking Solder on PA FET
2) Suspect PA
3) Lacking Solder on Resistor

TABLE I
DATA, VARIABLE, AND SAMPLE COUNTS FOR FINGERPRINT DATA USED IN OFFLINE MODEL LEARNING.

RF Generator	Fingerprint Count		Vars	Samples
	Normal	*Faulty*		
MKS LVG3527 27MHz	2037	1259	33	220
MKS LVG3560 60MHz	2343	686	33	220
MKS Keinos 2MHz	1292	0	29	220

The total number of fingerprints are summarized in Table I as well as the total number of variables and samples per fingerprint. There was no fault data currently available for the MKS Keinos 2MHz models.

The process of data-collection, pre-processing, feature selection, and model learning was done as discussed in Section III. The overall process is dependent on a set of parameters including: number of FFT bands (β), PCA variance, number of Guassian components (k), and EM likelihood threshold value (th). The PCA variance was fixed to be 99% and the other values determined based on experimental trials and changing one factor at a time. For all of the experimental results listed, the data was shuffled and each data point reported is the result of 1000 trials.

A. Factor Selection

Using the MKS LVG3527 27MHz RF generator's data, the number of bands for FFT β and EM likelihood threshold value th were the first parameters explored. For this, the number of components was fixed to $k = 1$ and number of FFT bands varied $\beta = [10, 20, \ldots, 100]$. From Fig. 2, it can be seen that a th with values $\mu - \sigma$ and $\mu - 2\sigma$ have poor classification performance. However th values of $\mu - 3\sigma$ and $\mu - 6\sigma$ have very high classification performance with 3σ classification accuracy near 95%. Fig. 3 provides a closer looker at the classification accuracy with respect to False Positive (FP) and False Negative (FN) counts. From Fig. 2 and 3 for the MKS LVG3527 27MHz RF generator data, the best number of FFT bands is $\beta = 60$ and $th = \mu - 3\sigma$; this is where the number of FP are minimized.

After determining β and th, the number of Guassian components k was explored. By fixing $\beta = 60$ and $th = \mu - 3\sigma$, k was varied with $k = [1, 2, \ldots, 30]$. Fig. 4 shows the 3σ robust classification accuracy, where the best accuracy has $k = 11$.

B. Robust Performance Results

As described in Section III, the data selection process during model learning splits the data into three sets for training, validation, and robust testing. The training data is used to train the MoG and the validation and robust data are used for testing the model. With respect to the RF generator data, the robust data-set is selected such that it contains data from generators that are not used in training or validation, to mimic using the learned model on a new generator not used in off-line training. From the factor exploration results in Section V-A obtained using data from the MKS LVG3527 27MHz RF generator, parameters of $\beta = 60$, $th = \mu - 3\sigma$, and $k = 11$ were selected.

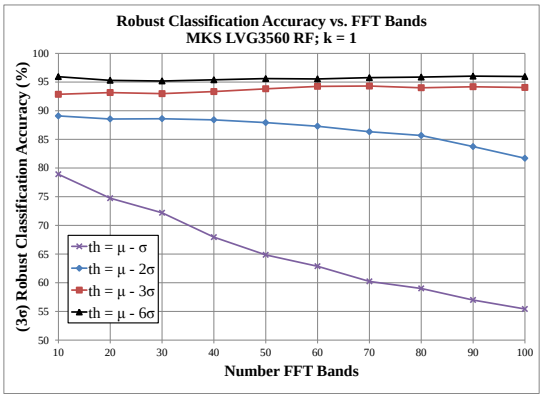

Fig. 2. 3σ robust classification performance with respect to number of FFT bands (β) and threshold value (th). Results are for single Gaussian component ($k = 1$) for the MoG and reported value from 1000 runs of shuffled data.

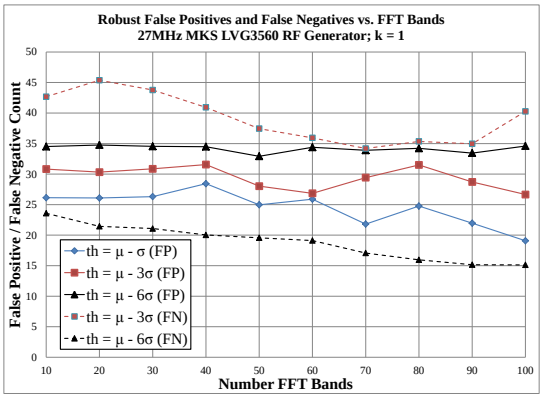

Fig. 3. False Positive (FP) and False Negative (FN) counts with respect to number of FFT bands (β) and threshold value (th). Results are for single Gaussian component ($k = 1$) for the MoG and reported value from 1000 runs of shuffled data.

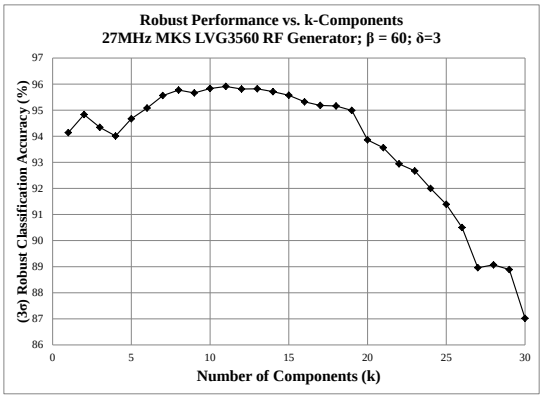

Fig. 4. 3σ robust classification performance with respect to number of Gaussian components k with number of FFT bands ($\beta = 60$) and threshold value ($th = \mu - 3\sigma; \delta = 3$). Reported values are from 1000 runs of shuffled data.

TABLE II
AVERAGE AND STANDARD DEVICE OF ROBUST PERFORMANCE AND FP/FN COUNTS COLLECTED OVER 1000 RUNS OF SHUFFLED DATA.

RF Generator MKS	**Feat**	**Robust Testing**		
		Performance	FP	FN
LVG3527	56	98.70 (0.93)	8.69 (8.04)	2.82 (4.2)
LVG3560	44	99.03 (1.24)	1.46 (2.58)	5.63 (8.26)
Keinos 2MHz	61	92.32 (6.51)	0.00 (0.00)	24.53 (20.93)

These parameters were then used to train, validate and perform robust testing on data from each of the three models, the performance results are listed in Table II. The results in Table II, are averages and standard deviation over 1000 trials, also listed are the number of features selected as a result of using PCA. The MKS LVG3527 27MHz and MKS LVG3560 60MHz were reported high performance even considering the parameters may not be optimum for the 60MHz model as they were chosen based on 27MHz data. The MKS Keinos 2MHz performance was significantly less with a higher std. This is most likely due to difference in variable count and potentially non-optimal parameters. From the results in Table II, it is suggested that parameters should selected/optimized per model. However, note that 0 FP were reported for the MKS Keinos 2MHz model, which is to be expected when no faulty data is used in training.

C. Embedded Results

For testing of the embedded implementation of the one-class classifier the chosen platform was a BeagleBone Black (BBB) with a 1-GHz Sitara™ARM®Cortex-A8 running a 32-bit Ubuntu 13.04 distribution. The OS was set to run on the BBB's embedded memory (eMMC) which offers a total of 2GB of space. The embedded code developed in this work was compiled with g++ 4.7.3 with C++98 standard and no extra optimization. Since the BBB is not one of the RF generators, fingerprint data files were uploaded to the BBB to simulate data collection. The one-class classifier read the data files then performed the algorithm and output to console classification of the file(s) as normal or not-normal. The resulting implementation classification output matched that of the MATLAB version.

Within an embedded environment importance is placed on code footprint (static memory required), dynamic memory, and speed of execution. Table III is a summary of recorded memory footprints and execution time. Execution time is the total time to classify a fingerprint, which includes time to read fingerprint data from a file as well as model information. The dynamic memory was determined by monitoring process memory consumption and recording the maximum memory consumed during execution. Reported measures are based on the model using a single Gaussian component ($k = 1$), future testing will include multiple components. Based on the memory analysis there exists no physical constraints for the code to be ported over to an RF generator.

TABLE III
EMBEDDED IMPLEMENTATION'S CODE FOOTPRINT, MEMORY USAGE AND ALGORITHM EXECUTION TIME.

Total Execution Time	600ms - 950ms
Model Information Footprint	113KB
Source Code Footprint	60KB
Executable Footprint	76.8KB
Total Code Footprint	250KB
Dynamic Memory Usage	1.1MB

VI. CONCLUSION

The proposed work is an implementation of a one-class classifier using MoG. The algorithm has been tested against time-series (fingerprint) data collected from three different MKS RF generators models. Preliminary results are promising demonstrating high 3σ robust classification accuracy ($> 95\%$). It has been demonstrated that algorithmic parameter selection is an important aspect, especially threshold selection.

Offline model training was conducted with code development in MATLAB. The MATLAB implementation has been converted to C++ and has been tested on an embedded platform. Using an off-line trained MoG and other supporting parameters, the embedded implementation has been completed. The execution time ($< 1sec$), code footprint ($205KB$), and memory requirement ($1.1MB$) have been reported and are reasonable for most modern embedded platforms.

Future work will focus on optimization of algorithm parameters, where additional machine learning techniques and/or statistical analysis will be used to select parameters. The parameter selection and performance will be compared against other time-series data-sets that are well known within the classification field. After optimization of such parameters, online MoG model learning will be explored as a robust in vivo health monitoring system for embedded systems.

ACKNOWLEDGMENT

This work was supported by MKS ENI Products.

REFERENCES

[1] G. Chandrashekar and F. Sahin, "In-vivo fault prediction for RF generators using variable elimination and state-of-the-art classifiers," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Seoul, Korea: Ieee, Oct. 2012, pp. 1800–1805.

[2] G. Chandrashekar, F. Shain, E. Cinar, A. Radomski, and D. Sarosky, "In-Vivo Fault Analysis and Real-Time Fault Prediction for RF Generators using State-of-the-art Classifiers," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Manchester, UK, 2013.

[3] D. P. Filev, R. B. Chinnam, F. Tseng, and P. Baruah, "An Industrial Strength Novelty Detection Framework for Autonomous Equipment Monitoring and Diagnostics," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 767–779, Nov. 2010.

[4] R. Tax, DavidM.J. and Duin, "Combining One-Class Classifiers," in *Multiple Classifier Systems*, ser. 2096, F. Kittler, Josef and Roli, Ed. Springer Berlin Heidelberg, 2001, pp. 299–308.

[5] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society . Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[6] E. Alpaydn, *Introduction to Machine Learning*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2010.

[7] J. J. Verbeek, N. Vlassis, and B. Kröse, "Efficient greedy learning of gaussian mixture models." *Neural computation*, vol. 15, no. 2, pp. 469–85, Feb. 2003.