Received: 17 April 2015 | Revised: 1 June 2016 | Accepted: 7 January 2017

DOI: 10.1111/exsy.12199

WILEY Expert Systems

ARTICLE

An intelligent system to monitor refrigeration devices

Damjan Kužnar | Rok Piltaver | Anton Gradišek | Matjaž Gams | Mitja Luštrek

Department of Intelligent Systems, Jožef Stefan Institute, Jamova Cesta 39 Ljubljana, 1000, Slovenia

Correspondence
Damjan Kužnar, Department of Intelligent Systems, Jožef Stefan Institute, Jamova Cesta 39, 1000 Ljubljana, Slovenia.
Email: damjan.kuznar@ijs.si

Abstract

Refrigeration systems have been a vital component of our lives for more than a century. Apart from storing food, they are used to store sensitive goods such as pharmaceutical products or reactive chemicals. The deterioration of the refrigeration system performance due to aging or malfunction directly affects the quality of stored goods. Therefore, an early detection of deviation in performance is an important task. This paper presents a system that monitors operation of refrigeration devices and alerts the user to possible irregularities in the operation run. The emphasis is on recognition of gradual changes of performance that indicate upcoming hardware problems. The system consists of 2 modules: human-defined expert rules and machine learning. The machine-learning module learns to recognize abnormal behaviour of devices automatically. Furthermore, it can distinguish between different abnormal events and allow the user to classify some of the types as normal, so that they not longer raise an alarm. The machine learning was evaluated by comparing its recognition of abnormal events and classification accuracy of such events to the performance of a human operator. The system can in principle be adapted to any electronic device that periodically applies some system for sustaining a predefined quality (e.g., temperature).

KEYWORDS

event classification, event detection, intelligent systems, machine learning, refrigeration devices, time series

1 | INTRODUCTION

Careful control of temperature inside refrigeration devices is crucial for management of stored goods, such as food (frozen food, seafood, or fresh produce), pharmaceutical products, chemicals, photographic film, etc. Ready-to-eat foods need to be properly stored in order to prevent growth of pathogens, such as *Listeria monocytogenes* in milk. When using refrigeration as a means of storing, a strict control of temperature is required, so that the temperature of the product never exceeds 6 °C (preferably 2–4 °C), in order to ensure that growth of pathogens to any significant degree does not occur before the product is consumed (Alimentarius, 2009). Refrigeration systems for vaccine storage must adhere to strict regulations (CDC,), where temperature inside the refrigerator must be maintained between 2 and 8 °C. In case of temperature excursions outside the defined temperature interval, an alarm is triggered and the operator is required to check temperature history in order to determine whether the vaccine is still safe to use or should be discarded. Many organic chemicals in the laboratory need to be kept refrigerated as well, in order to prevent further chemical reactions from occurring. Some widely used solvents, such as the highly volatile and extremely flammable diethyl ether, are often stored under cool conditions. Unprocessed photographic films can be damaged by high temperatures, and their photographic characteristics gradually change after manufacture—users are therefore advised to store professional color films at 13 °C or lower. For long-term storage of motion-picture film, temperatures down to −23 °C are recommended (Kodak, 2005).

In legacy refrigeration system, monitoring the temperature meant that the operator periodically checked a thermometer placed in the refrigerator and kept records on paper. Due to such sparse sampling, there were no records of what was happening to the temperature in the interim, therefore leading to discarding all the contents for safety reasons in case of refrigerator malfunction. Modern systems use continuous monitoring instead. As opposed to manual temperature inspection, continuous monitoring is more reliable, keeps permanent records, and reduces the risk of human error. As such, these systems are much better suited to supervise storage of sensitive products.

Because continuous monitoring system tracks the temperature in intervals on the order of magnitude of a second, detection of irregularities can in principle be instantaneous. In general, temperature deviations in refrigeration devices can be assigned to one of the following types: (a) excursion out of predefined temperature limits caused for example by an open door, (b) irregular patterns that may be connected to some system component deterioration, or (c) continuous large deviations from the average temperature that are still within the temperature limits but nevertheless affect the quality of the stored product.

Typically, commercially available devices have preset upper and lower temperature limits and an alarm is triggered in case of temperature excursions outside the defined interval. Some commercial systems employ additional predefined expert rules. For example, LabWare LIMS (LabWare,), an advanced laboratory management system, comes equipped with a set of rules and includes a module that allows the user to add or modify the existing rules according to their needs. Although these rules detect some of the most common irregular events (such as temperature excursions), they will not recognize most peculiar behaviour patterns that may affect the operation run of the system, such as those resulting from an upcoming malfunction of some of the components. To detect such behaviour, we required methods using advanced knowledge of signal processing. Often, the implementation of such complicated rules is either beyond the user expertise or the advanced mathematical functions required (such as Fourier transform or autocorrelation functions) that are not available in the programming module. Additionally, listing all types of unusual events and writing special rules for them is not possible in advance because new discrepancies may occur. Therefore, instead of fixed expert rules, a more suitable approach for complex event detection may be analysis of time series. Recently, Kang, Belušić, and Smith-Miles (2014) studied the possibilities of extracting and clustering events from time series without previous knowledge of their generating mechanisms. They used different algorithms for event detection and clustering. The main focus of their work is time series that consist of random noise and events, more specifically, atmospheric data gathered in the upper atmosphere. This differs from our problem, where the signal is usually not random noise but has distinctly observable periodic characteristics. They also used hierarchical clustering for the detected events: For each event, they extracted several features (standard deviation, nonlinearity, serial correlation, etc.) and then used Euclidean distance in the clustering process. In contrast, our work does not rely on any particular time-series features, but rather uses dynamic time warping measure to compute the distance between events directly, because event characteristics are usually directly observable and are not masked with high degree of noise. Similar work is also reported by Preston, Protopapas, and Brodley (2009) who dealt with detecting events in astronomical time series that also exhibit a high degree of noise and are nonperiodic. Sharifzadeh, Azmoodeh, and Shahabi (2005) exploited wavelet footprints to capture discontinuities in a signal. However, their work deals with nonperiodic signals, so we can assume that when applied to periodic signals (such as refrigerator temperature fluctuations), the wavelet footprints would detect periodic signal fluctuations as discontinuities.

Several patents related to our work have also been published. Chiu and Schneider (1986) patented an over-temperature warning system for refrigerator appliances aimed at avoiding damage to perishable items. Its monitoring capabilities are fairly simple and consist of two temperature thresholds and mechanism that raises alerts in cases when temperature raises above the thresholds. Sharood and Carr (2002) patented a refrigeration monitor unit that is aimed at retrofitting existing appliances and can alert the user of conditions within the appliance in which food spoilage may occur. Although with similar goals as our work, the patent provides no description of the monitoring methods and algorithms used; therefore, no direct comparison can be made. Ramey, Rozsnaki, and Osman (2008) patented a fault detection and diagnosis for refrigeration systems using distributed microsystems that determines whether a refrigeration (or other cooling)-related system is operating within normal parameters by comparing the current system's enthalpy curve to the reference enthalpy curve derived from ideal or normal conditions. This implies that additional sensor for pressure is needed and introduces additional complexity to the system. Moreover, because it monitors the refrigerant throughout the various steps of the thermal process rather than the actual temperature in the refrigerated compartment, it cannot detect events due to external factors, for example, opening refrigerator doors and placing a warm object in the refrigerator.

In our work, we focused on creating a sophisticated temperature monitoring system that not only notifies the user of irregular events when they occur but also recognizes changes in the refrigeration system behaviour or other unusual events that may signal an upcoming component malfunction. By detecting such events, maintenance can be scheduled for the system—thus preventing the imminent malfunction and possible loss of stored goods. Our patented system, called iLab (Kužnar, Gams, Marinčič, Lotrič, & Čufar, 2012), is based on intelligent computer algorithms that detect events, classify them, trigger alarm when necessary, and, furthermore, take advantage of the user feedback to learn which types of events are normal and which are not.

The rest of the paper is organized as follows. In Section 2, we describe the task we are tackling in detail. In Section 3, we first describe the iLab system and the two modules it contains: the expert rules module, in which rules are defined by a human expert, and the artificial intelligence (learning) module. In Section 4, we compare the performance of the learning module to a human operator. We demonstrate that the artificial intelligence module is equally accurate as the human operator and detects events that slip the detection of expert rules. In Section 5, we conclude the paper with an overview of the strenghts of the iLab system and the results of its evaluation.

2 | TASK DESCRIPTION

The task is to recognize different types of unusual events in the operating run of a refrigeration device by continuous monitoring of the inside temperature in order to prevent decrease of quality or loss of the content. The sources of the events can be classified into three categories:

- *User*: these types of events are caused by the user's interaction with the refrigeration device. These events can be a result of a normal operation, such as opening doors or abnormal operation, such as accidentally leaving the door open, turning off the refrigerator, etc.
- *Device*: events that are caused by some sort of device failure or change in operation that causes the device to operate differently. The difference in operation can be significant (e.g., coolant compressor malfunction) or subtle (e.g., crack in the door sealing).
- *Environment*: these types of events are a result of events from the device's environment and can be anything from electricity power outage, communication network failure, natural disasters (e.g., floods), heat wave, etc.

As said, the system's goal is to detect any kind of event that is reflected in the temperature inside the refrigeration device. Therefore,

many of the environmental events cannot be detected, because for instance in the event of the power outage, the iLab system would also be affected. However, these extreme types of events are usually detected by some other means. On the other hand, environmental events such as heat waves would probably be reflected in the device's inside temperature as more frequent cooling cycles. The main focus of iLab system is to detect device operational changes and user-generated events that are subtle and therefore difficult to detect by other means.

During the normal operating cycle, the temperature inside the refrigerator has characteristics of a periodic signal. When the temperature rises above a certain value, the cooling unit switches on and cools the insides to a set value. This periodic behaviour can be interrupted by external influences of several types (see real data examples in Figure 1):

(a1) Temperature spikes, indicating abrupt changes of the temperature due to external reasons. It should be noted that spikes may appear in some refrigeration systems that use defrost cycle (periodically heating the evaporator coil in order to melt the frost that has formed on it). Depending on the system, the defrost process can run either several times per day or on a couple of days' interval. If these defrost spikes do not exceed the upper temperature limit, they do not count as anomalous events.

(a2) Temperature excursion outside the operating temperature interval for a time longer than ΔT, which may occur due to scenarios such as an operator forgetting to close the door, putting a large warm object into the refrigerator, a power outage, etc. Both a1 and a2 types can be detected by simple rules, such as

$$\textbf{if } (T > T_{high}) \textbf{ or } (T < T_{low}) \textbf{ then } \text{alarm}, \tag{1}$$

where T_{high} and T_{low} are the upper and lower temperature limits, respectively.

(b1) Changes of the average temperature during the normal operation run may indicate set-temperature drifts due to software or hardware problems (unless these values were manually reset).

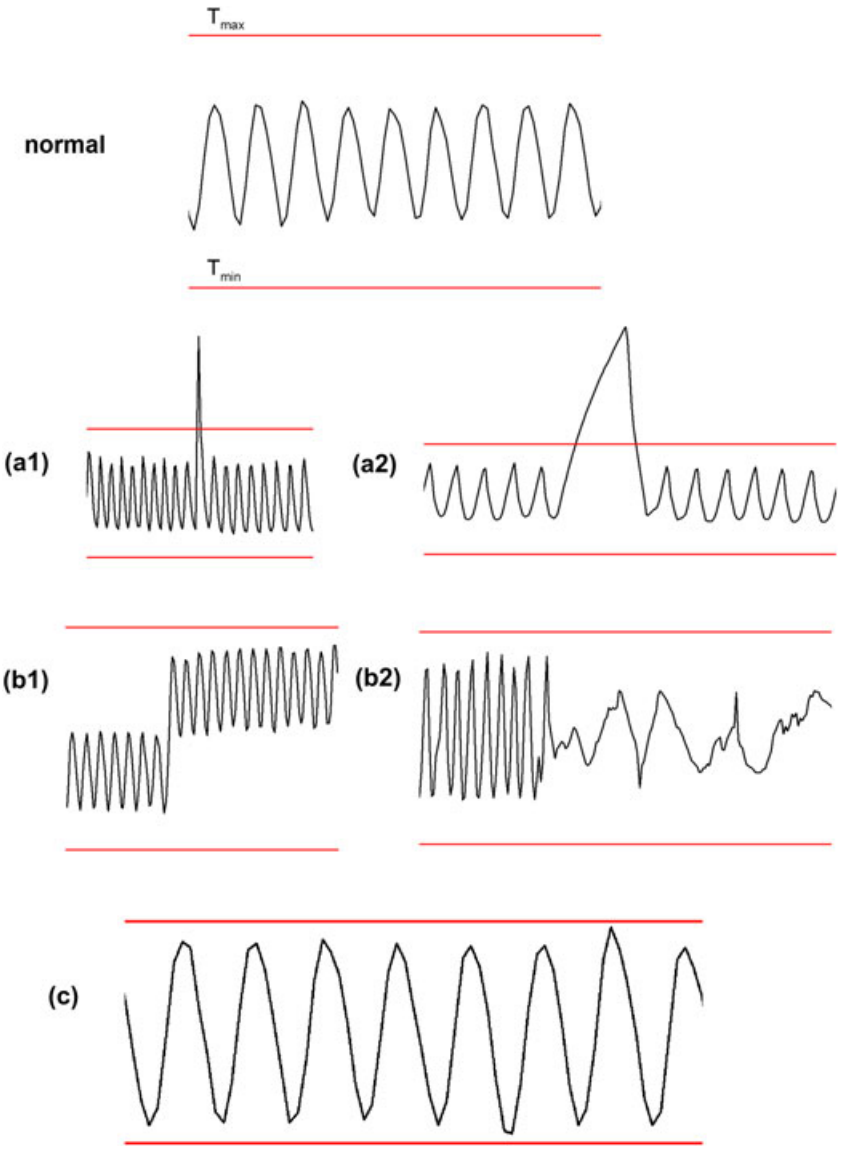

FIGURE 1 Real data examples: normal operating run (top), followed by different types of anomalous events: (a1) spike, (a2) temperature excursion, (b1) temperature drift, (b2) disruptions in the operating cycle, and (c) increased oscillation amplitude. Temperature limits T_{max} and T_{min} are marked by horizontal red lines for each case

(b2) Changes in periodicity of the normal operating run may indicate problems with the cooling system, such as cooling gas leakage or compressor malfunction.

(c) Faster oscillations or oscillations with amplitudes larger than desired but still within temperature limits may affect the quality of stored goods. For example, it has been demonstrated that fluctuations in temperature decrease the mechanical properties of frozen food (potatoes; Alvarez & Canet, 1998). The reason is periodical sublimation and freezing of water, leading to formation of ice crystals that damage the tissue at cellular level.

(d) Other events, such as problems in communication between the sensor and the computer. This may be seen as sudden appearance of invalid values or a constant temperature value over a longer time interval. We do not address these types of events in this work.

3 | SYSTEM ARCHITECTURE OVERVIEW

The iLab system monitors the data about a laboratory environment and informs the user about unusual events. Although it has only been used on refrigeration data so far, which is also the subject of this paper, the system can in principle monitor any type of time series. Figure 2 shows the main components of the system and related dataflow. Data about the laboratory environment are gathered using the appropriate sensors. It is sent to the iLab system via a network and the necessary middleware. The system stores the data into a database and processes it in order to detect unusual events (i.e., type *a*, *b*, or *c*). Upon a detected unusual event type, an alarm or a warning is written into a database and displayed to the user via a graphical user interface (e.g., sent to a mobile device). The user then either confirms the event or declares it as a false one in order to improve the system's performance by learning from the user's feedback.

Unusual events are detected with two independent complementary modules. The module based on rules detects unusual events by checking whether the current sensor data complies with the user-defined rules about sensor values and trends. Rules predominantly check events of type *a*; however, some events of types *b* and *c* are also possible to define by rules using more complex formulations. The rule-based module also enables automatic real-time checking of laboratory environment parameters as prescribed by law. Furthermore, it detects critical sensor and network malfunctions and offers the users a way to intuitively define events that should be detected by the system, which increases the users' trust in the system. The advantage of the rule-based module is its simplicity, which enables the user to understand how it functions, thus increasing its reliability. Rules are written as general mathematical expressions over the time series, as an appropriate trade-off between their generality on one hand and simplicity in terms of the time required by the user to learn it on the other hand. Details about rule language and its implementation are presented in subsection 3.1.

The problem of the rules is that they are designed to cope with typical situations that experts are aware of and can be described by the rule language. However, in real-life, unexpected events will eventually occur, and the detection of these is the task of the second module. It primarily deals with *b*- and *c*- type events, though it also detects *a*-type events. This module is based on unsupervised machine learning (ML). It first builds a model describing typical sensor values and trends, which is later updated in real time and used to check whether the current sensor data complies with the learned model. A significant deviation from the model is treated as an unusual event and displayed to the user as an alarm. The learning module supplements the rule-based module and mitigates its limitations. Because it is fully automatic and does not depend on user input, it can detect unusual events even if the user defines inappropriate rules (e.g., with too loose thresholds). It is also useful when the user forgets to or is not able to constrain sensor values or trends using the rules' module. Furthermore, it is able to detect an unusual event sooner than the rule-based module because it can detect an unusual trend even before the sensor values exceed the threshold prescribed by the user-defined rules. We use the unsupervised-learning approach in order to decrease the necessary user interaction: The goal of the system is to relieve the users from the tedious work and not to burden them with additional work. However, this approach may lead to some false alarms. Although the cost of false alarm is usually much smaller than the cost of unnoticed unusual event, a high number of false alarms decrease the user's trust in the system, which in extreme cases causes the user to ignore the system-triggered alarms altogether. Therefore, the learning module is extended with a supervised-learning capability. It groups the unusual events by their similarity using a clustering algorithm and learns which clusters of unusual events are regarded as false alarms by the user. If it detects an unusual event that belongs to a group of false alarms, it

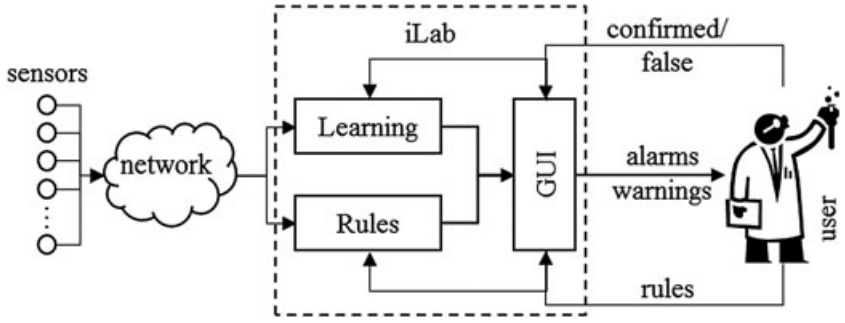

FIGURE 2 Main components of the iLab system and related dataflow. The expert module triggers alarm when rules are breached whereas the learning module warns the user of possible unusual behaviour when performance characteristics change. User's feedback is used to improve the learning module's performance. GUI = graphical user interface

does not trigger an alarm. For example, if a "tabula rasa" system monitors the temperature in the refrigerator, it will detect opening the doors of the refrigerator as an unusual event. The user will label the alarm related to the refrigerator door opening as false, and the system will eventually learn not to trigger the alarm when the door is opened. Nevertheless, it will still trigger an alarm if the refrigerator door is left open for an extended time or if the refrigerator compressor fails, because those events do not belong to the group of normal door opening events.

3.1 | Expert rules

The rule-based module is one of the two unusual event-detection modules in the iLab system. This section presents a technical overview of the module, including the language to define the rules and the implementation of these rules.

Its advantage is that it is reliable and easy to understand, which increases the user's trust in the system, and easily customizable, which enables the users to tailor the system according to their specific needs. During the initialization of the system, the user enters the rules in the form of (in)equations about the time series that represent the monitored environment parameters; for example, $T > 18$: The measured temperature must be above 18 °C (or F, depending on the units used). Thereafter, the rule-based module detects unusual events by checking whether the current sensor data complies with the user-defined rules. Using time-series (in)equations as the rule language is convenient for two reasons: the (in)equations are very expressive and general, which enables a high level of customization, and most users are familiar with such representation, therefore, they can start using the system with a minimal amount of training. The rule language offers common mathematical operations (addition, subtraction, multiplication, division, and exponentiation) and functions (logarithm, absolute value, square root, and trigonometric functions), comparison operators (equal to, not equal to, greater than, less than, greater than or equal to, and less than or equal to), constants, and variables, which correspond to time-series values relative to the current time. In addition, functions over intervals of time series can be used; those are minimum, maximum, average, sum, standard deviation, and slope of a linear fit. Compiler construction techniques (Aho, Sethi, & Ullman, 1986) were used in implementation of the rule-checking algorithm. The algorithm proceeds in three sequentially execute phases: lexical analysis (i.e., lexing), syntax analysis (i.e., parsing), and evaluation. Lexical analysis removes white spaces and recognizes function names, variable names, numeric constants, and lexical errors. Syntax analysis identifies the order of operations (e.g., additions and multiplications) to be performed in the evaluation step, which finally determines whether the time-series data comply with the rule. The three phases are described in more detail in the following paragraphs.

Lexical analysis breaks a rule represented as a sequence of characters (e.g., letters, digits, whitespace characters, brackets, etc.) into pieces called tokens. Each token is a single atomic unit of the rule language: an operator, function name, numerical constant, or variable name. For example, the inequation abs $(h - h[8]) < 10$ is broken into the following tokens: abs, (, h, –, h, [, 8,],), <, 10. The token syntax is a regular language; therefore, a finite state automaton is used to recognize lexically correct rules and output the sequence of tokens or detect the position of the first lexically nonvalid character in the rule. The tokenization algorithm—implementing an efficient finite state automaton based on the regular expression representing the token syntax—was generated using the JFlex tool (JFlex, 2014). The inputs to JFlex are pairs of regular expressions and Java code. When the generated tokenization algorithm finds text matching a regular expressions, it executes the corresponding Java code that returns the recognized token. Equations 2 and 3, for example, show the regular expressions representing a numerical constant (e.g., 12.03) and a variable name (e.g., tempFridge12), respectively, in the IEEE POSIX Extended Regular Expressions standard notation.

$$(0|[1-9][0-9]*)|(0|[1-9][0-9]*)\.[0-9]*, \tag{2}$$

$$[A-Za-z][A-Za-z0-9]*. \tag{3}$$

The next phase of checking a rule is syntax analysis. It parses the token sequence to identify the syntactic structure of the rule and builds a parse tree, which replaces the linear token sequence with a tree structure. The parse tree defines the operands and order of operations on them to be executed when evaluating the rule. The parse tree is built according to a formal grammar, which defines the rules' syntax. If the rule does not belong to the language corresponding to the formal grammar, an error and its location in the rule character sequence are returned instead. The syntactic analysis is done by parsing algorithm, called parser, which was generated using the CUP tool (CUP, 2014). The generated LALR(1) parser is implemented as a pushdown automaton. Figure 3 shows the context-free grammar in Backus-Naur Form used to generate the parser (nonterminals are in italics).

Figure 4 shows a parse tree generated for the rule: abs $(h - h[8]) < 10$. In order to calculate whether the inequality expressed by a rule holds, we evaluated the values in the parse tree nodes in a recursive bottom-up order. The value(s) of a node representing a variable (time series) is returned by a function that provides the value for a given environment parameter name and index (range of indices) relative to the current time. An operation, such as subtraction or calculation of absolute value, is executed after all of its operands are calculated. Figure 5 shows the bottom-up propagation of operand values during the evaluation phase executed based on the parse tree.

A rule is evaluated each time a new value of the time series referred to by the rule is obtained. Nevertheless, the lexical and syntax analysis of the rule are executed only once. Parse tree can also be

cond ::= *expr* = *expr* | *expr* != *expr* | *expr* > *expr*
| *expr* < *expr* | *expr* >= *expr* | *expr* <= *expr*
expr ::= *expr* + *factor* | *expr* – *factor* | *factor*
factor ::= *factor* * *term* | *factor* / *term* | *term*
term ::= (*expr*) | - *term* | *num* | *var* | POW (*expr* , *expr*)
| LOG (*expr*) | ABS (*expr*) | SQRT (*expr*)
| SIN (*expr*) | COS (*expr*) | TAN (*expr*)
| MIN (*series*) | MAX (*series*) | AVR (*series*)
| SUM (*series*) | STDDEV (*series*) | SLOPE (*series*)
num ::= REAL_N
var ::= VAR | VAR [*num*]
series ::= VAR [*num* .. *num*]

FIGURE 3 Context-free grammar defining the rules

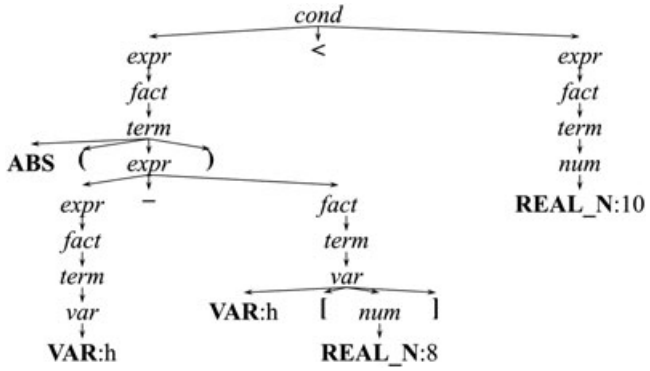

FIGURE 4 Parse tree for the rule abs $(h - h[8]) < 10$

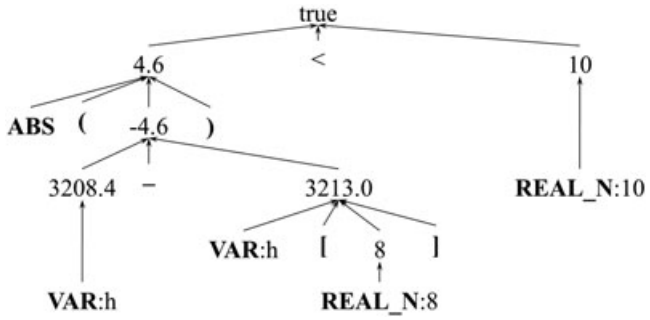

FIGURE 5 Bottom-up propagation of operand values during the evaluation of abs $(h - h[8]) < 10$

simplified if it includes operations on constant operands. For instance, the subtree representing the expression part 18/(3 * 60) can be replaced with a constant 0.1 in a parse tree representing the inequation SUM $(x[0..60])>18/(3 * 60)$. Another evaluation optimization can be achieved by an efficient implementation of the function returning time-series values: The size of the buffer used to cache the time-series values should be equal to the highest index referred to in the rules. For example, the buffer for parameter *h* should store eight values if the rule discussed in Figures 4 and 5 is concerned. The third optimization is possible by incrementally computing functions over intervals of time series. For example, SUM $(x[0..10])$ can be computed from the previous value of the sum as shown in Equation 4, which decreases the number of additions from nine in the naïve approach to two. Similarly, computing other functions over intervals of time series can be optimized.

$$\text{SUM}\quad(x[i..j]) = \text{SUM}\quad(x[i-1..j-1]) - x[i-1] + x[j]. \tag{4}$$

3.2 | Learning module

Figure 6 shows the overview of the processing involved in the learning module. The processing is performed in a continuous loop, depicted as *loop start* and *next loop* blocks in the flow chart.

In each iteration of the processing loop, new sensor data are appended to the time series that is processed in the following steps.

The first step of the processing is event detection, which is responsible for detecting unusual events in the time series. The detection of unusual events is performed by comparing recent "behaviour" with older behaviour to determine whether an unusual discrepancy has occurred. The comparison is made by taking into consideration the last *N* data points from the time series and splitting them into two parts—the more recent *head* and older *tail*, where each part consists of $N/2$ data points as shown in Figure 7 (in principle, the lengths of these two parts can be arbitrary, $N/2$ was chosen for convenience).

The *head* and *tail* parts are compared to each other by first applying the discrete wavelet transform (DWT; Jensen & Cour-Harbo, 2001) and obtaining the coefficients for both parts. DWT transforms the original signal from the time domain into the time-frequency domain, meaning that we can observe which frequencies are present in the signal at any given time, similar to the short-time Fourier transform. The maximum level of DWT decomposition depends on the length of each part $(N/2)$ and the length of the chosen wavelet (N_W) and is limited to

$$\text{floor}\left(\log_2 \frac{N}{N_W - 1}\right). \tag{5}$$

We have chosen the Haar wavelet (Haar, 1910), because it is the most simple one and has proven as a good choice during our initial experiments.

The results of the DWT are the coefficients for each of the decomposed frequencies. The coefficients are then summed overtime so that each frequency is represented with one aggregated coefficient and the head and tail parts are each represented with a vector of aggregated coefficient values. The final comparison is then made by computing the difference between the two vectors (for head and tail parts) using the Euclidean distance metric—we will refer to this value as anomaly level. Because the values for anomaly levels are domain dependant and there is no general threshold for distinguishing between normal and abnormal (event) parts of the time series, we propose a method for dynamically computing the threshold that is based on observing the past anomaly levels when no events have occurred. During the initialization phase of the algorithm, we assume that no unusual events will occur and we can adjust the threshold based on the anomaly levels during that phase. The dynamic threshold can therefore be computed as function of standard deviation, more precisely:

$$threshold(A, L) = A * std(anomaly\quad levels_L) + mean(anomaly\quad levels_L), \tag{6}$$

where $anomalylevels_L$ is a vector of the last *L* values of the computed *anomalylevels*, *std* is the standard deviation, and *A* is a scalar value by which we multiply the computed standard deviation. In each iteration of the processing loop thus computed both the anomaly level and threshold. If the anomaly level is greater than the threshold (signalling the start of an event), then an unusual event has occurred. Because we also have the information when the anomaly level drops below the threshold (signalling the end of an event), we can extract the time series that corresponds to the event and is later used in classification steps. Because the entire time-series segment of the detected event is needed in further steps, further processing needs to wait until the event has ended.

When an event is detected and the time-series segment that corresponds to the detected event is extracted, the classification of the event is performed. This is done in two steps: we first (a) perform an unsupervised event clustering and then (ii) classify the new event to the majority class of the cluster that the new event is assigned to.

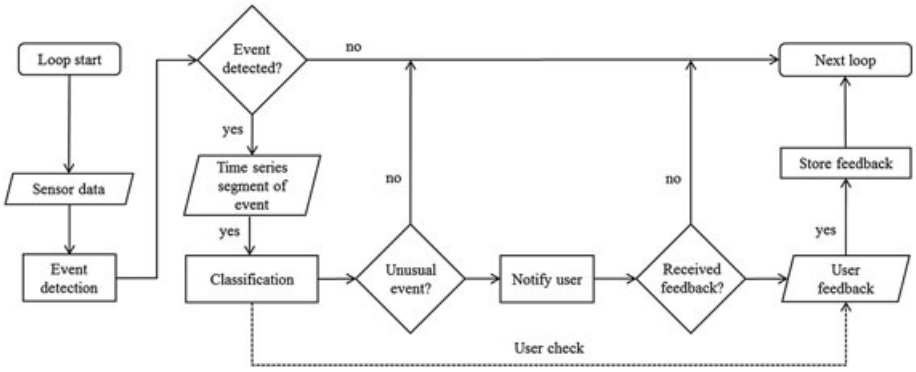

FIGURE 6 Learning module processing overview

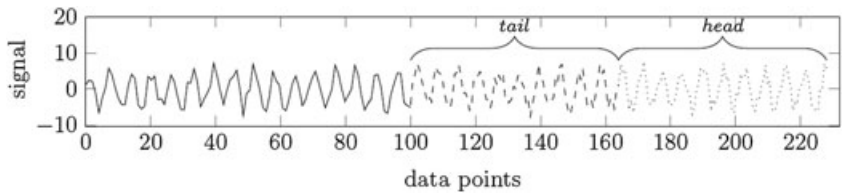

FIGURE 7 Head and tail parts of the time series

The events are classified into one of two classes: "unusual" and "normal," which reflect the user's interest in the event. The user can give this feedback information later in the processing loop.

Clustering is done using the single-linkage hierarchical clustering algorithm with dynamic time warping (Keogh & Ratanamahatana, 2005) as the method for computing distances between the events. To obtain flat clusters (partitioning of the events), the silhouette coefficient (Rousseeuw, 1987) is used to select the best number of clusters (event groups), that is, the partitioning with the highest silhouette coefficient value is selected. The results are clusters of similar events, which are later used for classification.

To classify a new event, we retrieve all other past events that were also assigned to the new event's cluster and were previously labelled by the user (by providing feedback information). If no such events exists, the new event is labelled as unusual, because a user feedback is needed. When labelled events are available, their majority class is assigned to the new event.

If the new event is classified as unusual then the user is notified with the possibility to provide a feedback whether the event is unusual or normal, which is stored internally for later use as described before. After storing the feedback, the next processing iteration starts with new data.

The learning module requires the following parameters that control its performance:

- *N* is the length the of signal used in the event detection for detecting changes in behaviour of a time series, which is an indicator of an event. The value *N* has great impact on the detection performance. Small values enable better detection of "short" events, greater susceptibility to signal noise and worsen the detection of slower changing behaviour. Large values enable the algorithm to detect more subtle, long-evolving changes in signal and are less susceptible to noise but result in worse detection of short events.
- *A* is the parameter used in computing the dynamic threshold for anomaly levels to control sensitivity of event detection. With smaller values, the algorithm will be more sensitive to anomaly level changes, but they can cause FPs. With greater values, the algorithm will similarly be less sensitive to changes.
- *L* specifies how many previously computed anomaly levels are used for computing the threshold. Smaller values allow the threshold to adapt more quickly to the changes in anomaly levels. This parameter can be tuned to obtain a desired responsiveness.
- DWT decomposition level is an optional parameter. If not specified, then the maximum decomposition level is used.

Because the learning module relies on signal processing, it inherently exhibits some delay at detecting events. The amount of delay is mainly related to the type of the event that occurred. In case of abrupt changes, such as spikes, the delay is minimal (in the range of 5–20 signal samples for *N* between 20 and 200) and is usually not dependent on the algorithm parameters. The reason is that abrupt changes quickly cause large changes in computed anomaly levels. Nonetheless, in extreme cases, when parameters are not suitably chosen for the monitored signal (for instance, a very large value of *A*), it could cause larger delays or even failure to detect an event. The detection delay is more pronounced in case of slowly evolving events, such as temperature drifts, where the point of change in signal cannot be clearly determined. In this case, the signal change is reflected more gradually in computed anomaly levels, therefore leading to longer detection delays. Figure 8 is showing a simulated example of slow drift that is starting at time 2,200 and lasts until time 2,700. The signal changes linearly from 0.5 to 1.5 mean signal value. The event is detected by three different window sizes: 200 (red box in top row), 100 (red box in middle row), and 20 (red boxes in bottom row). As can be observed, the event is detected after 30 signal samples in case of window 20, 100 samples for window 100, and 50 samples for window 200. We can also observe how window 20 is

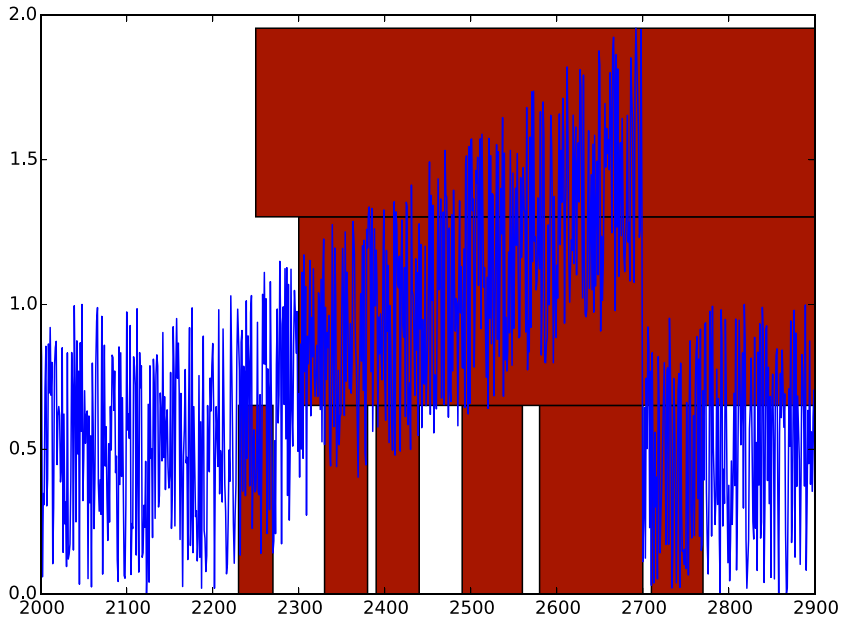

FIGURE 8 Example of slow-changing event (drift) and module's event detection delay

susceptible to signal noise, which results in large number of detected short events. On the basis of this, we could also argue that window size 20 is too small for this kind of event and that it detected the start of drift by chance, although window sizes 100 and 200 provide better detection results—one detected event with small detection delay.

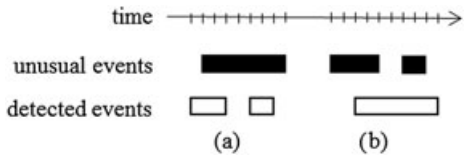

FIGURE 10 Splitting and joining unusual events

4 | EVALUATION

Only learning module was evaluated, because the detection performance of the expert rules module greatly depends on the rules that are defined by the human operator and must be defined separately for each time series (device) in the evaluation dataset, which is out of scope of this paper.

4.1 | Methodology

It is not always clear when an unusual event is detected correctly, as illustrated in Figures 9 and 10 showing various cases of detection.

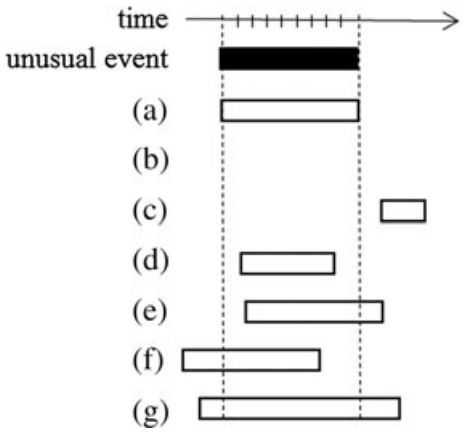

FIGURE 9 Overlapping of unusual and detected unusual events

Unusual event (black rectangle in Figure 9) is a sequence of time points with unusual values or trend, that is, a domain expert would label them as unusual. A detected event (white rectangles in Figure 9) is a sequence of time points labelled as unusual by an algorithm. The algorithm's performance can be evaluated on level of events or on the level of time points. On the level of events, a true positive (TP) event is declared when the algorithm labels at least one point belonging to an unusual event as unusual (Figure 9a—all points identified correctly—,d–g). On the other hand, there are two possible mistakes of the algorithm at the event level. The first is false negative event (FN, Figure 9b), which happens if the algorithm labels all points belonging to an unusual event as usual. In this case, the user is not notified about the unusual event—this represents the most severe mistake. The other type of mistake is a false positive event (FP, Figure 9c), detected as unusual although not unusual in reality. In such case, the user is notified about the detected event although no unusual event actually took place. This type of mistake can be tolerated to a certain extent, so the algorithm should be allowed to trigger a limited number of FP events if that simultaneously decreases the number of FN events. Nevertheless, the algorithm needs to minimize the number of both types of mistakes. Furthermore, the algorithm might make a mistake of splitting a single unusual event into two or more detected events (Figure 10a) or joining two or more unusual events into a single detected event (Figure 10b). In both cases, the user is correctly notified about the unusual event and no FP events are detected; however, this is still a problem because the algorithm needs to group similar detected events in order to learn

TABLE 1 Summary of possible mistakes of unusual event detection (see Figures 9 and 10)

	During detected event						
	During unusual event		remaining				
Fig.	FN	TP	FP	TN	start	end	unevent detected
9a		✓		✓	ok	ok	✓
9b	✓			✓			
9c	✓		✓	✓			
9d	✓	✓		✓	late	early	✓
9e	✓	✓	✓	✓	late	late	✓
9f	✓	✓	✓	✓	early	early	✓
9g		✓	✓	✓	early	late	✓
10a	✓	✓	?	✓			too many
10b		✓	✓	✓			too few

which groups are not interesting for the user based on his feedback. The grouping and learning about interestingness of groups become difficult in case of joining or splitting unusual events. Finally, a true negative (TN) is declared when both the algorithm and the domain expert label the overlapping sequence as normal,or in other words, do not detect an unusual event.

Additionally, mistakes originate from imprecise detection of the beginning or end of the event, with detection starting or ending too soon or too late in comparison with the actual event. The summary of these mistakes, referencing Figures 9 and 10, is presented in Table 1. However, it should be noted that the definition of the start and end points of the actual event may be arbitrary to some degree.

On the basis of the number of TP, FP, TN, and FN events, we compute several standard measures that evaluate different aspects of event detection performance, more precisely:

- Sensitivity ($\frac{TP}{TP+FN}$): the ratio between the number of detected events and the number of actual events. Greater values mean that more events were detected by the system.
- Precision ($\frac{TP}{TP+FP}$): the ratio of the actual events in a set of all events that were detected by the system. Greater values mean that there were less falsely detected events.
- False discovery rate ($\frac{FP}{TP+FP}$): inverse of the precision measure. Greater values mean that there were more falsely detected events.
- F_1 score ($\frac{2TP}{2TP+FP+FN}$): the harmonic mean of precision and sensitivity. Greater value indicate better overall performance of the system.
- Classification accuracy ($\frac{TP+TN}{TP+FP+TN+FN}$): the percentage of correctly detected events.

In addition to these measures, we also provide a visualization of algorithm performance in the receiver operating characteristic (ROC) space. It is necessary to note that our problem is not a standard binary classification problem, and our algorithm does not provide probability values (or ranks) for instances. Therefore, the ROC curve and consequently the area under the ROC curve (AUC) cannot be computed. However, we can approximate the ROC curve and AUC by running the algorithm using different parameters resulting in different TP rate (recall) recall and FP rate ($\frac{FP}{FP+TN}$) values, and then only plot nondominant ones in the ROC space.

Because the learning module can be tuned (via parameters) to detect events of various lengths, we used three independent parameter setups in parallel to detect short-, mid-, and long-term events. This is expected to give better results in cases when little is known in advance about the time-series data. The actual values of parameters are specified in subection 4.2.

The classifying aspect of the algorithm was evaluated by analysing the clustering performance—which is actually the basis for accurate classifications. The performance of the clustering was evaluated by the Adjusted Rand Index (ARI; Lawrence & Phipps, (1985)) that measures the similarity of two different cluster assignments of examples, ignoring permutations and with chance normalization. The measure is bounded by the range [−1, 1], where higher values reflect greater similarity (assignment consensus). True cluster assignments are known, because the simulated time series were used for evaluation; therefore, we can interpret the ARI as a measure of the clustering performance.

4.2 | Datasets

The performance of the learning module as compared to a human operator was tested on a two datasets.

The first dataset consists of 137 real-sensor time series from different refrigeration systems. Time series were labelled by three human operators, who independently from one another analysed the data and listed all events that they considered "unusual." If an event was labelled by at least two users, it was considered unusual for the purpose of the analysis (P). Over 500 unusual events were collected, including those shown in Figure 1. As mentioned in subsection 4.1, different algorithm parameter setups were used to accommodate the detection of events of various lengths. Generally, we can group the parameter setups into three groups, namely *short*, *mid*, and *long*. Each of these groups consists of 30 setups with parameter values that are drawn from uniform distributions as specified in Table 2. See subsection 3.2 for parameter descriptions.

TABLE 2 Uniform distributions for generating parameter setups for the real sensor data

Setup	N	A	L
W_{short}	[6, 10]	[1, 3]	[35, 90]
W_{mid}	[24, 40]	[1, 3]	[120, 270]
W_{long}	[48, 80]	[1, 3]	[240, 540]

TABLE 3 Uniform distributions for generating parameter setups for the simulated sensor data

Setup	N	A	L
W20	[15, 25]	[1, 3]	[75, 225]
W100	[75, 125]	[1, 3]	[375, 1125]
W200	[150, 250]	[1, 3]	[750, 2250]

Second dataset was artificially generated to overcome the limitations of real sensor data. The problem with sensor time series is that they are typically of limited length (several hours of data collection at most) and often contain only a small number of unusual events. In addition, each dataset usually contains just one or two types of unusual events; therefore, we are not able to fully evaluate the ability of the learning module to categorize the events into different types. Simulated time series were generated using a triangle wave function with a period of 40 points, combined by random noise both in amplitude and period. Additionally, different unusual events with random parameters were introduced, such as spikes, temperature drifts, oscillations, temperature ramps, and elongations of the period to simulate the real-world events. Because the period of the main function and the sampling rate are known in these simulated series, we set the windows close to 20 (half of the period), 100 (2.5 periods), and 200 (5 periods) points to cover both shorter and longer events. As for real sensor data, we generate 30 setups for each of the event lengths as specified in Table 3.

When reporting on combined performance results, we randomly choose (without replacement) one setup from each of the three groups (*short*, *mid*, and *long*) and generate a combined performance evaluation for the event detection. We then repeat this 30 times (the number of individual setups in each of the groups) and compute the mean value for each of the performance measures.

4.3 | Results

We first report on results from event detection evaluation on both datasets and then present results for classification aspect of the learning module only on simulated dataset, due to the limitations of real sensor dataset described in subsection 4.2.

Evaluation results for real sensor dataset are presented in Table 4, where the values represent the average of 30 runs, except in the column *Expert*, where each expert event detection result is tested against the other two experts' event detection results. The *Expert* column gives an insight into event detection consistency between the experts or, in other words, the level of agreement between the experts. The comparison of the column *Expert* with the column *Combined* gives us an insight on how well our algorithm compares to domain experts.

The sensitivity of the combined setups is very close to the domain experts' sensitivity, meaning that our algorithm fails to detect only slightly more events when compared to experts. Given that the data was (intentionally) analysed using fixed preselected windows, although it consisted of several different types of sensors with different sampling rates and different periodicity of cooling cycles, we can expect that optimizing the module parameters for a specific sensor will increase sensitivity considerably. On the other hand, the combined setups achieve considerably lower precision and consequently higher false discovery rate, meaning that our algorithm produces more FPs. This problem, however, is addressed in the classification step, where the module learns about the events from the user feedback and the number of false alerts usually decreases quickly in real life. The algorithm's lower precision can also be observed from the F_1 score and classification accuracy measures, which are both lower than the expert's. An example from the real sensor dataset analysis is shown in Figure 11.

Evaluation results for simulated dataset are presented in Table 5. We can see that the module's combined sensitivity is almost perfect and actually misses only one event in all of the 30 runs. Such a high performance can be attributed to the optimized window lengths, which is important for real-life applications. We can again see that combining three windows of different lengths leads to higher sensitivity at the expense of precision, which in our case is preferred, since FNs (missing important events) have higher real life cost than FPs.

As expected, we can see that for both the real and the simulated data, the shorter windows achieve higher sensitivity than longer windows. However, the combined setups always achieve the highest sensitivity. Inverse can be observed for precision. Although high precision is achieved by individual windows, when combined, the precision drops due to more FPs. Similar trends can be observed for false discovery rate, F_1 score, and classification accuracy.

ROC curve approximation for real (circles) and simulated (squares) data can be seen in Figure 12. As can be observed, both ROC curves

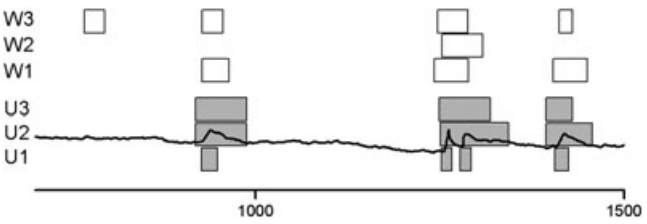

FIGURE 11 Real sensor data analysis. Black line represents the sensor data. Grey rectangles indicate the unusual events as labelled by the users (U1–3). White rectangles indicate the events, detected by the learning module with three windows (W1, W2, and W3 correspond to *short*, *mid*, and *long* window sizes)

TABLE 4 Sensitivity, precision, false discovery rate, and F_1 score for the performance of the learning module on real data

Window	W_{short}	W_{mid}	W_{long}	Combined	Expert
Sensitivity	0.77	0.45	0.30	0.84	0.88
Precision	0.92	0.89	0.88	0.62	0.89
False discovery rate	0.08	0.11	0.12	0.38	0.11
F_1 score	0.84	0.60	0.45	0.72	0.89
Classification accuracy	0.86	0.68	0.55	0.78	0.88

TABLE 5 Performance of the learning module for three different windows and overall performance on the simulated data

Window	W20	W100	W200	Combined
Sensitivity	0.9618	0.8392	0.7775	0.9996
Precision	0.9958	0.9956	0.9963	0.6207
False discovery rate	0.0042	0.0044	0.0037	0.3793
F_1 score	0.9785	0.9107	0.8734	0.7659
Classification accuracy	0.9790	0.9181	0.8826	0.8058

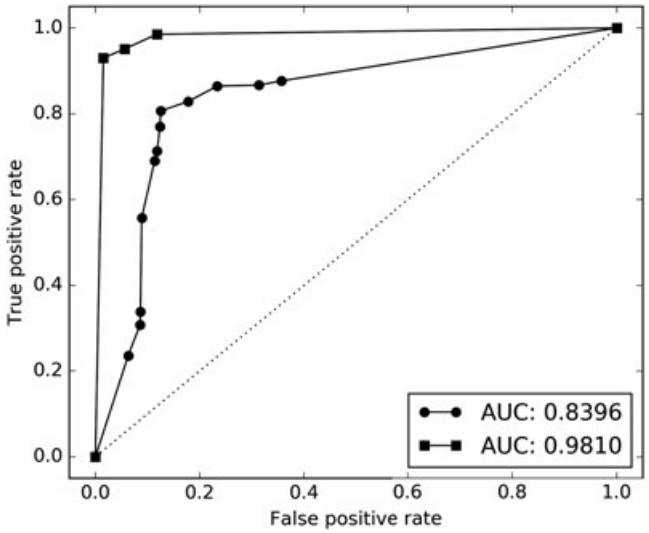

FIGURE 12 Receiver operating characteristic curve approximation for real data (circles) and simulated data (squares). AUC = area under the ROC curve

closely approach the upper left corner, meaning that our algorithm generally achieves a high TP rate and a low FP rate. As a result, both AUC values are also high, 0.9810 for the simulated data and 0.8396 for the real data. The ROC curve approximations are based on more parameter setups (130) with greater variability for *A* and *L* parameters than reported in Tables 4 and 5 to obtain better ROC curve approximation, because setups from both tables mostly fall near the upper left corner of ROC space. More specifically, *A* was picked from [0.5, 7.5] and *L* was computed by multiplying the window length *N* with a random number sampled from [1, 14].

The classifying aspect of the algorithm is shown in Figure 13, using the simulated data with three types of events in this particular example—spikes, temperature drifts, and temperature ramps. The computed ARI for W100 was 0.66, which would suggest a fairly weak performance. However, manual inspection of cluster assignments revealed that the events are not assigned into false clusters, e.g. spikes are not assigned into clusters with ramps, but rather that event types are further subdivided into more specialized clusters, e.g. the algorithm differentiates between high and low ramps. If we consider this and group the subclusters into a larger cluster corresponding to a particular event type (which can be obtained through user feedback in real-life), the obtained ARI is 1.00, which reflects a perfect similarity.

5 | DISCUSSION

In this paper, we described the iLab system, whose task is to monitor the data about a laboratory environment, particularly refrigeration devices, and inform the user about unusual events. It contains two independent complementary modules: one using user-defined rules and one using ML. Having two complementary modules is iLab's first advantage over similar systems and is already patented (Kužnar et al. 2012). The rules' module detects events that can be anticipated and described, such as excursions of the refrigeration temperature outside the prescribed boundaries. Its alarms are accompanied with a description text with the values exceeded, as well as possible instructions on how to react. The ML module detects any behaviour that is unusual compared to previous normal behaviour, be it known or new. Although the events vary in cause, severity, and consequences, ML module detects changes in behaviour, be it short or long term, and alarms users. Although many systems for the management of laboratory equipment or specifically refrigeration devices support rules to detect undesirable events, these rules are often not as flexible as iLab's. However, iLab's main advantage over competing systems is its ML module. Not only can it detect the majority of unusual events, it can automatically classify the types of events it detects, thus allowing the user to define some of them as normal. This way, it can overcome its weakness of raising false alarms, which inevitably occur because even humans do not always agree on what is unusual and what normal.

The iLab system was tested on both real and simulated data in order to evaluate the performance of the learning module in comparison with a human operator manually checking the data. The test on real data showed 84 % sensitivity, which is a rather high value, comparable to the domain experts' performance. We expect that most of the irregular events will already be recognized by the expert rules' module. In this case, three arbitrarily chosen windows were combined to detect events because the data originated from different types of devices with different typical behaviour and sampling times. In tests on simulated data, windows were optimized according to the period of the normal operating cycle. A window that is shorter than the period will detect short events, whereas a window that spans over several periods is better to detect temperature drifts. In this case, the combined sensitivity was 99.96%, which means that the system recognizes almost all irregular events. Precision can be improved through the user feedback that labels some false alarms as irrelevant. Despite high sensitivity, the precision was 62%, which is acceptable, especially considering the ability of the algorithm to learn from the user feedback and decrease the

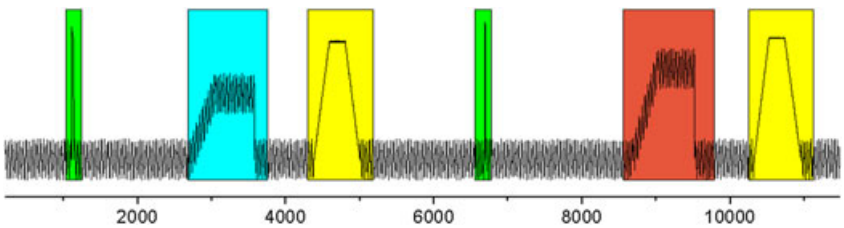

FIGURE 13 A section of the classification accuracy test on the simulated data. Coloured rectangles indicate the recognized events whereas the color refers to the event type. As seen in this example, spikes and temperature ramps are assigned to their respective groups whereas the temperature drifts are classified as two different types of events. The reason is that the second temperature drift has a considerable higher amplitude than the first one

number of false alarms. The ROC curve estimation enabled us to compute the AUC, which was 0.84 for real data and 0.98 for simulated data.

In a separate classification test, the system was always able to correctly classify three different types of unusual events. It initially divided the unusual events into more than three clusters, but each of the clusters contained only events of one type, so merging some of the clusters yielded perfectly accurate classification. If the events of a particular type are truly unusual, having them in multiple clusters is not a problem. If, however, the user wants to define one of the types as normal, then the user needs to do so for each of the clusters, which is an inconvenience, but a fairly minor one.

For future work, a detailed analysis of event detection delay is needed to determine how quickly does the system detect different types of events. In our preliminary manual survey, we have observed that the type or shape of event can have a big impact on the delay. Events that introduce gradual changes to the refrigeration device operation (e.g., temperature drift as a result of slightly open door) exhibit longer detection delays than events that have instant impact on operation (e.g., door left entirely open). There is also a need to study the effects of algorithm parameter choice on the detection delay and detection performance, which would give answers if and how we can tune detection delay and performance through algorithm parameters and if there are some trade-offs between delay and performance. This would be helpful in situations where one criterion is more important than the other and would allow tuning the parameters to specific needs.

ACKNOWLEDGEMENTS

Authors thank the Slovenian Ministry of Higher Education, Science and Technology for financial support. We also thank the anonymous reviewers for providing valuable comments that enabled important improvements.

REFERENCES

Aho, AV, Sethi, R, & Ullman, JD (1986). *Compilers: Principles, techniques, and tools*. Boston: Addison-Wesley.

Alimentarius, C (2009). *Food hygiene, basic texts* (4th ed.). Rome: World Health Organization, Food and Agriculture Organization of the United Nations.

Alvarez, MD, & Canet, W (1998). Effect of temperature fluctuations during frozen storage on the quality of potato tissue (cv. Monalisa). *Zeitschrift für Lebensmitteluntersuchung und - Forschung A, 206*, 52–57.

CDC (2012). Guidelines for storage and temperature monitoring of refrigerated vaccines. Retrieved from http://www.cdc.gov/vaccines/recs/storage/toolkit/toolkit-resources.pdf. [Accessed 11 May 2014]

Chiu, NH, & Schneider, DA (1986). Over-temperature warning system for refrigerator appliance: United States Patent.

CUP (2014). LALR parser generator in Java. Retrieved from http://www2.cs.tum.edu/projects/cup/. [Accessed 7 April 2013]

Haar, A (1910). Zur Theorie der orthogonalen funktionensysteme. *Mathematische Annalen, 69*, 331–371.

Jensen, A, & Cour-Harbo, AI (2001). *Ripples in mathematics: The discrete wavelet transform*. Berlin: Springer.

JFlex (2014). The fast scanner generator for Java. Retrieved from http://jflex.de/. [Accessed 5 December 2014]

Kang, Y, Belušić, D, & Smith-Miles, K (2014). Detecting and classifying events in noisy time series. *Journal of the Atmospheric Sciences, 71*, 1090–1104.

Keogh, E, & Ratanamahatana, CA (2005). Exact indexing of dynamic time warping. *Knowledge and Information Systems, 7*, 358–386.

Kodak (2005). Storage and Care of KODAK Photographic Materials. Retrieved from http://www.kodak.com/global/en/consumer/products/techInfo/e30/e30.pdf. [Accessed 4 June 2014]

Kužnar, D, Gams, M, Marinčič, D, Lotrič, M, & Čufar, K (2012). Method for intelligent control of refrigeration systems: Slovenian patent P-201200245: Slovenian Intellectual Property Office.

LabWare. Retrieved from https://www.labware.com/en/p/Products/LIMS/LIMS-V6-Technical-Guide. [Accessed April 2014]

Lawrence, H, & Phipps, A (1985). Comparing partitions. *Journal of Classification, 2*, 193–218.

Preston, D, Protopapas, P, & Brodley, C (2009). Event discovery in time series. In *SIAM International Conference on Data Mining*. Sparks, Nevada, USA.

Ramey, PM, Rozsnaki, JJ, & Osman, A (2008). Refrigeration System Fault Detection and Diagnosis using Distributed Microsystems: United States Patent.

Rousseeuw, PJ (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics, 20*, 53–65.

Sharifzadeh, M, Azmoodeh, F, & Shahabi, C (2005). Change detection in time series data using wavelet footprints, *Advances in spatial and temporal databases*, Vol. *3633*. Berlin Heidelberg: Springer, pp. 127–144.

Sharood, JN, & Carr, MD (2002). Refrigeration Monitor Unit: United States Patent.

Damjan Kužnar is a researcher at Jožef Stefan Institute in Slovenia with BSc degree in Computer Science and Informatics at the University of Ljubljana, Slovenia, in the field of machine learning and artificial intelligence. His research interests include machine learning, data mining, intelligent systems, natural language processing, and optimization.

Dr. Rok Piltaver is a researcher at Jožef Stefan Institute in Slovenia. His research interests include data mining, intelligent systems, and applications in ambient intelligence. He received the best Slovenian innovation award for detecting unusual movement of personnel and equipment in high security buildings and was part of the team that won activity recognition competition and the best paper award for person identification based on door acceleration. He also designed and developed other systems ranging from a robot that learns from humans to smart house that automatically adapts to residents' needs.

Anton Gradišek received his PhD in physics at the Faculty of Mathematics and Physics, University in Ljubljana. Currently, he is a researcher at Jozef Stefan Institute where he works on interdisciplinary projects, bringing together physics, artificial intelligence, and medicine.

Matjaž Gams is the head of Department of Intelligent Systems at the Jozef Stefan Institute and professor of Computer Science at the University of Ljubljana and MPS, Slovenia. He is or was teaching at 10 faculties in Slovenia and Germany. His professional interest includes intelligent systems, artificial intelligence, cognitive science, intelligent agents, business intelligence, and information society. He is a member of numer-

ous international program committees of scientific meetings, national and European strategic boards and institutions, and editorial boards of 11 journals and is the managing director of the Informatica journal, published for 39 years.

Dr. Mitja Luštrek is a researcher at Jožef Stefan Institute in Slovenia. He is the head of the Ambient Intelligence Group at the Department of Intelligent Systems. His main research interests are ambient intelligence and ubiquitous computing, with a focus on human behaviour analysis and health applications. He has also worked in several other areas of artificial intelligence, including game playing, heuristic search, and bioinformatics. He is the chair of the Slovenian Artificial Intelligence Society and an editor of the Informatica journal.

How to cite this article: Kužnar D, Piltaver R, Gradišek A, Gams M, Luštrek M. An intelligent system to monitor refrigeration devices. *Expert Systems*. 2017;34:e12199. https://doi.org/10.1111/exsy.12199