Automated Anomaly Detection in CPS Log Files
A Time Series Clustering Approach

Tabea Schmidt(✉), Florian Hauer, and Alexander Pretschner

Department of Informatics, Technical University of Munich, Munich, Germany
{tabea.schmidt,florian.hauer,alexander.pretschner}@tum.de

Abstract. When *Cyber-Physical Systems* (CPS) work incorrectly we would like to know the reason for this behavior. Experts inspect log files of CPS to get an idea about what went wrong. The large amount of information, which is stored in those log files, and the complexity of CPS pose a challenge to experts that try to manually detect anomalies in the system's behavior. We propose to automate anomaly detection in CPS log files by applying a clustering approach to find time spans, in which the regarded system behaves abnormal. With our approach, we aim to significantly reduce the time and effort that is needed by experts to discover anomalies in the log files without having to build a model of the system first. The results from our evaluation show that our generic approach can effectively find anomalies for different types of CPS.

Keywords: Anomaly detection · CPS · Time series data · Clustering · Unsupervised machine learning · Log analysis

1 Introduction

In *Cyber-Physical Systems* (CPS), the physical components of the systems are connected, controlled and monitored by software components. The sensors of the systems are permanently collecting data. On the one hand, this data is stored in log files and, on the other hand, it is processed by software components. We can find CPS in a broad range of application domains. Examples from different fields are unmanned aerial vehicles (UAVs), autonomous cars or smart grids [12].

In the following, we will take a look at a single UAV as an example for CPS. The UAV has the mission to fly to a specified waypoint to deliver a package. When the UAV reaches the waypoint, the package is damaged. As the company that wanted to deliver an intact package, we are interested in the reason for the damage of the package. A UAV normally logs at each time stamp more than 100 parameters including sensor data, messages and status of components. A flight of ten minutes yields a log file with 6,000 time stamps if the data is logged every 100 ms. If each of the 100 parameters is logged at each time stamp, the resulting log file contains 600,000 parameter values. An expert that tries to find the reason for the damage of the package will take a look at the behavior of

A. Casimiro et al. (Eds.): SAFECOMP 2020, LNCS 12234, pp. 179–194, 2020.
https://doi.org/10.1007/978-3-030-54549-9_12

the UAV during its flight by inspecting the log file. Throughout this process, the expert will look for anomalies in the behavior of the UAV. These anomalies in the system's behavior depict undesired behavior of the system, which can result, for example, from failures of the system or attacks on the system. When searching the log file for an anomaly, the expert might have to take a look at all 600,000 parameter values in the worst case. In the best case for anomaly detection, the UAV crashes during its flight, hinting that the underlying fault is probably located at the end of the log file. However, this is not necessarily the case. A short anomaly might occur that needs some time to manifest and to result in a crash. Then, taking a look at only the last parameter values will not reveal the anomaly and the expert still needs to inspect up to 600,000 parameter values. Even if the anomaly is located at the end of the log file, there are additional challenges that the expert has to handle. These are described in the following: The system's behavior might look normal when the expert only regards a single parameter each time. This means that the expert additionally needs to take the interaction of different parameters into account to detect abnormal behavior of the system. Also, the particular set of interactions that the expert needs to review to detect an anomaly is not known beforehand.

Therefore, we propose to automate the anomaly detection in CPS log files by using a clustering approach. With our approach, we reduce the log files of a CPS to the time spans, in which the system behaves abnormal. In this way, the expert does not need to review the complete log files. This saves him or her a large amount of time and effort when trying to discover an anomaly.

Previous approaches of anomaly detection in CPS log files [5,16] work with a model that describes the normal behavior of the system. This means that a complete and fine-tuned model of the system is needed to effectively detect anomalies in the log files of this system. If we want to apply these approaches to detect anomalies for different systems, we need to create a model for each of those systems. Our proposed approach for detecting anomalies is not dependent on a model of the system. When inspecting CPS log files, we aim to find time spans, in which the system does not behave as in the other parts of the log files. In this way, we create a generic approach, which can be applied to log files of any type of CPS without having to train or learn a model of the system beforehand.

The **contribution of this paper** is the following: We provide a generic approach to detect time spans, in which a CPS behaves abnormal. For this purpose, we apply clustering on the multidimensional time series data from the log files of the system. We evaluate our proposed anomaly detection approach with two CPS data sets, which include failures and attacks as abnormal behavior. One of these data sets includes log files from a UAV, whereas the other one contains data from an industrial water plant.

An overview of our proposed process for detecting anomalies in CPS log files is presented in Sect. 2. We then describe our clustering approach in detail in Sect. 3. Subsequently, Sect. 4 demonstrates the results of our experiments on two data sets. In Sect. 5 we discuss related work in the area of automated anomaly detection for CPS. Finally, we conclude our work in Sect. 6.

2 Anomaly Detection

When a CPS works incorrectly, we would like to find the reason for this behavior. Abnormal behavior can, for example, be caused by a failure within the system or by an attack launched on the system. When investigating the reason for the incorrect behavior, experts take a look at the log files of the system. In those log files, the parameter values of the system such as sensor data, messages or status of components are logged over the execution time of the system.

As mentioned earlier, there exist two main challenges for manually detecting anomalies in CPS log files: (i) the large amount of information included in the log files and (ii) the general complexity of CPS. Due to these challenges, we propose an automated approach to reduce the amount of information that needs to be inspected by the expert and enable easier detection of anomalies in CPS log files. Figure 1 depicts an overview of our proposed approach.

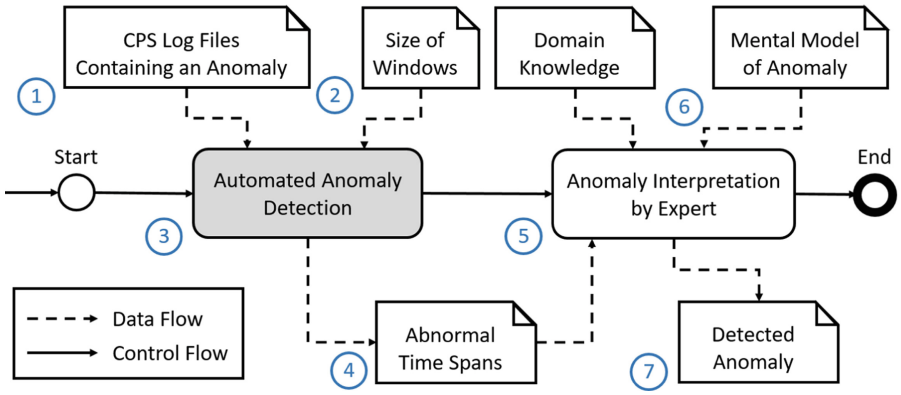

Fig. 1. Automated anomaly detection in CPS log files.

CPS log files (1) are the input to our approach. These log files include the logged parameter values over the execution time of the CPS. Additionally, they contain the anomaly that we are interested in discovering.

In this paper, we focus on the automated anomaly detection approach (3), which takes the CPS log files as input. We divide the log files into a number of time spans and compare the behavior of the parameters in these time spans. Therefore, we need as a second input the size of these time spans (2), which we call "windows". We will discuss different options for estimating this variable in this paper. Additionally, we would like to inspect methods for estimating a good window size in future work. With our clustering approach, we aim to detect time spans in the log files, in which the regarded system behaves abnormal (4).

In the next step, an expert can use the discovered abnormal time spans of the log files to interpret the abnormal behavior of the system (5). To detect anomalies in the provided time spans of the log files, additional expert knowledge (6) is

necessary. To find an anomaly in the provided time spans, the expert will need to have domain knowledge and a mental model of the anomaly to discover. This means that the expert has to know how the parameter values of the regarded system normally behave to detect anomalies.

As a result of this process, the expert might be able to detect the anomaly (7) in a reduced amount of time and effort. Our clustering approach provides time spans, in which the system does not behave normally. Thus, it is possible that also rare behaviors are represented in these time spans. The expert will not be able to detect an anomaly if only rare behavior is included in the provided time spans. If this is the case, we propose to change the window size and run the automated anomaly detection again.

3 Detecting Anomalies with Time Series Clustering

In this chapter, we explain the technical details of our method. In Fig. 2, we provide an overview of the methodology of our proposed approach for automated anomaly detection in CPS log files. With our approach, we try to find those time spans in which the system behaves abnormal. We expect these abnormal behaviors to be different from the normal behavior of the system. Additionally, we anticipate the abnormal behavior to only appear rarely. The idea is, therefore, to cluster the data and then look at the small clusters to detect anomalies in the system's behavior.

CPS have a list of parameters $P = p_1, p_2, ..., p_n, ..., p_N$ that are logged over time. A CPS log file $L = p_1(ts_0), p_2(ts_0), ..., p_1(ts_1), ..., p_N(ts_T)$ consists of parameter values $p_n(ts_t)$ for each recorded time stamp ts_t with $t \in [0, T]$. The parameter value $p_n(ts_t)$ is logged at time stamp t and represents the parameter with index n in the list of parameters P. When trying to find anomalies in the log file of CPS, we might encounter the problem of log-heterogeneity [4]. Different components of CPS might generate differently formatted log files. In this case, we need to pre-process these log files to bring them into one format L. [4] presents various techniques that can be used for this pre-processing step. We will divide the log files into smaller windows w_j for clustering with $j \in [1, J]$. Each window has the same size $windowSize$, which has to be specified beforehand by the expert. More details about this step can be found in the next subsection. The result of our proposed approach will be a list of windows, in which the system behaves abnormal. We use the variable $AW = aw_1, ..., aw_F$ to represent this list of abnormal windows.

3.1 Window Slicing

When we use a clustering approach to detect anomalies in CPS log files, we need to first divide the log files into subsequent time spans. Then, we can compare the behavior of the parameters P in these time spans. We cut the time series data of the log files $L_1, ..., L_L$ into consecutive windows w_j. Each window has a size of $windowSize$ and the first window of each log file starts at ts_0. The

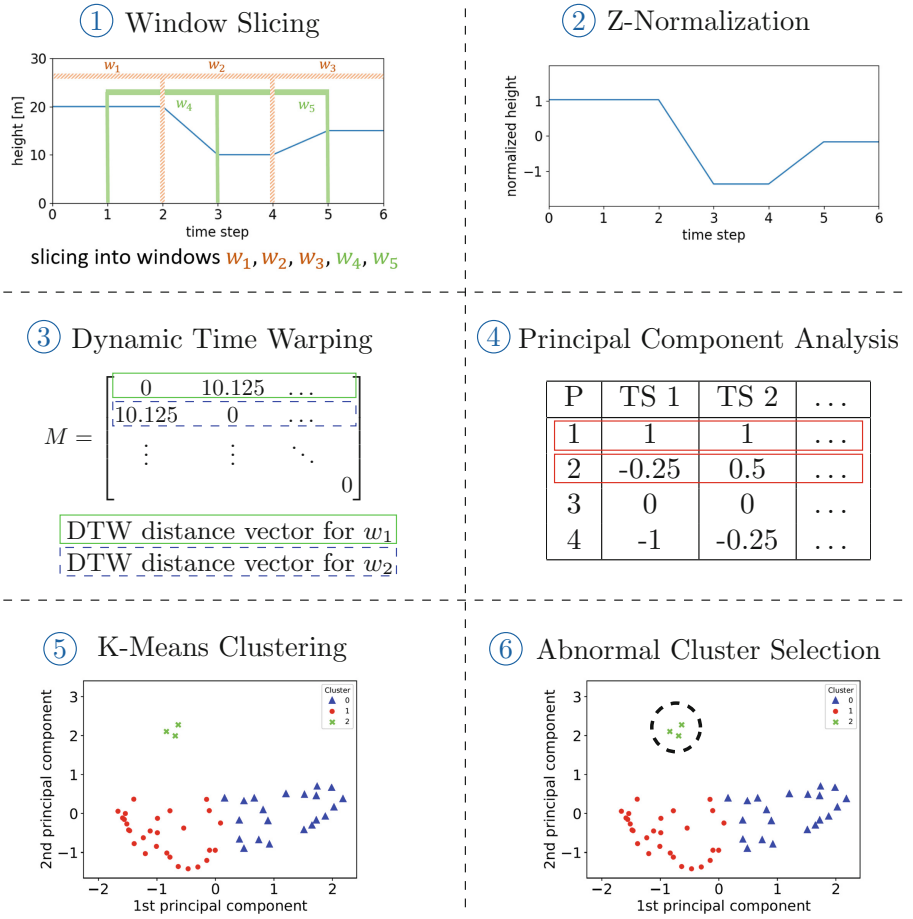

Fig. 2. Methodological overview of our proposed approach.

variable *windowSize* has to be specified by the expert beforehand. One way of estimating this variable is to take the knowledge about the system's behavior and its previously occurred anomalies into account. After detecting several anomalies in the system, we expect the expert to be able to estimate a good *windowSize* for each of these anomalies. Another approach is to execute the automated anomaly detection several times with different window sizes to find a good estimate. In our experiments, we discovered that a clustering with a low inertia value leads to good results for detecting anomalies, but not necessarily to the best results. Receiving good results for low inertia values seems intuitive since the inertia value represents the internal cohesion of the clusters. A low inertia value stands for a high internal cohesion within the clusters. An optimization technique can be used to find a window size, with which the lowest inertia values can be achieved. We believe that optimizing for a low inertia value will not be sufficient to find

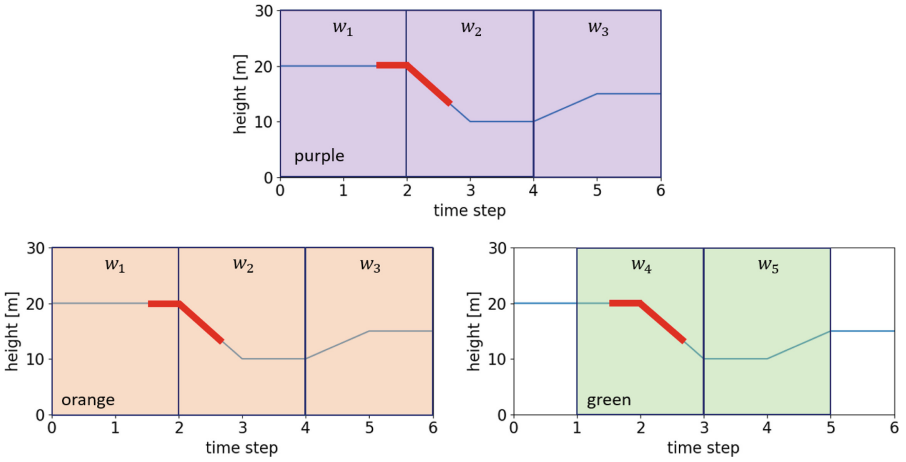

Fig. 3. If the log file L is only cut into the three purple windows w_1, w_2 and w_3 the abnormal behavior marked in bold might be missed. Therefore, we propose to instead use the five orange and green windows w_1, w_2, w_3, w_4 and w_5 to represent the log file. (Color figure online)

the best results for detecting anomalies though. In the future, we would like to assess several methods and optimization criteria for estimating a good value for the size of the windows. In this work, we will assume that the expert knows a good estimate of $windowSize$.

When slicing our data into windows w_j, we might run into the problem that the anomaly is lying directly on the border of two windows. Figure 3 illustrates this problem. In this figure, the abnormal behavior is located at the border of the windows w_1 and w_2. If the abnormal behavior wraps around the border of two windows w_j and w_{j+1}, it might not be detected since its effect on the complete behavior of w_j or w_{j+1} might be too small. Thus, we decided to consider additional windows that start with an offset at time stamp $ts_0+(windowSize/2)$. These windows have the same size as the previously regarded windows. In this way, we can find the complete abnormal behavior in at least one window as long as we choose $windowSize$ large enough for the behavior to fit in.

3.2 Z-Normalization

Since we want to detect anomalies in the behavior of a system, we are not interested in the absolute values of the parameters p_n. Instead, we would like to compare how the parameter values change throughout the log files. Essential for us is, for example, that the UAV reduces its height by 5 m in 1 s. Whether the UAV decreased its height starting from 20 m or 15 m is instead not relevant for the anomaly detection. To concentrate on the structure of the time series we apply *Z-Normalization* on the data from the log files as proposed in [6]. In this way, we equalize the amplitudes and spatial differences between the time

series. This enables the clustering algorithm to focus on the structure of the time series. To compute the normalized data points we need to look at each parameter p_n separately. For each parameter p_n we have to compute the mean and standard deviation over all time steps t_n in the log files. Then, we can calculate the normalized data point $p'_n(ts_t)$ for each data point $p_n(ts_t)$ by using the computed mean and standard deviation values.

3.3 Dynamic Time Warping

To compute the distance and therefore the similarity between two or more time series, *Dynamic Time Warping* (DTW) is commonly used [3,13]. With this method, we can find equal time series even if they are distorted along the temporal axis. Since we want to find equal behaviors in the log files that might be slightly distorted, this method seems to be suited very well for our approach.

To compute the DTW distance between two windows $w_1 = w_{1_1}, ..., w_{1_a}$ and $w_2 = w_{2_1}, ..., w_{2_b}$, we need to calculate the cumulative distances $D(w_{1_a}, w_{2_b})$ between the points of the time series.

$$D(w_{1_a}, w_{2_b}) = \delta(w_{1_a}, w_{2_b}) + min \begin{Bmatrix} D(w_{1_{a-1}}, w_{2_{b-1}}) \\ D(w_{1_a}, w_{2_{b-1}}) \\ D(w_{1_{a-1}}, w_{2_b}) \end{Bmatrix}$$

In this function, δ represents the Euclidean distance between the single data points w_{1_a} and w_{2_b} [3,19].

In the third step of our methodology, we compute the DTW distance of all parameter values p_n of one window to the parameter values p_n of all other windows. Our goal is to gain a single DTW distance value $D(w_i, w_j)$ between two windows w_i and w_j. To achieve this, we first need to calculate the DTW distance of these two windows for each parameter: $D_{p_n}(w_i, w_j)$ for $p_n \in P$. In a next step, we have to aggregate these single distance values of all parameters $D_{p_n}(w_i, w_j)$ to obtain a single DTW distance $D(w_i, w_j)$ for the two windows w_i and w_j. We use the sum of squares to aggregate these values. If two windows depict an identical behavior, the resulting DTW distance between these windows $D(w_i, w_j)$ will be 0. The matrix representing the DTW distances between all windows will be the input for the clustering algorithm in a subsequent step. Using DTW to calculate the distances between several time series enables us to work with data from different types of CPS. Since DTW can work on time series of different length and independent of the underlying parameters, we create a generic approach that is not tailored to one specific system.

3.4 Principal Component Analysis

Before clustering the derived DTW distances, the dimensionality of the data first needs to be reduced to gain decent clustering results. A *Principal Component Analysis* (PCA) [1] can be used for this purpose. It generates principle

components by combining redundant or similarly behaving features of the data. This results in a decreased dimensionality of the data. We apply the PCA with a retained variance of 0.95 as suggested by the literature. This means that we only reduce the dimensionality of the data until the specified variance is met. In this way, we can cluster data that includes a high number of parameters p_n and, thus, do not need to limit our analysis on only a few selected parameters.

3.5 K-Means Clustering

After applying the PCA, we can cluster the data and gain decent clustering results. We decided to use the clustering algorithm *K-Means*, which is suitable and often used for clustering time series data [13]. Additionally, the results from the algorithm are easier to interpret than results from other techniques. We compute the *K-Means* clustering for all possible number of clusters. Subsequently, we apply the elbow method on the inertia values of the resulting clusterings to determine the optimal number of clusters k. Instead of manually guessing the location of the elbow, we let the *Kneedle* algorithm, presented in [20], find the point of maximum curvature in the inertia values. The resulting clusters $C = c_1, ..., c_k$ represent similar behaviors of the system parameters P throughout the log files. In the next step, we have to select those clusters from C, which represent the abnormal behavior of the system.

3.6 Abnormal Cluster Selection

For detecting anomalies we are interested in the abnormal behavior of the system. In this paper, we assume that our system behaves normally most of its execution time. This means that the number of time stamps in which abnormal behavior occurs will be small. Therefore, we expect the abnormal behavior to differ from the normal behavior of the system and to appear only rarely. In our clustering, this is represented by separate clusters that include only a small amount of instances.

To determine the number of small clusters that we should regard as abnormal, we suggest using the occurrence rate of the anomaly to detect. This rate can, for example, be the failure rate of the system or the attack rate on the system. Depending on how often the system fails or gets attacked, we should consider a different amount of clusters as abnormal. We assume that an expert can provide such an occurrence rate from his or her knowledge about the system. This knowledge can include the expert's experience with the system, a mental model of the anomaly to detect or reports on the system's processing.

In a first step, the list of clusters C needs to be rearranged to enable the selection of the abnormal clusters. The list of clusters C is sorted by the number of instances that the clusters contain. This yields a sorted list of clusters $CS = cs_1, cs_2, ..., cs_k$ with $size(cs_1) \leq size(cs_2) \leq ... \leq size(cs_k)$. Additionally, we need to specify the maximum number of abnormal windows that should be collected. This number can be computed based on the occurrence rate r of the anomaly to detect. The maximal number of abnormal windows is the product

of the occurrence rate r and the number of all windows. In the next step, the selection of clusters starts from the beginning of CS to gain the smallest clusters first. When a cluster is selected, all windows w_i from this cluster are added to the list of abnormal windows AW. Clusters are selected from the beginning of CS until the maximum number of abnormal windows is reached. The result of this step is the list of abnormal windows AW for the regarded log files.

4 Evaluation

In our experiments, we aim to evaluate how effectively the proposed approach can detect abnormal time spans in CPS log files. The intended result is a reduced amount of time and effort that needs to be invested by an expert to find an anomaly in CPS log files. Therefore, a high recall value is important for us. The resulting list of abnormal windows AW should include all anomalies. Otherwise, the expert will not be able to find the anomalies in the reduced amount of windows that we provide. If the precision values are too low, the amount of windows that the expert needs to inspect is too large to save time and effort. However, we believe that as long as we can save the expert time and effort during the anomaly detection, the precision values do not need to be as high as the recall values. This means that the list of abnormal windows AW is allowed to contain some windows with normal behavior as long as their quantity remains small. We use two data sets from different CPS in our experiments to emphasize the generic applicability of our approach. Both data sets contain labels about the included failures and attacks. In the experiments, we only use these labels to compute precision and recall values for our proposed method. These labels are not used by our clustering algorithm at any time.

4.1 Implementation and Experimental Setup

In our implementation, we use methods from the scikit-learn machine learning library [18] for computing PCA and K-Means. Additionally, we use the *Kneedle* algorithm from [20].

The experiments are run on a machine with an *Intel Xenon E5-2687W* processor with 15 cores and 128 GB of RAM. The K-Means algorithm is run 10 times with different beginning seeds. The run with the smallest inertia value is chosen. Our experiments take 6 s to 16 min, depending on the length of the log files and the size of the windows.

4.2 Data Sets

ALFA Data Set. [10,11] the first data set in our experiments is the ALFA data set, which includes log files of a fixed-wing Unmanned Aerial Vehicle (UAV). This data set encompasses 36 log files, which describe the behavior of the UAV while various failures occur. The log files include four different types of failures: aileron, elevator, engine and rudder failures. For each log file L, the specific failure and

its occurrence time are given. Our list of parameters P for this data set consists of the main eight features of the UAV:

$$P = pitch, roll, yaw, velocity, position, orientation, airspeed, windestimation$$

This data set includes failures of different categories that vary widely in their behavior. To yield more detailed results, we conduct our experiments separately for each of those failure types. This allows us to explore whether or not there are specific types for which our approach performs considerably better or worse compared to the other types. In this way, we can understand how well the clustering separates the abnormal behavior of each of those failure types from the normal behavior of the system. To perform this category-specific analysis, we combine all log files that contain failures of one category into one log file L. Then, our approach is applied to this single log file L to find abnormal behaviors that represent the failures of this specific category.

SWaT Data Set. [7] as a second data set, we use the SWaT data set collected by the *Centre for Research in Cyber Security, Singapore University of Technology and Design* (iTrust). The institute collected data from an industrial water treatment plant and launched 36 attacks on it during execution. In our experiments, we use the data that was recorded over six days in 2015 and work with a resolution of one data point per minute. This means that we regard all parameter values once per minute. The list of parameters P for this data includes the most dominant features of the water plant. These can be found in [7], where the authors provide an overview of the process of their test bed.

$$\begin{aligned} P = {} & lit101, lit301, lit401, p101, p201, p203, p205, p301, p401, p501, p602, \\ & fit201, fit401, ait201, ait202, aitT203, ait402, ait503, ait504, dpit301 \end{aligned}$$

Since we do not have further knowledge about this water plant, we do not know what the parameters in P stand for or which effects they have on the system. But, we are still able to apply our clustering approach to find abnormal time spans without this further knowledge.

The data set does not provide a subdivision of the attacks into attack categories that are based on the behavior of the attack types. Therefore, we will consider all data from this data set at the same time in our experiments. We thus apply our clustering approach to a log file L, which includes all data from the data set.

4.3 Experimental Results

When applying our proposed clustering approach to the log files $L_1, ..., L_L$ in the experiments, we will get a list of abnormal windows AW as a result. To evaluate

Table 1. The characteristics of the ALFA and the SWaT data set as well as the window size, which achieves the best values in the experiments for each category.

Data set	Failure/attack category	Failure/attack rate	Best window size
ALFA	Aileron failure	0.3365	100
ALFA	Elevator failure	0.1114	109
ALFA	Engine failure	0.1350	138
ALFA	Rudder failure	0.1921	33
SWaT	All attacks	0.1205	34

our approach, we compare the resulting abnormal windows $aw_i \in AW$ with the time spans that were labeled to include a failure or an attack in the data sets.

For computing precision, recall and F_1 values for the evaluation of our approach, we need a clear definition of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). A block in the following definitions is a set of parameter values of subsequent time stamps from the log files. Each block has a size of $windowSize/2$ and the first block starts at time stamp ts_0.

- TP are blocks that include an anomaly and are correctly marked as 'abnormal' by our proposed approach.
- FP are blocks that contain no anomaly and are wrongly marked as 'abnormal' by our clustering approach.
- TN are blocks that include no anomaly and are correctly marked as 'normal' by our proposed approach.
- FN are blocks that contain an anomaly and are wrongly marked as 'normal' by our clustering approach.

As mentioned earlier, the abnormal clusters are selected depending on the occurrence rate of the anomaly. In our experiments, this is the failure rate of the system or attack rate on the system. Normally, an expert would provide this occurrence rate. Since we do not know the specific systems of the used data sets, we will instead estimate the occurrence rate of the anomalies from the included labels. For this purpose, we divide the total number of time stamps with the number of time stamps in which a failure or attack occurs concerning the labels. Table 1 displays the calculated failure or attack rates for the two data sets in our experiments.

Additionally, this table shows the window size per category which achieves the best results in the experiments. We mentioned several methods for estimating a good window size in Sect. 3.1. Since we are not familiar with the particular systems of the data sets, we have to find the best window sizes differently than experts would normally do. We decided to run our proposed clustering approach on the log files $L_1, ..., L_L$ for each category and data set with the size of the window $windowSize \in [10, 200]$. The best window size can be determined by looking at the resulting F_1 scores. Note that this process is only possible if we

can compute the F_1 scores. The window size with the highest F_1 score is the optimal window size for finding anomalies in the regarded log files.

Table 2 presents the precision values, recall values and F_1 scores computed in our experiments. These values were calculated using the following formulas and the aforementioned definitions of TP, FP and FN:

$$precision = \frac{TP}{TP + FP} \qquad recall = \frac{TP}{TP + FN} \qquad F_1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Table 2. The results of the experiments displaying the calculated precision and recall values as well as the F_1 score for the ALFA and the SWaT data set.

Data set	Failure/attack category	Precision	Recall	F_1 score
ALFA	Aileron failure	0.6848	0.8889	0.7736
ALFA	Elevator failure	0.8571	1.0000	0.9231
ALFA	Engine failure	0.6545	0.7500	0.6990
ALFA	Rudder failure	0.8209	0.9167	0.8661
SWaT	All attacks	0.8669	0.7357	0.7959

We can detect aileron failures from the ALFA data set with a precision of 0.6848, a recall of 0.8889 and a resulting F_1 score of 0.7736. For elevator failures, we can achieve even better results. The precision value raises to 0.8571 and we can identify all failures, which is presented by a recall value of 1.0. A precision of 0.6545 and a recall of 0.75 can be obtained for engine failures of the ALFA data set. When clustering the rudder failures from the same data set, we accomplish high precision and recall values of 0.8209 and 0.9167. The results of the SWaT data set show that we can detect attacks with a precision value of 0.8669 and a recall value of 0.7357. We can achieve high recall values for the SWaT data set and all categories in the ALFA data set. This shows that our proposed approach can detect a large amount of the included failures and attacks as abnormal behaviors of the system. From a practical perspective, the precision values for aileron and engine failures are high enough to save the expert time when trying to find an anomaly even though they are not as high as in the other categories. For elevator failures, rudder failures and all attacks in the SWaT data set, we achieve high precision values above 80%. This shows that we can efficiently reduce the amount of information that the expert needs to review.

In the following, we will compare the results of our experiments on the SWaT data set with results from experiments on the same data set in the literature. The authors of [9] present two model-based anomaly detection methods in their paper. They compare the performance of (1) *Deep Neural Networks* (DNN) and (2) *one-class Support Vector Machines* (SVM) on the SWaT data set. The parameters of these two unsupervised machine learning algorithms need to be fine-tuned to achieve good results. To train the DNN, a high amount of time and

data is necessary. For the SVM approach the classification boundary between normal and abnormal behavior needs to be learned beforehand. A third method for anomaly detection on the SWaT data set is evaluated in [14] and called (3) *TABOR*. The authors propose to first train a *Bayesian Network* (BN) to recognize the normal behavior of the sensors and actuators of the system. Then, this BN is used to distinguish between normal and abnormal behavior of this specific system. We will focus the comparison on the number of anomalies that were detected by each of these approaches. We were not able to compare other values from these papers since the authors use different methods to compute TP, FP and FN. The SWaT data set includes 36 attacks. From [9], we can obtain that the (1) DNN-based approach detects 13 and the (2) SVM-based solution finds 20 attacks. These numbers are derived from Table 3 in [9] and not computed by ourselves. The method (3) *TABOR* can discover 24 attacks in the data set, which is displayed in Table 4 in [14]. We compare these numbers that were directly derived from the two papers, with the performance of our approach. With our proposed approach, we can detect 23 of the 36 attacks. This shows that we can discover more attacks in the SWaT data set than the (1) DNN-based and (2) SVM-based methods. Also, we detect only one attack less than the third method called (3) *TABOR*. This indicates that we can effectively detect anomalies in the SWaT data set compared to current literature. Most importantly, we can apply our approach to log files of different CPS without the need to fine-tune a large number of parameters or learn a model beforehand. Our evaluations, therefore, show the generic applicability of our proposed approach and its efficiency in comparison with other approaches from the literature.

5 Related Work

Automating anomaly detection in Cyber-Physical Systems is a broad research field. Anomalies are either identified by working on log files of the system [5,8,9,15] or by monitoring parameters of the system in real-time [2,17]. We decided to develop an approach that a posteriori detects anomalies in CPS log files.

For anomaly detection in log files [8] proposes the *Local Outlier Factor* algorithm, which statistically computes the reachability distance to the neighboring points. In contrary to this approach, we plan to develop a method which is per se suited to work on time series data by using *Dynamic Time Warping* as our distance metric. In [15], a hierarchical clustering approach is applied to log files to detect reoccurring failures. A representative sequence is chosen for each cluster. The authors of [5] build a *Finite State Automaton* for each component of the system from its log files to represents the normal work process of each component. In the next step, both approaches compare new log files with the representative sequences or *Finite State Automata* to detect anomalies. In these two papers, the normal behavior and representative sequences are derived from previously collected log files of the system. In this way, they are only able to detect reoccurring failures and anomalies and need to build a new model for each new system. Our approach, on the contrary, can be applied to log files of

any CPS without having to train or learn a model beforehand. The authors of [9] suggest to use unsupervised machine learning approaches to find anomalies in log files from a water purification plant. In this approach, the authors do not need to build a fine-tuned model of the system. They still need to tune different parameters before they can start the learning phase of their machine learning approaches though. In contrary to this, the expert only needs to set a single variable in our approach. For real-time anomaly detection, [17] monitors one specific parameter (motor temperature) and [2] defines process invariants, which have to hold as long as the system is in a specified state. The invariants are derived from a model of the system. We aim for detecting anomalies of different origins, instead of focusing on only one parameter of the system as [17] suggests. Our goal is to a-posteriori find abnormal behaviors in CPS log files. In this way, we want to create an approach that is independent of the system itself and where we do not need to create a fine-tuned model of our system as needed in [2].

6 Conclusion

We first outlined the challenges of manually detecting anomalies in CPS log files. A large amount of logged parameter values, as well as the high complexity of these systems, complicate the manual discovery of anomalies. Therefore, we propose to automate this process by applying a clustering approach to the multidimensional time series data. The abnormal behavior is expected to occur only rarely and to differ from the normal behavior of the system. Thus, we anticipate the abnormal behaviors from the log files in separate and small clusters. For the selection of the abnormal clusters, the occurrence rate of the anomaly is utilized. We assume that an expert can provide this occurrence rate from previously collected data and his or her knowledge of the domain and system. In our experiments with two data sets from different types of CPS, we could show that our generic approach can effectively detect anomalies in the log files. Naturally, there exists a trade-off between the analytical run-time of our proposed approach and the size of the log files. If we encounter the problem of not possessing enough computational power, we have to reduce the size of the log files. This can, for example, be achieved by reducing the number of data points per time unit or by having an expert predict which parts of the log files should be considered in the analysis.

There is no need to create a fine-tuned model of the regarded system in our approach. The only variable that needs to be set is the size of the windows, in which the data is cut. We expect that the expert will be able to set this variable properly after discovering several anomalies in a system. Another option is the use of optimization techniques to find a good estimate of the window size. In this paper, we assume that the expert knows a good estimate of this variable. We would like to investigate various methods and optimization criteria for assessing a good window size in the future. Additionally, we would like to expand our experiments to several other types of CPS to strengthen the generic applicability of our proposed solution.

References

1. Abdi, H., Williams, L.J.: Principal component analysis. Wiley Interdisc. Rev. Comput. Stat. **2**(4), 433–459 (2010)
2. Adepu, S., Mathur, A.: Using process invariants to detect cyber attacks on a water treatment system. In: Hoepman, J.-H., Katzenbeisser, S. (eds.) SEC 2016. IAICT, vol. 471, pp. 91–104. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33630-5_7
3. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD Workshop, Seattle, WA, vol. 10, pp. 359–370 (1994)
4. Caporuscio, M., Flammini, F., Khakpour, N., Singh, P., Thornadtsson, J.: Smart-troubleshooting connected devices: Concept, challenges and opportunities. Future Gener. Comput. Syst. **111**, 681–697 (2019)
5. Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: 2009 Ninth IEEE International Conference On Data Mining, pp. 149–158. IEEE (2009)
6. Gillian, N., Knapp, B., O'modhrain, S.: Recognition of multivariate temporal musical gestures using N-dimensional dynamic time warping. In: Nime, pp. 337–342 (2011)
7. Goh, J., Adepu, S., Junejo, K.N., Mathur, A.: A dataset to support research in the design of secure water treatment systems. In: Havarneanu, G., Setola, R., Nassopoulos, H., Wolthusen, S. (eds.) CRITIS 2016. LNCS, vol. 10242, pp. 88–99. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71368-7_8
8. Harada, Y., Yamagata, Y., Mizuno, O., Choi, E.H.: Log-based anomaly detection of CPS using a statistical method. In: 2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 1–6. IEEE (2017)
9. Inoue, J., Yamagata, Y., Chen, Y., Poskitt, C.M., Sun, J.: Anomaly detection for a water treatment system using unsupervised machine learning. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 1058–1065. IEEE (2017)
10. Keipour, A., Mousaei, M., Scherer, S.: Alfa: a dataset for UAV fault and anomaly detection. arXiv, arXiv–1907 (2019)
11. Keipour, A., Mousaei, M., Scherer, S.: Automatic real-time anomaly detection for autonomous aerial vehicles. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 5679–5685. IEEE (2019)
12. Lee, E.A.: Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. IEEE (2008)
13. Liao, T.W.: Clustering of time series data-a survey. Pattern Recogn. **38**(11), 1857–1874 (2005)
14. Lin, Q., Adepu, S., Verwer, S., Mathur, A.: Tabor: a graphical model-based approach for anomaly detection in industrial control systems. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 525–536 (2018)
15. Lin, Q., Zhang, H., Lou, J.G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 102–111. ACM (2016)
16. Lou, J.G., Fu, Q., Yang, S., Xu, Y., Li, J.: Mining invariants from console logs for system problem detection, In: USENIX Annual Technical Conference, pp. 23–25 (2010)

17. Lu, H., et al.: Motor anomaly detection for unmanned aerial vehicles using reinforcement learning. IEEE Internet Things J. **5**(4), 2315–2322 (2017)
18. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
19. Petitjean, F., Gançarski, P.: Summarizing a set of time series by averaging: from steiner sequence to compact multiple alignment. Theoret. Comput. Sci. **414**(1), 76–91 (2012)
20. Satopaa, V., Albrecht, J., Irwin, D., Raghavan, B.: Finding a "kneedle" in a haystack: detecting knee points in system behavior. In: 2011 31st International Conference On Distributed Computing Systems Workshops, pp. 166–171. IEEE (2011)