Enable learning of Hybrid Timed Automata in Absence of Discrete Events through Self-Organizing Maps

Alexander von Birgelen and Oliver Niggemann

Ostwestfalen-Lippe University of Applied Sciences, Institute Industrial IT,
Langenbruch 6, 32657 Lemgo, Germany
{alexander.birgelen,oliver.niggemann}@hs-owl.de
http://www.init-owl.de

Abstract. Model-based diagnosis is a commonly used approach to identify anomalies and root causes within cyber-physical production systems (CPPS) through the use of models, which are often times manually created by experts. However, manual modelling takes a lot of effort and is not suitable for today's fast-changing systems. Today, the large amount of sensor data provided by modern plants enables data-driven solutions where models are learned from the systems data, significantly reducing the manual modelling efforts. This enables tasks such as condition monitoring where anomalies are detected automatically, giving operators the chance to restore the plant to a working state before production losses occur. The choice of the model depends on a couple of factors, one of which is the type of the available signals. Modern CPPS are usually hybrid systems containing both binary and real-valued signals. Hybrid timed automata are one type of model which separate the systems behaviour into different modes through discrete events which are for example created from binary signals of the plant or through real-valued signal thresholds, defined by experts.
However, binary signals or expert knowledge to generate the much needed discrete events are not always available from the plant and automata can not be learned. The unsupervised, non-parametric approach presented and evaluated in this paper uses self-organizing maps and watershed transformations to allow the use of hybrid timed automata on data where learning of automata was not possible before. Furthermore, the results of the algorithm are tested on several data sets.

1 Introduction

Increasing product variety, product complexity and pressure for efficiency in a distributed and global production chain cause production systems to evolve rapidly: they become modular, can be parameterized and contain a growing set of sensors [1].

In order to enable European SMEs to face these challenges and to utilize new technical possibilities, the Horizon2020 project IMPROVE is aimed at developing user support functions in terms of self-diagnosis (i.e. condition monitoring,

O. Niggemann and P. Schüller (Eds.), *IMPROVE – Innovative Modelling Approaches for Production Systems to Raise Validatable Efficiency*, Technologien für die intelligente Automation 8, https://doi.org/10.1007/978-3-662-57805-6_3

predictive-maintenance) and self-optimization (e.g. energy optimization, output optimization).

Models can be constructed manually by an expert, but this is difficult, costly and time consuming in today's complex and evolving production plants [21]. Instead of only relying on human expertise and additional engineering steps formalizing the necessary knowledge, the tasks stated above will be taken on in a data-driven way [14] where models are learned automatically from the data. For anomaly detection, the live data from the plant is compared to the predictions of the learned model and deviations from the normal behaviour are classified as anomalous.

The type of model depends on a variety of conditions such as available signals, the task of the model and the overall nature of the system. Modern cyber-physical production systems (CPPS) usually are hybrid systems, meaning they comprise both binary and real-valued signals. In general, these dynamic systems are state based, for example the system's state is defined by its current and previous binary control signals, and the actions taken out are time dependent [13].

One approach to perform anomaly detection in such systems is to use hybrid timed automata [11] as a normal behaviour model which utilize discrete events to learn the system's normal behaviour. These events often cause so called mode or state changes in industrial plants, e.g. *conveyor is running* or *heater is on*, which result in different behaviour of the system in each mode, as shown in Figure 1. Changes in the binary control and sensor signal values of the system can be utilized as discrete events directly. Another way to get discrete events is to set thresholds on continuous signals but this requires additional expert knowledge.

Hybrid timed automata are well suited to learn the normal behaviour in terms of the modes/states, transitions and corresponding timings from data in an unsupervised manner by using the aforementioned binary control signals of a system. The real-valued signals are processed within the states using other types of models such as regression models or models which reduce the dimensionality of the real-valued data. An example for this is the nearest-neighbor principal component analysis (NNPCA) [5] which is used for the experiments in section 4. The NNPCA was chosen because it is used frequently in a variety of research projects in our institute.

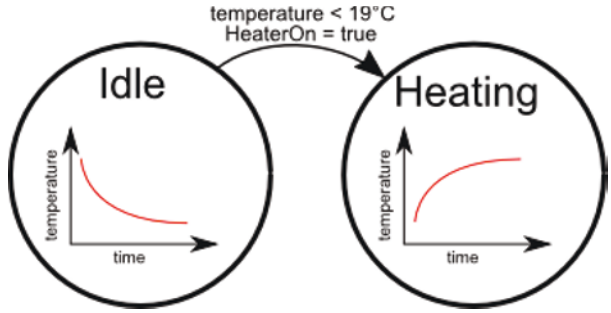


Fig. 1. An example of a hybrid automaton for a simple heating system with two states.

Unfortunately, expert knowledge about signal thresholds is almost never available and binary sensor signals are also not always available or not meaningful. This occurs for example when data is recorded using internal trace functionalities from drive controllers, which offer very high sampling rates but lack knowledge of variables from the programmable logic controller (PLC).

In this paper we present an unsupervised, non parametric approach to learn hybrid timed automata in absence of discrete events. The approach uses self-organizing maps (SOM) and watershed transformations to extract modes and generate discrete events in an unsupervised manner using only real-valued signals. The generated events can then used to capture the normal behaviour of the system by learning hybrid timed automata on data where they were not applicable before. The learned hybrid automaton is then used for anomaly detection in the real-valued signal values and in the time domain by analysing the transitions between the automaton's states.

The contents of this paper are structured as follows: section 2 introduces the existing modelling formalisms which are then combined in section 3 to generate discrete events from real-valued signals in an unsupervised manner. Experimental results from three different data sets, one artificial and two real world ones, are given in section 4. Finally, the paper is concluded in section 5.

2 Methodologies

2.1 Hybrid Timed Automata

Hybrid timed automata have proven to be a great tool to learn the normal behaviour of a system and detect deviations from it. Discrete events are required to learn an automaton. These events often cause mode changes in the system and the timing of these events is an important indicator for the health of the system. Hybrid timed automata are used to separate these modes, learn the transitions and timing between them and model the behaviour of the real-valued signals for each of the modes or states in the automaton.

An easy approach to obtain discrete events can be directly extracted from changes in the binary control and sensor signals of the system. It is also possible to obtain discrete events through thresholds for real-valued signals such as *temperature* $< 19^{\circ} C$ [7]. However, setting the thresholds and combinations of conditions for the continuous signals requires expert knowledge which is usually not available for real world automation systems. For unsupervised learning of these automata only binary control signals are used to obtain the discrete events, such as $HeaterOn = true$. Algorithms such as the online timed automaton learning algorithm (OTALA) [9] and its hybrid extension can work in an online, unsupervised manner, and do not require additional expert knowledge.

A hybrid automaton generated by the aforementioned algorithm can be defined as described in Definition 1.

Definition 1. *A hybrid timed, probabilistic automaton is a tuple* $A = (S, s_0, \Sigma, T, \delta, P, \theta)$*, where*

- *S is a finite set of states where $s \in S$.*
- *s_0 is the initial state which can be given by the systems state at the start of the training.*
- *Σ is the set of discrete events. Events $a \in \Sigma$ is linked to the transitions of the automaton.*
- *T is the set of transitions with $t \in T$ and $t = (s, a, s'))$, $s, s' \in S$ are source and destination state, $a \in \Sigma$ is the trigger event of the transition.*
- *The timing constraint $\delta : T \to I$ assigns a time interval to a transition $t \in T$, where I is a set of time intervals. The time here usually refers to the elapsed time since the last event occurred.*
- *P is a set of probabilities: for each transition $t \in T$ probability $p \in P$ is calculated.*
- *$\theta_{s \in S}$ describes a model for each state $s \in S$ which captures the behaviour of the real valued signals. Real valued signals are not captured by the discrete part of the automaton. These state models $\theta_{s \in S}$ are learned for each state of the automaton using other models such as linear regression, decision trees and others, such as the nearest neighbour principal component analysis (NNPCA) used in section 4 of this paper.*

The learned automaton can then be used to detect a variety of different classes of anomalies. This can for example be done using the anomaly detection algorithm (ANODA) [10] which can detect the following types of anomalies:

- **Unknown event / Wrong event sequence**: an event occurred which was not observed in the current state.
- **Timing error**: a transition occurred outside of the learned time bounds.
- **State remaining error**: when more time passed than for the latest event and the state is not a final state, then we have a state remaining error.
- **Probability error**: the probabilities of transitions for the new data are calculated and compared to the previously learned probabilities and an error is generated when deviations are too large.
- **Continuous error**: for each state, an additional anomaly detection for the real-valued signals can be performed using the internal state models.

2.2 Self-Organizing Map

The self-organizing map (SOM), also referred to as self-organizing feature map or kohonen network, is a neural network that can be associated with vector quantization, visualization and clustering but it can be used as an approach for non-linear, implicit dimensionality reduction [22]. The reduction is performed in a qualitative, implicit way. SOM's were chosen over classic clustering techniques due to their ability to find and visualize clusters in high dimensional data, which is often difficult with traditional clustering techniques [18]. A SOM consists of a collection of neurons which are connected in a topological arrangement which is usually a two dimensional rectangular or hexagonal grid. The input data is mapped to the neurons forming the SOM. Each neuron is essentially a weight vector of original dimensionality.

Definition 2. *the self-organizing map* $SOM = (M, G, d)$ *forms a topological mapping of an input space* $O \subset \mathbb{R}^m$, $m \in \mathbf{N}$ *and consist of*

- *a set of neurons* M.
- *each neuron* $n \in M$ *has a weight vector* $\mathbf{w}_n \in \mathbb{R}^m, m \in \mathbb{N}$.
- G *is usually a two-dimensional rectangular or hexagonal lattice in which the neurons* $n \in M$ *are arranged. Toroidal versions of these topologies are also common.*
- $d(\mathbf{x}, \mathbf{y})$ *is the distance measure to calculate the distance between two vectors* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *which can for example be weight vectors and/or vectors in the input space. Usually, the euclidean distance is used but other measures, such as the mahalanobis distance, can be used.*
- *an input sample* $\mathbf{o}_i \in \mathbb{R}^m, i \in \mathbf{N}$ *is mapped to the SOM through its best matching unit (BMU). The BMU is given by* $bmu(\mathbf{o}_i) = \operatorname{argmin}_{k \in M} d(\mathbf{o}_i, \mathbf{w}_k)$

One way to learn a SOM from data is a random batch training approach: The initial values of the neuron's weight vectors for the training can be randomly initialized or sampled from the training data. All samples from the training data are presented to the algorithm within one epoch. A best matching unit (BMU) is calculated for each input sample from the training data by finding the neuron which has the smallest distance to the sample. The BMU and all of its neighbouring neurons, assigned through the topology and neighbourhood radius, are shifted towards the input sample. Both the neighbourhood radius and strength of the shift decrease over time. The training stops after a chosen amount of epochs.

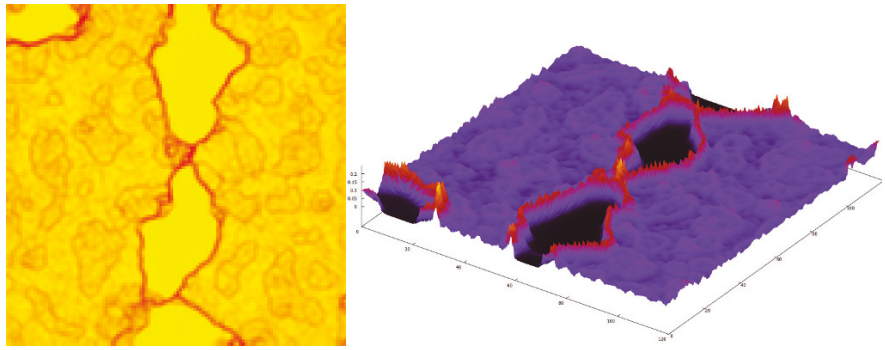
Each neuron of the SOM represents a part of the training data. Areas in the training input space with few examples are represented by few neurons of the SOM while dense areas in the input space are represented by a larger number of neurons.

The unified distance matrix (u-matrix) [19] allows a three-dimensional visual identification of clusters contained in the SOM. It calculates the sum of distances to neighbouring neurons according to the SOM's topology and visualizes clusters contained in the, usually high dimensional, training data. The neurons located on the borders of the non-toroidal SOM have fewer neighbours than the remaining neurons. Therefore, the summed distance is divided by the number of neighbours of the corresponding neuron to account for the different amounts of neighbouring neurons. The X and Y coordinates of the neurons represent the first two dimensions. The third dimension is given by the average distance to neighbouring neurons as in definition 3.

It can be visualized directly in 3D or in 2D using a color gradient as shown in Figure 2.

Definition 3. *for each neuron* $n \in M$ *and its associated weight vector* $\mathbf{w}_n$*, the u-matrix height is given by* $U(n) = \frac{1}{|NN(n,G)|} \sum_{k \in NN(n,G)} d(\mathbf{w}_n, \mathbf{w}_k)$*, where* $NN(n, G)$ *is the set of neighbouring neurons of* n *defined by grid* G *and* $d(x, y)$ *is the distance used in the SOM algorithm.*

The u-matrix representation illustrates why SOM's were chosen: SOM's tend to keep neurons with similar signal weights closely together. This results in a to-



(a) Colored 2D (b) 3D

Fig. 2. Different u-matrix visualizations of a 120x120 SOM.

pographic landscape with valleys and ridges. Valleys represent clusters which are separated by the ridges.

Other works such as [8], [16] and [3] already performed anomaly detection on different processes by tracking the trajectory of the working point on the SOM: the observations are mapped to their corresponding BMU's as soon as they are recorded. Over time, the path or trajectory of the BMU can be observed and deviations from the known path indicate anomalous behaviour.

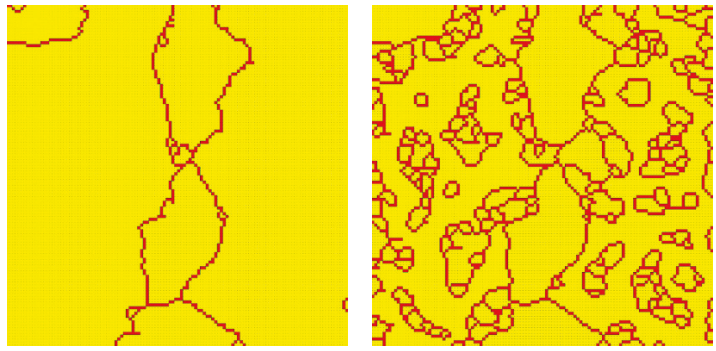
2.3 Watershed Transformation

The u-matrix representation allows visual identification of clusters in high dimensional data sets. This is easy to do for humans, but more difficult for a machine. Since the u-matrix representation can be plotted as an image, clustering algorithms from the image processing domain, such as the watershed transformation [12], can be used on the u-matrix representation of a SOM to identify the clusters in a mathematical way.

This works analogous to rain falling on top of the u-matrix. The water runs from higher regions to the lower regions and consequently flooding the basins. When the water level gets high enough so two basins merge, a ridge forms which separates them.

The watershed transformation dissects the u-matrix into different clusters, separated by the so-called watershed lines. Watershed lines separate the different basins and do not belong to any of the clusters. The basins can be interpreted as stationary process phases while the watershed lines represent transient process phases [6].

The implementation used here is the Vincent-Soille watershed algorithm which performs the watershed transformation in a non-recursive manner [20]. The sensitivity of the algorithm can be adjusted by setting a number of levels which in turn influences the number of final clusters found. Figure 3 shows examples using different levels on the same u-matrix.



(a) 4 level, 16 cluster (b) 6 level, 197 cluster

Fig. 3. Watershed transformations of a u-matrix.

In the end, we receive a mapping for each neuron of the map to its corresponding cluster (Definition 4).

Definition 4. *the watershed transformation maps each neuron* $n \in M$ *to a cluster* c*, with* C *being a set of clusters and* $c = [0, |C| - 1] \in \mathbb{N}$.

3 Learning hybrid timed automata without discrete events

In order to learn hybrid timed automata, without binary control signals and without expert knowledge about thresholds for real-valued signals, it is necessary to derive the discrete events using an alternative way. For complex systems, the events can also be related to combinations of different real-valued signal values. Here, SOMs are used as a preprocessing step to extract the different modes of the system in an unsupervised manner.

As mentioned before, the trajectory of the BMU can be tracked and recorded. An automaton can be used to learn and track the trajectory of the BMU's. This can result in a large number of states in the automaton and many of the states might not contain enough data to learn a model from the data within the state. In this paper, we use the modes extracted from the SOM's u-matrix watershed transformation to group the data from the different modes and ultimately reduce the number of states of the automaton.

This section describes the generation of discrete events using SOMs. Figure 4 shows the general steps of the presented approach. A SOM is learned from the input data and the transitions between emerging clusters or rather system modes are used to generate discrete events. This is essentially a preprocessing procedure which allows learning hybrid timed automata in case the input data does not contain discrete events. After events are generated an automaton can be learned and then used to detect anomalies within the real-valued signal values and also within the timing and probabilities of the events.

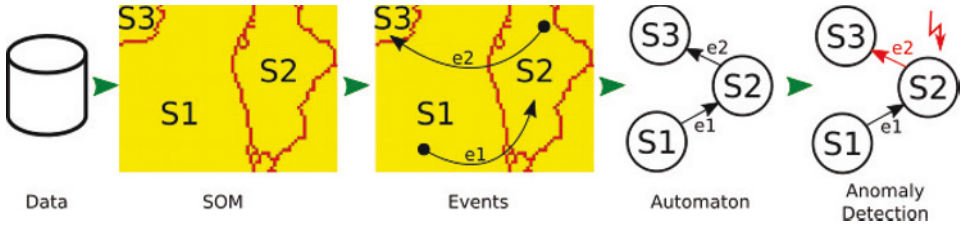


Fig. 4. General steps of the presented algorithm.

First, real-valued signals are recorded during normal operation of the plant to generate dataset O which consists of many obervations $o \in O$ and represents the input space of the model. The data can be seen as multivariate time series and each observation is equipped with a timestamp which is used when the automaton is learned. Then, the SOM-Discretization algorithm (Algorithm 1) generates discrete events for the data to allow learning of hybrid timed automata.

A SOM is trained on the recorded normal behaviour data (step 3). The size of the SOM is automatically calculated according to [17], where the the number of neurons $|M| \approx 5\sqrt{N}$ with N being the number of observations. The ratio of the side lengths is the ratio of the two largest eigenvalues of the data's covariance matrix. A normalization should generally be done before the SOM training but this depends on the input space and is therefore optional (step 2).

The SOM's u-matrix is calculated (step 4) and then clustered using the watershed transformation (step 5). The watershed levels are adjusted so the final cluster count is close to the shorter side length of the SOM. This approach seemed to work on all of the tested datasets (section 4 but this can probably be improved in future works. Each basin represents a stationary process phase and gets a unique number $i = [0, |C| - 1], i \in \mathbb{N}$ for identification. The watershed lines are transient process phase all receive a negative identification number to distinguish them from the stationary process phases. Observations mapped to transient process phases, the borders left after the watershed transformation, are assigned to the same cluster as the previous observation and therefore receive the same event vector as the previous observation (step 12). The event vector for cluster $c \in C$ in step 10 is created such that:

$$\mathbf{v}(c) = \left(v_1, v_2, ..., v_i, ..., v_{|C|-1} \middle| v_i = \begin{cases} 1 & \text{if } i{=}c \\ 0 & \text{else} \end{cases} \right)$$

The event vector is then concatenated with the original signal vector in step 14. The data now contain binary signals which are then interpreted as discrete events by hybrid automaton learning algorithms such as the hybrid OTALA used in this paper.

The anomaly detection can be performed either offline or online. Algorithm 2 shows the procedure for an online anomaly detection, where each observation o is tested as soon as it is received. When normalization was used for the training each new observation must also be normalized based on the normalization parameters

Algorithm 1 Generation of discrete events

```
1: procedure SOM-Discretization(O)
2:     O ← Normalise(O)
3:     SOM ← TrainSom(O)
4:     U ← U-Matrix(SOM)
5:     C ← Watershed(U)
6:     for all o ∈ O do
7:         bmu_o ← BMU(SOM, o)
8:         c_o ← C(bmu_o)
9:         if c_o ≥ 0 then
10:            v_o ← CreateV(c_o)
11:        else
12:            v_o ← v_{o−1}
13:        end if
14:        p ← {v_o, o}
15:    end for
16: end procedure
```

calculated before (step 2). The best matching unit is calculated and linked to its corresponding cluster (steps 3, 4). When the observation falls onto a transient process phase it is assumed to be in the same cluster as the previous observation (step 8). When mapped to a stationary process phase, the discrete event signal vector is generated and appended to the observation in steps 6 and 10. The observation is then given to the ANODA algorithm to perform the anomaly detection using the learned automaton.

Algorithm 2 Preprocessing for online anomaly detection

```
1: procedure SOM-Discretization-OnlineAD(o)
2:     o ← Normalise(o)
3:     bmu_o ← BMU(SOM, o)
4:     c_o ← C(bmu)
5:     if c_o ≥ 0 then
6:         v_o ← CreateV(c_o)
7:     else
8:         v_o ← v_{o−1}
9:     end if
10:    o ← {v_o, o}
11:    return o
12: end procedure
```

4 Experiments

This section presents experimental results of the presented approach using one artificial and two real world data sets.

4.1 Artificial test data

We created an artificial dataset through a simple PLC application. The PLC moves a virtual, linear SoftMotion [2] axis back and forth between two target positions. The drive uses trapezoid ramps with limited speed and accelerations. The drive related data from the software PLC are acquired through OPC-UA subscriptions in a 100ms publishing interval.

The data contains three real-valued signals: target position, actual velocity and actual speed. The training set contains 15 cycles, while the evaluation set contains five normal and five anomalous cycles. Maximum acceleration and deceleration are decreased during the anomalous cycles. An excerpt of the data is shown in Figure 5. The anomalies are labelled, to later evaluate the score for the anomaly detection. All of the machine learning methods here work unsupervised and have no knowledge about the labels.

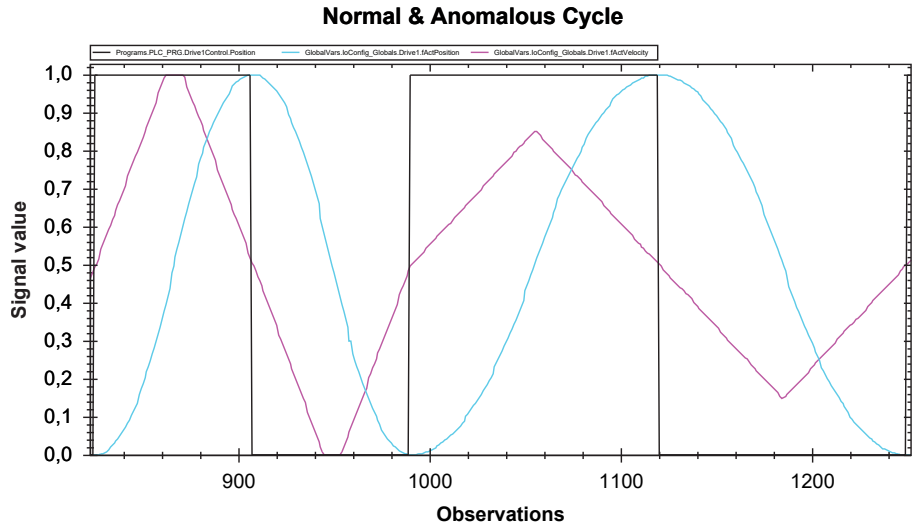


Fig. 5. One normal (827-991) and one anomalous (992-1251) cycle, scaled to range [0,1].

The nearest-neighbor PCA (NNPCA) [4] was first used on the full training data and then second as state-model for the hybrid automaton.

The data was reduced to two dimensions and then used for the anomaly detection. The training observations provide the reference to calculate the distance

for each evaluation observation. When the distance to the reduced training space exceeds a certain threshold, the observation is considered anomalous. The threshold is calculated using a mexican-hat wavelet, converting the distance to an error probability [5].

Setting the threshold is not trivial and highly depends on the input data. A 100% threshold is good against false positives but also might be not sensitive enough to find the true positives. Lowering the threshold usually increases the true positive rate but at the risk of more false positives.

Figure 6 shows a plot of two-dimensional NNPCA learned from the normal behaviour used as training data. The anomalies from the evaluation data are unknown to the model and are shown here to visualize the separation between normal and anomalous behaviour. A threshold of 25% was selected, so every observations which gets a probability greater or equal to 25% of being anomalous is marked anomalous. Even with this low threshold, not a single anomaly is found by the NNPCA on the full dataset as the separation between normal and anomalous behaviour is not large enough to be found with the given threshold. Furthermore, the NNPCA model does not include time in an explicit way and therefore detection of anomalies in the time domain difficult or often not possible.

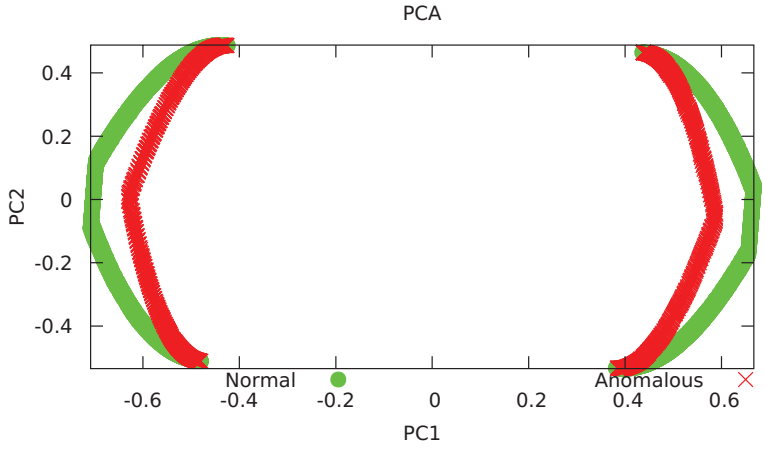


Fig. 6. Plot of the full dataset in the two-dimensional PCA space. Only normal behaviour is known to the learned model. Anomalies plotted to show the separation between normal and anomalous.

Now, algorithm 1 is used to generate discrete events so the data can be separated by a hybrid automaton. The automatic selection of parameters results in a SOM with 24x10 neurons and 11 clusters as shown in Figure 7.

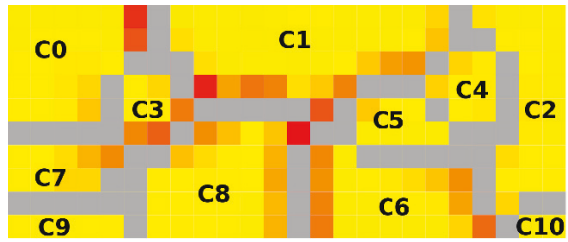


Fig. 7. Clusters on the SOM. Grey areas mark transient process phases.

With this, a hybrid automaton is trained using the hybrid OTALA algorithm and the NNPCA [5] as a state model with the exact same parameters as before. The resulting automaton is shown in Figure 8. Behind each state stands the state model from which two are shown in Figures 9a and 9b showing the separation of the normal and anomalous behaviour.

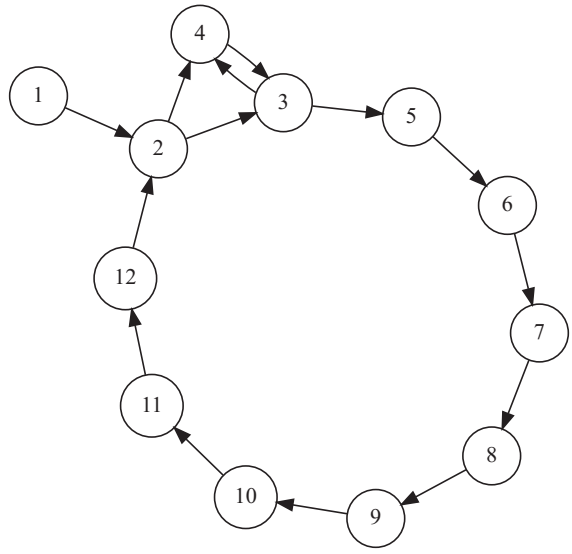
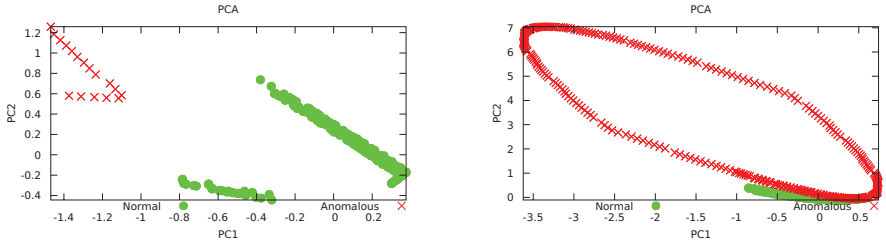


Fig. 8. The learned automaton with 12 states. State 1 is an initial state. Each cluster is mapped to one state.

The presented approach for the event generation in preparation for automaton learning calculates the necessary parameters from the training data as described in section 3. The settings for the NNPCA within the states were kept identical and the hybrid automaton achieves F1 measure of 97.53%, compared to the 0% F1 achieved by the NNPCA on the full dataset.



(a) State 3 (b) State 7

Fig. 9. Two state models from the automaton in Figure 8 showing the normal and mapped anomalous behaviour.

Table 1 shows some more details about the scores. The automaton offers another advantage: deviations in the timing and sequence of the system's behaviour can now be detected, which is not possible with the NNPCA alone.

4.2 High Rack Storage System

The High Rack Storage System or HRSS is a demonstrator from the SmartFactory-OWL which transports wares between its different shelves. The data from the system's drives includes position, power and voltage signals to a total of 17 real-valued signals, after removal of signals with zero-variance. The training data contains 107 cycles of the same transport operation. Evaluation data contains 113 cycles and was generated by modifying the training cycles in different ways such as increasing or decreasing one or multiple signals by different amounts (e.g. 20%), shortening of cycles and inserting constant sequences. This artificially generated anomalies are labelled and are used to calculate the scores of the anomaly detection.

The NNPCA reduces the data to 15 dimensions, keeping a variance coverage of 99.85%. A 60% threshold for the anomaly detection, successfully identifies 35 of the 7365 labelled anomalies resulting in an atrocious F1 score of 1.32%. The event generation creates a 34x22 SOM resulting in 23 clusters which are then captured by the automaton (Figure 10).

The automaton with the 15-dimensional NNPCA inside the states identifies 1794 true positives and reaches an F1 measure of 40.08% which is a drastic increase to the full-dataset NNPCA. It is to be noted that the anomalies were generated without respect to the overall dataset, so a 20% increase of a signal might not be significantly different from the overall training data and therefore it is possible that some of the anomalies are not significant enough compared to the whole dataset and might not be detectable.

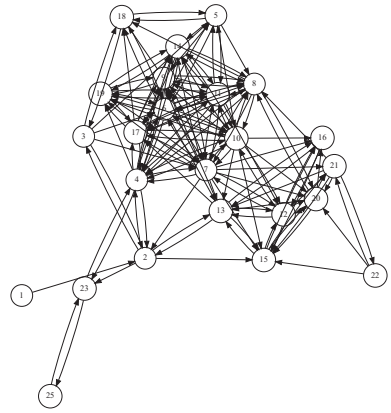


Fig. 10. The learned automaton for the storage system.

4.3 Film-Spool Unwinder

The third dataset presented in this paper was recorded from the cutting unit of a Vega shrink-wrap packer by OCME [15]. The machine groups loose bottles or cans into set package sizes, wraps them in plastic film and then heat-shrinks the plastic film to combine them into a package.

The drive controllers within the film cutting unit recorded chunks of data which each contain 2048 observations using their built-in scope functionality at a 4ms resolution. Here, we only consider two signals from the film unwinder drive: actual speed and lag error. 40 chunks were used for training and another 32 chunks were used as evaluation data. The film spool depleted during the last cycles, so every observation in the last 12 cycles was labelled anomalous. A dimensionality reduction is not necessary for two dimensions, but the NNPCA was still used to keep the model consistent for the experiments presented here. The NNPCA on the full dataset detected anomalous with an F1-measure of 18.63%. The automatically calculated size of the SOM is 64x22 neurons resulting in the automaton shown in Figure 11. The NNPCA on the full data and a selection of the 24 state models is shown in Figure 12.

The absolute scores for this dataset have to be taken with a grain of salt because the anomalies are not labelled observation-perfect but a generous range of observations was marked as anomalous. The F1 score for the anomaly detection with the automaton reaches 60.00%, resulting in an increase of roughly 41% (Table 1) compared to the NNPCA on the full dataset.

5 Conclusion

This paper presented an unsupervised, non-parametric approach which allows application of hybrid timed automata on data which do not allow their use due to a lack of knowledge about discrete events which are needed to learn and use hybrid

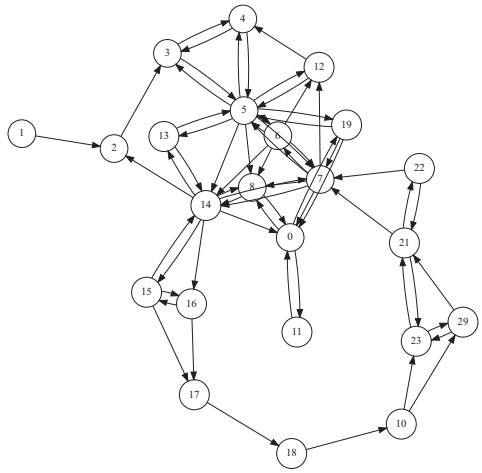
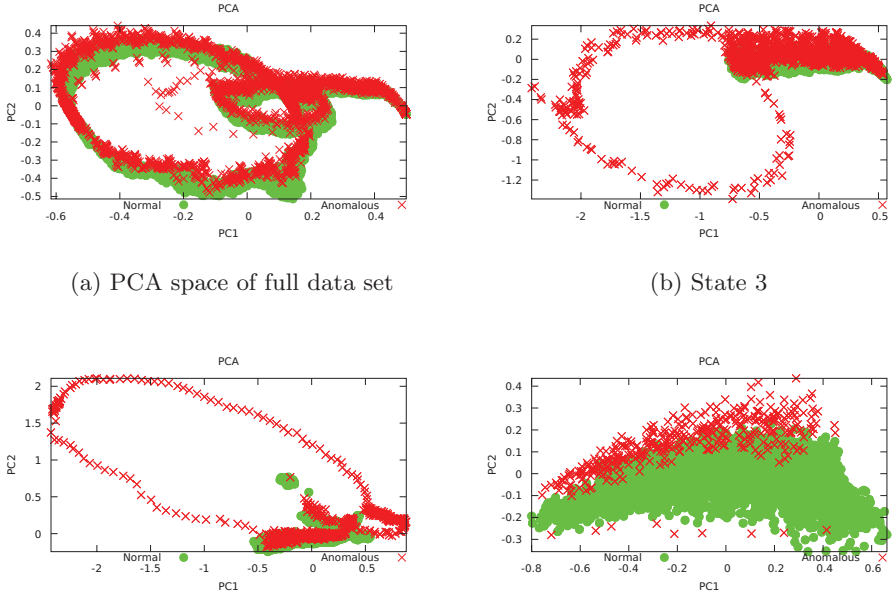


Fig. 11. Unwinder Automaton



(a) PCA space of full data set

(b) State 3

(c) State 7

(d) State 18

Fig. 12. PCA of the unwinder data (12a) and examples from the automaton states (12b, 12c, 12d).

Table 1. Scores for different datasets and methods. Explanations can be found in the corresponding sections.

Dataset	Method	TP	TN	FP	FN	ACC	F1
4.1: Artificial	NNPCA	0	991	0	1692	36.94%	0%
	Automaton, NNPCA only	1568	990	1	124	95.34%	96.17%
	Automaton, all errors	1621	980	11	71	96.94%	97.53%
4.2: HRSS	NNPCA	35	15130	0	5239	74.32%	1.32%
	Automaton, NNPCA only	1636	13503	1627	3638	74.20%	38.33%
	Automaton, all errors	1794	13247	1883	3480	73.72%	40.08%
4.3: Unwinder	NNPCA	2528	40926	34	22048	66.31%	18.63%
	Automaton, NNPCA only	11340	37931	3029	13236	75.18%	58.24%
	Automaton, all errors	11864	37851	3109	12712	75.86%	60.00%

timed automata. Hybrid automata are used for anomaly detection in the real-valued signal values as well as in the time domain by analysing the transitions between the states of the automaton.

The presented approach derives the different stationary and transient process phases based on a system's real-valued signals through the use of self-organizing maps and watershed transformations. Also, all necessary parameters for the SOM and watershed transformation are automatically estimated from data.

The discrete events generated from the transitions between the extracted process phases are used to learn a hybrid timed automaton which in turn is used for anomaly detection. The presented algorithms work offline during the learning phase and can later be used online with live data from the plant. Further, the presented algorithms are not linked to a specific automaton learning algorithm and can be used as a preprocessing step prior to automaton learning.

The experiments in section 4 show results from three different datasets, two of which come from real-world machines. The anomaly detection for real-valued signals through a model within the states of the learned hybrid automaton increases the performance of the anomaly detection in comparison to the same model working with the same parameters on the full dataset. For all tested datasets, the separation of modes through the automaton improves the scores for the anomaly detection of the real-valued signals, without any additional tuning of parameters for the anomaly detection. Furthermore, models such as the aforementioned NNPCA or SOM's do not model time in an explicit manner. The hybrid automaton adds the explicit modelling of time for transitions between a system's modes and consequently enables the detection of anomalies which were not detectable before.

Acknowledgments. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 678867.

References

1. Factories of the Future: MultiAnnual Roadmap for the contractual PPP under HORIZON 2020. European Union, Luxembourg (2013)
2. 3S-Smart Software Solutions GmbH: Codesys softmotion: Integrierte bewegungssteuerung in einem iec 61131-3 programmiersystem (2017), https://de.codesys.com/produkte/codesys-motion-cnc/softmotion.html
3. Alhoniemi, E., Hollmn, J., Simula, O., Vesanto, J.: Process monitoring and modeling using the self-organizing map. Integrated Computer Aided Engineering 6, 3–14 (1999), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.573
4. Eickmeyer, J., Krueger, T., Frischkorn, A., Hoppe, T., Li, P., Pethig, F., Schriegel, S., Niggemann, O.: Intelligente zustandsberwachung von windenergieanlagen als cloudservice. In: Automation 2015. Baden-Baden (Jun 2015)
5. Eickmeyer, J., Li, P., Givehchi, O., Pethig, F., Niggemann, O.: Data driven modeling for system-level condition monitoring on wind power plants. In: Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015) co-located with 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (Safeprocess 2015), Paris, France, August 31 - September 3, 2015. pp. 43–50 (2015)
6. Frey, C.: Monitoring of complex industrial processes based on self-organizing maps and watershed transformations. In: Industrial Technology (ICIT), 2012 IEEE International Conference on (Mar 2012)
7. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings 11th Annual IEEE Symposium on Logic in Computer Science. pp. 278–292 (Jul 1996)
8. Liukkonen, M., Hiltunen, Y., Laakso, I.: Advanced monitoring and diagnosis of industrial processes. In: 2013 8th EUROSIM Congress on Modelling and Simulation. pp. 112–117 (Sept 2013)
9. Maier, A.: Online passive learning of timed automata for cyber-physical production systems. In: 12th IEEE International Conference on Industrial Informatics (INDIN). pp. 60–66 (July 2014)
10. Maier, A.: Identification of timed behavior models for diagnosis in production systems. Ph.D. thesis, Paderborn, Univ. (2015)
11. Maier, A., Niggemann, O.: On the learning of timing behavior for anomaly detection in cyber-physical production systems. In: International Workshop on the Principles of Diagnosis (DX). Paris, France (Aug 2015)
12. Meyer, F.: Topographic distance and watershed lines. Signal Processing 38(1), 113–125 (jul 1994), http://dx.doi.org/10.1016/0165-1684(94)90060-4
13. Niggemann, O., Maier, A., Just, R., Jäger, M.: Anomaly detection in production plants using timed automata : Automated learning of models from observations. In: Proceedings of the 8th International Conference on Informatics in Control, Automation and Robotics. pp. 363 – 369. No. 1 (2013)
14. Niggemann, O., Maier, A., Vodencarevic, A., Jantscher, B.: Fighting the modeling bottleneck - learning models for production plants. Workshop "Modellbasierte Entwicklung Eingebetteter Systeme" (MBEES) (7 2011)
15. OCME: Shrink-wrap packers vega (Mar 2017), http://www.ocme.com/en/our-solutions/secondary-packaging/vega
16. Simula, O., Kangas, J.: Process monitoring and visualisation using self-organizing maps (1995)
17. Tian, J., Azarian, M.H., Pecht, M.: Anomaly detection using self-organizing maps-based k-nearest neighbor algorithm. Second European Conference of the Prognostics and Health Management Society 2014 (2014)

18. Ultsch, A., Ltsch, J.: Machine-learned cluster identification in high-dimensional data. Journal of Biomedical Informatics 66, 95 – 104 (2017), http://www.sciencedirect.com/science/article/pii/S153204641630185X
19. Ultsch, A., Siemon, H.P.: Kohonen's self-organizing feature maps for exploratory data analysis. In: Proceedings of the International Neural Network Conference (INNC'90 (1990), http://www.uni-marburg.de/fb12/datenbionik/pdf/pubs/1990/UltschSiemon90
20. Vincent, L., Soille, P.: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence 13(6), 583–598 (Jun 1991)
21. Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S.and Wollschlaeger, M., Goehner, P.: Challenges for software engineering in automation. Journal of Software Engineering and Applications 7(5) (May 2014), http://dx.doi.org/10.4236/jsea.2014.75041
22. Yin, H.: The Self-Organizing Maps: Background, Theories, Extensions and Applications, pp. 715–762. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)