Journal of Machine Engineering, 2021, Vol. 21, No. 2, 35–46
ISSN 1895-7595 (Print) ISSN 2391-8071 (Online)

Received: *13. October 2020* / Accepted: *30 April 2021* / Published online: *10 June 2021*

condition monitoring,
clustering tracking,
unsupervised learning

Jonas HILLENBRAND[1*]
Jürgen FLEISCHER[1]

UNSUPERVISED DETECTION OF STATE CHANGES DURING OPERATION OF MACHINE ELEMENTS

Interpretation of sensor data from machine elements is challenging, if no prior knowledge of the system is available. Evaluation methods must adapt surrounding conditions and operation modes. As supervised learning models can be time-consuming to set up, unsupervised learning poses as alternative solution. This paper introduces a new clustering scheme that incorporates iterative cluster retrieval in order to track the clustering results over time. The approach is used to identify changing machine element states such as operating conditions and undesired changes, like incipient damage or wear. We show that knowledge about the evolving clusters can be used to identify operation and failure events. The approach is validated for machine elements with slide and roll contacts, such as ball screws and bearings. The data used has been captured using vibration and acoustic emission sensors. The results show a general applicability to the unsupervised monitoring of machine elements using the proposed approach.

1. INTRODUCTION

Growing automation and digitalization in modern factories increase the importance of capital as a production factor for companies. Large investments in new, more capable machinery rather than human work force raise the necessity for appropriate maintenance solutions. In this context, maintenance must meet growing requirements in terms of availability, reliability, and flexibility of machinery or facilities [1, 2]. Achieving greater autonomy and self-reliance requires that these systems function without human intervention and decision making. Especially in setups where no prior knowledge of the system and historical data exists, supervised learning can be too time-consuming. Here, unsupervised learning poses an alternative solution to discover and structure useful information from uncharted data [3].

Real manufacturing equipment is subject to complex operation modes and, hence, data streams of installed sensor systems are subject to the related dynamic changes. There is a need of methods to recognize these changes and patterns continuously. State-of-the-art anomaly detection systems only identify these changes in constrained setups. When confronted with

[1] wbk Institute of Production Science, Karlsruhe Institute of Technology, Karlsruhe, Germany
[*] E-mail: jonas.hillenbrand@kit.edu
https://doi.org/10.36897/jme/136311

healthy and faulty modes superposed with different operation regimes, they fail to distinguish between anomalies and operation modes [4].

We address this issue introducing a new cluster model for tracking cluster events over time. These events can then be used to assess a system, machine, or component in a more reliable fashion although operating conditions are unknown and superposed by failure modes.

This cluster model is designed to be memory-efficient, stores historic cluster states, and enables analysis of temporal cluster evolution. Furthermore, we discuss cluster transition events of the state of the art in cluster tracking to further their use in a condition monitoring (CM) perspective.

2. STATE OF THE ART IN DATA STREAM CLUSTERING AND CLUSTER TRACKING APPLICATIONS

The term *data stream* describes a sequence of data items $x_1, \dots, x_i, \dots, x_n$ such that these items arrive or are being read in an increasing order of the index i, where $n \to \infty$ [5]. In the context of CM, data streams may arise from vibration sensors, measurements of motor current from feed axes, or temperature readings of machinery. With ever-increasing data streams, it is paramount to process data in an efficient way and store representative structures in a memory-efficient way. Common storage principles for representative data structures in clustering applications are feature vectors, originally introduced in [6], prototype arrays, and grids or trees. A summary of these structures can be found in [7].

Detecting state changes during the operation of mechanical systems that are exposed to natural wear is a popularly discussed field in anomaly detection and CM applications. A common technique for transition detection between a healthy and a degradation mode is the use of state-space model estimators as provided in [8]. These models are used to predict future states based on an estimated state vector learned from initial data. Although, if the underlying system changes due to wear, the model does not represent these changes anymore. Real-world systems are usually subject to such non-linearities and the missing causality between input data and output data due to degradation effects.

In contrast, data stream cluster algorithms and tracking of cluster evolution provide mechanisms to observe and follow these changes without supervised training. There are various techniques for data stream clustering, presented in a survey by [7]. Areas of interest are geographic data to observe forest coverage, grid computing, and network intrusion detection. Other data streams that have been addressed stem from stock market analysis, voice-over-IP data, sensor networks in the civil engineering context, social network analysis, and text data streams. One of the most used data stream algorithms based on the literature review is *CluStream* [9]. The algorithm enables processing of continuous data streams in a micro- and macro-clustering phase while maintaining so-called snapshots, i.e. historic cluster states, in a memory-efficient way. A drawback of this algorithm is the use of the clustering method k-means, which is unable to discover clusters of arbitrary shape and requires a-priori knowledge about the expected number of clusters k. This was addressed by [10] who introduced a hierarchical clustering scheme called *ClusTree*. The algorithm is hyperpara-

meter-free and enables the detection of changing data, novelty, and outlier detection on the stream. But still, both algorithms abstract the raw data in the form of micro clusters consisting of cluster feature vectors [7]. In the case of clustering objects from sensor data, such as extracted vibration features, using feature vectors results in a twofold dimensionality reduction, which leads to further loss of information. An alternative approach using grid-based clustering (*D-Stream*) is introduced in [11]. The approach does not require definition of the number of clusters a-priori, but introduces new hyperparameters in order to choose the grid size. Furthermore, the mentioned data stream algorithms operate with the concept of discarding old cluster objects, usually based on decay times [7], to avoid excessive memory requirements. This concept is in conflict with the goal of detecting evolving cluster structures over time. A recent approach from [12] solves this issue by introducing a two-level architecture with an online component, which clusters the stream data, and an offline component, which saves recurring cluster structures (called *concepts*) for later use. A drawback stems from the lack of support for arbitrarily shaped clusters, which decreases the usability in applications where no prior information about shape of clusters and data distribution is known.

In the context of CM, data stream clustering is discussed with regard to bearing prognostics [13]. The authors introduce a new data stream clustering method using belief function theory. They show that features extracted from vibration measurements correlate with the change from healthy to degradation modes during life cycle tests of bearings as new clusters develop. However, their approach does not cover mechanisms to identify these changes over time, but rather delivers them as output for further evaluation. Here, cluster tracking provides mechanisms to automatically uncover these changes over time.

Knowledge about the cluster evolution can be used to interpret a system's state and the changes it undergoes. A fundamental and algorithm-independent framework called *MONIC, modelling and monitoring cluster transitions*, has been introduced by [14]. The authors introduced definitions for external and internal transitions of a cluster. These transitions and their interpretation are then discussed for document and text mining tasks on the ACM digital library. The framework claims general applicability on any clustering problem and cluster algorithm but has not been applied to CM or anomaly detection setups yet. A more recent contribution is *ChronoClust*, a novel density-based clustering algorithm for processing time-series data and tracking its temporal evolution [15]. Even though the approach does not explicitly cover data stream clustering, it involves tracking mechanisms that address the time-sensitive supervision of cluster evolution. It was applied on biological cytometry data.

Regarding CM applications, a memory or storage problem arises that is not covered by common data stream clustering problems. As the time scale of a system's degradation (e.g. weeks, months, or even years) greatly differs from the sampling periods of the used sensors (ranging from a few Hz to several MHz), data must be compressed and analysed continuously. Otherwise, historic features extracted from the data stream must be discarded. Still, features extracted from a sensor data stream a week ago can be crucial for evaluation of current machine states concerning gradual wear effects. This work introduces a new data structure and cluster mechanism for describing evolving cluster states. The approach aims at achieving a balance between in-memory cluster history, memory efficiency, and execution speed in

terms of infinite data streams. Our proposed cluster model relies on detected cluster boundary points (hence *ClusterBoundaryTracking*) and applies and adapts the cluster tracking transitions from [7, 14]. Moreover, these general cluster transitions are related to system state changes in a CM context, such as detection of operation modes, but also occurrence of wear or degradation. The cluster model and tracking scheme is then evaluated on simulated data streams of bearing and feed axis experiments.

3. CLUSTER BOUNDARY TRACKING APPROACH

In this article, the terms *cluster state* and *cluster transition* are used to describe the results of the cluster algorithm and the change between different cluster results from one point in time to the next and from one cluster iteration to another. Within the clustering step, an adequate clustering algorithm is used to generate *cluster states*. As the number of clustering algorithms is vast and a lot of research has emerged in recent years, we decided to take this into account by defining a generalized interface for clustering algorithms in our MATLAB setup. On top of the clustering method, which is then interchangeable, we implemented our *ClusterBoundaryTracking* cluster tracking scheme.

To enable parametrization of the methods, the interface defines a *hyperparameters* property to store any kind of configuration parameters for a specific algorithm, e.g. *DBSCAN* contains the *Epsilon* and *MinPts* hyperparameters. See [16] for more details on *DBSCAN*. Furthermore, the interface defines two methods referred to as *cluster* and *estimate-Hyperparameters*. The method *cluster* receives the datapoints to cluster as input and delivers the *clusterIndices* that store the information as to which datapoint belongs to which cluster, and the number of resulting clusters *kCluster*. Whereas *estimateHyperparameters* can be used to continuously improve the clustering result through repeated estimation of the used hyperparameters. In the context of this paper, DBSCAN was used and its parameters estimated according to the combined procedures in [16, 17].

3.1. CLUSTER MODEL

In order to track the temporal evolution of clusters, two data structures are defined: *ClusterState* and *ClusterTransition*. The data structure *ClusterState* is used to describe the geometry of the gained cluster result, a central aspect being the representation of clusters via its border points, while other data points in the clustered group can be removed from memory. With growing cluster size, the border points do not increase as strongly as the number of total data points. Even though it depends on the dataset, a qualitive development of cluster and border points is shown in Fig. 1 (on the left).

Whenever new data arrives, a new *clusterState* is stored. These saved *clusterStates* can later be used for further analysis. Outlier points are also kept in the model, as these points can evolve from outlier to cluster points over time.

The overall ClusterState structure is shown as UML class diagram in Fig. 2b. The properties marked in green are the ones that are kept in memory, yellow properties can

be calculated on demand, and grey ones will be discarded as soon as they are not needed anymore. Outliers are not explicitly mentioned as a separate data property, but their association with their corresponding data points is contained within *clusterIndices* (index −1 as separate cluster group).

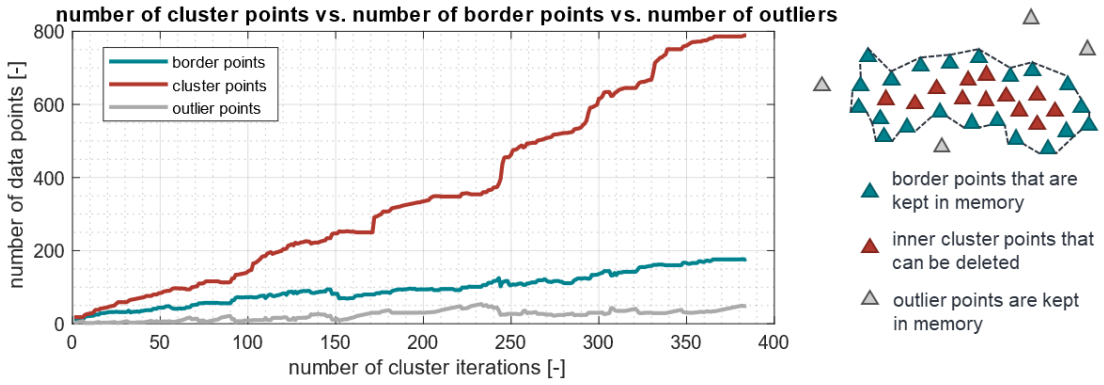

Fig. 1. Number of cluster points vs. border points during clustering (left), visualization of border and cluster points (right)

The definitions of possible cluster transitions based on findings in [14, 15] are compiled in Table 1. The *cluster transitions* can be divided into three categories: internal, external, and global. Internal *cluster transitions* describe the shape or data distribution of a single cluster group, whereas external transitions consist of the interaction between cluster groups (two or more). The global category designates transition types that refer to the entire dataset and/or are related to changes in the clustering method, e.g. change of hyperparameters.

Table 1. Possible cluster transition types and their condition monitoring perspective

Transition Type	Category	Description	Condition Monitoring Perspective
Compactness Transition	Internal	A compactness transition occurs if the contemplated cluster becomes compacter or diffuser compared to a previous state. The standard deviation can be used as a measure for compactness.	If the compactness increases and the cluster size stays the same, the system operates in a stationary mode.
Size Transition	Internal	A size transition occurs when the cluster shrinks or expands. A measure for the cluster size is the sum of all data points within the cluster.	If the cluster size grows, it can be due to process/operation instability.
Location Transition	Internal	A location transition occurs when the center or the distribution of the cluster shifts.	A cluster center moving away from its initial position can correspond to a sensor drift or slow wear process.
Creation Transition	External	A new cluster is created by the cluster method.	Announces a new operation mode or rapid degradation / failure.
Merge Transition	External	Two previously existing clusters are merged into one.	Two operation modes may exist between which it is shifted.
Vanish Transition	External	A previously existing cluster disappears.	Applies only when cluster objects are modelled with decay or forget mechanisms, which does not apply to this work‘s implementation
Split Transition	External	A previously existing cluster is split into two clusters.	Applies only when cluster objects are modelled with decay or forget mechanisms, which does not apply to this work‘s implementation

Survive Transition	External	The cluster exists in the previous and the current cluster states.	The system is operated stationarily and shows no degradation or wear.
Leaving Cluster Transition	External	If a new incoming data point leaves the cluster boundaries which the latest data point belonged to.	The system returns to a state it was already in. The involved clusters may be two reversible operation modes.
Outlier Transition	Global	If outliers exceed a relative threshold (e.g. 10%) transitioning from one cluster state to the next.	Announces a new operation mode or sudden degradation / failure. This can be a sign for incipient faults.
hyperparameter Transition	Global	If hyperparameters change by a relative threshold from one state to the next due to estimation of automatic hyperparameters.	If new hyperparameter values are estimated, a global change has occurred, e.g. a new system state, that delivers completely different sensor readings.
Number of Clusters Transition	Global	This transition only checks the change in cluster number from one cluster state to the next.	If the number of clusters increases, see CreationTransition or SplitTransition. If the number of clusters decreases, see MergeTransition or VanishTransition.
Indices Change Transition	Global	This transition can be computed very fast by comparing the new cluster indices with the latest.	If cluster indices change from state to state, any of the above may apply.

The proposed concept allows to find structural changes in the cluster result. These structural changes arise from the underlying data distribution, hence, the underlying observed system. Here, the term system describes a machine or component that is being monitored by sensors. From a CM perspective, the clustering results and historic changes relate to changing operating conditions, natural wear, or degradation. For the purpose of this work, the assumptions displayed in Table are made. This is an attempt to create explainable clustering results, when no prior knowledge exists and supervised learning is not possible.

3.2. CLUSTER TRACKING METHOD

During monitoring of a system, new cluster states are continuously created for incoming data. The course of processing is depicted in the flow chart in Fig. 2a. Starting with capturing of new data and possible transformations, such as feature extraction, Fourier transformation, or filtering, new data points are checked for clustering necessity (requiresClustering). The function verifies whether the new data points are already within existing cluster boundaries. In that case, the data points are appended to the recent ClusterState. Otherwise, a new ClusterState is created. In order to determine a datapoint within cluster boundaries, we approximate the cluster as a polyshape via its boundary points. For 2D (polygon) and 3D (polyhedron) datasets, we used MATLAB functions [18, 19], which check whether a point is within a triangulated area or volume. These geometric computations are faster than executing the clustering algorithm, lowering the execution time of the method compared to the DBSCAN run each iteration.

Based on this ClusterState, the cluster model data is scanned for ClusterTransitions within the function trackClusters whose procedure is depicted with pseudo code in Table 2. This function searches the newly created ClusterState for the cluster transitions defined in Table 1. After that, the newly created ClusterState and found ClusterTransitions are added to the model, changes are reported (reportClusterChanges), if asked for, and the next iteration of captured data can be processed. All acquired clusterStates and corresponding cluster-Transitions are stored in the cluster model and can be accessed for further analysis. The cluster model and the required functions are implemented in the class ClusterBoundaryTracking.

Table 2. Pseudo code for trackCluster routine

Routine: trackCluster	
	cs_i, cs_{i-1} denote the current and previous *clusterState* and τ the threshold for creating a transition
	→: denotes the creation of a transition object
1.	**if** $cs_i.kClusters \neq cs_{i-1}.kClusters$ → *NumberOfClustersTransition*
2.	**if** $\boldsymbol{cs_i.kClusters < cs_{i-1}.kClusters}$ → **MergeTransition**
3.	**if** $\boldsymbol{cs_i.kClusters > cs_{i-1}.kClusters}$ → **CreationTransition**
4.	**else**
5.	**if** $\frac{\lvert cs_i.densities - cs_{i-1}.densities\rvert}{cs_{i-1}.densities} > \tau$ → **CompactnessTransition**
6.	**if** $\frac{\lvert cs_i.sizes - cs_{i-1}.sizes\rvert}{cs_{i-1}.sizes} > \tau$ →**SizeTransition**
7.	**if** $cs_i.clusterIndices \neq cs_{i-1}.clusterIndices$ → IndicesChangeTransition
8.	**foreach** hp_i **in** $cs_i.hyperparameters$ **and** hp_{i-1} **in** $cs_{i-1}.hyperparameters$
9.	**if** $\left\lvert 1 - \frac{hp_i}{hp_{i-1}}\right\rvert > \tau$ → HyperparameterChangeTransition
10.	$o_i \leftarrow cs_i$.getOutliers **and** $o_{i-1} \leftarrow cs_i$.getOutliers
11.	**if** $\left\lvert 1 - \frac{o_i}{o_{i-1}}\right\rvert > \tau$ → OutlierChangeTransition

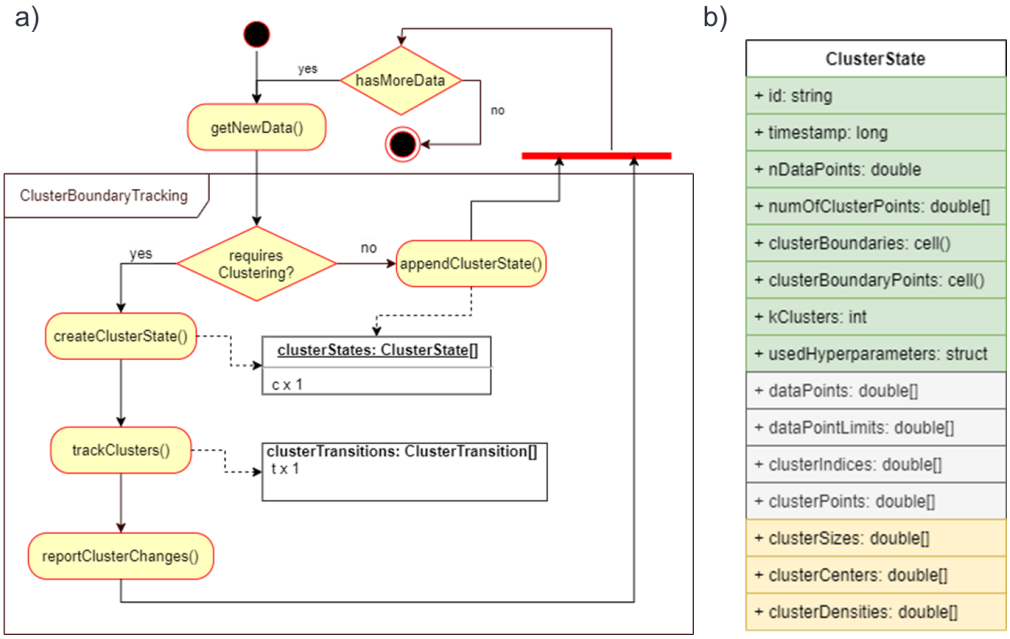

Fig. 2. Processing flow chart for cluster tracking (a), data structure ClusterState (b)

4. EXPERIMENTAL RESULTS AND DISCUSSION

In order to evaluate the performance of the proposed approach, two datasets created at our institute have been used. The first dataset stems from a lifecycle experiment conducted on a test bench for ball screw drives [20]. The data consists of continuous motor current, axis position, shaft speed, force, and temperature measurements. During the experiment, two ball

screw nuts were clamped with tension rods on one spindle, as displayed in Fig. 3 (for more information on the test bench setup, see [21]). The mechanical setup allows for high loads to be applied on the ball screw and executes cyclic strokes until the component's failure. The lifecycle test consists of three normal phases, where different loads have been applied and a last phase, where failure due to breakage of a raceway groove in the ball screw nut occurred.

The second dataset consists of acoustic emission measurements of an axial ball bearing at different shaft speeds [22]. The setup depicted in Fig 3b consists of two separate parts between which an axial bearing can be clamped. The upper part provides a controlled load via a pneumatic cylinder, whereas the lower part has a rotational degree of motion and supplies the driving torque via a stepper motor. The experiments conducted on the test bench include acoustic emission measurements at different shaft speeds.

We evaluate the performance of our approach in terms of the correct detection of states which the systems undergo based on the corresponding measurement values (here: motor current and acoustic emission). In our case, we shall detect *cluster transitions* between different load states and shaft speeds.

Both datasets have been captured in advance of the development of this approach. Thus, they are replayed via a data stream simulation in MATLAB. For the ball screw dataset, we used temperature ($T_{BS,peak}$) and motor current ($I_{Motor,peak}$) peak values as features for the clustering algorithm.

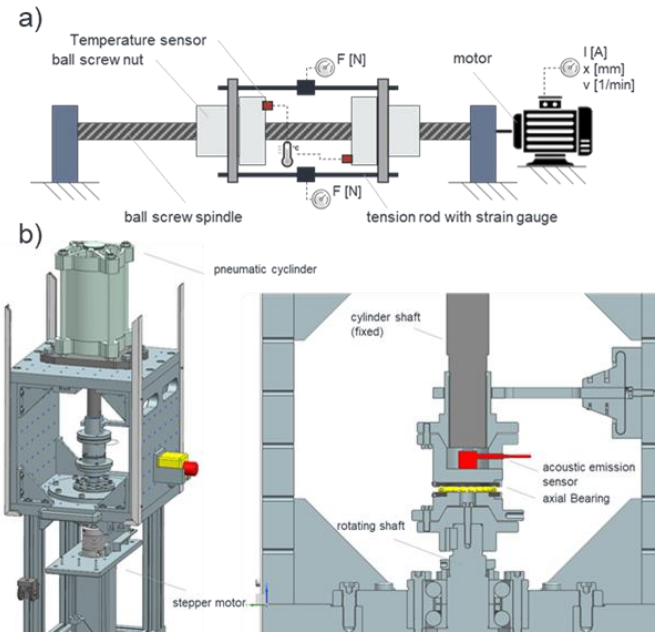

Fig. 3. Mechanical setup for ball screw drive (a) and axial ball bearing (b)

The complete experimental data sums up to 2302×2 samples and was tracked over a period of 28 days. These features have been continuously extracted from the data stream during an experiment, while applying increasing loads on the nut via tension rods until the component's failure. Fig. 4 shows the final clustering result for the given data points.

The algorithm finds consistent clusters to the ground truth, but is not able to associate class 2 correctly into one cluster. Datapoints from class 4, representing the ball screw failure, are only recognized as outliers due to having a lower density than the average of the other clusters. The increasing number of outliers, however, is properly detected by *trackClusters* when data points of class 4 arrive and can be interpreted as anomaly, as mentioned in Table 1.

The dataset of the axial ball bearing consists of acoustic emission measurements during different shaft speeds of the test bench. The features selected for clustering are the root-mean-square and peak-to-peak values of the raw acoustic emission data. The test series includes 600×2 samples. The clustering depicted in Fig. 5 represents the final cluster result after transitioning through different speed states. The upper graph shows the clustering result and the lower graph the corresponding ground truth. Concerning cluster purity, the clustering algorithm failed to separate *class1* and *class2*, two close shaft speeds (150 1/min and 200 1/min), due to varying densities in this cluster region and the close vicinity of cluster points. All other clusters coincide with their classes.

Both clustering results showed deviations from the ground truth, where clusters with different average densities developed. Here, OPTICS [23] poses an alternative to conventional density-based clustering algorithms.

Also, we can show that each cluster creation is correctly reported during *trackClusters*. Tracking of size and density changes are identified as well, but are currently implemented with a static threshold that reports proportionate changes from one state to the next. Slowly increasing clusters are therefore not recognized.

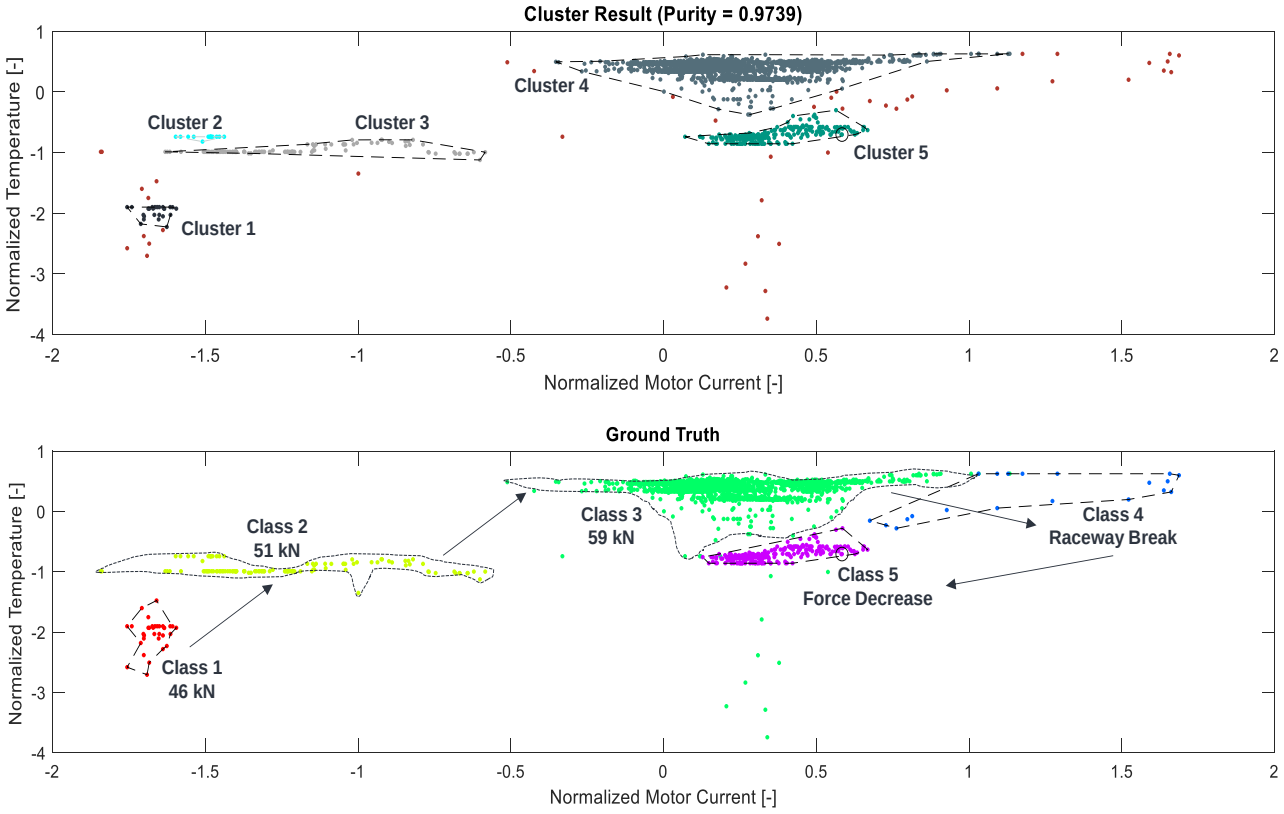

Fig. 4. Clustering result for ball screw dataset

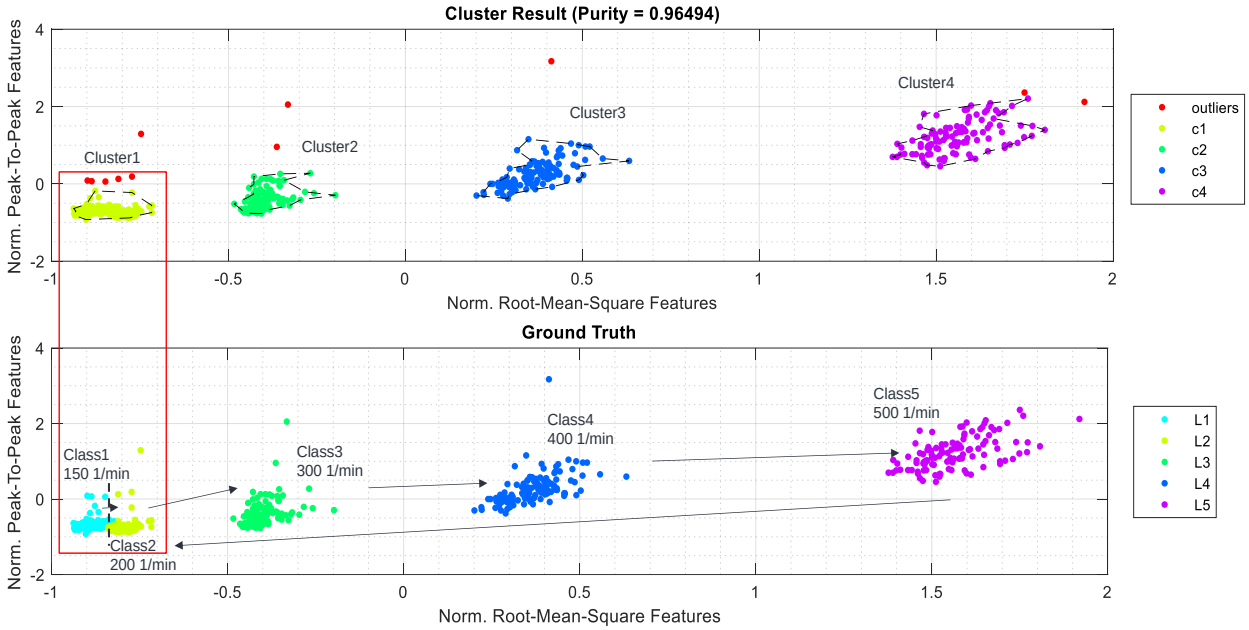

Fig. 5. Clustering result for axial ball bearing dataset

In terms of time efficiency, we can show that our approach saves computing time by applying the function *requiresClustering* before each new cluster iteration for new incoming data. Hence, the number of required cluster iterations can be reduced and therefore up to 50% of computing time can be saved compared to clustering each iteration for the tested datasets.

Figure 6 shows the performance in terms of required cluster iterations and consumed time for both datasets. For the computations, we used MATLAB on a common Desktop PC with an Intel Core i7-4790 processor (@ 3.60GHz, 4 cores). The computation time depicted in the figure represents the total amount of time elapsed to iteratively cluster the whole dataset as data points arrive one-by-one.

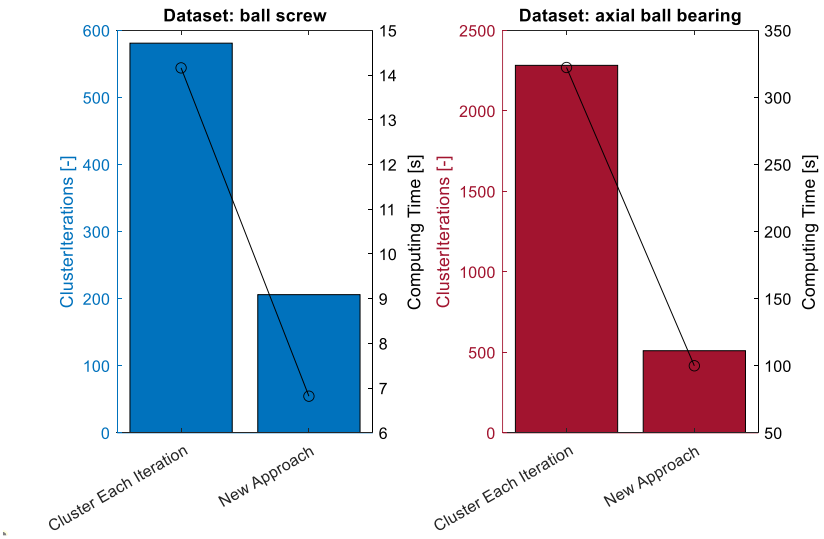

Fig. 6. Time performance and cluster iterations for new approach and conventional clustering

Using DBSCAN as clustering method, we reached cluster purities greater than 96% on both datasets. A major advantage of this clustering algorithm is its capability to define outliers without additional logic, which then can be tracked using *trackClusters.* Besides, DBSCAN does not require knowledge about the prevailing number of clusters, and its hyperparameters can be estimated properly using the data distribution. These features enable well-clustered datapoints without any prior knowledge. Performance drawbacks besides the already mentioned insensitivity to clusters with varying density have yet to be investigated. In general, using the procedure provided in Fig. 2 saves computing time by reducing the number of cluster method executions, leaving more time for other operations, such as the tracking step.

5. CONCLUSION AND OUTLOOK

We introduced a new cluster model to track cluster changes in the context of CM. The concept introduced allows for effortless integration of other clustering algorithms that operate on numerical feature spaces, such as DBSCAN [16]. Additionally, we provided an interpretation in terms of *cluster transitions* for a CM perspective, presented in Table 1. So far, we implemented basic *cluster transitions* to show the feasibilty of the tracking mechanism. More complex transitions, such as *LeavingClusterTransition* or *Location-Transitions*, have yet to be implemented. The current version of *ClusterBoundaryTracking* is conceived for 1D, 2D, or 3D data. In the case of data with more dimensions, a new generalized concept for high-dimensional cluster boundaries has to be conceived. This deficiency currently disqualifies *ClusterBoundaryTracking* for datasets with more than three dimensions and nonnumeric properties.

In further works, we will address the implementation of further *cluster transitions* and specifically concentrate on their CM interpretation. The abovementioned *LeavingCluster-Transition* and a *ReturnToClusterTransition* will be considered for further investigation. If newly clustered data points leave the previous cluster or return to an older cluster, our tracking approach shall assume that the observed system migrates to a new state or returns to an old state. Tracking these transitions enables the monitoring application to determine system states that have never been defined a priori but learned during the operation. This hypothesis will require further testing of the approach. We will therefore extend the used datasets and conduct other life cycle experiments.

So far, the clustering algorithm (DBSCAN) is run each iteration for all data points. This still imposes the greatest time consumption. In order to further reduce computing time, we plan on implementing incremental DBSCAN [24]. This algorithm does not cluster the whole dataset each time a new clustering is required. Instead, it only extends the existing clusters based on the new data points' vicinity to existing data points.

ACKNOWLEDGEMENTS

This research work is funded by DFG - Deutsche Forschungsgemeinschaft (German Research Foundation) – Project 388141462.

REFERENCES

[1] MÄRZ M., 2017, *Maintenance 4.0 Bestimmt Profitabilität der Fabrik von Morgen: White Paper zu Predictive Maintenance*, Mobile Instandhaltung und Asset Innovation, VDI-Z Integrierte Produktion, 159, 52–53.

[2] UHLMANN E., HOHWIELER E., GEISERT C., 2017, *Intelligent Production Systems in the Era of Industry 4.0*: *Changing Mindsets and Business Models*, Journal of Machine Engineering, 17/2, 5–24.

[3] CELEBI M.E., AYDIN K., 2016, *Unsupervised Learning Algorithms*, Springer.

[4] SAARI J., ODELIUS J., 2018, *Detecting Operation Regimes Using Unsupervised Clustering with Infected Group Labelling to Improve Machine Diagnostics and Prognostics*, Operations Research Perspectives, 5, https://doi.org/10.1016/j.orp.2018.08.002, 232–244.

[5] HENZINGER P., RAGHAVAN S., RAJAGOPALAN M.R., 1998, *Computing on Data Streams*, SRC Technical Note, 011, https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=A341F8AD36C78BF4B067C0A3456EA10F?, doi=10.1.1.19.9554&rep=rep1&type=pdf.

6] ZHANG T., RAMAKRISHNAN R., LIVNY M., 1996, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, SIGMOD Rec., 25, https://doi.org/10.1145/235968.233324, 103–114.

[7] SILVA J.A., FARIA E.R., BARROS R.C., HRUSCHKA E.R., DE CARVALHO A.C.P.L.F. GAMA J., 2013, *Data Stream Clustering*, ACM Comput. Surv., 46, https://doi.org/10.1145/2522968.2522981, 1–31.

[8] MathWorks, 2021, *Condition Monitoring and Prognostics Using Vibration Signals*, https://www.mathworks.com/help/predmaint/ug/condition-monitoring-and-prognostics-using-vibration-signals.html.

[9] AGGARWAL C.C., YU P.S., HAN J., WANG J., 2003, *A Framework for Clustering Evolving Data Streams*, Proceedings of the Twenty-Ninth International Conference on Very Large Databases, Berlin, Germany, 9–12 Morgan Kaufmann Publishers/Elsevier Science, St Louis, MO, 81–92.

[10] KRANEN P., ASSENT I., BALDAUF C., SEIDL T., 2009, *Self-Adaptive Anytime Stream Clustering*, 2009, Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, IEEE, 249–258.

[11] CHEN Y., TU L., 2007, *Density-Based Clustering for Real-Time Stream Data*, Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, USA.

[12] NAMITHA K., SANTHOSH KUMAR G., 2020, *Learning in the Presence of Concept Recurrence in Data Stream Clustering*, J. Big Data, 7/1, 1–28, https://doi.org/10.1186/s40537-020-00354-1.

[13] SERIR L., RAMASSO E., ZERHOUNI N., 2012, *Evidential Evolving Gustafson–Kessel Algorithm for Online Data Streams Partitioning Using Belief Function Theory*, International Journal of Approximate Reasoning, 53/5, 747–768, https://doi.org/10.1016/j.ijar.2012.01.009.

[14] SPILIOPOULOU M., NTOUTSI I., THEODORIDIS Y., SCHULT R., 2006, *MONIC: Modeling and Monitoring Cluster Transitions*, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – KDD-06, Philadelphia, PA, USA, ACM Press, New York, USA, 706–711.

[15] PUTRG G.H., READ M.N., KOPRINSKA I., SINGH D., RÖHM U., ASHHURST T.M., KING N.J., 2019, *ChronoClust: Density-Based Clustering and Cluster Tracking in High-Dimensional Time-Series Data, Knowledge-Based Systems*, 174, 9–26, https://doi.org/10.1016/j.knosys.2019.02.018.

[16] ESTER M., KRIEGEL H.-P., SANDER J., XU X., 1996, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, AAAI (Hg.) KDD-96 Proceedings, 226–231.

[17] SCHUBERT E., SANDER J., ESTER M., KRIEGEL H.P., XU X., 2017, *DBSCAN Revisited*, ACM Trans. Database Syst., 42, 1–21, https://doi.org/10.1145/3068335.

[18] SVEN, 2021, *Inpolyhedron – are points inside a triangulated volume?* MATLAB Central File Exchange, https://www.mathworks.com/matlabcentral/fileexchange/37856-inpolyhedron-are-points-inside-a-triangulated-volume.

[19] MathWorks Inpolygon, 2006, *Points Located Inside or on Edge of Polygonal Region*, https://de.mathworks.com-/help/matlab/ref/inpolygon.html.

[20] HILLENBRAND J., 2020, *Ball Screw Failure – Dataset*: v1, https://git.scc.kit.edu/ml-wzmm_public/ballscrewloadfailure_v1.

[21] HILLENBRAND J., SPOHRER A., FLEISCHER J., 2018, *Zustandsüberwachung bei Kugelgewindetrieben: Integration von DMS-Sensorik in Kugelgewindetriebemuttern*, wt Werkstattstechnik online, 8, 493.

[22] HILLENBRAND J., 2020, *Axial Ball Bearing Speeds v1*, ML-WZMM_Public., https://git.scc.kit.edu/mlwzmmpublic/Axial_Ball_Bearing_Speeds_v1.

[23] ANKERST M., BREUNIG M.M., KRIEGEL H.-P., SANDER J., 1999, *OPTICS: Ordering Points to Identify the Clustering Structure*, SIGMOD Rec., 28,49–60, https://doi.org/10.1145/304181.304187.

[24] ESTER M., KRIEGEL H.-P., SANDER J., WIMMER M., XU X., 1998, *Incremental Clustering for Mining in a Data Warehousing Environment*, Proceedings of the 24th VLDB Conference New York, USA.