2021 3rd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE) | 978-1-6654-4966-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICECIE52348.2021.9664675

Machine Failure Diagnosis by Combining Software Log and Sensor Data

Takako Onishi
Industrial Automation Company
OMRON Corporation
Shiga, Japan
takako.onishi@omron.com

Hisashi Kashima
Graduate School of Informatics
Kyoto University
Kyoto, Japan
kashima@i.kyoto-u.ac.jp

***Abstract*—Many studies have been conducted in the manufacturing industry to support the cause analysis and early recovery of production line shutdowns caused by machine failures. However, methods such as simple anomaly detection are not effective against large machines with complex behavior. In this study, we propose a method for such machines to show the estimated causes of failure by combining log text files and sensor data, which record software behavior and hardware status, respectively. The proposed method is twice as accurate as methods with only software logs or sensor data, and achieves explainability of the results.**

Keywords—Failure Diagnosis, Log Analysis, Anomaly Detection, Machine Maintenance

I. Introduction

In recent years, factory automation has rapidly advanced in the manufacturing industry. In highly automated production lines, the failure of a single machine has a significant impact on the production efficiency of the entire line. Therefore, preventive maintenance and measures for early recovery are important.

In response to the rise of AI and the trend of Industry 4.0, efforts are being made to accumulate data on the status of machines and use them to automate failure countermeasures. Particularly, for machines that repeat certain simple motions, it is effective to use methods such as detecting outliers or change points in the signal data of motors or other devices. Such technologies are being implemented to estimate the causes of failures and predictive failure detection [1]–[8].

However, for machines that perform complex behavior with built-in control devices or PCs, such as processing machines or inspection machines, simple conventional technologies provide insufficient results.

To address this problem, studies have been conducted to collect time-series numeric data from numerous sensors installed in machines or systems, and to utilize it with technologies such as deep learning [9]–[17].

Additionally, studies have been conducted to utilize text data instead of sensor data. For example, the text data includes software log files and maintenance records, which are used to estimate the optimal timing for preventive maintenance [18]–[22].

The drawbacks of the conventional technologies described above include the following.

- Data from sensors alone cannot address failures that occur in conjunction with software behavior.
- Analysis based on logs and maintenance records can be used to record and learn from known degradation conditions; it cannot support countermeasures with unexpected failure types.

In this study, we propose a method for extracting information on the causes of machine failure and supporting root cause analysis by combining the following types of data:

- Software log: text logs that record software behavior
- Sensor data: time-series numeric data that records hardware status (retrieved from devices like sensors, motors, position control systems)

The proposed method aims to combine these two data types to appropriately support root cause analysis for machines with complex behavior, which are difficult to handle by conventional techniques using only software logs or sensor data.

II. Problem Description

A. Target Machine

The target machines of the proposed method are automated machines with embedded PCs, where the control software runs, and a cycle of processing one object comprises multiple phases. For example, in the case of an automated inspection machine that inspects the appearance of products, the inspection process comprises six phases: (1) loading the object, (2) capturing images of the object, (3) image processing, (4) judgment, (5) result output, and (6) unloading the object.

Generally, the behavior of such machines is programmed based on the products processed in the production line. The number of repetitions and the time required for each phase change when the type of product is switched. For example, the number of captured images and the capturing position change depending on which part has to be inspected. Additionally, the required time changes by switching the image processing algorithm, and the motion time of the motors and other devices change by optimizing the movement route of the transport system and the imaging system.

Because the hardware motions differ from phase to phase, the sensor data recording the hardware status also varies depending on the number of phases and processing time, as shown in Fig. 1.

Usually, the control software running on the PC records the software behavior as log text files, whereas the recording of hardware status as numeric data was typically limited to a few important parts. However, in recent years, IoT technology and data utilization in the manufacturing industry have led to the adoption of a system to store related sensor data. Such data are especially necessary for complex machines, of which it is difficult to analyze the cause of failures, and the machines often have a system to record sensor data along with software logs.

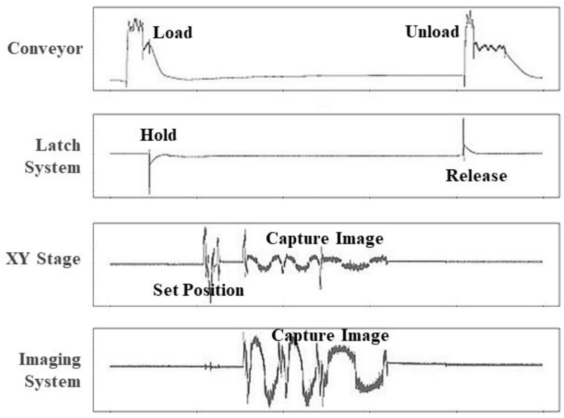
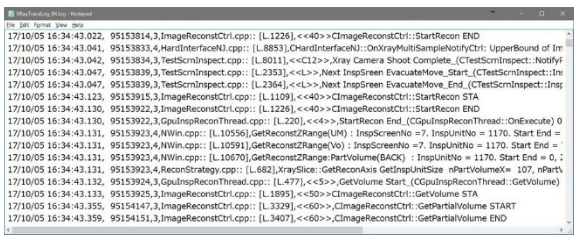


Fig. 1. Example of sensor data on hardware status.

B. Data Description

The inputs to the proposed method are software logs in text format that record the behavior of the control software, and time-series sensor data that record the hardware status. As shown below, both are common types of content and formats for the target machines and can be easily retrieved.

- Software log: Text files containing records of the actions performed by the software. Each item in the log comprises a timestamp and a text message. The timestamp is a fixed format, and the text message is an arbitrary string. Fig. 2 shows an example of a log file.
- Sensor data: Time-series numeric data such as sensor measurement values, motor torque values, a conveyor speed, xy-stage positions. Whether the data are recorded in a text or binary format depends on the machine specifications; however, any format is applicable as long as the time and numeric values can be read as a pair. Fig. 3 shows an example of sensor data.



```
17/10/05 16:34:43.022,  95153814,3,ImageReconstCtrl.cpp:: [L.1226],<<40>>CImageReconstCtrl::StartRecon END
17/10/05 16:34:43.041,  95153833,4,HardInterfaceNJ.cpp:: [L.8853],CHardInterfaceNJ::OnXrayMultiSampleNotifyCtrl: UpperBound of Im
17/10/05 16:34:43.042,  95153834,3,TestScrnInspect.cpp:: [L.8011],<<C12>>,Xray Camera Shoot Complete_(CTestScrnInspect::NotifyF
17/10/05 16:34:43.047,  95153839,3,TestScrnInspect.cpp:: [L.2353],<<L>>,Next InspSreen EvacuateMove_Start_(CTestScrnInspect::Ins
17/10/05 16:34:43.047,  95153839,3,TestScrnInspect.cpp:: [L.2364],<<L>>,Next InspSreen EvacuateMove_End_(CTestScrnInspect::Insp
17/10/05 16:34:43.123,  95153915,3,ImageReconstCtrl.cpp:: [L.1109],<<40>>CImageReconstCtrl::StartRecon STA
17/10/05 16:34:43.130,  95153922,3,ImageReconstCtrl.cpp:: [L.1226],<<40>>CImageReconstCtrl::StartRecon END
17/10/05 16:34:43.130,  95153922,3,GpuInspReconThread.cpp:: [L.220],<<4>>,StartRecon End_(CGpuInspReconThread::OnExecute) 0
17/10/05 16:34:43.131,  95153923,4,NWin.cpp:: [L.10556],GetReconstZRange(UM) : InspScreenNo =7. InspUnitNo = 1170. Start End =
17/10/05 16:34:43.131,  95153923,4,NWin.cpp:: [L.10591],GetReconstZRange(Vo) : InspScreenNo =7. InspUnitNo = 1170. Start End =
17/10/05 16:34:43.131,  95153923,4,NWin.cpp:: [L.10670],GetReconstZRange:PartVolume(BACK)  : InspUnitNo = 1170. Start End = 0,
17/10/05 16:34:43.131,  95153923,4,ReconStrategy.cpp:: [L.682],XraySlice::GetReconAxis GetInspUnitSize  nPartVolumeX=  107, nPart
17/10/05 16:34:43.132,  95153924,3,GpuInspReconThread.cpp:: [L.477],<<5>>,GetVolume Start_(CGpuInspReconThread::GetVolume)
17/10/05 16:34:43.133,  95153925,3,ImageReconstCtrl.cpp:: [L.1895],<<50>>CImageReconstCtrl::GetVolume STA
17/10/05 16:34:43.355,  95154147,3,ImageReconstCtrl.cpp:: [L.3329],<<60>>,CImageReconstCtrl::GetPartialVolume START
17/10/05 16:34:43.359,  95154151,3,ImageReconstCtrl.cpp:: [L.3407],<<60>>,CImageReconstCtrl::GetPartialVolume END
```

Fig. 2. Example of software log.

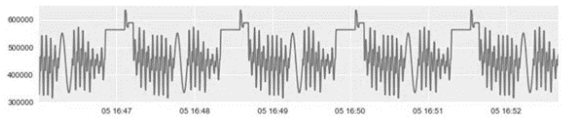


Fig. 3. Example of sensor data.

C. Problem Settings

Because it is difficult to collect sufficient failure sample data in manufacturing lines, we adopt the training data described as follows. Such data are typically available in actual operating lines.

- Collected while machines run normally; no failures occur
- Cases where multiple types of products are manufactured; not limited to a single type of product
- Containing only software log and sensor data; no information about the types of products required

The purpose of the proposed method is to extract the presumed causes from software logs and sensor data of failure cases based on training data learned in advance.

III. Methodology

A. Summary

In the proposed method, a chart (directed graph) showing the operation flow under normal conditions was created. Subsequently, by comparing a failure chart with the learned normal chart, the part related to the failure cause was extracted.

1. Create a chart based on text messages in the log.
2. Estimate the times when machine phases are switched based on sensor data and add weights to the parts of the chart that correspond to those times.
3. Integrate all charts created from the training data into a single chart as a normal chart.
4. Compare the normal chart with a failure chart and extract the differences.

The reason for weighting based on sensor data is that when analyzing failure causes, it is important to focus on change points from one phase to another and to consider the machine behavior in each phase.

The proposed method makes it possible to extract the differences when the machine phases are considered by combining software behavior information from the log and hardware status information from the sensor data.

B. Log-based Chart

The procedure to create a chart from software log is shown below.

1. Parse each line of the log and obtain the time from a fixed-format timestamp string. Create a token list from an arbitrarily formatted text message string by excluding symbols and breaking it down into words, as shown in Fig. 4.
2. Vectorize the token list of all the log lines using TF-IDF [23].
3. Classify all the lines into 200 clusters using the K-means method [24], as shown in Fig. 5.
4. Create a log cluster sequence with the cluster numbers in sequential order based on the time of each log line, as shown in Fig. 6.
5. Create a chart (directed graph) from the log cluster sequence using cluster numbers as nodes.
6. Weight the edges based on the transition frequency between nodes, as shown in Fig. 7.

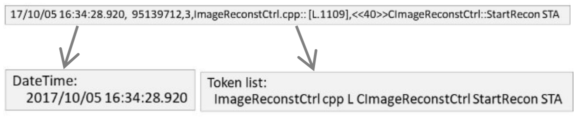
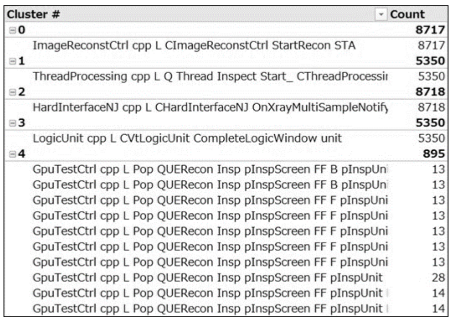

17/10/05 16:34:28.920, 95139712,3,ImageReconstCtrl.cpp:: [L.1109],<<40>>CImageReconstCtrl::StartRecon STA
DateTime:
2017/10/05 16:34:28.920
Token list:
ImageReconstCtrl cpp L CImageReconstCtrl StartRecon STA

Fig. 4. Parsing each log line.



Cluster #	Count
0	**8717**
ImageReconstCtrl cpp L CImageReconstCtrl StartRecon STA	8717
1	**5350**
ThreadProcessing cpp L Q Thread Inspect Start_ CThreadProcessi	5350
2	**8718**
HardInterfaceNJ cpp L CHardInterfaceNJ OnXrayMultiSampleNotify	8718
3	**5350**
LogicUnit cpp L CVtLogicUnit CompleteLogicWindow unit	5350
4	**895**
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF B pInspUn	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF B pInspUn	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF F pInspUni	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF F pInspUni	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF F pInspUni	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF F pInspUni	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF F pInspUni	13
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF pInspUnit	28
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF pInspUnit	14
GpuTestCtrl cpp L Pop QUERecon Insp pInspScreen FF pInspUnit	14

Fig. 5. Clustered log lines.

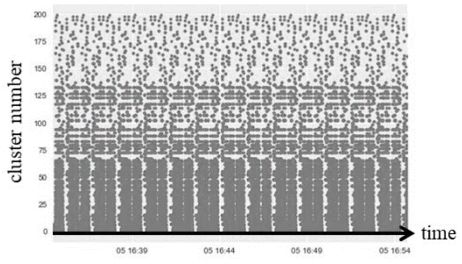


Fig. 6. Log cluster sequence.

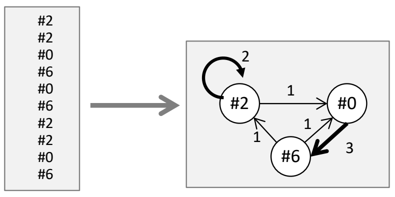


Fig. 7. Creating a chart from log cluster sequence.

Fig. 8 shows an example of a normal chart generated from 82,000 lines of log data using the above procedure.

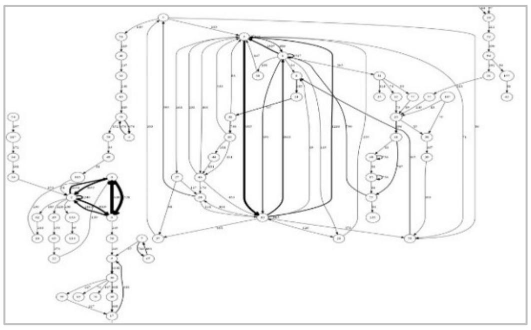


Fig. 8. Normal chart sample.

C. Weighting with Sensor Data

The chart created from the log using the procedure described in the previous section reflects only the frequency of transitions between nodes. Therefore, information about infrequent but important edges that correspond to phase switching and changes in the hardware status may be missing from the chart. To resolve this issue, the important edges are weighted by finding the change points of the sensor data.

1. Convert each of the sensor data retrieved from the hardware into time-series data representing change scores by the Change-Finder algorithm [25], as shown in Fig. 9.

2. Normalize the change score between 0 and 1.

3. Map the change score to the transitions in the log cluster sequence, as shown in Fig. 10.

4. Weight the edges of the chart created in the previous section based on the mapped change score, as shown in Fig. 11.

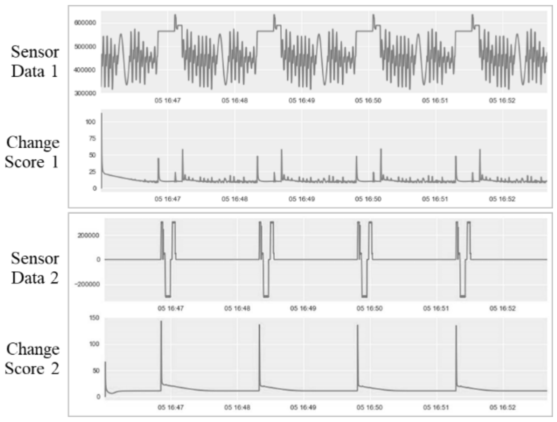


Fig. 9. Sensor data and change score.

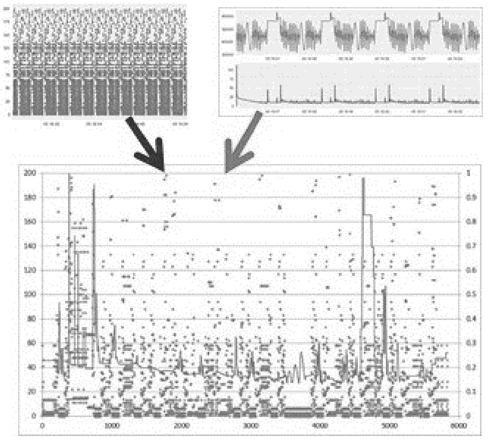


Fig. 10. Log cluster sequence with change score mapped.

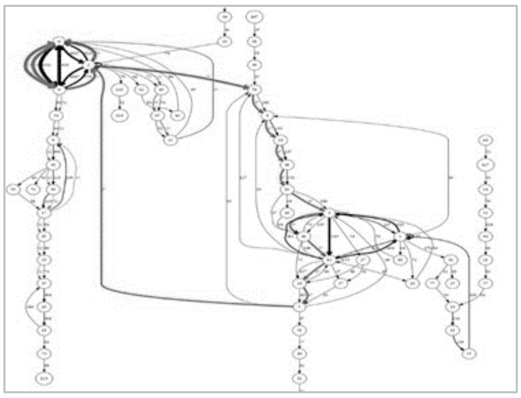


Fig. 11. Normal chart weighted with change score.

D. Normal Chart Integration

Using the procedures described in the previous section, a normal chart was created from each of the multiple normal case data included in the training data. Multiple normal charts are integrated to create a chart distribution matrix for comparison with a failure chart.

1. Create an edge weight matrix W for each normal chart, as in (1). Let w_{ij} be the weight of the edge that transitions from node i to node j. The weight matrix W corresponds to a single chart.

$$W = \begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n0} & w_{n1} & \cdots & w_{nn} \end{pmatrix} \quad (1)$$

2. Calculate the weight mean μ_{ij} and the weight variance σ_{ij} of each element of W created from each normal chart, as in (2).

$$\begin{aligned} \mu_{ij} &= Mean\ of\ normal\ chart\ w_{ij} \\ \sigma_{ij} &= Variance\ of\ normal\ chart\ w_{ij} \end{aligned} \quad (2)$$

E. Fault Chart Comparison

Calculate the differences in each part of the failure chart compared to the distribution in the normal chart.

1. Create the edge weight matrix of the failure chart using the same procedure as that of the normal chart.
2. For each element w_{ij} of the failure chart, calculate the anomaly degree a_{ij} as an indicator of how much it deviates from the normal chart, as in (3).

$$a_{ij} = \frac{|w_{ij} - \mu_{ij}|}{\sigma_{ij}} \quad (3)$$

3. Extract the edges with a high a_{ij} in the failure chart as the part related to the cause of failure.

The abnormal edges w_{ij} have cluster information of node i and node j, and can present the information for manual failure cause analysis.

With the method of extracting anomalous parts based on sensor data only, it is necessary to manually check how the control software behaves corresponding to the anomalous parts. In contrast, with the proposed method, the log text information is directly linked to the extracted clusters, which makes it possible to automatically show the anomalous status and log output by the control software at that time.

IV. Experiments

A. Experimental Conditions

To verify the effectiveness of the proposed method, we conducted experiments under the following conditions:

[Experimental data]
Normal condition (training data): 200 cases
Failure condition (test data): 20 cases

[Actual value]
Results of manual extraction from failure condition log data. One or more log lines were extracted from the log data of each case.

[True/False judgement]
If a node extracted by the method corresponds to one of the actual value log lines, the node is considered to be true.

[Methods for comparison]
To evaluate the effect of combining log and sensor data in the proposed method, we compare the proposed method with the results of weighting charts using log or sensor data only.

Method 1: Proposed method (log and sensor data)
Method 2: Weighting by log only
Method 3: Weighting by sensor data only

B. Experimental Results

With each of the three methods under comparison, a fixed number of nodes was extracted in descending order of the anomaly degree a_{ij}, and the number of "True" nodes was counted.

In this experiment, 200 nodes were generated per case; therefore, the total number of nodes in 20 cases was 4,000. The number of actual value "Positive" nodes was 118 out of 4,000.

For instance, Table 1 shows the results of Method 1, where the node count (the number of extracted nodes) is 10. For 20 cases, 200 nodes can be extracted with each method.

TABLE I. Result of Method 1 (node count = 10)

		Actual Value		
		Positive (118)	Negative (3882)	
Predicted Value	Positive (200)	41	159	Precision 0.21
	Negative (3800)	77	3723	Recall 0.35

Fig.s 12 and 13 show the values of precision and recall for the node count from 5 to 200, respectively. It can be seen that the proposed method yields better results than the other two methods for any node count.

The average number of actual value nodes per case was 5.9. If the number of extracted nodes is much larger than 5.9, the method does not refine nodes and does not meet the purpose of supporting failure cause analysis. In the proposed method, Recall = 0.6 and 0.75 for node count 15 and 20, respectively, which means the method can extract 60 to 70% of the actual value nodes within the useful range.

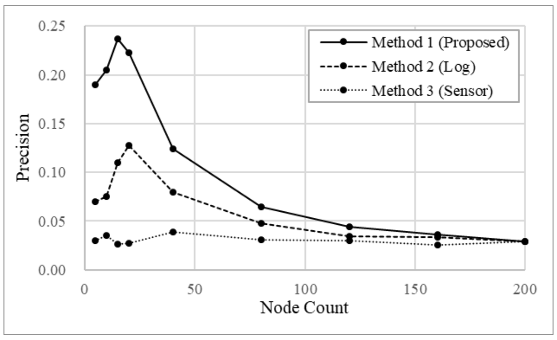


Fig. 12. Precision–Node count graph.

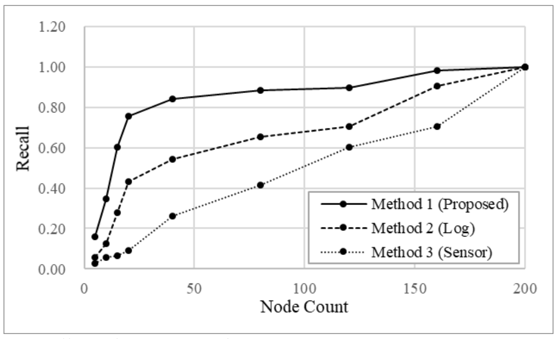


Fig. 13. Recall–Node count graph.

Fig. 14 shows the precision–recall graphs for the same experimental results. No significant variation exists in Method 3, suggesting the accuracy of the method of weighting with only sensor data is not very different from that of randomly selected nodes. Method 2 reflects the log information and is effective mainly for software-related failures; however, the proposed method achieves further improvement in accuracy.

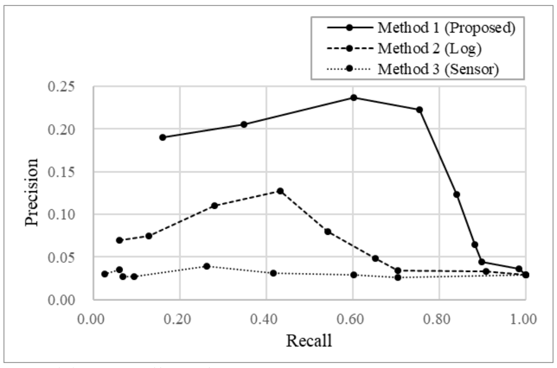


Fig. 14. Precision–Recall graph.

Fig. 15 shows an example of the chart differences between Method 1 and Method 2. Method 1 has an edge from node 114 to 67 weighted with sensor data (xy-stage position), and Method 2 has no edge between those nodes. Only Method 1 correctly extracted the edge, which showed a phase change in the imaging process. The token lists of Nodes 114 and 67 are as follows, and they show the cause of failure related to the position change after the image capturing process.

Node 114: capture process end clear volume
Node 67: start xy motion ready count

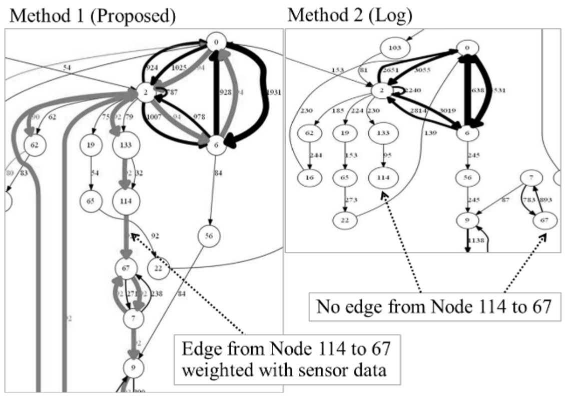


Fig. 15. Example of chart differences between Method 1 and Method 2.

V. Conclusions

In this paper, we proposed a method for automatically extracting log lines, which is highly relevant to failure causes, by combining software logs and sensor data to support failure cause analysis for machines with complex behavior.

By acquiring information on the control software behavior from the software log and information on the hardware status from the sensor data, the method achieved more accurate results than when each piece of information was used alone.

Additionally, the proposed method has improved the accuracy and explainability compared with the general methods of anomaly detection. The extraction results are provided with the differences between the anomalous and normal states as text and frequency of occurrence. This allows users to judge the validity of the results and provide feedback on the extraction errors.

With the development of AI technology, the manufacturing industry is moving toward automating failure countermeasures and preventive maintenance without human intervention; however, it has not yet reached a sufficient level for practical use. Even when such operations are fully automated in the future, technologies that improve accuracy and efficiency through human interaction will be still useful.

In future research, we would like to further improve the accuracy, verify the versatility of the method with different machines, add different sensor data types such as CPU or network load, and study ways to utilize human interaction.

References

[1] Y. Ono, Y. Onishi, T. Koshinaka, S. Takata, and O. Hoshuyama, “Anomaly detection of motors with feature emphasis using only normal sounds,” in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 2800–2804.

[2] W. Oh, “A simple sensor fault detection algorithm,” in Proc. of Int. Conf. on Computer Science, Data Mining&Mechanical Engineering, 2015, pp. 132–134.

[3] S. Ferreiro, E. Konde, S. Fernández, and A. Prado, “Industry 4.0: predictive intelligent maintenance for production equipment,” in European Conference of the Prognostics and Health Management Society, no, 2016, pp. 1–8.

[4] G. Manco et al., “Fault detection and explanation through big data analysis on sensor streams,” Expert Syst. Appl., vol. 87, pp. 141–156, Nov. 2017.

[5] Y. Merizalde, L. Hernández-Callejo, and O. Duque-Perez, “State of the Art and Trends in the Monitoring, Detection and Diagnosis of Failures in Electric Induction Motors,” Energies, vol. 10, no. 7, p. 1056, Jul. 2017.

[6] A. Lahrache, M. Cocconcelli, and R. Rubini, “Anomaly detection in a cutting tool by k-means clustering and support vector machines,” Diagnostyka, no. Vol. 18,3, pp. 21–29, 2017.

[7] P. Zhao, M. Kurihara, J. Tanaka, T. Noda, S. Chikuma, and T. Suzuki, “Advanced correlation-based anomaly detection method for predictive maintenance,” in 2017 IEEE International Conference on Prognostics and Health Management (ICPHM), 2017, pp. 78–83.

[8] F. Pittino, M. Puggl, T. Moldaschl, and C. Hirschl, “Automatic Anomaly Detection on In-Production Manufacturing Machines Using Statistical Learning Methods,” Sensors , vol. 20, no. 8, Apr. 2020.

[9] L. Li, K. Ota, and M. Dong, “Deep Learning for Smart Industry: Efficient Manufacture Inspection System With Fog Computing,” IEEE Trans. Ind. Inf., vol. 14, no. 10, pp. 4665–4673, Oct. 2018.

[10] S. Lee and D. Kim, “A Real-Time Based Intelligent System for Predicting Equipment Status,” in 2019 International Conference on Computational Science and Computational Intelligence (CSCI), 2019, pp. 429–432.

[11] S. Munirathinam and B. Ramadoss, “Big data predictive analtyics for proactive semiconductor equipment maintenance,” in 2014 IEEE International Conference on Big Data (Big Data), 2014, pp. 893–902.

[12] L. Martí, N. Sanchez-Pi, J. M. Molina, and A. C. B. Garcia, “Anomaly detection based on sensor data in petroleum industry applications,” Sensors , vol. 15, no. 2, pp. 2774–2797, Jan. 2015.

[13] G. S. Chadha, A. Rabbani, and A. Schwung, “Comparison of Semi-supervised Deep Neural Networks for Anomaly Detection in Industrial Processes,” in 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), 2019, vol. 1, pp. 214–219.

[14] T. T. Ademujimi, M. P. Brundage, and V. V. Prabhu, “A Review of Current Machine Learning Techniques Used in Manufacturing Diagnosis,” in Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing, 2017, pp. 407–415.

[15] Z. Li, Y. Wang, and K.-S. Wang, “Intelligent predictive maintenance for fault diagnosis and prognosis in machine centers: Industry 4.0 scenario,” Advances in Manufacturing, vol. 5, no. 4, pp. 377–387, Dec. 2017.

[16] J. Liu et al., “Anomaly Detection in Manufacturing Systems Using Structured Neural Networks,” in 2018 13th World Congress on Intelligent Control and Automation (WCICA), 2018, pp. 175–180.

[17] K. Kammerer, B. Hoppenstedt, R. Pryss, S. Stökler, J. Allgaier, and M. Reichert, “Anomaly Detections for Manufacturing Systems Based on Sensor Data—Insights into Two Challenging Real-World Production Settings,” Sensors , vol. 19, no. 24, p. 5370, Dec. 2019.

[18] J. Wang, C. Li, S. Han, S. Sarkar, and X. Zhou, “Predictive maintenance based on event-log analysis: A case study,” IBM J. Res. Dev., vol. 61, no. 1, pp. 11:121-11:132, Jan. 2017.

[19] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, “Log-based predictive maintenance,” in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, New York, USA, 2014, pp. 1867–1876.

[20] X. Zhang et al., “A Visual Analytics Approach for the Diagnosis of Heterogeneous and Multidimensional Machine Maintenance Data,” in 2021 IEEE 14th Pacific Visualization Symposium (PacificVis), 2021, pp. 196–205.

[21] Z. Li, J. Zhang, Q. Wu, and C. Kirsch, “Field-Regularised Factorization Machines for Mining the Maintenance Logs of Equipment,” in AI 2018: Advances in Artificial Intelligence, 2018, pp. 172–183.

[22] R. B. Patil, M. A. Patil, V. Ravi, and S. Naik, “Predictive modeling for corrective maintenance of imaging devices from machine logs,” Conf. Proc. IEEE Eng. Med. Biol. Soc., vol. 2017, pp. 1676–1679, Jul. 2017.

[23] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Inf. Process. Manag., vol. 24, no. 5, pp. 513–523, Jan. 1988.

[24] J. A. Hartigan and M. A. Wong, “Algorithm AS 136: A K-means clustering algorithm,” J. R. Stat. Soc. Ser. C Appl. Stat., vol. 28, no. 1, p. 100, 1979.

[25] Y. Kawahara and M. Sugiyama, “Sequential change-point detection based on direct density-ratio estimation,” Stat. Anal. Data Min., vol. 5, no. 2, pp. 114–127, Apr. 2012.