Received August 16, 2021, accepted September 18, 2021, date of publication September 22, 2021, date of current version October 5, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3114505*

# Programmable Motion-Fault Detection for a Collaborative Robot

**YE-SEUL PARK**[1], **(Member, IEEE), DONG-YEON YOO**[1], **(Member, IEEE), AND JUNG-WON LEE**[1,2], **(Member, IEEE)**
[1]Department of AI Convergence Network, Ajou University, Suwon 16499, South Korea
[2]Department of Electrical and Computer Engineering, Ajou University, Suwon 16499, South Korea

Corresponding author: Jung-Won Lee (jungwony@ajou.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government [Ministry of Science and ICT (MSIT)] under Grant 2020R1A2C1007400.

**ABSTRACT** Smart factories should be able to respond to catastrophic situations proactively, such as recalls caused by production line disruptions and equipment failures. Therefore, the necessity for predictive maintenance technology, such as fault detection or diagnosis of equipment has increased in recent years. In particular, predicting the faults of collaborative robots is becoming increasingly crucial because smart factories pursue efficient collaboration between humans and devices. However, collaborative robots have the characteristic of executing programmable motions designed by an operator, rather than performing fixed tasks. If existing fault diagnosis methods are applied to non–fixed programmable motions, problems arise in terms of setting absolute criteria for fault analysis, interpreting the meanings of detected values, and fault tracking or fault cause analysis. Therefore, we propose a method of programmable motion-fault detection by analyzing motion residuals to solve the three problems mentioned above. The proposed method can expand the fault diagnostic range of collaborative robots.

**INDEX TERMS** Programmable motion, collaborative robot, fault diagnosis, predictive maintenance, smart factory.

## I. INTRODUCTION

Smart factories should be able to respond to catastrophic situations proactively, such as recalls caused by production line disruptions and equipment failures. Therefore, predictive maintenance techniques are becoming increasingly important, including device fault detection and diagnosis [1]. Additionally, environments are being prepared such that big data can be collected from sensors embedded in equipment and processed using machine learning techniques. Predictive maintenance technology has attracted significant attention [2].

Unlike factory automation, where the entire production process is unmanned, smart factories use collaborative robots to perform specific delicate and repetitive tasks during the manufacturing process. The term "cobot," which is short for collaborative robot, refers to highly secure industrial robots that perform physical interactions in the same space with an operator. For conventional industrial robots, many overhead problems (e.g., reprogramming by experts and remodeling of production lines) occur when moving previously installed robots to perform different jobs. Cobots can freely learn motion commands by directly teaching by an operator. Cobots can also detect minor collisions. When a collision is detected, a cobot can be stopped and restarted immediately, facilitating human collaboration, and increasing productivity [3].

A cobot performs dynamic tasks driven by programmable motions rather than fixed tasks. First, the operator designs a program for the target task. They then instruct a cobot by considering the type of task, the end–effectors (e.g., gripper, vacuum pump, or presser) required for the target task, and collaboration processes (e.g., workspace size and range, and motion sequences) with humans. Typically, a cobot can execute two to three or more different programs per day, where each program is composed of tens to hundreds of motion command combinations. Each motion that makes up a program can be extremely diverse in terms of the radius or speed of rotation and the start and end points of movement (changing position coordinates).

The associate editor coordinating the review of this manuscript and approving it for publication was Ehsan Asadi.

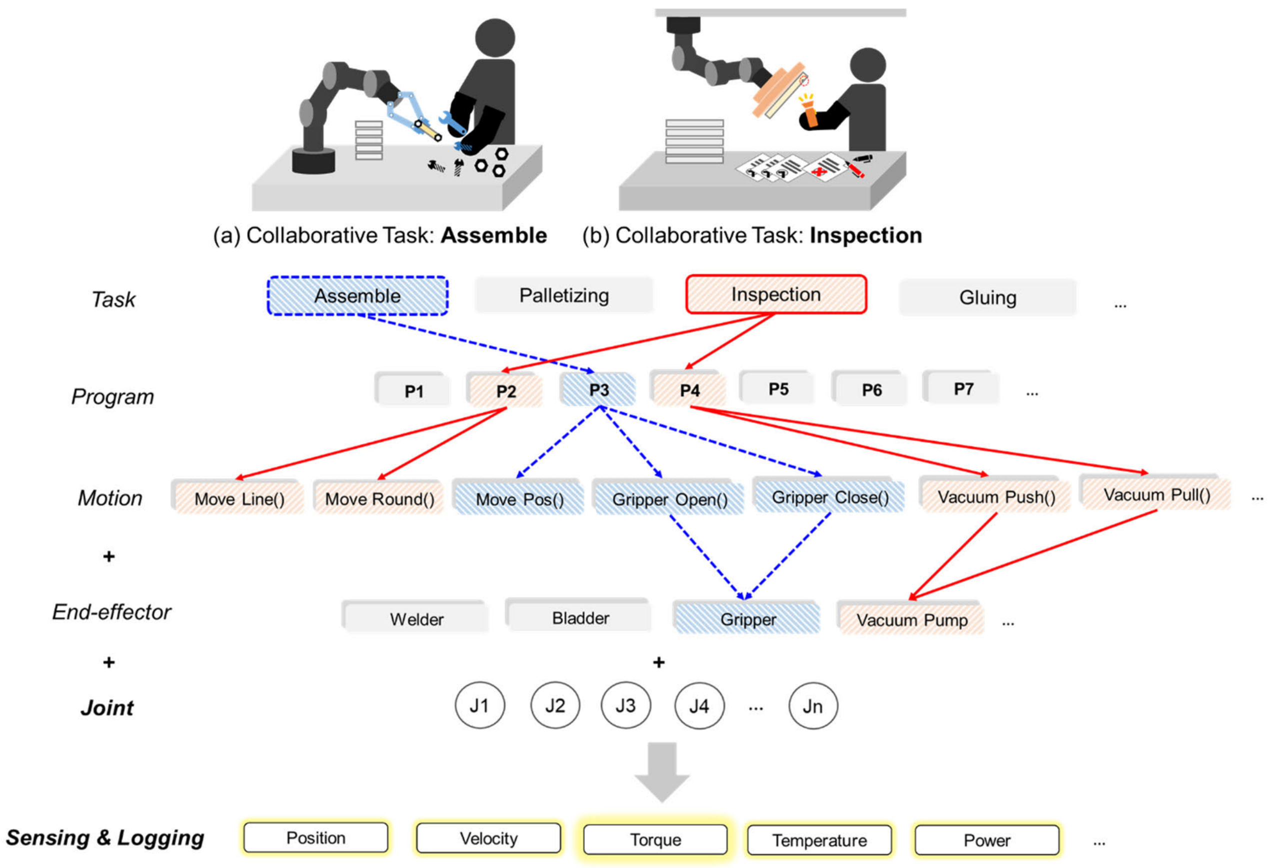


**FIGURE 1.** **The complexity of programmable motions in cobot.**

Fig. 1 illustrates the hierarchical structure of programmable motion for cobot tasks (Fig. 1(a) "Assemble" and Fig. 1(b) "Inspection"). A cobot can complete one task using multiple programs (Fig. 1(b) "Inspection: P2 and P4"), or by repeatedly calling a single program (Fig. 1(a) "Assemble: P3"). In addition, one program can perform a task by combining multiple motions. For example, in program "P3" in the assembly task, there are three motion commands: "Move_Pose()", "Gripper_Open()" and "Gripper_Close()". An operator can design a program ("P3") using various combinations of motions depending on the end effector type.

Cobot faults are detected by collecting and analyzing data such as position, speed, and torque data, as shown at the bottom of Fig. 1 ("Sensing & Logging"), from built–in sensors. Sensing data have very different patterns according to various settings for tasks, programs, motions, and end effectors in each axis ("J1" to "Jn") that is equipped with a sensor. Therefore, it is necessary to analyze the sensing data by considering all these factors for fault diagnosis. However, it is difficult to apply conventional fault diagnosis methods directly to cobots because such methods only focus on environments that utilize fixed motions [4–8]. The detailed problems faced when applying the existing methods are discussed below:

First, we cannot define the diagnostic threshold for detecting programmable motion-faults of cobots at the time of production line testing. A program is composed of various motions that are defined by an operator and one motion is operated under the various conditions, including load, end effector, moving speed, temperature, and so on. After shipment to the cobot's operator, a cobot is driven by many programmable motions, as shown in Fig. 1, depending on the target working conditions. Fig. 2 illustrates the data patterns from a torque sensor when a cobot repeatedly executes various motion commands according to the target manufacturing process. The x-axis represents the passage of time during which the cobot performs different motions and the y-axis represents the torque data from one axis collected while executing motions. The torque data sensed from each movement (motions A, B, and C) can be used to define a standard data pattern (green line) by statistically analyzing the data during each execution cycle. However, it is difficult to determine the expected value (traditional data pattern) of the green line in advance. This is because we cannot consider dynamic conditions such as all programs, operations, and

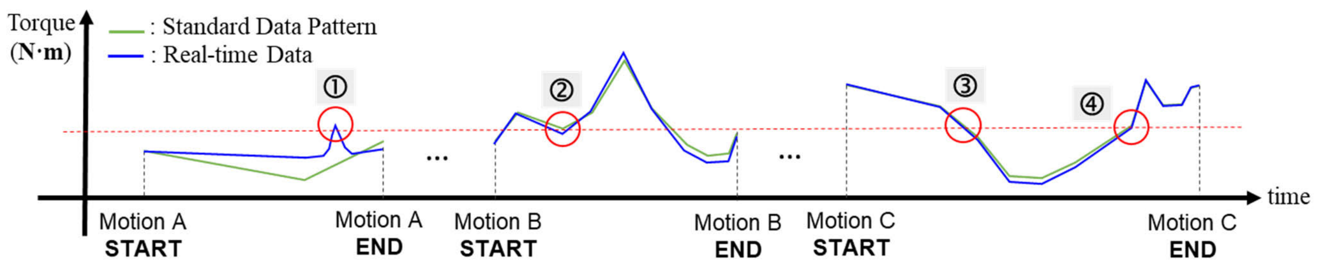


**FIGURE 2.** **Torque data generated from various motions.**

work environments defined by an operator at the time of shipment. In other words, it is difficult to accurately detect faults in a cobot using a predefined model. Therefore, it is necessary to find a reference pattern for the target situation by structurally analyzing the executed programs and motion commands.

Second, we face a problem in terms of interpreting the meaning of the detected values. Even if the sensing values obtained from a cobot appear to have similar values, they may have different meanings depending on the target tasks and motion commands. Fig. 2 presents a red dotted line through points ① to ④ in motions A, B, and C. All these points have the same torque values. However, a programmable motion-fault appears only in the case of point ①. This is because we must detect abnormalities based on a defined set of conditions (e.g., work performed, type of motion, or number of repetitions). Here, the meaning of 'programmable motion-fault' is not a physical fault but abnormal behavior of user-defined and command-level motions. If the 'move_pose()' motion, which stops at a fixed coordinate, does not work properly, the motion may occur to draw a slightly short-ended line. At this time, we want to prove that there are anomalous patterns of some sensing values hidden behind the faulty motion. As shown in "Motion A" in Fig. 2, there is an anomalous pattern. This is because the previously analyzed standard data pattern (green line) and actual measured value (blue line) differ when performing the same motion. Thus, we defined the standard data pattern based on the average values of several sections in which the cobot executes the same motions. However, because data–driven fault diagnosis methods cannot consider the diversity of the programmable motions of a cobot, anomalies may occur in a fixed test program. For example, it was difficult to detect abnormalities in programs designed by an operator [9]–[13]. Similarly, component–level fault diagnosis models such as motors [7], [8], rolling bearings [14]–[16], gears [17]–[19], and sensors [20] are also designed based on predefined test programs. Thus, it has a similar limitation in the conventional approach.

Third, it is difficult to explain the causes of abnormalities in a cobot using conventional fault diagnosis algorithms. Excessive program operations are the main cause of faults in the cobots. However, current fault diagnosis technology focuses only on detecting fault points based on predefined failure modes (e.g., position anomalies, vibration, or noise) [21], [22]. Therefore, it is difficult to reveal the underlying causes and prognostic symptoms in a scenario where a fault is detected. Additionally, it is challenging to provide traceability for programs and motions that affect the occurrence of faults. However, to prevent or predict failures, it is crucial to determine the causes of a fault by backtracking programmable operations when anomalies occur, including irrational program operations and motions.

In this paper, we propose the following methods for solving the three problems discussed above.

1. **We construct a data model that can hierarchically analyze the relationships between sensing values and cobot operation information.** To this end, in Section 3, three analysis data models (sensing, operation, and fault/failure) are proposed.
2. **We analyzed data correlations between the sensing data and operation data to track programmable motions with anomalies.** To this end, in Section 4, we propose a program and motion PM) indexing method and parsing technique.
3. **We define the detection criteria of a programmable motion-fault by statistically analyzing the sensing values with the same PM.** To this end, Section 5 proposes a motion residual analysis method for each parameter (position, speed, and torque). This method extracts a representative pattern based on the expected value for a regular operation and measures the abnormality of the current execution section.

In Section 2, we discuss existing predictive maintenance technologies for cobots based on related research. In Section 3, we present three data models. Section 4 presents a PM indexing method for understanding the correlations between tasks, programs, and motion. In Section 5, we propose a motion residual analysis method for programmable motion-fault detection. In Section 6, we present experimental results for the three proposed methodologies (data model, PM data indexing method, and analysis of motion residuals) based on data collected from a cobot. The conclusions are summarized in Section 7.

## II. RELATED WORK

Preventative maintenance is a method for maintaining device health by replacing parts regularly, regardless of the presence of malfunctions or abnormalities. Conventional preventive maintenance utilizes cost–effective maintenance strategies or work-scheduling techniques for equipment [23]. In this strategy, it is difficult to determine the appropriate replacement timings, which can lead to production line disruptions based on unnecessary replacement. Predictive maintenance technology attempts to detect faults in advance and predict failure [2]. This approach is primarily divided into physical–model–based approaches and data–driven approaches [24].

Physical model–based approaches attempt to diagnose failures by defining mathematical models of equipment physics. Many studies utilize this method based on its high accuracy for fault detection and the ability to diagnose based on small amounts of fault data [25]. In addition, [26] proposed a probabilistic fault estimation model for a single-link robot arm. However, it is difficult to define failure mechanisms in advance. This is because if a model has many conditions, then it is challenging to implement the model considering the complexity of multiple variables. Therefore, most factories apply these models based on experience from industry experts.

In contrast, data–driven approaches utilize built-in sensors to collect data. Technologies such as statistics, machine learning, and deep learning can be used to discover and detect device fault patterns. We can apply this approach to multivariate systems that are difficult to analyze using model–based approaches. However, this method has the disadvantage of requiring larger amounts of high–quality failure data compared to model–based approaches. Such data are not readily available when the sensor data collection is difficult.

For example, it is challenging to collect failure data in large–scale systems such as wind power generation [5] and steam turbine systems [6]. The authors of [5], [6] statistically analyzed a small amount of data to obtain a predefined fault detection model. In recent years, technologies of device connectivity, such as the Industrial Internet of Things (IIoT), have been applied, making it easier to collect sensor data. Therefore, recent studies have collected large amounts of big data to derive fault-detection models.

Data–driven fault detection methods can be divided into two main categories: statistics–based and learning–based methods. First, statistics–based methods attempt to detect device faults by numerically deriving health factors. For example, in studies related to the fault detection of motors [7]–[9], rolling bearings [14]–[16], gears [17]–[19], and sensors [20], the health factors of each device (e.g., vibration signal, speed signals, or kinetic signals) are derived via physical and spatial analysis (e.g., frequency domain analysis and spatiotemporal analysis). In addition, there is a feature extraction method for defect detection and isolation. The most representative feature extraction method is the principal component analysis (PCA) method, and various studies are being conducted using it. In [27], a PCA-based hidden Markov model was proposed for intelligent fault diagnosis. [28] used PCA method to extract spatial features from multivariate robot arm data.

In contrast, learning–based methods use collected data to generate a predictive model and derive health factors. For example, several machine learning techniques, such as linear discriminant analysis [29], support vector machine [30], and extreme learning machines [15], [16], have been used to diagnose faults in rolling bearings. Many recent studies have used deep learning techniques for fault diagnosis by collecting unprecedented amounts of failure data. In [31], [32], a rolling bearing fault diagnosis model was proposed by applying the structures of a convolutional neural network [31] and a recurrent neural network [32]. Additionally, many studies have used learning–based approaches to diagnose systems with high complexity, such as systems where physical models cannot be applied or fault patterns are difficult to analyze physically.

The cobot considered in this study is a multivariate system that detects signals from various sensors, such as position, speed, and torque sensors installed in industrial robot arms. The programs and operations are complex depending on the work environment. Therefore, a data–driven approach that analyzes the physical aspects of dynamic scenarios is a promising technique. In [10], a method for detecting faults in gearboxes based on vibration signals from industrial robots was developed. In this method, a health factor that maximizes and quantifies the fault information from vibration signals is defined.

However, this health factor does not consider dependencies on industrial robot operating environments (e.g., work type, PM type, end effector type, and load weight). Therefore, it cannot be generalized and applied to all the operation scenarios. Hidden Markov models [10], discrete wavelet transform artificial neural networks [11], and unsupervised learning models using signal analysis [12], [13] also support restricted operation environments. Most previous studies performed data–driven failure pattern analysis. However, because such methods have only been applied to fixed operating environments, it is difficult to detect faults when operating different programs in a smart factory. Therefore, it is necessary to analyze sensing data related to the conditions of various operating environments (e.g., PMs) and the criteria for determining ideality according to different operations.

Industrial robots should also consider risk, safety, and performance based on the ISO/TS 15066 standard [33]. Therefore, cobot developers perform various tests such as assembly tests, durability tests, and calibration for each joint, at the time of production. However, at the time of shipment, it is impossible to perform tests for all programmable motions that a cobot can perform. Therefore, various studies have attempted to develop more efficient testing methods.

The authors of [34] guided safe cooperation between humans and robots under the standard discussed in [33]. The research in [35] focused on welding work performed by cobots and humans and attempted to understand equipment conditions based on cobot performance evaluations in a work cell. Such industrial cobot testing methods aim to assess device health or performance and identify the causes of errors or failures (e.g., excessive program operations or overloading). However, previous studies have mainly performed testing under constraints related to specific tasks or scenarios (e.g., welding cells or assembly cells) and have focused on diagnosing relatively simple faults or failures. Therefore, the causes of faults, such as overloads or excessive program operations, cannot be analyzed in detail.

This study aimed to construct a data analysis model and develop an execution–oriented data analysis method for identifying the causes of faults under different conditions according to programmable motion, which is the most significant collaborative robot feature. The proposed method provides a basis for presenting sensing data and programmable motion information by backtracking the constructed data model when detecting anomalies.

## III. DATA ANALYSIS MODEL FOR PROGRAM-MABLE OPERATIONS

In this section, as illustrated in Fig. 3, we propose a data model that can analyze the execution data using built–in sensors. The proposed data model utilizes sensing data that are periodically generated, and operation data that are aperiodically generated. The details of this model are discussed in the following subsections.

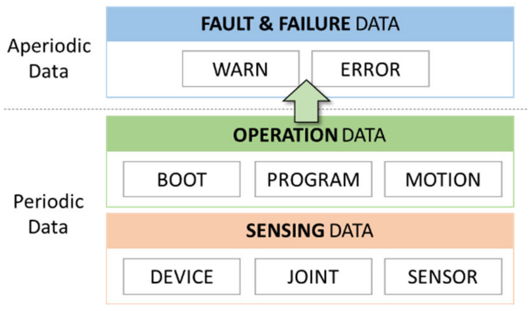


**FIGURE 3.** Data analysis model for a cobot.

### A. SENSING DATA

Cobots have six or seven joints depending on the desired degree of freedom (DOF), and each joint has various built–in sensors (e.g., torque, speed, and position). Fig. 4 presents the schema of the sensor data model. **Sensing data** are divided into three main categories, namely devices, joints, and sensors, according to the corresponding physical object. Figs. 4(a) - 4(c) present the structure of each type of data. The detailed characteristics of each element are summarized below.

- **Device Data**: Information regarding cobot specifications, including device identification number (ID), device name, DOFs, manufacturer, date of manufacture, and development version.
- **Joint Data**: Information regarding defined physical ranges, including joint IDs, load (payload), maximum speed (maxSpeed), radius of motion (range of travel), and peak torque.
- **Sensor data**: Various sensors can be installed on each joint, including position, speed, torque, temperature, pressure, and vision sensors. Sensor data can be generated according to each sensor's characteristics, and they include a sensor ID, sensor name, and measured value. In this case, the measured values were measured in real time. A detailed description of this is provided in Table 1.

**TABLE 1.** The description of sensor data.

| No | Data | Description |
|---|---|---|
| 1 | *tickCount* | indicates the time at which sensing data were generated. The unit is **ms** or **sec** depending on the sampling cycle. |
| 2 | *unit* | represents the units of data, such as **Nm** for torque, **rad/sec** for speed, and **degree** or **rad** for the position angle. |
| 3 | *sensingValue* | represents a sensing value such as **position**, **velocity**, or **torque**. |

### B. OPERATION DATA

We can collect the sensing data described above in real time according to the motion of the cobot. However, because sensing data are information acquired from sensors, it is non-trivial to derive operating information regarding the cobot's tasks, programs, and motions. Therefore, we propose a model for operation data (e.g., boot information, program information, motion information, and execution information) that can be analyzed by linking the sensing data of cobot. We classify the operation data into boot, program, and motion information according to the data characteristics. Data are generated aperiodically (intermittently) when an event such as a boot, program, or motion execution occurs. Fig. 5 presents the operation data schema, and a detailed description is provided below.

- **Datetime**: Indicates the date/time log. The logging format is "YYYY–MM–DD HH:MM:SS.SSS."
- **Level**: Indicates the importance of a recorded event. In the case of providing simple information, we can use level 1 (INFO), with level 2 for warnings (WARN) and level 3 for errors (ERROR).
- **Identifier**: As a field for identifying events, different identifiers can be recorded depending on the type of event that occurs. We can express boot information as #BOOT, program information as #PROGRAM, and motion information as #MOTION.
- **Message**: This is a field for detailed information in free text format. For example, in #PROGRAM, we can

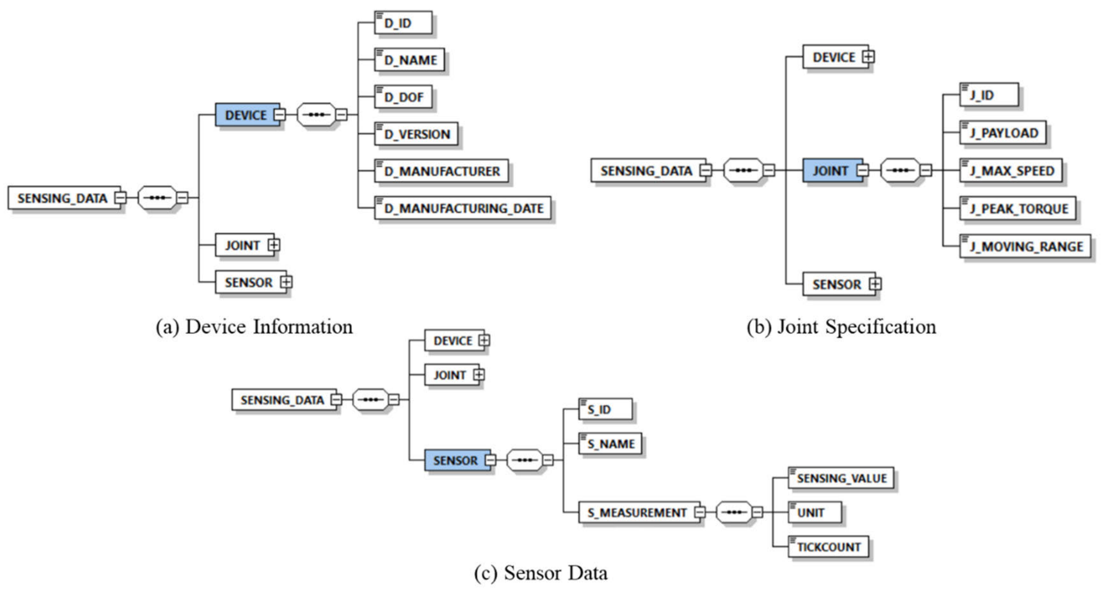


**FIGURE 4.** The schema for sensing data.

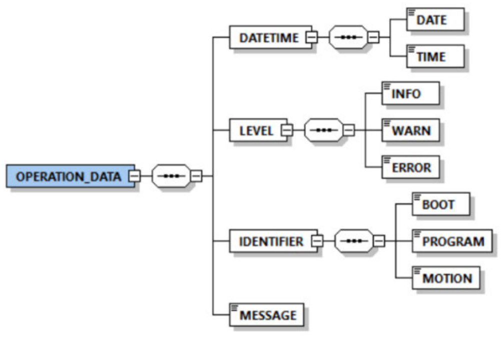


**FIGURE 5.** The schema for operation data.

record additional information regarding the program's name or version. In #MOTION, we can record information such as the type of movement or target position.

### C. FAULT & FAILURE DATA

Fault and failure data are data from the time that we recognize as a fault or failure. Sensing data and operation data are synthesized and generated aperiodically (intermittently) only in the fault and failure cases. The window size for synthesis is defined by a particular time before and after the occurrence of a fault or failure. It should be defined such that we can analyze data associations with failures. The event of a fault or failure refers to a period at level 2 (WARN) or level 3 (ERROR) in the operation data. The level of operation data is defined as follows depending on the severity of the fault.

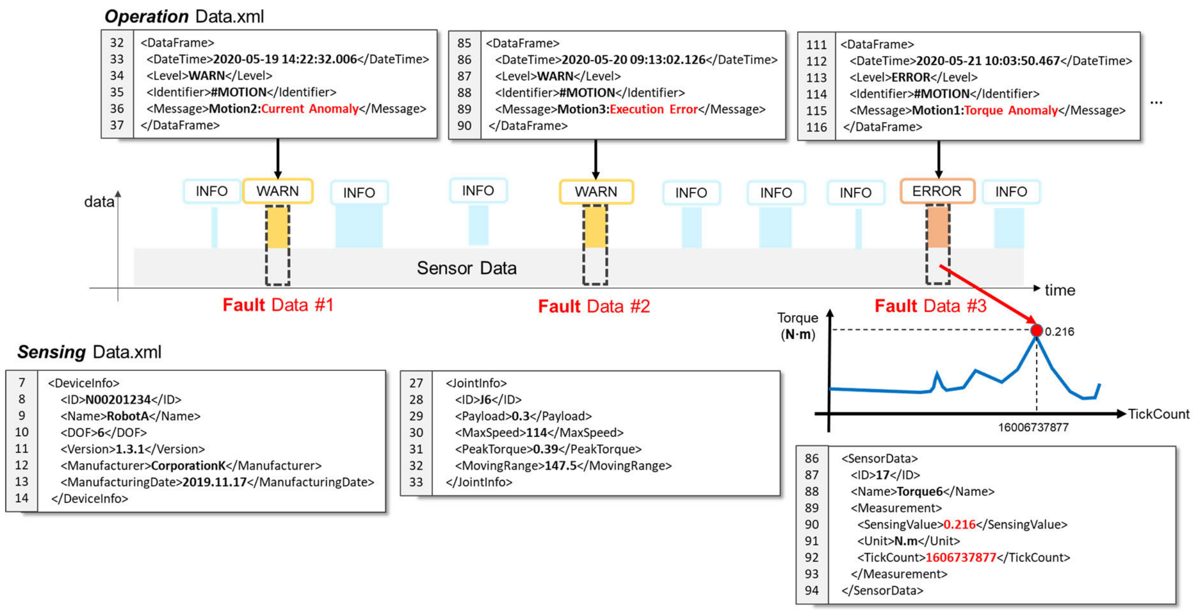
- **Level 2 (WARN)** represents an abnormal movement (such as ''Anomaly'') that can be a precursor to malfunctions or faults.
- **Level 3 (ERROR)** represents a case in which abnormal motion operations occur as a result of a malfunction or fault.

Fig. 6 presents examples of the data generated by a cobot based on the proposed model. Anyone is free to organize data generation based on the proposed schema. Sensing data are continuously generated according to the horizontal time axis and operation data are aperiodically generated when an event occurs. Here, we can assign INFO, WARN, and ERROR information according to the operation data level. The three XML–based operation data displayed at the top of Fig. 6 represent the cases of WARN and ERROR. Each block represents information including the current anomaly data, runtime errors, and torque anomaly data. Additionally, these data are synthesized as fault data because their severity is two or higher. The proposed data model can aid in analyzing the sensing and operation data of a cobot. Furthermore, fault and failure data can be systematically managed by defining the severity levels.

## IV. INDEXING OF PM DATA

Because cobots perform various programs and motions for workers, it is necessary to establish systematic analysis units, which are criteria for identifying normal and abnormal data sections. Therefore, in this section, we propose a method for extracting analysis units from the data model discussed above. We define analysis units in the form of the index hierarchy illustrated in Fig. 7 by reflecting the characteristics of repeated executions from the programmable motion relationship illustrated in Fig. 1. In Fig. 7, id is a unique number and idx is a number that automatically increases with iterations.

In summary, in Fig. 7, the designed index's roles are boot, program, motion, and each type of execution distinction. When the robot is booted up (**boot**) and the program (**program**) is repeatedly executed (**program execution**), the system counts the number of times of the internal



```
<DataFrame>
  <DateTime>2020-05-19 14:22:32.006</DateTime>
  <Level>WARN</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion2:Current Anomaly</Message>
</DataFrame>
```

```
<DataFrame>
  <DateTime>2020-05-20 09:13:02.126</DateTime>
  <Level>WARN</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion3:Execution Error</Message>
</DataFrame>
```

```
<DataFrame>
  <DateTime>2020-05-21 10:03:50.467</DateTime>
  <Level>ERROR</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion1:Torque Anomaly</Message>
</DataFrame>
```

```
<DeviceInfo>
  <ID>N00201234</ID>
  <Name>RobotA</Name>
  <DOF>6</DOF>
  <Version>1.3.1</Version>
  <Manufacturer>CorporationK</Manufacturer>
  <ManufacturingDate>2019.11.17</ManufacturingDate>
</DeviceInfo>
```

```
<JointInfo>
  <ID>J6</ID>
  <Payload>0.3</Payload>
  <MaxSpeed>114</MaxSpeed>
  <PeakTorque>0.39</PeakTorque>
  <MovingRange>147.5</MovingRange>
</JointInfo>
```

```
<SensorData>
  <ID>17</ID>
  <Name>Torque6</Name>
  <Measurement>
    <SensingValue>0.216</SensingValue>
    <Unit>N.m</Unit>
    <TickCount>1606737877</TickCount>
  </Measurement>
</SensorData>
```

**FIGURE 6. Data generation for the analysis of cobot operations: sensing data, operation data, and fault & failure data.**

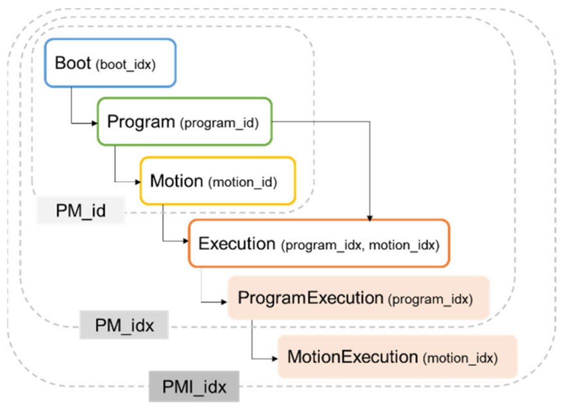

Boot (boot_idx)
Program (program_id)
Motion (motion_id)
PM_id
Execution (program_idx, motion_idx)
ProgramExecution (program_idx)
PM_idx
MotionExecution (motion_idx)
PMI_idx

**FIGURE 7. The PM index hierarchy for the analysis of cobot abnormalities.**

motions (**motion**) that make up the program have been repeatedly executed (**motion execution**).

## *A. HIERARCHICAL OPERATION DATA*

Operation data are designed to generate information regarding the tasks, programs, and motions executed by the cobots. Therefore, there is a dependency between the execution of programs and motion. There are four large hierarchical units: boot, program, motion, and execution. Boot and execution can be identified sequentially, and thus they are assigned in the form of idx. Because PM information is unique depending on the configuration, a value in the form of an id is assigned. A description of each component index is provided below.

- **boot_idx** is a section from powering the cobot on to turning it off.
- **program_id** is a unit according to the type of program and configuration content executed during one boot.
- **program_idx** is a unit of program execution when running a program that has the assigned program_id.
- **motion_id** is a unit of motion that comprises each program.
- **motion_idx** is a unit of motion execution when running a motion that has the assigned motion_id.

To determine whether there is an abnormality in the cobot, data generated under the same conditions (i.e., the same program (program_idx) and motion (motion_idx)) should be grouped. Therefore, in this study, we define three analysis units: PM_id, PM_idx, and PMI_idx. PM_id represents a PM type. For a given PM_id, we define the PM unit for each program execution as PM_idx and the motion execution section as PMI_idx (repetition of PM).

- **The PM Identifier (PM_id)** is a unit that can classify the same PM (Program + Motion). If both program_id and motion_id are the same, they can have the same PM_id.
- **The PM Index (PM_idx)** is a unit that can classify the same motion and program execution. When both program_idx and motion_id are the same, they can have the same PM_idx.
- **The PM–Iteration Index (PMI_idx)** is a unit that can classify the same motion execution and program execution. If both PM_idx and motion_idx are the same, they can have the same PMI_idx. Additionally, multiple PMI_idxs are included in one PM_idx.

Fig. 8 presents a diagram of an actual robot execution scenario using the proposed index hierarchy. As shown in Fig. 8, the operator powers the cobot on at 20/10/29 15:30:27.301 and four different programs are executed 18 times (e.g., repeat Program A 5 times, repeat Program B 2 times). Additionally,

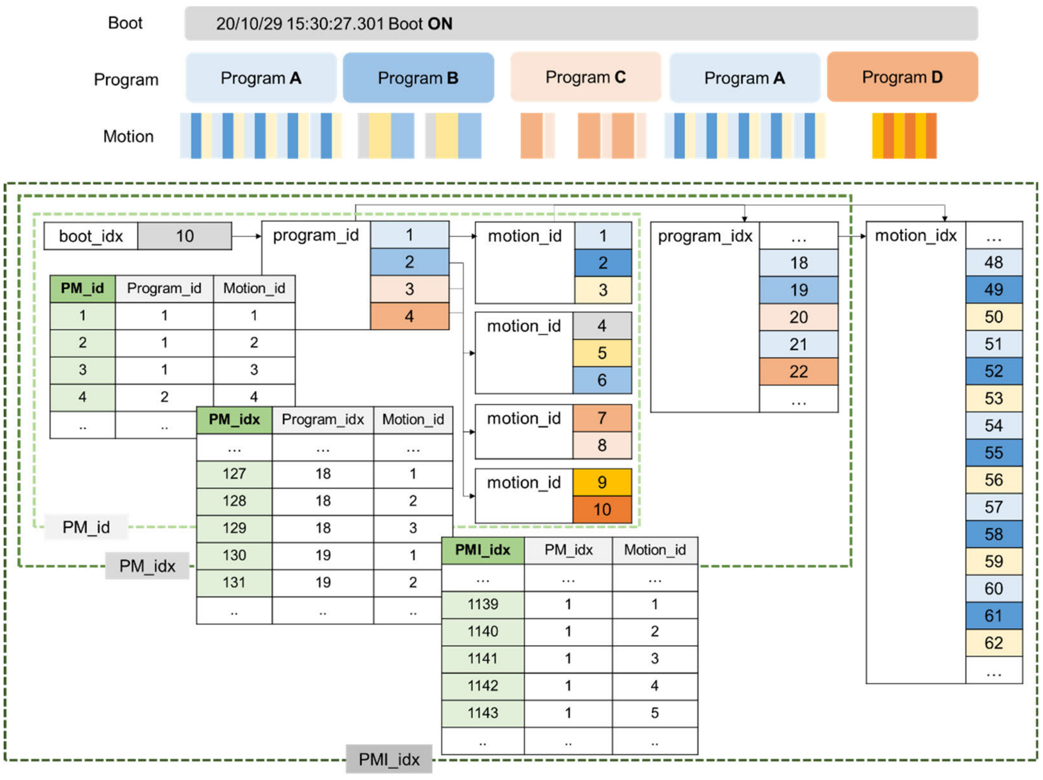


**FIGURE 8.** An Example of index hierarchy for operation data.

the type of motion that exists in each program differs depending on the program. For example, Program A contains three motions (light blue, blue, and yellow bars), and Programs C and D contain two motions. According to the proposed index hierarchy, 1 boot_idx, 4 program_ids (A to D), 10 motion_ids (A: 3, B: 3, C: 2, D: 2), 5 program_idx(A: 2, B: 1, C: 1, D: 1), and 48 motion_idx(A: 10 × 3, B: 2 × 3, C: 3 × 2, D: 3 × 2), B: are allocated. Additionally, 10 PM_ids (A: 3, B: 3, C: 2, D: 2), 13 PM_idxes (A: 3+3, B: 3, C: 2, D: 2), and 48 PMI_idxes are allocated according to the combination of PMs. Thus, we can use these index hierarchies to define the detection criteria for anomalous sections.

For example, suppose we have a PMI_idx execution section with a consistent PM_id or PM_idx. If we find any other data patterns in this section (PMI_idx with the same PM_idx), it represents a deviation from the commonly performed pattern and is detected as an abnormal execution section. Fig. 9 presents multiple PMI_idx values of 100, 104, 108, 112, 116, 118, 122, 124, and 132 for PM_id 3 and PM_idx 17. The execution start times of all indexes are normalized to zero and sampled.

In Fig. 9, the programmable motion information execution is consistent; therefore, the torque values of a similar pattern should be observed. If we observe a pattern exhibiting large differences from the common pattern, such as the torque value of PMI_idx 118, we can identify an abnormal section. The proposed index hierarchy can solve the difficulty of establishing analysis criteria that arise from the complexity of cobot tasks. If an abnormal section is detected, we can track the running PM information of the abnormal PMI_idx.

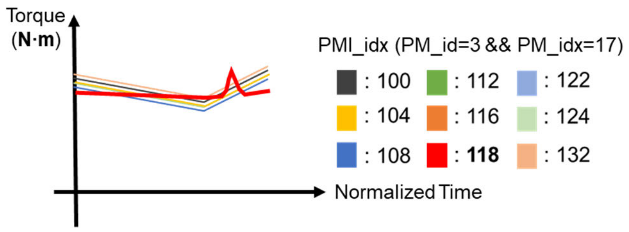


**FIGURE 9.** Torque data for a given PM_idx.

## B. PM PARSING RULES

This section discusses the extraction of PM information from the generated operation data. First, we define parsing rules to refine the operation data according to an XML schema. We then design the database schema presented in Fig. 10. This schema consists of five tables based on the indexes described in the previous section. In the database, the ID and index values that serve as primary keys for each table are automatically assigned incremental values. The value acting as a foreign key copied the value stored first in the other table and assigned it. The rule for extracting this value is defined in three steps, as shown in Fig. 11 (*Boot Info* → *Program Info* → *Motion Info*). Finally, we store

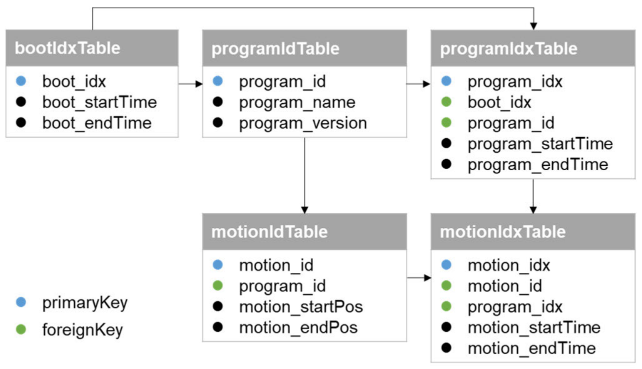


**FIGURE 10.** Data structure of PM indexes.

the information derived from each step in a table in the database.

The proposed method derives boot, program, and motion information by analyzing the generated operation data. We now describe the parsing process using example operation data generated based on the data model discussed in Section 3.

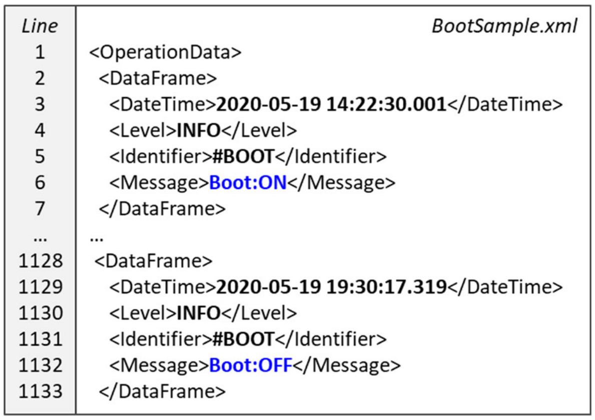
- *Parsing* **#Boot**: Fig. 12 presents the boot information of the cobot, and the time points of turning the cobot's power on and off are recorded as level 1 (INFO). This information can be attracted according to the rules in Fig. 11: # 1). The example in Fig. 12 can be extracted based on the ∗ON and ∗OFF strings recorded in the Message field of the data.
- *Parsing* **#Program**: Fig. 13 presents the information of the program executed by the cobot. First, we can refer to the Message field of the data and extract the program information, as shown in Fig. 11: # 2–1). Additionally, we can extract the program name and program version information as shown in the same figure. In Fig. 11: # 2–2), information regarding the program execution point and endpoint can be extracted based on ∗ON and ∗OFF by referring to the Message field for program execution information, such as boot information.
- *Parsing* **#Motion**: Fig. 14 presents the motion information generated by the cobot. Like the program information, we can extract motion information according to the rules established in Fig. 11: # 3–1) and # 3–2). First, we extract motion identification information according to the rules shown in Fig. 11: # 3–1). Here, we can obtain information such as commands, the current position, and target position by referring to the Message field. Motion execution information extraction is performed next. The rules in Fig. 11: # 3–2) can be applied, and the start and end points of motion are collected based on the ∗START and ∗END strings in the Message field and the DateTime field, where we record the corresponding strings. These data are extracted and saved simultaneously.

The extracted **#Boot**, **#Program**, and **#Motion** information is saved in the form shown in Fig. 15 based on the data structure in Fig. 10.

**Parsing Rule:** Algorithm for Parsing PM Information

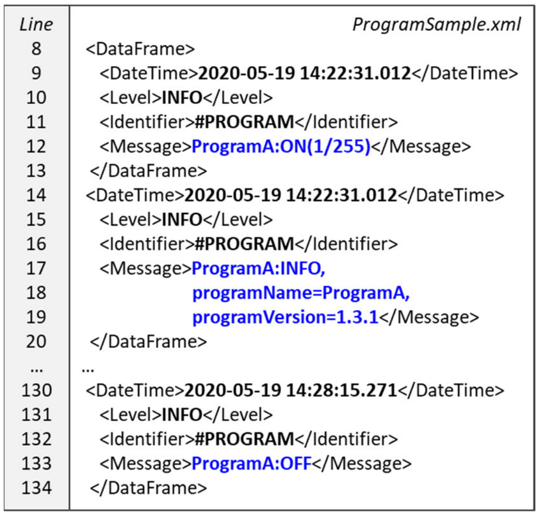
```
1: Start Procedure: Parsing (OperationData)
2: Input: OperationData.xml
3: Output: Tuple in DB table
4: switch(OperationData['Identifier'])
5:   case "#BOOT":
6:     # 1) PARSING BOOT INFO. (bootIdxTable)
7:       then set boot_startTime = currentTime
8:         do {tickCount++} while(O['Message'] == "Boot:OFF")
9:         then set boot_endTime = currentTime
10:             set boot_idx by AUTO INCREMENT
11:   case "#PROGRAM":
12:     # 2-1) PARSING PROGRAM INFO. (programIdTable)
13:       then set program_id by AUTO INCREMENT
14:           set program_name = programName in currentData
15:           set program_version = programVersion in currentData
16:     # 2-2) PARSING EXECUTION INFO. (programIdxTable)
17:       then set program_startTime = currentTime
18:         do {tickCount++} while(O['Message'] == " *programName :OFF")
19:           then set program_endTime = currentTime
20:               set boot_idx by searching bootTable
21:               set program_id by searching programIdTable
22:               set program_idx by AUTO INCREMENT
23:   case "#MOTION":
24:     # 3-1) PARSING MOTION INFO. (motionIdTable)
25:       then set motion_id by AUTO INCREMENT
26:           set program_id by searching programIdTable
27:           set motion_startPos = currentPos
28:           set motion_endPos = targetPos in currentData
29:     # 3-2) PARSING EXECUTION INFO. (motionIdxTable)
30:       then set motion_startTime = currentTime
31:         do {tickCount++} while(O['Message'] == " *motionName :OFF")
32:           then set motion_endTime = currentTime
33:               set program_idx by searching programIdxTable
34:               set motion_id by searching motionIdTable
35:               set motion_idx by AUTO INCREMENT
36: End Procedure
```

**FIGURE 11.** The pseudo code of PM parsing rules.



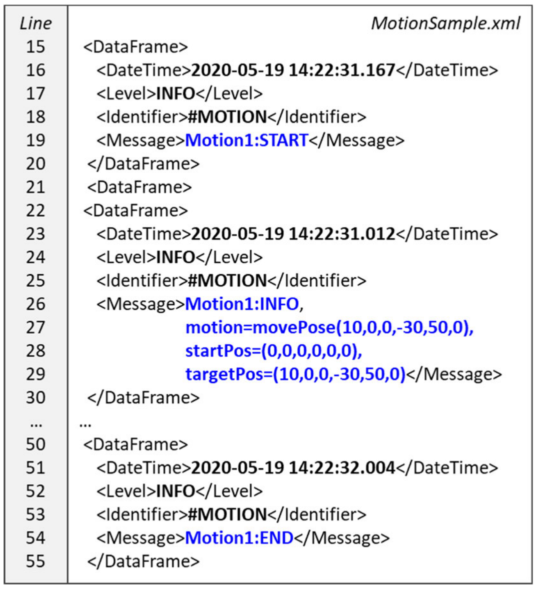
| Line | BootSample.xml |
|---|---|
| 1 | <OperationData> |
| 2 | <DataFrame> |
| 3 | <DateTime>**2020-05-19 14:22:30.001**</DateTime> |
| 4 | <Level>**INFO**</Level> |
| 5 | <Identifier>**#BOOT**</Identifier> |
| 6 | <Message>**Boot:ON**</Message> |
| 7 | </DataFrame> |
| … | … |
| 1128 | <DataFrame> |
| 1129 | <DateTime>**2020-05-19 19:30:17.319**</DateTime> |
| 1130 | <Level>**INFO**</Level> |
| 1131 | <Identifier>**#BOOT**</Identifier> |
| 1132 | <Message>**Boot:OFF**</Message> |
| 1133 | </DataFrame> |

**FIGURE 12.** An example of boot information.



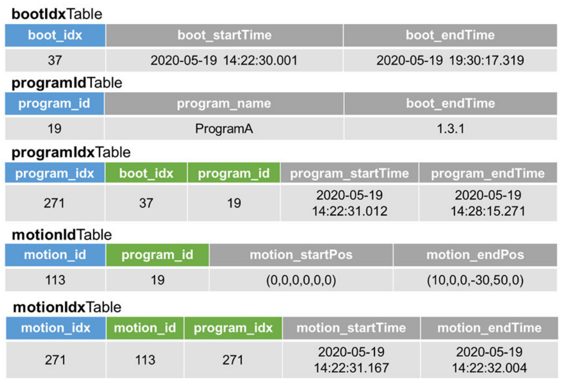
```
Line                                          ProgramSample.xml
<DataFrame>
  <DateTime>2020-05-19 14:22:31.012</DateTime>
  <Level>INFO</Level>
  <Identifier>#PROGRAM</Identifier>
  <Message>ProgramA:ON(1/255)</Message>
 </DataFrame>
<DateTime>2020-05-19 14:22:31.012</DateTime>
  <Level>INFO</Level>
  <Identifier>#PROGRAM</Identifier>
  <Message>ProgramA:INFO,
               programName=ProgramA,
               programVersion=1.3.1</Message>
 </DataFrame>
…
<DateTime>2020-05-19 14:28:15.271</DateTime>
  <Level>INFO</Level>
  <Identifier>#PROGRAM</Identifier>
  <Message>ProgramA:OFF</Message>
 </DataFrame>
```

**FIGURE 13. An example of program information.**



```
Line                                          MotionSample.xml
<DataFrame>
  <DateTime>2020-05-19 14:22:31.167</DateTime>
  <Level>INFO</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion1:START</Message>
 </DataFrame>
 <DataFrame>
<DataFrame>
  <DateTime>2020-05-19 14:22:31.012</DateTime>
  <Level>INFO</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion1:INFO,
              motion=movePose(10,0,0,-30,50,0),
              startPos=(0,0,0,0,0,0),
              targetPos=(10,0,0,-30,50,0)</Message>
 </DataFrame>
…
<DataFrame>
  <DateTime>2020-05-19 14:22:32.004</DateTime>
  <Level>INFO</Level>
  <Identifier>#MOTION</Identifier>
  <Message>Motion1:END</Message>
 </DataFrame>
```

**FIGURE 14. An example of motion information.**



**bootIdx**Table

| boot_idx | boot_startTime | boot_endTime |
|---|---|---|
| 37 | 2020-05-19 14:22:30.001 | 2020-05-19 19:30:17.319 |

**programId**Table

| program_id | program_name | boot_endTime |
|---|---|---|
| 19 | ProgramA | 1.3.1 |

**programIdx**Table

| program_idx | boot_idx | program_id | program_startTime | program_endTime |
|---|---|---|---|---|
| 271 | 37 | 19 | 2020-05-19 14:22:31.012 | 2020-05-19 14:28:15.271 |

**motionId**Table

| motion_id | program_id | motion_startPos | motion_endPos |
|---|---|---|---|
| 113 | 19 | (0,0,0,0,0,0) | (10,0,0,-30,50,0) |

**motionIdx**Table

| motion_idx | motion_id | program_idx | motion_startTime | motion_endTime |
|---|---|---|---|---|
| 271 | 113 | 271 | 2020-05-19 14:22:31.167 | 2020-05-19 14:22:32.004 |

**FIGURE 15. An example of the parsing data: boot, program, and motion index.**

In this manner, the proposed index system facilitates the hierarchical identification of programs and motions executed by the cobot, allowing consistent conditions (same PM) to be used for detecting abnormal motions. This allows us to support programmable motion tracking to determine the causes of abnormalities in the detected values.

## V. PROGRAMMABLE MOTION-FAULT DETECTION

This section discusses how to select a representative pattern for each programmable motion based on the PM index and detect programmable motion-fault based on motion residuals. Before describing the detailed method, we define anomaly, representative patterns, and motion residuals.

- **Anomaly**: Indicates an out–of–normal property and refers to a case that exhibits a pattern different from the typical pattern.
- **Representative**: Expected data pattern according to the target movement.
- **Motion residuals**: Differences between the expected values of representative patterns and measured values of the execution data.

As shown in Fig. 16, we can detect an abnormal section at a specific threshold value by analyzing the motion–specific residuals for each PMI_idx.

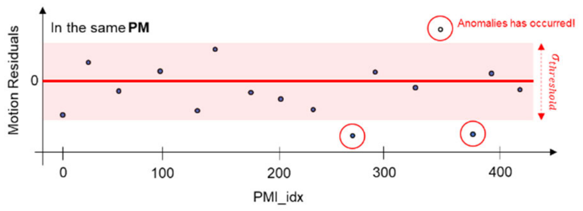


**FIGURE 16. Analysis of motion residuals for a target PM.**

The detailed process of calculating residuals to find typical patterns according to the target motion and detect execution anomalies is discussed in the following sections.

### A. REPRESENTATIVE PM PATTERNS

This section proposes a method for extracting representative patterns for each PM index based on the established analysis unit index hierarchy. First, the expected values of the sensor data for each PMI_idx are measured. Second, the PMI_idx with the median value is extracted and defined as a representative pattern. A detailed description of each procedure is provided in the following.

① **PMI Expectation** ($E_{PMI\_idx}$): The expected value can be measured by slicing the sensing data (e.g., torque, current, and speed) collected by the PMI. The expected value is calculated using Equation (1). In this equation, $D_{sensor}(t)$ is the value measured by the sensor at time t. Here, t is the TICK value of the PMI execution section, and T is the maximum execution time.

$$E_{PMI}\left(D_{sensor}\right) = \frac{1}{T}\sum_{t=1}^{T} D_{sensor}\left(t\right) \tag{1}$$

In Fig. 17, PM_id is assigned a value ranging from one to three. PM_idx is assigned a value ranging from 14 to 16 for motions A, B, and C in program A1. Each motion is executed repeatedly. Therefore, PMI_idx is ultimately assigned values ranging from 100 to 102. The y-axis represents the torque value measured during each PMI_idx and the length of each section is the same as those of $T_{100}$, $T_{101}$, and $T_{102}$. The expected PMI value (red line) represents the average value obtained by dividing the sum of the torque data for each section by the length of each PMI_idx.

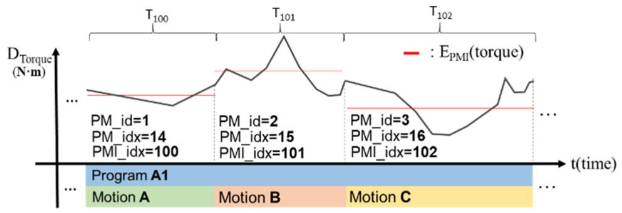


**FIGURE 17.** **Expected torque data for a partitioned PMI.**

The PMI expectation is the temporal mean of a series of sensing values over a single motion time. First, we cut out the sensor data for each section of the programmable motion. The length of each section varies depending on the type of motion. Second, we calculated the arithmetic mean of each parameter in the PMI_idx. Here, because all the comparative sections have the same motion type, the motion lengths are the same, and the calculated arithmetic means are all equidistant.

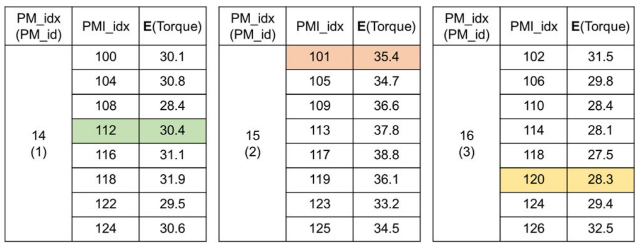
② **PMI Representative**($PMI_{Representative}$): The following process is used to select the index with the median value from a set of PMI_idxs with the same PM_idx. We define the representative PMI_idx using Equation (2). This equation targets PMI_idxs with the same PM_idx and selects the PMI with the median value among the calculated expected PMIs as the representative PMI_idx.

$$\boldsymbol{PMI}_{Representative} = medianIndex(List\,[E_{PMI}]) \quad (2)$$

In the example in Fig. 18, the estimated torque value of PMI_idx is displayed for each PM_idx. We select the PMI_idx with the median value for each PM as an execution index with a representative pattern. In the example in Fig. 18, we can extract PMI_idxs 112, 101, and 120 as $PMI_{Representative}$s. Representative pattern extraction is performed when the operation data level is 1 (INFO) for all PM_idx executions. In other words, we exclude exceptions such as level 2 (WARN) and level 3 (ERROR) to ensure standardness or ideality.

## B. MOTION RESIDUAL ANALYSIS

This section proposes a motion residual analysis technique based on the selected representative pattern ($PMI_{Representative}$). In legacy studies, residual analysis is widely used for fault



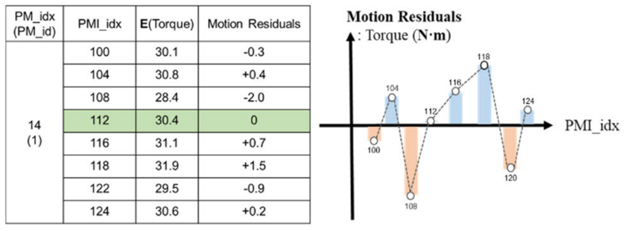
| PM_idx (PM_id) | PMI_idx | E(Torque) | PM_idx (PM_id) | PMI_idx | E(Torque) | PM_idx (PM_id) | PMI_idx | E(Torque) |
|---|---|---|---|---|---|---|---|---|
| 14 (1) | 100 | 30.1 | 15 (2) | 101 | 35.4 | 16 (3) | 102 | 31.5 |
| | 104 | 30.8 | | 105 | 34.7 | | 106 | 29.8 |
| | 108 | 28.4 | | 109 | 36.6 | | 110 | 28.4 |
| | 112 | 30.4 | | 113 | 37.8 | | 114 | 28.1 |
| | 116 | 31.1 | | 117 | 38.8 | | 118 | 27.5 |
| | 118 | 31.9 | | 119 | 36.1 | | 120 | 28.3 |
| | 122 | 29.5 | | 123 | 33.2 | | 124 | 29.4 |
| | 124 | 30.6 | | 125 | 34.5 | | 126 | 32.5 |

**FIGURE 18.** **The selection process of an $E_{PMI}$ (PMI expec-tation).**

detection in discrete-time systems, such as [36]. In this study, a motion residual was obtained by quantifying the difference between the current motion and a typical motion. Here, we use the PMI representative as a criterion for determining abnormality to calculate motion residuals using Equation (3). The **MR**(PMI) is calculated according to each PMI_idx, as shown in (3). For each PM_idx, we calculate the difference between the expected value of the PMI representative ($E_{Representative}$) selected using (2) and the expected value of the corresponding PMI ($E_{PMI}$).

$$\boldsymbol{MR}\,(PMI) = E_{PMI} - E_{Representative} \quad (3)$$

By applying (3) to the example in Fig. 18, we can analyze the residuals for each motion, as shown in Fig. 19. In the example in Fig. 19, we select a representative execution index of 112 for the nine execution indexes with a PM_idx of 14 (PM_id of 1). The expected torque value for the index of 112 is 30.4. For each PMI_idx torque value, calculating the residual for each motion yields values ranging from −2.0 to +1.5. This residual analysis method can be used as an indicator to evaluate the abnormality of each execution indicator.



| PM_idx (PM_id) | PMI_idx | E(Torque) | Motion Residuals |
|---|---|---|---|
| 14 (1) | 100 | 30.1 | -0.3 |
| | 104 | 30.8 | +0.4 |
| | 108 | 28.4 | -2.0 |
| | 112 | 30.4 | 0 |
| | 116 | 31.1 | +0.7 |
| | 118 | 31.9 | +1.5 |
| | 122 | 29.5 | -0.9 |
| | 124 | 30.6 | +0.2 |

**FIGURE 19.** **Analysis of motion residuals for a given PM.**

## C. PROGRAMMABLE MOTION-FAULT DETECTION

Finally, we propose a method for detecting programmable motion-fault exhibiting abnormalities using the calculated residual values for each operation. The proposed detection method consists of two main steps, as shown in Fig. 20.

First, we detect a case in which the program execution data show large differences from the average data for the same motion. Variance filtering was used to extract PM_idx with many anomalies. This primary classification process detects anomalies in an execution dataset with large fluctuations caused by persistent anomalies and filters small sections that temporarily take anomalies.

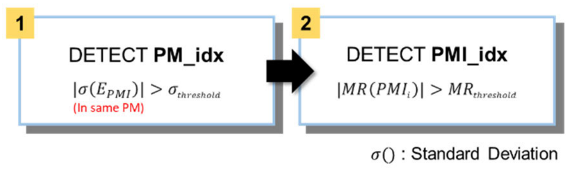

1
DETECT PM_idx
$|\sigma(E_{PMI})| > \sigma_{threshold}$
(In same PM)
2
DETECT PMI_idx
$|MR(PMI_i)| > MR_{threshold}$
$\sigma()$ : Standard Deviation

**FIGURE 20.** Analysis of motion residuals in PM.

Second, we detect execution indexes that show actual abnormalities based on the selected candidate set. The step–by–step detection method is summarized as follows.

① **PM anomaly detection**: For each PM_idx ($\sigma$(EPMI)), the variance (standard deviation) of the residual values for each motion is evaluated for each PMI_idx with the same PM_idx. Then, PM_idxs with a large variance (standard deviation) are detected.

② **PMI anomaly detection**: PMI anomalies are detected among indexes when executing the corresponding PM for the pre-selected PM_idxs. In this case, we use the residuals (MR: (3)) for each motion calculated for each PMI_idx. Additionally, any PMI_idx outside a certain threshold represents a section with an abnormality. Here, the threshold can be adjusted experimentally.

We proposed a data indexing method for modeling complex working conditions (e.g., programs and actions). This method makes it possible to identify abnormalities and perform a precise diagnosis based on typical data for a target PM. It also provides traceability of the type and behavior of work programs for identified anomalous executions, as well as the actions that generate faults.

## VI. EXPERIMENTAL RESULTS

The proposed method was applied to a real cobot to analyze and evaluate the proposed method. We describe data generation, data indexing, and data analysis (anomaly detection) in the following subsections. Additionally, the experimental results were verified and evaluated.

### A. DATA GENERATION

In this study, as shown in Fig. 21, we developed two software applications: one that generates sensing data and operation data (*dataGenerator*), and another that collects data from sensors built into or externally mounted on a cobot (*dataCollector*).

- **Configuration of the experimental environment**: Niryo One [37], which is a six–axis cobot, was used in our experiments. For motion control, position data (x, y, z, roll, pitch, yaw), angle value (radians), and temperature of each axis are internally recorded by sensors using the provided application programming interface. The sensors were mounted at the end of the robot arm. Additionally, by using a Raspberry Pi 3B+ board and installing sensors externally, we constructed an environment in which the current and voltage values applied to the motor could be collected. Therefore, a total of five parameters (position, angle, voltage, temperature, and current) were measured for each joint. We collected time-series data with a time step of 10 ms.
- **Test program design**: To fabricate car mats in a real car factory, we considered the gluing task as a target scenario. As shown on the right side of Fig. 21, the environment for producing car mats was simplified. Specifically, experiments were conducted by simulating the end effector of the cobot arm using touch and draw operations on a touch screen instead of attaching a real edge shape. There are five programs and the starting and ending points of each program are the same. However, the parameter compositions of the motion commands and number of motion commands vary.
- **Data generation and collection**: Each time we executed the task program, the sensing data were generated in XML format. Fig. 22 presents an example of constructing a database from sensing data and operation data generated in this environment. The detected end position values (x, y, z, roll, pitch, and yaw) can be observed in the sensing data. Events were generated in the format (OperationData.xml, SensingData.xml) shown in Fig. 6 according to the proposed data model standard.

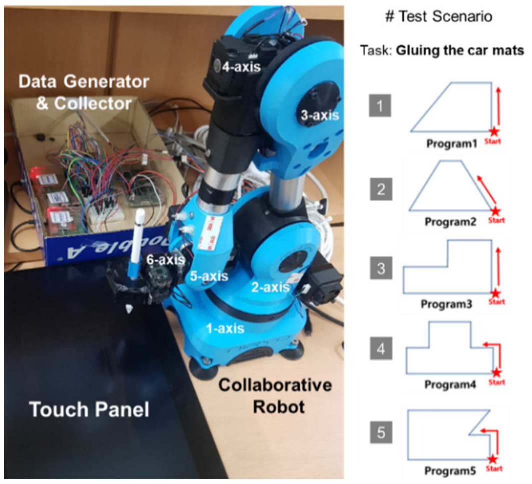
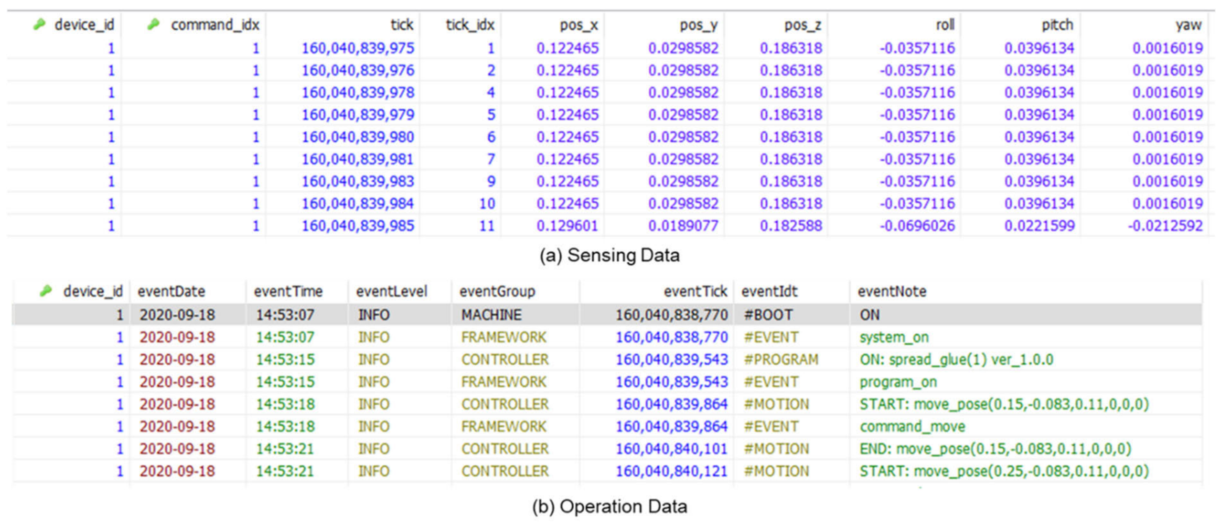


**FIGURE 21.** The environment for data generation.

### B. DATA INDEXING

This section presents the indexing results boot and PM information according to the proposed structure of the collected data. We executed the test program for approximately two weeks using five test programs described above. The system was turned on a total of 59 times and the five programs were executed 257 times. The index created during this process was analyzed, as shown in Fig. 23.

- **Results of #BOOT Indexing**: The horizontal axis in Fig. 23 represents boot_idxs by date. During the collection period, 59 boot_idxs were created, and the



| device_id | command_idx | tick | tick_idx | pos_x | pos_y | pos_z | roll | pitch | yaw |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 160,040,839,975 | 1 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,976 | 2 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,978 | 4 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,979 | 5 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,980 | 6 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,981 | 7 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,983 | 9 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,984 | 10 | 0.122465 | 0.0298582 | 0.186318 | -0.0357116 | 0.0396134 | 0.0016019 |
| 1 | 1 | 160,040,839,985 | 11 | 0.129601 | 0.0189077 | 0.182588 | -0.0696026 | 0.0221599 | -0.0212592 |

(a) Sensing Data

| device_id | eventDate | eventTime | eventLevel | eventGroup | eventTick | eventIdt | eventNote |
|---|---|---|---|---|---|---|---|
| 1 | 2020-09-18 | 14:53:07 | INFO | MACHINE | 160,040,838,770 | #BOOT | ON |
| 1 | 2020-09-18 | 14:53:07 | INFO | FRAMEWORK | 160,040,838,770 | #EVENT | system_on |
| 1 | 2020-09-18 | 14:53:15 | INFO | CONTROLLER | 160,040,839,543 | #PROGRAM | ON: spread_glue(1) ver_1.0.0 |
| 1 | 2020-09-18 | 14:53:15 | INFO | FRAMEWORK | 160,040,839,543 | #EVENT | program_on |
| 1 | 2020-09-18 | 14:53:18 | INFO | CONTROLLER | 160,040,839,864 | #MOTION | START: move_pose(0.15,-0.083,0.11,0,0,0) |
| 1 | 2020-09-18 | 14:53:18 | INFO | FRAMEWORK | 160,040,839,864 | #EVENT | command_move |
| 1 | 2020-09-18 | 14:53:21 | INFO | CONTROLLER | 160,040,840,101 | #MOTION | END: move_pose(0.15,-0.083,0.11,0,0,0) |
| 1 | 2020-09-18 | 14:53:21 | INFO | CONTROLLER | 160,040,840,121 | #MOTION | START: move_pose(0.25,-0.083,0.11,0,0,0) |

(b) Operation Data

**FIGURE 22.** An example of the sensing and operation data.

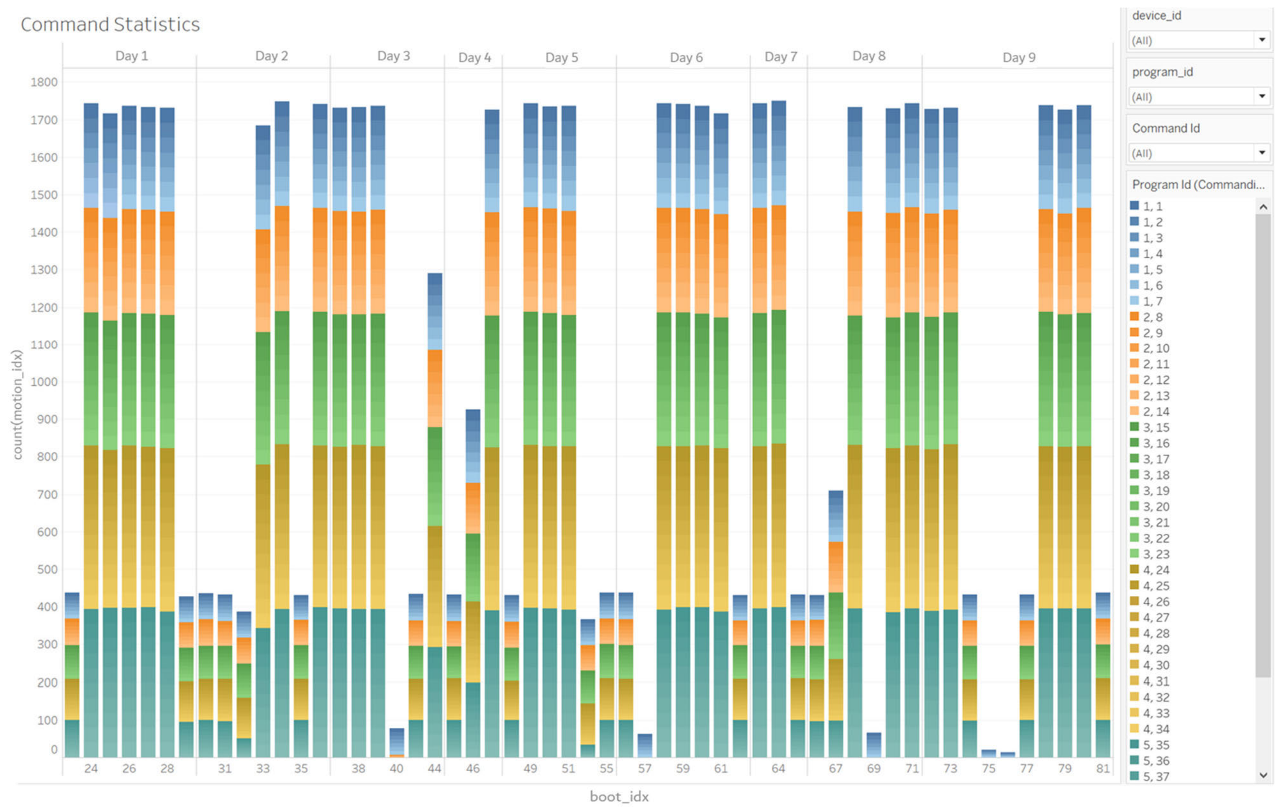


**FIGURE 23.** Results of data indexing: boot, program and motion information.

cobot was booted at least thrice daily and at most ten times per day.

- **Results of #PROGRAM Indexing**: Fig. 23 presents the number of executions of the programs (program_idxs) executed during the collection period. Five programs (five mat-drawing programs) were executed during the collection period.
- **Results of #MOTION Indexing**: Fig. 23 presents the number of executions for the executed motions (motion_idxs). Colors represent program types and

motion types are represented by different gradients within the same color tone. We confirmed that five programs were executed for each operation section (boot_idx) and that the number of motion executions for each program was different. Because the number of commands in each program differs, there are differences in the numbers of accumulated operations per program execution.

Based on the defined PM index, 44 PM_ids, 2,073 PM_idxs, and 57,855 PMI_idxs were generated for anomaly analysis. This indexing system supports systematic analysis by facilitating a hierarchical understanding of the work performed.

### *C. DATA ANALYSIS*

This section discusses the results of identifying anomalous sections based on motion residual analysis. We extracted the position, angle, current, voltage, and temperature data. We performed residual analysis on the four parameters of angle, current, voltage, and temperature (position values, which only detect exact endpoint positions, were excluded) to identify abnormal sections according to the derived criteria. The analysis process for the detailed residuals for each motion is summarized below.

① **Measurement of motion residuals**: First, we calculated the expected PMI values using Equation (1). We then extracted a representative pattern for each PM_idx according to Equation (2). In this study, 2,073 representative patterns were identified. The residuals of each motion for each PMI_idx were measured using Equation (3), as shown in Fig. 24. The partial PM_idx sections (995 to 1004) for joint four among the motion residual values extracted for each operation for the parameters of voltage, temperature, current, and angle at the bottom of the y-axis can be observed. For example, if the angle value of the PM_idx is either 996 or 1003, because each operation's residual value is almost zero, we can confirm that the operation is similar to the typical pattern. In contrast, we can observe that the residual value in 997 and 998 are significantly different from the representative pattern than other PM_idx (negative or positive depending on the execution).

② **PM anomaly detection**: The following results reveal the detection of many abnormal PM_idxs. As shown in Fig. 25, we analyzed the residual value variances for each operation according to the PM_idxs. We calculated the variance values by dividing them according to the parameters and joints. The PM_idxs with the top 10% of variance values are indicated by the orange color in Fig. 25. These PM_idxs were selected as candidate index for identifying abnormal sections.

The selected PM_idx refers to a group of motion performance indexes with large execution errors, even though they all represent the same motion when performing the motions corresponding to each index. Figure 26 shows two samples

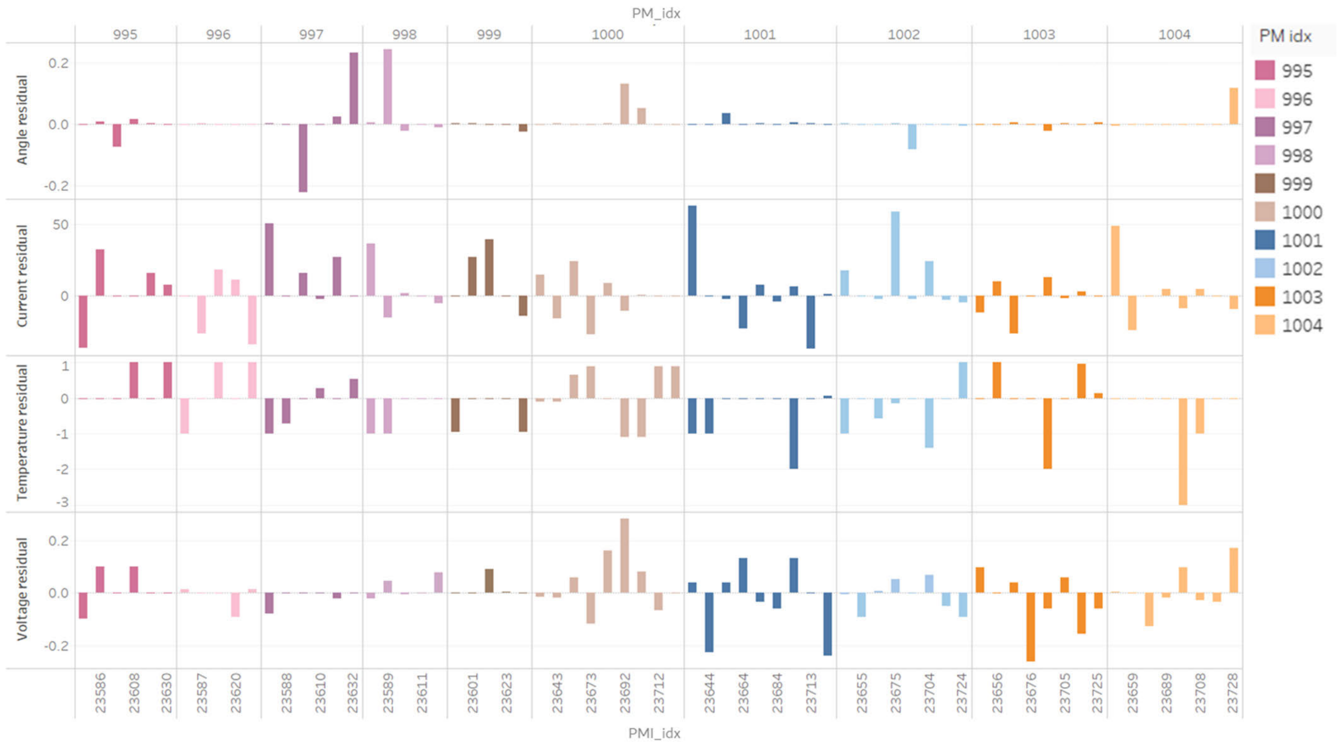


FIGURE 24. Results of motion residuals in PM (J4).

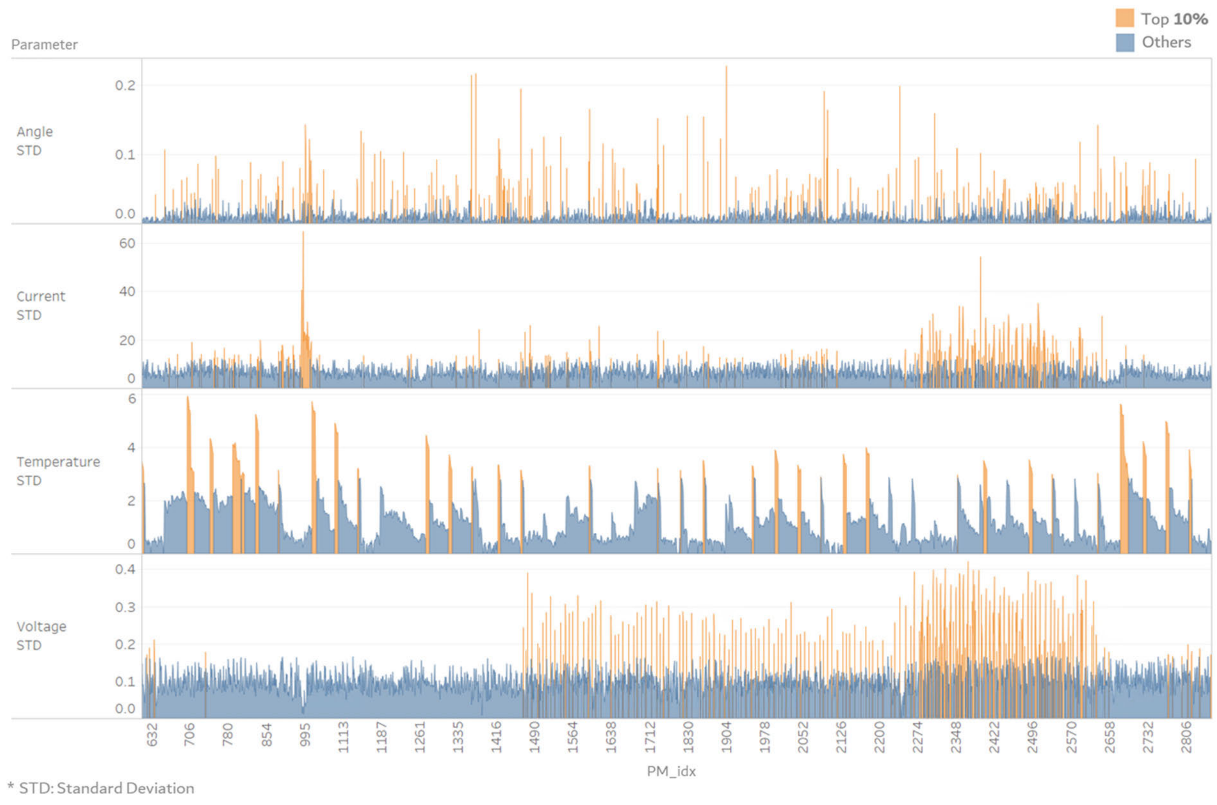


**FIGURE 25.** Variance analysis motion residuals by PM_idx (J4).

with a large difference in variance between PM_idx(1650, 2412) with the same PM_id(11). When PM_idx is 1650 and 2412, the mean and standard deviation of the motion residuals are (586.5, 0.3) and (597.1, 5.019), respectively. As shown in Figure 26, we can find that a continuous anomaly occurs with a large deviation in the execution cycle. Therefore, we used variance filtering to detect the PM anomalies.

1) **PMI anomaly detection**: Finally, a motion execution index (PMI_idx) with an abnormality for the selected PM_idx was detected. At this time, we should experimentally determine the index for which the residual value calculated for each PMI_idx exceeds a certain threshold. In this study, we performed the experiment shown in Fig. 26 to determine the optimal threshold. Specifically, we identified the detection error of residual values exceeding 5%, 10%, 15%, 20%, and 25% compared to the ideal value of zero. In this study, we set 15% as the abnormal PMI_idx detection threshold so that the number of indexes detected for each joint and parameter did not exceed 5% of the total execution indexes. This is a simple configuration and is necessary to determine an ideal criterion for a specific application. An operator who wants to strictly check the cobot's behavior can lower the threshold of the anomaly level, and an operator who wants to check loosely can increase the threshold. For example, the inspector can loosely check a cobot that manufactures a car-mat by gluing. Because the pasting range is wide, it can be neglected to some incorrect positions even if the exact location is not reached. In contrast, inspectors must rigorously inspect a cobot that assembles the tiny chips or semiconductors. Even a small error can be costly because it requires very sophisticated work.

The abnormal PMI_idxs detected with an error threshold of 15% are summarized in Table 2. Table 2 lists the incidence rates of anomalies for each parameter and joint according to the work program (P1 to P5). Over all executions (57,855 PMI_idxs), we found an anomaly rate of 5.01%. Specifically, in the cases of angle, current, and voltage, the closer it is to the end-effector (six axes), the higher the detection rate of abnormal sections is. This result is because the executed program is the gluing task implemented by the end effector and the 6-axis (J6) motion error was relatively large because of the load (weight) on the end effector. In contrast, with respect to temperature, we found no anomalies during the simple operations. This is because overheating can occur at high ambient temperatures or excessive program operation. However, our experiments did not reach this level of conditions.

Overall, in the case of the angular parameter, programmable motion errors of all axes can be checked.

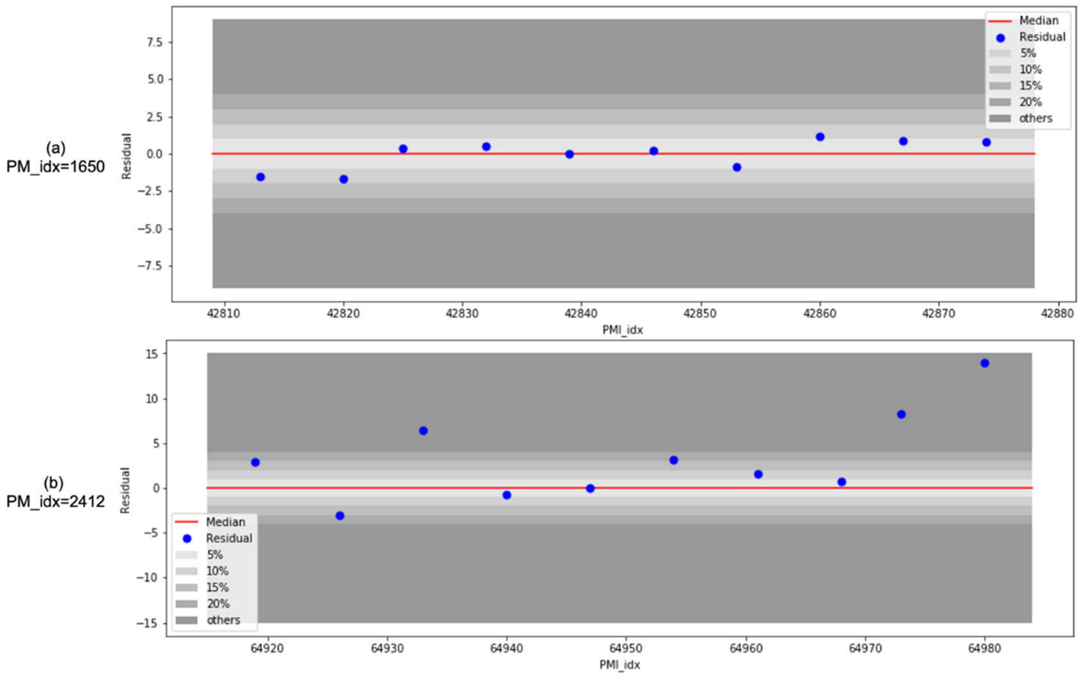


**FIGURE 26.** The threshold variation of motion residuals for parameter = angle, joint = J4, and PM_idx = {1650, 2412}.

**TABLE 2.** Anomaly Detection based on Motion Residuals.

| Joint | Parameter | Anomalies (%) | | | | |
|---|---|---|---|---|---|---|
| | | (P1) | (P2) | (P3) | (P4) | (P5) |
| 1 | Angle | 0.929 | 0.713 | 0.463 | 0.290 | 0.487 |
| | Current | 0 | 0 | 0 | 0 | 0 |
| | Voltage | 0 | 0 | 0 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |
| 2 | Angle | 0.538 | 0.324 | 0.219 | 0.186 | 0.275 |
| | Current | 0 | 0 | 0 | 0 | 0 |
| | Voltage | 0 | 0 | 0 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |
| 3 | Angle | 0.211 | 0.324 | 0.294 | 0.228 | 0.392 |
| | Current | 0 | 0 | 0 | 0.014 | 0 |
| | Voltage | 0 | 0 | 0 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |
| 4 | Angle | 1.003 | 0.735 | 0.412 | 0.276 | 0.385 |
| | Current | 0.802 | 0.756 | 0.589 | 0.435 | 0.235 |
| | Voltage | 0 | 0 | 0 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |
| 5 | Angle | 0.053 | 0.043 | 0.042 | 0.014 | 0.055 |
| | Current | 1.330 | 1.707 | 1.211 | 1.408 | 0.761 |
| | Voltage | 0.032 | 0 | 0.008 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |
| 6 | Angle | 2.164 | 1.944 | 1.455 | 1.808 | 1.068 |
| | Current | 2.417 | 3.478 | 5.012 | 3.630 | 3.744 |
| | Voltage | 0 | 0 | 0 | 0 | 0 |
| | Temperature | 0 | 0 | 0 | 0 | 0 |

Therefore, it can be concluded that using the angular variable to check for motion-fault has a high priority.

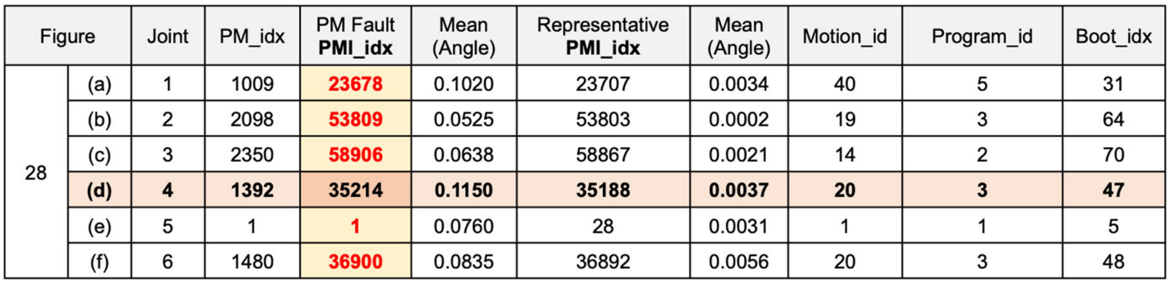
### D. DISCUSSION

In this section, the experimental results are described in terms of three goals: 1) hierarchical analysis of execution, 2) securing traceability for the causes of faults, and 3) establishing an abnormality determination criterion.

- **Hierarchical analysis of execution:** PM execution information can be analyzed as shown in Fig. 23 by relating data in the order of "boot–program–motion." In the example in Fig. 23, one can see that there are a total of five programs representing approximately 1700 operations over the 33 job sections executed on day two. Each program has a different number of instructions. For example, Program 1 has seven motion commands that are configured and executed. The proposed index system can hierarchically identify programmable motions.
- **Securing traceability for the causes of faults:** Fig. 27 presents detailed explanations regarding the traceability of each anomaly, and Figs. 28 presents the angle anomalies extracted for each joint. We can trace the extracted abnormal section to the PM command for which the abnormality appeared by applying the proposed indexing method. For example, PMI_idx 35214 (green line) in Fig, 28(d) exhibits a data pattern with more significant errors than other PMI_idxs corresponding to the same PM. Therefore, we can trace the responsible motion and program based on

the anomaly, as shown in Fig. 27. The PM_idx connected to the PMI_idx 35214 was 1392. Based on this information, we can match the corresponding PM index. In the example of index 35214, we can determine that the executed motion index is 20, the program index is 3, and the boot index is 47. These processes provide traceability for the type of work program and execution behavior over detected abnormal sections, allowing us to prevent faults by analyzing the work that causes faults and improving fault detection.

- **Establishing a programmable motion-fault detection criterion**: We grouped each executed motion that executes the same PM type to form a dataset that can represent abnormalities. We also the selected execution sections with representative patterns. The thick lines from (a) to (f) in Fig. 28 represent execution indexes with high motion residual values for each execution based on typical patterns. A detailed causal analysis of each abnormal section is presented below.
  (a) **PMI_idx = 23678 (J1)**: After identifying a typical pattern, the graph is cut in the middle, and we can determine that an interruption occurs during execution.
  (b) **PMI_idx = 53809 (J2)**: In another operation sections, the angle changes were small. An abnormality was identified based on external shaking during the forward motion.
  (c) **PMI_idx = 58906 (J3)**: As shown in Fig. 28(b), an anomaly is caused by external vibration in a static motion state.
  (d) **PMI_idx = 35214 (J4)**: We can identify Fig. 28(d) as an error based on the difference between the initial position values.
  **PMI_idx = 1 (J5)**: Because the graph of the typical pattern appears to have been shortened



| Figure | | Joint | PM_idx | PM Fault **PMI_idx** | Mean (Angle) | Representative **PMI_idx** | Mean (Angle) | Motion_id | Program_id | Boot_idx |
|---|---|---|---|---|---|---|---|---|---|---|
| 28 | (a) | 1 | 1009 | **23678** | 0.1020 | 23707 | 0.0034 | 40 | 5 | 31 |
| | (b) | 2 | 2098 | **53809** | 0.0525 | 53803 | 0.0002 | 19 | 3 | 64 |
| | (c) | 3 | 2350 | **58906** | 0.0638 | 58867 | 0.0021 | 14 | 2 | 70 |
| | **(d)** | **4** | **1392** | **35214** | **0.1150** | **35188** | **0.0037** | **20** | **3** | **47** |
| | (e) | 5 | 1 | **1** | 0.0760 | 28 | 0.0031 | 1 | 1 | 5 |
| | (f) | 6 | 1480 | **36900** | 0.0835 | 36892 | 0.0056 | 20 | 3 | 48 |

**FIGURE 27.** **The PM identification of angle anomalies.**

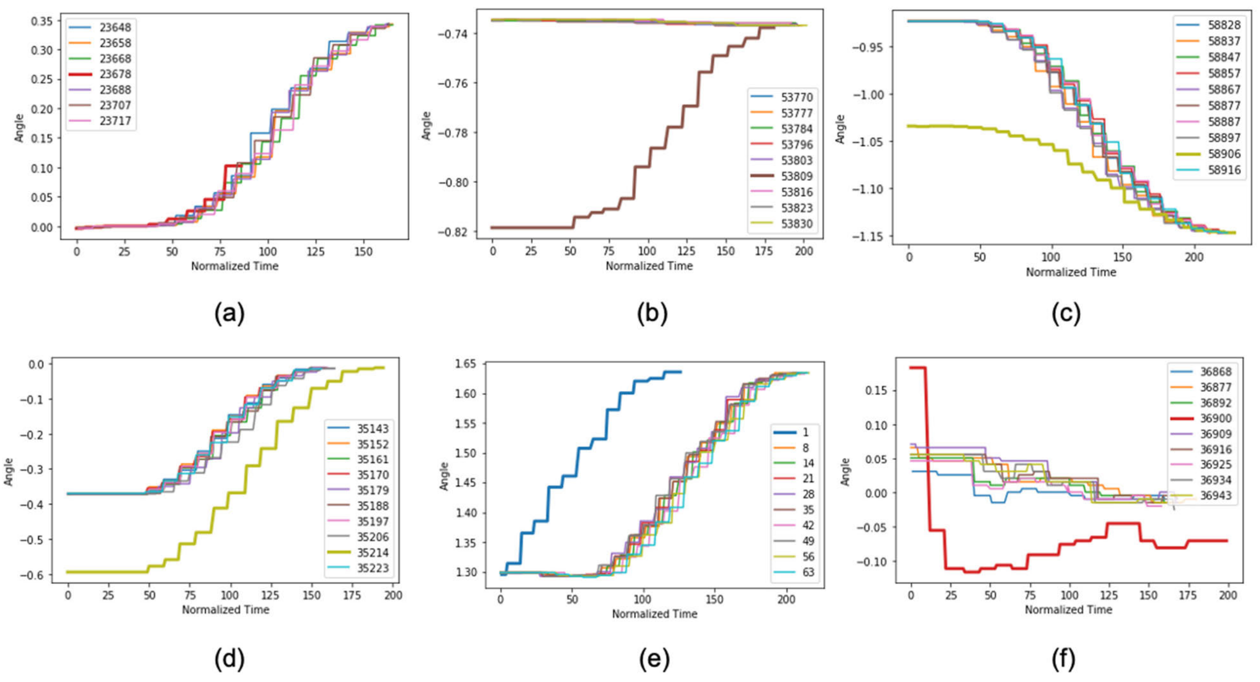


**FIGURE 28.** **Angle anomalies of each joint (J1 to J6).**

within a relatively short timeframe, we can judge that there is no problem reaching the location, but the execution time is short, representing an unstable operation.

(e) **PMI_idx = 36900 (J6)**: This can be deduced as a situation in which the initial position error and execution definition instability are revealed

Fig. 29 and 30 show an example of a programmable motion-fault detection. Fig. 29 shows a graph of the J3 angle with time. The green line is the normal pattern of representative execution, and the gray line represents the execution of the program index (299) holding the PMI anomaly on the 3-axis (J3). In Fig. 29, the execution of program index 299 was delayed and later finished owing to the error depicted by the red line. As shown in Fig. 30, the area of the green line occupies 234713.5 pixels, and the area of the gray line including an anomaly does 208249.0 pixels, and the difference is 26464.5 pixels. In other words, we can say that the 299th program execution commanded the same programmable motion, but with a higher level of error compared to the normal gluing task. This situation can be called a programmable motion-fault situation. To experiment programmable motion-fault, we overloaded the robot above a specific level, making a faulty situation for normal program execution 189 times and fault-causing program execution 68 times. The programmable motion-fault detection was successful with an accuracy of 92.6%.

In addition, we compared anomalies with and without the PM to validate the effect of PM identification. In Fig. 29 and 30, the red line represents the anomaly

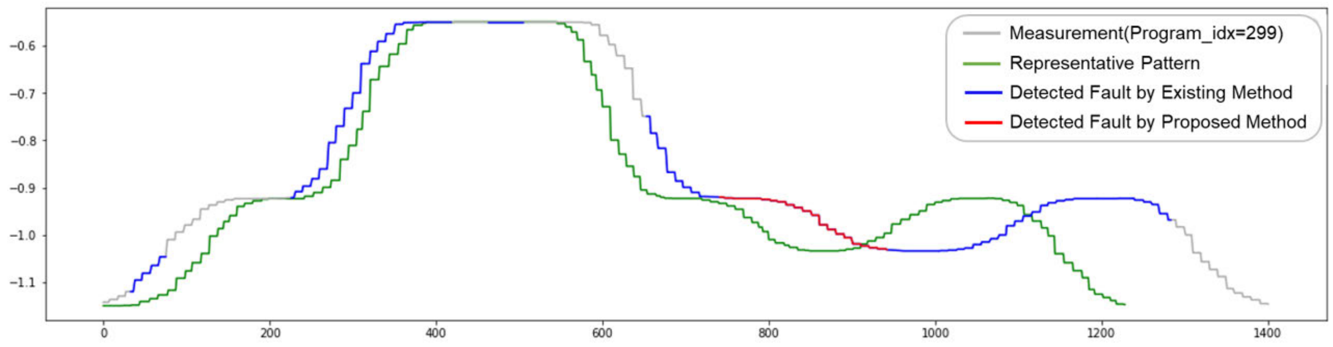


**FIGURE 29.** An example of programmable motion-fault.

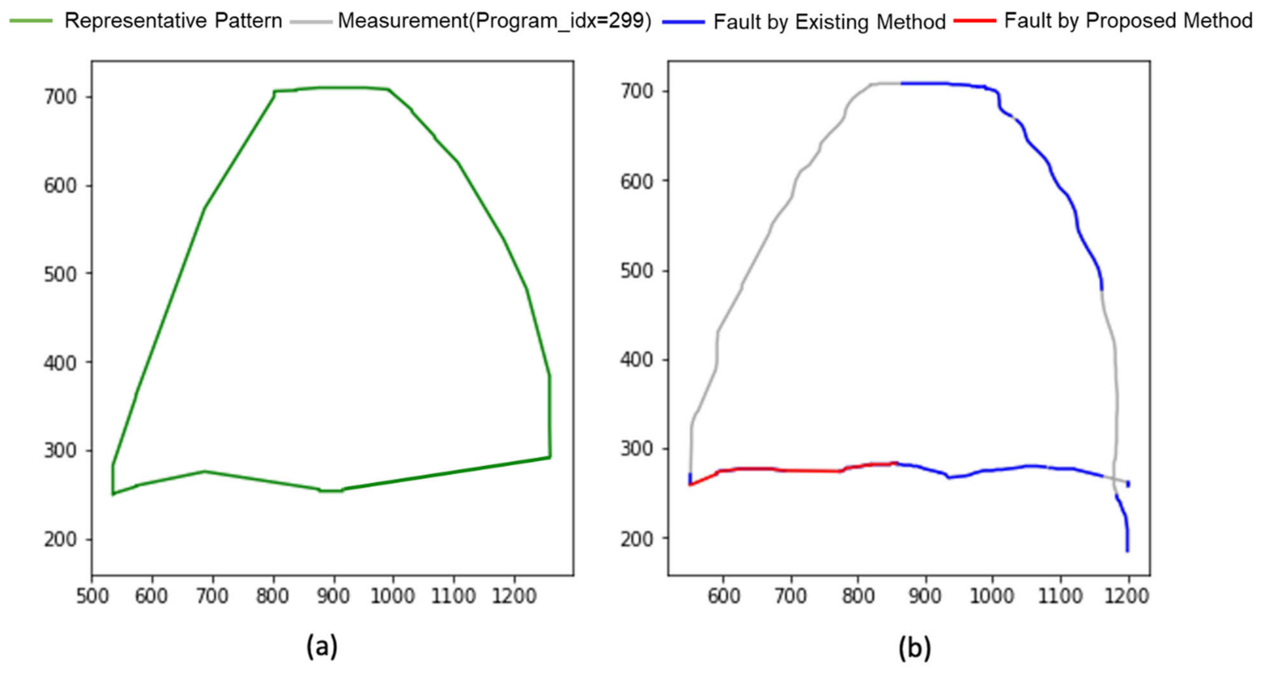


**FIGURE 30.** An example of gluing task failure due to programmable motion-fault on J3.

detected in PM, and the blue line represents the anomaly detected in a specific window through sequential comparisons. As a result, detection using a static window (40 ticks: 0.4 sec) showed a false-positive error interval that was 27.3% higher than that of the proposed method as shown in Fig. 29 and 30. Without PM identification, it is sensitive to noise and detects anomalies compared to other unequal motions.

## VII. CONCLUSION

This paper proposes a programmable motion-fault detection method based on the motion residual analysis of cobots. We constructed a data analysis model to identify the fault causes during programmable motions, which are the most significant feature of a cobot. Our method overcomes the limitations of existing fault diagnosis methods that can only be applied to limited operations and enable the following tasks. We can establish absolute standards for the programmable motion-fault analysis of cobots, interpret the meaning of detected values, and analyze the causes of faults. Therefore, it is possible to analyze the weak points in the operator work programs. Additionally, effective predictive maintenance is facilitated by expanding the fault diagnosis range of the cobots. However, the proposed method was tested on a dataset that simulated a manufacturing environment. Therefore, our fault diagnosis verification may not accurately represent a real–world industrial facility environment. We plan to expand the scope of data collection in the future, perform verification and validation in industrial settings, and construct a statistical or learning–based anomaly detection model.

## REFERENCES

[1] Z. Li, Y. Wang, and K.-S. Wang, "Intelligent predictive maintenance for fault diagnosis and prognosis in machine centers: Industry 4.0 scenario," *Adv. Manuf.*, vol. 5, no. 4, pp. 377–387, 2017.

[2] J. Wang, L. Zhang, L. Duan, and R. X. Gao, "A new paradigm of cloud-based predictive maintenance for intelligent manufacturing," *J. Intell. Manuf.*, vol. 28, no. 5, pp. 1125–1137, Jun. 2017.

[3] F. Ansari, P. Hold, W. Mayrhofer, S. Schlund, and W. Sihn, "AUTODIDACT: Introducing the concept of mutual learning into a smart factory industry 4.0," in *Proc. 15th Int. Conf. Cognition Explor. Learn. Digit. Age (CELDA)*, Oct. 2018, pp. 61–68.

[4] A. Rezaeipanah, H. Nazari, and G. Ahmadi, "A hybrid approach for prolonging lifetime of wireless sensor networks using genetic algorithm and online clustering," *J. Comput. Sci. Eng.*, vol. 13, no. 4, pp. 163–174, Dec. 2019.

[5] J. M. Ha, H. Oh, J. Park, and B. D. Youn, "Classification of operating conditions of wind turbines for a class-wise condition monitoring strategy," *Renew. Energy*, vol. 103, pp. 594–605, Apr. 2017.

[6] W. Choi, B. D. Youn, H. Oh, and N. H. Kim, "A Bayesian approach for a damage growth model using sporadically measured and heterogeneous on-site data from a steam turbine," *Rel. Eng. Syst. Saf.*, vol. 184, pp. 137–150, Mar. 2018.

[7] J. M. Ha, J. Park, K. Na, Y. Kim, and B. D. Youn, "Toothwise fault identification for a planetary gearbox based on a health data map," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5903–5912, Jul. 2018.

[8] S. Xue and I. Howard, "Torsional vibration signal analysis as a diagnostic tool for planetary gear fault detection," *Mech. Syst. Signal Process.*, vol. 100, pp. 706–728, Feb. 2018.

[9] Y. Kim, J. Park, K. Na, H. Yuan, B. D. Youn, and C.-S. Kang, "Phase-based time domain averaging (PTDA) for fault detection of a gearbox in an industrial robot using vibration signals," *Mech. Syst. Signal Process.*, vol. 138, Apr. 2020, Art. no. 106544.

[10] Y. Zhang, H. An, X. Ding, W. Liang, M. Yuan, C. Ji, and J. Tan, "Industrial robot rotate vector reducer fault detection based on hidden Markov models," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Dec. 2019, pp. 3013–3018.

[11] A. A. Jaber and R. Bicker, "Fault diagnosis of industrial robot gears based on discrete wavelet transform and artificial neural network," *Insight-Non-Destructive Test. Condition Monitor.*, vol. 58, no. 4, pp. 179–186, Apr. 2016.

[12] F. Cheng, A. Raghavan, D. Jung, Y. Sasaki, and Y. Tajika, "High-accuracy unsupervised fault detection of industrial robots using current signal analysis," in *Proc. IEEE Int. Conf. Prognostics Health Manage. (ICPHM)*, Jun. 2019, pp. 1–8.

[13] A. I. Karoly, J. Kuti, and P. Galambos, "Unsupervised real-time classification of cycle stages in collaborative robot applications," in *Proc. IEEE 16th World Symp. Appl. Mach. Intell. Informat. (SAMI)*, Feb. 2018, pp. 000097–000102.

[14] J. Zheng, H. Pan, and J. Cheng, "Rolling bearing fault detection and diagnosis based on composite multiscale fuzzy entropy and ensemble support vector machines," *Mech. Syst. Signal Process.*, vol. 85, pp. 746–759, Feb. 2017.

[15] D. Yao, J. Yang, Y. Bai, and X. Cheng, "Railway rolling bearing fault diagnosis based on multi-scale intrinsic mode function permutation entropy and extreme learning machine classifier," *Adv. Mech. Eng.*, vol. 8, no. 10, 2016, Art. no. 1687814016676157.

[16] Q. Tong, J. Cao, B. Han, X. Zhang, Z. Nie, J. Wang, Y. Lin, and W. Zhang, "A fault diagnosis approach for rolling element bearings based on RSGWPT-LCD bilayer screening and extreme learning machine," *IEEE Access*, vol. 5, pp. 5515–5530, 2017.

[17] J. Park, M. Hamadache, J. M. Ha, Y. Kim, K. Na, and B. D. Youn, "A positive energy residual (PER) based planetary gear fault detection method under variable speed conditions," *Mech. Syst. Signal Process.*, vol. 117, pp. 347–360, Feb. 2019.

[18] H. Oh, H. Choi, J. H. Jung, and B. D. Youn, "A robust and convex metric for unconstrained optimization in statistical model calibration—Probability residual (PR)," *Struct. Multidisciplinary Optim.*, vol. 60, no. 3, pp. 1171–1187, 2019.

[19] J. Park, Y. Kim, K. Na, and B. D. Youn, "Variance of energy residual (VER): An efficient method for planetary gear fault detection under variable-speed conditions," *J. Sound Vib.*, vol. 453, pp. 253–267, Aug. 2019.

[20] M. Yoo, T. Kim, J. T. Yoon, Y. Kim, S. Kim, and B. D. Youn, "A resilience measure formulation that considers sensor faults," *Rel. Eng. Syst. Saf.*, vol. 199, Jul. 2020, Art. no. 106393.

[21] T. Han, C. Liu, W. Yang, and D. Jiang, "Deep transfer network with joint distribution adaptation: A new intelligent fault diagnosis framework for industry application," *ISA Trans.*, vol. 97, pp. 269–281, Feb. 2020.

[22] H. Zhang and Y. Deng, "Weighted belief function of sensor data fusion in engine fault diagnosis," *Soft Comput.*, vol. 24, no. 3, pp. 2329–2339, Feb. 2020.

[23] A. T. de Almeida, C. A. V. Cavalcante, M. H. Alencar, R. J. P. Ferreira, A. T. de Almeida-Filho, and T. V. Garcez, "Preventive maintenance decisions," in *Multicriteria and Multiobjective Models for Risk, Reliability and Maintenance Decision Analysis*. Cham, Switzerland: Springer, 2015, pp. 215–232.

[24] M. Hamadache, J. H. Jung, J. Park, and B. D. Youn, "A comprehensive review of artificial intelligence-based approaches for rolling element bearing PHM: Shallow and deep learning," *JMST Adv.*, vol. 1, nos. 1–2, pp. 125–151, Jun. 2019.

[25] J. Chen and R. J. Patton, *Robust Model-Based Fault Diagnosis for Dynamic Systems*, vol. 3. Springer, 2012.

[26] J. Che, Y. Zhu, and D. Zhou, "Hidden Markov model-based robust $H_\infty$ fault estimation for Markov switching systems with application to a single-link robot arm," *Asian J. Control*, pp. 1–12, Jan. 2021.

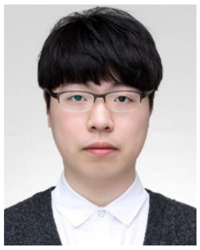
[27] A. Kouadri, M. Hajji, M.-F. Harkat, K. Abodayeh, M. Mansouri, H. Nounou, and M. Nounou, "Hidden Markov model based principal component analysis for intelligent fault diagnosis of wind energy converter systems," *Renew. Energy*, vol. 150, pp. 598–606, May 2020.

[28] K. Liu, H. Lin, Z. Fei, and J. Liang, "Spatially–temporally online fault detection using timed multivariate statistical logic," *Eng. Appl. Artif. Intell.*, vol. 65, pp. 51–59, Oct. 2017.

[29] C. P. Mbo'o and K. Hameyer, "Fault diagnosis of bearing damage by means of the linear discriminant analysis of stator current features from the frequency selection," *IEEE Trans. Ind. Appl.*, vol. 52, no. 5, pp. 3861–3868, Sep./Oct. 2016.

[30] X. Zhang, Y. Liang, and J. Zhou, "A novel bearing fault diagnosis model integrated permutation entropy, ensemble empirical mode decomposition and optimized SVM," *Measurement*, vol. 69, pp. 164–179, Jun. 2015.
[31] Y. Xie and T. Zhang, "Fault diagnosis for rotating machinery based on convolutional neural network and empirical mode decomposition," *Shock Vib.*, vol. 2017, Aug. 2017, Art. no. 3084197.
[32] L. Guo, N. Li, F. Jia, Y. Lei, and J. Lin, "A recurrent neural network based health indicator for remaining useful life prediction of bearings," *Neurocomputing*, vol. 240, pp. 98–109, May 2017.
[33] *Robots and Robotic Devices–Collaborative Robots*, document ISO/TS 15066:2016, 2016.
[34] M. J. Rosenstrauch and J. Krüger, "Safe human-robot-collaboration-introduction and experiment using ISO/TS 15066," in *Proc. 3rd Int. Conf. Control, Autom. Robot. (ICCAR)*, Apr. 2017, pp. 740–744.
[35] D. Antonelli and S. Astanin, "Qualification of a collaborative human-robot welding cell," *Proc. CIRP*, vol. 41, pp. 352–357, Jan. 2016.
[36] T. Sun, D. Zhou, Y. Zhu, and M. V. Basin, "Stability, $l_2$-gain analysis, and parity space-based fault detection for discrete-time switched systems under dwell-time switching," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 9, pp. 3358–3368, Sep. 2020.
[37] (2020). *Niryo One—An Accessible Educational 6 Axis Robotic Arm, Just for You–Niryo*. Niryo. Accessed: Oct. 15, 2019. [Online]. Available: https://niryo.com/niryo-one/

**YE-SEUL PARK** (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Ajou University, Suwon, South Korea, in 2015 and 2017, respectively, where she is currently pursuing the Ph.D. degree with the Department of AI Convergence Network. Her research interests include fault prognosis, predictive maintenance, knowledge engineering, artificial intelligence, big data modeling and analysis, and collaborative robot in smart factory.



**DONG-YEON YOO** (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Ajou University, Suwon, South Korea, in 2019 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the Department of AI Convergence Network. His research interests include statistics-based sensor data anomaly detection and fault detection for predictive maintenance.

**JUNG-WON LEE** (Member, IEEE) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in computer engineering from Ewha Womans University, Seoul, South Korea, in 1993, 1995, and 2003, respectively.

She worked as a Research Engineer with LG Electronics, from 1995 to 1997. She completed an internship at IBM Almaden Research Center, Data Mining Group, San Jose, CA, USA, in 2000. She was a Research Professor and a full-time Lecturer with Ewha Womans University, from 2003 to 2006. In 2006, she joined the School of Electrical and Computer Engineering and AI Convergence Network, Ajou University, Suwon, South Korea. Her research interests include context awareness, big data analysis and predictive maintenance, collaborative robots, and intelligent embedded software.