# An Empirical Study on Fault Diagnosis in Robotic Systems

Xuezhi Song, Yi Li*, Zhen Dong*, Shuning Liu, Junming Cao, Xin Peng
{songxuezhi,liy,zhendong,liushuning,xinpeng}@fudan.edu.cn, jamescao2048@gmail.com
School of Computer Science
*Fudan University*
Shanghai Key Laboratory of Data Science
*Fudan University*
Shanghai, China

***Abstract*—Fault diagnosis in robotic systems is challenging due to their complex and heterogeneous structures and complex interactions with physical environments. Given the complexities and uncertainties, we think it may be helpful to diagnose faults of a robotic system by understanding its behaviors from the perspective of observability. In this paper, we conduct an empirical study to explore the efficacy of combining different kinds of common observability data (i.e., logs, traces, and trajectories) for fault diagnosis in robotic systems. In the study, we investigate root causes of 398 bug cases in robotic systems to understand their characteristics. Furthermore, we replicate 23 bugs out of them and perform a fault diagnosis study in which participants diagnose each of the replicated bug with only observability data and record how useful observability data is. The bug case analysis study revealed that the root causes of bugs in robotic systems originate from various levels, including physical environment interaction (11.81%), hardware usage (14.82%), software implementation (49.25%), and system configuration (24.12%). The fault diagnosis study shows the combination of trace and trajectory data improves the fault diagnosis success rate by 58.33% and 8.33%, respectively, compared to using only logs. Our study promotes the vision of observability-based fault diagnosis in robotic systems.**

***Index Terms*—robotic system, ROS, debugging, fault diagnosis, observability**

## I. Introduction

Autonomous robots have been increasingly used in different areas such as agriculture, manufacturing, logistics, medical treatment, and services to automate various tasks that are critical to the users [66]. For example, logistics robots are used to move goods to support the operation of warehouses; mobile robots are used to deliver food to customers in restaurants. In these applications, faults may affect the robot's efficiency, cause failures, or even jeopardize the safety of the robot or its surroundings [27]. Therefore, when a fault is detected it is imperative to proceed with a diagnosis process to identify the root causes to enable fault recovery or decision making such as using undamaged redundant components or re-planning [44], [65].

Fault diagnosis in robotic systems is challenging due to their complex and heterogeneous structures and complex interactions with physical environments. A robot may comprise a significant number of hardware and software components that are quite heterogeneous in their structure and functionality [76]. Nowadays, Robot Operating System (ROS) [60], [61] has been widely used for the development and operation of robotic systems in both academia and industry. A ROS-based robotic system is a typical distributed system where the applications are implemented as a set of nodes that perform various computation and interact using distributed communication. Moreover, a robot and its components closely interact with dynamic physical environments, thus are frequently subject to faults caused by various problems like wear, damage, unexpected environmental changes, or design and implementation flaws [76], the fault diagnosis is complicated by the large number of heterogeneous hardware and software components and the uncertainties in their distributed communication and environment interactions.

It shows that most robotics systems frequently contain bugs [70]. The bugs are investigated for comprehensive understanding to take further action. Zampetti et al. [77] classify 22 different root causes of bugs in cyber-physical systems (CPSs) and group them into 8 high-level categories, including hardware, network, interface, data, configuration, algorithm, documentation and others. For unmanned aerial vehicles (UAVs) or drones, main hazards and accidents [28] respectively are classified into 19 and 7 categories respectively from reported safety issues. For autonomous vehicle (AV), Garcia et al. [30] investigate bugs from aspects of root cause, symptom, and affected AV component. Fischer-Nielsen et al. [29] empirically study dependency bugs in ROS, which are caused by configuration error due to the complex relationships of different components, and classify them by location, dependency fault, and failure. Our research focuses on runtime bugs of ROS-based robotic systems. We classify root causes and their impacts from the perspective of observability guided by the MAPE-K [43] reference model.

In current practice of ROS-based robot development, the developers highly depend on messages and logs collected from different nodes for fault diagnosis. However, messages and logs only reflect local behaviors and states of specific nodes and cannot provide information about execution processes through different nodes. Moreover, they cannot provide

Co-corresponding author

information about interactions with physical environments, e.g., point clouds, robot poses, and trajectories. Model-based diagnosis approaches compare the observed behaviors of a robot to the expected behaviors based on an explicit prior model of the normal system behaviors, structures, and/or known faults [44], [76]. These approaches may possess a global view of the robot, but it is often hard to construct a complete prior model that covers both internal component interactions and external environment interactions.

Given the complexities and uncertainties, we think it may be helpful to understand the behaviors of a robotic system and diagnose its faults from the perspective of observability. The term "observability" originates in control system theory and measures the degree to which a system's internal state can be determined from its output [35], [55]. For modern distributed systems such as microservice systems, observability usually means the ability of monitoring and understanding the internal states and execution processes of a system based on logs, traces, and metrics [48], [67]. For diagnose of a fault robotic system it may also be feasible and helpful to combine different kinds of observability data, which include log, trace and trajectory.

- **Logs** record events that occur in the operating system or applications, which are widely used for system debug.
- **Traces** are produced by distributed tracing and record the execution processes through different nodes.
- **Trajectories** are produced based on sensor data, robot model, and environment maps and reflect the trajectories of robots in physical environments.

In particular, we are concerned with whether traces and trajectories can be combined with logs for more effective fault diagnose.

In this paper, we conduct an empirical study on fault diagnosis in robotic systems to answer the following research questions.

- **RQ1**. What are the root causes of bugs and their impacts in robotic systems?
- **RQ2**. How can observability data be utilized in fault diagnosis of robotic systems.

To answer RQ1, we conduct a bug case analysis to manually analyze 398 bug cases reported in real-world robotic systems and study their root causes and impact on the robotic systems. To answer RQ2, we replicate 23 bug cases in a robotic application and recruit five graduate students to accomplish fault diagnosis tasks with the help of logs, traces, and trajectories.

The results of the bug case analysis indicate that root causes of robotic system bugs originate from different levels of the system, including environment interaction (11.81%), hardware usage (14.82%), software implementation (49.25%), and system configuration (24.12%). Furthermore, 91.45% of the bugs cause direct malfunction, while 8.54% result in performance degradation. The results of the fault diagnosis study show that, among the 84 (out of 115 tasks) successful tasks, 33.33% use log-based analysis, 58.33% further use trace-based analysis, and 8.33% further use trajectory-based analysis. The results confirm that the combination of traces and trajectory data with logs can significantly improve the effectiveness of fault diagnosis when the faults involve interactions among different nodes or with physical environments. Our study also reveals the requirements of infrastructures and tools for the collection, analysis, and visualization of observability data.

This research paper presents the following contributions:

- We conduct a bug case analysis of 398 real-world robotic system bugs to understand their root causes and impact on the systems, providing insights for fault diagnosis in robotic systems.
- We replicate 23 bug cases in a robotic application and recruit five graduate students to perform fault diagnosis tasks using logs, traces, and trajectories, demonstrating the effectiveness of combining different types of observability data for fault diagnosis in robotic systems.
- Our study highlights the requirements for infrastructures and tools needed for the collection, analysis, and visualization of observability data in robotic systems.
- The results of our research provide valuable guidance for practitioners in the field of robotic systems and fault diagnosis, as well as informing the development of more efficient diagnostic techniques and tools.

The rest of the paper is structured as follows. After some background in Section II: Section III introduces our study methodology; Section IV illustrates our results and findings; Section V states some threats; Section VI discusses related works. Concluding remarks are in Section VII.

## II. Background

A robotic system consists of various physical (hardware) components (e.g., camera, GPS sensor, arm, gripper) and cyber (software) components (e.g., a control algorithm) and interacts with the environment (e.g., a terrain) [46]. Typically, a robotic system has a layered design consisting of hardware layer, control layer, and application layer. A control layer implements a collection of drivers that manipulate and interact with hardware components; while an application layer implements more complex and specialized functionalities by utilizing the control layer to integrate and coordinate hardware components [21].

**Robot Operating System (ROS)**. ROS is an open-source robotic software framework and middleware and the latest version is ROS 2. ROS provides a set of software libraries (e.g., drivers, algorithms) and developer tools for building robot applications. It features a message-passing scheme for distributed robot processes, hardware abstraction, development tools (e.g., simulator), and robotic libraries (e.g., path planning algorithm) [46]. A ROS application can be implemented as a set of nodes each of which is a process that performs computation. For example, there can be nodes analyzing recorded images, planning paths, controlling wheel motors, and providing graphical view of the system. These nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the parameter server [61]. Therefore, a ROS-based robotic system is usually a distributed system where nodes are deployed in different devices and interact with distributed communication.

**Fault Diagnosis in ROS**. For ROS-based robotic systems the developers highly depend on logs for fault diagnosis. ROS provides a generic logging functionality that records information about its own status, sensor data, and messages between nodes. Moreover, the developers can record the internal status of a node using a logger. ROS provides a topic-based mechanism called rosout for reporting log messages and a message mechanism called rosbag to filter and record specific topic messages. To facilitate the monitoring of the runtime state of a robotic system, ROS provides a visualization program called *rviz*. Its visualization panels can be dynamically instantiated to view a large variety of sensor data, e.g., images, point clouds, geometric primitives, robot poses and trajectories [60]. *rviz* implements a plugin architecture, thus visualization plugins can be easily written to interpret and display different types of data. ROS also provides a simple fault diagnostic system [76]. The diagnostics system is designed to collect information from hardware drivers and robot hardware to users and operators for analysis, troubleshooting, and logging [62]. The system contains a tool chain for collecting, publishing, analyzing and viewing diagnostics data. The tool chain is built around a specific topic ("/diagnostics"), on which hardware drivers and devices publish diagnostic messages with information like device names, status and specific data points. Therefore, the diagnostics system can only deal with faults related to specific hardware or hardware drivers that publish data on the topic.

**Distributed Tracing**. In modern distributed systems such as microservice systems, distributed tracing has been widely used to record and understand the execution processes of the systems through different services and compute nodes [48]. For a robotic system distributed tracing can also be used to observe and analyze the service invocation and interaction processes through different nodes in ROS. A trace represents a series of causally related distributed events that encode the end-to-end request workflow through a distributed system [67]. According to the OpenTracing specification [16], a trace consists of a set of spans organized in a tree structure and each span represents a service interaction via service invocation or topic message. Open-source systems such as Skywalking [9], Zipkin [20], and Jaeger [12] follow the OpenTracing framework and provide the distributed tracing infrastructures distributed systems.

## III. Study Design

Our empirical study is planned as two stages: bug case analysis and fault diagnosis study, which correspond to **RQ1** and **RQ2** respectively. In the first stage, we investigate the characteristics of collected bugs in robotic systems, and prepare a fair dataset for the second stage based on bug characteristics. Then the dataset is used to evaluate different methods for fault diagnosis in robotic systems.

### A. *Bug Case Analysis*

To gain full understanding of robotic system bugs, we collect a set of bug cases reported in real robotic systems, and analyze bug cases manually to study their root causes and corresponding impacts on robotic systems.

*1) Dataset:* We have two sources available for our study: a public dataset Robust [17] and a dataset ARSB (Anonymous Robotic System Bugs), which is built upon bugs of a company focusing on developing robotic systems.

**Robust**. ROBUST is an output of a project funded by the European Union's Horizon 2020 research and innovation programme. It has been used in previous work on ROS bug analysis [29]. The dataset contains 219 ROS bugs collected from eight widely used ROS packages by manually analyzing 1,790 bug issues. The dataset statistics are shown in Table I, where the first two columns denote names and descriptions of the ROS packages; the third column indicates the number of Github stars; the fourth column shows the number of issues that were analyzed during bug collection; the last column reveals the number of robotic system bugs collected from these ROS packages. As shown in the table, the dataset is well created and maintained in terms of the popularity of collected subjects (with 316 Github stars on average) and bug distribution among them. Moreover, each bug in the dataset is well described, including reproducing steps beneficial to our fault diagnosis study.

**ARSB**. This dataset has 263 records of robotic system bugs from the past two years, originating from an anonymous company specializing in automated delivery and hotel service robots, primarily developed using the ROS architecture. Each bug in the dataset has already been resolved and is accompanied by a bug report discussing the process from bug discovery to resolution, including symptom and solution details. Notably, each bug occurred in a user scenario, providing insights into real-world issues that arise with the robotic systems. For commitment to privacy protection, we cannot disclose any detailed data owned by the company. However, we can further abstract the bug information, including descriptions of the symptoms and their root causes. This data can be accessed on our anonymous website. Although this abstraction covers details, it still provides valuable insights regarding the bug characteristics and their underlying causes.

TABLE I: The Statistics of the ROBUST Dataset

| Package | Description | Stars | Issues | Bugs |
|---|---|---|---|---|
| MAVROS [14] | Communication protocol drivers of autopilots. | 653 | 325 | 57 |
| Kobuki [13] | Kobuki's driver and Ros tools | 191 | 623 | 40 |
| Universal Robot [19] | Control and communication nodes. | 765 | 158 | 25 |
| Motoman[15] | Motoman industrial robot controllers. | 115 | 78 | 22 |
| TurtleBot [18] | Basic drivers for TurtleBot running ROS. | 257 | 170 | 12 |
| Care-O-Bot [10] | Packages for low level control tasks. | 67 | 182 | 11 |
| geometry2 [11] | Coordinate transforms track ROS package. | 166 | 254 | 42 |
| Confidential | – | – | N/A | 10 |
| Ave/Sum | | 316 | 1790 | 219 |

*2) Preprocessing:* As our study focuses on fault diagnosis, we are particularly interested in robotic system bugs that result in runtime errors. To obtain such bugs, we manually examine each bug case in both ROBUST and ARSB datasets. Our examination procedure involves checking if a bug case is labeled with the runtime error type. If there is no label, we scrutinize the bug description, commit message, and related code to confirm whether the bug cause a runtime error. Ultimately, we select 135 bugs out of 219 from ROBUST and all 263 bugs from ARSB, for a total of 398 bug cases that cause runtime errors. These bugs are then used for further analysis in our study.

*3) Analysis:* We first collect a set of labels for root causes and categories of bugs based on previous work [28], [29], [30], [77]. However, we conduct this research from the perspective of observability. We try to categorize root causes to match different layers of robotic systems. We manually analyze these 398 bug cases to examine their root causes and impacts guided by MAPE-K [43] reference model. Actually, the labels for root causes and their categories frequently change during analysis until they sounds fair enough to us all.

We classify the root causes of bug cases by analyzing their bug reports and fix commits. Given a bug case, we first review the report description and subsequently discuss to gain a preliminary understanding of the bug. Then we further analyze the fix commit of the bug to understand the changes made for the bug fix. Based on the analysis we decide a type of root causes that the bug case belongs to. Furthermore, we analyze the overall impact of a bug from the perspective of users, for example performance downgrade, or malfunction. To this end we analyze the issue description and follow-up discussion to understand the overall impact.

## B. Fault Diagnosis Study

In this stage, we aim to evaluate how and when observability data can help us in fault diagnosis for robotic systems. We first select and replicate 23 bug cases from the ROBUST and ARSB. We then recruit several participants to diagnose faulty system with our methods. Finally, we analyze the recorded fault diagnosis processes to understand how different kinds of observability data are used in fault analysis.

*1) Bug Replication:* Reproducing a bug in robotic systems is difficult since it often requires an environment in which a robot can interact with the physical world. Such an environment is typically built with specified hardware and software. Different bugs may require different types of devices and different versions of applications. It is challenging to obtain various devices for bug reproduction. To overcome this challenge, we implement a food delivering system in which a robot can perform various activities, such as grasping a cup and delivering it to a costumer. Specifically, our delivering system consists of two TurtleBot2 robots and a JAKA robot arm. The detail configuration of these devices is shown in our replication package. On the system, we can *replicate* bugs reported in different systems based on their descriptions. By replicating a bug, it means we simulate it on our food delivering system that “works” in the exact same way as described in the bug report.

Notice, considering our limited resource and time, we replicate a portion of bugs in ROBUST and ARSB instead of all the 398 bugs. To make replicated bugs more representative, we take the following procedure for bug replication. As these 398 bugs have been classified into different categories in the bug case analysis study (8 categories as shown in Section IV), we replicate at least two bugs for each category. For the category with more than 10 cases, we replicate relatively more bugs in our affordable efforts. Table II show the details of the 23 replicated bugs.

*2) Fault Diagnosis Method:* Our work aims to study how observability data is used in fault analysis of robotic systems. We particularly focus on log, trace, and trajectory, which are useful and commonly used in fault diagnosis of robotic systems. In ROS-based systems, logs and trajectory data are automatically generated during execution and can be simply collected. However, we leverage a distributed tracing framework Zipkin [20] to collect trace data for the ROS system. Zipkin is open-sourced and widely used in distributed systems such as microservice systems.

To observe how the three kinds of observability data are used in fault analysis, we design a staged diagnosis method. According to debugging experiences in practice, developers first try simple methods and exploit local information, such as log, for most bugs are simple and affect system immediately; traces are further employed with logs to tackle faults involving several components; trajectories are used to observe internal behaviors of robotic systems. Finally, our diagnosis approach is divided into three incremental stages:

- *Log-based Analysis.* Given a system failure, in this stage only the logs are available for a participant in the fault analysis, i.e., the participant only can check the logs to reason about the failure. If the participant fails to localize its root cause in this stage in the given time, then she/he moves to the second stage.
- *Trace-based Analysis.* In this stage, the participant is provided with not only the logs but also the traces generated during this failure for fault analysis. We call the analysis in this stage *trace-based analysis*. If failing to localize its root cause in the given time, then the participant moves to the third stage.
- *Trajectory-based Analysis.* In this stage, the participant is provided with additional observability data, i.e., trajectory data collected during the failure, i.e., the participant can analyze the log and trace data as well as trajectory data to diagnose the root cause. We call the analysis in this stage *trajectory-based analysis*. If the participant still fails to localize its root cause in this stage, we consider the participant fails to diagnose this failure.

*3) Fault Diagnosis Process:* We recruit several participants to use our 3-stage diagnosis method. When a failure occurs in a robotic system, typically, we first check logs of the failure to infer its root cause since the logs are easily accessed in the system. If not successful, we seek observability data which is

TABLE II: Replicated Bugs for Fault Diagnosis Study

| | Description | Type |
|---|---|---|
| B1 | Robot wobbles during movement due to inaccurate physical centroid description | Improper Physical Modeling |
| B2 | Robot keeps moving without responding to new tasks due to incorrect topic message | ROS Node Communication Bug |
| B3 | Robot remains stationary after accepting tasks due to incorrect API parameters for direction recognition | Application Logic Bug |
| B4 | Idle robot unresponsive to tasks because of incorrect namespace mapping between multiple robots | Improper ROS Configuration |
| B5 | Robot system crashes as driver nodelet thread is uninitialized | Application Logic Bug |
| B6 | Robot fails to move due to missing type conversion when parsing messages | Application Logic Bug |
| B7 | Robot moves in the opposite direction because of an incorrect numerical calculation formula | Application Logic Bug |
| B8 | Robot remains stationary due to improper usage of serial port API | Hardware Misuse |
| B9 | Robot fails to move under manual control due to incorrect remapping configuration | Improper ROS Configuration |
| B10 | Robot spins in circles without moving forward due to incorrect global coordinate system in topic message | ROS Node Communication Bug |
| B11 | Robot stumbles while moving because of incorrect subscription relationships | ROS Node Communication Bug |
| B12 | Robot system fails to start due to incompatible dependencies | Improper Operating System Configuration |
| B13 | Radar malfunctions because of missing runtime dependencies | Improper ROS Configuration |
| B14 | User-specified recovery behavior fails to execute due to incorrect array boundary access | Application Logic Bug |
| B15 | Robot response is slow as pass-by-reference communication is used in high real-time requirement nodes | Application Logic Bug |
| B16 | Robot arm remains stationary as hardware status is not checked after power-up | Hardware Misuse |
| B17 | Robot stops suddenly as minimum movement speed is insufficient to overcome physical friction | Unforeseen Physical Conditions |
| B18 | The robot gets stuck while passing through a half-open door due to the oversized inflation of the Costmap. | Unforeseen Physical Conditions |
| B19 | Robot fails to move because of incorrect pose in robot model | Improper Physical Modeling |
| B20 | Robotic arm stops suddenly during motion as it fails to avoid ”singularity” | Hardware Misuse |
| B21 | Robot system crashes during mission execution due to outdated Qt dependency versions | Improper Operating System Configuration |
| B22 | Robot remains stationary because of damaged radar | Hardware Malfunction |
| B23 | Robot stops suddenly while moving as USB serial cable becomes unplugged from Kobuki base | Hardware Malfunction |

*higher level* such as traces and trajectory data. In general, these kinds of data are more helpful in fault analysis but relatively expensive to obtain. Collecting trace and trajectory data often requires additional libraries or third party tools.

A participant diagnoses a failure with the following data:

- *Failure Description.* For each failure, we write a piece of text briefly describing the failure, e.g., a robot does not move its arm when a move command is sent.
- *Error messages.* When a failure is replicated, we identify the error massages that are generated when the failure occurs.
- *Logs.* For each failure, we collect all the logs generated in the ROS system.
- *Traces.* The trace data is automatically collected by our tracing tool Zipkin.
- *Trajectory.* The trajectory data is generated and collected by the *rivz* tool in the ROS system.

The participant has 3 hours to diagnose each failure. She/he first reads the failure description and error massages and makes sure she/he understands the failure. Then she/he follows the 3-stage diagnosis method above to localize the root cause of the failure. We ask all participants record their fault diagnosis processes. During analysis, the participant herself/himself decides whether she/he fails to localize the root cause in the current stage and whether moves to next stage for accessing more observability data. We do not set a fixed time budget for each stage because a participant with different background may require a different time for the same stage. Thus, we let the participant decide when to move to next stage. For each task, we record in which diagnosis stage the participant successfully localize the root cause and her/his fault analysis experience by conducting an interview.

## IV. Results and Findings

In this section, we present the results of our empirical study and summarize the findings. Based on the results and findings, we answer the three research questions. All the dataset and the results can be found in our replication package on anonymous link: https://sites.google.com/view/robot-bug-study/.

### A. Bug Case Analysis (RQ1)

In the bug case analysis, we investigate the characteristics of robotic system bugs from their root causes and overall impact from the perspective of users.

*1) Labeling:* With provided initial labels, two of the authors independently assign labels to each bug. In addition, we enhance the labels by utilizing an open-coding scheme to expand the list. In particular, when encountering bugs that do not fit within the initial labels, each author performs a manual analysis and chooses their own label for those bugs.

After all the bugs are labeled by each author, they come together to compare their assigned labels. To address any conflicts, a third party is brought in to facilitate a consensus-driven discussion. We calculate the inter-rater agreement using Cohen's Kappa. The final Cohen's Kappa coefficient for root causes and impact are 0.89 and 0.82 respectively.

*2) Root Cause:* The results are shown in Table III. We classify the 398 bug cases into four level, i.e., physical environment interaction, hardware usage, software implementation, and system configuration. Each category encompasses two types of root causes. Among the 398 bug cases, 47 (11.81%) are related to physical environment interaction, 59 (14.82%) are related to hardware usage, 196 (49.25%) are related to software implementation, 96 (24.12%) are related to system configuration.

**Physical Environment Interaction**. Root causes related to physical environment interaction include two types, i.e., Unforeseen Physical Conditions (UPC) and Improper Physical

TABLE III: Bug Case Analysis Results

| Root Cause | Robust | | | ARSB | | |
|---|---|---|---|---|---|---|
| | #Bugs | #Malf | #PerfD | #Bugs | #Malf | #PerfD |
| Physical Environment Interaction | 6 | 6 | 0 | 41 | 38 | 3 |
| Unforeseen Physical Conditions | 2 | 1 | 0 | 11 | 10 | 1 |
| Improper Physical Modeling | 4 | 5 | 0 | 30 | 28 | 2 |
| Hardware Usage | 13 | 13 | 0 | 46 | 43 | 3 |
| Hardware Malfunction | 1 | 1 | 0 | 23 | 22 | 1 |
| Hardware Misuse | 12 | 12 | 0 | 23 | 21 | 2 |
| Software Implementation | 89 | 77 | 12 | 107 | 94 | 13 |
| Application Code Logic Bug | 60 | 50 | 10 | 80 | 69 | 11 |
| ROS Node Communication Bug | 29 | 27 | 2 | 27 | 25 | 2 |
| System Configuration | 27 | 27 | 0 | 69 | 66 | 3 |
| Improper Operating System Configuration | 14 | 14 | 0 | 26 | 25 | 1 |
| Improper ROS Configuration | 13 | 13 | 0 | 43 | 41 | 2 |
| Total | 135 | 123 | 12 | 263 | 241 | 22 |

Modeling (IPM). UPC means that the developers do not foresee some environment conditions that break the assumptions of the development of the robotic system. In the dataset, there are a total of 13 UPC bug cases, with 2 occurring in Robust and 11 in ARSB. For example, a bug case (report ID *0416c81*[2]) in the Kobuki project reports that the robot does not move when it is expected to move to a specific location, for example for charging, with a low speed that is set in a configuration file. The default speed setting works most of the time, but may cause a fault when the robot carries such a heavy load that it cannot overcome the friction. IPM means that the developers provide an incorrect environment model that makes the robot malfunction. There are 34 IPM bug cases in the dataset, with 4 in Robust and 30 in ARSB. For example, a bug case (report ID *21b86f6* [4]) in the universal robot project is related to the URDF (Unified Robot Description Format) model, which is an XML format for describing a robot model. In this case the dimension of the robot arm in the UDRF model does not reflect the real dimension, thus the planned movement trajectory is wrong.

**Hardware Usage**. Root causes related to hardware usage include two types, i.e., hardware malfunction and hardware misuse. Hardware malfunction means that hardware components fail or work incorrectly. There are 24 hardware malfunction bug cases in the dataset, with 1 in Robust and 23 in ARSB. For example, a bug case ( ID *606b8b9*[7]) in the Kobuki project. In this case the USB serial cable is unplugged and then the Kobuki driver's node crashes. This fault is also related to the implementation of the node: it tries to read data from the Bluetooth interface when USB is unplugged, thus a crash may occur under certain conditions. Hardware misuse means that the developers use specific hardware components in a wrong way. There are 35 hardware misuse bug cases in the dataset, with 12 in Robust and 23 in ARSB. For example, a bug case (issue ID *89145c4*[8]) in the universal robot project reports that the multi-axis robot arm collides with itself. The cause for the bug is that the developers put an improper limit on the elbow joint and thus the arm may cross through the collision zone.

**Software Implementation**. Root causes related to software implementation include two types, i.e., application logic bug and ROS node communication bug. Application logic bugs are general software bugs in robotic systems, e.g., concurrency bugs, API misuse, numerical calculation bugs. There are 140 application logic bugs in the dataset, with 60 in Robust and 80 in ARSB. For example, a bug case (issue ID *1c141a5*[3]) in the Kuboki project reports that the robot is supposed to move forward but instead moves backwards. The cause for the bug is that the direction of the linear velocity is wrong after a conversion from floating number to short integer. ROS node communication bugs usually lie in incorrect data transmission or control in node communication. There are 56 ROS node communication bugs in the dataset, with 29 in Robust and 27 in ARSB. For example, a bug case (issue ID *594978d*[6]) in the Mavros project reports that the robot behaves in an unintended way. The cause for the bug lies in missing information in the packet header. As the information indicates the beginning of the packet, the receiving node will skip the package if the information is missing.

**System Configuration**. Root causes related to system configuration include two types, i.e., Improper Operating System Configuration (IOSC) and Improper ROS Configuration (IROC). IOSC means an incorrect configuration in the operating system (e.g., Ubuntu) of the robotic system. There are 40 IOSC bug cases in the dataset, with 14 in Robust and 26 in ARSB. For example, a bug case (issue ID *0000000*[1]) in the Care-O-Bot project reports that the motor for the tray is not moving. The cause for the bug is that the specific ROS packages deployed on the operating system are outdated. IROC means an incorrect configuration in ROS. ROS-based robotic systems typically use ROS configurations to register hardware devices and application nodes, and establish communications between different nodes. Improper ROS configurations may lead to hardware or node failures or faulty communications between nodes. There are 56 IROC bug cases in the dataset, with 13 in Robust and 43 in ARSB. For example, a bug case (issue ID *3e32933*[5]) in the TurtleBot project reports that image processing does not receive data. The cause for the bug lies in a wrong topic remapping in the launch file (an XML file specifying system node information, parameters, and remappings), which makes the sensor node publish messages with a wrong topic.

*3) Overall Impact:* In Table III, we list the number of bug cases, which are roughly labelled malfunction and performance degradation according to their impacts. Significantly, the number of malfunction (364) is much more than the one of performance degradation (34).

**Malfunction**. Malfunction (Malf in Table III) refers to the unexpected behavior of a robotic system, where the system deviates from its intended function or fails to perform its task altogether. From Table III, it is evident that all categories of root causes are highly likely to result in malfunction.

The category, physical environment interaction, has 44 bug cases causing malfunction, consisting of 11 UPC and 33 IPM bug cases. Both UPC and IPM bug cases cause a robotic system to have an incorrect perception of the real world, leading to erroneous behavior. In UPC case ARSB-8, the robotic system encounters electromagnetic interference from other electronic devices, such as the paging devices used by hotel staff, resulting in uncontrolled robot motion. Obviously the developers have no experiences about interfering signals in a physical environment.

The category, hardware usage, has 56 bug cases causing malfunction, consisting of 23 hardware malfunction and 33 hardware misuse bug cases. Hardware malfunction bugs can prevent a robotic system from executing tasks, whereas hardware misuse can cause malfunction by an incorrect understanding of the hardware and lead the robotic system to perform tasks that violate hardware constraints. For example, in hardware misuse bug case ARSB-11, the robotic system was using the serial line to communicate with the Kobuki base, but the *serial.write()* method was not concurrently protected, resulting in the Kobuki node receiving dirty data and moving incorrectly.

The category, software implementation, has 171 bug cases causing malfunction, consisting of 119 application logic and 52 ROS node communication bug cases. Application logic bug is a classic cause of malfunction in typical software systems, and ROS node communication bugs that cause interrupted interaction between nodes in robotic systems can further lead to malfunction. For instance, in ROS node communication bug case ARSB-13, after replacing the robot base with a new version, outdated communication parameters were used to communicate with its nodes, resulting in the robot being unable to move. We have further discovered that many software implementation bugs that might cause performance degradation in traditional software systems can directly result in malfunction in robots. For instance, we found that 93.47% of memory leaks directly caused malfunction in robots. We attribute this to the limited computing capabilities of robot systems, which typically use embedded chips and communicate via IoT protocols or serial ports, leading to a constraint in available computing resources and data transmission capabilities.

The category, system configuration, has 93 bug cases causing malfunction, consisting of 39 IOSC and 54 IROC bug cases. IOSC can interrupt the functionality of a robot due to unexpected system environment factors, such as missing runtime dependencies or unsynchronized distributed clocks. IROC can cause errors in default parameters for robot nodes and disrupt the subscription relationships between nodes. For example, incorrect resource path configurations can cause the robot's expression playback node to fail to play the correct interactive expressions, while incorrect node subscription relationships can cause some nodes to become unresponsive.

**Performance Degradation**. From Table III, it is evident that all categories of root cause can lead to performance degradation (PerfD in Table III). The Software Implementation category has a major count (59.09%), with 11 cases falling under the Application Code Logic Bug type and two cases under ROS Node Communication Bug type. Further analysis revealed that concurrency issues, handle leaks, and planning algorithms are the most common types of bugs that cause performance degradation in application code logic bug type. For example, in the ASRB-187 bug case, the robot's planning algorithm suffered from particle dispersion, which refers to the phenomenon where simulated particles, such as those used in path planning algorithms, begin to move away from each other due to various factors, including noise, errors in sensing or modeling, or imperfect algorithms. This can lead to suboptimal path planning, which can negatively impact a robot's performance by making it take longer to reach its intended destination or even cause the robot to fail to complete its task.

There are three bug cases in the hardware usage category that caused performance degradation, including one hardware malfunction case and two hardware misuse cases. For example, in hardware malfunction case ARSB-50, the robot needs to communicate with the elevator and open its doors. However, the communication equipment in the elevator is worn out, leading to unresponsive feedback. As a result, the robot needs to request entry into the elevator more times than usual.

There are three bug cases in the physical environment interaction category that caused performance degradation, including one UPC case and two IPM cases. For instance, in IPM case ARSB-86, noise in the environment during the modeling process resulted in the robot spending more time localizing its position in the real world, leading to performance degradation.

There are three bug cases in the system configuration category that caused performance degradation, including one IOSC case and two IROC cases. For example, in IOSC case ARSB-73, the robot needs to interact with the visual module and create real-time obstacle avoidance plans. However, due to an incorrect configuration of the NetworkManager in the operating system, the network communication is unstable, causing the robot's behavior to become sluggish.

*4) Summary:* We manually analyze bugs from dataset ROBUST and ARSB. Results show that characteristics of bugs agree with that robotic system is a type of software-hardware integrated system. Hence fault diagnosis for robotic systems is of challenge.

- Robotic system bugs have very diverse root causes in

different levels of the system, including environment interaction, hardware usage, software implementation, and system configuration.
- Robotic system bugs may cause both malfunction and performance degradation of the robots. Furthermore, due to the limited computing resources of a robot, most bugs will directly impact the robot's functionality.
- Bugs rooted in different levels may have similar symptoms from the perspective of users. Therefore, it is necessary to combine observability data from different levels for fault diagnosis in robotic systems.

## *B. Fault Diagnosis Study (RQ2)*

In the fault diagnosis study, we examine whether and how a participant successfully accomplishes a fault diagnosis task. In particular, we are concerned about how observability data (i.e., logs, traces, trajectories) is used in fault diagnosis.

*1) Participants:* We have recruited 5 graduate students in our department. Four (P2-P5) have worked on robotic systems and the other one (P1) on CPSs for at least two years. They all have over 3 years' programming experience. All are very good at debugging programs with logs. Four (P2-P5) can utilize trajectories to understand behaviours of a robotic system. They all have learned distributed tracing in a course, but without much practical experience.

There is apparent skills gap between our participants and software engineers from industry. To make our evaluation as sound as possible, we build two more extra bugs in the food delivering system so that participants can exercise their skills to use diagnosis methods without time budget.

*2) Overview:* As designed, we have 5 participants and 23 replicated bugs, leading to 115 fault analysis tasks (5*23). In the study we evaluate the 5 participants on all the 23 reproduced bugs, i.e., each participant is asked to diagnose all the 23 reproduced failures. In total we need to conduct 5*23=115 experiments. To reduce the total time, we setup five simulation environments and 1 real world robot for experiments such that they can be performed parallel. Overall, it takes a half month to complete those experiments.

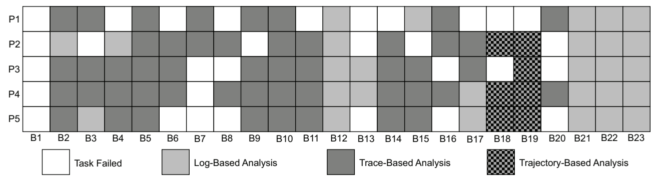

Fig. 1: Fault Diagnosis Study Results

Figure 1 shows the results of the fault diagnosis study, where X-axis denotes the bug cases (B1-B23) and Y-axis denotes the participants (P1-P5). It shows whether and how the participants accomplishes their fault diagnosis tasks: white box means that the participant fails to accomplish the task in the given time; light grey means that the participant accomplishes the task with logs; dark grey means that the participant accomplishes the task with logs and traces; dotted grey means that the participant accomplishes the task with logs, traces, and trajectories.

There are 84 successful tasks out of total 115. Among the 84 successful ones, 28 (33.33%) use log-based analysis, 49 (58.33%) further use trace-based analysis, and 7 (8.33%) further use trajectory-based analysis. Without no surprise, P1 perform the worst in this evaluation. As a representative of novice developers for robotic systems, P1 accomplishes 13 tasks, more than half of all. It may reveal that diagnosis methods based on observability data can help developers with little experience to debug robotic systems effectively.

To understand how different kinds of observability data complement each other, we further investigate the fault diagnosis processes of different tasks.

*3) Log-based Analysis:* As shown in the results, for a significant amount of tasks (28 tasks), the participants identify the root causes by only analyzing the logs. For the bug cases B12 and B13, all the successful fault diagnosis tasks are accomplished by only analyzing the logs. This is kind of reasonable since logs contain rich runtime information about a fault that is helpful to fault analysis. Meanwhile, a robotic system is distributed and involves multiple nodes in the execution and it is often insufficient to localize the root cause of a fault only with logs. To understand why the log-based analysis is so successful for B12 and B13, we further investigate the symptom details of the bugs and their fault diagnosis processes.

**Bug Case B12**. For this case, all the participants successfully localize its root cause using log-based analysis. This bug is caused by the incorrect version of a library involved in the execution. Specifically, the ROS system requires a pyYAML library with above 5.0 version to load YAML file, but is provided with a version 3.1.2 pyYAML library. Based on the error message in logs "AttributeError: 'module' object has no attribute 'FullLoader"', one easily identifies it as a wrong version problem.

**Bug Case B13**. This is a configuration error as well. The ROS system uses a model to describe hardware components and objects in the physical world and needs to load the description file of a hardware component or object. In this case, the path of the description file of the radar is misconfigured, causing a system crash. The error message in the logs shows that the model construction fails when the system is launched. This information often leads to a further investigation of the logs for checking the status of involved components. By analyzing the logs, the problematic radar modular can be identified.

*4) Trace-based Analysis:* In trace-based analysis, the participants are provided with one more kind of observability data, i.e., trace data generated by distributed tracing during system execution. As shown in the results above, when trace data is available, the participants successfully accomplish 49 more tasks (out of 87 tasks for which they fail using log-based analysis). That is, trace data significantly improves the success rate of root cause localization in robotic system, around 175% (49/28). To understand the reasons behind the impressive results, we further investigate the analysis processes of the 49 tasks. Here are two typical examples showing how trace data contributes to fault diagnosis in robotic systems.

**Bug Case B11**. In the robotic system, multiple nodes are involved when a robot moving. One of nodes called *velocity_smoother* is used to control the speed of the robot and make it move smoothly. If this node is not involved in the execution, the robot moves in a clumsy manner. In this case, the speed setting is not sent to the *velocity_smoother* node for smoothing before it is sent to the chassis, causing the robot moving in a clumsy way. It is sort of challenging to localize its root cause by analyzing logs. As described by a participant, he checks the logs of each node and it seems that all nodes work well, so he fails to localize the root cause. When entering the trace-based analysis, he quickly identifies the root cause by checking the traces. Within his experience, the node *velocity_smoother* typically follows *move_base* (a movement planning module in ROS) in the execution. However, the traces show that *velocity_smoother* is not involved in the execution. With this observation, he further analyzes the logs generated in the related nodes and successfully localize the root cause of the fault.

**Bug Case B10**. This bug is related to the robot movement modular as well. It is caused by that one of variables in the module is not initialized. The uninitialized variable results in that the navigation module is unable to use the global coordinate system in the planning. Without the global coordinate system, the robot moves in a circle instead of moving to the target. It is difficult to diagnose this issue by only checking the logs since the information about the uninitialized variable does not appear in the recent logs. When trace data is available, the participants can analyze the linked logs and track back the execution to identify its root cause.

*5) Trajectory-based analysis:* As robots often interact with the physical world, analyzing trajectory data helps fault localization in the robotic system. As shown in the results, when trajectory data is available, the participants successfully localize the root causes for seven more tasks (out of the remaining 38 tasks). This indicates that trajectory data can further improve the success rate of fault diagnosis in the robotic system. To understand how trajectory data contributes to fault diagnosis, we investigate the analysis processes of the seven tasks. Here are two bug cases that are successfully diagnosed using trajectory-based analysis.

**Bug Case B18**. The problem arises when a robot is going through an open door. The robot is supposed to go through the door and reach the target location. However, it stops at the front of the door. This is because the robot considers this door is too narrow to go through. Actually, the door is wide enough for the robot to go through. The reason for this is that the parameter of the expansion layer is misconfigured. The expansion layer is a safety distance between a robot and obstacles, which ensures that the robot can go around the obstacles without crashing. In this case, the expansion layer is configured with a too large value such that the robot fails to go through the door. Thus, it is difficult to diagnose this issue by analyzing logs and traces. When trajectory data is available, three of the five participants successfully localize the root cause.

**Bug Case B19**. This bug is caused by improper modeling of the physical world. In this case, a robot starts from its original position in the model and is supposed to move to a target location. However, it remains still and does not move forward. The reason is that the robot stands towards a wall with its back towards open ground, but the model tells that its back is towards a wall. Thus, the robot cannot turn back and move to the target as there is a “wall” behind it in the model. As described, this bug is closely related to the physical environment and it is difficult to diagnose without trajectory data.

*6) Special Cases:* For some bug cases the participants localize the root causes in different ways, for example using log-based analysis or trace-based analysis. They are B2, B3, B4, B15, and B17. Our analysis shows that the participants using simpler analysis (e.g., log-based analysis only) usually have priori knowledge on the bug. For example, for B17 two participants (P4, P5) succeed with only log-based analysis, while another two (P2, P3) succeed by further using trace-based analysis. This bug case reports that the robot suddenly stops when it moves to a location for charging. It is caused by that the speed setting in the configuration is too small to make the robot overcome the friction when carrying a heavy load. Based on their feedback, we know that P4 and P5 have some knowledge about the speed setting and thus directly choose to check the logs of the component for moving control for charging; while P2 and P3 do not have the knowledge and thus choose to go through the traces to identify where the speed of the robot for charging is set.

There is one bug case (B1) for which all the participants fail to identify the root cause. This bug case reports that the robot wobbles when moving. The root cause lies in an incorrect setting of the robot’s mass centre in the URDF (Unified Robot Description Format) file. This setting is important for the balance control of the robot. This bug is common when the moving control software is migrated to a robot of different model that has different mass centre, for example due to the different position of the battery. This bug does not cause explicit anomalies in the logs, traces, or trajectories. Moreover, different from B17 where the relationship between speed and friction is well known, the relationship between mass centre and posture control is not known for the participants. Two participants guess that the problem is related to the URDF file, but they lack the required knowledge to identify the root cause, i.e., the setting of mass centre.

*7) Summary:* The experiments show that collected observability data help participants succeed in diagnosing fault of robotic systems. They play different roles in fault diagnosis. Success ratio of diagnosis could be improved with proper combination of them.

- Logs are fundamental for fault diagnosis in robotic systems, which can provide detailed information. Log-based fault diagnosis succeeds when the root causes are not far from where the faults are detected, in which local information is enough for diagnosis.

- Traces are good at linking together logs generated in different nodes. Hence they play a key role in finding nodes related to a fault and information exchanged among them, which help developer narrow down search for fault diagnosis.
- Trajectory data help us understand how robots sense the world and behave continuously. Therefore trajectory plays a key role when faults involve physical environments.
- It is rare that some complex bugs can be diagnosed successfully with only logs if developers just have specific background knowledge about those bugs.
- Some bugs cannot be diagnosed even by combining all of logs, traces, and trajectories. They involve complex background knowledge and observability data available now show no explicit anomalies.

## V. Threats to Validity

A major threat to the internal validity lies in the subjective judgment in the bug case analysis. The classifications of root causes and impacts are conducted based on the understanding of the annotators. For minimizing the threat, we try to collect sufficient information for the annotators, including issue descriptions, follow-up discussions, fix commits and source code. Moreover, we follow commonly used data analysis techniques by involving multiple annotators and conflict resolution and reporting agreement coefficients. Another major threat to the internal validity lies in the differences of the participants in their experience and capability in the fault diagnosis study. To alleviate the threat, we conduct qualitative analysis on their fault diagnosis processes to understand the usefulness of different kinds of observability data and make our findings back on the analysis of concrete cases.

Threats to the external validity mainly lie in the representativeness and coverage of different kinds of bugs in real robotic systems. Our empirical study is based on a set of bug cases collected from the issues of robotic software projects, which may not cover robotic system faults that are caused by hardware or physical environment. To alleviate the threat, we add some bug cases collected from industrial robotic systems in the fault diagnosis study. The robotic application used in the fault diagnosis study is small and covers a limited number of different robots and their functionalities. Therefore, our findings may not be generalized to more complex robotic systems in industry. Another threat comes from what observability data we collect. Some data is critical to analyze robotic fault, which can determine whether our diagnosis is successful or not. In practice, the infrastructure can only collect limited capacity of predetermined data. How to determine the set of collected data is another research problem with limited budget.

## VI. Related Work

A typical robotic system consists of many heterogeneous hardware and software components with complex interactions between them. A survey [70] shows that most robotics systems frequently contain failures. Some work try to understand bugs in specific robotic area and general CPSs, including dependency bugs in ROS [29], configuration bugs in swarm drones [41], characterization of software bugs in open-source CPS [77] and of safety concerns in UAV software platforms [28], and a comprehensive study of autonomous vehicle bugs [30]. Malavolta et al. [50], [51] empirically study architectures of ROS-based systems and summarized 47 architecture guidelines for developers to write correct ROS-based programs. Specifically, based on the analysis of root causes of bugs in robotic navigation system, Bug Algorithms [52] are proposed to handle uncertainties of the environment and hardware components. This paper studies bug characterization from the perspective of observability, as a helpful supplement to previous work.

With understanding of bug characterization, RoboFuzz [46] is proposed to employs fuzzing techniques to find bugs in ROS-based robotic systems. SWARMBUG [41] is also proposed to automatically detect and fix configuration bugs in swarm robotics. We investigate how different kinds of observability data could help diagnose runtime faults of robotic systems. ROS provides a generic logging functionality [60] and a tool *rosbag* [63] to sample messages. However, messages cannot practically convey the stacktrace-level of detail and the detailed execution context information [24]. A multipurpose framework *ros2_tracing* [24] is proposed for ROS 2 to fill this gap. *ros2_tracing* instruments the source code of ROS 2 and allows extracting simple metrics, such as publishing rate and callback duration. It is applied to analyze, debug and optimize perception and mapping subsystems in ROS 2 [47]. *Autowre_Perf* is built upon *ros2_tracing* with an improvement that arbitrary nodes can be selected for instrumentation. RAPLET [56] is another real time instrumentation tool for binary compatibility based on the dynamic linker's LD_PRELOAD environment variable. To debug large multi-robot systems, De Rosa et al. [26] develop a technique *distributed watchpoint triggers* to recognize distributed conditions, which help us understand behaviors of robotic systems. To collect complicated runtime information, we rely on a more general monitoring system. Stadler et. al. [68] summarized monitoring challenges and proposed an architecture towards flexible runtime monitoring system support ROS application. Specially, the changes of a message [75] can also be tracked with a monitoring system.

Furthermore, multimodal data [58], sensed by robots or external sensors, are employed to extract more information. Typical debugging techniques, such as `printf` or log files, are not good at handling complex data types, such as radar signals and video streams. A visualization tool *rviz* [64] can interpret and display complex data in a way that developers understand easily. A more general visualization system named *Rviz* [42] accepts arbitrary data structures and algorithms. Basurto et al. [23] also propose a visual tool for monitoring and detecting anomalies in robot performance. Ikeda and Szafir [40] explore various design approaches towards such visualizations for robotics debugging support, especially including emerging immersive three-dimensional augmented reality.

Data are collected and managed for further analysis. For distributed logs, He [38] proposes an end-to-end management framework. Bédard et al. [25] design a tool based on *ros2_tracing* to keep complex causal links in message flow. Similarly, we instrument the source code generated for specific topic messages, manually set tracepoints for ROS and applications, and collect traces by Zipkin [20]. Then, we analyze and understand behaviors of a robotic system with managed data. A tool *Horus* [54] can analyze root cause from distributed logs. Especially, it is possible to diagnose a fault automatically, for a robotic system can be characterized by models specified by human or automatically discovered [22]. Then an expectation for the robot's behavior can be quickly predicated with an input (e.g., instruction) and its model. ROS has integrated a simple diagnostic system [62]. However, it only works for simple hardware devices. Based on diagnostic stack of ROS, Zaman et. al. [76] propose an online diagnosis and repair system for robotic systems, in which models and action rules are described using first order logic sentences.

Khalastchi and Kalech [44] make a survey on fault detection and diagnosis from characteristics of robotic systems and analyz the advantages and disadvantages of existing approaches, which are divide into three typical categories: data-driven, model-based, and knowledge-based. Model-based approaches [37], [69], [76] exploit an explicit a prior model. Data-driven approaches [32], [39] are model free and are able to handle unknown faults. They apply various analysis methods , in machine learning and statistics, on collected data of a robotic system to infer results. Obviously, they all heavily depends on online monitoring systems [58], [74], [75]. Knowledge-based approaches [36], [59] mimic a human expert, which associates recognized behaviors with predefined known faults and diagnosis. Mitrevski et al. [53] address how to utilize knowledge about the execution process to direct the diagnosis and experience acquisition process. Furthermore, it is possible for knowledge-based approach to combine model-based and data-driven approaches into a hybrid approach [45], [59].

It is costly and inefficient to develop robotic systems with real robots. Simulator plays a important role in the development process. Timperley et al. [71] disclose that the majority of bugs can be reproduced using software-in-the-loop simulation approaches without the need for complex triggering mechanisms. However, the environment is too complex. A robotic system runs in the nondeterministic physical world, e.g., movements are guided by motion planning [34] based on sensing uncertainties [33]. Stein and Elbaum [72] propose an approach to automatically reduce the key elements of the environment associated with identified failures, which are critical for faster fault isolation and, ultimately, debugging those failures. A simulator is also used to control or simulate uncertainties in the environment for the developing robotic system. There are several mainstreaming open source simulators, such as Gazebo [31], Webots [73], and Robogym [57]. For specific purpose, developers may develop a particular simulator, e.g., iGibson 2 [49] is an open-source simulation environment for household tasks.

## VII. Conclusion and Discussion

In this paper, we conduct an empirical study on fault diagnosis in robotic systems from the perspective of observability. The study includes a bug case analysis to understand the characteristics of robotic system bugs and a fault diagnosis study to analyze the fault diagnosis processes of a set of replicated bugs. The results indicate the effectiveness of traces and trajectory data and the necessity of combining different kinds of observability data for fault diagnosis in robotic systems.

Our study in this paper promotes the vision of observability-based fault diagnosis in robotic systems. In modern cloud-based systems such as microservice systems, it has been common to deploy observability infrastructures such as distributed tracing systems and combine different kinds of observability data such as logs, traces, and metrics for anomaly detection and fault localization [48], [78], [79]. For robotic systems, it is necessary to further combine physical world related monitoring data, e.g., status of hardware components and physical interactions with the environments, to construct a unified observability platform. The platform can be used to support the anomaly detection, fault diagnosis, and even runtime repairing of robots. The main challenge lies in the effective and efficient fusion of different kinds of observability data, especially observability data reflecting the status and behaviors of software components, hardware components, and physical environments. In future work, we will investigate software infrastructures and tools for the collection, analysis, and visualization of observability data of robotic systems and further explore the combination of the observability infrastructures and tools with robotic digital twin platforms for more effective operation management of robotic systems.

## Acknowledgment

We wish to thank deeply Tianyi Chen, Zhengming Zhang, Jingyuan Hao, and Hui Li for developing the food delivery system. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported by Shanghai Municipal Science and Technology Major Project (No.2021SHZDZX0103).

## References

[1] "0000000," https://github.com/robust-rosin/robust/blob/master/care-o-bot/0000000/0000000.bug, 2022, accessed: 2022-10-22.
[2] "0416c81," https://github.com/robust-rosin/robust/blob/master/kobuki/0416c81/0416c81.bug, 2022, accessed: 2022-10-22.
[3] "1c141a5," https://github.com/robust-rosin/robust/blob/master/kobuki/1c141a5/1c141a5.bug, 2022, accessed: 2022-10-22.
[4] "21b86f6," https://github.com/robust-rosin/robust/blob/master/universal_robot/21b86f6/21b86f6.bug, 2022, accessed: 2022-10-22.
[5] "3e32933," https://github.com/robust-rosin/robust/blob/master/turtlebot/3e32933/3e32933.bug, 2022, accessed: 2022-10-22.
[6] "594978d," https://github.com/robust-rosin/robust/blob/master/mavros/594978d/594978d.bug, 2022, accessed: 2022-10-22.
[7] "606b8b9," https://github.com/robust-rosin/robust/blob/master/kobuki/606b8b9/606b8b9.bug, 2022, accessed: 2022-10-22.

[8] “89145c4,” https://github.com/robust-rosin/robust/blob/master/universal_robot/89145c4/89145c4.bug, 2022, accessed: 2022-10-22.
[9] “Apache skywalking,” https://skywalking.apache.org/, 2022, accessed: 2022-10-22.
[10] “Care-o-bot,” https://github.com/ipa320/cob_robots, 2022, accessed: 2022-10-22.
[11] “geometry2,” https://github.com/ros/geometry2, 2022, accessed: 2022-10-22.
[12] “Jaegertracing,” https://www.jaegertracing.io/, 2022, accessed: 2022-10-22.
[13] “Kobuki,” https://github.com/yujinrobot/kobuki, 2022, accessed: 2022-10-22.
[14] “Mavros,” https://github.com/mavlink/mavros, 2022, accessed: 2022-10-22.
[15] “Motoman,” https://github.com/ros-industrial/motoman, 2022, accessed: 2022-10-22.
[16] “Opentracing specification,” https://opentracing.io/specification/, 2022, accessed: 2022-11-10.
[17] “Robust,” https://github.com/robust-rosin/robust, 2022, accessed: 2022-10-30.
[18] “Turtlebot,” https://github.com/turtlebot/turtlebot, 2022, accessed: 2022-10-22.
[19] “Universal robot,” https://github.com/ros-industrial/universal_robot, 2022, accessed: 2022-10-22.
[20] “Zipkin,” https://zipkin.io/, 2022, accessed: 2022-08-22.
[21] A. Ahmad and M. A. Babar, “Software architectures for robotic systems: A systematic mapping study,” *J. Syst. Softw.*, vol. 122, pp. 16–39, 2016.
[22] J. Aldrich, D. Garlan, C. Kästner, C. L. Goues, A. Mohseni-Kabir, I. Ruchkin, S. Samuel, B. R. Schmerl, C. S. Timperley, M. Veloso, I. Voysey, J. Biswas, A. Guha, J. Holtz, J. Cámara, and P. Jamshidi, “Model-based adaptation for robotics software,” *IEEE Software*, vol. 36, no. 2, pp. 83–90, 2019.
[23] N. Basurto, C. Cambra, and Á. Herrero, “A visual tool for monitoring and detecting anomalies in robot performance,” *Pattern Anal. Appl.*, vol. 25, no. 2, pp. 271–283, 2022.
[24] C. Bédard, I. Lütkebohle, and M. R. Dagenais, “ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ROS 2,” *IEEE Robotics Autom. Lett.*, vol. 7, no. 3, pp. 6511–6518, 2022.
[25] ——, “ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ROS 2,” *CoRR*, vol. abs/2201.00393, 2022.
[26] M. De Rosa, J. Campbell, P. Pillai, S. C. Goldstein, P. Lee, and T. C. Mowry, “Distributed watchpoints: Debugging large multi-robot systems,” in *2007 IEEE International Conference on Robotics and Automation, ICRA 2007, 10-14 April 2007, Roma, Italy*. IEEE, 2007, pp. 3723–3729.
[27] B. S. Dhillon, *Robot Reliability and Safety*. Springer New York, NY, 1991.
[28] A. Di Sorbo, F. Zampetti, A. Visaggio, M. Di Penta, and S. Panichella, “Automated identification and qualitative characterization of safety concerns reported in uav software platforms,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, apr 2023.
[29] A. Fischer-Nielsen, Z. Fu, T. Su, and A. Wasowski, “The forgotten case of the dependency bugs: on the example of the robot operating system,” in *ICSE-SEIP 2020: 42nd International Conference on Software Engineering, Software Engineering in Practice, Seoul, South Korea, 27 June - 19 July, 2020*. ACM, 2020, pp. 21–30.
[30] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, “A comprehensive study of autonomous vehicle bugs,” in *ICSE ’20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 385–396.
[31] Gazebo, “Gazebo,” https://www.gazebosim.org/, 2022.
[32] R. Golombek, S. Wrede, M. Hanheide, and M. Heckmann, “Online data-driven fault detection for robotic systems,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*. IEEE, 2011, pp. 3011–3016.
[33] A. González-Sieira, D. Cores, M. Mucientes, and A. Bugarín, “Autonomous navigation for uavs managing motion and sensing uncertainty,” *Robotics Auton. Syst.*, vol. 126, p. 103455, 2020.
[34] A. González-Sieira, M. Mucientes, and A. Bugarín, “Graduated fidelity lattices for motion planning under uncertainty,” in *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 5908–5914.
[35] M. Gopal, *Modern Control System Theory*, second edition ed. Halsted Press, New York, 1993.
[36] K. Hamilton, D. M. Lane, N. K. Taylor, and K. E. Brown, “Fault diagnosis on autonomous robotic vehicles with RECOVERY: an integrated heterogeneous-knowledge approach,” in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, ICRA 2001, May 21-26, 2001, Seoul, Korea*. IEEE, 2001, pp. 3232–3237.
[37] M. Hashimoto, H. Kawashima, and F. Oba, “A multi-model based fault detection and diagnosis of internal sensors for mobile robot,” in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, Nevada, USA, October 27 - November 1, 2003*. IEEE, 2003, pp. 3787–3792.
[38] P. He, “An end-to-end log management framework for distributed systems,” in *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*. IEEE Computer Society, 2017, pp. 266–267.
[39] V. J. Hodge and J. Austin, “A survey of outlier detection methodologies,” *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, 2004.
[40] B. Ikeda and D. Szafir, “Advancing the design of visual debugging tools for roboticists,” in *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2022, Sapporo, Hokkaido, Japan, March 7 - 10, 2022*, D. Sakamoto, A. Weiss, L. M. Hiatt, and M. Shiomi, Eds. IEEE / ACM, 2022, pp. 195–204.
[41] C. Jung, A. Ahad, J. Jung, S. G. Elbaum, and Y. Kwon, “Swarmbug: debugging configuration bugs in swarm robotics,” in *ESEC/FSE ’21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 2021, pp. 868–880.
[42] H. R. Kam, S. Lee, T. Park, and C. Kim, “Rviz: a toolkit for real domain data visualization,” *Telecommun. Syst.*, vol. 60, no. 2, pp. 337–345, 2015. [Online]. Available: https://doi.org/10.1007/s11235-015-0034-5
[43] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[44] E. Khalastchi and M. Kalech, “On fault detection and diagnosis in robotic systems,” *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018.
[45] ——, “A sensor-based approach for fault detection and diagnosis for robotic systems,” *Auton. Robots*, vol. 42, no. 6, pp. 1231–1248, 2018.
[46] S. Kim and T. Kim, “Robofuzz: Fuzzing robotic systems over robot operating system (ros) for finding correctness bugs,” in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 447–458.
[47] P.-Y. Lajoie, C. Bédard, and G. Beltrame, “Analyze, debug, optimize: Real-time tracing for perception and mapping systems in ros 2,” *arXiv preprint arXiv:2204.11778*, 2022.
[48] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, “Enjoy your observability: an industrial survey of microservice tracing and analysis,” *Empir. Softw. Eng.*, vol. 27, no. 1, p. 25, 2022.
[49] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. E. Vainio, C. Gokmen, G. Dharan, T. Jain, A. Kurenkov, C. K. Liu, H. Gweon, J. Wu, L. Fei-Fei, and S. Savarese, “igibson 2.0: Object-centric simulation for robot learning of everyday household tasks,” in *Conference on Robot Learning, 8-11 November 2021, London, UK*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 2021, pp. 455–465.
[50] I. Malavolta, G. A. Lewis, B. R. Schmerl, P. Lago, and D. Garlan, “How do you architect your robots?: state of the practice and guidelines for ros-based systems,” in *ICSE-SEIP 2020: 42nd International Conference on Software Engineering, Software Engineering in Practice, Seoul, South Korea, 27 June - 19 July, 2020*. ACM, 2020, pp. 31–40.
[51] ——, “Mining guidelines for architecting robotics software,” *J. Syst. Softw.*, vol. 178, p. 110969, 2021.
[52] K. N. McGuire, G. C. H. E. de Croon, and K. Tuyls, “A comparative study of bug algorithms for robot navigation,” *Robotics Auton. Syst.*, vol. 121, 2019.
[53] A. Mitrevski, P. G. Plöger, and G. Lakemeyer, “Robot action diagnosis and experience correction by falsifying parameterised execution models,” in *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi’an, China, May 30 - June 5, 2021*. IEEE, 2021, pp. 11 025–11 031.
[54] F. Neves, N. Machado, R. Vilaça, and J. Pereira, “Horus: Non-intrusive causal analysis of distributed systems logs,” in *51st Annual IEEE/IFIP*

*International Conference on Dependable Systems and Networks, DSN 2021, Taipei, Taiwan, June 21-24, 2021*. IEEE, 2021, pp. 212–223.

[55] S. Niedermaier, F. Koetter, A. Freymann, and S. Wagner, "On observability and monitoring of distributed systems - an industry interview study," in *Service-Oriented Computing - 17th International Conference, ICSOC 2019, Toulouse, France, October 28-31, 2019, Proceedings*, ser. Lecture Notes in Computer Science, S. Yangui, I. B. Rodriguez, K. Drira, and Z. Tari, Eds., vol. 11895. Springer, 2019, pp. 36–52.

[56] K. Nishimura, T. Ishikawa, H. Sasaki, and S. Kato, "RAPLET: demystifying publish/subscribe latency for ROS applications," in *27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2021, Houston, TX, USA, August 18-20, 2021*. IEEE, 2021, pp. 41–50.

[57] OpenAI, "Robogym," https://github.com/openai/robogym, 2020.

[58] D. Park, Z. Erickson, T. Bhattacharjee, and C. C. Kemp, "Multimodal execution monitoring for anomaly detection during robot manipulation," in *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, D. Kragic, A. Bicchi, and A. D. Luca, Eds. IEEE, 2016, pp. 407–414.

[59] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics Auton. Syst.*, vol. 53, no. 2, pp. 73–88, 2005.

[60] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, 2009.

[61] ROS, "ROS - robot operating system," https://www.ros.org/, 2022.

[62] ROS WiKi, "diagnostic," https://wiki.ros.org/diagnostics, 2022.

[63] ——, "rosbag," http://wiki.ros.org/rosbag, 2022.

[64] ——, "rviz," https://wiki.ros.org/rviz, 2022.

[65] J. Shin and J. Lee, "Fault detection and robust fault recovery control for robot manipulators with actuator failures," in *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*. IEEE Robotics and Automation Society, 1999, pp. 861–866.

[66] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*, ser. Springer Handbooks. Springer, 2016.

[67] C. Sridharan, *Distributed systems observability: a guide to building robust systemsy*. O'Reilly Media, Inc, 2018.

[68] M. Stadler, M. Vierhauser, and J. Cleland-Huang, "Towards flexible runtime monitoring support for ros-based applications," in *2022 IEEE/ACM 4th International Workshop on Robotics Software Engineering (RoSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 43–46.

[69] D. Stavrou, D. G. Eliades, C. G. Panayiotou, and M. M. Polycarpou, "Fault detection for service mobile robots using model-based method," *Auton. Robots*, vol. 40, no. 2, pp. 383–394, 2016.

[70] G. Steinbauer, "A survey about faults of robots used in robocup," in *RoboCup 2012: Robot Soccer World Cup XVI [papers from the 16th Annual RoboCup International Symposium, Mexico City, Mexico, June 18-24, 2012]*, ser. Lecture Notes in Computer Science, vol. 7500. Springer, 2012, pp. 344–355.

[71] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. L. Goues, "Crashing simulated planes is cheap: Can simulation detect robotics bugs early?" in *11th IEEE International Conference on Software Testing, Verification and Validation, ICST 2018, Västerås, Sweden, April 9-13, 2018*. IEEE Computer Society, 2018, pp. 331–342.

[72] M. von Stein and S. G. Elbaum, "Automated environment reduction for debugging robotic systems," in *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, 2021, pp. 3985–3991.

[73] Webots, "Webots," https://www.cyberbotics.com/, 2022.

[74] T. Witte and M. Tichy, "Inferred interactive controls through provenance tracking of ROS message data," in *3rd IEEE/ACM International Workshop on Robotics Software Engineering, RoSE@ICSE 2021, Madrid, Spain, June 2, 2021*. IEEE, 2021, pp. 67–74.

[75] ——, "Towards flexible runtime monitoring support for ROS-based applications," in *4rd IEEE/ACM International Workshop on Robotics Software Engineering, RoSE@ICSE 2022, Pittsburgh, USA, May 9, 2022*. IEEE, 2021, pp. 67–74.

[76] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran, "An integrated model-based diagnosis and repair architecture for ros-based robot systems," in *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. IEEE, 2013, pp. 482–489.

[77] F. Zampetti, R. Kapur, M. D. Penta, and S. Panichella, "An empirical characterization of software bugs in open-source cyber-physical systems," *J. Syst. Softw.*, vol. 192, p. 111425, 2022.

[78] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 623–634.

[79] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Software Eng.*, vol. 47, no. 2, pp. 243–260, 2021.