Received: 1 March 2021 | Revised: 22 February 2022 | Accepted: 18 March 2022

DOI: 10.1111/exsy.13004

ORIGINAL ARTICLE

Expert Systems WILEY

# Towards explainability in robotics: A performance analysis of a cloud accountability system

Francisco Javier Rodríguez-Lera[1,2] | Miguel Ángel González-Santamarta[1] | Ángel Manuel Guerrero-Higueras[1] | Francisco Martín-Rico[2] | Vicente Matellán-Olivera[3]

[1]Robotics group, Escuela de Ingenierías Industrial, Informática y Aeroespacial, Universidad de León, León, Spain

[2]Intelligent Robotics Lab, Universidad Rey Juan Carlos, Fuenlabrada, Spain

[3]Supercomputación Castilla y León (SCAYLE), León, Spain

**Correspondence**
Francisco Javier Rodríguez-Lera, Robotics group, Universidad de León, León, Spain.
Email: fjrodl@unileon.es

**Funding information**
This work has been partially funded by Instituto Nacional de Ciberseguridad de España (INCIBE), 4th addendum to the framework agreement INCIBE–Universidad de León, 2019–2021; Spanish Ministry of Science, Innovation, and Universities, RTI2018-100683-B-I00.

## Abstract

Understanding why a robot's behaviour was triggered is a growing concern to get human-acceptable social robots. Every action, expected and unexpected, should be able to be explained and audited. The formal model proposed here deals with different information levels, from low-level data, such as sensors' data logging; to high-level data that provide an explanation of the robot's behaviour. This study examines the impact on the robot system of a custom log engine based on a custom ROS logging node and investigates pros and cons when used together with a NoSQL database locally and in a cloud environment. Results allow to characterize these alternatives and explore the best strategy for offering a fully log-based accountability engine that maximizes the mapping between robot behaviour and robot logs.

**KEYWORDS**
accountability, autonomous agents, explainability, traceability

## 1 | INTRODUCTION

Managing a robot in a well-known environment, such as a laboratory, might be an affordable task. However, the complexity grows exponentially when moving to a public space. Such spaces do not use a predefined input-data set since the interaction with a robot is open and free. Therefore, it is necessary to have an auditing system, which allows for knowing the trigger of a transient or even unexpected behaviour on a device in regular use and provides a mechanism to analyse the device status after an incident.

Accountability in robotics implies that any robot is in charge of logging its activities with verifiable evidence so that any robot action is traceable, and the events that triggered the action are identified (Xiao, 2009). Thus, current cybersecurity standards are pushing for reaching that point. IEC 62443 is an Industrial cybersecurity standard defining the set of specifications and requirements fundamental to obtain compliance. A closer look at this standard presents items such as the one presented in IEC 62443-3-3 that specifies systems capable of generating audit records through the process of spawning logs (DesRuisseaux, 2018). These audit records or logs should generate accountability, which is the most important requisite towards explainability. The aim is to establish an accountability mechanism as a healthcare system does in its standards. For

**Abbreviations:** CG, conceptual graph; PM, Perfect Mapping; ROS, robot operating system.

Francisco J. Rodríguez-Lera and Miguel Angel González-Santamarta contributed equally to this work.

instance, the IEC 60601-1-10 rule establishes general requirements for safety and clinical essential performance proposing a log-based auditing system.

Consequently, to find out who is legally responsible for any incorrect behaviour performed by an autonomous agent, it is necessary to establish monitoring, logging, and secure recording mechanisms in the current logging system not at odds with performance. Nevertheless, at a first sight, any logging system would be considered as an accountability system, there are different open perspectives that add complexity for its use. First, is the normalization perspective, like the one explored by Yoon and Shao. They propose a formal approximation proposing the Accountable Data Logging Protocol (ADLP) (Yoon & Shao, 2019). This mechanism is tested in a real robot evaluating its performance over the robot; however, the results imply spoilage of performance. This opens the performance perspective (Ding et al., 2015), where the overhead incurred by intensive logging is non-negligible added to the concept of redundant logs. Finally, there is a legal perspective facing the problems with logs such as privacy (Miller, 2017) or cybersecurity to log files, which opens the use of BlockChain-based technologies (White et al., 2019) or encrypted decentralized logging infrastructures like the one proposed by Lee et al. (2020), which are not explored in this research but leave the way open for further development and analysis.

This research proposes to use these logs for providing evidence of responsibility of the software components that trigger the facts of robot behaviour. Our aim is to provide a real-time inspection tool like the one proposed by Theodorou et al. (2017) and its tool ABOD3 (Theodorou & Bryson, 2017), but using an approach based on Robot Operating System ROS (Quigley et al., 2009), as well as quantitative measuring the centralized versus decentralized alternatives of logs. Two logging perspectives are considered: a low-profile perspective to manage raw information, useful for developers and deployers; and a high-profile perspective to explain robot behaviour in a human-understandable manner, useful for the general public.

The accountability system should be faced from a middleware-free approach, for this reason, this research has been working in a three layer approach described in previous research (Matellán et al., 2021; Rodríguez-Lera et al., 2019; Rodríguez-Lera, Santamarta, et al., 2020). However, the authors also believe that the accountability system should be faced from any of the most extended robotic middleware. Robot Operating System (ROS 1 (Quigley et al., 2009)) is probably the best framework to study accountability in autonomous robots since it is the de-facto standard in robotics. ROS logging engine is the primary tool for explaining robot behaviour. Besides, there is a considerable number of ad-hoc tools to evaluate robot performance. For instance, there are solutions to get statistics about message interchange or to provide network introspection services; see (Bihlmaier et al., 2016). In (Rodríguez-Lera, Santamarta, et al., 2020), authors show some of the most popular tools according to a three-factor metric: (1) they are available online in ROS repositories or ROS wiki-forum; (2) they are available online in GitHub, and it has been forked, or (3) the tool has been used in a well-known conference, such as the ROSPlan dispatcher or the rqt plugin (Lima & Ventura, 2018).

Thus, the Research Questions that arises would be presented as: What is the impact on the robot system when the pre-defined middleware is customized for deploying a logging system using both on-board and a cloud computing approach.

In a prior work (Rodríguez-Lera, Santamarta, et al., 2020), authors naively proposed a formal view of an audit process in autonomous robots tracing the links between logs and robot behaviours. This study presents an update based on what we call facts, revisiting the three-layer accountability. The main contributions posed were an overview, modelling, and formalization of the three perspectives of accountability for autonomous robots; a GUI tool to graphically illustrate the knowledge supported on Conceptual Graphs, specifically designed for event identification and environmental monitoring; and finally, a proof of concept summarizing the information dumped on each level using a set of debugging tools developed during the research. However, all this visual information is still supported on logs. These tools simplify the process of auditing and visualizing at a high level, but still, the logs take an important role in the system.

The main drawback associated with logs has to do with performance. In order to gather enough data to provide accounting services, it is necessary to set the logger level as verbose as possible. Such configuration is compute-demanding and usually has a negative effect on the robot's performance. We have explored alternatives to that path too (Fernández-Becerra et al., 2021). This work investigates a mechanism based on syscalls and the *sysdig* tool; however, it would be complicated for an external auditor to trace this option. Besides, our findings on logging storing at least hint that it is necessary to explore a mechanism for optimizing the logging process in size and number.

Authors in Shishkov et al. (2017) explore an alternative mechanism based on middle-out modelling supported on the robot architecture. However, there is a gap between the presentation of the information and the linking of both perspectives, raw logs and architecture components. Niemueller et al. (2012) partially faced logging issues with a ROS package that harvests logs in a well-defined manner and dumps them into a non-relational database (MongoDB). This study challenges the problem through the use of a NoSQL database to gather and store raw data; however, we are different from previous studies in two main ways. First, we present a ROS-integrated solution by rewriting the ROS core logging engine and evaluating adapted and full logging options. Secondly, we have evaluated the system performance when facing the issue on-board or into a cloud solution, which in turn we have divided into two options, a custom private option and a commercial option MongoDB Atlas[1] in its free version.

According to the above, this paper contribution is presented as the corollary of our previous research in order to exhaustively analyse an answer to the research question. Here, it is bounded the performance concept to system efficiency, number of logs and space required for storing the logs and finally it is outlined the challenges explored in this research and summarized the results grading each alternative explored.

The rest of this paper is organized as follows: Section 2 describes our proposal for accounting services introduces our accountability formalization and discusses the different experiments proposed in this paper. Section 3 presents and discusses the results obtained. In Section 4, we expose our observations from the experiment results. Finally, Section 5 summarizes the conclusions and the further work.

## 2 | MATERIALS AND METHODS

In this section, we first introduce an architecture to provide accounting services. Afterward, we describe the experimental procedure for evaluating the research question and the associated metrics.

### 2.1 | Accountability architecture

This section poses our proposal for a three-layer accountability architecture. As mentioned above the first layer matches the logging engine, the second layer is in charge of gathering data about events related to the robot's interaction with its environment, and finally, the third layer illustrates the robot's behaviours. To match behaviours with software components we must consider two main aspects: first, we need to gather raw data from software components; and second, we have to establish the relationships between the elements of different layers. Thus, the logging layer is the most important artefact in the proposed accountability system.

The tools used in this and prior research allow for fulfilling both open data and transparency requisites (Carolan, 2016). According to Carolan's and Attard's research (Attard et al., 2015), it is necessary to reach two milestones: first ($\mathscr{M}_1$), anyone should be able to find, access and use the accountability data; and second ($\mathscr{M}_2$), our accountability system needs to provide accuracy, completeness, consistency and accessibility, thus we must provide the right information, in the right way, and at the right time.

#### 2.1.1 | Accountability dimensions

Setting down the accountability concept (Deber, 2014) into an autonomous robot environment means having robot events to be answerable to someone, for meeting defined objectives by all actors involved from manufacturer and client/user perspectives. Establishing an accountability engine in an autonomous robot varies attending the components and actors involved. The aim is to answer the what, by whom, to whom and how defined by Deber (2014). Every robot act has potential expected or unexpected consequences. Therefore, an accountability system should ensure the robot's duty while maintaining a good performance.

Any robot event should be liable to be traced. Given the event concept, defined as a fact that happens, there are some events associated with its physical nature, visually identified or tangible measured presenting a set of physical effects (for instance: a broken gear). Besides, there are purely logical events at a software layer, where a set of components are doing tasks and making decisions (for instance, the laser sensor driver is sending malformed messages). Finally, there are events at robot behaviour that have associated multiple physical and robot events (laser malfunctioning and malformed message triggers a blind robot). This is what we define as mixed accountability. Figure 1 presents our description for a high-level three-dimensional accountable robot. It is necessary to account, for physical and logical events in addition to the robot behaviour

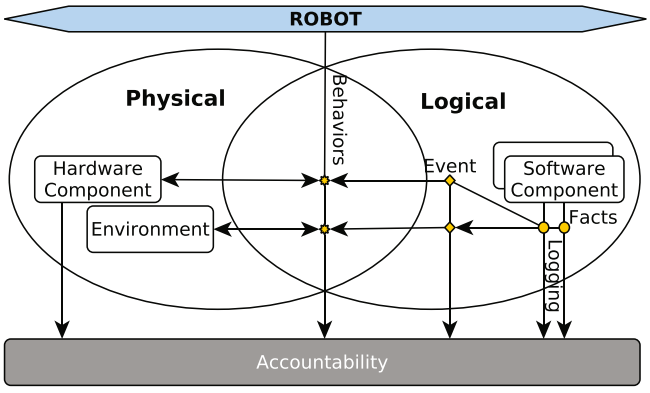


**FIGURE 1** High-level dimensions for an accountable robot

level events located in the mixed area. This study focus on the logical and mixed area. The accountability of pure physical components (as legal regulation of robot components) is out of the scope of this study.

Among the element enumerated, an accounting system should provide a set of mapping functions between robot software components, behaviours and the associated events. We need to think about the sources of information that must be included in the accountability model and which ones are the most reliable for external auditors.

This research started with a three-layer accountability system that models occurrences attending to its nature.

*Logging*: The accountability in this layer is understood as a system that manages information dumped by software components running on the robot.

*Event*: This accountability engine here should manage data triggered from specific situations. From our perspective, an event is just a piece of data that depicts a specific situation happening, such as recognizing an object or triggering an inner status.

*Behaviour*: Accountability here is viewed in terms of data related to the robot's behaviour allowing to know which components run a specific behaviour and the events that triggered it. Consequently, we can identify the software components responsible for the robot's behaviour. At this layer, it is required a certain level of interaction with the physical world (a dialogue, a gesture or a robot action).

In the first layer (logging layer), the mapping between components and events is performed from raw data registered in system logs. This research uses two different modes inspired by Butin et al. (2013): DEBUG and ADAPTED. The DEBUG mode is used when users need to check every log engine running: OS, middleware (e.g., ROS), and robotics applications. Nevertheless, such a mode is meant to be used during specific periods of time. On the other hand, the ADAPTED mode is meant to gather just specific data from some components focusing on critical information about the robot's behaviour.

In the second layer (event layer), we gather information about planning. The mapping is done by matching each event with its associated components. The events are depicted using conceptual graphs (CGs) (Van Harmelen et al., 2008) since they are the most popular method to illustrate such kind of information (Kapitanovsky & Maimon, 1993; Manso et al., 2015; Zender et al., 2008). This layer proposes two main elements: Nodes and Links. Nodes are unique pieces of knowledge. The nodes depict people, robots, objects, and anything in the environment. A node is the smallest unit of knowledge in our proposal and it cannot be split. Links committed for connecting nodes are those for introducing position and those for introducing actions.

The third layer (behaviour layer) is the log information associated with cognition and behaviour data. It maps the events triggering a specific robot's behaviour and the physical actions associated. This means that it is required an external witness, which can be an individual or a third-party component.

This alignment process between software components and robot actions was considered in (Rodríguez-Lera, Guerrero-Higueras, et al., 2020). The three layers are highly supported on logging, which guarantees ($\mathscr{M}_1$). As a consequence, we have been working on them separately.

### 2.1.2 | Formalization

Given the ROS nature, previous accountability research in distributed, networked, or cloud systems gave us the inspiration for providing a formalization of an accountability system for autonomous robots (Xiao et al., 2016). An accountability service ($\mathcal{A}$) for an autonomous robot $\mathscr{R}$may be posed as a three-element tuple: components ($\mathcal{C}$), events ($\mathcal{E}$), and behaviours ($\mathscr{B}$); see Equation (1).

$$\mathcal{A}_R = \langle \mathcal{C}, \mathcal{E}, \mathscr{B} \rangle. \tag{1}$$

A component ($\mathcal{C}$) identifies both a hardware or software item. However, this paper emphasizes software items that are continuously generating facts during its operational mode. Thus, a component is a software item with a specific interface and functionality (Oreback, 1999). An event ($\mathcal{E}$) describes an occurrence happening in a bare instant and a specific scenario as a result of *n* facts. A behaviour ($\mathscr{B}$) describes a robot action triggered by one or multiple events.

An accountable engine handles the assignment of responsibilities. It can be modelled as the mapping functions shown in Equations (2) and (3). $\mathcal{C}_e$denotes a subset of event/s caused by one or many specific facts available in a component $\mathcal{C}$. $\mathcal{E}_b$denotes a subset of events $\mathcal{E}$that cause a specific robot behaviour *b*. Therefore, the $\alpha$ function takes an event *e* as input and returns the facts from the component/s that triggers it. The $\beta$ function takes a robot's behaviour *b* as input and returns the event/s causing it.

$$\alpha : \mathcal{E} \rightarrow \{\mathcal{C}_e | \mathcal{C}_e \subseteq \mathcal{C}\}, \tag{2}$$

$$\beta : \mathscr{B} \rightarrow \{\mathcal{E}_b | \mathcal{E}_b \subseteq \mathcal{E}\}. \tag{3}$$

Under ideal conditions, mapping functions always output the correct results. Such a situation is known as Perfect Mapping (PM) (Xiao et al., 2016). Any $\alpha$ mapping function becomes a PM if and only if $\mathcal{C}_{e|\mathrm{PM}}$includes all facts that a component/s cause the event *e*; see Equation (4). Moreover, any $\beta$ mapping function becomes a PM if and only if $\mathcal{E}_{b|\mathrm{PM}}$includes all the events that triggers behaviour *b*; see Equation (5).

$$\alpha(e) = \mathcal{C}_{e|\mathrm{PM}} \subseteq \mathcal{C}, \tag{4}$$

$$\beta(c) = \mathcal{E}_{b|\mathrm{PM}} \subseteq \mathcal{E}. \tag{5}$$

Nowadays, the diversity of onboard and cloud software components makes difficult the PM. Every software component may provide enough information for a correct mapping as long as none of the results are incorrect. However, external components make the accountability system less reliable, since not all responsible parties can be identified straight away. For instance, an online recognition service would offer minimal log information based on REST API. Typically, we have non-complete sets of components $\mathcal{C}_{e|\mathrm{PM}}$, see Equation (6); or a non-complete sets of events $\mathcal{B}_{c|\mathrm{PM}}$, see Equation (7).

$$\alpha(e) = \mathcal{C}_e \subset \mathcal{C}_{e|\mathrm{PM}} \subseteq \mathcal{C}, \tag{6}$$

$$\beta(c) = \mathcal{E}_b \subset \mathcal{E}_{b|\mathrm{PM}} \subseteq \mathcal{E}. \tag{7}$$

This formalization is illustrated in Figure 2. The presented diagram simplifies the process explained in Matellán et al. (2021). There we can see a set of observable robot behaviour, events and logging facts. The Perfect Mapping is illustrated in 1, 3 and 4 events, where it is possible to track from component facts, the events triggered and the visible results. However, we have visible no traceable behaviours (number 2) which are arisen with no direct translation (for instance a stalled behaviour that is triggered after a period of time). Component $C_2$ shows a fact that is not associated (apparently) with any direct behaviour. Component $C_7$ shows an event that triggers a new event and behaviour, with no clear fact. We can have this situation from a cloud solution given events out of the robot system.

### 2.1.3 | Example

This section illustrates the formalization and the three dimensions explained above through an example. We have deployed an accountability system according to the model depicted in Section 2.1. The system is supported by prior researches (Rodríguez et al., 2011; Rodríguez-Lera, Guerrero-Higueras, et al., 2020). The goal is to present a process to somehow check if the robot is performing adequately and audit that each robot behaviour has associated some information in its logs minimizing its number in space and processing. The scenario presents a robot in a

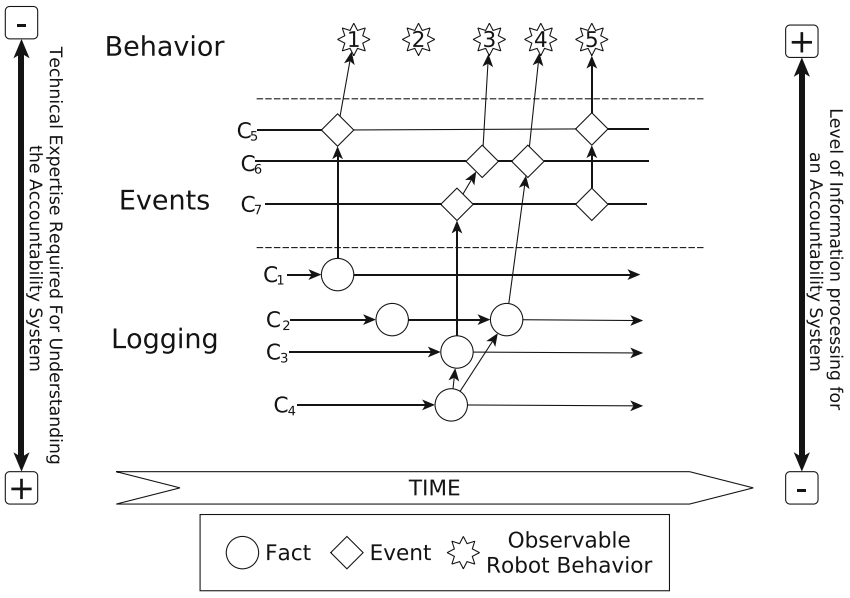


**FIGURE 2** High-level dimensions for an accountable robot

green RoboCup soccer field with white lines, blue and yellow nets and an orange ball, which is the only element that would change its position. The robot has three main behaviours: 1) look for the orange ball, look for the yellow net and look for the blue net. These three "look for" behaviours imply the navigation of the robot running in a Finite State Machine manner, jumping from one to another behaviour every 30 s. Figure 3 presents the Turtlebot robot in the simulated environment. There is a video demonstration illustrating the example proposed above and the experiment is available online.[2]

To justify the accountability dimensions and their relationship with the logging system, each element is explored individually.

*Logging*: As mentioned above, this layer gathers raw data from every robot system (OS, middleware, applications). Below an illustration of such data is shown: messages from the robot's navigation stack, or its localization system.

```
[Adapted] Received new location requests (x) 25, (y) 33
[Debug] Sending wheel request
[Adapted] [Navigation] Starting drive now
[Debug] [Navigation] Moving the four wheels at speed 1
[Debug] I see yellow net at: 21 x, 33 y
[Debug] I see yellow net at: 22 x, 33 y
[Debug] I see yellow net at: 23 x, 33 y
[Debug] I see yellow net at: 24 x, 33 y
[Adapted] [Localization] Reaching position (x) 25, (y) 33
[Adapted] [Perception] Selected object 12
```

***Event***: The logging layer also captures the event-related information. Despite the number of events can be huge, the amount of data is lower than in the logging layer.

```
[Adapted] Robot Leia wants see ball
[Adapted] Robot Leia wants see blue net
[Adapted] Robot sees yellow net
```

*Behaviour*: The behaviour logging is not different from previous, with elaborated behaviour-related information and represents actions running.

```
[Adapted] [state 1] Searching Blue Net.
[Adapted] [state 2] Searching Yellow Net.
[Adapted] [state 3] Searching Ball.
```

This example shows a representation of the information gathered from the ROS logging engine. Nevertheless, in this research, it is proposed the use of visualization tools for guaranteeing $\mathscr{M}_1$and ($\mathscr{M}_2$), simplifying the accessibility to auditors.

In our research, event-related data is visualized by using a tool we have developed for such a purpose and the information about behaviour is obtained.

*Event*: Figure 4 (right) illustrates the event layer. It shows the Computational graph of the system. It is a rqt package of the ROS framework. It depicts an illustrative abstraction of each object's location in the World. ROS uses the tf engine for managing relative locations. Specifically, it shows a scene in which the robot, the goals, and the balls, have already a relative location in the World. Besides, it shows an action carried out by the robot: the robot (Leia) sees an object (orange ball): $|\text{Leia}| \xrightarrow{\text{sees}} |\text{ball}|$.

*Behaviour*: Figure 4 (left) illustrates the behaviour layer. It shows a workflow of finite state machines through the VICODE tool (VIsual COmponent DEsigner) (Martín et al., 2013). The tool allows robot behaviour generation (and also source code generation) and it also provides

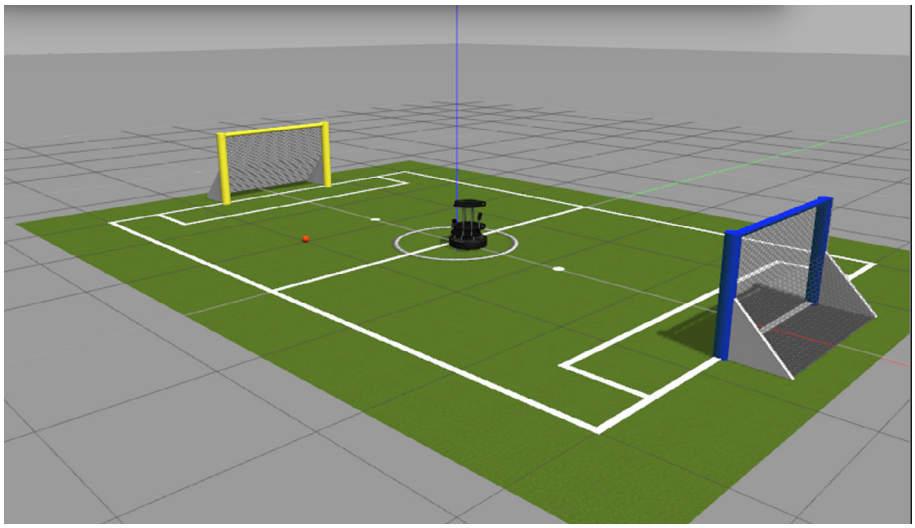

**FIGURE 3** Gazebo simulator with Turtlebot 2 robot deployed in a soccer field

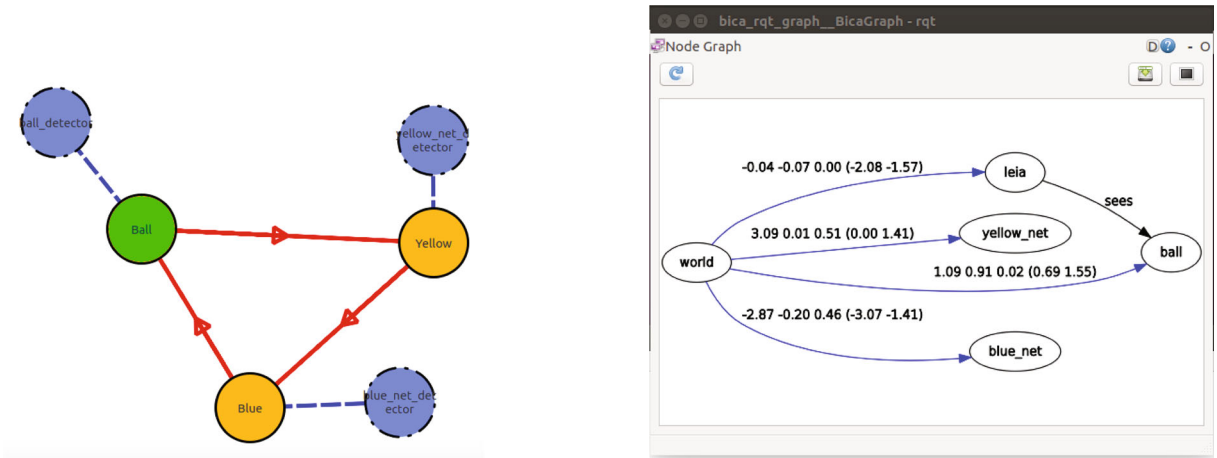


**FIGURE 4** Visualization tools involved in event and behaviour accountability dimensions: (Left) Visual accounting through VICODE tool. It illustrates robot behaviour at finite state machine stile. (Right) Visual accounting through knowledge graph tool. It presents the set of events triggered in a robot

visualization tools for comprehending robot behaviours easily. The finite state machines depict a set of behaviour related to the search for different objects. It is supported on three elements: (a) States: the robot looks for the yellow goal, the blue goal, and again the ball; (b) Initial state: the robot starts looking for an orange ball; and (c) Final state: the application shuts down. The system iterates between states over time; every 15 s, the control architecture changes the robot state; therefore, changing the robot behaviour (green colour highlights the current state). Every state is associated with a blue ball that defines the software component running the robot behaviour at a specific time.

For the purpose of ensuring future compatibility, the main challenge we have to face and evaluate is to handle system performance when deploying an accountability system based on logs. The high number of software components in a distributed middleware, such as ROS, generates an overwhelming amount of information. Thus, when adapting logs it is necessary to avoid missing information and protect the mapping between robot behaviours and logs. Thus, a DEBUG-mode logging engine is too demanding in terms of computing, and it would affect the robot's performance. On the other hand, an ADAPTED-mode logging engine does not allow for reaching the PM. The study proposed here introduces an alternative to external log managers, such as Kafka (Kreps et al., 2011) that has been explored by the authors (Fernández-Becerra et al., 2021). Instead, the ROS core is modified to adapt the logs and store them not only on-board the robot but also in the cloud. The next experiment will explore the system performance under different strategies in order to guarantee Perfect Mapping in a robot accountability system.

## 2.2 | Experimentation

In order to explore further the logs' conduct, it is proposed a new experimental layout. It is proposed an experiment which is based on moving a Turtlebot robot in a simulated apartment. We have defined a set of points that the robot has to reach. Thus, the robot is navigating between these points chosen randomly. After navigating to each point, the robot starts rotating, simulating that it is inspecting its surroundings.

To evaluate the logging behaviour in this scenario we have tested the performance of our solution by comparing it with the ROS default configuration on seven alternative assessment scenarios.

### 2.2.1 | Assessment scenarios

We have selected seven assessment scenarios to evaluate our proposal. Each one defines a different configuration for the accountability engine. We have carried out a single experiment with every scenario. In our experiment, the robot navigates through different points of the mock-up apartment at Leon@home Testbed,[3] an official testbed of the European Robotics League. The task is carried out for 15 min. Such a time period will provide us with an overview of the performance and the impact on the system.

Scenarios 1 and 2 state the comparison point with other configuration schemes.

1. Baseline_File. This scenario matches the default accountability engine in ROS where log data are stored in files. Besides, it gathers all the information managed by ROS Topics.
2. Baseline_MongoDB. In this scenario, we also use the default accountability engine in ROS, but instead of dumping log data to files, we use a NoSQL database (MongoDB) to store logging data. The entries of the database are based on the ROS logging messages. As a result, they are composed of the username, the ROS Master URI, the ROS IP, the ROS Hostname, the logging level; the logging message text; and the file, the function, the ROS node and the line where it takes place.

The following scenarios proposes two alternatives for evaluating the system performance with high-volume logging schema.

3. File_Adapted. This scenario also poses a dump-to-file approach. Unlike scenario 1, it is not stored the topic information of the node generating the logs. This is because in every ROS 1 log the information from topics is gathered and added into the logs, which is irrelevant for the Perfect Mapping process. It is used for evaluating the ratio of processing data.
4. MongoDB_Dates. This scenario is similar to the above, but as in scenario 2, we use a NoSQL database to gather accountability data slightly modified (Figure 5). In addition to the basic ROS message, it is included two timestamps: creation_date, which defines when the ROS message was generated; and save_date, which is stored in the message just before committing into the database. This will help us to understand the time that a message has been queued in the system.

Traditional deployments of accountability systems use local filesystems to store log data. We want to evaluate the benefits of cloud storage for logging data. A cloud log data repository provides a custom-built and high-performance infrastructure that ensures the logging data availability out of the robot. Similar approaches have been proposed using Blockchain methods (White et al., 2019). However, here is suggested an alternative supported in MongoDB. We promote the usage of MongoDB since not only it can store a vast volume of data, but also it has a set of query


```
_id: ObjectId("6149c037619ed2b2a33299a9")
username: "ubuntu"
ros_master_uri: "http://127.0.0.1:11311"
ros_ip: "127.0.0.1"
ros_hostname: ""
level: "INFO"
node: "/talker"
msg: "hello world 1"
file: "/tmp/binarydeb/ros-noetic-roscpp-tutorials-0.10.2/talker/talker.cpp"
function: "main"
line: 111
creation_date: "21/09/2021, 13:21:27.948245"
save_date: "21/09/2021, 13:21:27.952750"
```


**FIGURE 5** Message proposal for MongoDB

engines and indexing features that simplifies the accountability process. Besides, we have used Multi-Thread (MT) to parallelize the storage of the logs. The following scenarios aim to evaluate the performance of accountability systems with cloud storage.

5. MongoDB_Cloud. This scenario moves the logging data database to a cloud custom environment.
6. MongoDB_Cloud (MT). This scenario is similar to the above but it uses an enhanced version of our software supporting Multi-Thread.
7. MongoDB_DBaaS. This scenario proposes the use of a free-of-use MongoDB system deployed in the cloud. Specifically, we use MongoDB Atlas.

#### 2.2.2 | Metrics

We have used several metrics to benchmark the impact of the accountability system over the above scenarios. Regarding the logs, we have considered the overall number of logs processed during an experiment and their size. Concerning the system performance, we consider not only CPU and disk usage, but also network and messages sent and dropped. Relating to the performance of software components, it is important to point out that our solution overwrites the ROS default logging engine (Rosout). Thus, this study analyzes the impact of the accountability system just focusing on this component in each scenario.

We use Dstat to gather data for the evaluation. It replaces vmstat, iostat, netstat and ifstat. Dstat overcomes some of its limitations and includes some extra features. It is handy for monitoring systems during performance tuning tests, benchmarks or troubleshooting.

#### 2.2.3 | Hardware description

The onboard evaluation was carried out on a desktop with 16 GB RAM of DDR4. It has an Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz with 4 cores and is able to run 2 threads per core. On the other hand, the network card is an Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller. Finally, the disk used was a Western Digital Blue Hard Disk, specifically WDC WD10EZEX-08WN4A0.

#### 2.2.4 | Software artefacts

To address the accountability issue in ROS (Noetic), we have developed an alternative to the Rosout node. Rosout provides the logging feature for messages in ROS and it stores information in different log files attending the framework guidelines. We have selected a NoSQL database to store accountability data. Specifically, we use MongoDB (version 4.4.2). The communication with the database is deployed in the cloud both using TLS 1.3 in the MongoDB Cloud option and TLS 1.2 in the Atlas option. Our solution is developed in Ubuntu 20.04 and Python3 (specifically 3.8.5) and it is freely available online in GitHub.[4]

## 3 | RESULTS

This section summarizes the results of our experiments in order to find out which is the most suitable logging level in terms of performance using both on-board and cloud computing.

### 3.1 | Log benchmarking

Table 1 shows the number of log files as well as their overall size for assessment scenarios 1–7. The data establishes the average number logs into 1145967.143. Results on scenarios 1–4, and 6, are close to the mean value. Best performance is offered in scenario 2 where the higher number of logs are stored in the database. On the other hand, the size required for storing the logging information is higher than the average. Figure 6 allows for visualizing the differences between scenarios. This way, comparing scenarios 2 and 3, it is possible to see that increasing the data of the MondoDB entries, more storage is needed and fewer logs are stored. This also affects the file versions. Scenario 3, which has fewer data for each log, achieves a great number of logs than scenario 1. Finally, the use of a cloud solution affects drastically the amount of stored logs but it can be improved using Multi-Thread (MT).

**TABLE 1** Number, and overall size of log files

| Scenario | Logs | Size |
|---|---|---|
| 1_Baseline_File | 1,129,854 | 1096.916 MB |
| 2_Baseline_MongoDB | 2,011,471 | 488.100 MB |
| 3_File_Adapted | 1,681,427 | 180.320 MB |
| 4_MongoDB_Dates | 1,732,463 | 567.700 MB |
| 5_MongoDB_Cloud | 198,388 | 69.900 MB |
| 6_MongoDB_Cloud (MT) | 1,171,505 | 389.100 MB |
| 7_MongoDB_DbaaS | 96,662 | 34.000 MB |

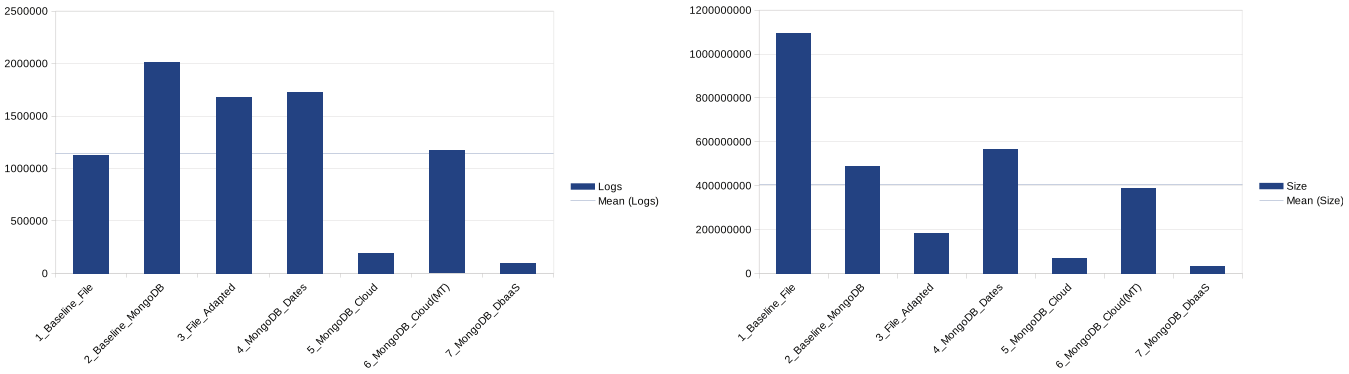


**FIGURE 6** Number, and overall size of log files

Attending to log benchmarking, scenario 2 is the best option when thinking about the number of logs and the size of the database is bounded slightly upper of the average. Considering cloud log data storage, the best option is scenario 6 since it is closest to the average values. Differences between the cloud options have to do with the 15-min time window. Without using Multi-Thread, the accountability system stores logs with lag. Besides, scenario 7 has an extra issue, the free version of the Atlas service has a 100 messages-per-second limit, motivating new delays in the system.

## 3.2 | System benchmarking

We performed a system data collection with the Dstat tool. The benchmarking gathers stats of CPU performance, Network performance, and disk usage.

### 3.2.1 | CPU performance

When dealing with CPU usage, Dstat provides CPU usage by user processes (usr), system processes (sys), as well as the number of idle (idl) and waiting for processes (wai). However, we have focused on the user and sys CPU usage. Table 2 summarizes the data gathered.

The CPU usage has an average value of 47.651 for usr time, and 7.238 for sys time. Firstly, usr time in scenario 1 presents the highest value, it is motivated by the huge amount of data that should be dumped to file. CPU usage values in scenarios 3, 5, and 7 are under the average when benchmarking sys time. However, the reasons are different, scenario 1 will use this time for data dumping, while scenarios 5 and 7 are running slightly slower because of not using MT features in scenario 5, and the maximum burst limit of the MongoDB Atlas free-tier in scenario 7.

### 3.2.2 | Network stats

Controlling the messages travelling through the network is a cornerstone in publish/subscribe paradigm used by ROS. It is important to know the total number of bytes received and sent on the robot interfaces. Table 3 shows the network performance on each scenario. Since are interested in the total flow, we consider the overall network usage provided by Dstat.

**TABLE 2** usr and sys CPU usage statistics

| | Usr | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 944 | 928 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 58.966 | 50.264 | 50.444 | 48.643 | 40.195 | 46.077 | 39.078 |
| Std. Deviation | 10.762 | 7.143 | 6.925 | 13.137 | 6.727 | 7.854 | 5.970 |
| Minimum | 8.000 | 4.000 | 6.000 | 2.000 | 5.000 | 6.000 | 5.000 |
| Maximum | 83.000 | 78.000 | 74.000 | 76.000 | 64.000 | 66.000 | 70.000 |
| | **sys** | | | | | | |
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Local** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 944 | 928 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 7.247 | 7.432 | 6.668 | 7.350 | 7.056 | 7.956 | 6.962 |
| Std. Deviation | 1.463 | 1.456 | 1.333 | 2.017 | 1.477 | 1.582 | 1.384 |
| Minimum | 1.000 | 1.000 | 0.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Maximum | 14.000 | 15.000 | 12.000 | 14.000 | 14.000 | 12.000 | 14.000 |

Considering the Sum value exhibit in recv data (Table 3). Data supports expected similarities between scenarios. Scenarios with local storage (1–4) present low data inference. The data also address the increase in data interchange for cloud scenarios (5–7) with a higher value for the Multi-Thread option. The Scatter diagram present in Figure 7 shows the correlation between received and sent data in each scenario. Besides, the scatter also shows the density diagram supporting the findings that data sent during the task is higher by the cloud solutions. Besides, they also show some message reception (recv) due to the fact that they are crossing TLS messaging with the database deployed in the cloud.

### 3.2.3 | Disk stats

Theoretically, the performance of a software application is limited by its I/O operations on disk. Besides, CPU activity must be suspended during a cycle of I/O completion. Using the tool Dstat we analyse the total number of reading and write operations carried out during the experiment. Table 4 summarize such data. The differences vary between scenarios: mean figures of reading and writing in scenario 3 (Files_Adapted) are lower than the ones of scenario 1 (Baseline_Files). On the other hand, scenario Baseline_MongoDB presents lower mean values than scenario 1. Besides, the use of cloud solutions produces different results. This way, scenario MongoDB_DBaaS has the greater mean values of reading and writing among the MongoDB solutions, while scenario 6_MongoDB_Cloud (MT) has the lower values.

### 3.2.4 | Message benchmarking

This metric aims to measure the number of messages delivered and dropped during the 15 min of the experiment paying attention to the message interchange of ROS nodes. The solution described above added new information to the Rosout node to maximize the description and ease the log storing process in the database. Thus, the metric is divided into two trends, the impact of the Rosout node and the impact on overall message interchange in ROS.

The delivered/dropped messages trend is presented in Table 5. The data shows significant differences in each scenario associated with their transmission rate. The scenario 6 sample presents the worst scenario attending the number of dropped messages; however, its mean rate is 0.06 which is not far from the 0.03 and 0.04 window that bounds the other 7 scenarios.

The data associated with delivered messages reveals significant differences in their performance. Results are divided into sets that present regular behaviour. Firstly, the data associated to dump to file scenarios 1 and 3, which maintain a mean rate of 19 messages. Then the cloud scenarios 5 and 7 present a mean rate of 34 and 23 messages. Scenario 6 presents a mean of 134 messages and beyond that, MongoDB working locally presents a mean ratio higher of 200 messages. This experimental design overviews a huge increase of ROS messages when dealing with MongoDB locally, so it is necessary to have it in mind when scaling the system.

## 3.3 | Rosout component benchmarking

It is investigated the influence of dumping the logging information to a MongoDB database exploring both local and cloud storage, as well as the limits imposed by single thread programming or the cloud storage. Our approach was designed to maximize and unify the ROS logging system Rosout both file and MongoDB logging storing processes. Thus, it is necessary to know the performance of this node along time during the assessment scenarios. Table 6 summarizes such information.

The delivered/dropped messages trend is outlined in Table 7 that presents the performance of the Rosout node during the 15-min experiment in the MongoDB scenarios. It shows that scenario 2 and scenario 4 produce the greatest value of delivered messages. This is due to the fact a message is sent after storing each log and both scenarios store more logs than scenario 5, scenario 6 and scenario 7. On the other hand, the data shows that the number of dropped messages is the same in all scenarios. As a result, the use of the different MongoDB scenarios does not affect the dropped messages. This is because all MongoDB scenarios use the same ROS communication mechanism that is based on an infinite queue of messages.

# 4 | DISCUSSION

The answer to our Research Question: What is the most suitable logging level in terms of performance, size and number of logs using both onboard and cloud computing?

**TABLE 3** Network descriptive statistics using Dstat

| | recv | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 944 | 928 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 68253.801 | 57894.142 | 157895.238 | 46971.150 | 144263.878 | 163944.950 | 55868.244 |
| Mode | 0.000 | 0.000 | 73000.000 | 156.000 | 28000.000 | 156000.000 | 32000.000 |
| Std. Deviation | 233939.353 | 158786.891 | 206936.320 | 173530.150 | 324247.477 | 36494.652 | 114840.973 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 1.868e + 6 | 1.221e + 6 | 1.562e + 6 | 1.878e + 6 | 1.900e + 6 | 283000.000 | 1.784e + 6 |
| Sum | 6.443e + 7 | 5.373e + 7 | 1.517e + 8 | 4.660e + 7 | 1.395e + 8 | 1.559e + 8 | 5.257e + 7 |
| | **send** | | | | | | |
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 944 | 928 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 6802.579 | 6248.233 | 6383.887 | 5966.762 | 132707.249 | 758112.343 | 81768.310 |
| Mode | 3561.000 | 3561.000 | 11000.000 | 3655.000 | 132000.000 | 848000.000 | 80000.000 |
| Std. Deviation | 18841.425 | 15818.480 | 4262.636 | 6179.513 | 15972.320 | 171086.970 | 8362.710 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 554000.000 | 473000.000 | 33000.000 | 66000.000 | 235000.000 | 1.264e + 6 | 175000.000 |
| Sum | 6.422e + 6 | 5.798e + 6 | 6.135e + 6 | 5.919e + 6 | 1.283e + 8 | 7.210e + 8 | 7.694e + 7 |

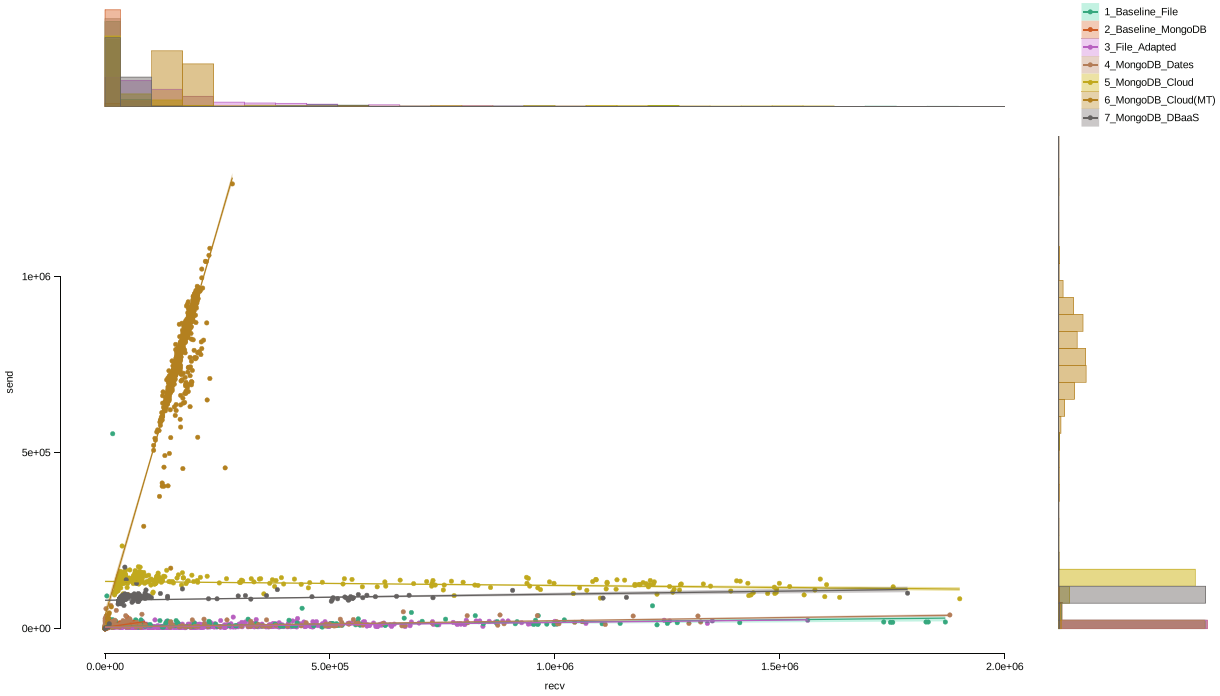


**FIGURE 7** Network benchmarking bytes received and sent

The Rosout performance is summarized, attending memory and CPU. Firstly, it was examined the real mean memory expended in each scenario. All database options require extra memory. The results present about double in regular MongoDB scenarios and triple when the process of sending messages is optimized with Multi-Thread. This scenario is generated because of the high number of messages queued during the logging process. The left diagram of Figure 8 illustrates the behaviour. On the other hand, the right image (Figure 8) illustrates CPU requirements. In this case, the consumption is greater in the baseline (around 25% more) and 3_File_Adapted scenarios. This is due to the fact that they are processing a huge number of messages and their dump to files.

In order to appraise the impact of the log in Rosout it is depicted the Figure 9, which rates the performance of the Rosout component using three metrics: total CPU usage, memory employed and total messages managed (it only applies to MongoDB scenarios). Previous findings support the notion that storing files has a huge effect on the CPU mean load but on the contrary, it has less influence on real memory mean. When dealing with the number of messages processed, scenarios 2_Baseline_MongoDB processes 78% more, 3_File_Adapted 48% more and 4_MongoDB_Dates achieves 53% better performance. Although the 6_MongoDB_Cloud(MT) performance is similar to baseline, and the percentage of space required is almost 75% lower, the effects on memory and CPU load we would advise against its use in very restrictive hardware environments.

Since the previous chart is too narrow, focusing on Rosout collateral effects, it depicts a new Spider diagram in Figure 10. It summarizes the impact of each scenario against a set of Log, System and Message metrics. The scoring system is calculated by applying a normalization from 1 to 5 (1 the worst to 5 the better).

Attending the total CPU usage, all options reduce around 20% the general CPU performance. Following the log data size and the total number of logs generated, and establishing a scoring system based on % of enhancement, scenarios 2, 3 and 4 get the best score. Network usage is clearly coped by cloud systems, particularly by scenario 6 where Multi-Thread multiplies the mean amount of bytes sent to the net observing around 6k in local scenarios (1,2,3,4), 132k in scenario 5, 758k in scenario 6. Scenario 7 illustrated the classical issue associated with cloud services, the option chosen (the free one) does not cover the system needs fixing a maximum number of messages per second. Then disk usage (read and write processes) presented a low number on the written option on Cloud scenarios 5 and 6. This is also happening in scenario 3 given the reduction in log size. Finally, the number of messages sent in each scenario where scenarios 2, 4 and 6 present 11, 10 and 6 times more in the number of messages. Besides, the dropped messages are again a cornerstone. In order to guarantee the Perfect Mapping, this number has to be minimized, and in this case, the Multi-Thread should be handled with care. The reason is that it increases the dropped messages ratio to a 116%, instead of alternative scenario balance in 16% and 38% except in scenario 3, where the ratio of dropped messages decreases a 16% which would guarantee the PM process.

Attending scenarios 1 and 6 are devoted to those systems with high spec characteristics. On the contrary, scenarios 2 and 4 would provide a reliable manner of storing the logging system locally, as previous researchers have shown (Niemueller et al., 2012). Thus, using our MT approach

**TABLE 4** Statistics of disk usage

| | read | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 944 | 933 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 447469.415 | 257761.415 | 211878.710 | 206337.589 | 318229.468 | 196264.757 | 635690.278 |
| Mode | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Std. Deviation | 2.517e + 6 | 1.876e + 6 | 1.816e + 6 | 1.202e + 6 | 1.690e + 6 | 1.452e + 6 | 2.300e + 6 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 2.300e + 7 | 2.300e + 7 | 2.500e + 7 | 1.800e + 7 | 1.700e + 7 | 2.100e + 7 | 1.900e + 7 |
| Sum | 4.224e + 8 | 2.405e + 8 | 2.036e + 8 | 2.047e + 8 | 3.077e + 8 | 1.866e + 8 | 5.982e + 8 |
| | **writ** | | | | | | |
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaSaS** |
| Valid | 944 | 933 | 961 | 992 | 967 | 951 | 941 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 2.559e + 6 | 1.352e + 6 | 586369.207 | 1.505e + 6 | 582782.139 | 252778.086 | 1.141e + 6 |
| Mode | 0.000 | 1.164e + 6 | 0.000 | 1.120e + 6 | 0.000 | 0.000 | 0.000 |
| Std. Deviation | 9.168e + 6 | 910039.458 | 1.716e + 6 | 1.270e + 6 | 1.606e + 6 | 803786.928 | 4.456e + 6 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 7.600e + 7 | 8.756e + 6 | 1.700e + 7 | 1.000e + 7 | 2.200e + 7 | 5.932e + 6 | 6.300e + 7 |
| Sum | 2.416e + 9 | 1.261e + 9 | 5.635e + 8 | 1.493e + 9 | 5.636e + 8 | 2.404e + 8 | 1.074e + 9 |

**TABLE 5** Delivered and dropped messages in each scenario

| | delivered_msgs | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Dates** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 6752 | 9520 | 7192 | 9266 | 10,022 | 10,163 | 9774 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 19.928 | 225.063 | 19.794 | 201.560 | 34.720 | 134.324 | 23.785 |
| Mode | 11.000 | 11.000 | 11.000 | 11.000 | 11.000 | 11.000 | 11.000 |
| Std. Deviation | 13.398 | 831.806 | 13.349 | 1036.658 | 31.619 | 428.802 | 13.236 |
| Minimum | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Maximum | 138.000 | 5799.000 | 161.000 | 7899.000 | 459.000 | 3007.000 | 133.000 |
| Sum | 134557.000 | 2.143e + 6 | 142355.000 | 1.868e + 6 | 347964.000 | 1.365e + 6 | 232474.000 |
| | **dropped_msgs** | | | | | | |
| | **1_Baseline_File** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_File** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud (MT)** | **7_MongoDB_DBaaS** |
| Valid | 6752 | 9520 | 7192 | 9266 | 10,022 | 10,163 | 9774 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 0.042 | 0.035 | 0.033 | 0.041 | 0.038 | 0.060 | 0.040 |
| Mode | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Std. Deviation | 0.232 | 0.215 | 0.201 | 0.235 | 0.224 | 0.300 | 0.380 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 4.000 | 5.000 | 3.000 | 4.000 | 4.000 | 4.000 | 22.000 |
| Sum | 283.000 | 330.000 | 238.000 | 382.000 | 382.000 | 613.000 | 392.000 |

**TABLE 6** Statistics of Rosout CPU and memory usage

| | cpu_load_mean | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1_Baseline** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Local** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud(MT)** | **7_MongoDB_DBaaS** |
| Valid | 465 | 276 | 299 | 417 | 413 | 431 | 416 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 105.857 | 79.642 | 102.726 | 75.156 | 13.660 | 87.568 | 7.901 |
| Mode | 108.495 | 82.500 | 104.490 | 82.500 | 12.000 | 96.995 | 6.500 |
| Std. Deviation | 14.941 | 11.744 | 16.599 | 22.493 | 2.505 | 14.996 | 1.779 |
| Minimum | 0.000 | 0.505 | 0.000 | 0.000 | 3.990 | 1.500 | 5.500 |
| Maximum | 111.020 | 84.000 | 109.010 | 85.510 | 25.500 | 110.995 | 19.495 |
| Sum | 49223.675 | 21981.252 | 30714.930 | 31340.051 | 5641.705 | 37741.609 | 3286.779 |
| | **real_mem_mean** | | | | | | |
| | **1_Baseline** | **2_Baseline_MongoDB** | **3_File_Adapted** | **4_MongoDB_Local** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud(MT)** | **7_MongoDB_DBaaS** |
| Valid | 465 | 276 | 299 | 417 | 413 | 431 | 416 |
| Missing | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mean | 2.422e + 7 | 5.376e + 7 | 2.361e + 7 | 5.491e + 7 | 5.153e + 7 | 8.070e + 7 | 5.146e + 7 |
| Mode | 2.401e + 7 | 5.396e + 7 | 2.362e + 7 | 5.536e + 7 | 5.236e + 7 | 8.306e + 7 | 5.220e + 7 |
| Std. Deviation | 576714.909 | 822321.068 | 91560.919 | 1.057e + 6 | 1.620e + 6 | 5.842e + 6 | 2.055e + 6 |
| Minimum | 2.336e + 7 | 4.858e + 7 | 2.308e + 7 | 4.842e + 7 | 4.627e + 7 | 4.906e + 7 | 4.490e + 7 |
| Maximum | 2.587e + 7 | 5.562e + 7 | 2.389e + 7 | 5.593e + 7 | 5.238e + 7 | 9.218e + 7 | 5.247e + 7 |
| Sum | 1.126e + 10 | 1.484e + 10 | 7.058e + 9 | 2.290e + 10 | 2.128e + 10 | 3.478e + 10 | 2.141e + 10 |

**TABLE 7** Rosout delivered and dropped messages in MongoDB scenarios

| | delivered_msgs | | | | |
|---|---|---|---|---|---|
| | **2_Baseline_MongoDB** | **4_MongoDB_Local** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud(MT)** | **7_MongoDB_DBaaS** |
| Valid | 2977 | 2531 | 2905 | 3064 | 2756 |
| Missing | 0 | 0 | 0 | 0 | 0 |
| Mean | 673.653 | 683.642 | 71.318 | 399.896 | 34.095 |
| Mode | 36.000 | 91.000 | 82.000 | 36.000 | 35.000 |
| Std. Deviation | 1385.581 | 1901.352 | 33.430 | 713.170 | 6.410 |
| Minimum | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Maximum | 5799.000 | 7899.000 | 459.000 | 3007.000 | 133.000 |
| Sum | 2.005e + 6 | 1.730e + 6 | 207178.000 | 1.225e + 6 | 93966.000 |
| | **dropped_msgs** | | | | |
| | **2_Baseline_MongoDB** | **4_MongoDB_Local** | **5_MongoDB_Cloud** | **6_MongoDB_Cloud(MT)** | **7_MongoDB_DBaaS** |
| Valid | 2977 | 2531 | 2905 | 3064 | 2756 |
| Missing | 0 | 0 | 0 | 0 | 0 |
| Mean | 0.002 | 0.003 | 0.002 | 0.002 | 0.003 |
| Mode | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Std. Deviation | 0.048 | 0.053 | 0.049 | 0.048 | 0.050 |
| Minimum | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Maximum | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Sum | 7.000 | 7.000 | 7.000 | 7.000 | 7.000 |

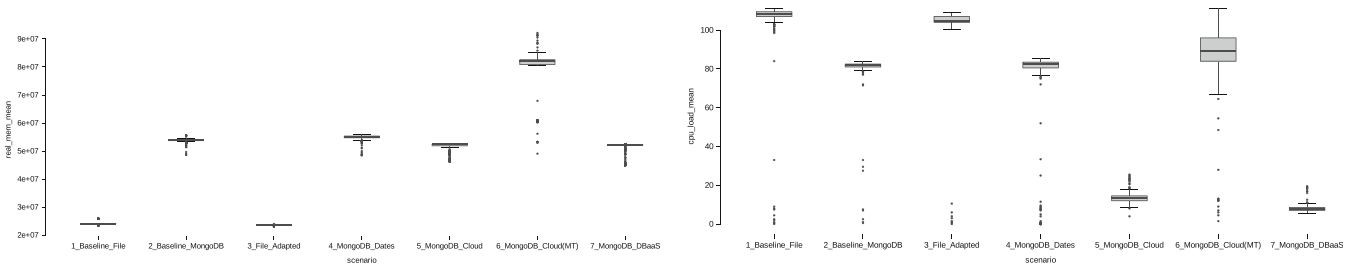


**FIGURE 8** Rosout cpu mean and real memory mean performance per each scenario

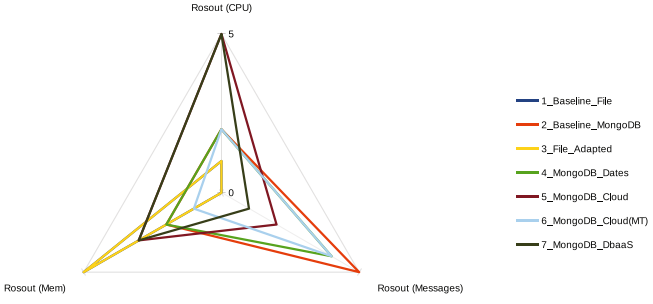


**FIGURE 9** Spider chart evaluation for Rosout

will provide a reliable manner of storing the logs outside the robot would be considered a good praxis; however, it is necessary to consider the impact of Multi-Thread and check third party services when needed.

Overall, the dump to file option (scenarios 1 and 3) has more data redundancy and provides reduced flexibility in retrieving data. Instead, when using the database it is provided extended flexibility in accessing data and offers the possibility of normalizing ROS logs.

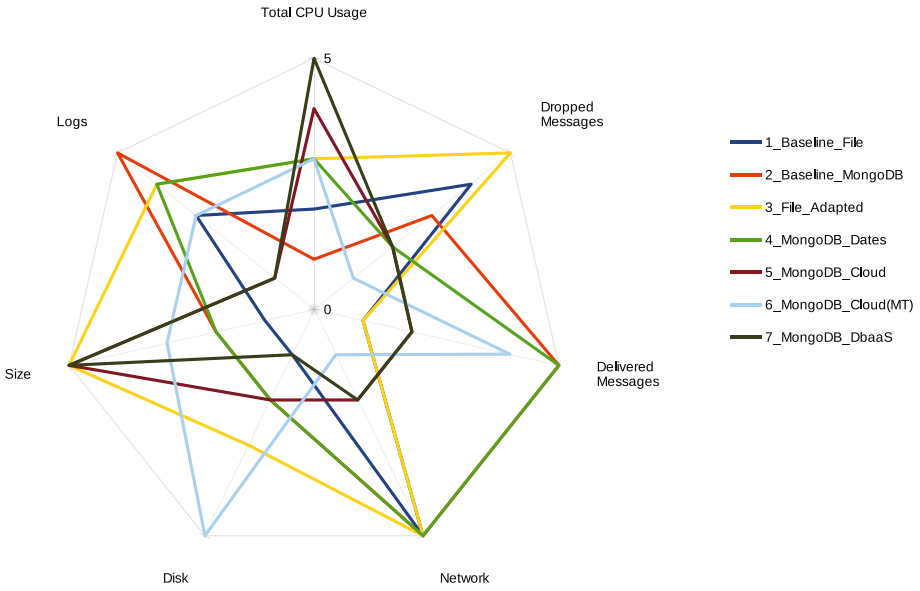


**FIGURE 10** Spider chart evaluation of assessment scenarios and main features analysed

On the whole, the challenges of these kinds of approaches, on-board or in the cloud layer in three different aspects: specification, interpretability and performance. The most critical is the specification of a single message type protocol for all developers. This specification has to define those basic elements that guarantee the process of signalling the component that generates a robot behaviour including not only the proprioception and reasoning system but also the world knowledge that causes the event.

The second crucial element is how human-interpretable should be these logs. Here we have proposed different types of interpretability attending three layers of abstraction; however, even with this approach, it will be necessary to provide some level of explanation for some of the logs opening a question about, who should write the logs, a software developer or a linguist.

The third vital aspect of including the perspective of accountability in the robot is the performance impact. The addition of onboard elements for being fully accountable with Perfect Mapping would be a killer to the decision-making system in those robots running all software on-board.

These challenges should be overcome in order to avoid unreasonable criminal punishment on the programmer or manufacturer, who might not specifically intend or even foresee, the robot's commission of wrongful acts (Docherty, 2015).

## 5 | CONCLUSIONS

This paper depicted the three-dimensional accountability model for autonomous robots. The main reason for dealing the accountability using a multi-layer model is the multi-dimensional challenges associated with logging each robot action. Achieving the Perfect Mapping between the robot's actions and the logging data dumped by its software components will clear the way towards explainability in robotics.

All things considered, the impact on the robot system when the middleware is customized for deploying a logging system using both on-board and a cloud computing approach is remarkable attending each of the alternatives. This research considered six options and measured their impact mainly in the performance perspective, analysing the impact on a robot using the facto standard ROS, and a MongoDB approach in a local, cloud and a fully commercial off-the-shelf cloud-based database solution such as Atlas.

The findings present a straightforward conclusion, a verbose logging system drives to serious performance issues on the platform, and the detailed information provided is far from explaining the robot's behaviour. Besides, large volumes of data might not be managed properly in specific situations; also, the access to such information sacrifices the privacy of the human-robot interaction. Future work will focus on the message normalization and the links between the different layers to create an accountability system flexible enough to be used among different actors and situations, and specific enough to understand contextual particularities and unique relationships between robot behaviours events and software components. As a result, the robot will provide the mechanisms that help manufacturers, client/user and developers to recognize the reasons that trigger the robot behaviour as well as the impact that this explainability in robot behaviours has on individuals.

## ENDNOTES

[1] https://www.mongodb.com/cloud/atlas.

[2] https://youtu.be/VoYLna-S-oI.

[3] https://robotica.unileon.es/index.php?title=Testbed.

[4] https://github.com/mgonzs13/ros_comm/tree/noetic-devel/tools/rosout.

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## ORCID

*Francisco Javier Rodríguez-Lera* https://orcid.org/0000-0002-8400-7079

*Miguel Ángel González-Santamarta* https://orcid.org/0000-0002-7658-8600

*Ángel Manuel Guerrero-Higueras* https://orcid.org/0000-0001-8277-0700

*Francisco Martín-Rico* https://orcid.org/0000-0003-3121-5744

*Vicente Matellán-Olivera* https://orcid.org/0000-0001-7844-9658

## REFERENCES

Attard, J., Orlandi, F., Scerri, S., & Auer, S. (2015). A systematic review of open government data initiatives. *Government Information Quarterly*, *32*(4), 399–418.

Bihlmaier, A., Hadlich, M., & Wörn, H. (2016). In A. Koubaa (Ed.), *Advanced ROS Network Introspection (ARNI)* (pp. 651–670). Springer International Publishing. https://doi.org/10.1007/978-3-319-26054-9_25

Butin, D., Chicote, M., & Le Métayer, D. (2013). Log design for accountability. In *2013 IEEE security and privacy workshops* (pp. 1–7). IEEE.

Carolan, L. (2016). *Open data, transparency and accountability: Topic guide*. GSDRC, University of Birmingham.

Deber, R. B. (2014). Thinking about accountability. *Healthcare Policy*, *10(SP)*, 12–24.

DesRuisseaux D. (2018). Practical overview of implementing IEC 62443 security levels in industrial control applications. Schneider Electric White Paper.

Ding, R., Zhou, H., Lou, J. G., Zhang, H., Lin, Q., Fu, Q., Zhang, D., & Xie, T. (2015). Log2: A {Cost-Aware} logging mechanism for performance diagnosis. In *2015 USENIX annual technical conference (USENIX ATC 15)* (pp. 139–150). USENIX.

Docherty, B. L. (2015). *Mind the gap: The lack of accountability for killer robots*. Human Rights Watch short reports, Universal, Human Rights Watch. https://books.google.es/books?id=HGeHAQAACAAJ

Fernández-Becerra, L., Guerrero-Higueras, Á. M., Rodríguez-Lera, F. J., & Fernández-Llamas, C. (2021). Analysis of the performance of different accountability strategies for autonomous robots. In *Computational intelligence in security for information systems conference* (pp. 41–51). Springer.

Kapitanovsky, A., & Maimon, O. (1993). Robot programming system for assembly: Conceptual graph-based approach. *Journal of Intelligent and Robotic Systems*, *8*(1), 35–62.

Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB* (Vol. 11, pp. 1–7). ACM.

Lee, S., Jo, H. J., Choi, W., Kim, H., Park, J. H., & Lee, D. H. (2020). Fine-grained access control-enabled logging method on ARM TrustZone. *IEEE Access*, *8*, 81348–81364.

Lima, O., Ventura, R.. (2018). *ICAPS-2018 tutorial on integrating classical planning and mobile service robots using ROSPlan*. Institute for Systems and Robotics. https://github.com/oscar-lima/rosplan_tutorial

Manso, L. J., Bustos, P., Bachiller, P., & Núñez, P. (2015). A perception-aware architecture for autonomous robots. *International Journal of Advanced Robotic Systems*, *12*(12), 174.

Martín, F., Agüero, C., Cañas, J. M., Abella, G., Benítez, R., Rivero, S., Valentí-Soler, M., & Martinez-Martin, P. (2013). Robots in therapy for dementia patients. *Journal of Physical Agents*, *7*(1), 48–55.

Matellán, V., Lera, F. J. R., Martín, F., Ginés, J., & Guerrero-Higueras, Á. M. (2021). The role of cybersecurity and hpc in the explainability of autonomos robots behavior. In *2021 IEEE International Conference on Advanced Robotics and Its Social Impacts (ARSO)*. IEEE.

Miller K. A; 2017*New Framework for Robot Privacy*.

Niemueller, T., Lakemeyer, G., & Srinivasa, S. S. (2012). A generic robot database and its application in fault analysis and performance evaluation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 364–369). IEEE.

Oreback, A. (1999). *Components in intelligent robotics*. Technical report, Royal Institute of Technology, Stockholm.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: An open-source robot operating system. In *ICRA workshop on open source software* (Vol. 3, p. 5).

Rodríguez, V., Rodríguez, F. J., & Matellán, V. (2011). Localization issues in the design of a humanoid goalkeeper for the RoboCup SPL using BICA. In *11th International Conference on Intelligent Systems Design and Applications* (pp. 1152–1157). IEEE.

Rodríguez-Lera, F. J., Guerrero-Higueras, Á. M., Martín-Rico, F., Gines, J., Sierra, J. F. G., & Matellán-Olivera, V. (2019). Adapting ROS logs to facilitate transparency and accountability in service robotics. In *Iberian Robotics Conference* (pp. 587–598). Springer.

Rodríguez-Lera, F. J., Guerrero-Higueras, Á. M., Martín-Rico, F., Gines, J., Sierra, J. F. G., & Matellán-Olivera, V. (2020). Adapting ROS logs to facilitate transparency and accountability in service robotics. In *Robot 2019: Fourth Iberian robotics conference Cham* (pp. 587–598). Springer International Publishing.

Rodríguez-Lera, F. J., Santamarta, M. A. G., Guerrero, A. M., Martín, F., & Matellán, V. (2020). Traceability and accountability in autonomous agents. In *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020) Burgos, Castilla y León, España* (pp. 295–305). Springer. https://doi.org/10.1007/978-3-030-57805-3_28

Shishkov, B., Hristozov, S., Janssen, M., & van den Hoven, J. (2017). Drones in land border missions: Benefits and accountability concerns. In *Proceedings of the 6th International Conference on Telecommunications and Remote Sensing* (pp. 77–86). ACM.

Theodorou, A., & Bryson, J. (2017). ABOD3: A graphical visualisation and real-time debugging tool for bod agents. In *CEUR workshop proceedings* (Vol. 1855, pp. 60–61). CEUR-WS.

Theodorou, A., Wortham, R. H., & Bryson, J. J. (2017). Designing and implementing transparency for real time inspection of autonomous robots. *Connection Science*, *29*(3), 230–241.

Van Harmelen, F., Lifschitz, V., & Porter, B. (2008). *Handbook of knowledge representation* (Vol. 1). Elsevier.

White, R., Caiazza, G., Cortesi, A., Im Cho, Y., & Christensen, H. I. (2019). Black block recorder: Immutable black box logging for robots via blockchain. *IEEE Robotics and Automation Letters*, *4*(4), 3812–3819.

Xiao, Y. (2009). Flow-net methodology for accountability in wireless networks. *IEEE Network*, *23*(5), 30–37.

Xiao, Z., Kathiresshan, N., & Xiao, Y. (2016). A survey of accountability in computer networks and distributed systems. *Security and Communication Networks*, *9*(4), 290–315.

Yoon, M. K., & Shao, Z. (2019). ADLP: Accountable data logging protocol for publish-subscribe communication systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* (pp. 1149–1160). IEEE.

Zender, H., Mozos, O. M., Jensfelt, P., Kruijff, G. J. M., & Burgard, W. (2008). Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, *56*(6), 493–502.

## AUTHOR BIOGRAPHIES

**Francisco Javier Rodríguez** received his PhD degree in intelligent systems for engineering in 2015 from the School of Industrial Engineering and Information Technology at University of León (Spain). Currently, he is working as a researcher in the Mobile Robotics Group at University of León (Spain) after two years working as a postdoctoral research associate in the AI Robolab, which belongs to the Department of Computer Science (DCS) at University of Luxembourg. Much of his research interest lies on developing technologies for social robots. Targeting human-robot interaction scenarios, he has been collaborating with different cognitive architectures development, dealing with cybersecurity issues associated with robot middleware and facing new trends in explainable and accountable robotics.

**Miguel Ángel González-Santamarta** received his Computer Science degree in 2020. He is currently finishing his Computer Science Master's. He is currently pursuing his PhD studies at the University of León. He has been a Research Associate in the Robotics Group at the Universidad de León since 2018. His research interests include cognitive robotics.

**Ángel Manuel Guerrero-Higueras** worked as IT engineer at several companies in the private sector (2000–2016), worked as researcher at the Atmospheric Physics Group at Universidad de León (2011–2013) and received his PhD at the University of León in 2017. He currently works as lecturer at Universidad de León and researcher at Research Institute of Applied Science to Cyber-Security. His main research interests include robotic software architectures, cyber-security, and learning algorithms applied to robotics.

**Francisco Martín Rico** is a member of the Signal Theory, Communications, Telematic Systems and Computation Department, and of the Communications and Systems Group. His research interests are in Artificial Intelligence, Mobile Robotics, Computer Vision, Behaviour Architectures, Assistive Robotics, Robot Localization and Navigation and related technologies. He has developed applications for real robotic systems such as Aibo, Nao Humanoid, Kobuki (turtlebot 2), RB-1, Tiago, Lego NXT, AR-Drone in languages such as C, C++, Java, Python, and Ada. He is expert in ROS, using technologies such as OpenCV, Point Cloud Library (PCL), among others. He has taught in courses such as Operating Systems, Advanced Programming, Real Time Systems, Communications and Robotics.

**Vicente Matellán Olivera** received his MSc and PhD from Universidad Politécnica de Madrid (1993, 1998). He was a lecturer at Universidad Carlos III de Madrid (1993–1999) and Assistant Professor at Universidad Rey Juan Carlos (1999–2008) and currently is Associate Professor at Universidad de León. He has been interested in robotics since he can remember, but also in almost any computer-related area. He is currently part of the Robotics Group at Universidad de León.

**How to cite this article:** Rodríguez-Lera, F. J., González-Santamarta, M. Á., Guerrero-Higueras, Á. M., Martín-Rico, F., & Matellán-Olivera, V. (2022). Towards explainability in robotics: A performance analysis of a cloud accountability system. *Expert Systems*, *39*(9), e13004. https://doi.org/10.1111/exsy.13004