Testing Service Robots in the Field: An Experience Report

Argentina Ortega[1,2] Nico Hochgeschwender[1] Thorsten Berger[2]

Abstract— **Service robots are mobile autonomous robots, often operating in uncertain and difficult environments. While being increasingly popular, engineering service robots is challenging. Especially, evolving them from prototype to deployable product requires effective validation and verification, assuring the robot's correct and safe operation in the target environment. While testing is the most common validation and verification technique used in practice, surprisingly little is known about the actual testing practices and technologies used in the service robotics domain. We present an experience report on field testing of an industrial-strength service robot, as it transitions from lab experiments to an operational environment. We report challenges and solutions, and reflect on their effectiveness. Our long-term goal is to establish empirically-validated testing techniques for service robots. This experience report constitutes a necessary, but self-contained first step, exploring field testing practices in detail. Our data sources are detailed test artifacts and developer interviews. We model the field testing process and describe test-case design practices. We discuss experiences from performing these field tests over a 10-month test campaign.**

I. Introduction

Service robots are increasingly becoming essential for our daily lives—not only during emergencies, such as the current pandemic. Their widespread use requires robotic software of much higher maturity, beyond the current state of practice, which is often limited to prototypes that are hardly maintainable and evolvable in the long term. In fact, the deployment of service robots in real-world applications requires engineers to efficiently evolve such prototypes and proof-of-concept systems into fully deployable products. Most importantly, developers need to demonstrate that these robots operate correctly and safely in their intended operational environment. Validation and verification techniques—most commonly testing—are especially difficult to realize for service robots, given their operation in uncertain environments, under conditions that can hardly be encoded in traditional test cases, such as unit tests. Unfortunately, while software testing is very well understood in traditional software domains, with a large set of testing techniques available, surprisingly little is known about testing practices for service robots.

In fact, service robots, as special cases of autonomous, cyber-physical systems, face unique challenges, requiring specialized testing techniques. Specifically, challenges arise from: (i) ad hoc development processes and the lack of sufficient technologies to integrate systems of systems [1]; (ii) the characteristics of robotic systems as safety-critical, real-time, and embedded systems of systems, among others, increasing their complexity, amount of integration testing, or risk management efforts [2]; (iii) the complexity of the operational environments, resulting in unpredictable corner cases, complicating test design that accounts for many possible conditions [3]; and (iv) as cyber-physical systems, the non-deterministic behaviors, noisy sensors, and AI components, challenging the definition of test oracles (expected results) [4], [3].

To advance software engineering technology, particularly testing, we need to improve our empirical understanding of real-world practices. Studying cases and reporting detailed experiences in publications will eventually allow building theories [5], [6] of effectively building respective systems. Specifically, eliciting testing practices, including identifying concrete challenges and solutions, facilitates addressing our long-term goal of building effective testing techniques for service robotics systems.

In this experience report, we focus on field testing, as a common technique to validate service robots. It facilitates evaluating the full system in its real runtime environment [3]. Field testing addresses the unlimited space of runtime behavior, which challenges defining test data, and especially observing and measuring actual robot behavior in real environments, as well as defining and comparing this behavior to test oracles. Although there are a handful of reports about the deployment of robots with long-term autonomy capabilities [7], [8], [9], [10], studies suggest that in general, the robotics software are validated or demonstrated in a laboratory environment [11].

Our subject system is an industrial-strength disinfection robot (Kelo AD). The goal of field tests is to assess the efficiency, reliability and safety of the robot in a real-world operational environment early in the development process. Unlike existing definitions of field tests [3] and other long-term deployments that report system-level tests, the reported approach uses real-world environments and inputs for component and integration level tests as well. To the best of our knowledge, there are no other reports that describe testing practices that bridge the gap between the development and deployment of a service robot.

Our contributions are: the models of the test processes of the field testing approach; the quantitative analysis of the test design practices compared to their execution; and a discussion of challenges and best practices, including aspects to consider for field testing that can be tailored to other robotic systems.

II. Related Work

Two of the biggest challenges in software engineering for service robots, namely robustness, and validation, are tightly related to testing [12], and testing of AI-based systems has

[1] Hochschule Bonn-Rhein-Sieg, Germany. {argentina.ortega, nico.hochgeschwender}@h-brs.de
[2] Ruhr University Bochum, Germany. thorsten.berger@rub.de
This work has partly been supported by the European Union's Horizon 2020 project SESAME (Grant nr. No 101017258), and the Graduate Institute of the Hochschule Bonn-Rhein-Sieg.

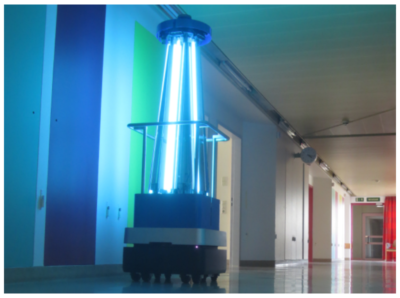


Fig. 1: Kelo AD disinfection robot in its target environment

challenges in and of itself [4], [3]. Studies summarize testing techniques and practices for autonomous systems [13], but rather on a general level without going into details of a particular system and setup. Other studies describe methodologies for integration testing based on simulated environments [14] or describe high-level testing processes [15].

Benchmarking has long been used to compare the performance of algorithms or systems. While competitions [16], [17], [18] strive to replicate real-world conditions for the tests, the test objectives have mostly focused on the performance of the system under test [19] as opposed to verifying the properties of a product that should operate autonomously over long periods of time. Reports of long-term deployments [7], [8], [9], [10] usually report system tests on TRL7 or 8, but little is known about the V&V practices employed by the developers and that likely played a key role in their success. Furthermore, few works report on the conformity of robots to safety standards [11].

Frameworks to test long-term autonomy [20], [21] using hardware-in-the-loop test are also promising; however, the adoption of these tools depends on the contextual factors that influence the testing process, their ability to support real-world environments on a variety of scales, and the level of detail of the test basis. Furthermore, even for cases where manual testing is needed, studying current practices can reveal opportunities for optimization [22].

III. Subject System

We now describe our subject—a.k.a., system under test (SUT)—with contextual factors(e.g., operational constraints, available resources, organizational policies, internal and external standards) that influence testing.

Autonomous Disinfection Robot. Our subject is the Kelo AD disinfection robot, shown in Fig. 1, developed by the company Kelo. Its hardware is designed in-house, featuring the omnidirectional base and six UVC lamps for disinfection of surfaces. To avoid overexposure of UVC to humans who share its environment, the robot is equipped with a 360° camera system used for people detection as a safety feature to turn off the UVC lamps in the vicinity of people.

The software stack is based on ROS; most packages have been developed by the company. These include the lamp control package, a people detection component used to turn off the lamps to avoid overexposure, and their navigation stack that performs the disinfection tasks based on pre-specified routes. Other components include sensor drivers and their ROS wrappers for the lasers, LiDAR, and cameras that are used for localization and obstacle avoidance.

TABLE I: Background details of interview participants

Participant	Position & Experience	Background
P01	Sr. Developer (13 yrs)	Computer Science
P02	Jr. Developer (3 yrs)	Mechatronics
P03	Jr. Developer (3 yrs)	Mechatronics

Organization and Testing Team. Kelo is a small start-up with 20 software developers, five of which are in charge of developing the ROS packages and other software components used by the robot. The team does not strictly follow agile practices as part of their organization, but has adopted parts of them into their development workflow, e.g. the duration of the sprints is variable according to the feature or issues currently being addressed. We interviewed the subset of the developers that are mostly in charge of the tests of all features; Table I shows the details of the participants interviewed in this study.

In parallel to the software development, hardware engineers develop some components of the robot. On a case-by-case basis, hardware engineers will join the testing efforts when their expertise is required to find the root cause of the issue or to solve hardware-related problems. Risk analysis activities are performed together with software developers, and include failure mode and effects analysis (FMEA) and hazard identification based on ISO 12100.

Development Toolkit. Several tools are used to support the testing activities of the robot. Risk analysis activities are documented using worksheets in Microsoft Excel. Test reporting is done on an in-house tool called “Technology Readiness Level Test Report Library” (TRL^2), and on a “testing” git repository where test incidents—“issues requiring further action that were identified during the test execution process” [23]—are documented. Log files are recorded using the rosbag tool[1] and only contain the subset of topics relevant to the test case being executed. They post-process these log files to have easy access to the metrics and store them as person detection logs. Figure 2 shows the information flow between different artifacts, and Table II a summary of the test work products we analyzed.

The tool TRL^2 relies on the ISO 29119 standard for

[1]`http://wiki.ros.org/rosbag/Commandline`

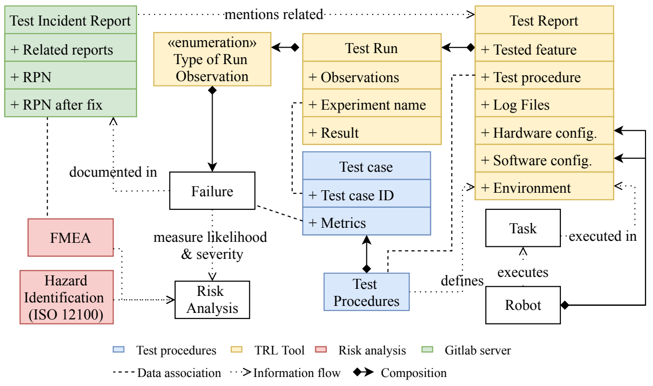


Fig. 2: Information flow for the Kelo AD tests

software testing, including defining test report templates. After a test, engineers manually capture each run, including the experiment name, robot versions (using wstool[2]), any failures, and the names of any log files, among other data. The tool was originally developed to improve the traceability between requirements, FMEA, software bugs, hardware issues, and test runs; however, in practice, it is only used to occasionally document the test runs because the time required to record and manually annotate test runs, along with the cost of development and maintenance of the tool itself, are prohibitive for such a small team.

IV. Field Testing Strategies

We now describe the test processes and activities used for field testing. The engineers follow two different test processes for the Kelo AD: *Exploratory field testing* and *Endurance field testing*. The process models we derived from the interviews are shown in Fig. 3. Table III summarizes the test activities with examples for each process.

Exploratory field tests. (Fig. 3a) follow short, agile-like development-test cycles. Exploratory testing is an experience-based technique where the tester designs and executes tests based on prior knowledge, previous tests, curiosity, and other heuristics for common failures [23]. The main goal of these tests is to cause new failures by trying new scenarios that cause the robot to deviate from its expected behavior or that cause undesired side effects. Exploratory field tests take advantage of the variability in the input data, i.e. real-world environments and the execution of tasks, to find corner cases, fix bugs on the spot, and identify improvements for existing features.

Obs. 1: Exploratory testing helped find defects caused by the variability in real-world inputs, to identify corner cases that could have been missed by the risk assessment.

Field endurance tests. (Figs. 3b and 3c) follow a more classical test process. Endurance tests are a type of efficiency testing, where the SUT operates at high levels of load for long periods of time [23]. In the robotics context, we define endurance tests as the operation of a robot, a subsystem, or component operating continuously in its operational environment. In practice, field endurance tests also incorporate elements of performance testing, e.g., to measure the time required to disinfect a variety of rooms; load testing to verify how the robot handles increasingly larger areas; stress testing, particularly of hardware components as in Hawes et al. [8]; and compliance testing to identify violations of safety requirements, e.g., overexposure to UVC light.

Obs. 2: Data collected from field tests was useable to validate multiple test types, factoring the amount of information gained given the limited number of testers available into the cost of testing.

Field endurance tests are carried out when features have reached certain maturity or stability, usually after several exploratory field sessions. When the tests are run at system integration or system level, they can be used to verify the autonomous operation of the robot as well.

Obs. 3: Exposing the robot early in the development process to real-world environments reduced the gap between development and deployment.

V. Design of Field Endurance Tests

Engineers designed the field endurance tests, covering **Test analysis (A4)** and **Test design (A5)** on Table III, based on software testing concepts from the literature [24], [2], [25], as follows. The Kelo engineers followed three main steps. First, they identified *test conditions* (testable system aspects) relevant for test objectives (system safety and reliability); then they defined or generated the preconditions, inputs, actions, expected results (test oracle), and post conditions of the SUT or the testers. Finally, engineers specified the *test data*.

Test Condition Identification. The first step for identifying *test conditions* is to review the *test basis*, i.e. the body of knowledge to be used as a source during test analysis and design [24]. To validate that the robot operates correctly and safely, engineers commonly treat the robot and its components as a black-box. Black-box testing techniques derive test cases from the specification of the SUT, which vary in levels of detail and formality; they are especially suited to evaluate the behavior of a system, regardless of how that system achieves the behaviour [25]. In general, specifications contain the required or desired behavior of the system and can include non-functional requirements, including reliability, performance, and safety of the robot.

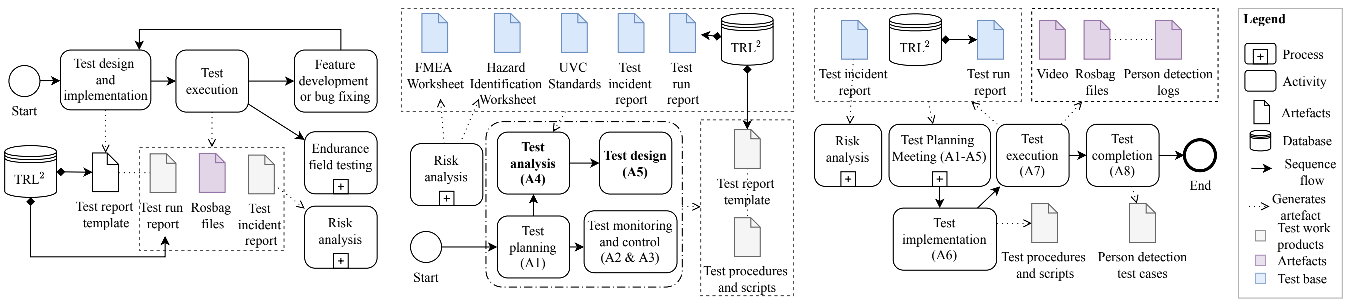
For the Kelo AD robot, the test basis includes the experience of developers, previous tests (including past test incident reports and reports in the TRL[2] tool), architecture and components of the robot, and the application use cases. Key Performance Indicators (KPI) about the expected performance of the robot are defined as part of their project planning; these include business requirements, like the time required to disinfect an area, but also elements of the organization's testing policy, such as the number of kilometers traveled by the robot for a given project milestone.

The risk analysis activities reveal defects of the system under test and help developers prioritize which features to test first. These activities add standards to the test basis: DIN EN 14255-1 and DIN EN 62471 are used to define the safety requirements related to overexposure to UVC light;

[2]`http://wiki.ros.org/wstool`

TABLE II: Test documents and artifacts

Source	Description
Test procedures	A markdown file that contains the specification of the endurance field test cases
Test reports	46 JSON files exported from an in-house tool along with the JSON schema that describes them
Rosbag files	A list with the file names of 1656 rosbag files, out of which we had access to 1517 bag files.
Test incident reports	16 test incident reports recorded as GitLab issues and exported through their API
Person detection test cases	A spreadsheet containing the test cases used to test the person detection models



(a) Exploratory field testing process.

(b) Endurance field testing process: Analysis and design

(c) Endurance field testing process: Execution

Fig. 3: Test processes for field testing. Activities are described in Table III

TABLE III: Test activities

ID	Test activity group	Description	Exploratory Field Testing	Field Endurance Testing
A1	Planning	Definition of testing objectives and testing approach	No formal planning, takes place two or three times a week.	Test plan and schedule are created. Tests take place full-time over a week
A2	Monitoring	Test monitoring of progress using metrics defined in the test plan.	-	Performed periodically and on demand by P01
A3	Control	Actions taken to meet the (revised) test objectives	-	Performed periodically and on demand by P01
A4	**Analysis**	Analysis of the test basis to identify test conditions	-	Test performance, reliability and compliance based on experience and risk analysis.
A5	**Design**	Definition of test cases, identification of test data and design of test environment	No formal test cases defined. Participants sometimes use existing test cases from endurance tests.	Equivalence partition is used to minimize test case explosion. Test data includes maps and task definitions but mostly consists of real-world inputs.
A6	Implementation	Creation of testware, e.g. test procedures or scripts, test environments, preparation of data, etc.	Modifying existing task definitions with challenging waypoints.	Definition of new disinfection routes. Setup of real-world environment, e.g. obstacles positions, sensor placement for ground truth.
A7	Execution	Running the test suite and recording their results.	P02 tries to find ways to break the robot.	All participants run tests. Results are recorded on a variety of tools and formats.
A8	Completion	Data collection for reporting and future tests	Adding relevant edge cases to field endurance test cases.	Selecting test cases to reuse for testing the person detection DNN.

and hazard identification is performed according to ISO 12100:2010, ISO/TR 14121-2:2012, and ISO 13482:2014. Similarly, FMEA is used to systematically analyze and document failure modes of both hardware and software components. The hazard identification and FMEA worksheets record estimates about the severity and likelihood of occurrence of events; the former also includes estimates about the frequency and probability of the hazard occurring; while FMEA worksheets also estimate the likelihood of detecting such a defect. Overall, this leads to tests that focus on two main aspects of the system: safety and reliability, i.e. measuring that the failure rates of components are acceptable according to standards and the risk assessment of the robot.

We identified three test conditions in their test procedures: person detection, navigation, and the disinfection workflow. The document is mostly written in natural language and is divided in three main sections corresponding to each of the test conditions. Each of these sections contains logical test cases targeting more specific test conditions, e.g. person detection with occlusions, which are then further refined into concrete test cases with specific values, such as occlusions from a desk or a wall. Logical test cases were used by developers to group concrete test cases according to their difficulty, e.g. by accounting for the interaction between components and how this influences the performance of the SUT. We classify these test cases according to their test levels and the TRL scale, as shown in Fig. 4. In this classification, TRL is a proxy for the complexities introduced by the operational environment, while the test levels are related to the complexity of the (sub)system under test. For example, TPerc01 (A and B) and TPerc04 focus solely on the person detection component while the robot is static; environmental complexities for TPerc01B are occlusions and for TPerc04 is the movement of the person being detected; and in TPerc02 and TPerc03 the robot movement's affect the performance of the person detection. The TNav01 specifies tests for the wall following component, obstacle avoidance, and lamp management; while both TDisinf test cases are system-integration level tests with the objective of testing the full disinfection workflow, combining the disinfection task with its safe operation based on the people detection component.

Obs. 4: The design of test cases that gradually introduce interactions between components helped handle system complexity. Similarly, environmental complexity can be tackled by introducing real-world inputs in a controlled manner.

TPerc and TNav evaluate the performance of the people detection and navigation components, respectively. TDisinf are mainly performance tests where they measure how efficient the system is at disinfecting. In addition to performance, these tests also serve to stress-test both software and hardware components, identifying operational limits and which components and failure modes happen due to the stress; to verify safety requirements, i.e., that no overexposure occurs; and to collect data about the reliability of the components based on the risk assessment metrics, i.e., the mean time to failure of the system.

Test Case Design. Test procedures document most of the information for the logical test cases. Table IV summarizes the test case properties identified in the test procedures, along with examples from sampled concrete test cases.

Obs. 5: Early on in the development process, focusing on collecting data about the performance of the system was useful to define pass/fail criteria based on acceptable thresholds.

Because there is a high number of possible combinations for these properties, engineers use equivalence partitioning to minimize test case explosion and target those use cases where they would gain the most (new) information. Equivalence Partitioning is a test design technique that treats inputs within "classes" as equivalent[2], i.e. test cases are grouped by how they handle valid or invalid input data. In field endurance tests, possible variations of test pre-conditions and inputs are divided into equivalent classes, taking into account information from safety risks identified during the FMEA and their own experience. The criteria for the equivalence classes of the test cases are summarized on Table V. All test cases have criteria that capture the complexity of the test based on the specification of the position and motion of the robot and any obstacles in the environment. These criteria capture not only interactions between the robot and the environment, but also between components, e.g. for the people detection component whether the robot is static, if it rotates in-place, or whether it is performing a navigation task concurrently introduces different types of failures.

Discrete variables, such as the presence of occlusions, or the standing/sitting position are straightforward to identify as partitions. Continuous variables are partitioned with information from the FMEA, component specifications (e.g. UVC lamps and sensors), and engineers' experiences. Let us use the person detection tests to illustrate how engineers identify the partitions of a continuous variable. To identify partitions in the distance between robot and person for person detection tests, engineers created a table that listed possible sensor configurations as rows and the distance to people as

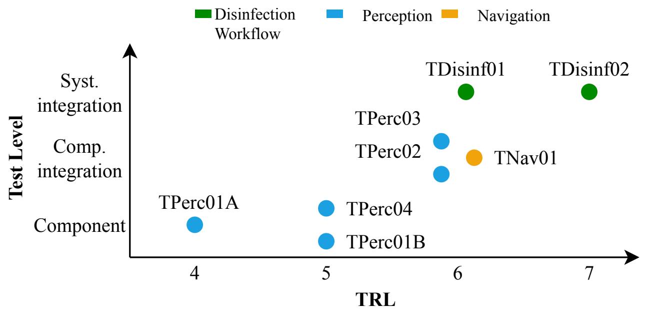


Fig. 4: Logical test cases on the TRL scale

		Detection range in m (in direction of travel)							
Object	Sensor	0	0.1	0.5	1	2	3	5	10
People	Sensor A	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	Sensor C	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥
	Sensor G	🟨	🟨	🟩	🟩	🟩	🟩	🟩	🟨
Objects with high material reflectivity	Sensor C	🟥	🟩	🟩	🟩	🟩	🟩	🟥	🟥
	Sensor D	🟥	🟥	🟥	🟨	🟨	🟨	🟥	🟥
	Sensor E	🟥	🟨	🟨	🟨	🟨	🟥	🟥	🟥

Fig. 5: Experience-based equivalence partition testing

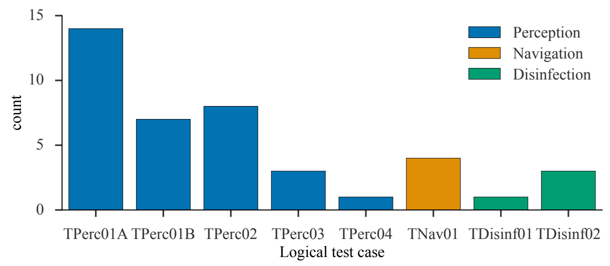


Fig. 6: Number of concrete test cases on the test procedures

columns. Fig. 5 illustrates a simplified table for this analysis with two types of objects and five safety-related sensor configurations; the original table also includes information about the number of sensors, their configuration, and where they are mounted on the robot. The increments are based on their experience and the boundaries at which the sensor could detect a person, as well as the range at which UVC light is no longer dangerous. Based on this information, cells were classified into (a) high confidence of detection or no danger (green), (b) uncertain about detection (yellow), and (c) high confidence of failure to detect people (red). Cells for (b) and their boundaries to (a) and (c) are the main focus of the test cases, since they provide information about the boundaries at which the robot could operate safely.

Obs. 6: Test design techniques that minimize test case explosion, such as equivalence partitioning, were useful to minimize environmental and system complexity.

We analyze which partitions are being tested using the TPerc test cases since they contain the most concrete test

TABLE IV: Test case elements

Concept	Definition	Examples
Pre-conditions	State of the robot and environment before tests start.	"The robot is placed at a given distance from a poster or a real person, with one camera facing the person,..."
Inputs	Data used by the robot to execute the test	Occupancy grid and/or YAML file with task specification
Actions	Actions by the testers during test execution	The motion of the person is limited to natural movements of the head and extremities.
Expected results	Usually the pass/fail criteria for a test, but can be "None" for performance tests [2]	Ground-truth positions of the person relative to the robot, false positives and false negatives of the people detection module, robot battery level, route length, etc.

TABLE V: Criteria for equivalence classes

Criteria	TPerc	TNav	TDisinf
Robot motion	✓		
Disinfection route		✓	✓
Person motion	✓		
Person distance	✓		
Person angle	✓		
Person posture	✓		✓
Person position			✓
Occlusions	✓		✓
Obstacle positions		✓	
Sensor network placement		✓	✓

cases and variations. We exclude TPerc04 which has a single test case defined; in all the other test cases, the person is usually static. Fig. 7 shows the number of test cases found on the test procedures with respect to the main criteria used for equivalent classes in the person detection test cases: posture, distance and angle of the person being detected, and presence of occlusions. Note that test cases define other variables implicitly, e.g. all TPerc experiments use natural light only, are conducted in the same room, and only involve a single person being detected. Similar observations can be made for TNav and TDisinf cases about the obstacles in the disinfection route, the position of the sensors, and the motion of the persons. Fig. 6 shows the number of concrete test cases defined in the test procedures.

Test Data Specification. Most of the inputs and preconditions are defined in the test procedures and are real-world inputs. For TNav and TDisinf tests, additional test data includes the task specification and the occupancy grid used by the robot for localization. Depending on the test cases, the complexity of the environment can be gradually increased. In addition to the static and in-place motion of the robot, the robot can execute different disinfection routes shown in Fig. 8.

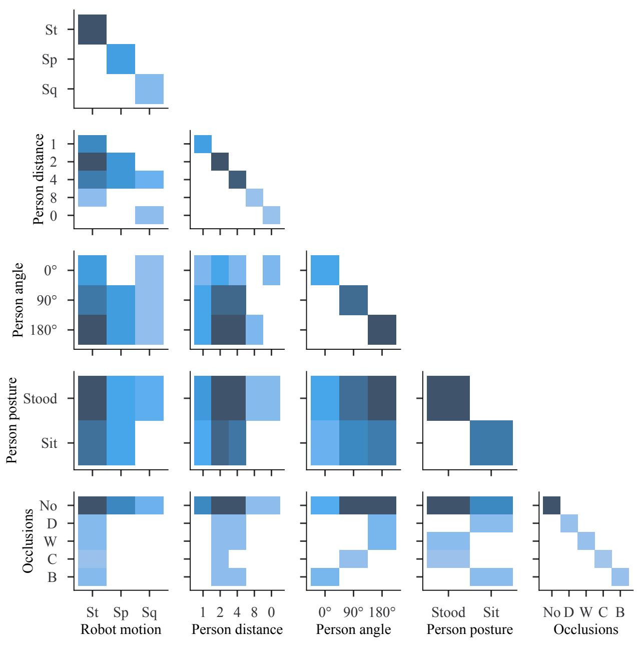


Fig. 7: Partition criteria of the person detection test cases

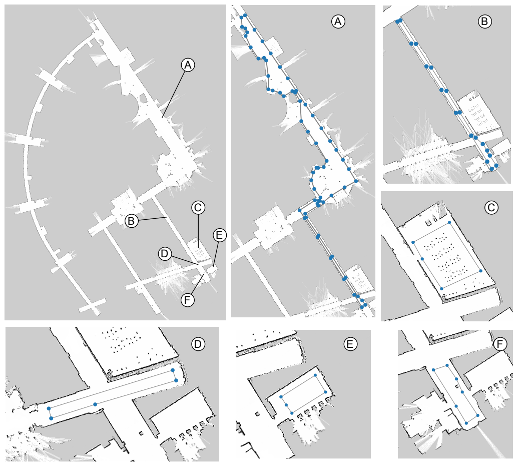


Fig. 8: Environment and tasks used for the tests

Obs. 7: Partitioning the environment was another way to handle environmental complexity for spatially-defined tasks. Characteristics of each area or room, allowed developers to deliberately control the difficulty of the system load, i.e., the task to be validated.

VI. Post-mortem analysis of test execution

We now analyze the test execution based on the resulting test artifacts. As a preliminary step, we examined the artifacts from Table II to ensure they are within our scope. We started by examining the bag files and merged those that were recorded using the `--split` flag, i.e., those that had a suffix and contiguous start and end timestamps. Similarly, we exclude 4 TRL[2] reports that do not have any runs and analyze the remaining 37 reports that cover a total of 238 runs. Figure 9 shows the test runs recorded between June 16th, 2020 and April 28th, 2021. Note that this timeline includes bags from the file list even if they are not available. Because, by default, the rosbag tool produces files using ISO 8601 as their file names, we can use them as indications of *when* tests took place even if we do not know *what* was tested. For each day, we considered the number of runs as the number of files with the lowest suffix in the time series of that day.

The tracing between test reports and log files is done manually by developers. Developers usually match bag files to test reports based on the date and start time of each test run, and the file name of the bag file, which defaults to the date as mentioned before. Once the files are copied from the robot to a developer's laptop for analysis, they are merged and renamed following a naming convention that includes the test case, date, and time of the test. We use the naming convention on the list of bag files to extract information about the testing activities on 132 of the unavailable bag files. A total of 116 bag files were mentioned by name in 6 test reports, and matched 117 test runs. Note that this means that not all reports have corresponding bag files, and a large number of bag files do not have an associated report. We categorize the remaining 35 test reports and their 119 test runs with the help of developers, and exclude the remaining 238 bag files without explicit relationships.

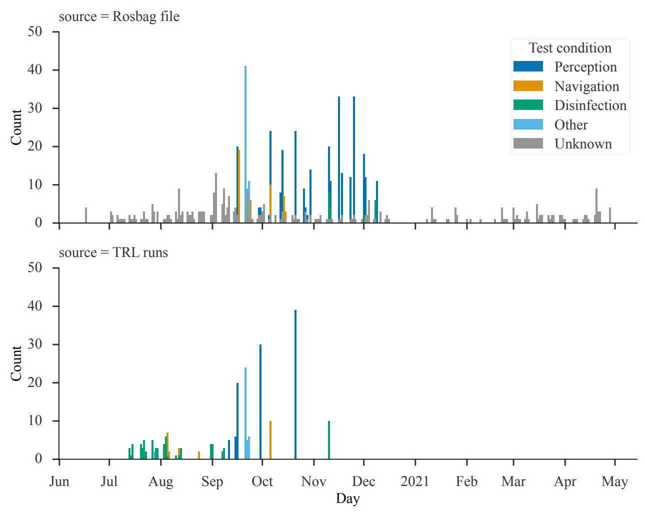


Fig. 9: Timeline of the rosbag files (top) and reported test runs (bottom) between June 16th, 2020 and April 28th, 2021

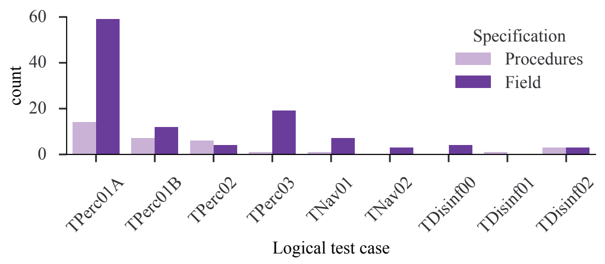


Fig. 10: Executed concrete test cases identified in the test procedures and test artifacts

Obs. 8: Traceability of requirements to test cases was paramount to determine the progress according to the test plan. Manual traceability between test cases and test artifacts was error-prone and effort intensive.

We also analyze the test execution to identify when and how often each test case was carried out. TPerc tests were conducted regularly during the campaign, and TNav was stress-tested towards the start of the campaign. We identified additional test runs for TDisinf and TNav in the TRL reports of early July and August, and examine the TDisinf reports to understand why they seemingly deviate from the pattern of increasing complexity. Upon closer inspection, these are early system integration tests defined in the field where perception and navigation components were tested in parallel. Figure 10 shows the number of executed test cases, including those defined in the field but not documented on the test procedures (including additional variations of specified test cases). Perception tests were the focus of this test campaign due to their impact on the safe operation of the robot and represent the largest proportion of the number of tests and variations to be tested. Note that there is a higher number of concrete test cases found on test artifacts than on the specifications, suggesting that developers tested more variations than those expected or documented at design time.

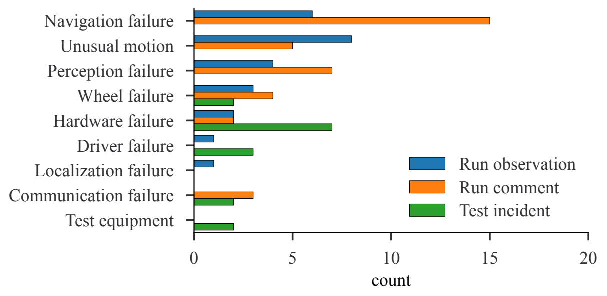


Fig. 11: Test failures reported in test incidents or as test observations in TRL^2

Obs. 9: Exploratory testing techniques were useful to define new test cases based on reactions to what happened in other tests, allowing to define test cases in the field as a response to some unexpected event. Such techniques have little or no documentation, but with good data collection, automating the documentation of additional variations should be possible.

Several types of failures were reported during the testing campaign. Fig. 11 summarizes the failures reported as test incidents and in the TRL^2 reports. A total of 16 field test incidents were recorded on their git server for tests over a span of 10 months. Test incident reports were overwhelmingly related to hardware failures, while test reports focused on the failures of software components. Out of those test incidents, 3 hardware incidents were related to 4 TRL^2 reports.

Obs. 10: On systems of systems, establishing a single pass/fail criteria might not correctly describe the outcome of the test. Interactions between components and individual pass/fail criteria need to be considered as well but are difficult to specify and effort-intensive to document.

VII. Discussion

In this section we discuss our observations from sections IV to VI. Our contributions focus on improving the empirical understanding of testing practices for service robots, but more case studies and experience reports are needed to build theories and make generalizations. Although this experience report describes context- and robot-specific test processes, it shares many commonalities with other applications and systems, and our insights into the testing practices of the Kelo AD could be of use to others. Thus, we also identify open challenges and derive research directions combining our observations and state-of-the-art practices.

The field testing approach partially tackles some testing challenges discussed in Sect. I. First, it bridges the development and deployment gap by using real-world inputs early on in the testing process, giving developers a better idea of the expected performance in the deployment environment (observations O1 and O3). Second, it manages the system complexity by designing tests to gradually increase the effects of the interactions between components by targeting test conditions at different test levels (observations O4 and O6); similarly, it handles the environmental complexity at design time by introducing real-world inputs in a controlled manner (e.g. occlusions and dynamic obstacles), and using test design techniques to minimize test case explosion (O7). Third, keeping in mind the safety-critical aspect of the system, it intentionally seeks defects caused by the variability in real-world inputs to identify corner cases that could have been missed by its risk assessment, and adds elements of stress, load, reliability, and compliance testing early in the development process, making the return on investment of the tests bigger by maximizing the information gained given the limited number of testers available (observations O1 and O2). Finally, focusing on the data collection of the robot's performance at early iterations will make the definition of pass/fail criteria based on thresholds easier in the future (O5).

Even so, some open challenges remain due to the manual nature of the tests (O8 to O10). Manual system testing is expensive and does not scale well; it is constrained by the available resources, including the time and effort for setup and execution; and tend to be brittle, particularly if lower-level components and subsystems do not have tests of their own. Even if resources were not a constraint, engineers lack the ability or resources to manufacture real-world scenarios that cause corner-cases and environment variations beyond what their test environment offers. Lastly, the large state space of the real world makes it difficult to identify how much testing is sufficient, particularly when testing safety-critical features [26].

To tackle those challenges, the design of a field testing strategy should: *(i)* Facilitate the definition of new test cases in the field, thus enabling developers to react to unexpected behaviors and effects of the SUT in the real world and support exploratory testing techniques (O9). *(ii)* Apply suitable test design techniques to handle the large state space inherent to real-world environments. *(iii)* Automate the data collection from field tests, including data provenance and traceability of requirements and test cases, particularly in cases where testing is being used to document safety assurances (observations O8 and O9). Although rosbag is a common tool mentioned in the literature for data collection and commonly used for playing back data [3], in-house tools are often required to record and manage logged data, configuration files, and run-time parameters [7], [8], [27]. This was part of the original goal for the TRL^2 tool; however, most of the data is entered manually, making the traceability between test cases and test artifacts error-prone and effort intensive. And *(iv)* reuse data from past field tests in other automated test types, similar to [28], [8].

VIII. Conclusion

We presented an experience report on testing an industrial-strength service robot, discussing the field-testing techniques and practices in-depth. The testing processes employed at low TRLs focus on exposing the robot to real-world inputs, to identify and mitigate defects, failures, and risks required for the safe operation once ready for deployment. We hope to contribute to the improvement of software engineering practices of autonomous systems by reporting detailed empirical data and experiences of a substantial case. We modeled the field testing processes, described test design and testing activities with details about artifacts, and reported challenges and solutions.

References

[1] "Robotics 2020 Multi-Annual Roadmap," Tech. Rep., 2016.

[2] G. Bath and J. McKay, *The Software Test Engineer's Handbook*. Rocky Nook, Inc., 2014.

[3] A. Afzal, C. Le Goues, M. Hilton, and C. Timperley, "A Study on Challenges of Testing Robotic Systems," in *ICST*, 2020.

[4] *Software and Systems Engineering – Software Testing – Part 11: Guidelines on the Testing of AI-based Systems*, IEEE Technical report ISO/IEC/IEEE 29 119-11:2020 (E), 2020.

[5] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*, 2012.

[6] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Case Studies," in *Experimentation in Software Engineering*, 2012.

[7] J. Biswas and M. Veloso, "The 1,000-km Challenge: Insights and Quantitative and Qualitative Results," *IEEE Intelligent Systems*, 2016.

[8] N. Hawes, C. Burbridge, F. Jovan, *et al.*, "The STRANDS Project: Long-Term Autonomy in Everyday Environments," *IEEE Robot. Autom. Mag.*, 2017.

[9] F. Duchetto, P. Baxter, and M. Hanheide, "Lindsey the Tour Guide Robot - Usage Patterns in a Museum Long-Term Deployment," in *RO-MAN*, 2019.

[10] S. Wang and H. I. Christensen, "TritonBot: First Lessons Learned from Deployment of a Long-Term Autonomy Tour Guide Robot," in *RO-MAN*, 2018.

[11] D. Bozhinoski, D. Di Ruscio, I. Malavolta, *et al.*, "Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective," *J. of Syst. and Softw. (JSS)*, 2019.

[12] S. García, D. Strüber, D. Brugali, *et al.*, "Robotics software engineering: A perspective from the service robotics domain," in *ESEC/FSE*, 2020.

[13] Q. Song, E. Engström, and P. Runeson, "Concepts in Testing of Autonomous Systems: Academic Literature and Industry Practice," in *WAIN*, 2021.

[14] M. A. S. Brito, S. R. S. Souza, and P. S. L. Souza, "Integration testing for robotic systems," *Software Quality Journal*, 2020.

[15] J. Laval, L. Fabresse, and N. Bouraqadi, "A methodology for testing mobile autonomous robots," in *IROS*, 2013.

[16] F. Amigoni, E. Bastianelli, J. Berghofer, *et al.*, "Competitions for Benchmarking: Task and Functionality Scoring Complete Performance Assessment," *IEEE Robot. Autom. Mag.*, 2015.

[17] T. Niemueller, G. Lakemeyer, S. Reuter, *et al.*, "Benchmarking of Cyber-Physical Systems in Industrial Robotics," in *Cyber-Physical Systems*, 2017.

[18] Y. Sun, J. F., M. A. Roa, and B. Calli, "Research Challenges and Progress in Robotic Grasping and Manipulation Competitions," *IEEE Robot. Autom. Lett.*, 2022.

[19] G. Bardaro, M. El-Shamouly, G. Fontana, *et al.*, "Toward model-based benchmarking of robot components," in *IROS*, 2019.

[20] G. Kanter and J. Vain, "TestIt: An Open-Source Scalable Long-Term Autonomy Testing Toolkit for ROS," in *DESSERT*, 2019.

[21] A. Babić, G. Vasiljević, and N. Mišković, "Vehicle-in-the-Loop Framework for Testing Long-Term Autonomy in a Heterogeneous Marine Robot Swarm," *IEEE Robot. Autom. Lett.*, 2020.

[22] R. Haas, D. Elsner, E. Juergens, *et al.*, "How Can Manual Testing Processes Be Optimized?" in *ESEC/FSE*, 2021.

[23] *Software and Systems Engineering –Software Testing –Part 1:Concepts and Definitions*, IEEE Std. ISO/IEC/IEEE 29 119-1:2013(E), 2013.

[24] Standard Glossary of Terms used in Software Testing. International Software Testing Qualifications Board (ISQTB). [Online]. Available: http://glossary.istqb.org/

[25] P. Morgan, A. Samaroo, G. Thompson, and P. Williams, *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide*, 4th ed. BCS Learning & Development Limited, 2019.

[26] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, 2016.

[27] S. Wang, X. Liu, J. Zhao, and H. I. Christensen, "Rorg: Service Robot Software Management with Linux Containers," in *ICRA*, 2019.

[28] T. Winters, T. Manshreck, and H. Wright, *Software Engineering at Google: Lessons Learned from Programming over Time*, 1st ed. O'Reilly Media, 2020.