RESULTS OF THE SURVEY

FAILURES IN ROBOTICS AND INTELLIGENT SYSTEMS

JOHANNES WIENKE* & SEBASTIAN WREDE*

Version: 1.0, August 24, 2017

ABSTRACT

In January 2015 we distributed an online survey about failures in robotics and intelligent systems across robotics researchers. The aim of this survey was to find out which types of failures currently exist, what their origins are, and how systems are monitored and debugged – with a special focus on performance bugs. This report summarizes the findings of the survey.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

* *Research Institute for Cognition and Robotics (CoR-Lab) & Center of Excellence Cognitive Interaction Technology (CITEC), Bielefeld University, Germany. Contact:* `{jwienke,swrede}@techfak.uni-bielefeld.de`

1 INTRODUCTION

Despite strong requirements on dependability in actual application scenarios, robotics systems are still known to be error prone with regular failures. However, not many publications exist that have systematically analyzed this situation. Therefore, we have decided to carry out a survey to get an assessment of the current situation in research robotics[1]. The aim of this survey was to collect the impressions of robotics developers on the reliability of systems, reasons for failure, and tools used to ensure successful operation and for debugging in case of failures. The survey specifically focused on software issues and software engineering aspects. Apart from general bugs, performance bugs have been specifically addressed to understand their impact on robotics systems and to determine how performance bugs differ from other bugs. A considerable amount of work in this direction has been done in other computing domains like high-performance computing or for cloud services [e.g. Gun+14; Jin+12; ZAH12]. However, in robotics such work is missing. To our knowledge, only Steinbauer [Ste13] presents a systematic study on general faults in robotics systems, but without a specific focus on performance aspects.

Our survey was implemented as an online questionnaire (following methodology advices from Gonzalez-Bañales and Adam [GA07]) which was distributed around robotics researchers using the well-known mailing lists euRobotics (`euron-dist`)[2] and `robotics-worldwide`[3] as well as more focused mailing lists. The detailed structure of the survey can be found in Appendix A. Please refer to this appendix for details on the phrasing of questions and permitted answers. Results presented in the following sections are linked to the respective questions of the survey.

In total, 61 complete submissions and 141 incomplete ones[4] were collected. 86 % of the participants were researchers or PhD candidates at universities, 7 % regular students and 7 % from an industrial context (A.12.1). On average, participants had 5.8 years of experience in robotics (sd: 3.3, A.12.2). Participants spend their active development time primarily with software architecture and integration as well as component development, despite individual differences visible in the broad range of answers (cf. Figure 1, A.12.3). Other activities like hardware or driver development are pursued only for a limited amount of time.

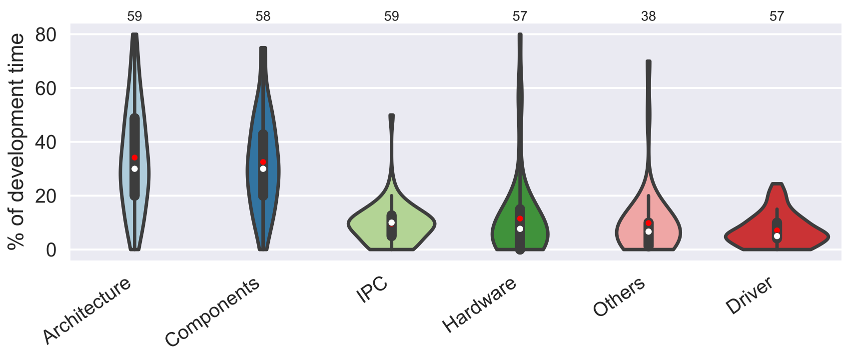

Figure 1: Development time spent by survey participants on different development aspects of robotics systems. Individual answers have been normalized to sum up to 100 %. Inside the violins, a box plot is shown with the white dot representing the median and the red dot the mean value. Numbers above the plot express the sample size, which differs as answers were optional.

1 Parts of the results have previously appeared in Wienke et al. [WMW16]
2 `https://lists.iais.fraunhofer.de/sympa/info/euron-dist`
3 `http://duerer.usc.edu/mailman/listinfo.cgi/robotics-worldwide`
4 Incomplete submissions also include visitors who only opened the welcome page and then left.

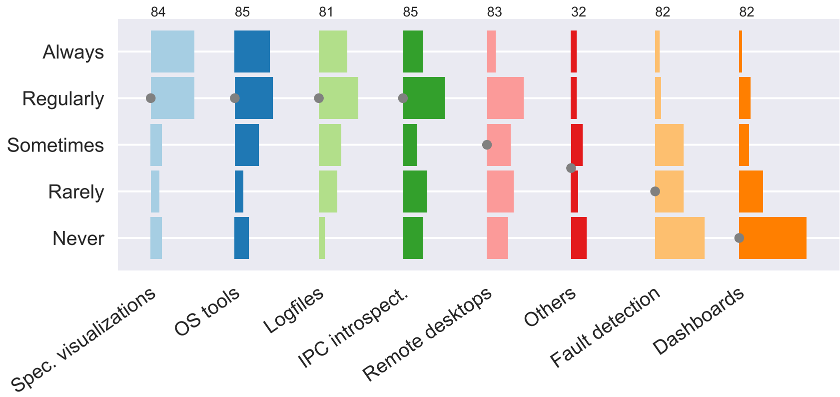

Figure 2: Usage frequency for different categories of monitoring tools. For each category, the answer counts are displayed as a histogram and the grey point marks the median value. The categories are ordered by median and – if equal – mean values. Numbers above the plot express the sample size, which might differ as answers were optional.

2 TOOL USAGE

A first set of questions tried to assess which software tools are used to monitor and debug robotics systems in general. For different types of tools, participants could rate on a 5 point scale from 0 (Never) to 4 (Always), how often the respective type of tool is used during usual development and operation of their systems. For general monitoring tools (A.2.1) the answers are depicted in Figure 2. According to the developers' opinion, special purpose visualization tools like *RViz* [RViz] or debug windows for image processing operations are most frequently used to monitor systems. These are followed by low-level operating system level tools like `ps` or `htop` and logfiles. Tools related to distributed systems like utilities of the IPC mechanism form the last category of tools that is regularly used. Remote desktop connections are used only sometimes. In contrast, autonomous fault detection methods and special dashboards for visualizing system metrics are only rarely used, despite the fact that such tools are well-established for operating large-scale systems with high dependability requirements.

A second question regarding monitoring tools asked for the exact names of tools that are used (A.2.2). The answers to this question are summarized in subsection B.1. The most frequently mentioned category of tools matched the previous question (visualization tools, most notably *RViz* [RViz]). These tools are followed by middleware-related tools, most notably the ROS command line tools and `rqt`, as well as operating system tools with `htop` and `ps` being the most frequently mentioned examples. Finally, manual log reading, remote access tools, custom mission-specific tools, and generic network monitoring tools like *Wireshark* [WiSha] are used. Additionally, one participant also explicitly mentioned hardware indicators like LEDs for this purpose.

Regarding tools used to debug robotics systems (A.3.1), participants mostly use basic methods like `printf` or log files as well as simulation (cf. Figure 3). General-purpose and readily available debuggers are less frequently used than these basic methods. Unit testing seems to be partially practiced and accepted in the robotics and intelligent systems community.

The actual tools being used have been summarized in subsection B.2 as a result of question A.3.2. Debuggers represent the most frequently mentioned category of tools with *GDB* [GDB] leading this category. Another frequently used debugging tool is *Valgrind* [Valg] for checking memory accesses. Besides `printf` debugging, other categories of used tools comprise middleware utilities, simulation and visu-

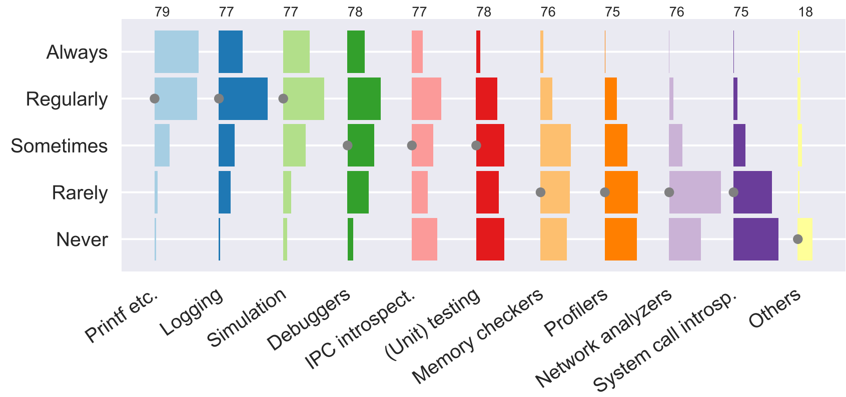

Figure 3: Usage frequencies for different categories of debugging tools and methods.

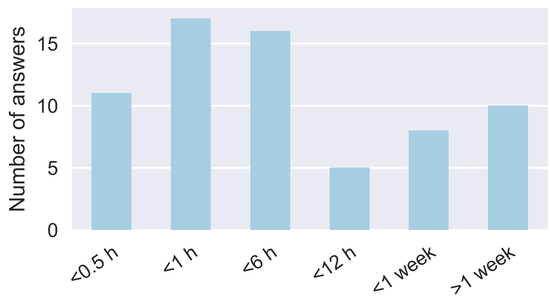

Figure 4: Participant answers for the observed MTBF in their systems.

alization (with gazebo being the most frequently mentioned software), and unit testing.

3 BUGS AND THEIR ORIGINS

In a second set of questions we have addressed the reasons for and effects of bugs in robotics systems. As actual numbers for failure rates in robotics systems are rarely available, one question asked participants for the MTBF they have observed in systems they are working with (A.4.1). As visible in Figure 4, the answers form a bimodal distribution where one part of the participants rates MTBF of their systems to be within the range of minutes to a few hours, whereas others indicate MTBF rates in the range of days to weeks. One can think of multiple explanations for these diverging replies:

- The systems participants have been working with are different in nature and some are closer to production systems.
- Answers with higher MTBF include the system's idle time in the calculation, despite an explicit indication in the explanation of the question that the *operation* time is the basis for this number.
- Differences can be explained by the way people use debugging or monitoring tools in their systems. However, no significant relations could be found in the results.

As for the first two hypotheses no data is available to validate them and the third one cannot be proven using the survey results, the effective reasons for the bimodal distribution are unknown.

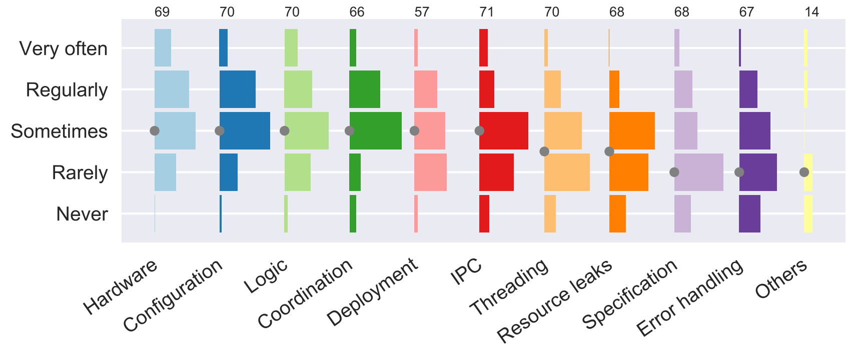

Figure 5: Rating of different bug categories being the reason for system failures.

To generally understand why systems fail, participants were asked to rate how often different bug categories were the root cause of system failures (A.4.2). The categories have been selected based on related survey work from robotics and other domains [Ste13; Gun+14; Jin+12; McC04]. Figure 5 displays the results for this question. Hardware bugs represent the most frequent category followed by a set of categories representing high-level issues (configuration, coordination, deployment, IPC) as well as general logic bugs. Most of the high-level issues seem to be technical problems and not specification problems because specification issues only rarely cause failure (median).

Apart from the aforementioned categories, participants could provide further causes in text form (A.4.3). After removing items that relate to categories already presented in the previous question, answers can be summarized as a) environment complexity/changes (8 mentions) b) low-level driver and operating system failures (3 mentions) c) hardware configuration management (1 mention) and d) hardware limitations (1 mention). subsection B.3 shows the answers in detail as well as how categories have been assigned. In the survey, we explicitly excluded the environment as an origin of system failures because it does not represent a real defect in any component of the system. However, the results still show how important the discrepancy between intended usage scenarios and capabilities of systems in their real application areas is in robotics and intelligent systems.

4 PERFORMANCE BUGS

In order to understand performance bugs in robotics and intelligent systems, a dedicated set of questions was added to the survey. First, participants were asked for the percentage of bugs that affected resource utilization (A.5.1). On average, 24 % (sd 17 %) of all bugs affected system resources. Participants also had to rate how frequently different system resources were affected by performance bugs (A.6.1). These results are visualized in Figure 6. Memory, CPU and network bandwidth are the most frequently affected resources. Network bandwidth can be explained by the distributed nature of many of the current robotics systems. These three primarily affected resources are followed by disk space. Countable resources like processes or network connections are only rarely affected. A question for further affected resources (A.6.2) yielded IPC-related virtual resources like event queues and IO bandwidth in addition to the previous categories (cf. subsection B.4).

To get an impression of common causes for performance issues in robotics and intelligent systems, a question asked participants to rate how frequently different categories of root causes were the origin of performance bugs in their systems (A.7.1). The categories are the ones of the previous general questions on bug origins (A.7.1) extended with two items specifically targeting performance bugs: skippable com-

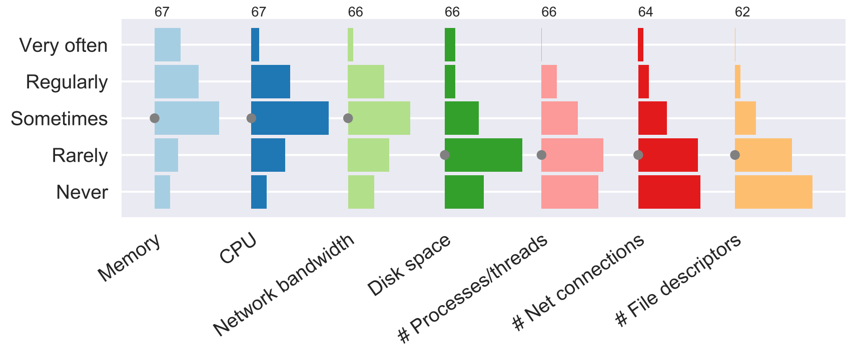

Figure 6: Frequency of bug effects on system resources.

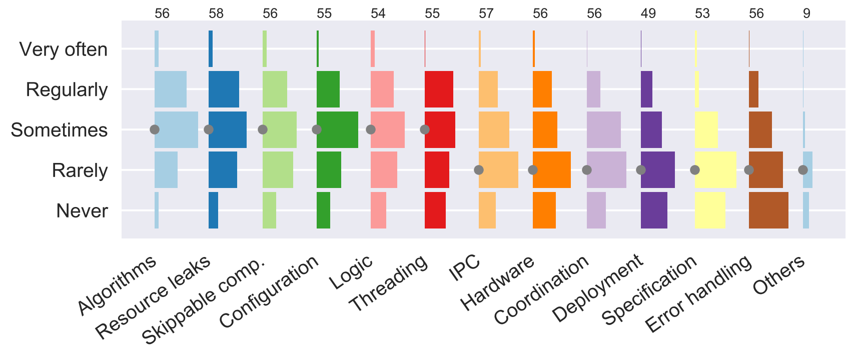

Figure 7: Frequency of reasons for performance bugs.

putation, i.e. unnecessary computation that does not affect the functional outcomes (based on the results in Gunawi et al. [Gun+14]) and algorithmic choices. Figure 7 depicts the results for this question. The most frequent reason for performance bugs is the choice of inappropriate algorithms followed by resource leaks and unnecessary computations. Interestingly, configuration issues are also among the frequent causes for performance bugs. When comparing answers to this questions with the answers for origins of general bugs (A.4.2), most categories are less likely origins for performance bugs than for general bugs apart from resource leaks (cf. Table 1). Interestingly, threading issues do not significantly affect performance bugs differently than general bugs.

5 BUG EXAMPLES

Finally, participants were asked to provide detailed descriptions of bugs they had observed in their systems. Two questions in this direction were asked with four sub-answers explicitly requesting a) the visible effects on the system, b) the underlying defect causing the bug, c) the steps performed to debug the problem, and d) the affected system resources. These questions were added to the survey to get an impression of the actual problems current robotics developers are facing with their systems and how they are addressed.

The first of these questions asked for a description of any type of bug that participants remembered from their systems that is particularly representative for the kind of bugs frequently observed (A.9). In total, 21 answers were submitted for this question with a complete listing of the answers available in subsection B.5. Most notably, 10 of the answers (48 %) were related to basic programming issues like seg-

CATEGORY	CHANGE
Communication	−0.28
Configuration	−0.51**
Coordination	−0.79****
Deployment	−0.71****
Error handling	−0.44*
Hardware	−0.98****
Logic	−0.44*
Resource leaks	0.47**
Specification	−0.46*
Threading	0.04
Others	−0.58

Table 1: Changes to the mean ratings for different categories being the origins of failures when comparing performance bugs to general bugs. A change of 1 would indicate a shift from one answer category to the next higher one. Significances have been computed using a Mann-Whitney U test.

mentation faults or memory leaks, for instance caused by C/C++ peculiarities. 8 answers (38 %) described an issue that can be classified as a performance bug. Issues related to the IPC usage or infrastructure were mentioned by 4 answers (19 %). Also, 4 answers indicated bugs related to the coordination of the system (for instance, loops in the controlling state machines) of which 2 answers were related to unexpected environment situations. Additionally, 2 answers were related to timing aspects and another 2 answers indicated that a bug was never or only accidentally understood and solved. Please refer to the tagging in subsection B.5 for details.

A second question asked participants to describe the most interesting bug they could remember in the same format. This was done to get an impression of which extreme types of bugs are possible in robotics systems. 14 participants answered this question and their answers are listed in subsection B.6. In line with the previous question, programming problems related to low-level issues also represent the most frequently mentioned type of bugs with 6 answers (43 %). Furthermore, 3 answers (21 %) described bugs caused by driver or operating system problems.

Answers to both questions indicate that memory-management-related programming issues are often debugged using established tools like *GDB* [GDB] or *Valgrind* [Valg] – however – with varying success. One answer specifically mentioned that these tools are often not helpful for distributed systems.

6 RESULT INTERPRETATION

The presented survey results show that there is still a great potential for improvements in the dependability of robotics systems. With MTBF rates in the range of hours, a major part of the surveyed systems is far from being reliable enough for longer-term operations and work in this direction is needed, even if the majority of developers reached with this survey is working on research systems, which rarely end up in production use cases. Nevertheless, an appropriate level of dependability is required also in this domain to allow efficient experimentation and reliable studies. Still, monitoring tools that are specifically geared towards operating systems with high availability and reliability like fault detection or dashboards for a quick manual inspection of systems states are only rarely applied in robotics. The survey does not provide answers why this is the situation. Reasons could include the overhead of deploying such approaches which might not be feasible in smaller, relatively short-lived systems, or the lack of knowledge about such approaches, especially as many robotics researchers do not have a strong background in maintaining large-scale systems. Therefore, improving approaches and making them more easily usable is one promising direction to foster their application.

With respect to system failures and their origins, the quantitative results from this survey indicate that hardware faults are among the most frequent causes for failure. This contradicts the findings from Steinbauer [Ste13], which might potentially be caused by the wider range of applications covered in this survey. Generally, system failures seem to originate more frequently from bugs occurring in high levels of abstraction like coordination, deployment or configuration and less often from component-internal issues like crashes. Still, a majority of the requested bug descriptions for representative bugs actually dealt with such component-internal issues. One reason for this might be that, while frequently being observed, such component-related issues are often noticed immediately and therefore are perceived as part of the development work and not as system failures. In any case, these issues are strikingly often caused by basic programming issues, often related to the manual memory management and syntax idiosyncrasies of C/C++. A general shift in robotics towards less error-prone languages with automatic memory management, clearer syntax and better debugging abilities has the potential to avoid a major amount of bugs currently found during development and operations.

With respect to the performance aspects, one quarter of the bugs found in current systems can be classified as performance bugs. In the descriptions of representative bugs even more than one third of the answers was performance-related. Therefore, specifically addressing such issues is not only a niche but instead provides the potential to avoid a major amount of failures in the future. The survey has indicated that performance bugs are significantly less often caused by high-level aspects like coordination or deployment and also by hardware issues. Therefore, addressing them on a component-level should already result in reasonable improvements.

Generally, systems are often debugged using log files and `printf` instructions specifically placed for debugging. Participants have indicated that debuggers and memory checkers like *Valgrind* [Valg] are used less frequently. This is probably caused by the fact that such tools cannot be used for all problem kinds. The detailed bug reports still show that these tools are frequently used to debug programming issues on the component level. Participants have also indicated that these tools cannot be easily used for problems related to the distributed systems nature of current robots. Further work on debugging infrastructure respecting this fact might improve the situation. Finally, simulation seems to be an important tool for debugging robotics systems and explicit support for simulation-based testing and debugging might provide one future avenue for more dependable robotics systems.

7 THREATS TO VALIDITY

The survey results represent the opinions and memorized impressions of interviewed developers, not objective measurements of the real effects. As such, results may be biased. However, general tendencies derived from the results should still be valid as a complete misassessment is unlikely across all participants.

Due to the distribution of the survey via primarily research-related mailing lists, results are only representative for systems developed in this context and cannot be generalized towards industrial, production-ready systems.

The categories used in questions regarding the frequencies of bug origins may have partially been hard to distinguish from each other. Therefore, in some cases, ratings might be blurred between multiple categories due to the imprecise definitions. When possible, the conclusions drawn from the survey have been based on a grouping of multiple categories to mitigate this effect.

A QUESTIONNAIRE STRUCTURE

The following sections represent the structure of the online survey. This is a direct export of the survey structure without modifications.

A.1 Introduction

Thank you very much for taking the time to participate in this survey. This survey is part of my PhD project with a focus on exploiting knowledge about computational resource consumption in robotics and intelligent systems, persued at Bielefeld University. Therefore, in order to participate, you should be involved or have been involved in the development or maintenance of such systems. In case you have worked or are working with mutiple systems in parallel, please provide answers on the combination of all theses systems.

Participating in this survey should not take longer than 15 minutes. The survey consists of several questions and you are free to skip questions in case you do not want to answer them. Moreover, you can go back and forth between the questions you have already answered in order to revise them. All data you enter in this survey will be anonymized.

Johannes Wienke
jwienke [at] techfak.uni-bielefeld.de

A.2 Monitoring Tools

The first part of this survey addresses how robotics and intelligent systems are monitored at runtime in order to assess their health and understand the ongoing operations. Monitoring includes the ongoing collection of runtime data, the observation of operations as well as the assessment of system health.

A.2.1 *How often do you use the following kinds of tools to monitor the operation of running systems?*

Rate individually for:

- Operating system command line tools e.g. htop, iotop, ps (OS)
- Logfiles (LOG)
- Dashboard views e.g. munin, graphite, nagios (DASH)
- Inter-process communication introspection e.g. middleware logger (IPC)
- Autonomous fault or anomaly detectors (FD)
- Special-purpose visualizations e.g. rviz, image processing debug windows (VIS)
- Remote desktop connections e.g. VNC, rdesktop (RDP)
- Others (OTH)

ANSWER TYPE Fixed choice

- Never (0)
- Rarely (1)
- Sometimes (2)
- Regularly (3)
- Always (4)

A.2.2 *Please name the concrete tools that you use for monitoring running systems.*

Separate different tools with a comma.

ANSWER TYPE longtext (length: 40)

A.3 Debugging Tools

This part of the survey addresses tools that are used in order to debug systems in case a failure has been detected. Debugging is the process of identifying the root cause of an observed abnormal system behavior.

A.3.1 *How often do you use the following tools for debugging?*

Rate individually for:

- Console output e.g. printf, cout (PRNT)
- Logfiles (LOG)
- Debuggers e.g. gdb, pdb (DBG)
- Profilers e.g. kcachegrind, callgrind (PROF)
- Memory checkers e.g. valgrind (MEMC)
- System call introspection e.g. strace, systemtap (SYSC)
- Inter-process communication introspection e.g. middleware logger (IPC)
- Network analyzers e.g. wireshark (NWAN)
- Automated testing e.g. unit tests (TEST)
- Simulation (SIM)
- Others (OTH)

ANSWER TYPE Fixed choice

- Never (0)
- Rarely (1)
- Sometimes (2)
- Regularly (3)
- Always (4)

A.3.2 *Please name the concrete tools that you use for debugging.*

Separate different tools with a comma.

ANSWER TYPE longtext (length: 40)

A.4 General Failure Assessment

Please provide information about failures you have observed in the systems you are working with.

A.4.1 *Averaging over the systems you have been working with, what to do you think is the mean time between failures for these systems?*

The mean time between failures is the average amount of operation time of a system until a failure occurs.

ANSWER TYPE Fixed choice

- < 0.5 hours (0)
- < 1 hour (1)
- < 6 hours (2)
- < 12 hours (3)
- < 1 week (4)
- > 1 week (5)

A.4.2 *Please indicate how often the following items were the root cause for system failures that you know about.*

Rate individually for:

- Hardware issues (HW)
- System coordination e.g. state machine (COORD)
- Deployment (DEPL)
- Configuration errors e.g. component configuration (CONF)
- Logic errors (LOGIC)
- Threading and synchronization (THRD)
- Wrong error handling code (ERR)
- Resource leaks or starvation e.g. RAM full, CPU overloaded (LEAK)
- Inter-process communication failures e.g. dropped connection, protocol error (COMM)
- Specification error / mismatch e.g. component receives other inputs than specified (SPEC)
- Others (OTH)

ANSWER TYPE Fixed choice

- Never (0)
- Rarely (1)
- Sometimes (2)
- Regularly (3)
- Very often (4)

A.4.3 *Which other classes of root causes for failures did you observe?*

Separate items by comma.

ANSWER TYPE text (length: 24)

A.5 Resource-Related Bugs

The following questions deal with the consumption of computational resources like CPU, memory, disk, network etc.

A.5.1 *How many of the bugs you have observed or know about had an impact on computational resources, e.g. by consuming more or less of these resources as expected?*

Please approximate the amount with a percentage value of the total number of bugs you can remember. A quick guess is ok here.

ANSWER TYPE integer (length: 10)

A.6 Impact on Computational Resources

The following questions deal with the consumption of computational resources like CPU, memory, disk, network etc.

A.6.1 *Please indicate how often the following computational resources were affected by bugs you have observed.*

A computational resource was affected by a bug in case its consumption was higher or less than expected, e.g. in comparable or non-faulty situations.

Rate individually for:

- CPU (CPU)
- Working memory (MEM)
- Hard disc space (HDD)
- Network bandwidth (NET)
- Number of network connections (CON)
- Number of processes and threads (PROC)
- Number of file descriptors (DESC)

ANSWER TYPE Fixed choice

- Never (0)
- Rarely (1)
- Sometimes (2)
- Regularly (3)
- Very often (4)

A.6.2 *If there are other computational resources that have been affected by bugs, please name these.*

ANSWER TYPE longtext (length: 40)

A.7 Performance Bugs

The following question specifically addresses performance bugs. A system failure or bug is a performance bug in case it is visible either through degradation in the observed performance of the system (e.g. delayed or very slow reactions) or through an unexpected consumption of computational resources like CPU, memory, disk, network etc.

A.7.1 *Please rate how often the following items were the root causes for performance bugs you have observed.*

Rate individually for:

- Hardware issues (HW)
- System coordination e.g. state machine (COORD)
- Deployment (DEPL)
- Configuration errors e.g. component configuration (CONF)
- Logic errors (LOGIC)
- Threading and synchronization (THRD)
- Wrong error handling code (ERR)
- Unnecessary or skippable computation (SKIP)
- Resource leaks or starvation e.g. RAM full, CPU overloaded (LEAK)
- Inter-process communication failures e.g. dropped connection, protocol error (COMM)
- Specification error / mismatch (SPEC)
- Algorithm choice (ALGO)
- Others (OTH)

ANSWER TYPE Fixed choice

- Never (0)
- Rarely (1)
- Sometimes (2)
- Regularly (3)
- Always (4)

A.8 Case Studies

For the following questions, please provide descriptions of any kind of bug that you remember.

A.8.1 *Thinking about the systems you have worked with so far, is there a bug that you remember which happened several times or which is representative for a class of comparable bugs?*

ANSWER TYPE Fixed choice

- Yes (Y)
- No (N)

A.9 Case Study: Representative Bug

Please briefly describe the representative bug that you remember.

A.9.1 *How was the representative bug noticed?*

Please explain the observations that were made and how they diverged from the expectations.

ANSWER TYPE longtext (length: 40)

A.9.2 *What was the root cause for the bug?*

Please explain which component(s) of the system failed and in which way.

ANSWER TYPE longtext (length: 40)

A.9.3 *Which steps were necessary to analyze and debug the problem?*

Please include the information sources that had to be observed and the tools that got applied.

ANSWER TYPE longtext (length: 40)

A.9.4 *Which computational resources were affected by the bug?*

Computational resources include CPU, working memory, hard disc space, network bandwidth & connections, number of processes and threads, nubmer of file descriptors etc.

ANSWER TYPE longtext (length: 40)

A.10 Case Studies

For the following questions, please describe any kind of bug that you remember.

A.10.1 *Thinking about the systems you have worked with so far, is there a bug that you remember which was particularly interesting for you?*

ANSWER TYPE Fixed choice

- Yes (Y)
- No (N)

A.11 Case Study: Interesting Bug

Please describe briefly the most interesting bug that you remember from one of the systems you have been working with.

A.11.1 *How was the interesting bug noticed?*

Please explain the observations that were made and how they diverged from the expectations.

ANSWER TYPE longtext (length: 40)

A.11.2 *What was the root cause for the bug?*

Please explain which component(s) of the system failed and in which way.

ANSWER TYPE longtext (length: 40)

A.11.3 *Which steps were necessary to analyze and debug the problem?*

Please include the information sources that had to be observed and the tools that got applied.

ANSWER TYPE longtext (length: 40)

A.11.4 *Which computational resources were affected by the bug?*

Computational resources include CPU, working memory, hard disc space, network bandwidth & connections, number of processes and threads, nubmer of file descriptors etc.

ANSWER TYPE longtext (length: 40)

A.12 Personal Information

As a final step, please provide some information about your experience with robotics and intelligent systems development.

A.12.1 *In which context do you develop robotics or intelligent systems?*

ANSWER TYPE Fixed choice

- Student (excluding PhD students) (STUD)
- Researcher at a university (PhD students, scientific staff) (RES)
- Industry (IND)
- Other (OTHER)

A.12.2 *How many years of experience in robotics and intelligent systems development do you have?*

ANSWER TYPE integer (length: 10)

A.12.3 *How much of your time do you spend on developing in the following domains?*

Please indicate in percent of total development time. Numbers may not sum up to 100.

Rate individually for:

- Hardware (HW)
- Drivers (DRV)
- Functional components (COMP)
- Inter-process communication infrastructure (COMM)
- Software architecture and integration (ARCH)
- Other (ANY)

ANSWER TYPE integer (length: 3) Hint: Percent of development time

A.13 Final remarks

Thank you very much for participating in this survey and thereby supporting my research.

In case you have further questions regarding this survey or the research topic in general, please contact me via email.

Johannes Wienke
jwienke [at] techfak.uni-bielefeld.de

B RESULT DETAILS

B.1 Used monitoring tools

The following table presents the results for question A.2.2. The free text answers have been been grouped into categories (caption lines in the table). For each answer that included at least one item belonging to a category, the counter of each category was incremented. Hence, the counts represent the number of answers that mentioned a category at least once. Additionally, for each category, representative entries have been counted the same way. Some of the answers include uncommon or special-purpose tools or techniques. These have not been counted individually and, hence, are only visible in the category counts.

TOOL	ANSWER COUNT	
VISUALIZATION	27	
rviz		22
gnuplot		2
matplotlib		1
MIDDLEWARE TOOLS[5]	23	
ROS command line		14
rqt		5
RSB		4
BASIC OS TOOLS	22	
htop		12
ps		7
top		7
acpi		1
du		1
free		1
lsof		1
procman (gnome)		1
pstree		1
screen		1
tmux		1
MANUAL LOG READING	13	
REMOTE ACCESS	9	
ssh		5
putty		1
rdesktop		1
vnc		1
CUSTOM MISSION-SPECIFIC	4	
GENERIC NETWORK	2	
netstat		1
tcpdump		1
wireshark		1
HARDWARE SIGNALS	1	

B.2 Used debugging tools

The following table presents the results for question A.3.2. The free text answers have been been grouped into categories (highlighted lines in the table). For each

5 Represents entries that are specific to the middleware-related aspects of an ecosystem. For instance, ROS_DEBUG has not been counted here. Instead, this belongs to the "Manual log reading" category.

answer that included at least one item belonging to a category, the counter of each category was incremented. Hence, the counts represent the number of answers that mentioned a category at least once. Additionally, for each category, representative entries have been counted the same way. Some of the answers include uncommon or special-purpose tools or techniques. These have not been counted individually and, hence, are only visible in the category counts.

TOOL	ANSWER COUNT	
DEBUGGERS	19	
gdb		17
pdb		3
VS debugger		2
ddd		1
jdb		1
RUNTIME INTROPSECTION	13	
valgrind		12
callgrind		2
kcachegrind		1
strace		1
GENERIC	15	
printf, cout, etc.		14
logfiles		4
git		1
MIDDLEWARE TOOLS[6]	12	
ROS command line		5
RQT		2
RSB		2
SIMULATION & VISUALIZATION	7	
gazebo		4
rviz		1
Vortex		1
stage		1
FUNCTIONAL TESTING	6	
gtest		2
junit		2
cppunit		1
rostest		1
IDES	4	
Qt Creator		2
KDevelop		1
LabVIEW		1
Matlab		1
Visual Studio		1
GENERIC NETWORK	2	
wireshark		2
tcpdump		1
DYNAMIC ANALYSIS	1	
Daikon		1

6 Represents entries that are specific to the middleware-related aspects of an ecosystem.

B.3 Summarization of free form bug origins

The following table presents all answers to question A.4.3. Individual answers have been split into distinct aspects. These aspects have either been assigned to an existing answer category from question A.4.2 or to new categories.

ANSWER	CATEGORY	
	EXISTING	NEW
unknown driver init problems (start a driver, and works only after second trial)		Driver & OS
environment noise (lighting condition variation, sound condition in speach recognition) hard to adapt to every possible variation		Environment
Insufficient Component Specifications	Specification	
Changed maps/environments		Environment
lossy WiFi connections	Hardware	
unreliable hardware	Hardware	
in Field robotics, the environment is the first enemy. . .		Environment
Environment changes		Environment
sensor failures	Hardware	
unprofessional users		Environment
Operation System / Firmware failure		Driver & OS
network too slow	Hardware	
Loose wires	Hardware	
other researchers changing the robot configuration		Config mgmt
coding bugs	Logic	
algorithm limitations		Environment
sensor limitations		Hardware lim
perception limitations		Environment
wrong usage		Environment
Failures in RT OS timing guarantees		Driver & OS

B.4 Summarization of other resources affected by bugs

The following table presents the free text results of question A.6.2. Answers have been split into distinct aspects and these aspects have either been assigned to one of the existing categories from question A.6.1 or – if these did not match – new categories have been created to capture the answers. Parts of answers that did not represent system resources which have a resource capacity that can be utilized have been ignored. These are marked as strikethrough text.

ANSWER	RESOURCE	
	EXISTING	NEW
USB bandwidth and or stability		IO bandwidth
locks on files/devices/resources ~~permissions~~ ~~file system integrity~~	File descriptors	
interprocess communication queues, e.g. queue overflow		IPC
Files (devices) left open. ~~Wrong operation in GPU leads to restart.~~	File descriptors	
Memory leak – not sure why or where	Memory	

B.5 Representative bugs

The following subsections present answers to the questions for representative bugs (A.9). For the analysis, answers have been tagged for various aspects and types of bugs being mentioned in them. Raw submission texts have been reformatted to match the document and typographical and grammatical errors have been corrected.

B.5.1 *Representativ bug 8*

OBSERVATION computer system unresponsive

CAUSE memory leak

DEBUGGING

- find faulty process
- analyze memory usage (`valgrind/gdb`)
- repair code

AFFECTED RESOURCES main memory

TAGS basic programming issue; performance bug

B.5.2 *Representativ bug 10*

OBSERVATION System got stuck in infinite loop.

CAUSE Unexpected infinite loop in the behaviour (state machine). Noise in the data caused the system to infinitely switch between two states.

DEBUGGING

1. Detection of which states were affected.
2. Detection of the responsible subsystem(s).
3. Detection of the responsible functions.
4. Recording data that caused the problem.
5. Analyzing the data and searching for unexpected situations.
6. Modification of the system in order to handle such situation correctly.

AFFECTED RESOURCES CPU

TAGS coordination; environment-related

B.5.3 *Representativ bug 14*

OBSERVATION high latency in spread communication

CAUSE wrong spread configuration/wrong deployment of components

DEBUGGING trial & error: reconfiguration, stopping and starting components, monitoring of latency via `rsb-tools`

AFFECTED RESOURCES network-latency

TAGS communication; performance bug

B.5.4 *Representativ bug 21*

OBSERVATION Incorrect response of the overall system according to requested task request. System thinks it did not grasp an object although it did and restarts grasping operation or cancels the task due to the missing object in hand.

CAUSE State machine design and/or logic error and/or untriggered event due to sensor not triggering as expected (hardware) or too much noise (environment noise). The root cause is often a case not being handled correctly in a big system with a lot of sensors and possible case.

DEBUGGING event logger analysis over XCF XML data, unit test of single sensor output to see noise level or false positives.

AFFECTED RESOURCES Hardware (noise in the sensor)

TAGS coordination; environment-related

B.5.5 *Representativ bug 26*

OBSERVATION Segfault

CAUSE Segfault

DEBUGGING `gdb`

AFFECTED RESOURCES

TAGS basic programming issue

B.5.6 *Representativ bug 30*

OBSERVATION Unexpected overall behavior.

CAUSE Wrong logic in the abstract level.

DEBUGGING Run simulation in the abstract layer.

AFFECTED RESOURCES None.

TAGS coordination

B.5.7 *Representativ bug 41*

OBSERVATION Failure to observe expected high-level output. More specifically, a map that was being built was lacking data.

CAUSE Congested wireless network connection. The amount of data could not be transmitted within the expected time frame.

DEBUGGING Logging of signals between modules on the deployed system to verify data was being produced and transmitted correctly, and logging of data received.

AFFECTED RESOURCES Network connection

TAGS communication; timing

B.5.8 *Representativ bug 42*

OBSERVATION Because of timing mismatch the planning system was working with outdated data.

CAUSE Non-event based data transfer.

DEBUGGING Going through multiple log files in parallel to find the data that was transmitted in comparison to the data that was used in the computation.

AFFECTED RESOURCES Non. Mostly mismatch between specification and performed actions.

TAGS coordination; timing

B.5.9 *Representativ bug 46*

OBSERVATION Navigation did not work correctly

CAUSE Algorithmic errors

DEBUGGING Dig in and verify steps in the algorithm

AFFECTED RESOURCES

TAGS

B.5.10 *Representativ bug 60*

OBSERVATION delays in robots command execution

CAUSE supervision and management part of the framework

DEBUGGING benchmarking, profiling

AFFECTED RESOURCES

TAGS performance bug

B.5.11 *Representativ bug 69*

OBSERVATION memory leak

CAUSE resource management, dangling pointers

DEBUGGING check, object/resource timeline, usually start with resources that are created often and handed over regularly and therefore might have unclear ownership

AFFECTED RESOURCES memory, CPU

TAGS basic programming issue; performance bug

B.5.12 *Representativ bug 70*

OBSERVATION constantly increasing memory consumption

CAUSE Memory leaks

DEBUGGING Running the code in offline mode with externally provided inputs and observing the memory consumption pattern. Tools like `valgrind` or system process monitor helps to discover the problem

AFFECTED RESOURCES Working memory

TAGS basic programming issue; performance bug

B.5.13 *Representativ bug 76*

OBSERVATION Visually in system operation. In one case, elements within a graphical display were misdrawn. In another, command codes were misinterpreted, resulting in incorrect system operation.

CAUSE Variable type mismatch e.g. integer vs. unsigned integer – such as when a number intended to be a signed integer is interpreted as an unsigned integer by another subsystem.

DEBUGGING Debugger using single step and memory access.

AFFECTED RESOURCES None

TAGS basic programming issue; performance bug

B.5.14 *Representativ bug 81*

OBSERVATION segfault

CAUSE C++ pointers

DEBUGGING `gdb`, `valgrind`

AFFECTED RESOURCES none

TAGS basic programming issue

B.5.15 *Representativ bug 96*

OBSERVATION segmentation fault

CAUSE logical errors, bad memory management

DEBUGGING using debuggers, looking and studying code

AFFECTED RESOURCES working memory, number of process and threads

TAGS basic programming issue

B.5.16 *Representativ bug 128*

OBSERVATION Robot software is not working / partially working (e.g. recognizing and grasping an object)

CAUSE Wrong configuration and/or API changes that hasn't been changes in all components (Problem with scripting languages like python)

DEBUGGING

- identify error message and component via log files / console output
- Think about what could have caused the problem (look into source code, `git`/`svn` commit messages/diffs)
- try to fix it directly or talk with other developers in case of bigger changes / out of my responsibility

AFFECTED RESOURCES none

TAGS

B.5.17 *Representativ bug 135*

OBSERVATION middleware communication stopped / was only available within small subsets of components

CAUSE unknown

DEBUGGING

AFFECTED RESOURCES

TAGS not/accidentally solved; communication

B.5.18 *Representativ bug 136*

OBSERVATION

1. Application/process hang.
2. 100 % core usage on idle
3. Unbalanced load between cores (Monolithic code).

CAUSE

1. Loose wire/couple (mostly USB)
2. Active wait
   ```
   while(1)  while(!flag); process(); flag = 0;
   ```
3. A bad design. No threads were used, but time measurements to switch between tasks.

DEBUGGING

1. Check everything, realize that the file-device is open but device is no longer present or has different pointer or has reseted

2/3. Check every code file. People use to make old-style structured programming when using C/C++

when you notice the performance go brick, check CPU/memory usage with OS tools and notice one process is using everything but is idle.

AFFECTED RESOURCES Mostly CPU

TAGS basic programming issue; performance bug

B.5.19 *Representativ bug 156*

OBSERVATION Difficult to reproduce, random segmentation faults

CAUSE 90 % of the time it has been either accessing unallocated memory (off-by-one errors) or threading issues

DEBUGGING When working with a system with many processes, threads, inter-process communications, etc., the standard tools (`gdb`, `valgrind`) are often not that helpful. If they can't immediately point me to the error, I'll often resort to print statement debugging.

AFFECTED RESOURCES Memory leaks, CPU usage

TAGS basic programming issue

B.5.20 *Representativ bug 190*

OBSERVATION unforeseen system behavior, decreased system performance

CAUSE misconfiguration of middleware

DEBUGGING

- monitoring middleware configuration of concerned components
- checking log-files
- sometimes debug print-outs

AFFECTED RESOURCES CPU, network load

TAGS communication; performance bug

B.5.21 *Representativ bug 191*

OBSERVATION Software controlling the robot crashed immediately after started in robot or robot stop to move when has to perform certain operation

CAUSE The error was caused by not checking range of allocated memory in some object's constructor, we used `sprintf` instead of `snprintf`

DEBUGGING

- `gdb` – did not find anything
- `valgrind` – did not find anything

Both tools were run on PC, where the error did not occur, but we did not use them on the robot's pc. The bug was found accidentally.

AFFECTED RESOURCES access to non-allocated memory lead immediately to crash of program.

TAGS basic programming issue; not/accidentally solved

B.6 Interesting bugs

The following subsections present answers to the questions for intersting bugs (A.11). Answers have been processed the same way as for subsection B.5.

B.6.1 *Interesting bug 5*

OBSERVATION There are too many to remember. A recent one got noticed by surprisingly high latency in a multithreaded processing and visualization pipeline.

CAUSE Sync to vblank was enabled on a system and due to a possible bug in Qt multiple GL widgets contributed to the update frequency. The maximum display update frequency dropped below 30 Hz.

DEBUGGING Compare systems and analyze timing inside the application. Google the problem.

AFFECTED RESOURCES None

TAGS driver & OS

B.6.2 *Interesting bug 21*

OBSERVATION On an arm and hand system, with hand and arm running on separate computers linked via an Ethernet bus, timestamped data got desynchronized. This was noticed on the internal proprioception when fingers moved on the display and the arm did not although both moved in physical world.

CAUSE NTP not setup correctly. University had a specific NTP setting requirement that was not set on some computers. Could actually never synchronize.

DEBUGGING Looking at timestamps in the messages over `rosbag` or `rostopic` tools. Analysing system clock drift with command line tools.

AFFECTED RESOURCES working memory and CPU would be used more due to more interpolation/extrapolation computation between unsynced data streams.

TAGS configuration

B.6.3 *Interesting bug 32*

OBSERVATION PCL segfaulted on non-Debian/Ubuntu machines when trying to compute a convex hull.

CAUSE The code was written to support Debian's `libqhull`, ignoring the fact that Debian decided to deviate from the upstream module in one compile flag that changed a symbol in the library from `struct` to `struct*`. That way all non-Debian ports of `libqhull` failed to work with PCL, and instead segfaulted while trying to access the pointer.

DEBUGGING

- minimal example
- `printf` within the PCL code
- `printf` within an overlayed version of `libqhull`
- `gdb`
- Debian package build description for `libqhull`
- upstream `libqhull` package
- 12 hours of continuous debugging.

AFFECTED RESOURCES Well, segfault, the entire module stopped working. So basically everything was affected to some degree..

TAGS driver & OS; basic programming issue

B.6.4 *Interesting bug 46*

OBSERVATION The robot kept asking someones name.

CAUSE Background noise in the microphone

DEBUGGING The bug was obvious: no limit on the amount of questions asked. Simply drawing/viewing the state machine made this very obvious.

AFFECTED RESOURCES

TAGS coordination; environment-related

B.6.5 *Interesting bug 60*

OBSERVATION signal processing in component chain gave different results after several months

CAUSE unknown

DEBUGGING

AFFECTED RESOURCES

TAGS not/accidentally solved

B.6.6 *Interesting bug 69*

OBSERVATION segfault

CAUSE timing and location of allocated memory

DEBUGGING memory dumps. . . many many memory dumps

AFFECTED RESOURCES it did not affect resources constantly, but system stability in general; maybe CPU and memory

TAGS basic programming issue

B.6.7 *Interesting bug 76*

OBSERVATION While operating, a robot system normally capable of autonomous obstacle avoidance would unexpectedly drop communication with its wireless base station and drive erratically with high probability of collision.

CAUSE The main process was started in a Linux terminal and launched a thread that passed wheel velocity information from the main process to the robot controller. When the terminal was closed or otherwise lost, the main process was terminated but the thread continued to run, supplying old velocities to the robot controller.

DEBUGGING top, debugger, thought experiments

AFFECTED RESOURCES None

TAGS coordination

B.6.8 *Interesting bug 83*

OBSERVATION Random segfaults throughout system execution.

CAUSE Bad memory allocation: `malloc` for `sizeof(type)` rather than `sizeof(type*)`.

DEBUGGING Backtrace with `gdb`, profiling with `valgrind`, eventual serendipity to realize the missing * in the code.

AFFECTED RESOURCES Memory

TAGS basic programming issue

B.6.9 *Interesting bug 133*

OBSERVATION memory mismatch, random crashes

CAUSE different components using different `boost` versions

DEBUGGING `debugger`, `printf`. Finally solved after hint from colleague

AFFECTED RESOURCES

TAGS basic programming issue

B.6.10 *Interesting bug 149*

OBSERVATION Erratic behaviour of logic

CAUSE Error in mathematical modeling

DEBUGGING Unit tests

AFFECTED RESOURCES None

TAGS

B.6.11 *Interesting bug 150*

OBSERVATION An algorithm was implemented in both C++ and MATLAB exactly the same way. However, only the MATLAB implementation was working correctly.

CAUSE Difference in storing the float point variables in MATLAB and C++. MATLAB rounded the numbers, however, C++ cut them.

DEBUGGING Step by step tracing and debugging, and watching variables. Then, comparing with each other.

AFFECTED RESOURCES Working memory

TAGS basic programming issue

B.6.12 *Interesting bug 153*

OBSERVATION Control Program crash after a consistent length of time

CAUSE Presumably memory leak. Never knew for sure.

DEBUGGING Not sure

AFFECTED RESOURCES Not sure

TAGS basic programming issue; performance bug

B.6.13 *Interesting bug 156*

OBSERVATION Visualization window crashing 100 % of the time I open it. Running the program inside of `gdb` resulted in the program successfully running 100 % of the time.

CAUSE ??? Likely something internal to closed-source graphics drivers interacting with OpenGL/OGRE

DEBUGGING Was able to eventually generate a backtrace that pointed to graphics drivers.

AFFECTED RESOURCES CPU/Memory/GPU were all affected because I had to run the program inside of `gdb`

TAGS driver & OS

B.6.14 *Interesting bug 162*

OBSERVATION bad localization of a mobile robot in outdoor campus environment. Jump of the estimation

CAUSE Bad wheel odometry reading.

DEBUGGING Analyze log file

AFFECTED RESOURCES None. Loss of performance due to incorrect position tracking

REFERENCES

[GA07] Dora Luz Gonzalez-Bañales and Manuel Rodenes Adam. "Web Survey Design and Implementation: Best Practices for Empirical Research." In: *Proceedings of the European and Mediterranean Conference on Information Systems 2007*. 2007, pp. 1–10 (cit. on p. 2).

[Gun+14] Haryadi S. Gunawi et al. "What Bugs Live in the Cloud?" In: *Proceedings of the 5th ACM Symposium on Cloud Computing*. Ed. by Edward D. Lazowska et al. New York, NY, USA: ACM, 2014, pp. 1–14. DOI: 10.1145/2670979.2670986 (cit. on pp. 2, 5, 6).

[Jin+12] Guoliang Jin et al. "Understanding and detecting real-world performance bugs." In: *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*. Ed. by Jan Vitek, Lin Haibo, and Frank Tip. ACM Special Interest Group on Programming Languages. New York, NY, USA: ACM, 2012, p. 77. DOI: 10.1145/2254064.2254075 (cit. on pp. 2, 5).

[McC04] Steve McConnell. *Code Complete. A practical handbook of software construction*. 2nd ed. Redmond, WA: Microsoft Press, 2004. 960 pp. ISBN: 978-0-7356-1967-8 (cit. on p. 5).

[Ste13] Gerald Steinbauer. "A Survey about Faults of Robots Used in RoboCup." In: *RoboCup 2012: Robot Soccer World Cup XVI*. Ed. by Xiaoping Chen et al. Lecture Notes in Computer Science 7500. Berlin, Heidelberg: Springer, 2013, pp. 344–355 (cit. on pp. 2, 5, 8).

[WMW16] Johannes Wienke, Sebastian Meyer zu Borgsen, and Sebastian Wrede. "A Data Set for Fault Detection Research on Component-Based Robotic Systems." In: *Towards Autonomous Robotic Systems*. Ed. by Lyuba Alboul, Dana Damian, and Jonathan M. Aitken. Lecture Notes in Artificial Intelligence 9716. Springer International Publishing, 2016, pp. 339–350. DOI: 10.1007/978-3-319-40379-3_35 (cit. on p. 2).

[ZAH12] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. "A qualitative study on performance bugs." In: *9th IEEE Working Conference on Mining Software Repositories (MSR), 2012*. Ed. by Michele Lanza, Massimiliano Di Penta, and Tao Xie. Piscataway, NJ: IEEE, 2012, pp. 199–208. DOI: 10.1109/MSR.2012.6224281 (cit. on p. 2).

SOFTWARE PACKAGES

[GDB] *GDB. The GNU Project Debugger*. URL: https://www.gnu.org/software/gdb/ (visited on 2017-06-08) (cit. on pp. 3, 7).

[RViz] *RViz*. URL: http://wiki.ros.org/action/recall/rviz?action=recall&rev=79 (visited on 2017-05-24) (cit. on p. 3).

[Valg] *Valgrind*. URL: http://valgrind.org (visited on 2017-06-06) (cit. on pp. 3, 7, 8).

[WiSha] *Wireshark*. URL: https://www.wireshark.org/ (visited on 2017-06-06) (cit. on p. 3).