A Data Set for Fault Detection Research on Component-Based Robotic Systems[⋆]

Johannes Wienke[1], Sebastian Meyer zu Borgsen[2], and Sebastian Wrede[1]

[1] Research Institute for Cognition and Robotics (CoR-Lab), Bielefeld University, Germany
[2] Center of Excellence Cognitive Interaction Technology (CITEC), Bielefeld University, Germany
{jwienke,semeyerz,swrede}@techfak.uni-bielefeld.de

Abstract. Fault detection and identification methods (FDI) are an important aspect for ensuring consistent behavior of technical systems. In robotics FDI promises to improve the autonomy and robustness. Existing FDI research in robotics mostly focused on faults in specific areas, like sensor faults. While there is FDI research also on the overarching software system, common data sets to benchmark such solutions do not exist. In this paper we present a data set for FDI research on robot software systems to bridge this gap. We have recorded an HRI scenario with our RoboCup@Home platform and induced diverse empirically grounded faults using a novel, structured method. The recordings include the complete event-based communication of the system as well as detailed performance counters for all system components and exact ground-truth information on the induced faults. The resulting data set is a challenging benchmark for FDI research in robotics which is publicly available.

1 Introduction

Like most other technical systems, robots are not free from faults that occur at runtime and affect the mission success as well as the safety for physically interacting systems. In contrast to pure software systems, classical testing and modelling methods which prevent faults in the first place are often only applicable to parts of robot systems and with a high overhead, i.e. due to the interaction of robots with their environment, in particular humans. Therefore, failures can frequently be observed in these systems and further measures need to be taken to improve the situation. For this purpose we use the definitions from Steinbauer [13], where "a failure is an event that occurs when the delivered service deviates from correct service. An error is that part of the system state that can cause a subsequent failure. A fault is the adjudged or hypothesized cause of an error."

[⋆] This work was funded as part of the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University and by the German Federal Ministry of Education and Research (BMBF) within the Leading-Edge Cluster Competition "it's OWL" (intelligent technical systems OstWestfalenLippe) and managed by the Project Management Agency Karlsruhe (PTKA).

One common method to address such failures is *autonomous fault detection*, which is an ongoing research topic since many years with several different directions. A significant portion of fault detection research for robotics so far has focused on sensor and actuator faults. However, on the level of the complete software system controlling complex robotic applications (e.g. mobile platforms like in the RoboCup@Home league) research is much more fragmented and results are often hardly comparable. Nevertheless, software faults occur and handling these would increase system stability. One reason for this is that common data sets and established benchmarking methods, as e.g. the Tennessee Eastman Process [2] for industrial FDI, do not exist (cf. Pettersson [11]). While erroneous behaviors are frequently observed when operating robots, gathering usable data from these executions is often impossible. Because of the much more dynamic nature of current robotics systems, such data sets usually cannot be acquired from existing systems, because either the appropriate execution traces were not recorded in a sufficient quantity at all, or in case recordings exist, ground-truth information about the exact faults that occurred and their timing is missing. Therefore, reference data sets need to be explicitly created.

This paper introduces a novel data set for developing and benchmarking fault detection approaches for autonomous robotics with a focus on the overarching system. The data set contains recorded executions of our RoboCup@Home system in a typical task for this RoboCup league to provide a realistic and challenging scenario. During execution, selected faults have been induced in the system. These faults are based on empirical findings from an online survey about faults in robotics to ensure realistic conditions and the scheduling is based on an algorithm which maximizes the amount of usable fault data while maintaining clear properties on the timing of faults. The data set comprises the complete system communication of the robot, detailed performance counters for all system components, ground-truth information about the induced faults and videos from a camera observing the scene for manual inspection and further annotation. All data is synchronized and available for download[3].

To provide a challenging and profitable benchmark, the data set specifically focuses on performance-related faults. Such faults do not immediately render the system unusable, e.g. through crashing important components, but instead slowly degrade its perceived or computational performance (following the ideas of Application Performance Management [15]). Hence, they are much harder to detect and easily missed during short testing cycles in active development work. Also, these non-catastrophic issues have a higher potential for being recovered at runtime and therefore are the most valuable ones to detect with FDI methods.

2 Related Work

Up to our knowledge, publicly available data sets for FDI research regarding the overarching software system of robots do not exist yet. Nevertheless, several

[3] At `https://doi.org/10.4119/unibi/2900912` and `https://doi.org/10.4119/unibi/2900911`. Detailed technical usage instructions are given there.

authors have performed research in this direction and reported on evaluation methods and internal data sets that have been used. Steinbauer and Wotawa [14] propose an FDI approach based on observers which check the communication and process spawning behaviors of system components against modelled frequency or threshold rules. For the evaluation two distinct faults have been induced in a test system. Neither their origin nor the amount of recorded trials is mentioned. A closely-related approach is described in Peischl, Weber, and Wotawa [10]. Here, the evaluation consists of 20 experiments in which a process crash is simulated and 6 additional case studies which test faults designed to analyze properties of the detection method. Golombek et al. [3] learn the statistics for communication patterns of the robot system. For the evaluation, a data set with 4 different fault types was recorded within an existing scenario of a service robot. For each fault type 10 runs of the scenario were recorded, each approximately lasting 5 minutes. The induced faults were based on the experience of an expert user with the specific system. Similarly, Jiang, Elbaum, and Detweiler [6] describe an approach which learns communication invariants from successful trials. For evaluation, a UAV had to land on a moving target area. In contrast to other approaches, no explicit software faults were induced but instead the task conditions were made more challenging, e.g. by producing wind, and success was measured in the task space instead of measuring the detected faults.

3 Robot System and Recording Scenario

Our new data set has been recorded using the mobile robot platform *ToBi* [9] (cf. Figure 1), which successfully participates at the RoboCup@Home challenge since 2009. The robot consists of a mobile base with differential drive and two laser range finders with 360° coverage for distance data. Mounted on top of the robot are two RGBD cameras for object-recognition, obstacle avoidance, gesture-recognition and scene interpretation as well as an RGB camera for face recognition. For manipulation, the robot is equipped with a 5 DOF manipulator. The robot carries two Linux-based laptops which are connected via Gigabit Ethernet. They run the distributed software system controlling the platform.

The software architecture of *ToBi* (cf. Figure 1) consists of multiple sensor components that provide the system with information extracted from the scene like object recognition, person tracking and speech recognition. Actuator components that allow manipulation comprise navigation, text to speech and grasping. All components are distributed via an event-based middleware called RSB [17], which has tool support for transparently introspecting and recording the execution of the system at runtime. For the coordination, the BonSAI framework [12] abstracts the different components as software sensors and actuators and allows to model the system behavior as a finite state machine.

We chose a modified version of the restaurant task from the RoboCup@Home competition 2015 [1] as the recording scenario for the data set. RoboCup is one of the leading robot competitions and therefore provides a challenging and realistic environment for current capabilities of mobile robots. In the scenario the robot

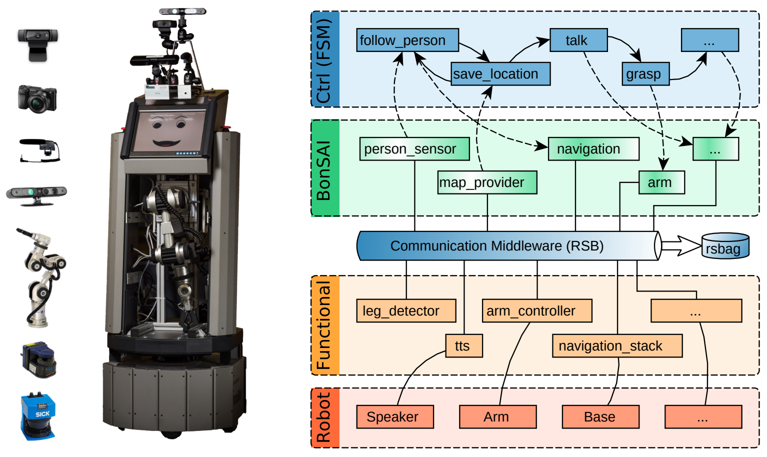
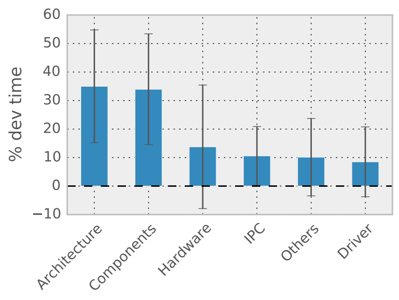


Fig. 1: Overview of *ToBi* and the system architecture. All inter-process communication is performed via the RSB middleware and can be recorded transparently using the `rsbag` utility.

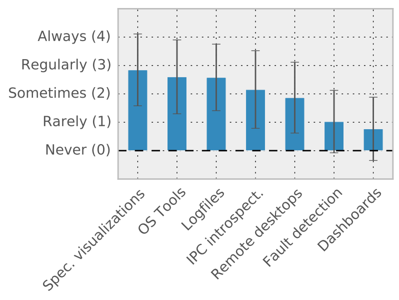
acts as a waiter in a restaurant. The plot consists of three phases. At first, an operator shows the robot around and trains multiple locations i.e. where it can pick up drinks as well as the locations of the tables where drinks have to be delivered to. In the second phase the robot waits for somebody waving to take the order of that person. The robot asks for the person's name and the desired drink. After taking all orders *ToBi* can be told to enter the third phase in which the orders get executed one after another. During the different phases SLAM is used to build up the map of the environment, face recognition and object recognition are used to identify persons and objects, and the RGBD sensors are used for planing grasping tasks. The scenario thereby integrates a diverse range of behaviors and skills necessary for a mobile robot with a high complexity and variability, especially due to the involved human-robot interaction. One iteration of the scenario with 2 to 3 guests takes approx. 11 min.

4 Induced Faults

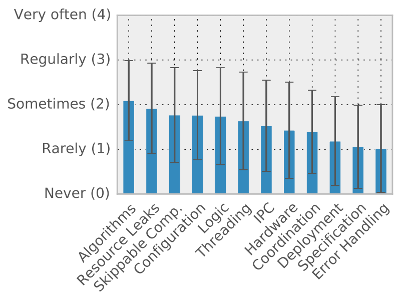
To create a data set for fault detection research in such a complex system, we did not only rely on our own experience to ensure that the resulting data set is representative for the domain. Instead, we created an online survey to acquire empirical and quantitative data on the types and distribution of different kinds of faults in comparable robotics and intelligent systems. For related domains like cloud computing or classical desktop software, a considerable amount of research on the types of faults and their occurrence frequencies exists. However, for robotics the situation is different. Up to our knowledge, there is only Steinbauer [13] which contains a systematic survey on faults in platforms participating



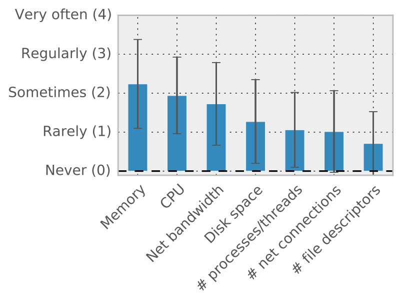
(a) Participants' development time



(b) Usage of monitoring tools



(c) Performance bug origins



(d) Affected computational resources

Fig. 2: Survey results as averages over all participants with standard deviation.

at the RoboCup challenges. Despite giving valuable insights, it is not detailed enough to support the construction of a data set with actual fault instances.

We implemented our survey as an online questionnaire (following methodology advices from Gonzalez-Bañales and Adam [4]) which was distributed around robotics researchers using well-known mailing lists. In total we received 61 completed submissions and 141 incomplete ones[4]. 86 % of the participants were researchers or PhD candidates at universities, 7 % regular students and 7 % from an industrial context. On average, participants had 5.8 years of experience in robotics (sd: 3.3). When asked with which activities participants spend their active development time, software architecture and integration as well as component development are the most frequent activities (cf. Figure 2a).

To determine the types of faults to induce we asked how often certain faults were the origin for performance bugs in the systems participants have been working with. Different categories could be rated with 5 choices ranging from *Never* to *Very often*. The categories have been chosen based on existing surveys from related domains [7, 5, 13, 8]. The results in Figure 2c indicate that algorithmic faults are the most frequent cause for failures, followed by resource leaks (not limited to memory) and skippable computations. Along these lines we also

[4] Incomplete submissions include visitors which only opened the welcome page once.

asked the participants to rate how often different computational resources were affected by faults. The corresponding results can be seen in Figure 2d.

In addition to these quantitative answers, we also asked participants to describe a prototypical failure situation they can remember, which is representative for the systems they have been working with. Four text input fields have been presented asking for the observable behavior of the system, the underlying fault, steps required to debug the system, and the computational resource that had been impacted. While such a case-study approach cannot provide reliable quantitative data, it served as an inspiration for fault instances to include in the data set. Since answering this question was optional, we received 25 distinct reports and a manual clustering of the answers suggests that software crashes are the most prevalent issue along the reported failures, followed by algorithmic issues.

Finally, participants had to rate how often they use certain tool types to monitor their systems. As visible in Figure 2b, autonomous fault detection is only rarely used, which motivates that more research on this topic is required.

Based on the results obtained from the survey we designed the following performance-related faults to induce in the system, which are listed according to the categories from the survey:

- Algorithms & Logic[5]:
  - A mathematical error is added to the conversion between Euler angle and quaternion representation, which is e.g. used to determine the location of persons in front of the robot.
  - The grasping controller for the arm performs unnecessary movements due to a bug in generating a trajectory in a graph of valid postures.
- Resource Leak:
  - The central state machine did not deallocate unused IPC connections, which results in a leak in TCP connections.
  - The speech recognizer did not deallocate memory for the sound buffers, which results in a memory leak.
- Skippable Computation:
  - The component which tracks persons transforms egocentric coordinates for detected person into global SLAM coordinates multiple times instead of only once per person.
  - Removes the throttling of the main loop in the face detection component, which increases the CPU load.
  - The detector for legs in the laser scans performed operations multiple times.
- Configuration:
  - The configuration of the state machine used a wrong middleware address to communicate with the text to speech engine and had to wait for a timeout before resorting to the correct address.
  - To emulate a configuration issue with the clock synchronization via NTP, the clock of one computer was shifted at system runtime.

[5] Categories were combined as it turned out to be hard to distinguish between them.

- Threading:
  - An unnecessary sleep instruction was added to the object recognition component, which delayed the classification results for 5 s to simulate the effect of inefficient threading strategies in this component.
- IPC:
  - The central communication daemon of the middleware is affected by constantly adding and removing a participant to the daemon network, which is a costly operation. As a consequence, IPC messages have much higher latencies and jitter.

These instances represent the 7 most frequent kinds of performance-related faults according to the survey.

5 Data Acquisition Method

5.1 Recording Setup

For realizing the data set acquisition process, we have used a method along the lines of Wienke, Klotz, and Wrede [16]. The core idea is to use the transparent recording capabilities of the robot middleware whenever possible. This automatically solves the synchronization for most parts of the data set, assuming that the middleware records accurate timing information. Consequently, we used the `rsbag` tool, which is part of the RSB middleware, to record the communication of the system, as depicted in Figure 1.

In order to include ground-truth information about induced faults and their exact timing within this recording method, the induced faults have been made triggerable via middleware events. A scheduling component was added to the system, which issued the required trigger events. This way fault ground-truth is included in the middleware recordings.

In the case of performance bugs, performance counters like CPU usage or network bandwidth are important information sources which had to be included in the data set. For this purpose we have added monitor components to the system, which uncover performance counters for all functional system components and the hosts and expose them via RSB for the inclusion in the data set. These monitors operate as external processes to the core system, which prevents changes in the system behavior due to the added monitoring. They are implemented as C++ programs with minimal processing overhead and acquire the required information via Linux interfaces like the `/proc` filesystem.

In addition to these data sources we have recorded each trial of the data set with an external HD camera to provide the possibility for human inspection and annotation. Since processing video inside the middleware infrastructure, in contrast to the aforementioned information, generates a significant load on the system, which differs from the usual usage of the system, we recorded the video out of band with the system communication. In order to synchronize this data, a special middleware event at the start of each trial was generated in parallel to a recognizable beep sound using a loudspeaker on the robot. As this sound was

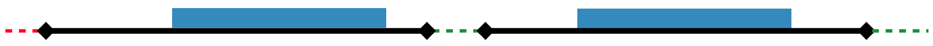


Fig. 3: Scheduling of induced faults (blue). The time of a trial (horizontal axis) is separated into slices (black) with an initial offset at the trial start (red dashed) and pauses between each slice (green dashed).

recorded in the external video camera as well, the videos could be synchronized to the system time using an automated cross-correlation analysis, which was manually checked for all videos.

To ensure the validity of the recorded data, each trial was automatically checked against completeness of the recorded middleware communication (RSB scopes and event rates). Additionally, the software system was restarted for each trial to prevent undesired effect from previous runs. Finally, in case unexpected errors or behaviors of the robot occurred during executing, these were annotated, so that trials without unexpected faults[6] are clearly recognizable.

5.2 Scheduling of Faults

Existing work like Golombek et al. [3] and Jiang, Elbaum, and Detweiler [6] used a simple strategy to induce faults into the system: at a certain point in time of each trial a single fault is induced and maintained until the end of the trial or system crash. While this approach provides an easy to analyze data set, it would be very time-consuming given the more complex scenario and number of faults in our data set, to provide a statistically feasible amount of fault occurrences. Therefore we opted for a strategy where multiple reversible faults were triggered during each recording trial, to maximize the amount of fault occurrences being recorded. This means that each fault that was triggered via the middleware communication could also be reverted back to a healthy system state as if the fault had never occurred. Such an approach forbids faults that are catastrophic to the system execution, however, we have already argued in the introduction, why such faults are less interesting for autonomous fault detection.

With this general approach we realized the exact scheduling of faults during each trial as follows (cf. Figure 3): starting with the initial execution of the system state machine, the trial was separated into consecutive time slices of a fixed length, which were additionally separated by a fixed length pause. Within each of the slices, a single fault was scheduled for a fixed time interval. The fault instance was uniformly drawn from the available ones and the start time of fault within the slice was also determined using a uniform distribution[7]. This procedure was chosen to prevent accidental correlations between system states and the induced faults, e.g. through an operator manually triggering the faults. A uniform selection was chosen to provide the same statistical confidence for each fault type. To further reduce potential correlations the start of the first slice was

[6] Up to the knowledge of an expert user.
[7] Limited so that the target fault time fits into the slice.

randomly offset after the state machine start using a uniform distribution up to 30 s. For the length of each fault 80 s was used. This time was determined by an expert developer of the system so that this person was able to detect each of the fault types from appropriate visualizations of the event communication and the performance counters[8], but without resulting in a catastrophic failure of the system due to the implemented resource leaks. The length of each slice was selected to be 160 s to provide sufficient variation for the fault occurrence and the pause time between slices was set to 20 s as the minimum acceptable distance between consecutive faults. This reflects the maximum time the recovery from any of the faults might take (after the signal to recover normal state, heuristically determined) so that instances are correctly separated in any case. The start of each slice is exposed via RSB to include scheduling information in the data set.

6 Corpus Content

With the explained method we have recorded a data set which consists of 10 executions of the system without induced faults as a baseline and 33 successful trials with induced faults. This number was chosen so that at least 10 complete instances of each induced fault were recorded during execution. The total time of recordings is 8:16 h, which results in an average of 11:33 min per trial. The recorded system, on average, exchanged middleware events with 113 Hz, including performance counters and fault scheduling information, which results in approx. 190 MB of raw event data per trial. For each RSB event, the middleware provides the following information, which is available in the data set:

Scope Middleware channels the event has been sent to (string).
ID A unique ID for the event including the sender ID.
Type A string describing the data type of the contained data.
Data The user-defined payload (binary, basic types or Google Protocol Buffers).
Method An optional string describing a method call associated with the event.
Causes A vector of other event IDs an event refers to.
Timestamps Timestamps in µs precision describing event processing steps.
User-defined Infos A user-defined set of string key-value pairs.

Performance counters have been recorded at 1 Hz, to align with the update rate for the counters from the Linux kernel and to prevent heavy load on the system. Only the network bandwidth information per process has been sampled at 0.5 Hz as it is more costly to generate this information. In detail, the following counters have been recorded:

CPU Time spent in user and kernel mode.
Memory Virtual and Resident Set Size.
Threads Number of threads of the process.
Descriptors Open files and file descriptors.

[8] In appropriate situations. For instance, the grasping controller fault is only detectable in case the arm is used.

I/O Total reads and writes and disk-specific reads and writes.
Network Receiving and sending bandwidth.

For each host the following information was recorded at 0.5 Hz:

CPU Total CPU usage per state for all virtual cores.
Load The 1 min, 5 min and 15 min load average.
Memory Total and used system memory.

The trial recordings are available in the original `rsbag` format as well as in CSV files, where apart from the raw data the network bandwidth counters, the fault state of each component and relevant encoded events for each component have been conformed to the 1 Hz sampling of the other performance counters.

In addition to the trial data, machine-readable meta-data (CSV and JSON) describing the available trials (with contained faults, additional operator notes and results of the automatic validation), the structure of the system (component processes, relevant RSB scopes and distribution across the two laptops for each component, dependencies on other components), and the affected components for each fault is available.

Apart from the 33 trials of the core data set, 23 additional trials are available, which contain unexpected behaviors of the robot or system. These are separated from the core data set and can be used for explorative analyses based on manual annotations. Projects for the ELAN[9] annotation tool are included for this purpose, which include the system communication. The number of trials with undesired faults is that high because during the recording session an actual hardware issue appeared. A lose screw inside the gripper resulted in a situation where the sensors did not detect whether an object was inside the gripper or not. As a consequence, grasping frequently stopped at the point where the gripper was closed and did not recover. We have annotated these situations, which results in an additional fault type being included in the data set.

6.1 Data Set Examples

Figure 4 displays an exemplary trial of the data set for the central state machine component to visualize the kind of data available within the data set. In the upper part of the figure, 3 of the available performance counters are shown. The lower part displays all events received and sent by the state machine. Each row of this plot relates to a single RSB scope and each vertical line in each row relates to a single event being communicated via this scope. The colored areas at the top of the first plot indicate the fault ground-truth information for this trial, where red areas indicate a fault that directly affects the state machine component whereas the orange area relates to an unrelated fault in other parts of the system. Finally, the dark shaded areas in all plots report the results of a fault detection approach which we have trained on the data set in order to verify the applicability for the intended purposes.

[9] `https://tla.mpi.nl/tools/tla-tools/elan/`

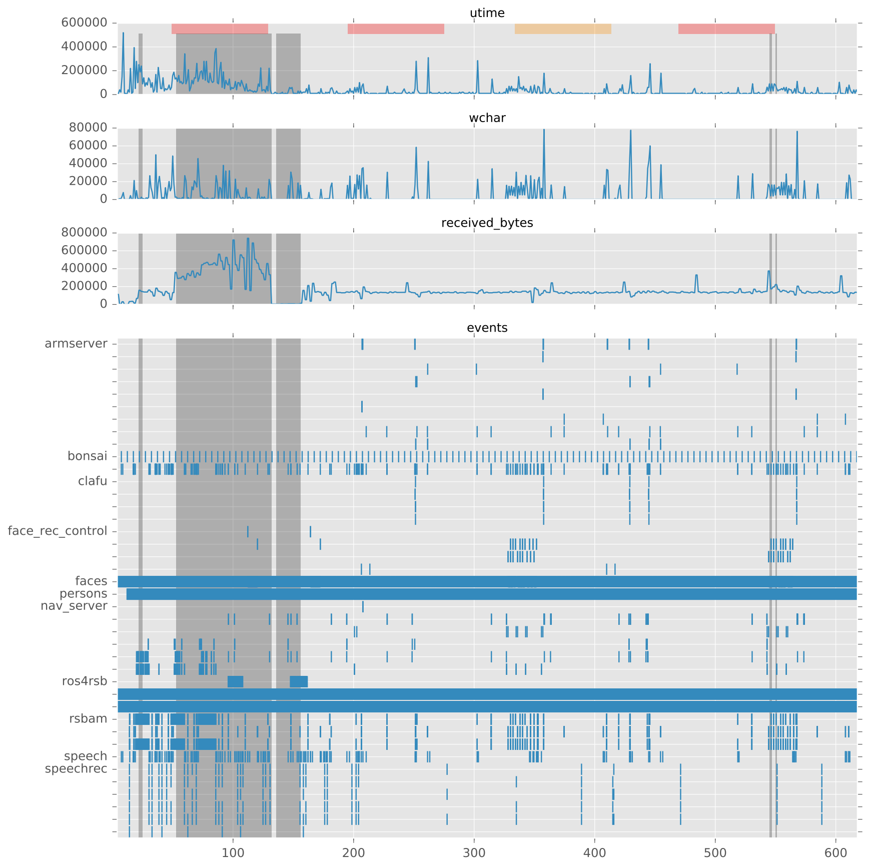


Fig. 4: Visualization of the recorded performance counters and events for the state machine component in one of the data set trials. The x-axis measures trial time in seconds. RSB scope names have been grouped by purpose.

7 Conclusion

With this contribution we have introduced a new publicly available data set for fault detection research on robotics systems. The data set specifically focuses on the overarching software system of event-based robots, an area currently lacking established benchmarks and publicly available data sets. It comprises a unique combination of performance information for system components and the complete system communication, which enables the application for a variety of purposes. We are actively using the data set for our own research, which validates its applicability. Our data set is generated in a challenging state of the art scenario and provides accurate ground-truth information and annotations. For this purpose, a novel mechanism of algorithmically inducing faults into a running system via the middleware has been proposed and the induced faults are empirically grounded through a survey. Both aspects improve the knowledge about underlying properties of the data set. With the available variety in covered faults and system states the data set presents a new challenging benchmark for fault detection research and contributes to scientific progress in this area.

References

1. Beek, L. van et al. *RoboCup@Home 2015: Rule and Regulations.* 2015. URL: `http://www.robocupathome.org/rules/2015_rulebook.pdf`.
2. Downs, J.J. and Vogel, E.F. "A plant-wide industrial process control problem". In: *Computers & Chemical Engineering* 17.3 (1993), pp. 245–255.
3. Golombek, R. et al. "Online Data-Driven Fault Detection for Robotic Systems". In: *Intelligent Robots and Systems.* San Francisco: IEEE, 2011, pp. 3011–3016.
4. Gonzalez-Bañales, D.L. and Adam, M.R. "Web Survey Design and Implementation: Best Practices for Empirical Research". In: *European and Mediterranean Conference on Information Systems.* Valencia, Spain, 2007.
5. Gunawi, H.S. et al. "What Bugs Live in the Cloud?: A Study of 3000+ Issues in Cloud Systems". In: *Proceedings of the ACM Symposium on Cloud Computing.* ACM. 2014, pp. 1–14.
6. Jiang, H., Elbaum, S., and Detweiler, C. "Reducing failure rates of robotic systems though inferred invariants monitoring". In: *Intelligent Robots and Systems.* IEEE. Nov. 2013, pp. 1899–1906.
7. Jin, G. et al. "Understanding and detecting real-world performance bugs". In: *ACM SIGPLAN Notices* 47.6 (2012), pp. 77–88.
8. McConnell, S. *Code Complete.* 2nd. Microsoft Press, 2004.
9. Meyer zu Borgsen, S. et al. *ToBI-Team of Bielefeld: The Human-Robot Interaction System for RoboCup@ Home 2015.* 2015.
10. Peischl, B., Weber, J., and Wotawa, F. "Runtime fault detection and localization in component-oriented software systems". In: *17th International Workshop on Principles of Diagnosis (DX'06).* Penaranda de Duero, Spain, 2006, pp. 195–203.
11. Pettersson, O. "Execution monitoring in robotics: A survey". In: *Robotics and Autonomous Systems* 53.2 (2005), pp. 73–88.
12. Siepmann, F. and Wachsmuth, S. "A Modeling Framework for Reusable Social Behavior". English. In: ed. by Silva, R.D. and Reidsma, D. Work-in-Progress Workshop Proceedings. Amsterdam: Springer, 2011, pp. 93–96.
13. Steinbauer, G. "A Survey about Faults of Robots used in RoboCup". In: *RoboCup 2012: Robot Soccer World Cup XVI.* Springer, 2013, pp. 344–355.
14. Steinbauer, G. and Wotawa, F. "Detecting and locating faults in the control software of autonomous mobile robots". In: *International Joint Conference on AI.* Ed. by Kaelbling, L.P. 2005, pp. 1742–1743.
15. Sydor, M.J. *APM Best Practices: Realizing Application Performance Management.* Apress, 2010.
16. Wienke, J., Klotz, D., and Wrede, S. "A framework for the acquisition of multimodal human-robot interaction data sets with a whole-system perspective". In: *Multimodal Corpora: How Should Multimodal Corpora Deal with the Situation? Workshop Programme.* 2012.
17. Wienke, J. and Wrede, S. "A middleware for collaborative research in experimental robotics". In: *System Integration (SII), 2011 IEEE/SICE International Symposium on.* IEEE. Kyoto, 2011, pp. 1183–1190.