Fault Diagnosis in Robot Task Execution

Siddhartha Banerjee, Sonia Chernova
siddhartha.banerjee@gatech.edu, chernova@cc.gatech.edu

Abstract

Robots operating in human environments can be made more dependable and reliable by automated fault diagnosis. In this work, we focus on the specific diagnosis problem of fault isolation, with the goal of isolating those faults that are causes of a robot task execution failure. Our method extends existing multiple-fault isolation processes by adding heuristics for matching fault residuals to human-expert specified fault signature matrices. We also extend the existing isolation processes to generate time-series of fault diagnoses. In doing so, we are able to isolate non-fatal faults that can sometimes be the causes of task failure. Finally, we describe a dataset of mobile manipulation tasks in simulation that we plan to use for future development and benchmarking.

1 Introduction

Robots operating in human environments as part of human-machine teams will need to be made dependable and reliable (Guiochet, Machin & Waeselynck, 2017). Dependability and reliability are affected by faults and errors that are resolved through interventions, which is defined as unplanned assistance provided to a robot by a human agent. In order reduce the time taken during an intervention and increase the time between interventions, automated identification of a fault and its context is important (Shah, Saleh & Hoffman, 2008).

Fault diagnosis is the process of explaining the possible causes of a set of observed fault symptoms (Travé-Massuyès, 2014). This diagnosis can include (1) *fault detection*, which discriminates normal system states from abnormal ones; (2) *fault isolation*, which isolates the faulty components in the system; and (3) *fault identification*, which provides a taxonomy and possible consequences of the isolated fault.

In this work, we focus on developing a mechanism of automated fault isolation for a robotic system requiring an intervention. Specifically, in the event of both fatal and non-fatal faults during task execution, our goal is to isolate the, perhaps non-fatal, faults that led to the ultimate failure in the task. Fixing such faults would allow the task to proceed, and thus these faults would be termed as causes for the task failure—a counterfactual notion of causality that is popularly used in the literature (Gerstenberg & Tenenbaum, 2017; Pearl & Mackenzie, 2018).

More generally, in addition to finding the causes of task failure, an effective fault isolation system for robotic applications has the following properties:

1. It does not need an extensive fault forecasting, such as is commonly done through Failure Modes Effects and Criticality Analyses (FMECA), Fault Tree Analyses (FTA), etc. (Guiochet et al., 2017), to aid in fault detection and identification.

2. It is model-free and does not need extensive system modeling (Pettersson, Karlsson & Saffiotti, 2007).

3. It is able to handle temporal dependencies between faults (Travé-Massuyès & Calderon-Espinoza, 2007; Gössler & Le Métayer, 2015).

4. It is independent of the task execution mechanism (Pettersson et al., 2007).

5. It is able to leverage expert human knowledge about the fault when it is available and can be shared (perhaps using the techniques of Abreu, Zoeteweij & van Gemund, 2011), but is capable of using data-driven means in the absence of expertise

In this paper, we describe our work towards developing a fault isolation system satisfying the above goals. We extend the multiple-fault isolation technique of Abreu et al. (2011) by adding heuristics to their methods of matching fault residuals to human-expert specified fault signature matrices. We also

examine the use of their methods in generating a time-series of fault data to begin isolating possible non-fatal causes of task failure.

2 Related Work

Khalastchi and Kalech (2018a) define Fault Detection and Diagnosis (FDD) as the field of study dedicated to detecting and identifying faults in sophisticated machines or programs, with the ultimate aim of facilitating recovery from them. In a complex system, such as a robot, FDD faces many challenges, including the facts that:

1. Faults are often rare, and, as a result, training data is hard to obtain. Moreover, faults might manifest or behave differently between simulation and the real-world.
2. Modeling the robot system, the system's behaviour, and environmental uncertainties are often time-consuming and hard, if not impossible.
3. Faults can be contextual, with anomalous behaviour in one setting serving as expected behaviour in another.
4. Paradoxically, adding sensors to aid in fault detection has the potential to induce additional complex faults, as the sensors themselves might be prone to faults.
5. Methods need to be quick, have low computational overhead, and be easy to develop to justify their inclusion in the system.

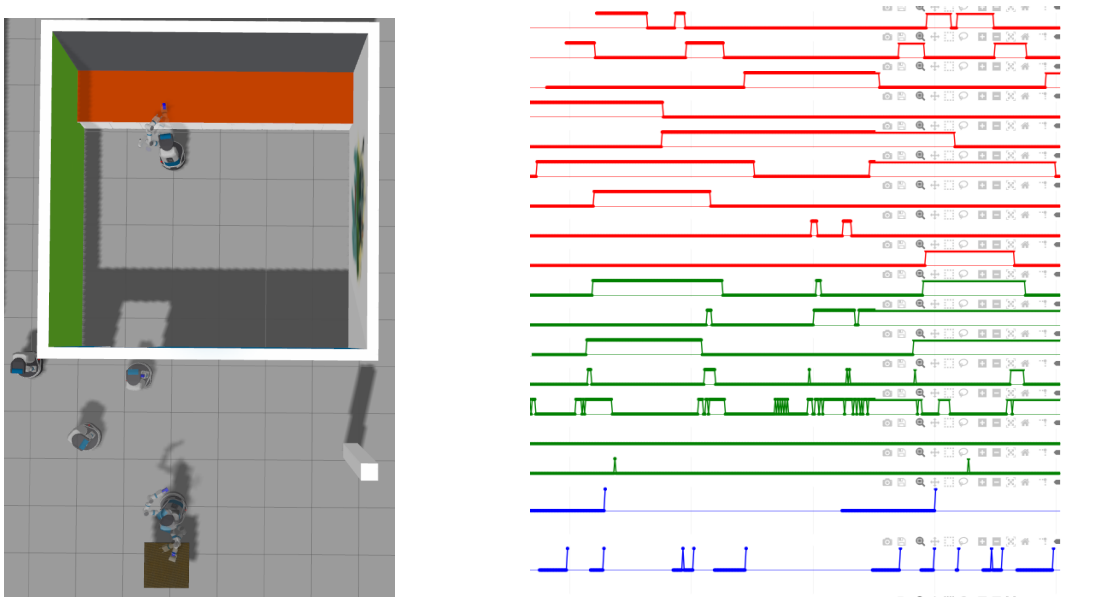
Reiter (1987), in one of the pioneering works of FDD, identified two approaches to FDD: (1) *Diagnosis from first principles*, which aims to find conflicts between the expectations of a system's behaviour and observations of that system when given a formal description of the system or its desired behaviour; and (2) *Diagnosis through experience*, which relies upon heuristics obtained through past experience or human expertise to infer the faults and their causes when given observations of the system. The first class of diagnostic approaches are now termed as *Model-Based Diagnosis (MBD)* approaches (Travé-Massuyès, 2014), while the second class of approaches are termed *Knowledge-Based* or *Data-Driven* approaches depending on the source of the heuristics (Pettersson, 2005; Khalastchi & Kalech, 2018a).

MBD methods require the specification of system component models or the system's behavioural model (Travé-Massuyès, 2014). The formalism used to specify such models differentiates the two schools of thought with regard to MBD approaches. When models are specified as first-order logic formulas, fault detection requires finding inconsistencies between observations and the system's model. In the event of an inconsistency, the fault is diagnosed by examining the conflict set of logical fluents in the inconsistency (Reiter, 1987; Gössler & Le Métayer, 2015). This approach has been used extensively in robotics to detect and diagnose faults in simple robot tasks (Steinbauer & Wotawa, 2005; Gspandl, Podesser, Reip, Steinbauer & Wolfram, 2012; Zaman, Steinbauer, Maurer, Lepej & Uran, 2013). When models are specified as continuous dynamical systems, as is common in the controls community, fault detection is viewed as a matter of constructing observers for state tracking, parameter estimation, or the elimination of unknown variables in the system. The process then uses sets of filters to derive residual values that are sensitized to different faults (Isermann, 2011; Gertler, 2014). In robotics, such residual generation is frequently used to detect hardware faults (Khalastchi & Kalech, 2018b). The residuals are in fact fault signatures, which can be combined offline into signature matrices to diagnose faults in residuals generated online (Crestani, Godary-Dejean & Lapierre, 2015; Khalastchi & Kalech, 2018b).

Both MBD approaches allow for straightforward detection and diagnosis of faults, which can be extended to track faults temporally (Travé-Massuyès & Calderon-Espinoza, 2007; Zhou, Feng, Zhao & Zhao, 2015). They suffer, however, from the drawback of requiring the difficult step of model specification.

Data-Driven approaches ameliorate the specification drawback of MBD approaches by learning to detect faults from data in an unsupervised manner, and often without a system model (Christensen, O'Grady, Birattari & Dorigo, 2008; Hornung, Urbanek, Klodmann, Osendorfer & van der Smagt, 2014; Khalastchi & Kalech, 2018b). However, Data-Driven approaches can be very sensitive to the quality of the training data, can ignore very rare faults, and can often lack a straightforward method for diagnosis (Khalastchi & Kalech, 2018a).

Knowledge-Based approaches provide opportunities to combine the strengths of Data-Driven and MBD approaches. These approaches augment human-knowledge into the FDD process by either specifying simplified models of the system and the signature matrices used in MBD, or by constructing expert systems for detection and diagnosis based on human input (Pettersson, 2005; Khalastchi & Kalech,



(a) Overhead view in simulation

(b) Belief (red), Fault (green), and Task (blue) residuals

Figure 1: The domain within which we operate in this paper.

2018a). Fault forecasting, using methods such as FMECA and FTA, are a popular and principled method for eliciting the expert knowledge required in Knowledge-Based FDD (Guiochet et al., 2017). Unfortunately, Knowledge-Based approaches suffer from an inability to detect unknown faults (Khalastchi & Kalech, 2018a), and the fault forecasting methods can be tedious (Guiochet et al., 2017).

Newer research has been advocating for the integrated use of MBD, Data-Driven, and Knowledge-Based mechanisms in the applications of FDD in order to complement their strengths. Crestani et al. (2015), for example, advocate for (1) the codesign of FDD with the recovery mechanisms used to recover from the fault, and (2) developing fault detection mechanisms that are specifically diagnosis-driven. As a result, the authors rely on a system model to create a slightly modified FMECA, which in turn is used to guide the development of MBD fault detection, the signature matrix generation for residual matching, and the steps taken during fault recovery. Similarly, Khalastchi and Kalech (2018b) use data-driven methods alongside a simple system model to learn fault detection mechanisms offline. They then generate diagnoses from those detections online by using the techniques for parsing signature matrices developed by Abreu et al. (2011). In this paper, we build upon the lessons learned from the systems described by Crestani et al. (2015) and Khalastchi and Kalech (2018b) to extend the techniques of Abreu et al. (2011).

3 Problem Domain

We focus on developing a fault isolation system for a robot engaged in a mobile manipulation task, but the faults that we explore in this paper typify common fault mechanisms we expect to find in other task domains as well. We assume that:

1. There is an error that stops execution in the robot task, and that this failure is communicated to the fault isolation system (and shared with human partners).
2. There are predesigned fault detection capabilities for failure in components, or sets of components, but that each of these mechanisms are not sufficient for all robot faults by themselves (see Sec. 3.4).
3. Once faults are isolated, the identification process of characterizing the class and consequences of the fault(s) are the purview of a subsequent system module.

3.1 Domain Implementation

We experiment with a Fetch Robotics mobile manipulator in simulation (Fig. 1a) and in the real-world. In simulation, the robot operates in a modified version of the default Gazebo simulation environment

#	Actions
1	Reset the arm, torso, and motion planning state. Set beliefs stating that the cube is on the table, and that the doorway is open.
2	Navigate to the table, raise the torso, and prepare the arm to pick up the cube.
3	Look at the table and identify the cube.
4	Calculate grasps on the cube (Kent & Toris, 2018).
5	Execute the top 10 grasps, in order, until there is no arm motion failure during the execution of a grasp. If successful, update beliefs to indicate the cube is in the gripper.
6	Spin away from the table, tuck the arm (with the cube in the gripper), and lower the torso.
7	Navigate to the doorway and check if it is unobstructed using the 2D LIDAR.
8	Navigate through the doorway and then on towards the shelf.
9	Approach the shelf, raise the torso, and prepare the arm to place the cube.
10	Place the cube on the shelf.
11	Find the cube on the shelf to verify that it is present there.
12	Update beliefs on the cube location, spin away from the shelf, tuck the arm, and lower the torso.

Table 1: High-level description of the mobile manipulation task.

provided by Fetch Robotics (Wise, Ferguson, King, Diehr & Dymesich, 2016); in the real world, the robot operates in an environment analogous to the simulation environment. We observed the typical faults described in Sec. 3.3, including perception and motion planning errors, in both the simulation and real-world environments.

3.2 Task Specification

The robot task, specified as a hierarchical state machine,[1] requires the robot to pick up a blue cube from the table (at the bottom of Fig. 1a), stow it in its gripper, travel through the doorway (in the middle of Fig. 1a) and drop it off at the orange shelf (at the top of Fig. 1a). Concretely, the high-level steps followed by the robot are specified in Table 1.

3.3 Observed Fault Classes

During executions of the robot, we commonly observe the following non-mutually-exclusive classes of faults, which we believe correspond to common hardware and software faults seen in robotics:

Component Faults. Such faults can occur at any time during the task and are either hardware failures or software component crashes; e.g.: `WiFi-signal-lost` or `Point-cloud-dead`.

Contextual Fault. Such faults often occur during specific task steps and are related to behaviours or components active at that step; e.g.: `Door-is-closed` when the robot checks in task step 7.

Multiple Correlated Faults. Such faults often co-occur with other related faults; e.g.: `Base-in-collision`, `Crowded-costmaps`, and `Invalid-plan` often co-occur during a navigation collision.

Multiple Uncorrelated Faults. Sometimes multiple unrelated faults might surface together; e.g.: `Arm-in-collision` and `Point-cloud-dead`.

Distal Root Faults. Due to the (often implicit) dependencies in some robot tasks, some non-fatal faults might expose other fatal faults later in task execution; e.g.: `Object-not-picked` in step 5 leads to the fatal `Object-not-found` error in task step 11. We seek to isolate the non-fatal `Object-not-picked` fault.

Detector or Belief Faults. Sometimes fault detectors or robot beliefs fail; e.g.: The `Door-is-closed` detector sometimes fails in task step 7 leading to an erroneous belief.

Undetectable Faults. Despite a designer's best efforts, some faults are hard to capture using an existing system of detectors and monitors; e.g.: In our system, `Robot-mislocalized` generates fault residuals that do not differ from those of the navigation collision faults mentioned previously.

Our goal is to be able to isolate as many of the above classes of faults as possible.

[1] Code available at `https://github.com/GT-RAIL/assistance_arbitration`

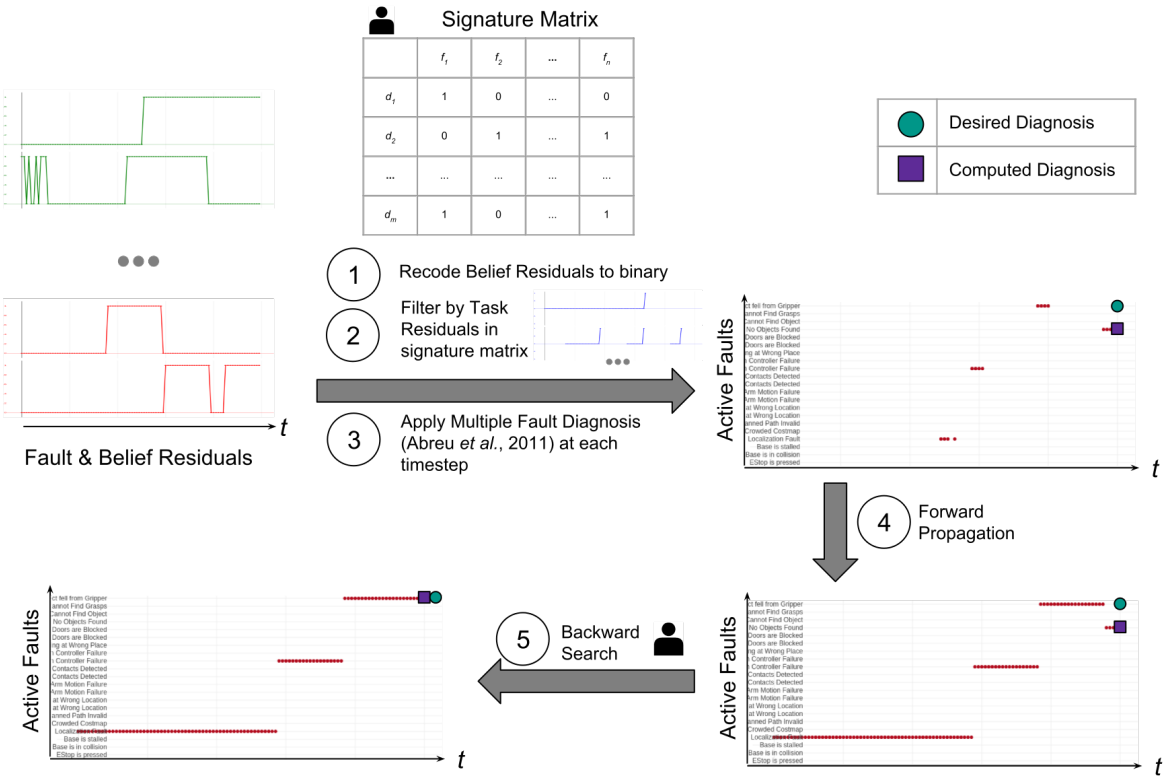


Figure 2: Five-step process for generating a desired *Distal Root Fault* diagnosis given a time series of residuals (Sec. 4). Until step 5 is applied, the final computed diagnosis does not match the desired diagnosis. Steps with the outline of a person are informed by expert-human knowledge.

3.4 Residuals for Fault Detection

Residuals are diagnosis-driven outputs in a system that are sensitised to different faults. In the realm of Model-Based Diagnosis, residuals are compared to an expert-annotated signature matrix of faults to isolate any particular fault (see Sec. 2). Our system generates 96 residuals of three types—*fault residuals*, *belief residuals*, and *task residuals* (green, red, and blue traces respectively in Fig. 1b).

Fault residuals are generated from several dedicated modules to observe reasonableness checks and timing constraints in specific components, such as monitoring if the robot base footprint is in collision or if the point cloud is refreshing at an expected frequency. These residuals are boolean values indicating if the corresponding checks are violated or not, and they can be noisy.

Belief residuals are generated because faults can sometimes arise when there is a mismatch between the robot's beliefs on the world and the actual world state, such as when the robot thinks that a door is open when it is actually closed (Pettersson et al., 2007). Clearly, the specific value of a robot belief is faulty only in certain settings and not in others. Therefore, we allow belief residuals to adopt any valid value within their domain with the understanding that the diagnosis process might use the different values of the belief residuals differently based on the context.

Task residuals aid in the diagnosis of contextual faults. We include the activation, or the completion status, of subtask actions with the residuals, noting that such an approach is agnostic to the formalisms used to specify the task (Pettersson et al., 2007).

4 Fault Isolation Method

The goal of the fault isolation step is to take as input multiple time series of residual values and output a set of fault diagnoses that indicate the cause of a task failure. In this work, we use a five-step process to extend the multiple-fault isolation method developed by Abreu et al. (2011) (see Fig. 2). Specifically, we (1) augment the representation of residuals in the signature matrix to parse belief residuals; (2) encode context specific signature matrices by including task residuals in the signature matrix; (3) apply fault isolation at multiple timesteps to generate a time series of diagnoses; and (4,5) search over the time series

of diagnoses to isolate possible *Distal Root Faults*.

The signature matrix is a key component in our method. In Model-Based Diagnosis, the matrix is a powerful Knowledge-Based method for incorporating human expertise into the fault isolation process—by setting the residuals of faults to expected values, it allows easy interpretation and use of the encoded knowledge (see Sec. 2). In the prior work of Abreu et al. (2011), the signature matrix consists of boolean indicators that associate residual values at a given time to known faults, which the authors then use to provide rankings of single or multiple fault candidates. We extend prior work by enhancing the expressivity of the signature matrix by allowing the entry of expected non-binary values for belief residuals. At runtime, belief residuals are recoded to binary based on whether the observed runtime residuals correspond to the expected values in the signature matrix or not.

We also enter task residuals, which are non-binary, into our signature matrix. These residuals are used to filter the fault hypotheses generated by the multiple-fault isolation system (see Abreu et al., 2011 for details). Our filtering step mimics the process of generating the multiple signature matrices that were used by Crestani et al. (2015) to isolate *Contextual Faults*.

Diagnosing *Distal Root Faults* requires additional heuristics because despite its expressivity, the signature matrix is a non-temporal means of diagnosis: it is unable to isolate faults that might have occurred in earlier timesteps because it is unable to encode a trajectory of fault residuals over time. Prior works have achieved a temporal trajectory of diagnoses by applying particle filters to dynamical system models (Zhou et al., 2015) or by applying search heuristics to causal system models (Travé-Massuyès & Calderon-Espinoza, 2007). In the absence of such models, we simply create a time series of diagnoses by applying fault isolation at each timestep of the residuals. It is important to note that as the symptoms of faults may be intermittent, not all timesteps contain fault diagnoses after the above step.

We therefore apply two additional heuristics to help in the diagnosis of distal faults: forward propagation of diagnoses in timesteps where we are unable to generate hypotheses, and backward search through the time series for the expected distal faults. The first heuristic aids the backward search by propagating relevant intermittent faults. The second heuristic is a Knowledge-Based search for those distal fault scenarios that we have encountered through experience. Our goal, in the future, is to automatically surface distal fault diagnoses as hypotheses without the need for experience-guided search.

5 Validation and Future Work

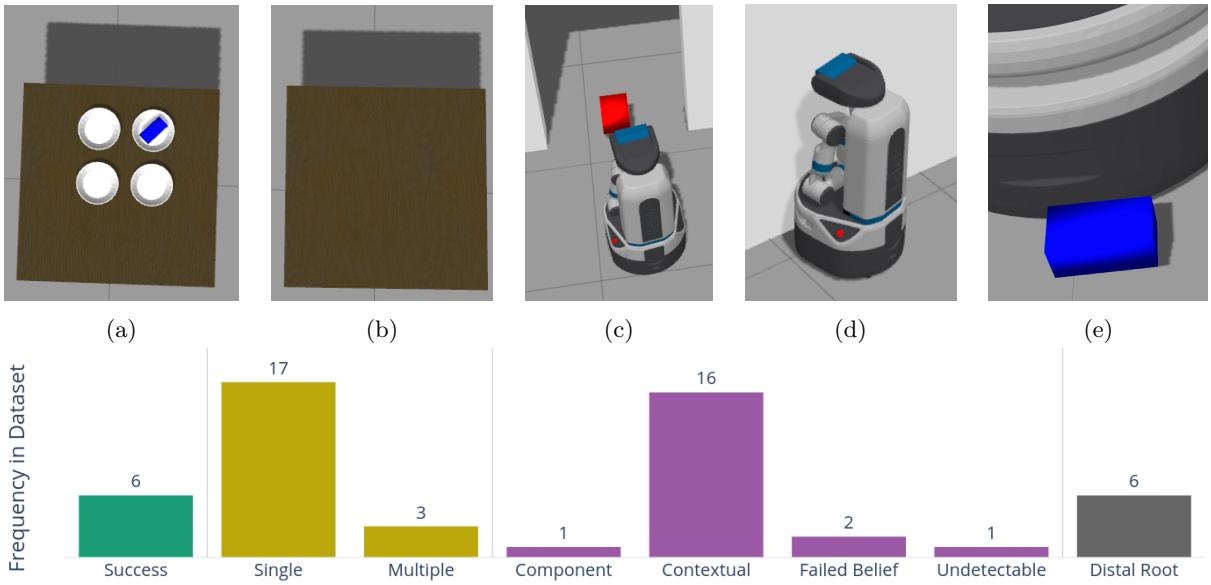


Figure 3: Examples of fault injection in simulation dataset (a-e) and the distribution of fault classes there in. (a) Cube cannot be segmented from plates; (b) Cube not at table; (c) Doorway is blocked; (d) Robot driven into the wall by malicious code; and (e) Robot base stalled by an undetectable door-stop.

In order to validate our method, we collected a dataset of 26 task executions in simulation, in which we injected various faults in 15 of the runs. Examples of the injected faults are shown in Fig. 3 a-e. The robot successfully completed its task in 6 of the runs (see *success* in Fig. 3), and the other 20 runs were

annotated with one of 29 possible causes of failure. The distribution of the non-mutually-exclusive fault classes (see Sec. 3.3) among all 26 runs is shown in Fig. 3. In this dataset, our current implementation is able to isolate single or multiple component and contextual faults, some of which are distal root faults, with 100% accuracy.

However, our method cannot yet diagnose faults in beliefs and fault detectors unless those faults are recast into component or contextual faults with an additional detection mechanism. More importantly, our methods require extensive experiential background with the system to drive both the detection and diagnosis processes, which is not desirable. Our ongoing efforts are aimed at addressing these limitations in a data-driven manner, while still leveraging the knowledge gained from experience whenever such knowledge is available.

References

Abreu, R., Zoeteweij, P. & van Gemund, A. J. (2011, Apr). Simultaneous debugging of software faults. *Journal of Systems and Software*, *84*(4), 573–586. doi: 10.1016/j.jss.2010.11.915

Christensen, A. L., O'Grady, R., Birattari, M. & Dorigo, M. (2008, Jan). Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, *24*(1), 49–67. doi: 10.1007/s10514-007-9060-9

Crestani, D., Godary-Dejean, K. & Lapierre, L. (2015, Jun). Enhancing fault tolerance of autonomous mobile robots. *Robotics and Autonomous Systems*, *68*, 140–155. doi: 10.1016/j.robot.2014.12.015

Gerstenberg, T. & Tenenbaum, J. B. (2017). Intuitive theories. *Oxford handbook of causal reasoning*, 515–548.

Gertler, J. (2014). Fault Detection and Diagnosis. In *Encyclopedia of systems and control* (pp. 1–7). Springer London. doi: 10.1007/978-1-4471-5102-9_223-1

Gössler, G. & Le Métayer, D. (2015, Dec). A general framework for blaming in component-based systems. *Science of Computer Programming*, *113*, 223–235. doi: 10.1016/j.scico.2015.06.010

Gspandl, S., Podesser, S., Reip, M., Steinbauer, G. & Wolfram, M. (2012, May). A dependable perception-decision-execution cycle for autonomous robots. In *2012 ieee international conference on robotics and automation* (pp. 2992–2998). IEEE. doi: 10.1109/ICRA.2012.6225078

Guiochet, J., Machin, M. & Waeselynck, H. (2017, Aug). Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, *94*, 43–52. doi: 10.1016/j.robot.2017.04.004

Hornung, R., Urbanek, H., Klodmann, J., Osendorfer, C. & van der Smagt, P. (2014, Sep). Model-free robot anomaly detection. In *2014 ieee/rsj international conference on intelligent robots and systems* (pp. 3676–3683). IEEE. doi: 10.1109/IROS.2014.6943078

Isermann, R. (2011). *Fault-Diagnosis Applications*. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-12767-0

Kent, D. & Toris, R. (2018). Adaptive autonomous grasp selection via pairwise ranking. In *2018 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 2971–2976).

Khalastchi, E. & Kalech, M. (2018a, Jan). On Fault Detection and Diagnosis in Robotic Systems. *ACM Computing Surveys*, *51*(1), 1–24. doi: 10.1145/3146389

Khalastchi, E. & Kalech, M. (2018b, Aug). A sensor-based approach for fault detection and diagnosis for robotic systems. *Autonomous Robots*, *42*(6), 1231–1248. doi: 10.1007/s10514-017-9688-z

Pearl, J. & Mackenzie, D. (2018). *The Book of Why: The New Science of Cause and Effect*. Hachette UK.

Pettersson, O. (2005, Nov). Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, *53*(2), 73–88. doi: 10.1016/j.robot.2005.09.004

Pettersson, O., Karlsson, L. & Saffiotti, A. (2007, Aug). Model-Free Execution Monitoring in Behavior-Based Robotics. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *37*(4), 890–901. doi: 10.1109/TSMCB.2007.895359

Reiter, R. (1987, Apr). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95. doi: 10.1016/0004-3702(87)90062-2

Shah, J. A., Saleh, J. H. & Hoffman, J. A. (2008, Aug). Analytical basis for evaluating the effect of unplanned interventions on the effectiveness of a humanrobot system. *Reliability Engineering & System Safety*, *93*(8), 1280–1286. doi: 10.1016/j.ress.2007.06.007

Steinbauer, G. & Wotawa, F. (2005). Detecting and locating faults in the control software of autonomous mobile robots. In *Ijcai* (Vol. 5, pp. 1742–1743).

Travé-Massuyès, L. (2014, Jan). Bridging control and artificial intelligence theories for diagnosis: A survey. *Engineering Applications of Artificial Intelligence*, *27*, 1–16. doi: 10.1016/j.engappai.2013.09.018

Travé-Massuyès, L. & Calderon-Espinoza, G. (2007, Jul). Timed fault diagnosis. In *2007 european control conference (ecc)* (pp. 2272–2279). IEEE. doi: 10.23919/ECC.2007.7068364

Wise, M., Ferguson, M., King, D., Diehr, E. & Dymesich, D. (2016). Fetch and freight: Standard platforms for service robot applications. In *Workshop on autonomous mobile service robots.*

Zaman, S., Steinbauer, G., Maurer, J., Lepej, P. & Uran, S. (2013, May). An integrated model-based diagnosis and repair architecture for ROS-based robot systems. In *2013 ieee international conference on robotics and automation* (pp. 482–489). IEEE. doi: 10.1109/ICRA.2013.6630618

Zhou, G., Feng, W., Zhao, Q. & Zhao, H. (2015, Nov). State Tracking and Fault Diagnosis for Dynamic Systems Using Labeled Uncertainty Graph. *Sensors*, *15*(11), 28031–28051. doi: 10.3390/s151128031